PhD Thesis

OPTIMIZATION OF YARD OPERATIONS IN CONTAINER TERMINALS FROM AN ENERGY EFFICIENCY APPROACH

Author:

Pablo Terán Cobo

PhD supervisor:

Dr. Sergi Saurí Marchán

Industrial PhD Pilot program in Civil Engineering

Escola Tècnica Superior d'Enginyers de Camins, Canals i Ports de Barcelona (ETSECCPB)

Universidad Politécnica de Cataluña – Barcelona Tech

Barcelona, January 2016

OPTIMIZATION OF YARD OPERATIONS IN CONTAINER TERMINALS FROM AN ENERGY EFFICIENCY APPROACH

Author:

Pablo Terán Cobo

PhD supervisor:

Dr. Sergi Saurí Marchán

Memoria presentada per optar

al título de Doctor Ingeniero de Caminos, Canals y Puertos

E.T.S. d'Enginyers de Camins, Canals i Ports de Barcelona (ETSECCPB)

Universitat Politècnica de Catalunya –BarcelonaTech (UPC)

Barcelona, Enero 2016

To my family

OPTIMIZATION OF YARD OPERATIONS IN CONTAINER TERMINALS FROM AN ENERGY EFFICIENCY APPROACH

Pablo Terán Cobo

Abstract

OPTIMIZATION OF YARD OPERATIONS IN CONTAINER TERMINALS FROM AN ENERGY EFFICIENCY APPROACH

Pablo Terán Cobo

Abstract

Ongoing containerization of maritime transportation has forced container terminals to cope with unprecedented volumes of containers, making the terminal efficiency a critical factor. In addition, operators must also face increasing operational costs deriving from the energy crisis and new regulations enforcing ports to become environmentally friendly. As a consequence, inefficiencies deriving from congestion require innovative solutions and optimization techniques to improve the efficiency and productivity of yard operations.

With that in mind, this thesis introduces a model to characterize energy expenditure of yard cranes. For each gantry, trolley and hoist crane movement, the model takes into account the different resistances that must be overcome during the acceleration, constant speed and deceleration phases of each movement. The energy consumption model is coupled to a discrete event simulation models of parallel and perpendicular container terminals yard, with the goal to analyze the handling operations and optimize energy efficiency and productivity. In particular, the works deal with (1) providing two numerical discrete event simulation models to analyze parallel and perpendicular terminals, (2) proposing a new stacking algorithm to reduce energy expenditure and improve crane productivity; (3) optimizing the dimensions of a perpendicular layout; and (4) analyzing the distribution of containers in the yard layout as a function of the moment at which space for export containers is reserved while looking at the operational costs.

In the first place, results show the models are capable of characterizing in detail the energy consumption of the parallel and perpendicular terminals. With respect to perpendicular terminals, the proposed stacking algorithm is capable of improving the energy efficiency around 20% while achieving greater productivity at the same time. In addition, results show that the dimensions of a perpendicular terminal block can be optimized so as to improve the productivity; with respect to energy consumption, although smaller block induce lesser consumption, the random nature of housekeeping operations introduce distortion in the results. Finally, considering parallel terminals, a greater degree of clustering is observed as the reservation is made earlier. When considering the associated operational costs associated to yard cranes and yard trucks, greater clustering results in more efficient use of the energy, and therefore reservation may be desirable when possible to enhance terminal productivity.

Keywords: container terminals, yard planning, storage capacity, allocating strategies, stochastic analysis, rehandling movements, yard cranes, automatic stacking cranes, operational strategies, energy consumption.

Sergi Saurí, Ph.D

Assistant Professor of Transportation

School of Civil Engineering–UPC BarcelonaTech

January, 2016

OPTIMIZATION OF YARD OPERATIONS IN CONTAINER TERMINALS FROM AN ENERGY EFFICIENCY APPROACH

Pablo Terán Cobo

Abstract

Acknowledgments

Llegado el momento de hacer balance y preguntarse “cómo he llegado hasta aquí”, es inevitable volver la vista atrás y reparar en que las decisiones, por mucho haga tiempo que uno camina solo sobre sus propios pies, no pueden disociarse de la persona que es, y la persona no puede entenderse sin su entorno y sin tod@s aquell@s que lo han acompañado a uno en el largo camino. Así pues, me gustaría expresar aquí mi sincero agradecimiento a los protagonistas principales y secundarios de esta tesis.

En primer lugar, quisiera agradecer especialmente a mi tutor y director de tesis, el Dr. Sergi Saurí Marchán, la confianza puesta en mí, así como su esfuerzo y sus gestiones para lograr que este programa de doctorado llegara a ser una realidad. Quisiera agradecerle también sus valiosos consejos y comentarios a lo largo de estos años para que la calidad del trabajo estuviera a la altura de las expectativas.

En segundo lugar, me gustaría agradecer a mi co-tutor de tesis en la empresa Sener Ingeniería y Sistemas, S.A., Xavier Pascual Lorente, por impulsar este proyecto y colaborar también desde el primer momento a hacerlo posible. También quisiera agradecerle su paciencia y el que haya velado en todo momento para que los trabajos se desarrollaran conforme a los planes previstos.

También por parte de Sener, me gustaría reconocer a mis compañeros de puertos por los ánimos y el apoyo recibidos antes y durante el proyecto, especialmente a Silvia, Alba y Pedro; a éste último además su inestimable ayuda por compartir conmigo su experiencia en el campo de las terminales de contenedores.

En el mundo de la industria de las terminales de contenedores, quiero expresar mi más sincero agradecimiento a Santiago Pallarés y a Estefania Soler por su generosidad y su disposición para ayudarme a entender muchos de los procesos que intervienen en una terminal; gracias a ellos la calidad del trabajo ha dado un salto de calidad inimaginable en otras circunstancias.

En cuanto a mis compañeros del CENIT, quisiera agradecer muy especialmente a los doctores Pau García Morales, y aún en mayor medida a Enrique Martín Alcalde, sus conocimientos, esfuerzo y dedicación a resolver los retos prácticos que ha ido planteando el día al día. A ellos sumo a Francisco, Pilar, Carles, Mireia, Albert y Carla por todos los buenos ratos que me habéis hecho pasar allí, no recuerdo un solo día con vosotros en el que no me haya reído.

Finalmente, en un plano más personal, quisiera expresar mi agradecimiento a mi familia y amigos, por cuidarme y cargarme las pilas cada vez que lo he necesitado: a Antonio, Naiara, María y Paula. A Jose, César e Isabel. A Nina y Chelo, Chuchi, Eva, Conchi y Jesús, Alberto, Raquel y Cipriano, Yaiza, Asun y Pepe (sentado para siempre al sol del verano, con una limonada en la mano, en la terraza de mi memoria), Sina y Luis (cuyo espíritu recuerdo gracias al telescopio que fabricamos, porque habita el cielo junto a los fantasmas del pasado). A Diego, Eva y Carmelo. A Scott, que tan pacientemente me ayudó a corregir el estilo y pulir el inglés de los textos. Y como no podía ser de otro modo, a mi otro cuarto, Roberto, Teresa, Rebeca y Sara; y mitad, Nuria, Martí y Adriá: gracias por estar ahí.

Barcelona, Enero de 2018.

Foreword

When the possibility of enrolling in a PhD program began to take shape, it had been already a few years since an issue had gathered my attention: the energy crisis and its relationship with the physical limits of planet Earth. Two blogs are the main reference for informative articles concerning this topic: “The Oil Drum” (internationally) and “The Oil Crash” (in Spanish). They both cover a wide variety of aspects of the energy crisis that range from analysis on scientific publications to less technical reflections of social character. Also, a great deal of information sources can be accessed from the abundant links provided in the articles. The fact that my research could be related somehow to this topic was a big motivation and one of the main reasons that led me here. This prologue intends to be a summary of what I consider the most relevant data needed to introduce the energy crisis problem; thus, and it is not meant to be exhaustive. My intention is not so much to convince the reader of the importance of the energy crisis immediately, but to grasp his attention and encourage contrasting the evidence provided here on his own, for which abundant sources of data and literature can be easily found.

> *If we were to keep a growth rate of energy consumption as the historical average (2.9 % per year), in less than 400 years we would have to absorb all the Sun radiation reaching the Earth; in 1300 years, we would have to absorb all the energy emitted by the Sun, and in 2500 years, we would need to absorb the radiation of all the stars of our entire galaxy. But there are previous insurmountable limits: in "only" 450 years the heat dissipated by our machines would boil the oceans of the Earth.*
>
> *Tom Murphy, from his blog “Do the Math!"*

At the time of writing this Thesis, it is seven years since the bankruptcy of Lehman Brothers on September 15th 2008. It is considered the largest bankruptcy in history considering the company’s assets and debts, and the commencement of a “financial” world economic crisis termed as “*the worst the world has seen since the Great Depression of the 1930s*”. Despite the optimistic forecasts projected year after year by the most important economic institutions in the world, the reality of the global economy reveals signs of stagnation and/or decline, most pronounced in the high-income regions (see Figure 1).

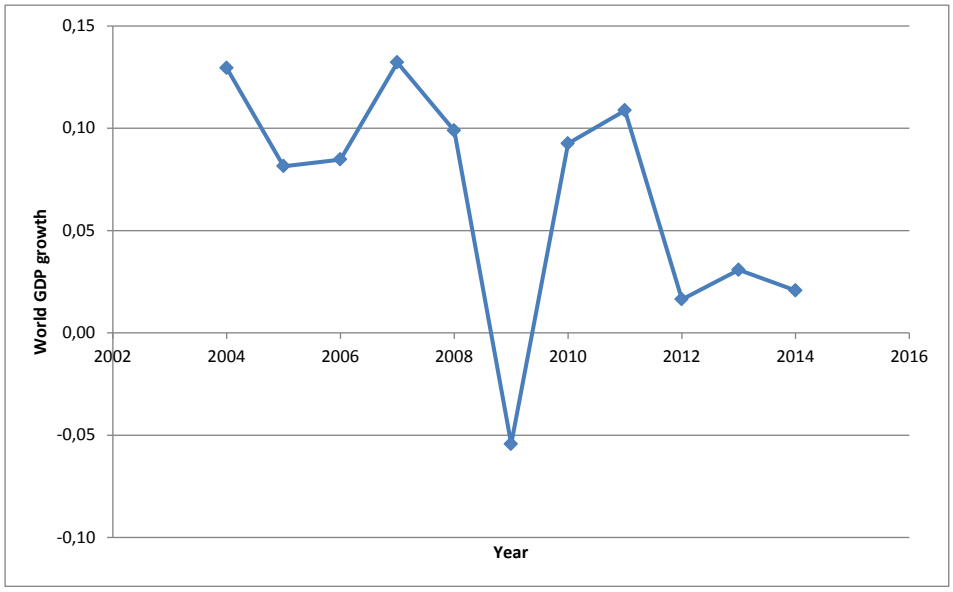


Figure 1. World GDP growth from 2004 to 2014.

Whereas the diagnosis of the crisis from mainstream economists focuses on diverse reasons of financial nature, a debate is growing among the scientific community regarding the role of the physical limits of the planet Earth on the economy. Their argument is that, as the human activity -and therefore the economy- require energy, the shortage or simply a shifting toward lower EROI forms of energy will necessarily reduce the level and growth of economic output. I.e., the U.S. Energy Information Administration (USEIA) predicted that implementing the Kyoto Protocol would reduce U.S. economic output by up to 4.3%in the year 2010 (Impacts of the Kyoto Protocol on U.S. Energy Markets and Economic Activity" (Washington, DC: U.S. Department of Energy, 1998).

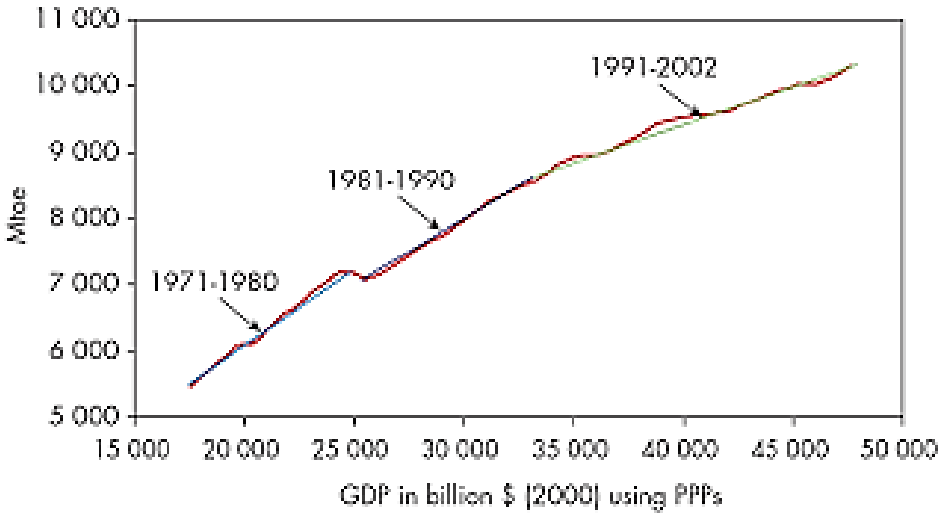


Figure 2. World GDP vs. energy consumption in millions of oil-equivalent tons (Mtoe). Source: World Energy Outlook 2014 (IEA).

Figure 2 also illustrates the relationship between the world's GDP and the utilization of energy, showing an almost perfect correlation.

The *primary energy* sustaining human life and activity mainly comes from fossil fuels (oil, coal and natural gas). I.e., in 2012, fossil fuels accounted for 87% of global primary energy consumption. The relative weight of these energy sources shifts with time: natural gas increased its share of energy consumption from 23.8% to 23.9% during 2012, while coal rose from 29.7% to 29.9 %, and oil fell from 33.4% to 33.1%. The properties of oil make this source of energy especially important; the concept *oil subsidy* expresses the degree of dependence of other sources to oil in order to be exploited (i.e. mining activities powered by diesel machinery required to produce uranium, construction materials deriving from oil required by power plants, and so on). When compared to the others, oil is not only easily manageable and transportable; it has a high energy density per unit of volume and also a high Energy Return on Energy Investment (EROI, see Figure 3). In addition, it is very versatile, which makes it suitable for a great variety of applications.

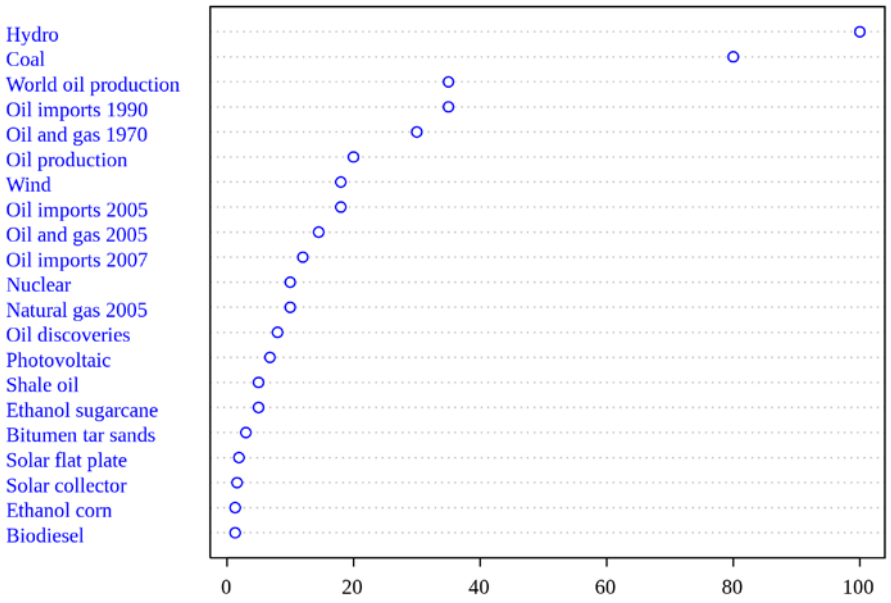


Figure 3. EROI of the most common energy sources (Murphy and Hall, 2010).

As it is well known, fossil fuels are a non-renewable form of energy. The first scientific that paid attention to the durability of these resources was Marion King Hubbert (October 5, 1903 – October 11, 1989), a geoscientist who worked at the Shell Research Laboratory in Houston (Texas). Back in 1956, Hubbert looked not only at the oil consumption rate of growth in the USA,

but also to the volume of oil reserves, and the rate of well discoveries. Based on his theory, he presented a paper to the 1956 meeting of the American Petroleum Institute in San Antonio, Texas, predicting that overall USA petroleum production would peak between 1965 (which he considered most likely) and 1970, which he considered an upper-bound case. Although his prediction received much criticism, it proved correct in 1970 (see Figure 3).

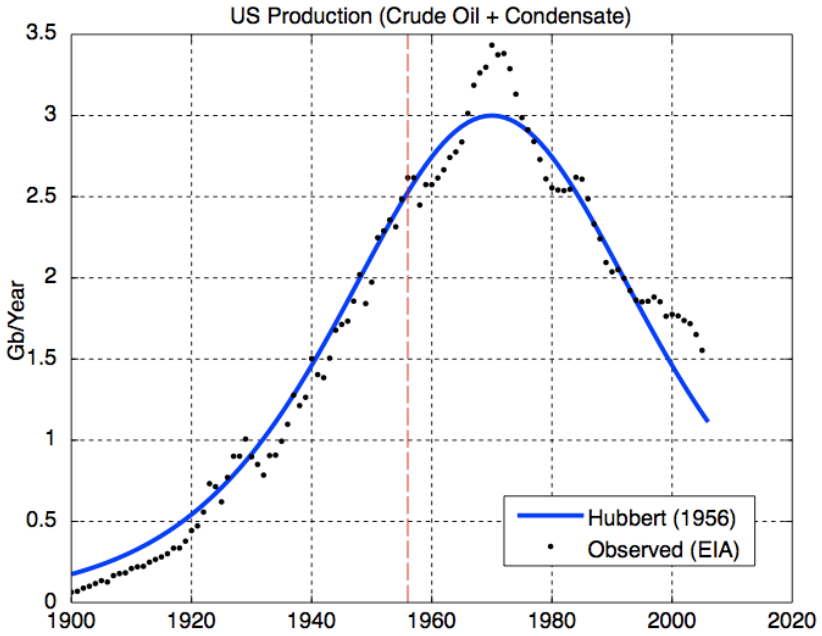


Figure 4. USA production from 1900 to 2005.

As time passed, the same phenomenon observed in the USA was reproduced in many other countries. Out of the 48 oil producer countries, 33 have passed their oil peak, as illustrated in Figure 4.

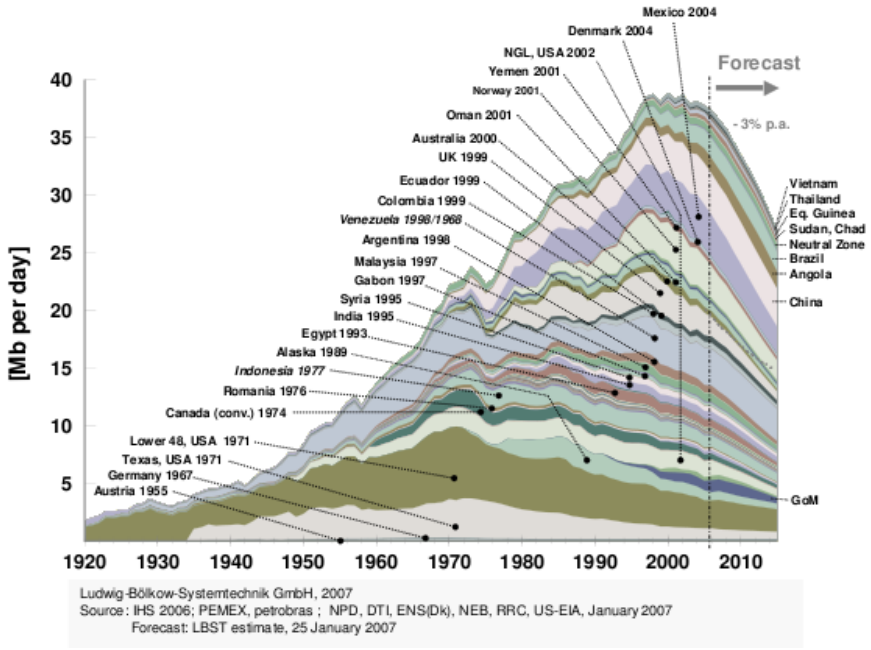


Figure 5. Oil producer countries past peak.

This fact is intimately related to the rate at which new oil is discovered for future exploitation in the whole world, which is in continuous decline since 1970, as illustrated in Figure 5.

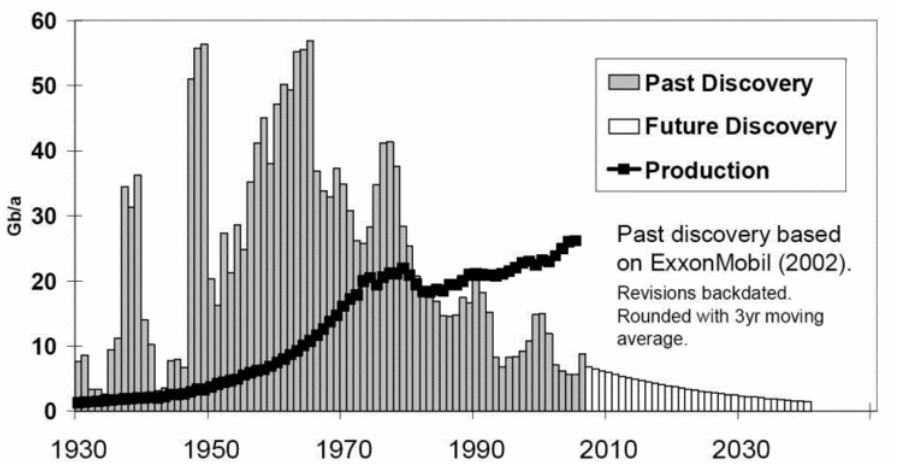


Figure 6. Global volume of new oil discoveries (in gigabarrels) from 1930 to 2007.

Based on the data shown above, it is not surprising that the total *crude oil* production should decline, which already happened in July 2006 at an estimated 85.43 Mbpd. Curiously, Hubbert predicted the world's oil peak "about half a century" from his 1956 publication. Even the IEA indirectly recognized this fact in its World Energy Outlook (2010). Since then, the world's total oil production (in volume) has remained constant, and the 6% volume decline in *crude* oil production has been compensated with other liquids of diverse nature. Among them, the greatest

contribution to the global production in terms of volume is the US light tight oil, which is extracted by the use of fracking techniques (Figure 6). This paradigm shift has been frequently renowned as the "US energetic revolution".

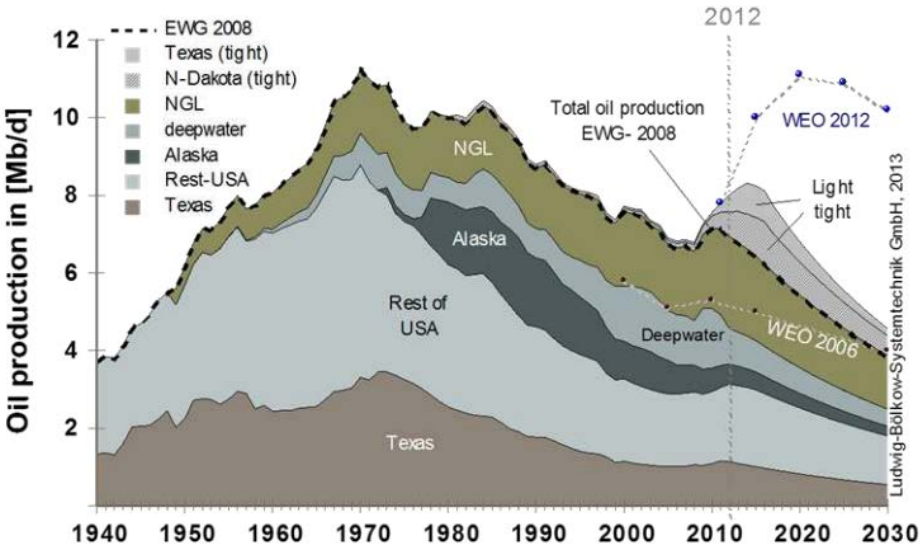


Figure 7. Historical US oil production (WEO 2012).

Although fracking's first commercially successful application dates from 1950, this technique has not been extensively utilized until recently, mainly to exploit three types of products: kerogen (the most abundant), tight oil and shale gas. Besides the severe environmental consequences and the short life of fracking exploitations (around 5 years, which is one of the main reasons why its production is transported by trucks instead of building pipelines), products extracted with this technique have a low EROI. According to Cleveland and O'Connor (2011), kerogen EROEI is as low as 2, whereas estimations for shale gas yield an even smaller value. Tight oil is not being exploited yet intensively, and only the Bakken formation has commercial exploitations so far. Although tight oil seems to have a better EROEI (≈12), the total reserves are around a hundred times smaller than those of the Kerogen, and the peak production is reached very rapidly, as experienced in the small commercial exploitation in Montana and the new production areas in North Dakota (Figure 7).

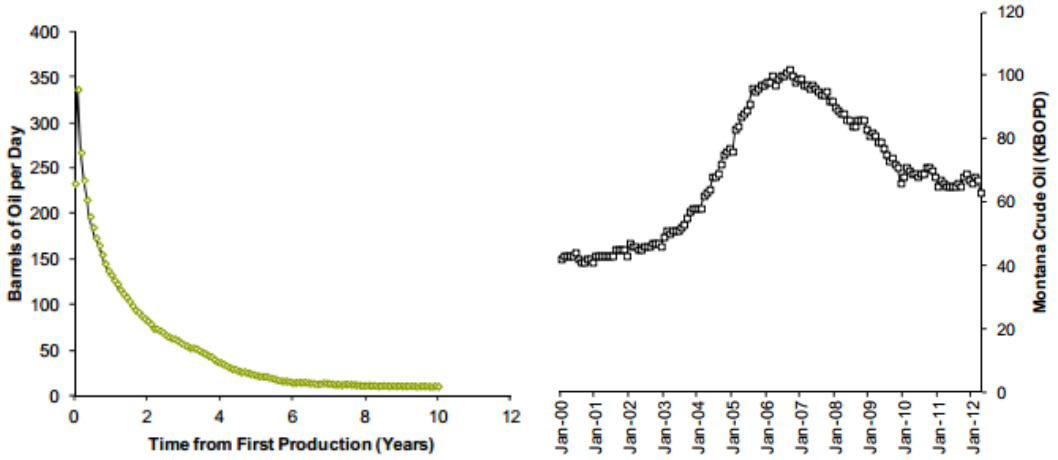


Figure 8. Left: Bakken (Montana and North Dakota) horizontal well decline curve (3,694 samples) (source: HPDI, Bernstein analysis) Right: Historical production in Montana (source: EIA, Bernstein analysis).

Considering this, the production associated to fracking seems distant from being revolutionary. According to Baker-Huges and other sources, at the time of writing this thesis the total number of rigs in the US is rapidly declining. As an indication, only the period from since October 2014 to October 2015 the number of rigs in the US declined in 1,154, remaining only 775.

In its last Oil Market report (August 2015) the IEA recognized a declination in the US oil production: "*A sharp decline is already underway, with annual gains shrinking from more than 1 mb/d at the start of 2015 to roughly half that level by July. Rigorous analysis of our data suggests that US light tight oil supply, the engine of US production growth, could sink by nearly 400 kb/d next year as oil's rout extends a slump in drilling and completion rates"*. Even the industry foretells a complicated future, and statements like this can be found in the media frequently: "*According to Bloomberg, half of all fracking companies will be out of business or sold by the end of 2015"*. Rex W. Tillerson, the chief executive of Exxon Mobil said, "*we are all losing our shirts today. We're making no money. It's all in the red*". Peter Voser, Chief executive of Shell, claimed "*the failure of Royal Dutch Shell's huge bet on US shale was a big regret*". Shell has invested at least $24bn in so-called unconventional oil and gas in North America.

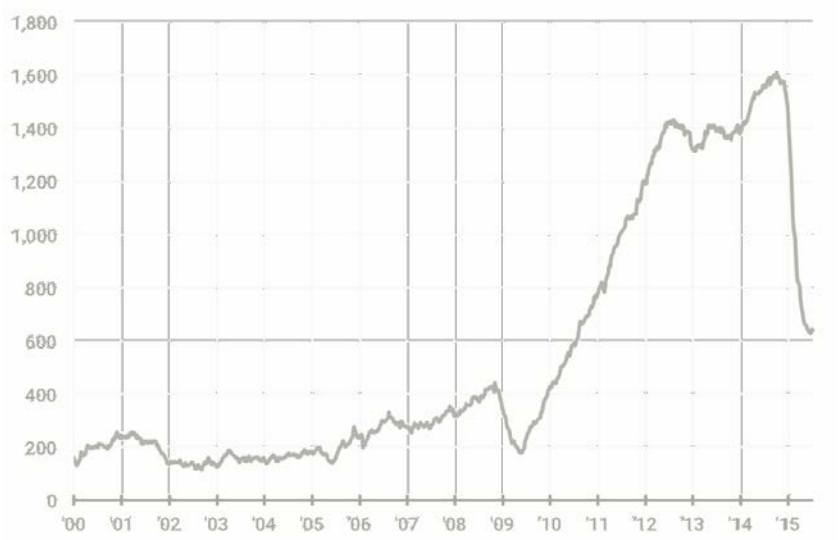


Figure 9. Historical number of oil rigs in the US (Source: Bakker Hughes).

The major companies' debt, which kept increasing even during time of high crude prices, keeps rising. As indicated by the IEA, "*cash from operations for 127 major oil and natural gas companies totaled $568 billion, and major uses of cash totaled $677 billion, a difference of almost $110 billion. This shortfall was filled through a $106 billion net increase in debt and $73 billion from sales of assets, which increased the overall cash balance. The gap between cash from operations and major uses of cash has widened in recent years from a low of $18 billion in 2010 to $100 billion to $120 billion during the past three years.*

Financial problems aside, when looking to the medium term future, projections of the IEA (Figure 11) under the optimistic scenario (sometimes referred to as New Policies Scenario, which considers that all oil reserves not exploited today will be exploited with an optimal throughput), reveal the production of *crude* oil in 2035 will approximately be ≈36% of that in 2005 (dark blue bar).

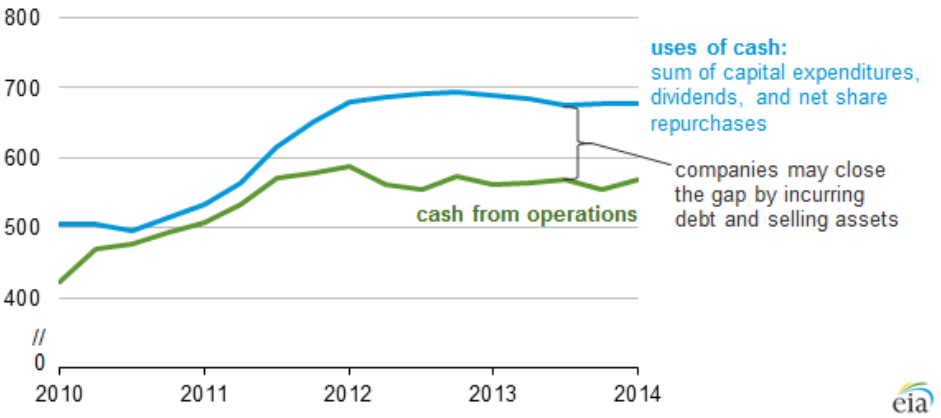


Figure 10. Major energy companies' cash rom operations and uses of cash in billion 2014 dollars, annualized values from quarterly reports[1]. Source: U.S. Energy Information Administration, based on Evaluate Energy database.

Surprisingly, the IEA forecasts the sharp decline in production will be compensated, reaching 100mbpd in 2035. However, when looking at the rate of new discoveries, such projections seem unrealistic. The oil production associated to "fields to be found" and "fields to be developed" do not compare well to historical data from Figure 6.

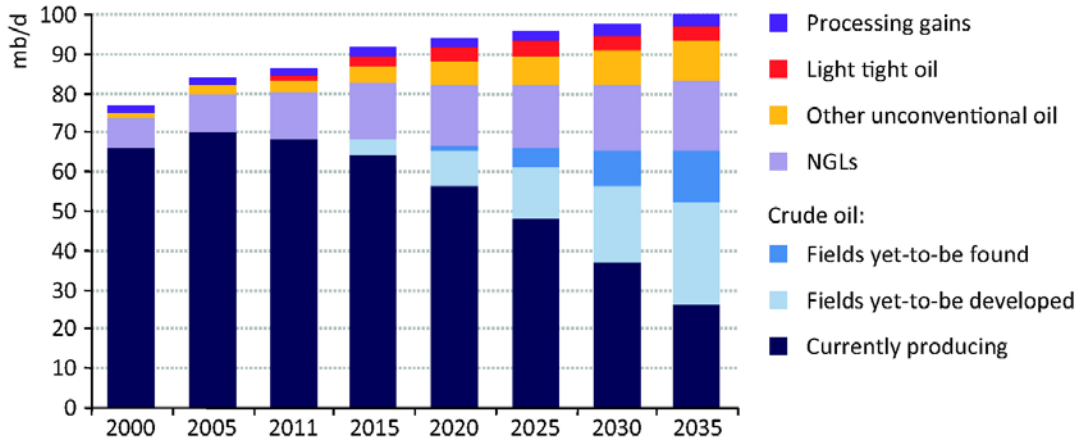


Figure 11. World oil supply by type in the New Policies Scenario (Fig 3.15 of the WEO 2014, IEA).

To this extent, we have considered only the *volumes of oil* produced and consumed, but it is also important to take into account the concept of EROEI (Energy Returned On Energy Invested). The EROEI of an energy source is a non-dimensional measure of the amount energy invested (input E_{in}) to produce a unit of usable energy (output E_{out}). The ratio E_{out} / E_{in} constitutes a convent way of expressing the efficiency of the energy production process. Evidently, energy sources with a high EROI are more convenient: as we approach an EROI of 1:1 (e.g. consuming 1 barrel of oil to produce 1 barrel of oil), it simply does not make sense to exploit that source.

[1] Annualized means each point on the graph is the sum of the previous four quarters. Thus, the first-quarter 2014 results on an annualized basis mean the data represent the sum of the four quarters ending March 31, 2014. The data above are the aggregate results of 127 global oil and natural gas companies.

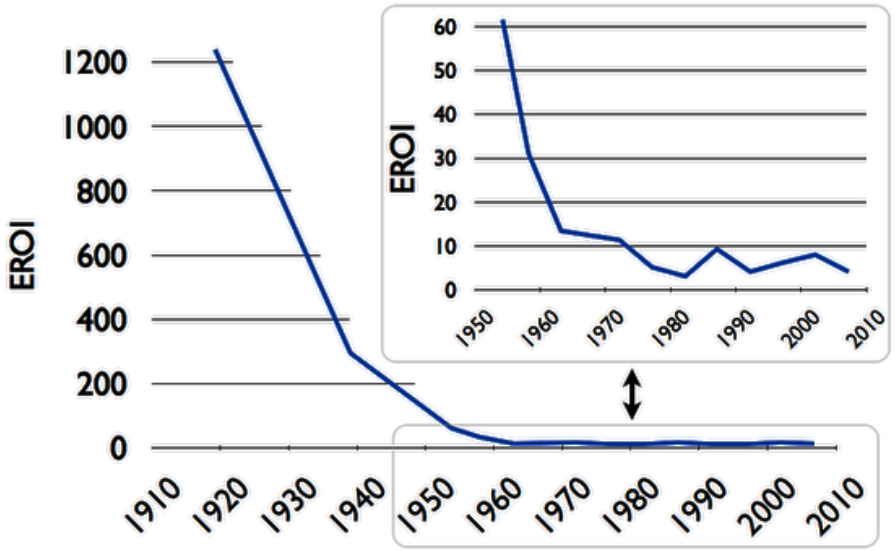


Figure 12. EROI for discoveries for the US oil and gas industry (Guilford et al., 2011).

According to Hall et al (2009) and others, a minimum EROEI to sustain a society is roughly 10, including the costs of transporting the energy to the places where it is finally needed for consumption. As the oil fields are exploited, their EROEI gets drastically reduced. This is not only because the best fields have already been utilized in the past, but also due to the nature of the fields themselves. When extraction starts in an oil field the more liquid, more energetically dense crude flows first. As the field gets mature, denser, poorer quality crude requires more efforts (energy) to be extracted, leaving extraction curves like that depicted in Figure 4. According to Murphy and Hall (2010), the EROI's domestic oil production in the US has decreased from 100:1 in 1930 to 40:1 in 1970, and to about 14:1 today (Figure 10). According to Guilford et al. (2011) the differences in the EROI from the early oil produced in the US and today's oil could be even bigger (Figure 11).

The oil's EROI reduction exacerbates the consequences of the oil shortage in supply to meet the world's increasing demand. The conclusions of the work by the Energy watch Group (an international network of scientists and parliamentarians that conducts research and publishes studies on global energy) are overwhelmingly explicit: "*the decline of oil production* (…) *will lead to a rising energy gap which will become too large to be filled by natural gas and/or coal. Substituting oil by other fossil fuels will also not be possible in case gas and coal production would continue to grow at the present rate*" [2]. The International Energy Agency predicts that by 2017 coal will replace oil as the dominant primary energy source worldwide.

[2] *Fossil and Nuclear Fuels – the Supply Outlook*, 2013

A more assertive statement about the current level of awareness can be found in Chapman (2013) and is literally cited here:

> *The conclusions are that, supported by commercial interests, an unsubstantiated belief in market and technical solutions, and a narrow paradigmatic focus, critics of Peak Oil theory have used unreliable reserve data, optimistic assumptions about utilisation of unconventional supplies and unrealistic predictions for alternative energy production to discredit the evidence that the resource-limited peak in the world's production of conventional oil has arrived, diverting discussion from what should be a serious topic for energy policy: how we respond to decreasing supplies of one of our most important energy sources.*

Considering again the production of crude oil in volume forecasted by the EIA and applying the concept of EROI as a correction factor to obtain the net energy that will be likely available for the economic activity, under the optimistic scenario, calculations indicate that the amount of energy in 2035 will be merely 15% of that in 2005. Although oil can be substituted by other sources to generate electricity, according to Maggio and Giacolla (2012) the natural gas is expected to peak in 2035, and coal in 2052. Less optimistic forecast, such as the EWG's, predicts the natural gas will peak as early as 2019, coal peak in 2020, and uranium sometime between 2020 and 2035.

In summary, the foreseen scarcity of this fundamental energy resource that probably commenced in 2006 underscores our responsibility to make every effort needed to use energy more efficiently, not only with the goal to minimize the impact of the emission of CO_2 and pollutants to the atmosphere that contribute to climate change, but also to mitigate undesirable economic consequences in our economies. In this context, from a personal point of view, the most important objectives of this Thesis is the analysis of the energy consumption of handling equipment in container terminals, with the aim to identify sources of superfluous expenditure and provide tools and guidelines to help making an efficient use of the energy without further need for new investment.

Table of Contents

List of Figures

List of Tables

List of abbreviations and symbols

The following list of abbreviations and symbols is thoroughly used throughout the document.

AGV	Automated Guided Vehicle
AM	Annual Maxima
AS/RS	Automated Storage/Retrieval System
ASC	Automatic Stacking Crane
BAP	Block Allocation Problem
BFR	Beaufort (Wind Scale)
BOL	Block Occupancy Level
CET	Container Exit Time
CDF	Cumulative Distribution Function
CT	Cycle Time
CTO	Container Terminal Operator
DES	Discrete Event Simulation or Discrete Event Simulator
DRMG	Double RMG (equivalently, crossable RMGs)
ET	External Truck
EWG	Energy Watch Group
GA	Genetic Algorithm
GEV	Generalized Extreme Value
GPD	Generalized Pareto Distribution
GRC	Gross Crane Rate
IP	Integer Programming
IT	Internal Truck
ITF	International Transport Forum[3]

[3] The ITF is an intergovernmental organization with 57 member countries at the OECD, acting as a think-tank for transport policy. ITF is the only global body that covers all transport modes.

KPI	Key Performance Indicator
MAE	Mean Average Error
MHE	Mechanical Handling Equipment
MIP	Mixed Integer Programming
Mbpd	Mega Barrels per day (oil production)
Mph	Movements Per Hour
O( )	Order of magnitude
OHBC	Over-Head Bridge Cranes
OPEX	OPerational EXpenditure
OR	Operations Research
PDF	Probability Density Function
PoT	Peaks over Threshold
RMSE	Root Mean Squared Error
QC	Quay Crane
RTG	Rubber Tired Gantry Crane
RMG	Rail Mounted Gantry Crane
SSAP, SAP Problem	Storage Space Allocation Problem or simply Slot Allocation
SC	Straddle Carrier
SRMG	Single RMG
STS	Sea To Shore (often to refer to a commonly used type of QC)
TA	Transfer Area, also called TP
TEU	Twenty-foot Equivalent Unit
TOS	Terminal Operationing System
TP	Transfer Point
TV	Transport vehicle
TRMG	Twin RMG
UNCTAD	United Nations Conference on Trade and Development

YC	Yard Crane
YT	Yard Truck

Chapter 1

Introduction

1.1 Containerization

Today, containers are the most important mean of intermodal transportation. The perks are numerous and varied: versatility, standardized sizes that make them suitable for ships, trucks or trains; safety, ease of management and rapidly interchangeable between different modes of transportation, etc. The shipping container has revolutionized freight and port operations and has been a catalyst in the growth of global trade. In 1990, world container port throughput volumes were around 85 million TEUs, and they have since grown sevenfold to 684.4 million TEUs (Figure 13) over 20 years (UNCTAD, 2015). This figure represents a 5.1% growth over the 651.1 million TEUS registered in 2014.

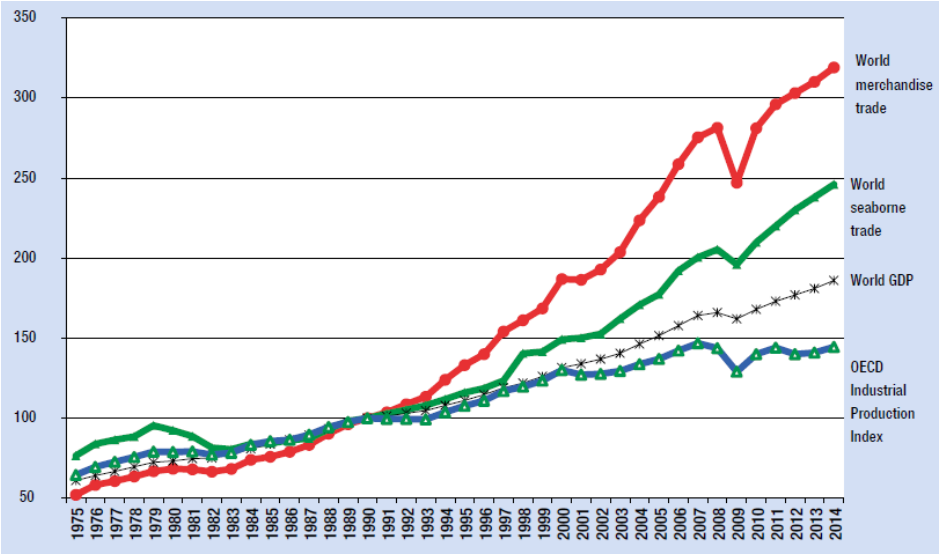


Figure 13. The OECD Industrial Production Index and indices for world GDP, merchandise trade and seaborne shipments (1975–2014) (base year 1990 = 100). Source: RMT UNCTAD 2015.

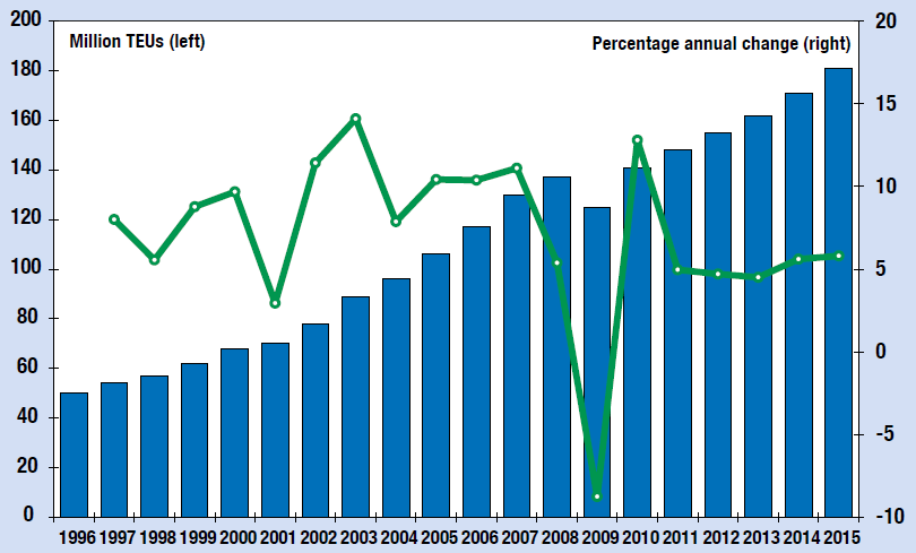


Figure 14. Global containerized trade, 1996–2015 (million TEUs and percentage annual change). Source: RMT UNCTAD 2015.

Container ships represent approximately one eighth of the total world fleet, but they are essential for the transport of goods all over the world. As a consequence of the "container revolution", vessels' size has increased rapidly over the last decades, faster than any other vessel type. In one decade, the average capacity of a container ship has almost doubled (Figure 15). In January 2015 the MSC Maya and her sister Oscar set the new largest container ship world record with 19,200 containers.

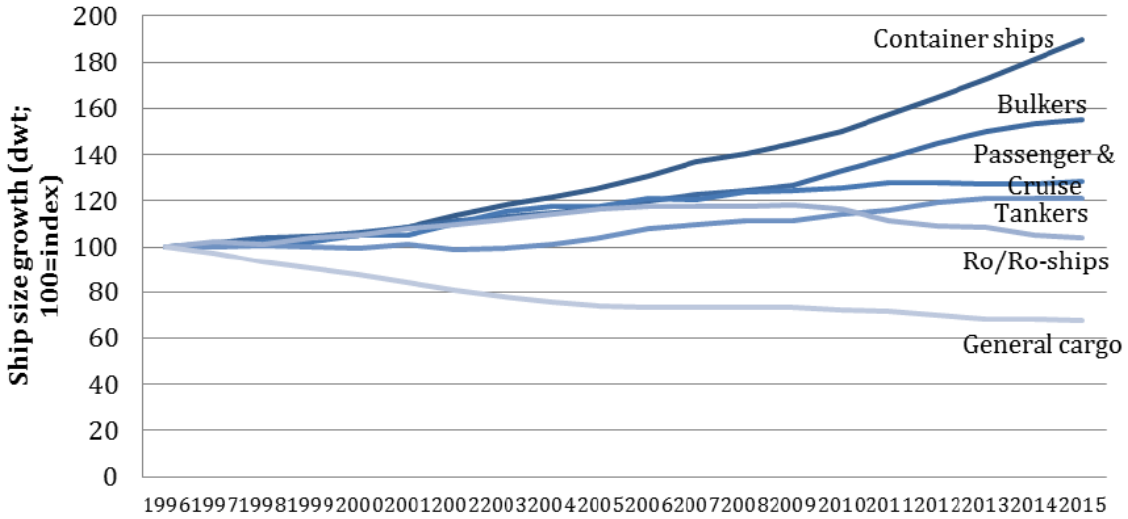


Figure 15. Ship size development of various ship types 1996-2015. Source: The Impact of Mega-Ships, ITF 2015.

Looking at the April 2015 order-book, the tendency for the coming years is that both maximum and average size of containerships will grow, as reflected from the ship orders that have already been placed for ships with capacity of more than 21,000 TEUs. Such ships are currently under construction and will be delivered over the years 2015-2017. In addition, many shipping lines owing no container ships of at least 18,000 TEU are now ordering such ships: the orderbook included 52 ships with capacity larger than 18,000 TEU: according to the Journal of Commerce, in the first half of 2015, CMA CGM placed an order for six vessels with capacities of 14,000 TEUs, after an earlier order of three 20,000 TEU ships. As for Maersk, the company recently ordered 11 ships with capacities exceeding 19,500 TEUs, and G6 Alliance members MOL and OOCL have each placed orders for 20,000 TEU ships.

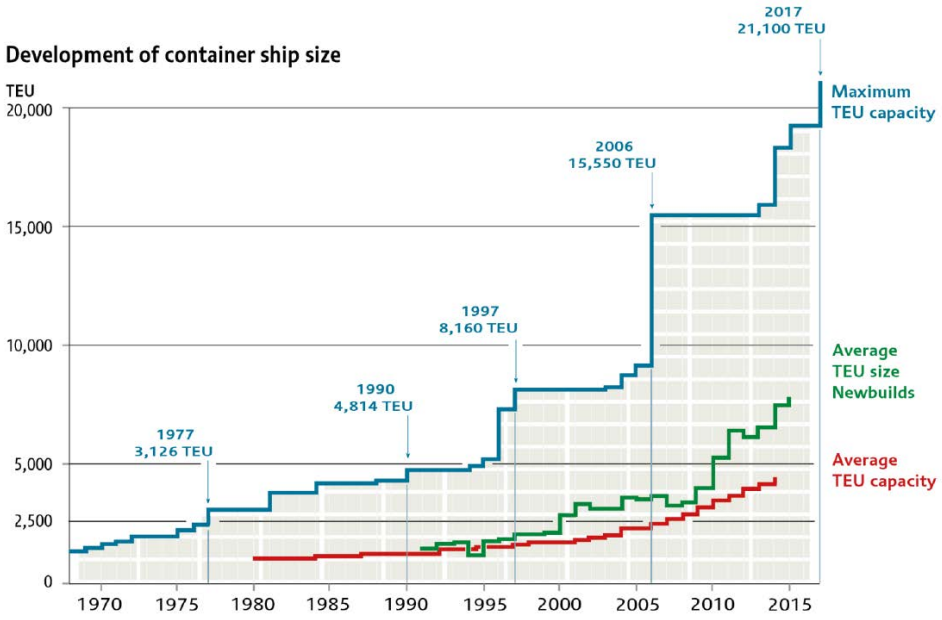


Figure 16. Development of the container ship size. Source: The Impact of Mega-Ships, OECD/ITF (based on data from Clarkson Research Services).

While mega-ships keep growing and benefit from the increased economies of scale that produce a cost reduction per container, they introduce additional pressure on ports, terminal operators and shippers, i.e.:

- Adaptation of port infrastructure: increase the depths of their navigable channels, quays, etc. The Organization for Economic Co-operation and Development (OECD) estimates that megaships are increasing landside costs by up to $400 million per year (one third for extra equipment, one third for dredging, and one third for port infrastructure and hinterland costs)

- Additional operators investments such as larger cranes capable of reaching across these ships, or changes to operations to provide extra night and weekend shifts of dock workers, and even face the costly automation of the terminal equipment.
- Congestion problems: mega-ships typically call at fewer ports, causing surges of unloadings.
- Mega ships force other large ships to relocate on smaller trades
- Increased probability of delays due to congestion, which is a concern for shippers.

Although the World Shipping Council states that energy savings of mega-ships and their range of service coverage makes them inevitable, ultimately at least part of the cost is translated to the port side of the transport chain. In this context, container terminals become forced to struggle to maintain quality of service, reduce service costs and increase the cargo throughput. As a result, storage space into a more limited resource (Steenken et al. 2004), operations are now more complex than ever before, congestion problems arise, and terminal capacity becomes a critical issue. In addition, new requirements to reduce port emissions (White Paper on transport, 2011) put additional pressure on terminal operators, which are facing new challenges to become environmentally friendly. On top of that, global peak of oil production in volume expected in 2015 (Maggio and Cacciola, 2012) threatens to increase fuel and electricity costs in the short/mid-term, which already amount to a significant portion of the terminals' operational expenditure.

In response to those challenges, terminal operators are making efforts to reduce inefficiencies and increase terminal throughput by introducing significant improvements in operational planning. In such context, complex strategies are required and particularly, storage location decisions for incoming and reshuffled containers have a great impact on the overall efficiency of container terminals since it is considered one of the critical operational problems (Park et al., 2010).

1.2 Future macro tendencies: towards an end of the current maritime transportation model in the context of the international crisis?

According to the International Transport Forum (ITF), bigger mega-ships will not be needed in the near future, as the point of optimal ship size is being approached. Latest ITF Statistics Brief of the Institutions like the Boston Consulting Group point out that the continued economic crisis has had an impact on the transportation that are summarized here as quick facts:

1) The decrease on the transport infrastructure investment

- Investment in inland transport infrastructure, as a share of GDP, has declined from a peak in 2009 to a record low (0.8%) in the OECD while the volume of investment has fallen back to 1995 levels.
- Investment levels in Central and Eastern European countries have nearly halved since 2009 in real terms, accounting for 1.0% of GDP in 2013 (compared with 1.9% in 2009).

- Western European and North American economies invest increasingly in rail while in Central and Eastern European countries the focus continues to be on roads.

2) On the decrease of the maritime transport

- Cost savings from bigger container ships are decreasing: it is estimated that increasing ship capacity from 19,000 TEUs to 24,000 TEUs would only cut costs by five percent,
- Further increase of maximum container ship size would raise transport costs, and so the transport costs due to larger ships could be substantial,
- Supply chain risks related to mega-container ships are rising,
- Public policies need to better take account of this and act accordingly,
- Overcapacity in container shipping: carriers are already having difficulty filling the larger vessels of the fleet; capacity oversupply is estimated to be about 20-30 %. According to the Boston Consulting Group, newbuilding order book shows overcapacity in the container shipping sector will last for several years.

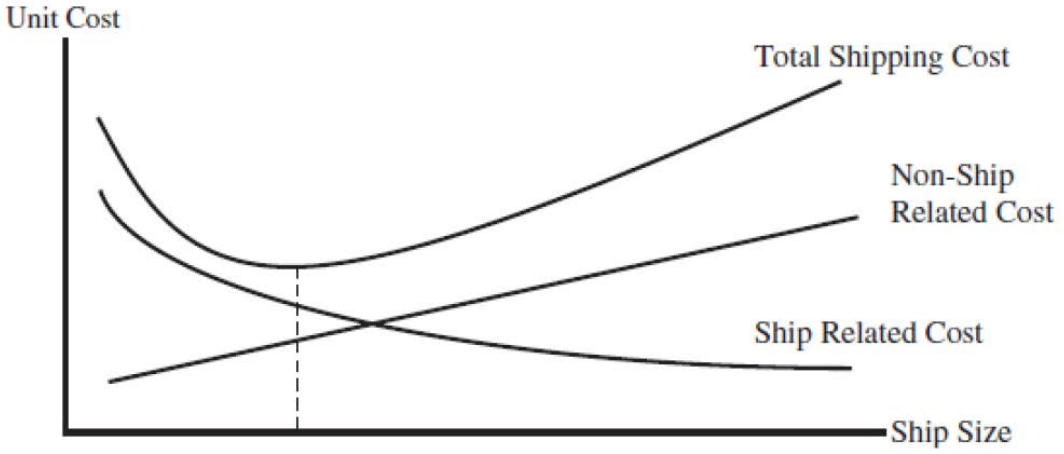


Figure 17. The total shipping cost including ship and non-ship related components (Gkonis and Psaraftis, 2009).

3) On the slow steaming practices

Slow steaming refers to the practice of reducing the container ships speed significantly (typically from ≈27 knots to ≈18 knots) with the aim to reduce fuel costs at the expense of longer travel times. The OECD estimates that "*between 55 and 63 percent (at least) of the savings per TEU when upgrading the vessel size from an early 15,000 TEU design to a modern 19,000 TEU design are actually attributable to the layout for lower operation speeds*". This practice was adopted in 2007 as a response to the rising fuel oil prices, but soon became a standard for the industry, to an extent that the design of new mega-ships is being optimized for lower speeds. Considering that the typical lifespan of containerships is around 10-15 years, it can be inferred that the mid-long term evolution of the fuel prices is a major concern for the industry.

Summarizing, future macro-tendencies suggest that, in one hand, the industry is shifting towards environmentally friendly management policies characterized by a more efficient use of the energy and lower emissions; on the other hand, container terminals are yet to suffer increasing pressure from the shipping companies to increase throughput and productivity and to cope with increasing traffic demands.

1.3 Container terminal management

Container terminals are interfaces between different means of container transportation, mainly maritime, road and railway. As the volume and frequency of container arrivals greatly varies among ships, trucks and trains, terminals also serve as buffers for temporary container storage to regulate the different container flows.

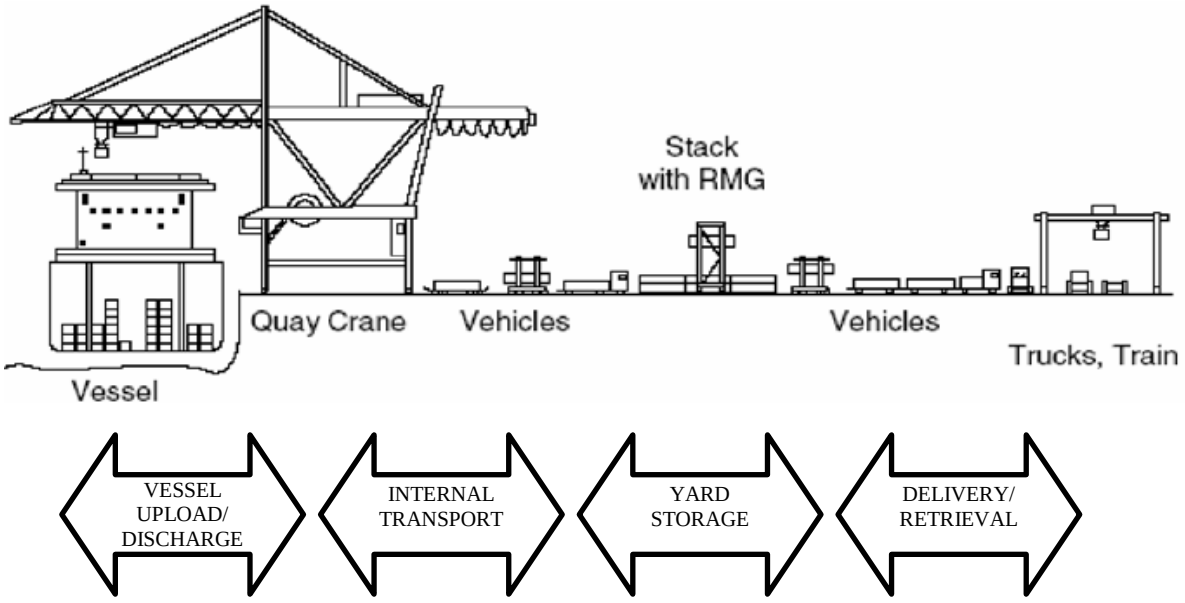


Figure 18. Container transport chain (Source: www.maersk.com).

1.3.1 Cargo flows

Several types of cargo flows typically coexist within a terminal: outbound (O/B or export), inbound (I/B or import), and transshipment containers. Import containers are unloaded from the ship and laid on the operations area (or on an AGV or IT), then transferred to the storage yard, stacked by a YC in a block, and then retrieved by another YC and delivered onto an external truck for dispatching. Export containers follow the reverse path: they are carried by trucks or trains to the terminal with the aim to be uploaded onto ships to be carried to the final port of discharge. Finally, transshipment containers differ from the previous in that they do not require the use of external trucks, as they are off-loaded from one ship and then loaded onto another ship.

Figure 19. STS cranes at quayside.

Inbound, outbound and transshipment containers usually coexist in the same block; however, operational rules enforce outbound containers to be placed in bays close to the waterside, while landside bays are preferred for inbound containers. In reality, as a result of the complexity of daily operations in container terminals, a degree of mixing of bays of import and export containers is always observed.

1.3.2 Container terminal subsytems

Although import, export and transshipment operations in a terminal occur simultaneously, when considering each cargo flow individually, terminal operations can be seen as a chain of consecutive links (Zondag et al., 2010) or independent transport subsystems, that is: ship to shore, transfer, storage, and delivery/reception. Each subsystem has differences depending on the terminal layout, size or handling equipment, as described next.

1.3.2.1 Ship to shore subsystem

Ship to shore (STS) operations are carried out by quay cranes (QCs) that interchange containers between the ships and the so called *operational area*. Depending on the container terminal, several types of QCs can be found, among which the STS cranes are the most common. Containerships are berthed alongside the quays; as several ships can be present at the same time, the traffic in the operational area can be complex.

1.3.2.2 Transfer subsystem

This subsystem is in charge of transferring containers between the quays and the storage yard and vice versa. Again, different types of vehicles can carry out these operations: automated terminals

utilize guided vehicles (AGVs) or lifting vehicles (ALVs); manned terminals make use multi-trailer systems or straddle carriers (SC).

Figure 20. AGVs from an automated container terminal.

1.3.2.3 Storage subsystem

Containers typically stay for several days at the terminal yard (Zhang, 2003) stored in vertical piles or stacks. Hence, the storage yard acts as a buffer for containers from the moment they are delivered to the terminal until they are reclaimed for final departure. Several types of Yard Cranes (YCs) are usually deployed to perform the stack and retrieval operations required to manage the containers: parallel terminals utilize rubber-tired or rail-mounted gantry cranes (RTG/RMG), whereas perpendicular terminals usually deploy SCs or ASCs, which are easily automated. The level of utilization also determines the type of equipment used to operate the yard: RMGs, RTGs and ASCs are usually found in highly occupied/ high density yard terminals, whereas SC are suitable for lower degrees of storage utilization.

Regardless the terminal layout, the storage yard is divided into areas called blocks. Usually, terminal blocks are of identical size and are composed of around 40-50 bays (in length), 6 to 9 rows (width), and 3 to 6 tiers high stacks. Storage blocks are usually laid either parallel or perpendicular with respect to the berth line, and so terminals are mainly classified as parallel or perpendicular. As the majority of container terminals in Asia utilize the parallel layout whereas the European counterparts prefer the perpendicular one.

Perpendicular yard layouts prevent the external trucks to access the storage area, and so delivery and receipt operations take place at the transfer point areas, located at the land tip of each block. One or two YCs manage the containers within each block.

Figure 21. Perpendicular Euromax Terminals in Rotterdam (The Netherlands).

Contrarily, parallel yard layouts allow internal and external trucks to travel through block aisles. Transfer lanes alongside the blocks are used by the trucks to interchange the containers with the YCs. The number of YCs per block may vary depending on the terminal, and under specific circumstances YCs can move from one block to another. As for the container mixing, very high throughput terminals mix inbound, outbound, and transit containers in the storage blocks, but other terminals separate cargo flows in dedicated blocks (no mixed blocks).

Figure 22. Parallel Container Terminal in Busan, South Korea.

1.3.2.4 Delivery and receipt subsystem

Finally, containers are interchanged with terrestrial modes of transportation such as trucks and trains. In parallel terminals, external trucks have access to the storage yard, whereas in perpendicular terminals the container interchange is carried out in designated *transfer areas* (TAs). Gates located at the boundaries of the terminal are the main infrastructures for external truck and cargo control. Railway is often used as terrestrial transportation, and in such cases internal vehicles are in charge of transferring the containers between the railway RMG crane and the YCs.

1.3.3 Planning problems and decision making levels in container terminals

The complexity of terminal operations requires careful preparation also with consideration to processes that take place at different time scales. Operational planning help operators to attain two conflicting objectives: in one hand, satisfy terminal customers by providing competitive servicing and pricing, and on the other hand to maximize profit while reducing costs, when possible. Meersmans and Dekker (2001), Vis and De Koster (2003) and others classify handling operations according to their time horizon in namely strategic (long-term), tactical (mid-term) and operational (short-term), as described in subsequent sections.

1.3.3.1 Strategic planning

Strategic planning refers to decisions made at design stages of the terminal: degree of automatization, terminal layout and capacity, handling equipment selection, etc. Terminal design is one factor affecting handling operations and their productivity (Wiese et al., 2011). These decisions are based on economic and technical feasibility studies, which help determining the optimal solution for the trade-off existing between the CAPEX, OPEX and projected sources of revenue.

At this stage there is a considerable degree of uncertainty (Saanen, 2009; Schütt, 2011), and the data needed for these studies must be estimated and projected: traffic composition and volume, revenue models, etc. In addition, important stochastic effects concerning the traffic data are difficult to estimate: annual workflow of the terminal, seasonal variations, peak factors, dwell time, etc.

Operations System	Required equipment per Quay Crane$^{(2+3)}$	Stacking Tiers [1-over-*n*-high]	Yard Capacity [TEU / ha]
Reachstacker & TTU	3–4 Reachstackers + 4–5 TTUs	3	350
		4	500
		5	950–1,000[4]
Pure SC	4–5	2	500
		3	750
RTG & TTU	2–3 RTGs 4–5 TTUs	4–5[5]	1,000
RMG & TTU (blocks parallel to quay)	2 RMGs 4–5 TTUs	4–5	1,000[6] (or more)
RMG & ShC (blocks perpendicular to quay)	2 RMGs 2–3 ShCs	4–5	1,000[6] (or more)
RMG & AGV	5–6	4–5	1,000[6] (or more)

Figure 23. Yard capacity as a function of the MHE (Brinkmann and Böse, 2011).

1.3.3.2 Tactical planning /Operative planning

At the operational level, decisions affect operations on a time horizon ranging from days to weeks, involving logistic processes such as resource allocation problems or equipment deployment, as illustrated in Figure 18. As indicated by Murty et al. (2005) operational decisions inherit the uncertainty of the information available at the time terminal operations are adopted, and so they will be likely modified by real-time decisions adopted later on.

1) Schedule and stowage plans of vessels

The stowage plan describes the exact position of each outbound container in the vessel once terminal operations commence, as well as the loading sequence of containers. Stowage plans must ensure vessel stability while minimizing operational costs and vessel turnaround times.

2) Berth allocation

Berth allocation refers to the allocation of vessels to berthing positions with the goal of minimizing ship waiting time (queues), operational costs, etc., and maximizing utilization of seaside resources (berths and QCs).

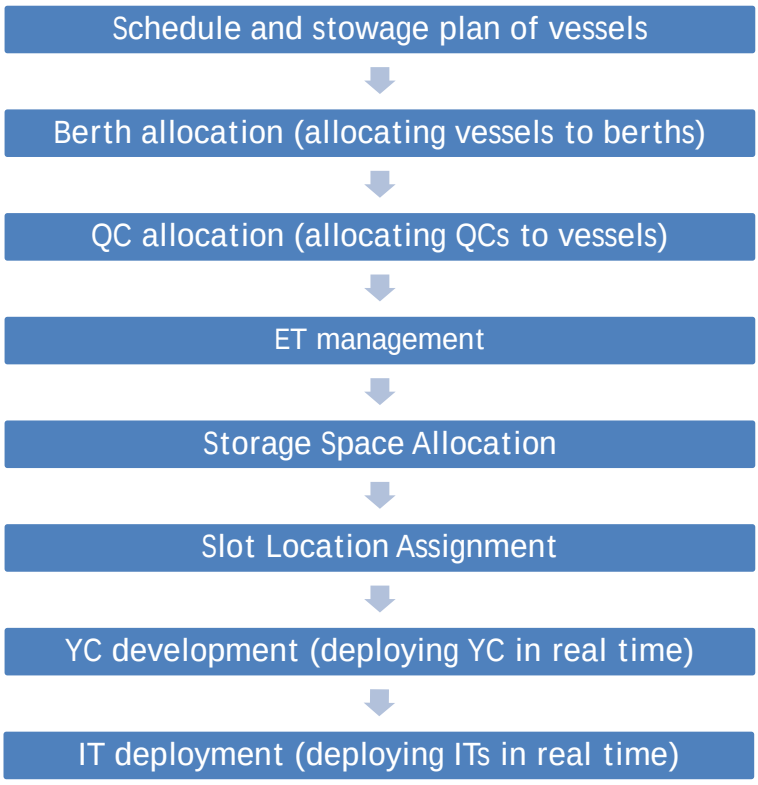


Figure 24. Hierarchical structure of operational decisions in a container terminal (Zhang et al., 2003).

3) QC allocation

Decision problems deal with the allocation and scheduling of QCs to work on berthed vessels. QC allocation greatly influences the turnaround time of the vessels and the throughput rate of the terminal.

4) ETs management

Management of external trucks involves the resolution of several decision problems. Arrival of external trucks is not usually well know, as customers do not always call beforehand to make appointments, hence online rules must be used to generate appointment times when they call to minimize ET waiting time and congestion in the road network. In addition, ETs (and ITs) routing and dispatching inside the terminal must be dealt with to minimize congestion on the terminal aisles.

5) Storage space assignment

This problem, which constitutes one of the main research topics of this Thesis, deals with the assignment of space in the storage yard for inbound and outbound containers. The goal is to place the containers in the optimal position in order to maximize yard productivity and minimize operational costs (reduce rehandling, YC deployment, inefficiencies, road congestion, etc.).

6) YC deployment

YC deployment aims the determination of the number of YCs allocated to each block in the working period, or determining the best time to move a YC from one block to another. Lastly,

7) IT/AGV/SC deployment

A number of ITs are allocated to each QC to ensure minimization of QC waiting times, and maximize vessel turnaround time. IT requirements are usually estimated in half-hour intervals, hence a plan is needed to minimize the total number of ITs hired each day, and to maximize their utilization.

1.3.3.3 Real-time planning

Container terminals are highly dynamic environments in which many operations are characterized by a significant degree of uncertainty, and therefore many decisions cannot be planned in advance. Hence, real time decisions are required in many circumstances, for example to assign vehicles to transportation orders, assign storage slots to stack containers, etc. Real planning decisions are often made by computers that carry out calculations based on real time data by means of complex algorithms built into Terminal Operation Systems. TOS are commonly implemented in modern terminals to help operators managing activities and allow recording terminal data for posterior analysis.

1.3.4 Characterization of terminal performance

As of today, the characterization of physical parameters that describe the performance of a container terminal is an essential task, and involves the acquirement of data regarding simultaneous events by sensors that send the real-time data to the TOS. The post-processing of data is used to describe the overall efficiency and performance of a terminal, which depends directly on the individual performance of the individual links of the transport chain (Figure 18). With respect to the storage yard, which constitutes an intermediate link of that chain, all upstream and downstream processes (container interchange between blocks and quay cranes, vessel upload and discharge performance, and external trucks and train operations) are strongly affected by the productivity of the yard cranes. As point out by Chen et al. (2003) yard performance alone is an indicator of a terminal's competitiveness.

Terminal management is usually quantified by the measurement of different key performance indicators (KPIs). One of the most common KPIs is the *vessel turnaround time* (in hours), which is intimately related to other KPI, the *QC throughput rate* (in movements/hour) and *berth productivity* (in movements per ship and per hour).

Container terminals are rated according to berth productivity (Figure 25), and therefore resources devoted to other subsystems are oversized to ensure berth operations are not constricted by other

links of the transport chain. Regardless the capacity of each subsystem, berth productivity indirectly depends on the performance of YCs, and also on the transport handling equipment that interchanges the containers between the QCs and the YCs.

TERMINAL	PORT	COUNTRY	2013 BERTH PRODUCTIVITY
APM Terminals Yokohama	Yokohama	Japan	163
Tianjin Xingang Sinor Terminal	Tianjin	China	163
Ningbo Beilun Second Container Terminal	Ningbo	China	141
Tianjin Port Euroasia International Container Terminal	Tianjin	China	139
Qingdao Qianwan Container Terminal	Qingdao	China	132
Xiamen Songyu Container Terminal	Xiamen	China	132
Tianjin Five Continents International Container Terminal	Tianjin	China	130
Ningbo Gangji (Yining) Terminal	Ningbo	China	127
Tianjin Port Alliance International Container Terminal	Tianjin	China	126
DP World-Jebel Ali Terminal	Jebel Ali	United Arab Emirates	119
Khorfakkan Container Terminal	Khor al Fakkan	United Arab Emirates	119

Figure 25. Top ten container terminals in 2013 according to berth productivity.

In addition to KPIs that characterize terminal productivity in global terms, specific aspects of the handling operations are also accounted for. With respect to yard operations, several common indicators are extensively used in this thesis. First, *YC throughput rates* measure the number of containers transiting through the transfer areas per hour. In addition, block performance is measured as the *Container Exit Times* (CET), which accounts for the time between the moment an import container is requested (and therefore added to the crane's workload list) and the moment the crane places that container on the external truck. Finally, with respect to export containers, block performance is calculated by the Vessel Service Time (VST), or the time it takes to load all the containers in the vessel.

It is worth noticing that CET is preferred over YC throughput rates to characterize block productivity. In reality, the BAP is a real-time decision problem, and therefore the arrival of containers to the sea TP may depend on the situation of the block at a given moment, i.e. sea ASC workload. In such case, the state of the block has an *upstream* effect, meaning that processes that goes first in the transport chain are affected by subsequent processes. However, the perpendicular model setup described Section 3.2 is such that the BAP is solved in advance; the arrival of containers to the TPs is deterministically given by the traffic generator. Therefore, the rates at which import and export containers are dropped on the TPs for final departure, which is considered a *downstream* effect, derive from the performance of the stacking algorithms.

Chapter 2

State of the Art

This chapter presents a general summary of the scientific literature regarding various aspects of storage yard optimization, including state-of-the-art models, methodologies, strategies and other contributions. Those studies constitute the basis and starting point of the research presented in this thesis.

Throughout time, research on topics related to container terminals have focused on analytical models, deterministic optimization methods, stochastic optimization model, and more recently discrete event simulation methods.

2.1 Literature reviews

Several literature reviews are worth the mention as they have a close relationship to the research presented in this Thesis. Literature reviews analyzed here do not only provide a useful general overview of the state of the art, but also provide a critical analysis of the work done until the publication date. This analysis helps inferring conclusions on diverse aspects of the state of the art, such as the evolution of the research and past research tendencies, or the identification of lack of research on determined areas and future lines of research.

Literature on container terminals is abundant and encompasses a wide variety of aspects; the problem approach can range from very specific - short term operational aspects to large-scale – long term generic aspects such as yard planning. Thus, when analyzing the main contributions found in the literature, it is important to point out not only the problem approach and methodology

used to solve it, but also to pay attention to three different aspects that constrict the behavior of the terminal:

- Terminal layout (parallel/perpendicular) and equipment (quay and yard),
- Container traffic typology (import, export, transshipment) and magnitude
- Operational strategies.

By classifying the work according to these criteria, it will be easier for the reader to position the contribution of the work in the general picture, and will also help establishing comparisons between the different contributions.

2.1.1 General reviews

Steenken (2004) provides a thorough review with more than two hundred references covering a wide range of issues: from terminal structure and handling equipment, to logistics, optimization methods, simulation, etc. One of the main conclusions of his work is that few studies consider 'integrated problems', that is, ship, berth, yard, and gate and utility agents for quay crane, gantry crane and transport, despite their importance for enhanced terminal performance. He also points out that major research is needed regarding the topics in the area of stochastic optimization and scenario based planning.

In their own words, Stahlbock and Voss (2008) present an *extension and update of Steenken (2004)*, reviewing research works on operations and methods applied on maritime container terminals. Their work investigate a number of aspects of container terminal operations, including berth allocation, stowage planning, crane optimization, terminal transport optimization, and storage and stacking logistics.

Carlo et al. (2013) provide another thorough overview of storage yard operations, for which they propose and use a classification scheme for scientific journal papers published between 2004 and 2012. The review also includes the material handling equipment used, current industry trends and developments, and discusses and challenges the current operational paradigms on storage yard operations.

2.1.2 Layout design

The review by Carlo et al. (2013) includes an analysis of the works on layout design. As they point out, the storage yard layout to be implemented is determined after deciding the level of automation and mechanical handling equipment to be used. Once this decision is made, the layout design is divided into two stages: first, the overall yard layout design, which aims to determine relative positioning of blocks in the yard and the required number of blocks; and second, block yard layout design, which focuses on the optimization of the block size dedicated to either import or export containers.

2.1.3 Storage Space allocation strategies

Many researchers have investigated operations efficiency in container terminals by optimization methods and operations research models such as storage and stacking logistics. Outstanding surveys of research into container terminal operations and decision problems can be found in the literature: Meersmans and Dekker (2001), Steenken (2004), Vis and De Koster (2003), and Günther and Kim (2006). Murty et al. (2003) provide a thorough description of terminal operations and strategies while providing the formulation needed to help developing decision support system (DSS). For a concise overview of storage and stacking logistics problems and decisions, refer to (Luo et al., 2011).

2.1.4 Simulation on container terminals

An outstanding review of simulation on container terminals is given by Cartenì and de Luca (2009). One of the main conclusions of such effort is that half of works reviewed adopt a stochastic approach to produce the numerical inputs for the model. Another interesting review can be found in kemme (2004). The work by Petering and Murty (2009) also provides an exhaustive review on simulation studies, containing thirty-eight citations.

2.1.5 Yard cranes deployment

The crane deployment problem has gathered less attention in the literature. The review by Kemme (2012) cites eight papers in total, out of which six are directly applicable to the perpendicular layout crane system and the other two to the parallel layout. Stahlbock and Vos (2008) provide another useful review on crane transport optimization.

2.2 Yard design

2.2.1 Yard space

Few studies deal with the determination of yard space (capacity) and handling equipment needed to operate a terminal, a topic on which more literature may be found in books (Thorensen, 1998, Kemme, 2013) or PhD thesis (Sharif, 2011). Kim and Kim (2002) proposed a cost model for import container yards that includes the space cost, the investment cost of transfer cranes, and the operating cost of transfer cranes and trucks. The authors provide the optimal solution in terms of the amount of space (slots in a bay) needed and the number of transfer cranes, and a sensibility analysis to assess the sensibility of the solution with respect to handling costs, arrival rate of ETs or yard cranes speed. The numerical examples showed that the optimal number of slots per bay was 22 and 17 for minimizing terminal operator cost and integrated total cost, respectively. No stochasticity is introduced in the parameters.

As simulation and emulation DES models became the trend, the use of these tools is extended to the planning and designing of container terminals (Veeke et al., 2003). With regard to the

calculation of yard capacity, Boll (2004) created a simulation model to determine the capacity of the container stacking area as well as the quay. Sgouridis et al. (2003) simulated a medium-size terminal in which containers are handled with SCs. The model is utilized to optimize various yard parameters (number of cranes, yard layout, stacking techniques and working shifts).

Complementarily, Brinkmann (2005) developed a study in which the required storage capacity was approximately calculated for different types of handling equipment with consideration of the main traffic characteristics such as the annual container turnover, average dwell time and hourly peak factor. Chu and Huang (2005) follow a similar approach by deriving a general equation that relates the total number of container ground slots for different yard sizes to the type of handling system (SC, RMG and overhead bridge cranes (OHBC)), the equipment dimensions, the transshipment ratio, and the average container dwell times.

2.2.2 Yard layout

Once determined the capacity, the next step in container yard design is the layout design. According to the literature review by Carlo et al. (2013), layout design studies can be divided into two streams: (1) overall yard layout design, including determining the number of blocks, and (2) block design in terms of length, width and height.

2.2.2.1 Overall yard layout

Several studies on container terminal design compare the parallel and the perpendicular yard layouts by means of numerical simulation. Liu et al. (2004) emphasized the effect of the yard layout on the terminal performance. They utilized a numerical simulation focused on automated import/export container terminal and concluded that the perpendicular layout yields better performance regarding QC moves and amount of horizontal transport equipment required. On the contrary, Petering (2008) stated that the parallel layout is preferable to the perpendicular layout, although in some cases a perpendicular layout outperforms a parallel one considering the QC rate. Further.

Kim and Park (2008) compared parallel and perpendicular terminal layout designs in terms of number of blocks and aisles. They elaborate a cost objective function that includes both the travel cost and the relocation cost. As external trucks can access the blocks in both layouts, travel functions are customized to account for the differences in traveling patterns. The parallel layout results in shorter expected travel distance for the same layout parameters, as well as for the best set of parameters, although no stochasticity is introduce in the model either. The optimal layout of an entire container yard, specified by the dimensions of a block and the number of aisles, was investigated again by Lee and Kim (2013). An optimization model is developed including the construction cost of the ground space, the fixed overhead cost of yard cranes, and the operating

costs of yard cranes and transporters. The total cost of the terminal is minimized under certain constraints related to road truck turnaround time and transporter cycle time. Results showed that, regardless the layout, wider blocks yield a better performance and a lower total cost in the yard. With respect to the comparison between the two types of layouts, the parallel layout is preferable to the perpendicular layout in terms of total cost.

2.2.2.2 Yard block design

Regarding the design of the storage block, Petering (2009) and Petering and Murty (2009) studied the optimal block width and length (measured as numbers of bays). A numerical simulation was used to assess the performance of a parallel and a perpendicular terminal for pure transshipment traffic. Both studied since account for the influence of the YC cycle times as well as the travel distance of road trucks and transport vehicles on the QC movements per hour. The results showed that the optimal block width ranges from 6 to 12 rows and block length between 56 and 72 since these values guarantee the highest QC work rate and greater YC mobility.

Lee and Kim (2010a) determined the optimal size of a single block (given as the number of bays, rows and tiers) of parallel and perpendicular terminals by considering the throughput requirement of YCs and block storage requirements. according to the following objective functions and constraints: minimizing the weighted expected YC cycle time for various operations subject to the minimum block storage capacity provided, maximizing the storage capacity subject to the maximum expected cycle time of a YC, minimizing the weighted expected truck waiting time for various operations subject to the minimum block storage capacity provided, and maximizing the storage capacity subject to the maximum expected truck waiting time.

2.3 Yard management

A significant proportion of the literature has dealt with the strategies to operate the yard. Chronologically speaking, as the bottleneck of the terminal operations is usually the at the quay cranes, most of the literature focused on this particular issue. With time, other subsystems of the container management sequence such as the yard management gathered more attention, and an increasing amount of studies on this particular research topic can be found in the literature.

2.3.1 Space planning principles

Regardless the type of planning strategy, widely accepted principles of space planning of export operations are implemented to optimize the space allocation process, that is, minimize the *operational costs* and improve the *service level*.

The first of such principles is the *Container Grouping*. According to Murty (2007), outbound containers of the same length, destination port, same liner service, and same weight class can be loaded into the same hatch of the vessel in any order. Therefore, such containers (up to 20 or more

in practice) can be stored in a single stack in any order as they will be retrieved from the stack without any reshuffling. Other common planning principles are described by Woo and Kim (2010). The *Nearest Location Principle* (NL) prioritizes the bays closer to the target vessel berth, thus minimizing the travel time of Yard Trucks (YTs). The *Least Relocation Principle* (LR) prevents from mixing groups of containers in the same stack, thus avoiding reshuffle movements needed to retrieve containers. The *Concentrated Location Principle* (CL), however, enforces containers belonging to a group to be placed on bays or stacks located as near as possible, in order to decrease the gantry travel of yard cranes during the ship loading operation. Finally, since excessive concentration may lead to interference among yard cranes during ship loading, the *Least Congestion Principle* (LC) favors the dispersion of container clusters among the blocks. Thus, the number of blocks among which the work is distributed depends not only on the existing yard layout, but also on the number and type of available handling equipment.

2.3.2 Types of yard operational strategies

Steenken et al. (2004) distinguished between two types of yard operational strategies regarding the use of space reservation. *Storage Planning* reserves areas of the yard for the containers bound to a determined vessel prior to its arrival. Conversely, other terminals make use of *Scattered Planning* when yard areas are no longer assigned to a specific ship's arrival but to a berthing place. Instead, the position of an arriving container in the yard is determined in real time according to the container grouping. Scattered planning leads not only to greater container scattering over the yard, but also to larger ground occupation, which reduces the number of reshuffles. According to Taleb Ibrahimi et al. (1993), storage planning may require the terminal to dedicate large amounts of empty space (up to 50%) awaiting future arrivals, and so they proposed a Dynamic strategy to improve the use of yard space made by the *Static strategy*. Thus, instead of reserving space prior to the arrival of export containers by truck, the *Dynamic Strategy* (often referred to as *Segregation Strategy*) makes use of a temporary storage area to rough pile containers until vessel assignment takes place and space is actually reserved in the yard. From that moment on, incoming containers are sent to reserved slots, and rough piled containers are then transferred to the assigned slot at their best convenience. This way, the dynamic strategy requires less ground space, virtually eliminating wasted space, at the expense of greater handling costs.

2.3.3 Container allocation problem (CAP)

Import and export containers bound to a vessel are respectively unloaded and loaded in a sequence determined by the so-called loading plan (Figure 26). A load profile (an outline of a load plan) is usually sent by an agent to the terminal operating company several days before the ship's arrival. The load profile specifies only the container groups that are stowed in each ship cell. For export containers, ship's cells are filled with any containers from its assigned group, and so the loading

operation can be eased by sequencing containers in the marshalling yard, hence optimizing the handling effort.

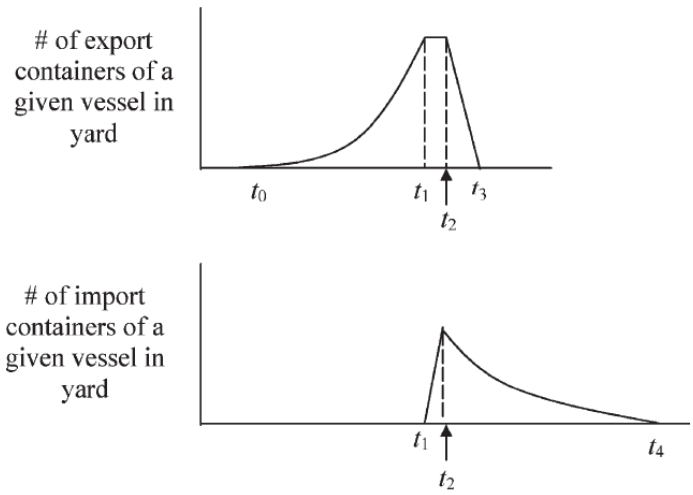


Figure 26. Export (top) and Import (bottom) containers in the yard associated with a vessel versus time (Zhang et al., 2003). t_0 = start unloading EC, t_1 = start time of downloading IC, t_2 = end time of download IC and start loading EC, t_3 = end time of loading EC, t_4 = end time of IC pickup.

When space is reserved for a vessel, the planning process is usually broken down into a two-stage problem (Murty, 1997): *Block Allocation*, which aims to determine the amount and location of bays devoted to for a given number of containers for a time horizon of hours or days, and *Slot Allocation*, in which a bay, stack and tier for a container is found within the block, and thus can be considered as a real-time problem. When considering the *Block Allocation Problem*, optimal solutions are sought while considering several factors: yard crane (YCs) workload balance, which in turns depends on the current distribution of containers in the yard, and the deployment of ITs and ETs to avoid congestion in the roads. A different approach is followed to solve the *Slot Allocation Problem*, for which the aim is to reduce the number of container rehandling in one bay or within a group of bays.

Different strategies are adopted to address the CAP for import and export flows, as well as for conventional and automated container terminals; therefore, several types of stacking strategies are followed. The next sections describe the main contributions to the slot and the block allocation problems, as well as literature that deal with both problems at the same time.

2.3.3.1 Slot allocation problem (SAP)

The *Slot allocation* problem may not be considered a stage of the planning process itself, but as a real time decision made to determine the best slot for the container among a target bay or a number of pre-allocated bays. In general, the main objective of slot allocation is to minimize the overall rehandling effort needed to retrieve all the containers in a bay. As inbound and outbound

containers have different departure patterns and are not mixed in the same bay, strategic decisions need to be made in advance to determine the ideal bay configuration. To this matter, the availability of departure information and the loading sequence of the ship stowage plan play a significant role in how the inbound and outbound container stacks are configured.

Two main aspects of the slot allocation problem are usually considered: the number of rehandling, and stacking strategies to reduce the amount of rehandling.

Regarding the calculation of rehandling moves, few methodologies and algorithms are available to evaluate the number of unproductive moves. In general, these studies base their formulations on probabilistic methods and expected values, and relate directly the average height of the stacks to the expected replacements. Sculli and Hui (1988) developed the first relation between stacking height and reshuffles by using a simulation model. Later, Watanabe (1991) developed a more conventional method to quantify the overall amount of replacements called Index of Selectivity (IOS). Later, Ashar (1991) pointed out that Watanabe's IOS should take into account the storage density and handling convenience.

Later, considering the random retrieval of import containers in a bay, Kim (1997) proposed a methodology, based on an exact procedure and a regression analysis, to calculate the expected number of unproductive moves to retrieve a container and the total number of rehandles required to pick up all containers. The total number of unproductive moves directly depends on the stacking height and number of rows; hence, it can be concluded that higher stacks increase handling effort because the number of unproductive moves increases proportionally.

However, retrieval of import containers is not usually random, as some information regarding the container dwell time or arrival time is always available. DeCastilho and Daganzo (1993) extended the analysis to an entire bay, presenting a method to measure the amount of handling effort for two different strategies for stacking import containers. The first strategy tries to keep all stacks the same size (allowing containers to be stacked on top of containers bound to other vessels), while the other segregates containers according to arrival time. The strategies are compared in an idealized situation.

Kim et al. (1999) also analyzed the slot allocation of outbound containers, and a methodology is proposed to determine the best location for the container considering three weight categories and reduce the number of relocation movements during the loading operation of a vessel. The work by Chen (1999) and Chen et al. (2000) on import containers related the available storage capacity and stacking height, which ultimately depends on the operational efficiency.

Kim et al. (2000) introduced the container weight information to analyze the SAP in the first stage. Dynamic programming was then used to solve the problem with the objective minimize the

number of relocation movements during the loading operations of a containership. They also assume that containers are relocated no more than once; however this assumption may not be always correct, as analyzed in Chapter 5.

Kang et al. (2006) proposed a stacking strategy based on uncertain weight information for export containers, with containers being classified in three categories (heavy, medium, light). They applied a simulated annealing search algorithm to find a good strategy and developed a methodology to calculate the expected number of container rehandles.

As the relocation of containers may produce additional handling effort, Kim and Hong (2006) proposed two methods for determining the sequence of relocation movements inside a bay, considering the order of retrieval of export containers is known. The branch-and-bound (B&B) algorithm outperforms the decision rule based. Ünlüyurt and Aydin (2009) also find exact locations of relocated containers while retrieving all the containers from a bay according to a predetermined order. Two versions of the problem are solved considering whether the retrieval order applies to individual containers or groups of containers. The model is formulated and first solved via a branch and bound search with the objective of minimizing the number of relocations. Imai et al. (2002, 2006) introduce the constraints imposed by the vessel stowage plan to generate a vessel loading plan sequence that satisfies the ship's stability requirements (metacentric height, list and trim) to minimize the number of rehandles.

Huynh (2008) evaluated the effects of storage policies and import container dwell time on the terminal throughput and rehandling productivity. The study considered two storage strategies for import containers regarding the possibility of mixing new containers on top of old ones or not. Monte-Carlo simulations were conducted to evaluate the effect of dwell time on throughput and rehandling productivity. Results indicate that increasing container dwell time lowers throughput and average stack height for the non-mixed storage policy, increasing rehandling productivity. As for the mixed storage policy, increasing container dwell time raises throughput and average stack height – resulting in a decrease in rehandling productivity.

Wan et al. (2009) also studied the allocation problem considering an entire sub-block composed of several bays. The static version of the problem is solved with a integer program formulation, and then propose a heuristic method to reduce the computational time, with variants of the IP[4]

[4] (Wikipedia) An integer programming problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers. In many settings, the term refers to integer linear programming (ILP), in which the objective function and the constraints (other than the integer constraints) are linear.

model embedded and run in the rolling horizon fashion. Then they consider a dynamic version of the problem, with containers being stacked and retrieved from the block, concluding that the heuristic approach is capable of reducing reshuffles within a reasonable computation time.

Kozan and Preston (2006) developed a genetic algorithm model, a tabu search and a tabu search/genetic algorithm hybrid to solve the storage location in order to minimize the turnaround time of all the containerships in a yard with marshalling area. Their results indicate that reducing the maximum storage height results in lower turnaround times.

2.3.3.2 Block allocation problem (BAP)

One of the first contributions is due to Teleb-Ibrahimi et al. (1993), who focused on the determination of the amount of space to be allocated for outbound container bound to each vessel, based on a cyclic space requirement in container terminals, but did not consider the storage locations of containers in the yard. In their work, yard, ground space for a ship must be reserved as soon as containers for that ship start to arrive, and so almost 50% of the space may be empty awaiting future arrivals. The authors proposed two different handling and storage strategies (static space allocation and dynamic strategy) and developed an operating procedure and a heuristic algorithm to determine the minimal storage space needed to implement both strategies. In addition, their procedure was capable of minimize and predict the amount of handling work, but their model does not specify the terminal configuration.

Kim and Kim (1994) proposed a methodology to determine the amount of space allocated for export containers to be loaded in a ship, which is then released by the actual loading sequence. A quadratic programming model is developed to minimize the YCs handling cost under the several space-related constraints. Later, Kim and Kim (1999) focused on the stacking of inbound containers using a segregation strategy, considering also the case of a dynamic space requirement, and provide a formula that relates the stacking height and the number of rehandles.

Zhang et al. (2003) developed a rolling-horizon approach in order to solve the BAP for import and export containers in a RTGCs parallel terminal of 10 blocks of 6-stacks, 5-tier bays. The number of bays in each block is different, ranging from 7 to 42. The 3 day planning horizon is decomposed into six 4hour periods. The solution to the problem for each period is decomposed into two stages. The first stage seeks to determine the number of containers bound to each vessel to be placed in each storage block by balancing the workloads among blocks. The second stage aims to determine the number of containers associated with each vessel that constitutes the total number of containers in each block. It is worth noticing that departure time for import containers is known in advance, and therefore the exact workload for each YC can be accurately estimated during the planning period. Only the first day of the plan is executed and a new three-day plan is formed at the end of the first day (beginning of the second day) based on the latest information.

A mathematical programming model is proposed to solve each stage, and the numerical simulations show the workload imbalance in the yard is reduced, hence helping avoid possible bottlenecks in terminal operations. As Zhang et al. (2003), Bazzazi et al. (2009) considered the allocation of the inbound/outbound containers to the storage blocks at each time period with aim of balancing the workload between blocks in order to minimize the storage/retrieval times of containers, but they also consider the container typology at the time of making the decision on the allocation of containers to the blocks using a meta-heuristic approach (that is, a genetic algorithm) to solve the programming model. The work by Nishimura et al. (2009) also analyzed this problem in a pure transshipment terminal by proposing a heuristic method whose solution is based on a Lagrangian relaxation technique[5].

Kim and Park (2003) analyzed the BAP for outbound containers. They proposed an analytical model to compare two dynamic allocation methods: the least duration of stay, and the sub-gradient optimization heuristic algorithm[6] (SGHA). The objective function seeks to minimize the travel cost between the apron and the marshalling yard, which depends on the position of the allocated space. The duration of stay based decision rule performs almost as good as the sub-gradient optimization, but requires much less computational time.

Lim and Xu (2006) proposed a new metaheuristic procedure to determine the minimum required space, but they do not specify the type of container traffic, where space requests are allocated according to their priority, from the highest to the lowest. The so-called critical-shaking neighborhood search (CSNS) starts from an initial random sequence, by picking some critical requests, then shaking their priorities randomly, and finally exploring a local search.

Up to this point, the amount of space reserved for each planning period is equal to the number of containers to be stacked in the yard during that period. Woo and Kim (2011) investigated allocation strategies for outbound containers in which the amount of space is allocated taking into account the arrival rates of containers, or the length of the planning period. Irregular vessel arrivals and numbers of containers downloaded are used to feed the model. Best results are obtained when considering the square root of arrival rates, which reserves empty stacks for a container group in proportion to the square root of the arrival rate of containers in that group. The

[5] Lagrangian relaxation is type of relaxation method, which approximates the solution to a difficult problem of constrained optimization by solving a simpler version of the problem. The approximate solution provides useful information about the original problem.

[6] The SGHA is a well-known solution procedure for solving integer programming problems.

study also indicates that more detailed simulation models are needed to assess the impact of different reservation rules and values of parameters of each rule.

As for pure transshipment terminals, space is consigned in entire sub-blocks. This way, containers are unloaded and stored according to their destination vessel, reducing the number of reshuffles and enhancing terminal productivity. Optimization methods are thoroughly employed to solve an objective function with multipurpose constraints. I.e., Lee et al. (2006) minimize the number of yard cranes to deploy with a high/low workload balancing protocol used to reduce the potential traffic congestion of prime movers and determine the location where unloaded containers should be stored. Han et al. (2008) extended the previous study by considering not only the number of incoming containers and the smallest number of yard cranes to deploy in each shift, but also the storage locations of incoming containers. Jiang et al. (2012) proposed a space-sharing yard template to reduce the under-utilization of space while ensuring the efficiency of yard operations. Later, another meta-heuristic method was developed by Zhen (2014) to investigate the yard template of a pure transshipment parallel terminal. The number of containers downloaded and loaded onto vessels fluctuates, making the number of sub-blocks bounded to each vessel uncertain. The study reveals that the proposed meta-heuristic algorithm outperforms two different strategies in terms of the associated space and handling equipment costs.

Especially in large terminals or terminals with low workloads, export containers arriving too early (i.e., before the vessel stowage plan is available or the berthing position is known) may be stored (pre-marshalled) in separated areas, and then stacked in the yard in order to minimize vessel loading times. The location assignment problem with the utilization of marshaling areas has been also addressed in several studies. To this regard, Kim and Bae (1998), Imai et al. (2002), Hirashima et al. (2006), Lee and Hsu (2007), Lee and Chao (2009), Han et al (2008) and Fan et al. (2010) are some of the most significant contributors.

2.3.3.3 Simultaneous Block and Storage Position Assignments

Fewer studies analyze the BAP and SAP problem as a whole. Murty et al (2003) points out that the space planning process is highly inter-related to the ETs and ITs dispatching and routing, and need to be studied together. Considering planning periods of 4 hours (the beginning or the ending half of a shift), they propose a three stage approach to solve the CAP: first, solve the block assignment problem at the beginning of the period with the objective of minimize the differences in occupation of the blocks at the end of the planning period, which is solved by linear programming; second, a dispatch policy with the double goal of minimizing congestion at the blocks (YCs) and the on the roads (ETs and ITs), and third the slot allocation problem, which is made with the objective of minimizing the total amount of reshuffling in this block.

Dekker et al. (2006) created a simulation model based on the ECT's DDE terminal. This perpendicular automated terminal has one ASC per block, and import and export containers can be mixed in the same blocks, but not in the same bays. The average utilization of the base stack configuration is 50%. Block allocation is made by balancing the workload among ASC, or random to evenly spread the workload among the blocks. They distinguish two types of stacking strategies: category stacking and residence time stacking. The former strategy assumes that containers of the same category (e.g., having the same size, destination, weight, etc.) are interchangeable, and can thus be stacked on top of each other without the risk a lower container in a stack is needed before the ones on top of it have been removed. The latter strategy does not use categories, but instead looks at the departure times of the containers: a container can only be stacked on top of containers that all have a (planned) departure time that is later than the departure time of the new container. They concluded that category stacking performs better than random stacking, but information on the departure time of containers needs to be known in order to create ordered piles.

Since the evaluation of offline search algorithms require quite a heavy computation involving a detailed operational simulation, Park et al. (2011) investigate an online search algorithm to stack containers. They utilized a single-berth, seven blocks perpendicular ASC terminal, with each block consisting of 41 bays, 10 rows and 5 tiers of 20 ft. containers. Two non-cross over ASCs are deployed in every block, and the yard is initially occupied to 70% of its maximum capacity. They evaluate the stacking policy represented as a vector of weight values for the cost of (1) stacking an incoming container, (2) retrieval of the container, (3) rehandling, and (4) the waste of space for the stacks. The stacking policy in each period is dynamically adjusted to optimize QC delay time, AGV waiting time, and truck waiting time. Each simulation period is followed by an evaluation period in which the optimal values of the weights given to each criterion are calculated, and then used in the next simulation period. Results from the simulation support the conclusion that online search is a good option in dynamic settings where there is not enough time for computation before taking actions. However, the cost model is deterministic and the simulation model does not take into consideration the effects of YC interference, inventory size, etc. that can be reproduced by a DES model.

Chen and Lu (2012) analyzed the problem for outbound containers in a parallel terminal. 10 blocks are used for stacking outbound containers, each consisting of 20 bays. Each bay has 6 stacks, 4 containers high. Import and export traffic containers are considered, although the volumes of traffic or the block occupation are not explicitly indicated. For each simulation interval, the block allocation problem is solved using a mixed integer programming model, while the slot allocation problem is dealt with by means of a hybrid sequence stacking algorithm whose performance is compared with random and vertical stacking algorithms. In the study, weight

distribution of containers arriving to the terminal on each plan period is assumed to be know. The performance of the proposed dispatching algorithm, measured as the amount of rehandling, outperforms the benchmarking algorithms.

Borgman et al. (2010) utilizes a DES model that reproduces 6 blocks of the EDT perpendicular terminal, which is operated by single RMGs. The simulation considers the detail motions of the cranes, and import/export traffic is pre-generated and fed into the model. The inventory size is not directly indicated, although a value of 42.1% is specified for one of the experiments, which is relatively low. As a consequence, the conclusions are difficult to extrapolate to higher degrees of inventory occupancy. Several stacking algorithms are used to investigate the efficiency of the slot allocation problem and reduce rehandling. One algorithm uses knowledge about container departure times by stacking containers leaving shortly before each other on top of each other. Thus, the algorithm makes use of container departure time information. The second algorithm is used to analyze the trade-off between stacking further away in the terminal versus stacking close to the exit points and accepting more reshuffles. Unlike other studies, no space reservation is made to solve the block allocation problem. Instead, two online approaches are compared: (1) a random block allocation, in which the block is randomly selected, and (2) a leveling algorithm, in which preference is given to lowest stacks, and then to the stack closest to the sea side TP. Results show that even rules with a limited number of classes for the remaining residence time work very well. In addition, the single RMG system may improve its performance when stacking further from the TP (longer travel times) to reduce the incidence of reshuffling.

2.3.4 Segregation stategies

Segregation strategies were further investigated by Kim and Kim (1999) for inbound containers in a parallel terminal with 20-30 bay-long blocks of 6 stacks width and 3-4 tiers height. They develop a cost objective function that includes the total cost of space, yard cranes, and internal trucks, and they determine the best combination of space and number of yard cranes, although the level of occupation in the blocks is not explicitly indicated.

2.3.5 Yard Cranes deployment

Several common types of crane systems for parallel terminal can be found depending on the number of cranes deployed and their crossability. According to Stahlbock and Vos (2008) most of the literature focus on single crane operations (i.e., Ng and Mak, 2005a,b), while less attention is paid to the routing or scheduling algorithms for multiple cranes.

The thesis by Zyngiridis (2005) presents IP models to prescribe routes for SRMG and TRMG, concluding that block length and occupancy of the block significantly affect the performance of

the SMRG, while the TRMG performance is only influenced by the block length. However, the RMGs seem to spend little time in the trolley and hoist movements.

Saanen and Valkengoed (2005). They compare four types of YC systems for automated perpendicular terminals: single RMG, Cross-over RMG, Twin RMG. YC control operates in two ways: FIFO rule (handling the orders in the order of the order list as created) and "nearest neighbor search", a heuristic which aims to reduce empty travel distance. Every time a RMG finishes a task, an order is chosen from the first five orders available in the order list, with its pick up location closest to the current location of the RMG. With regard to the traffic, the study also analyzes two scenarios, an average scenario and peak scenario, differing in the number of external truck moves per hour (20 and 80 mph respectively). The inventory size is 70%. Results indicate that the twin RMG is more productive, although the CAPEX and OPEX are also higher.

Choe et al. (2007) consider crane a deployment strategies for TRMG based on a local-search-based real-time scheduling method in an automated container terminal, and conduct simulation experiment in which their algorithm outperforms a heuristic-based method.

The single block twin-crane scheduling problem with both storage and retrieval jobs was also investigated by Vis and Carlo (2010) developed a mathematical model to minimize the makespan for both cranes. An algorithm to derive a lower bound for the makespan was then introduced and a simulated annealingbased heuristic was proposed to solve the problem. Stahlbock and Voß (2010) provide a simulation study of a DRMG system based on operational data from Container Terminal Altenwerder (CTA) to investigate the effectiveness of different online algorithms for the sequencing and scheduling of jobs at a storage block.

The crane scheduling problem for the innovative TriRMG is first regarded by Dorndorf and Schneider (2010), who proposed a scheduling approach combined with a crane routing algorithm to compare crane productivities measured during the simulation of an isolated yard block.

The review by Carteni and de Luca (2009) provide gantry crane operation times from other works, as indicated in Table 1. However, the parameters used to characterize such distributions do not depend on the block configuration, crane speed and acceleration, or cargo weight.

Yun and Choi (1999)	exponential	mean = 1.00 (min)
Merkuryeva et al. (2000)	triangular	40' loading : mean = 6.00 (min) 40' unloading : mean = 4.00 (min) s.d. = 0.41 (min)
Lee and Cho (2007)		mean = 1.55 (min) s.d. = 0.08 (min)
Bielli et al. (2006)	deterministic	mean = 1.50 (min)

Table 1. Gantry crane operation time distributions.

Park et al. (2010) proposes a method of optimizing a stacking policy using a multi-objective evolutionary algorithm (MOEA). Their stacking policy takes account of both the locations for the incoming containers and those for temporary movement within the yard. Although details on the simulation model are not provided, the authors show that the obtained Pareto set of stacking policies provides satisfactory service levels on both the seaside and the landside.

Park et al. (2010b) also provide a realistic simulation model of an automated block of 40 bays with twin RMGs. As soon an RMG finishes a job, the algorithm schedules a new job based on two real-time methods for a given fixed-length look-ahead horizon: a heuristic-based and local-search-based. In addition, rehandling operations are treated as independent jobs to reduce the delay of the crane operations. Results show that AGVs and ETs waiting times are significantly reduced due to the increased utilization of RMGs through cooperation. The probability of interference of RMGs is also provided.

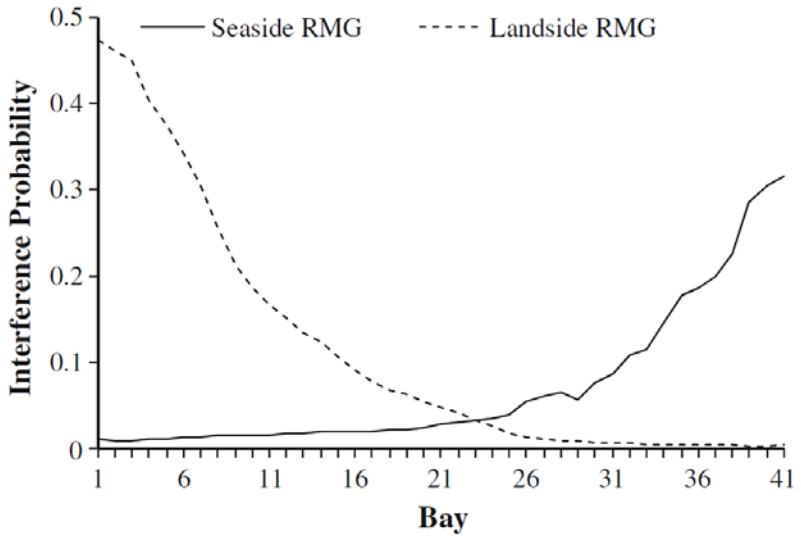


Figure 27. Interference probabilities of RMGs depending on the pickup/drop-off bay locations. Bay 1 and Bay 41 are the seaside and landside ends of the block, respectively (Park et al., 2010b).

Speer et al. (2011) develops a procedure to deploy twin ASC of the Container Terminal Altenwerder (CTA) in Hamburg, Germany. Each block is composed 37 bays of 10 stacks and 5+1 tiers. They utilize a branch and bound algorithm to create the sequence of jobs for each crane, minimizing delays for the jobs and the cycle times of the cranes. In addition to in-motion times also other parts of the cycle time, as waiting and blocking times resulting from other cranes, are taken into account in the scheduling approach.

The work by Kemme (2012) investigates the design of rail-mounted-gantry-crane systems. He conducted a simulation to evaluate the effects on the yard and terminal performance of four rail-

mounted-gantry-crane systems (single, twin, double, and triple crane) and 385 yard block layouts differing in block length, width, and height. One of the conclusions of this work is the strong relation found between the stacking height and the average waiting time of SCs for waterside retrievals. This correlation is found for all crane systems. The triple crane system performs better than its counterparts. Double and twin crane systems showed intermediate performances for most layouts, and the single crane system performs worst. The performance of the two cranes system improves as the block increases.

Works on the deployment of multiple cranes in parallel terminals can be found in Cao et al. (2008) and Bohrer (2005), who present MIP models for a DRMG and RMG crane systems respectively. In the second case, the author presents two models for crane scheduling and shows that the problems are NP-hard in the strong sense; hence heuristic solution procedures are developed and evaluated by numerical simulation.

2.4 Simulation

As indicated by Brinkman (2005), the design and planning of a container terminal depends on a large number of constraints and boundary conditions like the area layout, the operational requirements, or even legal restrictions that vary from on location to another. For this reason, each container terminal requires an individual solution. To this respect, Dekker et al. (2006) indicated that not only the performance but the design of container terminals can be drastically improved through detailed simulation. As a consequence, during the last years advanced simulation-based modelling is being extended to the terminal planning and design process, and hence detailed simulation is also becoming more and more relevant in the literature. The main advantage of simulation compared to traditional approaches is that models enable designers or planners to investigate the subject in a cost-efficient way and allows for the consideration of the dynamics in the system (Saanen, 2011). Vis and de Koster (2003), Steenken et al. (2004), Stahlbock and Vo β (2008), and mainly Angeloudis and Bell (2011) are extensively cited studies that follow the simulation approach.

Another good example can be found in Liu et al 2002 and Liu et al 2004. The authors consider a single-berth facility, but their simulation model is very complete as it provides measurements related to several performance indicators: QC throughput, average vessel turnaround time; yard utilization, truck productivity, and YC productivity. In addition, they analyze the sensibility of the performance indicators with respect to the input parameters (e.g. the type of handling equipment used, the vehicle fleet size, or the layout of the terminal).

Numerical models also have restrictions; as indicated by Petering et al. (2009) most simulation models on container terminals are limited, as are restricted to only one vessel at the same time and to simulation periods in the order of one day. Their work, in addition to Petering and Murty

(2009), overcome these limitations and provide a complete DES model to analyze the layout of parallel multi-berth container terminals. However, it is often said that most common disadvantage is that results generated through model simulation have a strong dependence on the experimental setup. In reality, this fact strengthens the idea that, as terminal operations comprise a multitude of highly interrelated processes, the complex interactions between the transport systems may have a significant influence on the upstream and downstream processes. As a consequence, simulation techniques are considered more suitable than optimization methodologies or analytical formulation, as they easily introduce stochasticity in the analysis or reproduce intricate handling procedures.

2.5 Energy expenditure

Although abundant literature analyzes the operational cost associated with yard operations and some works include the fuel as part of the variable cost related to equipment deployment, little work is found on the specific analysis of energy consumption.

Discussion on energy consumption and efficiency of container terminals at the strategic level is found in Rijsenbrij, 2011; or Wijlhuizen, 2008, while more analysis may be found about the overall electrical consumption of Terminals. For example, Thanh (2012) provides a method to estimate the electrical usage and demand at container terminals, and provides values for the electrical consumption of the handling equipment. However, the utilization of the terms “peak demand” and “maximum demand” for cranes may be misleading, as the values shown differ greatly.

At the operational level, the first contribution to energy consumption in container terminals is found in Chang (2010), who focuses on the berth allocation and the quay crane assignment problem by introducing a multi-objective function which was formulated as a programming model and solved by a genetic algorithm.

Container weight was also considered by Hussein et al. (2012) so as to minimize fuel consumption using a genetic algorithm in order to solve the so-called block relocation problem. Later, Xin et al. (2013a, b and 2015) proposed several models to achieve optimal performance of QC, AGV and ASCs, balancing the handling capacity and energy consumption. Although they assume fixed travel distances for the equipment, especially for the ASCs, and therefore the setup is somehow simplistic.

2.6 Objectives and methodology of the Thesis

As a general objective, this thesis aims to continue research topics related to the optimization of the container terminal operations with the focus on the storage yard. As previously indicated, a terminal yard is a complex subsystem that acts like a buffer for storage, handling and transport

operations of import, export and transshipment flows. As in previous studies, this work takes into consideration two different aspects of yard operations which constitute a primary concerns for operators: the *efficiency*, by accounting the energy consumption of yard cranes, which is intimately related to the OPEX, and the *productivity*, which indicates the quality of service of the terminal as perceived by clients. Efficiency and productivity often come at the expense of the other, therefore optimization often relies on decisions of strategic nature. Anyhow, both concepts depend not only on productive movement stack and retrieve containers in a block, but also secondary operations needed to rearrange containers, i.e.:

- Unproductive movements during retrieval of import and export containers,
- Inefficient handling of heavy containers,
- Inefficient housekeeping operations,
- Inefficient crane management, etc.

As it is obvious, operational inefficiencies may lead to significant cost increase and higher engine emissions from yard equipment, and so this Thesis aims to characterize and quantify such inefficiencies.

In addition to the efficiency and productivity of yard operations, a second general aspect has been inferred from the review of the state of the art: as time passes and the literature tackles with problems of increasing complexity, more sophisticated simulation models are adopted as study approach. As a consequence, a second objective is to adopt a methodology of analysis based on the realization of extensive numerical computations, for which two custom Discrete Event Simulation (DES) models have been built from scratch in Matlab to reproduce the yard operations in detail. In addition, as this Thesis belongs to the pilot program of the Industrial PhD, one of the main objectives is to contribute to the development and the innovation of the industry. Hence, the development of these two models can be used both for research and industrial purposes directly.

DES models have several advantages over optimization models frequently used in the literature (Cartenì and de Luca, 2009), i.e.:

- Allow a very high detailed and realistic representation of terminal processes and characteristics, and make -computer-generated strategies/policies more understandable.
- Overcome mathematical limitations of optimization approaches, i.e. study the effects of stochasticity in the analysis,
- Support decision makers in daily decision processes through assessment of “what if” scenarios.

On the other hand, DES simulations usually require the characterization of large volumes of input data, and therefore results are more sensitive to the experimental setup, and complicate the

realization of sensitivity analyses. More details on the advantages of DES models can be found in Section 3.1.

With all that in mind, more specific objectives of this Thesis are drawn next:

- In perpendicular semi-automated terminal, while introducing energy costs into the analysis, two particular problems are studied with the focus on the optimization of yard operations:
    - First, the so called *storage location assignment problem* in stacking inbound and outbound containers. An efficient stacking algorithm is introduced for deciding, in real time, the optimal position in the corresponding block for each arriving container in order to minimize the energy consumption of yard cranes and, at the same time, maximize cranes' productivity.
    - Secondly, we study the optimization of the block size (in length and width) while maintaining the height and total capacity.
- In parallel terminals, the influence of the time at which the space for outbound containers in the yard template is reserved in a container terminal. Space reservation can be made at any time with respect to the period during which outbound containers arrive to the terminal by truck (delivery period). Containers arriving prior to/after the reservation are allocated following an online/offline manner, respectively, inducing differences in the yard template in terms of the number and size of clusters of containers bound to each vessel.
- Transversally to the abovementioned objectives, one last objective is to assess the influence of traffic volumes –and therefore yard occupation- Numerical simulations are conducted on an import/export parallel terminal under two levels of yard occupancy in order to evaluate each strategy.

Chapter 3

Discrete Event Simulation Models for Terminal Yards

The present chapter describes the two main DES models elaborated from scratch to reproduce the parallel and perpendicular terminal layouts. The design of these models is one of the objectives of the Thesis, as they are intended to serve as a tool to help supporting the yard design process or optimizing operational procedures deployed in a terminal. This chapter is organized as follows: first, a general description of the DES models and their main characteristics is given in Section 3.1; Section 3.2 describes the characteristics of the perpendicular terminal model, and finally the parallel layout model is analyzed in Section 3.3.

3.1 DES models

Discrete-Event Simulation (DES) models reproduce the operation of a complex system as a sequence of individual events in time. The logic beneath the model is that, at each time step, an

event takes place and marks a *change of state* in the system (Figure 31). I.e., the arrival of a container to the land transfer area of an ASC block triggers the operation of the land ASC, causing the crane to move. Between two consecutive events, no changes in the system properties occur, and so the simulation can directly jump in time from the previous event to the next one.

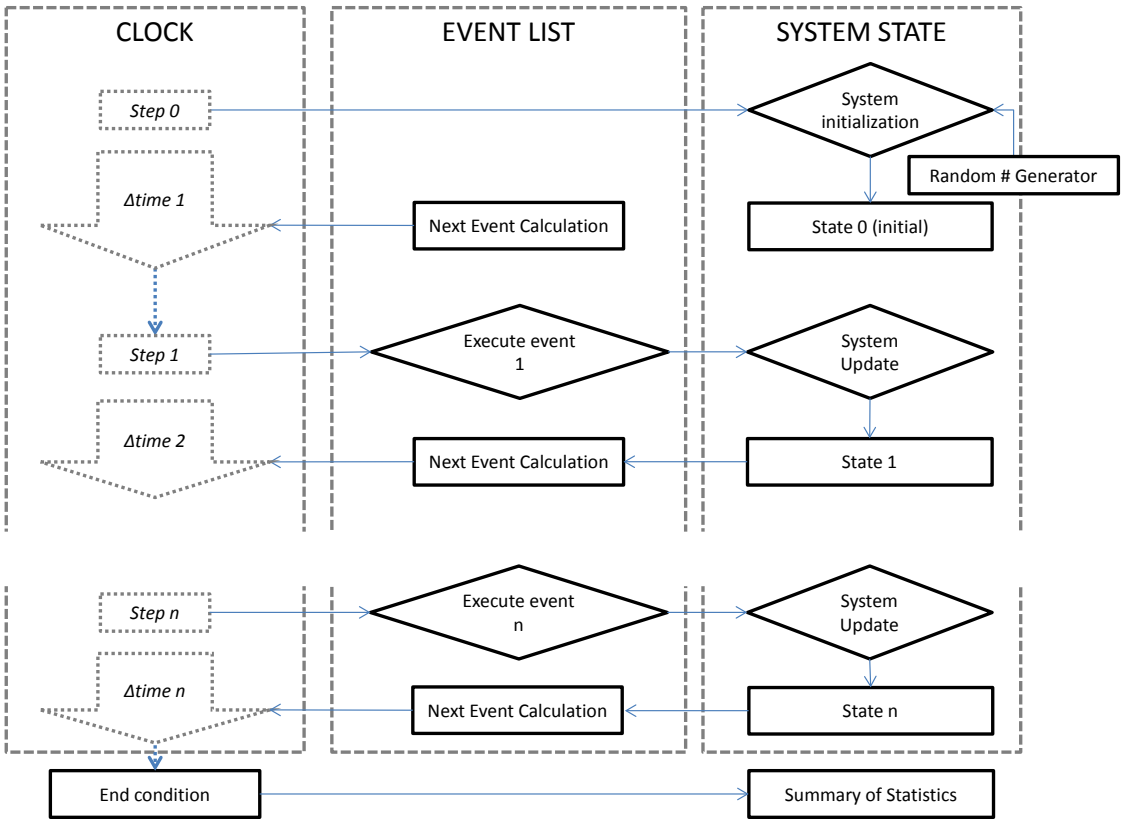


Figure 28. DES model logic.

In addition to the logic of the simultaneous actions when system events occur, DESs include a number of components of the two main DES models that are described next.

3.1.1 System state

The system state is the set of variables that gather information on the system properties. As this information changes over time, state trajectories can be acquired and reproduced by a discrete function whose values change in correspondence of discrete events. The most important system state variables are:

- Containers
    - Physical features (Flow type, weight, etc.)
    - Positions occupied along the block
    - Rehandles:
        - Suffered along the lifecycle
        - Height of the pile when the container is retrieved

        - Induced to other containers during retrieval processes
    - Events: arrival to Transfer Areas (TAs), picked by the crane, dropped in a block, rehandled, “housekept”, picked by the second crane, dropped in opposite TA.
    - Movements:
        - Type: productive and unproductive gantry, trolley, hoist/lower.
        - Duration of each movement
        - Energy associated to the realization of each movement by the crane
    - Dwell time
- ASC/RTG cranes:
    - Position occupied along the simulation
    - Status (idle, busy, waiting for another crane to perform a task)
    - Workload (container list, time, etc.)
    - Operation being executed: stacking, delivery, housekeeping.
- Block:
    - Buffers
    - Ground slots:
        - Number, type (import/export), height.
    - ASC dedicated areas, which coincide with the housekeeping bay limits (sea and land)
    - Bays
        - Containers IDs
        - Traffic type (import/export)
        - Occupation
- Berths:
    - Position
    - Occupation
- External Trucks:
    - Type: import, export, dual
    - Arrival time, exit time
- Vesssels:
    - Arrival and exit time
    - Number of containers of each type

3.1.2 Clock

Both models track the current simulation time, in units of seconds. At each time step, after the system is updated, the time left for each pending task is computed. The next task to be completed

is the one having the minimum pending time. Then again, the clock skips to the next event start time as the simulation proceeds by an amount of time different for each step.

3.1.3 Events list

Simulation models make use of a list of simulation events. The two models are *single-threaded*, with just one current event. Single threaded engines are simpler than multi-threaded engines because they avoid synchronization problems between multiple current events. Simultaneous events are avoided by artificially adding a small amount of time to the less urgent event of the list, which helps the system not to skip them when the current event takes place and the system is ready to advance to the next time step.

An event is described by the time at which it occurs and a code that will be used to identify and simulate that event. The list of main events is provided next:

- Arrival of vessels to the terminal: when a vessel arrives to the terminal, the vessel downloading process starts.
- Arrival of import/export/dual trucks: trucks arrive to the land transfer point to drop export containers or retrieve import containers. As soon as a container arrives to the TP, it is incorporated to the last position of the land ASC task list.
- Arrival of containers to the land/sea transfer areas: in parallel terminal, the containers are placed at the berth location, whereas for perpendicular terminals, the containers are dropped at the sea transfer point by straddle carriers at a given rate.

Sometimes events are pending, as they need previously simulated events to be completed before they can be simulated themselves. Pending events are listed next:

- Vessel loading process: upon completion of the discharging process, the system is ready to start delivering export containers to the sea TP for vessel loading.
- Crane movements: crane will start stacking or retrieving containers as soon as the task list has containers in it.

Events can be also classified as instantaneous or not instantaneous. Activities that extend over time are sometimes modeled as sequences of events, i.e. the crane movements when stacking a container. Sometimes, the time of an event is specified as an interval, giving the start time and the end time of each event. Crane translations due to interferences fall into this classification.

Typically, crane events are scheduled dynamically as the simulation advances. I.e., in the perpendicular terminal model, the event “land crane gantry” at time t would, if the sea crane has greater priority, include the creation of the subsequent event “translate” to occur at time t+s, where s is a number generated by the calculation of new crane movement.

3.1.4 Random-number generators

The simulation needs to generate random numbers in order to create model inputs with stochasticity. This is accomplished by Matlab's number generator. The use of a seed to generate pseudo-random numbers (as opposed to true random numbers) allows the user to create and replicate the same random number list, which is very helpful for debugging purposes, should a simulation need a rerun with exactly the same behavior.

3.1.5 Initialization

One of the difficulties that need to be dealt with discrete-event simulation is the system initialization. The objective is to obtain a steady-state situation that is representative of the real phenomenon to be simulated. In this case, models overcome this issue by allowing the execution to run enough time to reach the steady state. Starting from an empty terminal, new events schedule additional events of the simulation, and as the simulation progresses, their time distribution approaches the steady state, in which the terminal or block inventory size is stable.

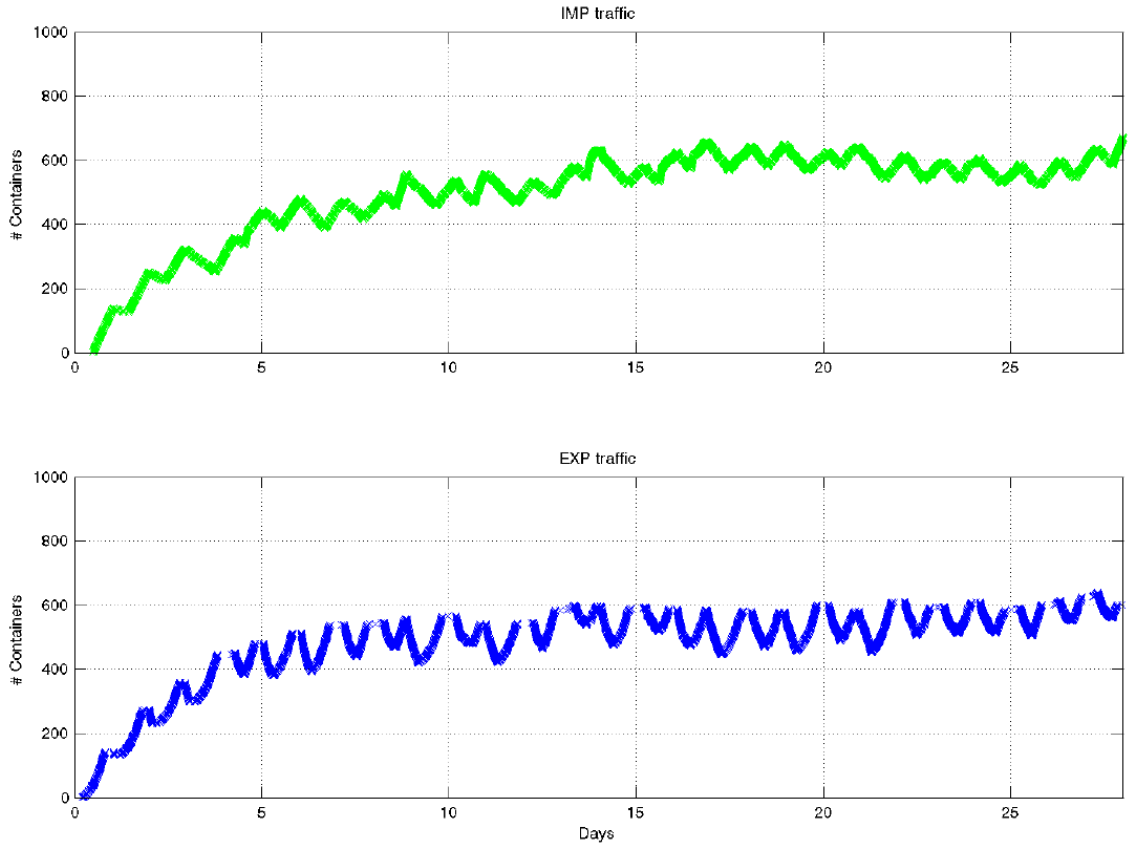


Figure 29. Example of the evolution of the inventory size during one simulation experiment.

In gathering results, events that occur before the steady state is reached are disregarded.

3.1.6 Statistics

The simulation keeps track of the system's state with respect to time. In a simulation model, performance metrics are obtained as averages over replications, that is, different runs of the model.

3.1.7 Ending condition

The ending condition determines the end of the simulation. In this Thesis, the choice in both models is “at time t”, where “t” is the desired duration of the simulation. The criterion to determine the value of t is described in subsequent sections.

3.2 Perpendicular layout model: ASC block model

The perpendicular layout DES model is a fully user configurable model designed reproduce an isolated yard block from a perpendicular terminal, operated by two non-crossable ASCs. Figure 33 illustrates an example of the yard block simulated by the model.

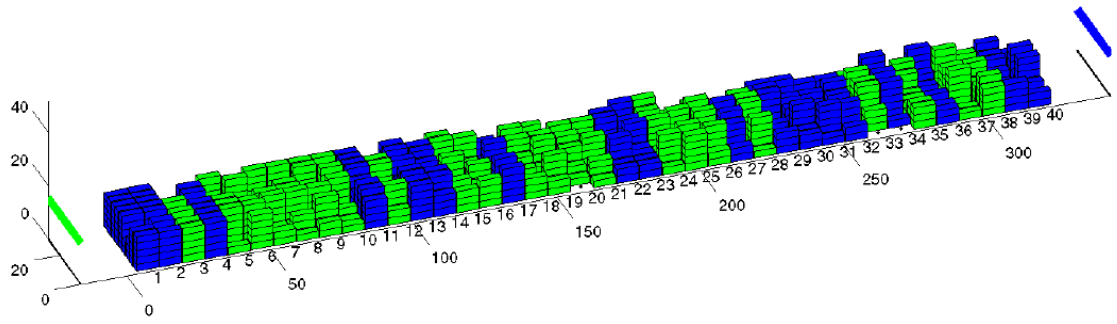


Figure 30. ASC block model with import (green) and export (blue) containers. Sea (green) and land (blue) cranes are depicted as horizontal bars.

The model does not include the simulation of other terminal processes occurring in the terminal such as the land gates system, transfer vehicle system or QC system (Figure 24), although they are indirectly considered and simplified in the model. As previously mentioned, this methodology simplifies the simulation without loss of generality as, under normal circumstances, it can be hypothesized that blocks can operate independently from one another. This approach is also found in Saanen et al. (2005), Speer et al. (2011) or Xin et al., (2015) and it is justified since slot assignment algorithms are applied within a storage block due to the assumption that the block is already chosen.

The DES model of the block is coded in Matlab. The program has the capability of reproducing in detail different simultaneous operations, including the complete lifecycle of containers being stacked or retrieved. As previously indicated, both import and export flows take place at the same time.

It is worth noticing that stochastic distributions are utilized to generate traffic inputs for the model. This way, random properties that characterize traffic can be reproduced in the model using statistical distributions and event generators. When doing so, for simplicity events in the model take values of entire seconds. Examples of stochastic generated variables in the model are external truck arrivals, vessel berthing events, or ASC operations already in progress such as the spreader positioning on top of the container by the crane operator.

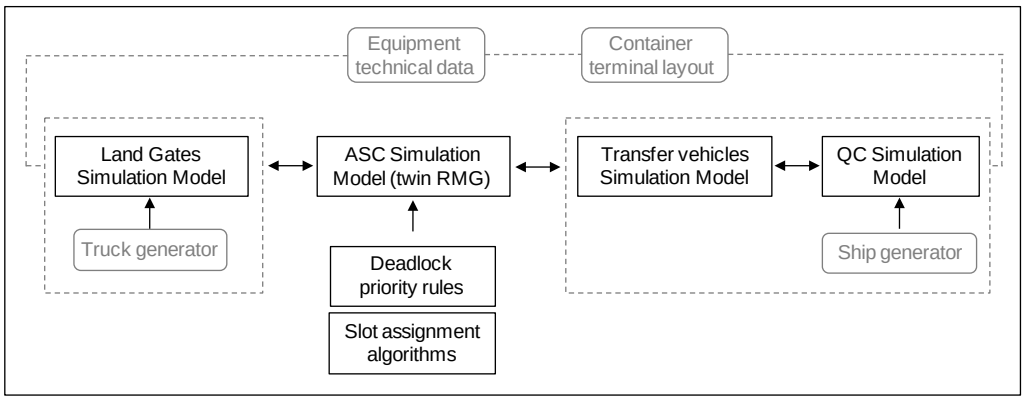


Figure 31.Semi automated container terminal simulation model scheme.

3.2.1 ASC block characteristics

The block is fully configurable, and can be modeled with standard 20feet or 40feet slots. The block total length, width, and height results from the user configurable variables, which are indicated next:

- Dimensions:
  - Length: number of bays, slot size for each bay (20feet or 40feet), and longitudinal spacing
  - Width: number of stacks per bay, and transversal spacing.
  - Height: number of tiers
- Minimum working spacing between two ASCs (by default, 2 TEUs or 20-foot containers)

3.2.2 Transfer areas or transfer points

Differentiated import and export two-container-high transfer areas are located at the sea end of the block to allow the interchange of containers between the straddle carriers and the ASC cranes, whereas a simple transfer area at the land side allows for the interchange between the land ASC and the ETs. The number of lanes per TA can be configured by the user.

3.2.3 ASC modelization

The ASCs are modeled as entities that move back and forth along the block, either “laden” (carrying a container) or “empty”. Upon container arrival to the transfer area, the ASC places the

container's ID on the last position of its workload. Once all previous tasks have been completed in the order of arrival (FIFO rule), the ASC crane calculates the stacking cycle for that container according to one of the stacking algorithms described in subsequent sections. Before taking action, the system calculates each individual movement of the cycle taking into account not only the speed but also the periods of acceleration and deceleration. The values of speed and acceleration used in the simulation are obtained from a commercial make of ASCs; with the speed being a function of the container weight. Also, the weights of the crane itself and its moving parts are obtained from real ASCs (see Section A.1.1).

Delivery operations are triggered either by the arrival of import trucks or the loading sequence of a vessel. If reshuffling movements are necessary to retrieve a container, the energy consumption is used as criterion to relocate the containers above within the same bay. Export containers to be loaded onto a vessel are stacked in uniform piles to minimize the secondary movements during the ship loading operation.

ASC user-defined characteristics are listed next:

- Crane movements
    - Speed at both fully laden condition and empty for gantry, trolley and hoist movements.
    - Acceleration at both fully laden condition and empty for gantry, trolley and hoist movements.
- Weights of the moving parts associated to each type of movement: total ASC weight for gantry, spreader weight for hoist, and spreader plus cabin weight for trolley.

Dimensions of the crane are automatically calculated taking into account the height and width of the block in which the ASC is located. As for the fine positioning of the spreader, a Poisson random generator is utilized with the values indicted in the next table:

Spreader Positioning	Poisson's λ Parameter (secs)	Standard deviation (secs)
Sea side TP	10.0	3.17
Land side TP	30.0	5.50
Yard block	7.0	2.66

Table 2. Values of the λ Parameter (in seconds) used to generate values for the fine positioning of the spreader.

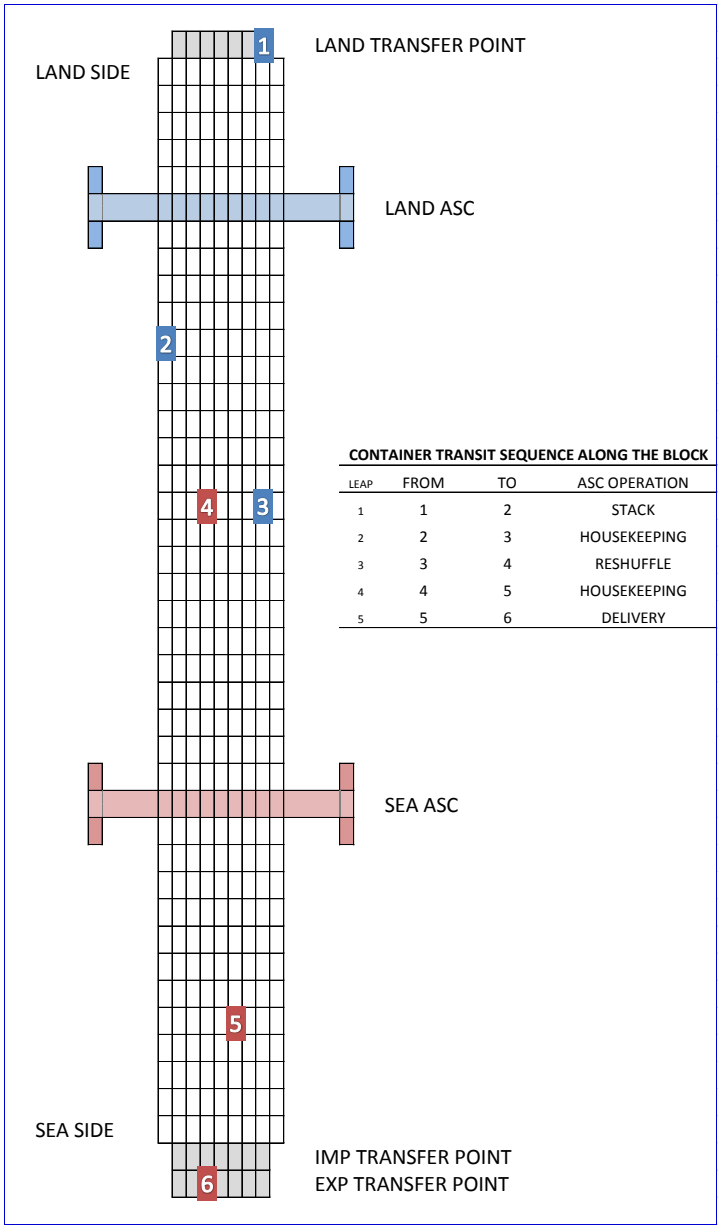


Figure 32. Positions occupied by an Export Container during its transit along the block.

3.2.4 Stacking algorithms

Different stacking algorithms are implemented to operate the ASCs. These algorithms are described next:

3.2.4.1 Logic stacking assignment (LSA)

The logic stacking assignment algorithm, currently used in a container terminal, is used to stack import and export containers by evaluating all the candidate slots according to several quality criteria.

The categorization of the candidate slots considers three different levels of quality. Thus, provided the ground slot under evaluation pertains to the desired import/export category (otherwise the slot

is directly disregarded), the height of the stack and bay occupancy are used to classify the slots into three quality levels: high, medium and low (H, M, and L, hereinafter). The thresholds of stack height and bay occupancy are not provided as this information is confidential. Candidate slots of greater quality (H > M > L) are preferred over those of lower quality. In case no ground slots qualify as H quality, the algorithm will search among those with quality M, and so on. Among candidates of the same quality, preference is given to the ground slots closer to the transfer point (landside for imports and waterside for exports).

3.2.4.2 Random stacking assignment (RSA)

In random stacking there is no preference for particular places, and it is used to spread containers evenly over the block. Basically, and as it is stated in Dekker (2006), new containers will be placed at a randomly chosen allocation in which each candidate location has an identical probability of being chosen.

3.2.4.3 Pseudo Random stacking assignment (PRSA)

A different version of the random stacking algorithm is also evaluated in which a previous classification of containers is made assigning them into piles according to category groups. Thus, upon container arrival, the PRSA gives preference to slots located on piles of the same category as the new container and then it is placed at a randomly selected location among the candidates. Piles of a different category are only considered when no piles of the same category are found. This algorithm has been also used in the real terminal to evaluate and compare the performance of the LSA.

3.2.5 Housekeeping algorithms

Housekeeping is a common practice in container terminals, and consists on the execution of additional ASC movements to improve the quality of the block piles. The goal is to relocate containers as close as possible to the transfer areas to enhance faster retrieval movements, especially for vessel loading. As these operations have no priority over regular stacking or retrieval of containers, they can only take place only when an ASC is idle. By default, two types of housekeeping movements are implemented in the model:

- Prior housekeeping movements: when a container is known to be retrieved in the near future (30 min for import containers and 24h for export containers).
- Non-prior housekeeping movements: for containers whose retrieval time is not known yet.

Import and export housekeeping movements can be indistinctly executed by both cranes. Hence, every time an ASC becomes idle, the system randomly searches into the import/export worklists taking into account the priority rules indicated above. A container in the list with priority is

randomly selected, and its new position in the block determined using the same stacking algorithm. However, a minimum leap of 6 slots is required by the model to consider worth the execution of a housekeeping movement. In addition, no housekeeping movements are allowed for containers belonging to the 8 bays located nearest to the final TP. When the terminal gates are open (7:00 to 21:00) priority is given to import operations, whereas export operations are left for the night shift. In addition, export containers bound to a vessel are also given priority if the vessel's arrival is due within the following 24 hours.

As with other procedures, housekeeping can be configured by the user too by simply modifying the input parameters:

- Stacking algorithm (random, random with container grouping, user defined)
- Minimum leap length of housekeeping movements
- Bay limits (areas in which containers cannot be relocated anymore)
- Priority time intervals for import and export containers.

3.2.6 Crane interference

As the two cranes move simultaneously to perform stack or retrieval operations, interference may take place at any location along the block. In order to avoid such scenarios, the complete ASC stacking or delivery operations are calculated before the actual crane movements take place to detect whether the crane trajectories intersect or approach in excess. A complex algorithm manages the different intersection scenarios based on heuristic rules and decides whether, for instance, an idle ASC simply recedes just enough to allow the other crane accessing the position in conflict.

However, when the two cranes have tasks to complete, priority is given to the sea ASC to provide the fastest possible service to the vessel, unless the land ASC has initiated a task before the sea ASC initiates the next.

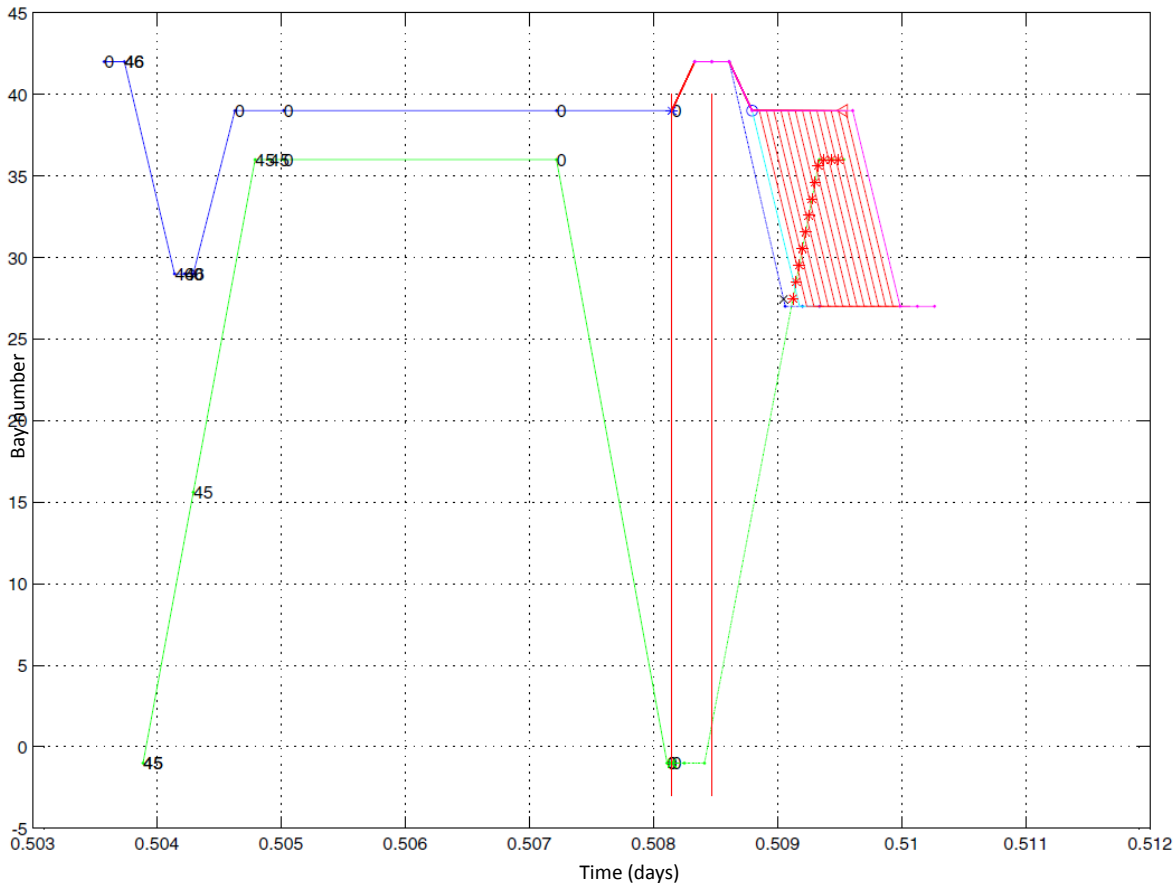


Figure 33. Example of the resolution of a conflict between ASCs. The trajectory of the land ASC (blue line) experiences a delay at bay 31 to allow the sea ASC (green line) to stack a container on bay 36.

3.2.7 Model assumptions and limitations

In this section, the most important model limitations are summarized. First of all, only one yard block is modelled. Hence, interdependencies between this block and other processes of the storage yard are neglected. Secondly, QCs and horizontal MHE are not explicitly modelled. As indicated in section 3.2, it is assumed that the processes taking place at the interfaces of these subsystems are deterministic, and that a sufficient number of transport equipment is always available, so that no waiting times for the ASCs are induced due to late arrivals of SCs. As a consequence, usual performance indicators like the Gross Crane Rate (GCR) cannot be obtained, and therefore productivity will be assessed by means of ASC throughput rates.

Only 20feet containers are utilized, disregarding 40feet units for simplicity. Containers of other sizes (45feet long, foldable, etc.) and boxes for special goods (refrigerated goods, liquids, dangerous goods, etc.), are normally accountable for a limited amount of the total container traffic volume.

3.3 Parallel terminal model

The parallel terminal model is a fully user configurable model designed to reproduce a yard terminal in the same fashion as that depicted in Figure 22. All blocks in the terminal are modeled as having the same dimensions, which are ultimately given by the number of standard 20feet slots in length, width, and height. As in Section 3.2, container spacing needs to be defined both in longitudinal and transversal directions. In addition, the parallel layout will need the width of the vertical and horizontal aisles, as well as the width of the operations area.

3.3.1 Transfer points

Transfer lanes of one standard 20 feet slot width are located at the sea side of the blocks to allow the interchange of containers between the ETs and the RTG cranes. This free lane is dedicated to the ETs to carry out the maneuvering and wait for the crane to interchange the containers, as depicted in Figure 34.

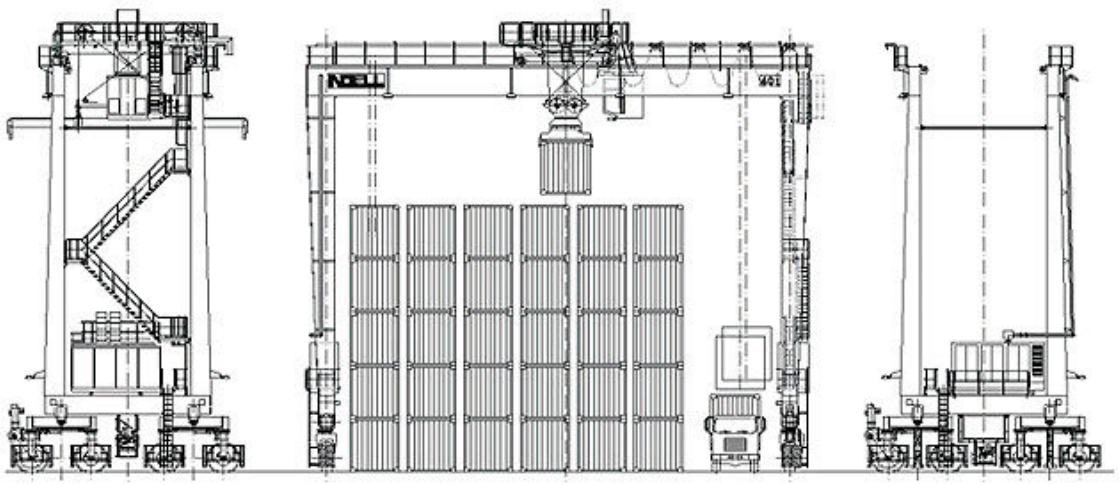


Figure 34. Example of a RTG in a 6 stack and 5+1 tiers block.

3.3.2 Operational strategies

Following the two stage approach for allocating containers in the yard, several different operational strategies can be selected by the user in order to solve the block allocation problem and the slot allocation problem, as indicated next.

3.3.2.1 Block allocation problem

As previously indicated, the block allocation calculates the amount of space needed in the yard, as well as the exact stacking positions to be occupied by the containers. The strategy can be configured to select the time at which space for a vessel is reserved with respect to the moment containers start arriving to the terminal. The main criterion to allocate containers is the minimization of the distance to the berth, enforcing the Nearest Location principle. By default,

the import allocation is configured to take place upon vessel arrival to the port, and so the algorithm also takes into account the YC workload as second criterion to evaluate candidate stacks in the yard. The user can also configure the weight given to each criterion, which by default is set equal to 0.5.

3.3.2.2 Slot allocation problem

Since the BAP consigns entire sub-blocks to containers that will be allocated in the future, the system will need to evaluate all the candidate stacks so as to determine the best position to allocate an incoming container. The way the model handles this problem is by adopting a strategic decision: the system shall stack containers in a way that each bays in the sub-block shall match the so-called *ideal bay configuration*, which is different for import and export containers.

For import containers, the user may choose whether departure information is known or not. If the answer is yes, the system will prevent new containers to be stacked on top of older containers. However, if no departure information is available, the system will evaluate the quality of the stacks taking into consideration the amount of time the containers in that pile have spent in the yard. In addition, as import containers are downloaded sequentially from the vessel in a limited amount of time, the YC workload is taken into account to evaluate the quality of each pile. The system will then estimate what YC is more likely to serve a stack based on the current position. Finally, for each candidate stack, both criteria are non-dimensionalized and summoned according to equation (Equation 22. The best scoring candidate according to that equation will receive the container.

$$S_i = q^{WL} \cdot \frac{WL_i}{\max(WL)} + q^{QD} \cdot \frac{WD_i}{\max(WD)} \qquad \textbf{(Equation 1)}$$

Where:

- i = subindex denoting the candidate stack, 1 < i < n, with n = B x S
- B = number of bays
- S = number of stacks
- S_i = score assigned to each candidate stack
- WL = workload of the crane assigned to that stack
- WD = weight class difference between the container and the candidate stack
- q^{WL}, q^{WD} = weights assigned to each criteria, satisfying the following condition:

$$q^{WL} \cdot + q^{QD} \cdot = 1 \qquad \textbf{(Equation 2)}$$

For export containers, weight information is used by the system to determine in real time the weight difference between the available stacks of each candidate bay. Among the stacks with a

smallest weight difference, the system will choose the candidate whose location is closest to the vessel berth. Weight differences are computed utilizing a simple *ideal bay configuration* (see Figure 3). In this bay, since nine diagonals can be drawn in a 6x4 matrix, containers are classified from 1 to 9 according to their weight class. Ideally, the system will try to stack an export containers in a bay slot that matches perfectly its weight class; when not possible, the position that minimizes the absolute difference of weight classes is sought. This diagonal arrangement has several advantages (Chen and Langevin, 2009): it reduces the handling effort associated with unproductive movements during ship loading sequence, because containers can be retrieved sequentially from the heaviest to the lightest, and containers on the left are retrieved earlier, which eases future YC retrieval operations of containers placed on the right. Additionally, less energy expenditure associated to heavier containers is needed with respect other ideal arrangements, as more stacking positions are available at the time of stacking. In the end, heavier containers tend to be close to their ideal position.

9	8	7	6	5	4
8	7	6	5	4	3
7	6	5	4	3	2
6	5	4	3	2	1

Figure 35. Ideal bay configuration. Container weight classification goes from 1 (lightest) to 9 (heaviest).

3.3.2.3 Retrieval procedures

The retrieval of import and export containers differs in many ways. In the first case, ET arrive to the terminal and request an import container, and the order is transferred to a YC. The specific YC is selected according to its proximity to the stack, and also the YCs workload. In this case, it is assumed the arrival time of ETs is not known, and so rehandling is a common practice when the container position is not at the top of the pile. As for export containers, the retrieval process follows the loading plan, and so containers in a bay are retrieved in order of weight, with heavier containers being retrieved first.

3.3.3 Terminal equipment

3.3.3.1 RTG modelization

Parallel terminals usually deploy several YCs per row of blocks. In this case, the model can be user-configured so as to select the desired number of RTGs. As before, RTGs are modeled as entities that move back and forth along the block, either “laden” (carrying a container) or “empty”. The YC stack and retrieval operations are calculated similarly to the perpendicular DES model described in section 4.2. When an ET or IT arrive to the side transfer lane of a block to either delivery of retrieve a container, the RTG calculates the stacking or retrieval cycle for the container

according to one of the algorithms described in subsequent sections. The model calculates each individual movement of the cycle taking into account not only the speed but also their periods of acceleration and deceleration. The values of speed and acceleration used in the simulation are obtained from a commercial make of RTGs. Again, the value of the speed is a function of the container weight. The weight of the crane and its moving parts are specified by the user.

As for the retrieval of containers from the blocks, whenever rehandling is required the system will seek a solution always within the same bay, for which enough space must be left empty in the bays to allow these operations. The system will calculate the energy consumption and the time needed for each container being relocated as criteria to select among the best candidate stacks, and repeat this process as many times as needed until the desired container can be finally delivered. Export containers to be loaded onto a vessel are stacked in uniform piles to minimize the secondary movements during the ship loading operation.

3.3.3.2 RTG control

For simplicity, in this case the movement of YCs is not simulated as discrete events. Instead, movements are directly executed as soon as an ET arrives to the transfer lane. This approach is different from the parallel terminal DES model, but can be also found in other studies of parallel terminals such as Dekker et al. (2006) or Lee and Kim (2013). As YC events are driven by the continuous arrival of trucks to the block transfer lanes, the system assumes that one YC will be immediately stack or retrieve interchange a containers between the block and the truck. The first consequence is that tasks are handled by YCs on a FIFO (First In, First Out) basis. More sophisticated policies than the FIFO rule are implemented in practice to optimize travel times, which may even lead to different distribution of containers in the yard when YCs position is considered in the allocation. In this case, the order of the tasks in the YC workload cannot be managed with respect to time, hence the operational costs deriving from YC movements along the blocks may be considered as an upper bound limit of YC performance under the proposed YC management policies. However, since no maximization of the YC productivity is attempted, the workload split among the YCs and their performance does not constitute a constraint to the system.

The second consequence is that no control is implemented in the model to handle YC interference. In general, interference among YCs highly depends on the efficiency of the operating algorithm, but in the parallel layout interference is not considered critical, as opposed to that in the perpendicular layout. As this model aims at supporting the yard design process and the quality of the yard template as a function of the container clustering, interference among YCs is assumed to be negligible.

3.3.3.3 YC performance

As a consequence of the lack of YC control, it is not possible for the model to assess YC waiting time or throughput rates, and so the YC quality of service cannot be measured directly.

However, the model is capable of measuring the size of the workload of each YC at any given time as the number of jobs assigned to any YC in the last hour. If the workload surpasses 25 movements per hour (mph), which is a usual value in the industry, the system will consider that YC overloaded until the number of assignments fall below the threshold. As crane overload will produce waiting times for the pending assignments, it can be considered as an indirect measure of the quality of service. A similar approach is followed by Dekker et al. (2006), who determined ASC workloads every quarter of an hour as the proportion of time the ASC are busy. Their simulation program allows workloads to exceed the capacity, i.e., workloads of more than 900 s per quarter. Likewise, the purpose of the parallel model setup is the evaluation of the stacking algorithm, and not on ASC scheduling; hence, these overloads are allowed and considered as one of the criteria to indirectly measure the performance of the algorithm.

Provided the number of YCs is sufficient to handle the yard operations, the system will produce no overloads if container clusters are well distributed over the yard, whereas a poor distribution will result in unequal distribution of tasks among the YCs and consequently, overloads. Whenever overload occurs, the terminal is considered to be capable of assuming the delays resulting from YC overload.

The model will also evaluate YC performance by calculating the electric consumption associated to crane movements. To this matter, the amount of gantry travel will be determined by the distribution of container clusters, which is given by the reservation strategy adopted to solve the BAP. As for the operational cost associated to hoisting, it will be determined by the strategy adopted to solve the slot allocation problem, as indicated in Section 4.3.3.2.

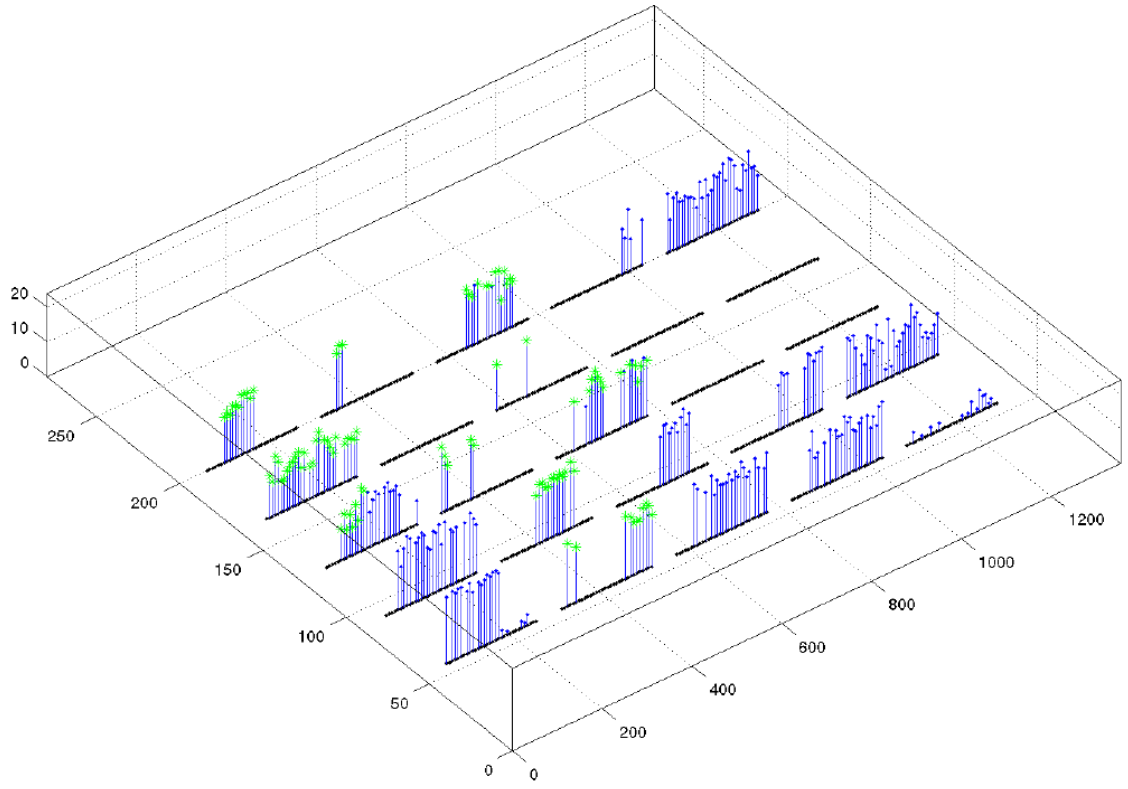


Figure 36. Example of the distribution of containers in the yard layout. Bays allocated to one vessel are highlighted in green. The height of the stem represents the bay occupation.

In any case, in addition to gantry travel, the model will measure the quality of the yard distribution be directly by accounting the number of sub-blocks and their size (measured as the number of bays and containers per sub-block).

In addition, the following assumptions are made regarding the YCs:

1) YCs are not explicitly dedicated to one QC nor a vessel or flow (imp/exp), allowing the provision of service to operations of different kind if that fits better the terminal needs. Likely, terminal blocks are not dedicated to a single (import or export) type of traffic.
2) As a consequence of the previous, crane overload is measured as the number of containers assigned to the YCs with respect to a time window of one hour duration. When the workload is greater than 25 mph, the YC is considered to surpass its theoretical throughput, and thus the crane is said to be overloaded for as long as the arrival rate of new containers assigned to that crane does not make that value fall below the threshold.
3) The evaluation of the rehandling effort under this approach is also possible by quantifying the number of containers piled on top of the container being delivered. Reshuffles are done within the same bay relocating the containers back on the original stack.

3.3.3.4 Yard Tractors

Yard Tractors are not simulated in detail as they are not considered the bottleneck of the yard system. Borgman et al. (2010) and many other studies utilize a similar approach considering that one YT is always available every time a container needs to be transported from the quay to the yard and vice versa. This simplification eases the simulation computational effort and has no influence on the evaluation of the way containers are distributed among the blocks.

YTs and also ETs travel through vertical and horizontal aisles to transport containers in single cycle mode, carrying a CT and back empty. YTs are allowed to circulate freely among the blocks in a clockwise direction, whereas External Trucks travel counterclockwise, and their vertical displacements are confined to the left and rightmost aisles of the terminal. This scheme results in unidirectional flows in horizontal aisles and bidirectional flows in vertical aisles, which are the most popular in real-world container terminals. No traffic interferences among the yard equipment are simulated.

Similar to the YCs, the performance of the reservation algorithms is measured for the YTs in terms of the operational costs, which are considered as proportional to the sum of the distance travelled between the berths and the bay for all the vessel loading/unloading operations.

3.3.3.5 Quay cranes

In order to provide efficient service to the vessel, quay cranes are assumed to work without downtimes during the vessel loading and unloading operations, delivering 25 containers/hour. Along with the arrival of pattern of containers to the sea side, this value is used to characterize the rates at which SCs arrive to the sea TP to interchange containers with the YC.

3.3.4 Model assumptions and limitations

In this section, the most important model limitations are summarized. First of all, YCs are not discrete event simulated. This assumption is based on the hypothesis that crane interference is small. The second consequence is that no control is implemented in the model to handle YC interference.

As in the Perpendicular Terminal Model, QCs are not explicitly modelled here. It is assumed that the pace at which QCs deliver and pick up containers are deterministic, and also that a sufficient number of YTs is always available, so that no late arrivals can induce waiting times for the YCs. As a consequence, the performance indicators of productivity used in the analysis will be the Vessel Service Time, the Container Exit Time, and the YCs throughput rate.

As before, only 20feet containers are utilized, disregarding 40feet units for simplicity. Containers of other sizes (45feet long, foldable, etc.) and boxes for special goods (refrigerated goods, liquids, dangerous goods, etc.), are normally accountable for a limited amount of the total container traffic volume.

Chapter 4

An Efficient Stacking Strategy for Perpendicular Terminals

4.1 Introduction

Modern container terminals rely on the so-called *Terminal Operating Systems* to manage and monitor activities within the terminal premises in an automatic way. These systems provide a set of computerized procedures to help the MHE managing the cargo. In this context, slot allocation strategies are coded in algorithms and then built into TOSs to calculate the optimal position of incoming containers in the yard and increase storage productivity and density. As container traffic keeps growing and yard occupancy increases, some negative effects on the YC performance arise. Higher stacks increase the incidence of rehandling movements, hence decreasing the productivity (operating costs) and efficiency (delays) of the handling process.

The nature of the location assignment problem is both combinatorial and dynamic. Decisions must consider the procedures to empty the stacks or place reshuffled containers, combined with the original placement operation. As is shown in the literature, storage location assignments differ from import and export flows and for conventional and automated container terminals; therefore several types of stacking strategies are observed.

Regarding the type of flow, export containers are usually stacked according to groups specified in the outline of the vessel loading plan. From that moment on, incoming containers are stacked in the block in piles of uniform category. The final loading plan will indicate the specific sequence of containers to be loaded in the ship and so rehandling may occur when containers listed onward in the loading plan are piled higher than the container being retrieved. For import containers, truck

arrival time and thus container departure time are unknown at the moment of stacking, hence rehandling is more frequent than for export containers.

From an energy cost perspective, unproductive movements during retrieval of import containers and inefficient handling of heavy containers are some of the most significant causes of operational inefficiency, which leads to dramatic cost increases and higher engine emissions from yard cranes.

In such context, stacking

The remainder of this chapter is organized as follows: section 4.2 analyzes previous storage strategies. Section 4.3 introduces a new strategy and its operating procedure. Next, Section 4.4 presents the DES model used to simulate the operations in a perpendicular block. Results and discussion is provided in Section 4.5, while Section 4.6 provides conclusions of the work and guidelines for future research.

4.2 Overview

Research topics on operations efficiency by means of optimization methods and operations models, such as storage and stacking logistics, has been a major trend in the last years. The problem approach usually varies depending on the type of flow (import/export) and also the type of terminal (parallel/perpendicular). In addition, some researchers consider the allocation problem for one bay, whereas a more general approach gives considers allocation of each container in a whole subset of blocks previously reserved as a solution to the BAP.

For outbound containers, Kang et al. (2006) considered the problem of export containers with uncertain weight information, and solved the allocation problem in a single bay by applying a simulated annealing algorithm for finding a good stacking strategy. When dealing with the allocation in a subset of bays, most of the literature focuses on transshipment terminal, in which the arrival time of export containers is known and thus a solution can be found for each of the planning periods. I.e., Wan et al. (2009) also studied the allocation problem in a bay for export containers, but their approach can also handle the location problem for blocks. Following the same research line, Park et al. (2011) proposed an online search algorithm which dynamically adjusts the stacking policy represented as a vector of weight values for automated container terminals. They support the fact that online search is a good option in dynamic settings where there is not enough time for computation before taking actions.

For inbound containers, the study of rehandling problem for import containers, was firstly analyzed by Castilho and Daganzo (1993). They developed a method for estimating the expected number of moves required to remove a single container from a bay, and also extend the formula for estimating the expected number of moves needed to retrieve several containers from a group of bays. For the second, two different realistic operating strategies are used to stack import

containers: non-segregation and segregation strategies. Under the second strategy, containers from different ships are separated in the stacks. Kim (1997) also proposed an exact procedure and a regression analysis to calculate the expected number of unproductive moves to retrieve a container from a bay in a random way. Later, Kim and Kim (1999) analyzed the effect of constant, cyclic, and dynamic arrival rates of import containers to a block. A segregation strategy is used to find the optimum stacking height that minimizes rehandling.

With respect to automated container terminals, several studies investigate segregation strategies. Duinkerken (2001) Dekker et al. (2006) and Borgman (2010) evaluated different stacking strategies in the EDT Terminal. They conclude that, if container departure time is known at the time of stacking, segregation strategies are more efficient.

With respect to the particular topic of energy consumption, as indicated in Section 2.5, little work is found in the literature, none of which is related to the slot allocation problem. Hussein et al., (2012) consider the container weight to minimize fuel consumption in order to solve the so called block relocation problem for container retrieval within a bay. Xin et al. (2013, 2013b and 2015) investigate the energy expenditure in a simplified way, as a first step to introduce this issue in the operational research literature.

In contrast, the objective of this work is to develop a stacking algorithm to improve the energy efficiency and the productivity of yard cranes, for which a detailed energy consumption model is introduced to fill the gap existing in the literature. The previous studies just consider as an objective the minimization of the number of rehandlings but none of them consider the energy efficiency of handling processes.

In addition, literature usually pays little attention to the performance of stacking algorithms with respect to block occupation. In the long run, yard occupancy is determined by the inflow and outflow of containers, which in turn are forced by the inter-arrival patterns of containerships and external trucks and the container dwell times for import and export trucks, respectively. The greater the number of containers in a block, the more restricted the container stacking operation, and the more inefficiencies (i.e. re-handles) expected in the retrieval process. Thus, a secondary objective of this Thesis is to evaluate the role of block occupancy in the ASCs performance, and to assess the sensibility of the proposed stacking algorithm to the yard occupancy with regards the benchmarking algorithm.

4.3 An Efficient Storage Stacking Algorithm

This section describes an Efficient Storage Stacking Algorithm (ESSA) to minimize energy consumption and unproductive movements of all those stacking processes which take place in the storage yard when an import container arrives at the block yard.

Let's assume an export container has arrived to the TP and the land ASC is ready to stack it in the block. The next step is to identify all the candidate slot positions to receive the incoming container when the stacking crane is ready to initiate this task. The identification of candidate stacks requires the searching algorithm to disregard stacks or even bays based on a number of safety and operating reasons. The following restrictions, which commonly used in the industry, will be taken into account:

- Containers of different size, weight category and/or type of cargo cannot be mixed together in the same stack (often referred to as container *grouping*).
- Refrigerated containers must be placed together in a reserved area and dangerous goods must follow safety rules in the storage yard.
- Import and export containers are not stacked on the same stacks/bays to prevent incurrence of rehandling jobs.
- Bay capacity and stacking height is limited to allow reshuffles to take place within the same bay.

In addition to the above-mentioned criteria, the candidate slot selection procedure will also take into account the stack and bay occupation, and pile composition, as illustrated in Figure 39.

Once all the candidate slots are identified, the ESSA algorithm proceeds to evaluate the slot quality according to the objectives of the ESSA algorithm. The scoring process requires evaluating the pick-up and drop-off processes for any incoming container in all the candidate slot positions available at the job starting time, as described next.

4.3.1 Efficient stacking algorithm sequence

The ASC duty cycle can be divided into two main different processes or operations: stacking and retrieval. In order to estimate the energy consumption and the time needed for a container to complete its transit along the block, the ESSA takes into account both stacking and retrieval operations when selecting the candidate slot. This approach is innovative compared to other scheduling algorithms found in the literature, as they in general they focus on the minimization of stacking times only.

Thus, at the time of stacking a new container, the ESSA evaluates each candidate slot by characterizing the energy consumption and the time needed to complete the transit of that container along the block. As previously indicated, a container typically experiences one stacking operation, one retrieval, and a different number of intermediate housekeeping movements, and each of this movements are characterized in a different way. The stacking algorithm procedure executed after each container arrival is indicated in section 4.3.3.

4.3.1.1 Stacking

When an ASC is about to start the stacking operation of a new container, all the information needed by the system is available (ASCs position and workload, block layout, etc.) or can be readily calculated (such as crane interference). As a consequence, the ESSA can accurately characterize the energy consumption for both cranes and duration of the stacking operation for each candidate pile to receive the container.

4.3.1.2 Retrieval

Contrarily, the retrieval process will take place in the future; since the arrival and departure of containers change the configuration of the block continuously, retrieval cannot be defined accurately at the stacking time. As explained next, hypothetical housekeeping operations are not considered in the analysis for simplicity, and therefore the ESSA assumes that any container will be retrieved from the position where the container was originally stacked in the block. As a consequence, the ESSA algorithm calculates the movement of the ASCs considering the probability of the cranes to be located at any particular bay of the block.

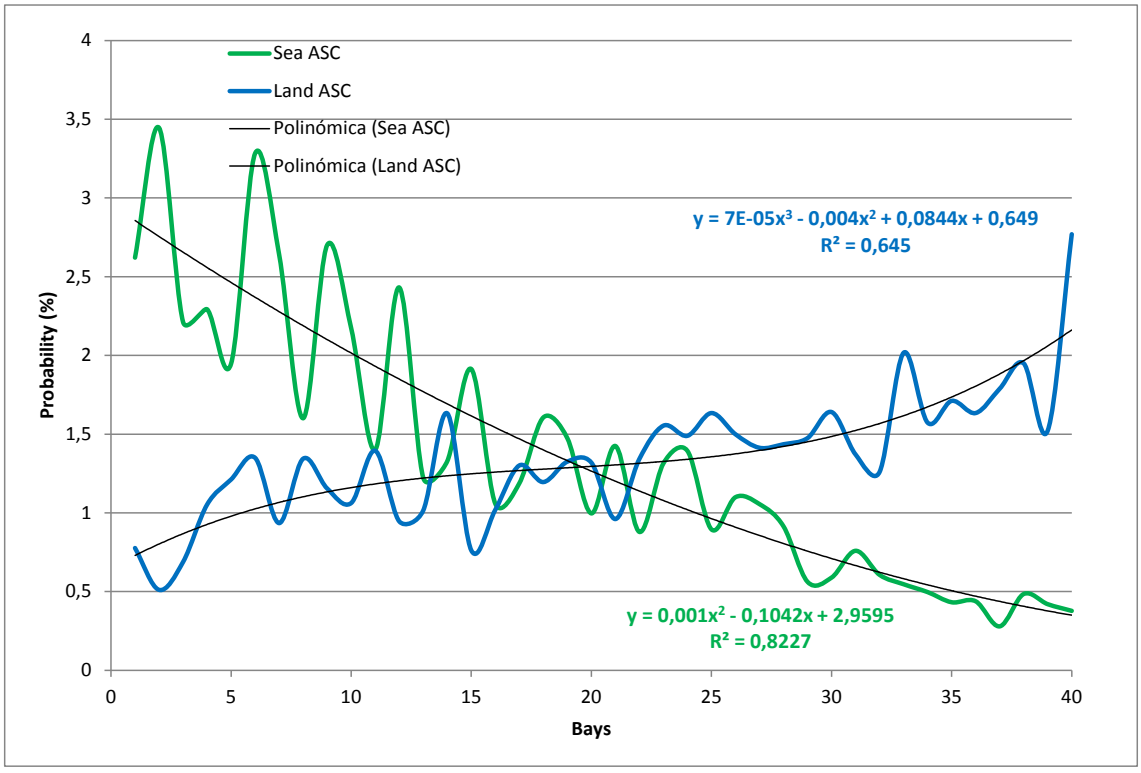


Figure 37. Probability of finding the sea (green) and land (blue) ASCs in the block bays.

The probability distribution of the ASCs is depicted in Figure 37. Both probability distributions fit well to polynomial curves, which are used to calculate the retrieval times and energy consumption at the time of stacking, including the possibility of interference. As is readily apparent, the location of import and export containers can be inferred by the peaks and troughs of

the curves at complementary bays. The fact these peaks exist indicates that bays don't easily change from one type of traffic (import to export) to another. As a consequence, results may show a great degree of dependence on the initial block setup, and so several initial states must be simulated to eliminate that effect from the final results.

4.3.1.3 Housekeeping

Finally, with respect to housekeeping, no consideration is given to this type of operation, as the complexity of estimating these operations at stacking time are paramount. The estimation of housekeeping movements, provided they could be useful to improve the performance of the ESSA, is left for future research. However, a methodology is suggested in Appendix B that may be used as guideline for the development of such procedure.

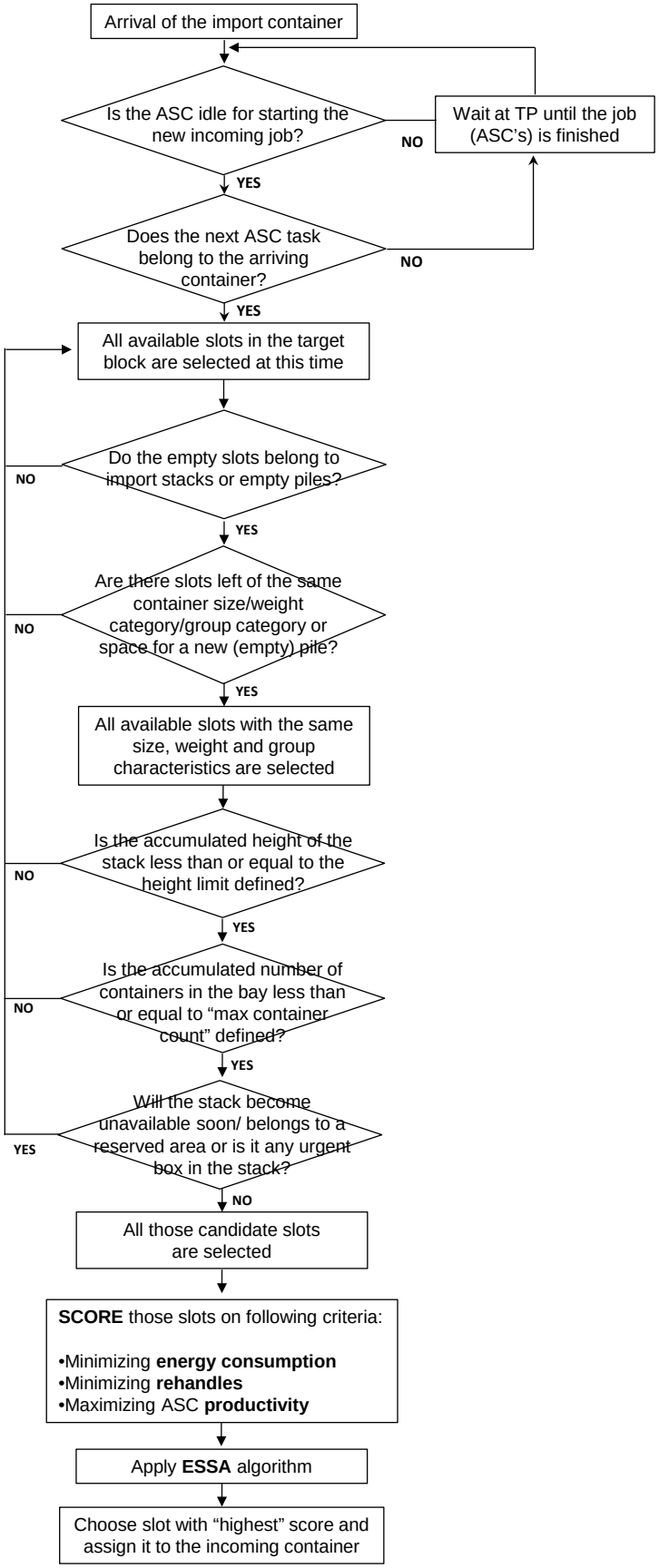


Figure 38. Selection process and scoring method through ESSA algorithm to determinate the target slot (slot assignment).

Contrarily, housekeeping and retrieval processes take place much further in time, and cannot be accurately defined at stacking time since the block yard will continue to change due to the continuous arrival and retrieval of containers.

4.3.2 Scoring formula

In this section the scoring formula which allows choosing the best candidate location to stack the incoming container is introduced. The main criteria to choose the target slot is, on one hand, minimizing energy consumption and, on the other hand, maximizing ASC productivity. Then, each candidate slot ($Q_j, j \in \{1, n\}$) will be scored according to the following expression:

$$\boldsymbol{Q_j(e,\rho) = \sum_{z} w^z q_j^z = w^e q_j^e + w^\rho q_j^\rho} \qquad \boldsymbol{\sum_{z} w^z = 1} \qquad \textbf{(Equation 3)}$$

where:

$$\boldsymbol{q_j^e = \frac{Min\left[(e_j)\right]}{(e_j)}} \qquad \textbf{(Equation 4)}$$

$$\boldsymbol{q_j^\rho = \frac{Min\left[(t_j^{-1})\right]}{(t_j^{-1})}} \qquad \textbf{(Equation 5)}$$

In this case, e_j (energy consumption) is calculated according to the Potential Energy Consumption Model described in Section 3.5 with respect to t_j (ASC productivity) it will be calculated by implementing **Equation 7** and **Equation 8**. The variable w^z is the weight associated to each criteria, that is energy consumption (z=e) or ASC productivity (z=ρ).

Finally, the selected slot (j^*) will be the one satisfying the next expression:

$$\boldsymbol{j^* = argmin_j\{Q_j(e,\rho)\}} \qquad \textbf{(Equation 6)}$$

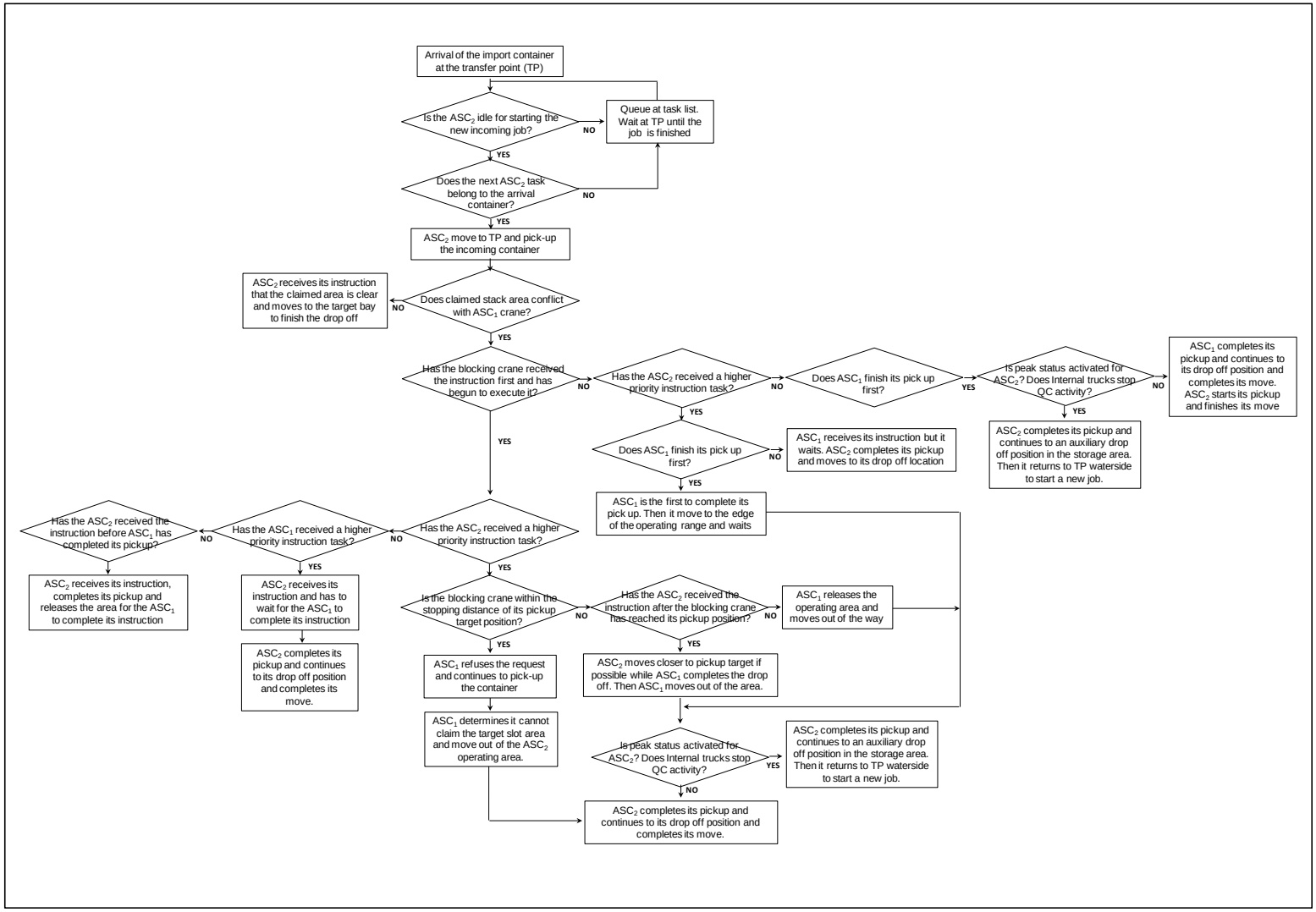


Figure 39. Storing/stacking operational framework of ASCs at the block yard

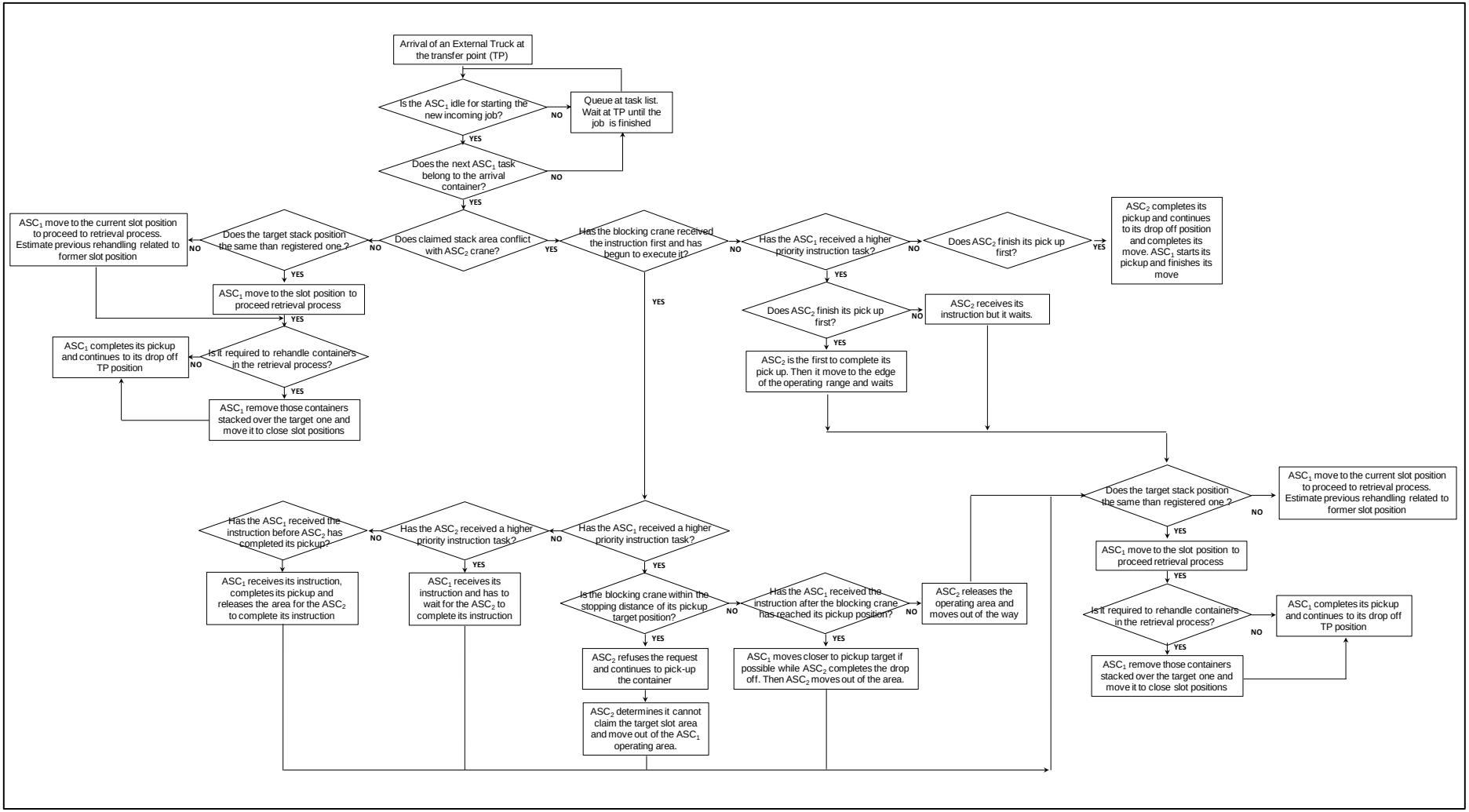


Figure 40. Retrieving operational framework of ASCs at the block yard

The following **notations** are used in describing the stacking algorithm procedure:

i: ASC sub index in the block yard (ASC tag)

j: Candidate slots sub index ($j \in \{1, n\}$, n is the total number of candidates). Result from the selection process of Figure 38 (Slot tag).

$[k]$: Indicates the type of procedure which is being proceed: pick up ([a], [c]), drop-off ([b], [d]), rehandling move ([r]) or crane conflict move and repositioning ([cc])

$\Delta x_{[k_j]_{ASC_i}}$: Gantry crane distance alongside axis "x" run by ASC_i during procedure [k] and for candidate slot j.

$\Delta y_{[k_j]_{ASC_i}}$: Spreader distance alongside axis "y" run by ASC_i during procedure [k] and for candidate slot j.

$\Delta z_{[k_j]_{ASC_i}}$: Hoisting/lowering distance alongside axis "z" run by ASC_i during procedure [k] and for candidate slot j.

Z_{Wh}: ASC working height (m).

$(X^0_{ASC_i}, Y^0_{ASC_i}, Z^0_{ASC_i})$: Initial coordinate position of ASC_i (i=1,2) before starting pickup process during storing/stacking procedure (at time t_0).

$(X^1_{ASC_i}, Y^1_{ASC_i}, Z^1_{ASC_i})$: Initial coordinate position of ASC_i (i=1,2) before starting pickup process during repositioning procedure or crane conflict solving movement (at time t_1).

$(X^2_{ASC_i}, Y^2_{ASC_i}, Z^2_{ASC_i})$: Initial coordinate position of ASC_i, ($i \in \{1,2\}$) before starting pickup process during retrieval procedure (at time t_2).

$(X_{cs_j}, Y_{cs_j}, Z_{cs_j})$: Coordinates of the candidate slot j, $j \in \{1, n\}$.

(X_{as}, Y_{as}, Z_{as}): Coordinates of the auxiliary slot used for repositioning

$(X_{rs_j}, Y_{rs_j}, Z_{rs_j})$: Coordinates of the slot which receives a rehandled container from slot j, $j \in \{1, n\}$.

$(X_{WTP}, Y_{WTP}, Z_{LTP})$: Coordinates of the waterside transfer point

$(X_{LTP}, Y_{LTP}, Z_{LTP})$: Coordinates of the landside transfer point.

To sum up, Figure 41 shows the different processes included in the evaluation process of each candidate slot are depicted schematically.

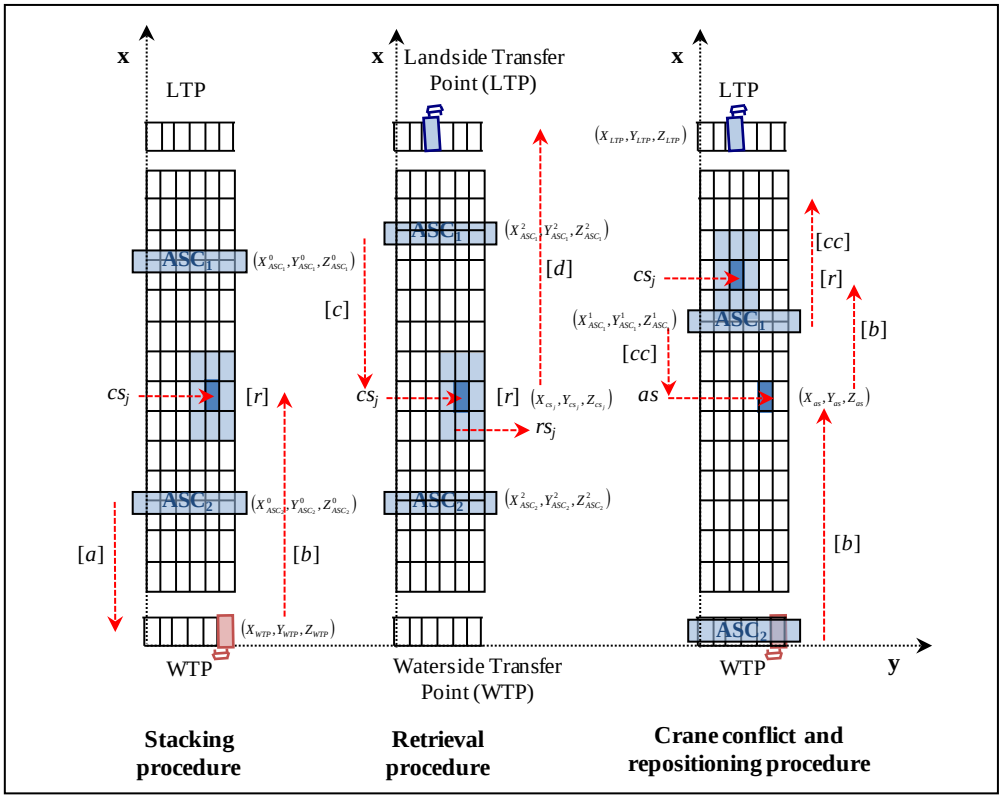


Figure 41. Pickup and drop off schematic procedures in the block yard. Crane conflict movements and reposition procedures.

4.3.3 Algorithm sequencing for ASC

The algorithm sequencing is given by the following steps:

Step 1: Select the total number of candidate slots (n) from selection procedure (Figure 38).

Step 2: For each candidate slot assign a sub-index $j, j \in \{1, n\}$ (onwards from waterside to landside and from left to right side) and determine its location coordinates $(X_{cs_j}, Y_{cs_j}, Z_{cs_j})$.

Step 3: Determine the waterside transfer point position where new incoming container is placed $(X_{WTP}, Y_{WTP}, Z_{LTP})$.

Step 4: Update the ASC_i tasks list and determine the coordinate position$(X^0_{ASC_i}, Y^0_{ASC_i}, Z^0_{ASC_i})$ at that time (t_0) just before starting new incoming job (an import container is being picked up from the waterside transfer point).

Step 5: Calculate the rectilinear distance from WTP to initial position of ASC_2 (pick up procedure $[a]$) as:

$$\Delta x_{[a_j]_{ASC_2}} = \left|X^0_{ASC_2} - X_{WTP}\right| \quad \Delta y_{[a_j]_{ASC_2}} = \left|Y^0_{ASC_2} - Y_{WTP}\right| \quad \Delta z_{[a_j]_{ASC_2}} = \left|Z_{Wh} - Z_{WTP}\right| \qquad \textbf{(Equation 7)}$$

It should be notice that this distance is the same for all those candidate slots since the pickup procedure just depends on the initial position of ASC_2 (ASC task responsible) and the waterside transfer point position.

Step 6: Calculate energy consumption required by ASC_2 to perform task $[a]$,$e_{[a_j]}$, by using the results from step 5 and expressions (X-Y). That is:

$$e_{[a_j]} = e_{[a_j]_{ASC_2}} = e_g\left(\Delta x_{[a_j]_{ASC_2}}\right) + e_s\left(\Delta y_{[a_j]_{ASC_2}}\right) + e_h\left(\Delta z_{[a_j]_{ASC_2}}\right) + e_l\left(\Delta z_{[a_j]_{ASC_2}}\right) \quad \textbf{(Equation 8)}$$

Since ASC_1 does not participate in this particular task, the energy required will be zero ($e_{[a]_{ASC_1}} = 0$).

Step 7: Calculate the time spent by ASC_2 to perform pick up process $[a]$ as:

$$t_{[a_j]} = t_{[a_j]_{ASC_2}} = \frac{\Delta x_{[a_j]_{ASC_2}}}{v_g} + \frac{\Delta y_{[a_j]_{ASC_2}}}{v_s} + \Delta z_{[a_j]_{ASC_2}}\left(\frac{1}{v_{l,empty}} + \frac{1}{v_{h,loaded}}\right) \quad \textbf{(Equation 9)}$$

Step 8:

- If ASC interference does not exist, determine rectilinear distance for each candidate slot $j, j \in \{1, n\}$ during drop-off procedure $[b]$ as:

$$\Delta x_{[b_j]_{ASC_2}} = \left|X_{cs_j} - X_{WTP}\right| \quad \Delta y_{[b_j]_{ASC_2}} = \left|Y_{cs_j} - Y_{WTP}\right| \quad \Delta z_{[b_j]_{ASC_2}} = \left|Z_{Wh} - Z_{cs_j}\right| \quad \textbf{(Equation 10)}$$

- If ASC interference exists (since the two ASCs are non-crossable) several different situations must be considered and solved efficiently (see deadlock rules at Section 4.3.4) as they have a negative effect on the productivity of the stacking yard and on the energy consumption.

  The additional travelling distance in each case is:

  1. An ASC has to wait until the other finishes its job: no distance is travelled
  2. An ASC needs to move to the edge to keep clear the claimed area:

$$\Delta x_{[cc_2]_{ASC_1}} = 2 \cdot \left|X_{edge} - X^1_{ASC_1}\right| \quad \Delta y_{[cc_2]_{ASC_1}} = 0 \quad \Delta z_{[cc_2]_{ASC_1}} = 0 \quad \textbf{(Equation 11)}$$

  3. A repositioning procedure is required and the container must be stacked temporarily in an auxiliary slot:

$$\Delta x_{[cc_3]_{ASC_1}} = \left|X_{as} - X^1_{ASC_1}\right| \quad \Delta y_{[cc_3]_{ASC_1}} = \left|Y_{as} - Y^1_{ASC_1}\right| \quad \textbf{(Equation 12)}$$

$$\Delta z_{[cc_3]_{ASC_1}} = \left|Z_{Wh} - Z_{as}\right| + \left|Z_{Wh} - Z_{cs_j}\right|$$

In such particular case (repositioning procedure) the above additional travelling distance must be added to the following one:

$$\Delta x_{[cc_3]_{ASC_2}} = \left|X_{as} - X_{WTP}\right| \quad \Delta y_{[cc_3]_{ASC_2}} = \left|Y_{as} - Y_{WTP}\right| \quad \Delta z_{[cc_3]_{ASC_2}} = \left|Z_{Wh} - Z_{as}\right| \qquad \textbf{(Equation 13)}$$

Step 9:

- If ASC interference did not exist in the previous step, calculate the energy consumption required by ASC_2 to perform task $[b]$ by using the results from step 8 and expressions (X-Y) for each candidate slot $j, j \in \{1, n\}$.

$$e_{[b_j]} = e_{[b_j]_{ASC_2}} = e_g\left(\Delta x_{[b_j]_{ASC_2}}\right) + e_s\left(\Delta y_{[b_j]_{ASC_2}}\right) + e_h\left(\Delta z_{[b_j]_{ASC_2}}\right) + e_l\left(\Delta z_{[b_j]_{ASC_2}}\right) \qquad \textbf{(Equation 14)}$$

- If ASC interference exists, the energy consumption in each situation is:

  1. An ASC has to wait until the other finishes its job: the same as before $(e_{[b_j]} = \mathrm{e}_{[b_j]_{ASC_2}})$

  2. An ASC needs to move to the edge to keep clear the claimed area:

$$e_{[b_j]} = e_{[b_j]_{ASC_2}} + e_{[cc_2]_{ASC_1}} = e_{[b_j]_{ASC_2}} + \left(e_g\left(\Delta y_{[cc_2]_{ASC_1}}\right) + e_s\left(\Delta y_{[cc_2]_{ASC_1}}\right)\right) \qquad \textbf{(Equation 15)}$$

  Where $e_{\mathrm{j},[b]_{ASC_2}}$ is calculated in eq. X and the second and third terms shows the additional energy consumption due to ASC_1 needs to be moved to an edge to clear the claimed are by the other ASC.

  3. A repositioning procedure is required and the container must be stacked temporarily in an auxiliary slot:

  Deriving from results from eq. (X), the total energy consumption to drop-off the incoming container in such situation is:

$$\begin{aligned} e_{[b_j]} = e_{[cc_3]_{ASC_1}} + e_{[cc_3]_{ASC_2}} &= e_g\left(\Delta x_{[cc_3]_{ASC_1}}\right) + e_g\left(\Delta x_{[cc_3]_{ASC_2}}\right) + e_s\left(\Delta y_{[cc_3]_{ASC_1}}\right) \\ &+ e_s\left(\Delta y_{[cc_3]_{ASC_2}}\right) + e_h\left(\Delta z_{[cc_3]_{ASC_1}}\right) + e_h\left(\Delta z_{[cc_3]_{ASC_2}}\right) \\ &+ e_l\left(\Delta z_{[cc_3]_{ASC_1}}\right) + e_l\left(\Delta z_{[cc_3]_{ASC_2}}\right) \end{aligned} \qquad \textbf{(Equation 16)}$$

Step 10:

- If did not exist ASC interference in step 8, calculate the time required by ASC_2 to perform task $[b]$ by using the results from step 8 and ASC kinematic characteristics for each candidate slot $j, j \in \{1, n\}$.

$$t_{[b_j]} = t_{[b_j]_{ASC_2}} = \frac{\Delta x_{[b_j]_{ASC_2}}}{v_g} + \frac{\Delta y_{[b_j]_{ASC_2}}}{v_s} + \Delta z_{[b_j]_{ASC_2}} \left(\frac{1}{v_{l,loaded}} + \frac{1}{v_{h,empty}} \right) \quad \textbf{(Equation 17)}$$

- If ASC interference exists, the time spent in each situation is:

1. An ASC has to wait until the other finishes its job:

   Since $t_{[cc_1]_{ASC_1}}$ is the waiting time that the blocking crane has to wait until the other crane finishes its job, it will not be considered within the time required to finish the procedure [b]. Therefore the time required is:

   $$t_{[b_j]} = t_{[b_j]_{ASC_2}} \quad \textbf{(Equation 18)}$$

   where $t_{[b]_{ASC_2}}$ is calculated according to eq. (X).

   It should be mentioned that the $t_{[cc_1]_{ASC_1}}$ does not decrease vessel discharging productivity since waterside ASC is continuously working and operations are not stopped. Nonetheless, block yard productivity is affected since an ASC is stopped due to crane conflict. The waiting time could be approximate to the length of time that ASC_2 requires to drop-off the container and finish its job.

2. An ASC needs to move to the edge to keep clear the claimed area:

   The time required to move to an edge will just depends on the distance and the gantry crane system speed (v_g), that is:

   $$t_{[cc_2]_{ASC_1}} = \left(\frac{\Delta x_{[cc_2]_{ASC_1}}}{v_g} \right) \quad \textbf{(Equation 19)}$$

   On the other hand, the time required to drop-off the container will be the same than in the case where it does not interference exists, as ASC_2 task is not interrupted. Therefore, the time required to achieve procedure [b] is:

   $$t_{[b_j]} = t_{[b_j]_{ASC_2}} \quad \textbf{(Equation 20)}$$

3. A repositioning procedure is required and the container must be stacked temporarily in an auxiliary slot:

   In such situation the time required to finish procedure [b] will be the addition of the both ASC tasks, given that the procedure is interrupted and both ASC participate in.

$$\begin{aligned} t_{[b_j]} &= t_{[cc_3]_{ASC_1}} + t_{[cc_3]_{ASC_2}} \\ &= \left(\frac{\Delta x_{[cc_3]_{ASC_1}}}{v_g} + \frac{\Delta x_{[cc_3]_{ASC_2}}}{v_g} \right) + \left(\frac{\Delta y_{[cc_3]_{ASC_1}}}{v_s} + \frac{\Delta y_{[cc_3]_{ASC_2}}}{v_s} \right) \\ &+ \left(\Delta z_{[cc_3]_{ASC_1}} + \Delta z_{[cc_3]_{ASC_2}} \right) \left(\frac{1}{v_{l,empty}} + \frac{1}{v_{h,loaded}} \right) \end{aligned} \quad \textbf{(Equation 21)}$$

Due to ASC_2 finishes its task before, as it stacks the incoming container closer, this ASC is going on with next job. Therefore, vessel turnaround time might be reduced and waterside productivity increased, although block yard productivity is reduced.

Repositioning technique is usually applied during pick times, since it is advisable to reduce loading and discharging time of vessels.

Step 11: Calculate the expected number of rehandles ($E_j[R]$) for each candidate slot $j, j \in \{1, n\}$ using expression X (procedure [r]).

Step 12: Determine the coordinates of an arbitrary slot where a rehandled container will be placed from candidate slot j, ($X_{rs_j}, Y_{rs_j}, Z_{rs_j}$).

In order to characterize an arbitrary rehandled movement, first we determine the coordinates ($X_{rs_j}, Y_{rs_j}, Z_{rs_j}$) by assuming the following hypothesis:

- Rehandled containers will be moved to stacks within the same bay or closer ones.
- Rehandled containers will be temporarily stacked in the highest stacks with the aim of reducing energy consumption in hoisting and lowering movements.
- Secondary rehandles have not been taken into consideration, since it is assumed that rehandled containers are moved back to their original stack once the target container was retrieved from the block yard as assumed also in Imai et al. 2002 and Imai et al. 2006.

Step 13: For each result obtained in step 11 and 12, calculate the rectilinear distance between ($X_{cs_j}, Y_{cs_j}, Z_{cs_j}$) and ($X_{rs_j}, Y_{rs_j}, Z_{rs_j}$) as:

$$\Delta x_{[r_j]_{ASC_i}} = \left|X_{cs_j} - X_{rs_j}\right| \quad \Delta y_{[r_j]_{ASC_i}} = \left|Y_{cs_j} - Y_{rs_j}\right|$$
$$\Delta z_{[r_j]ASC_i} = \left|Z_{Wh} - Z_{cs_j}\right| + \left|Z_{Wh} - Z_{rs_j}\right| \qquad \textbf{(Equation 22)}$$

Step 14: For each result obtained in step 11-13, calculate the energy consumption required to move $E_j[R]$ rehandled containers as:

$$e_{[r_j]_{ASC_i}} = e_g\left(\Delta x_{[r_j]_{ASC_i}}\right) + e_s\left(\Delta y_{[r_j]_{ASC_i}}\right) + e_h\left(\Delta z_{[r_j]_{ASC_i}}\right) + e_l\left(\Delta z_{[r_j]_{ASC_i}}\right) \qquad \textbf{(Equation 23)}$$

In such case we will considered the container weight of the incoming container, since it will be rehandled in case that $E_j[R]$ would be higher than zero.

Step 15: Calculate the total energy consumption required for each candidate slot j, as:

$$E[e_{[r_j]}] = e_{[r_j]_{ASC_i}} \cdot E_j[R] \qquad \textbf{(Equation 24)}$$

Step 16: Calculate the working time an ASC requires overcoming $E_j[R]$ rehandle movements.

Assuming that time required to do a rehandled movement from candidate slot j is:

$$t_{[r_j]_{ASC_i}} = \left(\frac{\Delta x_{[r_j]_{ASC_i}}}{v_g}\right) + \left(\frac{\Delta y_{[r_j]_{ASC_i}}}{v_s}\right) + \left(\Delta z_{[r_j]_{ASC_i}}\right)\left(\frac{1}{v_{l,empty}} + \frac{1}{v_{h,loaded}}\right) \quad \textbf{(Equation 25)}$$

The total expected working time of procedure [r] will be:

$$E\left[t_{[r_j]_{ASC_i}}\right] = E_j[R] \cdot t_{[r_j]_{ASC_i}} \quad \textbf{(Equation 26)}$$

Step 17: Estimate the ASC_i task list at future time t_2 and determine the coordinate position $(X^2_{ASC_i}, Y^2_{ASC_i}, Z^2_{ASC_i})$at that time ($t_2$) just before starting pick up process of retrieval procedure [c].

Step 18: Calculate the rectilinear distance from the candidate slot $j, j \in \{1, n\}$ to initial position of ASC_1 (pick up procedure [c]) as:

$$\Delta x_{[c_j]_{ASC_1}} = \left|X_{cs_j} - X^2_{ASC_1}\right| \quad \Delta y_{[c_j]_{ASC_1}} = \left|Y_{cs_j} - Y^2_{ASC_1}\right| \quad \Delta z_{[c_j]_{ASC_1}} = \left|Z_{Wh} - Z^2_{ASC_1}\right| \quad \textbf{(Equation 27)}$$

Step 19: Calculate energy consumption required by ASC1 to perform task [c], that is:

$$e_{[c_j]} = e_{[c_j]_{ASC_1}} = e_g\left(\Delta x_{[c_j]_{ASC_1}}\right) + e_s\left(\Delta y_{[c_j]_{ASC_1}}\right) + e_h\left(\Delta z_{[c_j]_{ASC_1}}\right) + e_l\left(\Delta z_{[c_j]_{ASC_1}}\right) \quad \textbf{(Equation 28)}$$

Step 20: Calculate the time spent to perform pick up process [c] as:

$$t_{[c_j]} = t_{[c_j]_{ASC_1}} = \frac{\Delta x_{[c_j]_{ASC_1}}}{v_g} + \frac{\Delta y_{[c_j]_{ASC_1}}}{v_s} + \Delta z_{[c_j]_{ASC_1}}\left(\frac{1}{v_{l,empty}} + \frac{1}{v_{h,loaded}}\right) \quad \textbf{(Equation 29)}$$

Step 21: Determine the landside transfer point position where external truck is placed waiting for the retrieved container $(X_{LTP}, Y_{LTP}, Z_{LTP})$.

Step 22: Calculate the rectilinear distance from the candidate slot $j, j \in \{1, n\}$, where the ASC_1 has just picked up the container, and the LTP as:

$$\Delta x_{[d_j]_{ASC_1}} = \left|X_{LTP} - X_{cs_j}\right| \quad \Delta y_{[d_j]_{ASC_1}} = \left|Y_{LTP} - Y_{cs_j}\right| \quad \Delta z_{[d_j]_{ASC_1}} = |Z_{Wh} - Z_{LTP}| \quad \textbf{(Equation 30)}$$

Step 23: Calculate energy consumption required by ASC_1 to perform task [d], that is:

$$e_{[d_j]} = e_{[d_j]_{ASC_1}} = e_g\left(\Delta x_{[d_j]_{ASC_1}}\right) + e_s\left(\Delta y_{[d_j]_{ASC_1}}\right) + e_h\left(\Delta z_{[d_j]_{ASC_1}}\right) + e_l\left(\Delta z_{[d_j]_{ASC_1}}\right) \quad \textbf{(Equation 31)}$$

Step 24: Calculate the time spent to perform drop-off process [d] from candidate slot j, as:

$$t_{[d_j]} = t_{[d_j]_{ASC_1}} = \frac{\Delta x_{[d_j]_{ASC_1}}}{v_g} + \frac{\Delta y_{[d_j]_{ASC_1}}}{v_s} + \Delta z_{[d_j]_{ASC_1}} \left(\frac{1}{v_{l,loaded}} + \frac{1}{v_{h,empty}} \right) \quad \textbf{(Equation 32)}$$

Step 25: Determine the total energy consumption required for each candidate slot $j, j \in \{1, n\}$ as:

$$e_j = \sum_i \sum_k e_{[k_j]_{ASC_i}} \qquad j \in \{1, n\},\ \ i \in \{1,2\},\ \ k \in \{\mathrm{a}, b, c, d, r, cc\} \quad \textbf{(Equation 33)}$$

Step 26: Determine the total time required to perform the stacking and retrieval process, rehandling movements and additional movements due to crane interference. That is:

$$t_j = \sum_i \sum_k t_{[k_j]_{ASC_i}} \qquad j \in \{1, n\},\ \ i \in \{1,2\},\ \ k \in \{\mathrm{a}, b, c, d, r, cc\} \quad \textbf{(Equation 34)}$$

The above expressions include the energy consumption (38) and the time (39) required doing all ASC movements for both working cranes in a block yard when an incoming container goes through it. That is, it takes into account useful working movements ([a], [b], [c] and [d] procedures) and, on the other hand, unproductive moments and waiting times due to inefficiencies ([cc] and [r] procedures).

Step 27: Calculate the score of each candidate slot ($Q_j, j \in \{1, n\}$) according Equation 1 from section 4.3.2.

4.3.4 Crane interference

It is worth noticing that in some cases the resolution of crane conflicts (see section 3.2.6) may not be aligned with the priorities given to the productivity/energy consumption of the ESSA. For example, the ESSA may be set to minimize energy consumption; however the sea ASC may claim a slot located behind the land ASC and, instead of waiting and penalizing productivity, the algorithm may force the land ASC to move backwards at greater energy expenditure.

4.4 Experimental setup

This section provides the specific details of the configuration of the DES model for the perpendicular layout.

4.4.1 Block characteristics

The block is modeled with standard 20feet slots, with a total length, width, and height of 40x9x5 slots respectively. A summary of block module dimensions is given next:

- Typical length: 40 TEUs (20-foot containers)
- Span: 32.5 m for 9 container rows
- Typical container spacing: 500 mm end-to-end, 400 mm side-to-side

- Minimum working spacing between two ASCs: 2 TEUs (20-foot containers)

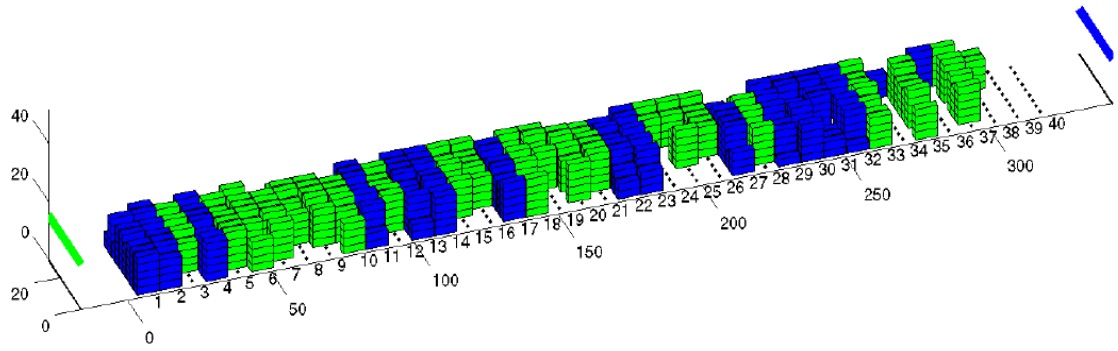


Figure 42. Example of the ASC block model with import (green) and export (blue) containers. Bay numbers range from 1 to 40. Longitudinal distances are indicated in 50 m intervals. Sea and Land cranes are depicted as green and blue horizontal bars.

4.4.2 ASC characteristics

4.4.2.1 ASC specifications

The ASC dimensions are provided next:

- Working height: 1-over-5 high-cube containers
- Working span: 9 containers
- Track gauge: 28 m
- Length:13.5 m
- Weights:
    - Whole crane: 185 tons
    - Cabin (trolley): 25 ton
    - Spreader (hoist): 10 ton

The next kinematic characteristics are used for the calculation of the crane movements.

- Gantry travel: 200 - 240 m/min (full – empty)
- Cross travel: 60 m/min
- Hoisting/lowering: 39 - 72 m/min (full – empty)
- Gantry acceleration: 0.4 m/s²
- Trolley acceleration: 0.4 m/s²
- Hoisting/lowering acceleration: 0.35 m/s²

4.4.2.2 Dispatching rules

As previously indicated, containers in the ASC workload are dispatched in the order of arrival. The FIFO rule (or earliest due date priority rule) is preferred in this study because the utilization of more sophisticated crane deployment strategies may increase enormously the computational cost of the efficient stacking algorithm in the simulations.

4.4.3 Container traffic generation

Data from a real container terminal was utilized to generate the traffic inputs for the simulation model, resulting in different sequences of containers arriving to both transfer points of the block. Poisson distributions were used to introduce stochasticity into the analysis.

The distribution of container weights (Figure 41) is also taken from data from a real container terminal, although variations are introduced so as to maintain the confidentiality of the data.

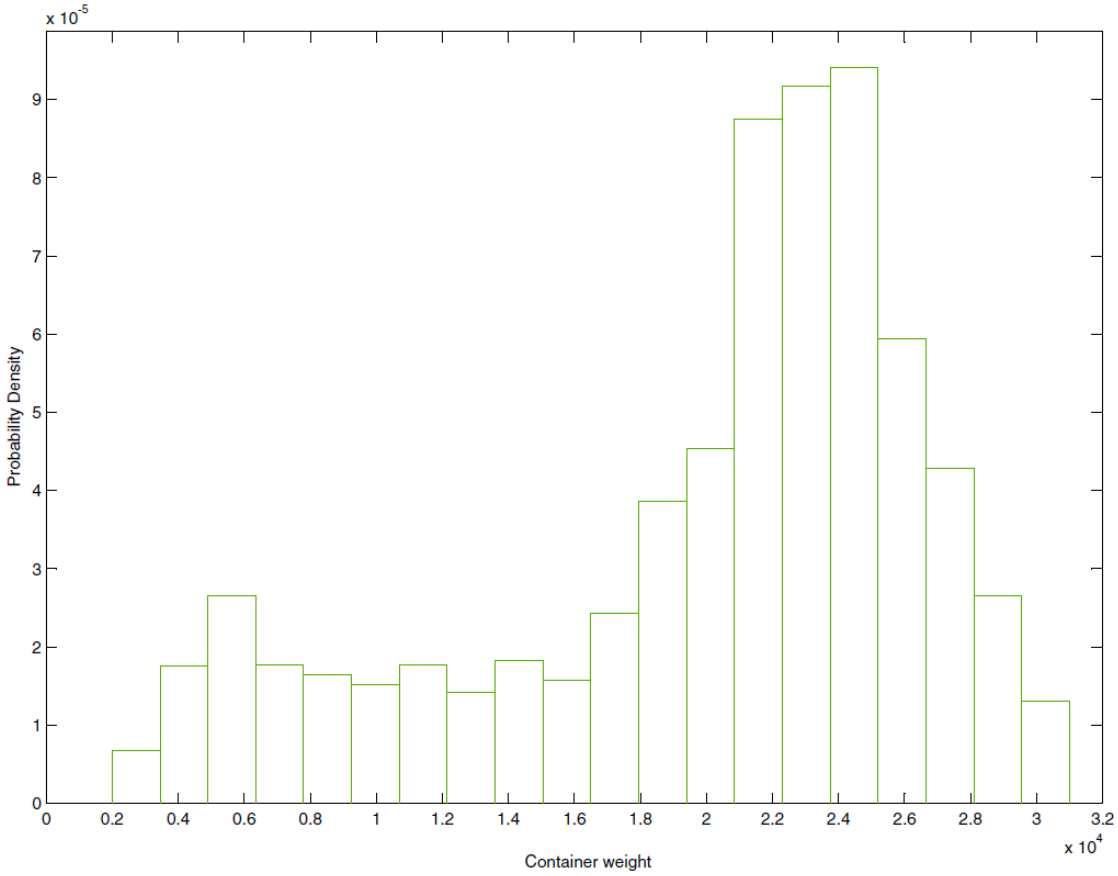


Figure 43. Probability density function of the container weights used in the simulation.

With respect to the ETs, arrivals take place only when terminal gates are open. Again the generation of ET daily arrivals is stochastically generated from the probability density function illustrated in Figure 42. The actual number of containers arriving to the terminal per day is

calculated taking into account the dwell time so as to produce a stable inventory with the desired level of occupancy.

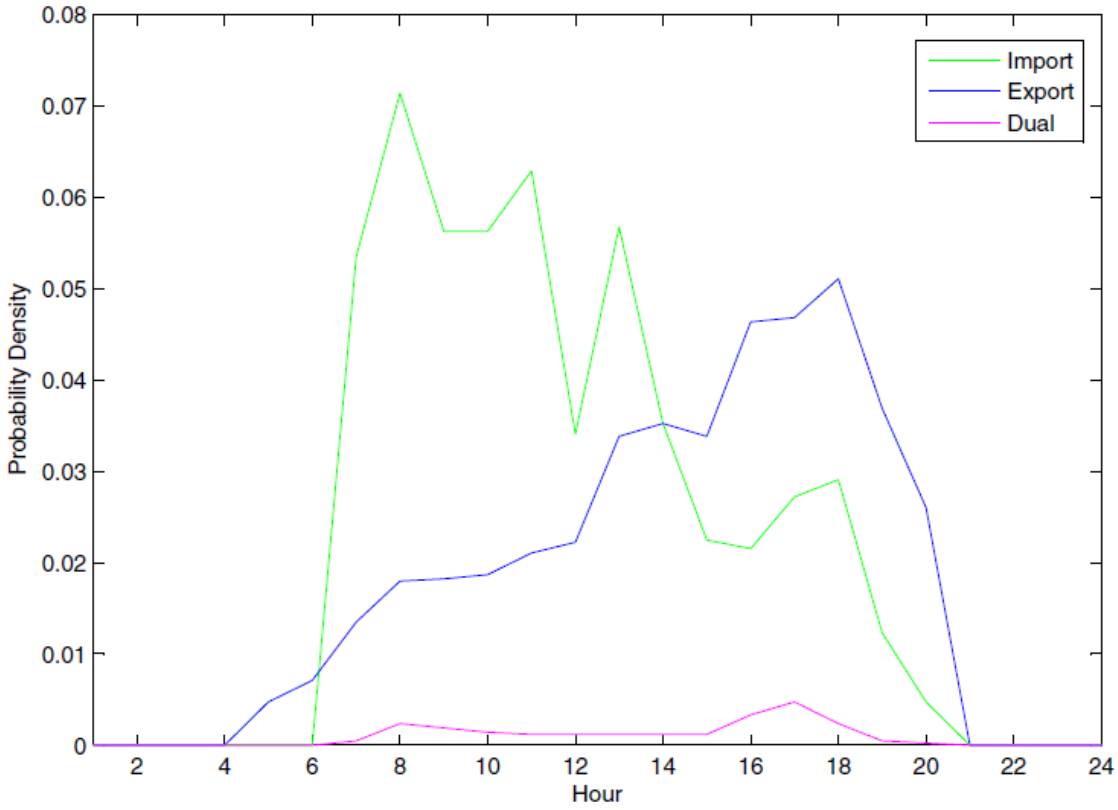


Figure 44. Probability density function of the Import, Export and Dual ET arrivals used in the simulation.

4.4.4 Calculation scenarios

This way, two levels of block occupancy, low and high, are tested in the experiments, as indicated in Table 4. The first set of tests cause the block inventory size to fluctuate around 40% of its capacity, which is considered a low degree of occupancy. In contrast, the second set of tests increases the average occupancy up to 60% which further restricts ASC operations. The different arrival pattern of containers from sea and land causes the block occupancy to fluctuate around the average value around ± ≈20% during the simulation period, and so the simulation reproduces peak situations that bring the block close to its maximum capacity. Note that since stacks are limited to five slots and container reshuffles are forced to take place within the same bay, four slots must remain empty in each bay to allow the relocation of a complete pile to recover a container when stacked in the lowest position. This constraint implies that at least ≈9% of the block must always remain empty in order to make container re-handling possible.

The duration of the experiments was set to 28days, starting from an empty block. A warm-up period of 10 days allows the block inventory to grow until the desired value of occupancy. Warm-

up is followed by an additional 18-day period for data collection. This duration of this period is established so that no significant fluctuations are observed in the average value of the key performance indicators after that time.

Traffic scenario	Units	Low traffic	High traffic
Average block occupancy	%	40	60
Import trucks	Number per day	90	120
Export trucks	Number per day	150	250
Dual trucks	Number per day	4	8

Table 3. Definition of traffic scenarios.

The block layout, defined as the distribution of import and export bays in the block, is important at the beginning of the data collection period because it determines not only the relative distance travelled back and force by the two cranes carrying out stacking and retrieval operations, but also the occurrence of crane trajectory intersections. The dedication of bays to import or export containers does not change frequently during the simulation, and so the block layout at the 10th day of the simulation may bias the final results. As a consequence, several measures shall be taken to avoid this effect. First, for each simulation experiment, the different stacking algorithms carry out operations after day 10 starting from a common initial layout, and so differences in results will be due only to the stacking rules of each algorithm. For stacking algorithms with a random component, the generation of random numbers in Matlab is seeded to ensure that the same random sequences are always used. As for the initial layouts, import and export bays are assigned randomly. This way, segregation of import and export containers arising from the utilization of the LSA from an empty block was avoided.

With respect to the container traffic, pre-generated arrivals are fed into the model in order to provide the same data to all the stacking algorithms. Container characteristics include flow (import/export), weight (or equivalently, weight category), port of destination and traffic line. The combination of such factors results in 24 different categories to be considered for export container stacking. As for the import container dwell time, it follows a Weibull distribution with an average stay of 3 days.

As indicated above, the daily arrival pattern for external trucks is generated from real data pertaining to a container terminal, where the gates operate from 7 am to 9 pm. Dual traffic is also considered, meaning that some external trucks bring an export container to the block and then take in import container in the same service.

As for the vessel traffic arrivals, one vessel berthing per day is stochastically generated. The vessel discharges a number of import containers per block according to Table 4. Upon discharge of the

last container, the simulation model performs a block search reserving between 70% and 90% of the export container inventory to be sequentially uploaded onto the vessel later on.

4.5 Results and discussion

Simulations are carried out using a single-core 2.7 gigahertz personal computer with 8 gigabytes of RAM. Each run typically consumes 6-32hrs depending on the stacking algorithm, with the ESSA being the most time consuming of all.

The stacking algorithms are analyzed with the focus on two main aspects, with several key performance indicators (KPIs) will be used to characterize both energy efficiency and productivity. First, *efficiency* is examined by computing the average energy expenditure per container; second, *productivity* is evaluated by looking at the time needed by the ASCs and block performance.

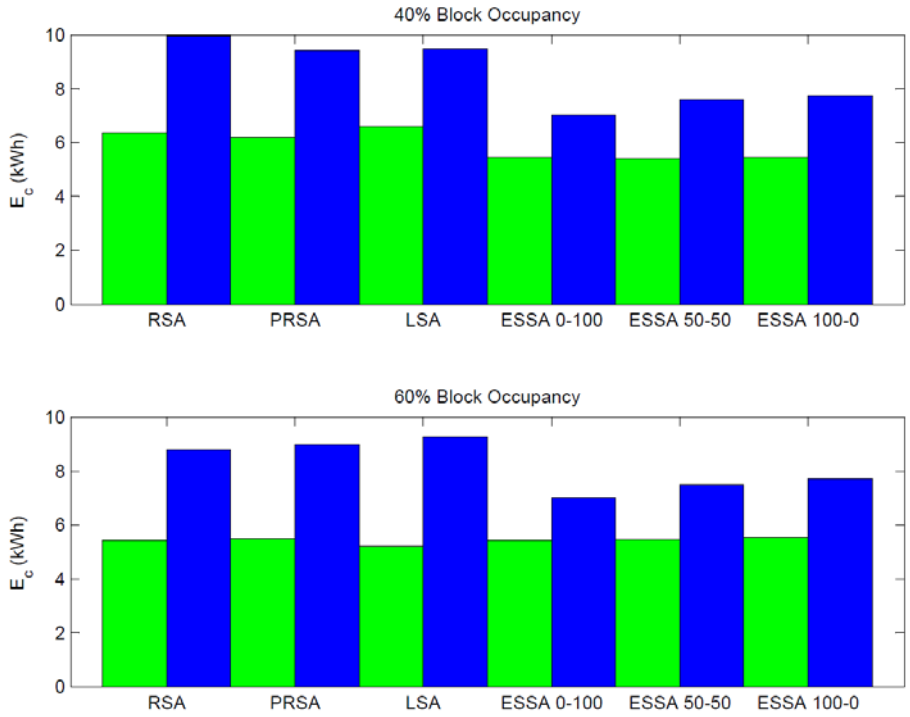


Figure 45. $\overline{E_C}$ in kWh per Import (green) and Export (blue) for the stacking algorithms. Top: 40% Block Occupancy Level (BOL); bottom: 60% BOL.

Results prove the capability of the ESSA to improve significantly both efficiency and productivity of the operations compared to other algorithms. With respect to the energy efficiency, the ESSA improves the efficiency about 20% when considering export operations. For import operations, the ESSA is 15% more efficient under low occupancy level, while the similar performance is observed for higher levels. The weight combination W_E = 0% / W_P = 100% provides the best results for the ESSA, therefore no trade-off is observed between the weights assigned to the criteria and the energy consumption.

With respect to productivity, the weight combination W_E = 0% / W_P = 100% minimize duration of the ASC retrieval and housekeeping operations, while the W_E = 100% / W_P = 0% yields better results when considering stacking operations and CET. VST is also improved, but the best weight combination depends on the block occupancy level.

In the next sections, results are analyzed in detail.

4.5.1 Energy consumption

Average energy consumption per container is evaluated by adding the energy associated to the ASCs movements needed to carry out the container transit along the block, which will typically comprise one stacking operation, a number of housekeeping operations, and a retrieval operation. In addition, the container may be relocated within the same bay in order to retrieve a container buried underneath.

4.5.1.1 General

The evolution of the energy expenditure per container with respect to time is depicted in Figure 49. Regardless the stacking algorithm, the average energy expenditure for import and export containers becomes rapidly stable, and the average value after day 24 varies within a small percentage, and so the duration of the experiments is considered adequate.

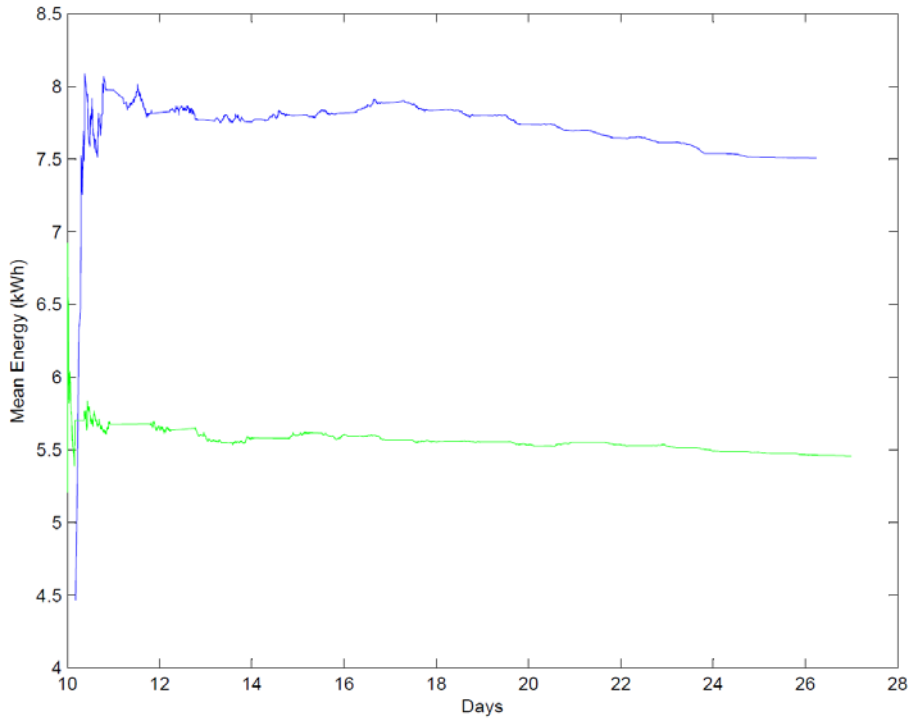


Figure 46. Average energy consumption per Import (green) and Export (blue) container with respect the date of entrance of the container in the terminal.

From the figure it is evident that, as expected, more energy is required to handle export containers due to the greater amount of handling needed to carry out housekeeping operations.

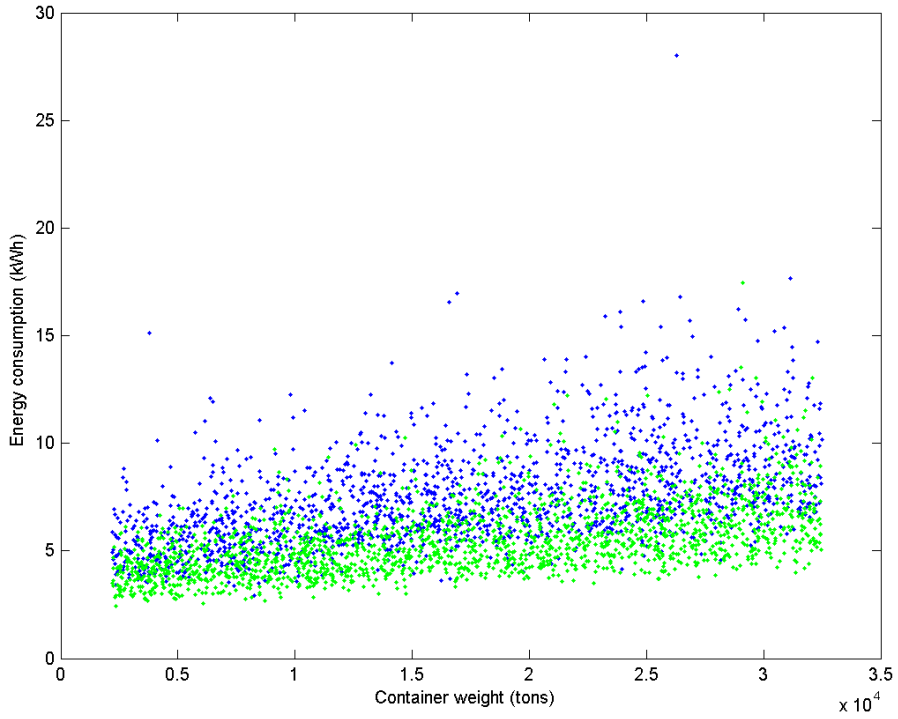


Figure 47. Example of the $\overline{E_C}$ per Import (green) and Export (blue) container weight.

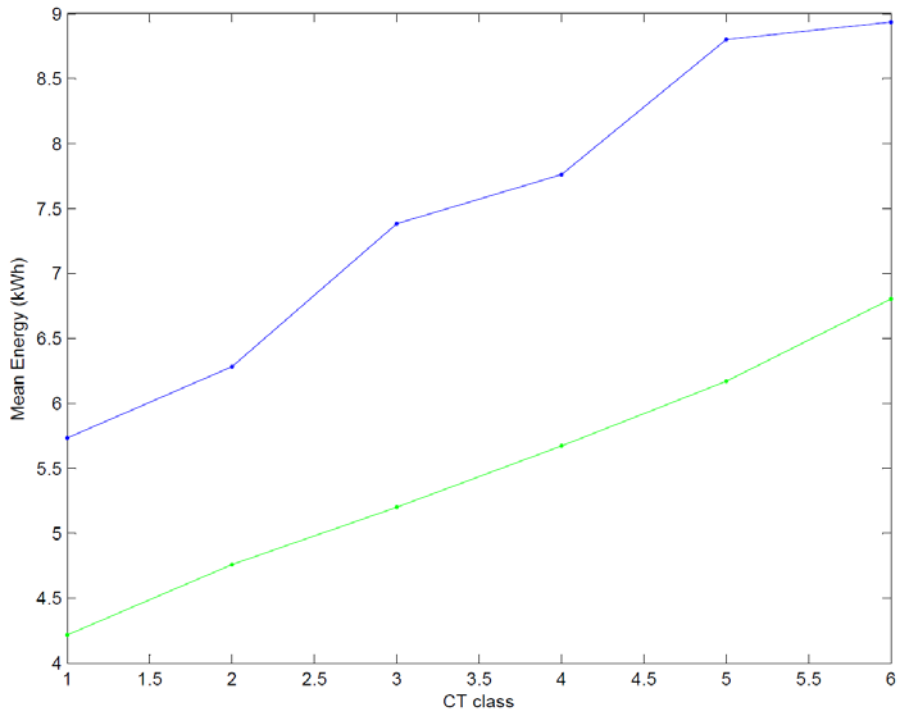


Figure 48. Example of the $\overline{E_C}$ per Import (green) and Export (blue) container weight class.

With respect to the average energy consumption per container weight ($\overline{E_C}$ hereinafter), the value increases as container weight (W_{CT}) increases (Figure 44). Although this result is somewhat obvious, it validates and quantifies an essential assumption used by the ESSA, which is that weight information is important at the time of stacking a container. The relationship between container weight and container energy consumption is more evident when computing the energy with respect to the container weight class (Figure 45).

From this point of view, the Potential Energy Model is capable of capturing the influence of the container weight on $\overline{E_C}$, which is around 5% for import containers and 8% for export containers. Since containers of weight category 6 are six times heavier than those of category 1, the influence of W_{CT} on $\overline{E_C}$ is somehow limited.

This fact already suggests that gantry displacements, which the mobilization of the much greater ASC mass compared to W_{CT}, plays a significant role in the total $\overline{E_C}$ compared to the expenditure due to container lifting movements.

4.5.1.2 Composition of the energy consumption

Table 15 and Table 16 summarize the results with respect to the average energy consumption per container ($\overline{E_C}$) for the 40% and 60% occupancy scenarios respectively. The results can be summarized in the following statements:

- Contrarily to intuition, $\overline{E_C}$ decreases as the occupancy increases, mainly because, as more containers arrive to the TPs, ASCs devote more time to prior tasks, having less time to carry out housekeeping operations, which are responsible of a significant portion of the total expenditure. This outcome suggests that further improvement in the energy efficiency must focus on the optimization of housekeeping operations.
- When comparing import and export operations, more housekeeping is observed for export operations, hence $\overline{E_C}$ is always smaller for import containers.
- With respect to the strategy, the ESSA performs better than the benchmarking algorithms for the export operations. For import operations, the ESSA is more efficient under low block occupancy, and almost as good when occupancy is high.
- Overall, the ESSA is capable of improving the efficiency around 20%, which is a considerable amount. To put this in context, considering the ASC characteristics and the annual volume of traffic of the present study (around 2 Million TEUs), with the day / night industrial price of around 0.1/0.05€ per kWh, the potential savings could be in the order of magnitude of one hundred thousand euros (O(10^5€)).

Operation	Movement	Type	Algorithm					
						ESSA (W_E / W_P)		
			RSA	PRSA	LSA	0% / 100%	50% / 50%	100% / 0%
IMPORT CONTAINERS								
Stack	Gantry	P	0.93	0.98	1.00	0.82	0.84	**0.79**
		U	**0.01**	0.03	**0.02**	0.06	0.03	0.03
	Trolley	P	0.01	0.01	0.01	0.01	0.01	**0.01**
		U	0.00	0.00	0.00	0.00	0.00	0.00
	Hoist	P	1.40	1.43	1.36	1.36	**1.35**	1.36
		U	0.00	0.00	0.00	0.00	0.00	0.00
Delivery	Gantry	P	0.96	0.91	**0.83**	0.86	0.89	0.90
		U	0.06	0.06	**0.03**	0.04	0.05	0.05
	Trolley	P	0.02	0.02	0.02	**0.01**	**0.01**	**0.01**
		U	0.00	0.00	0.00	0.00	0.00	0.00
	Hoist	P	1.79	1.67	1.67	**1.15**	**1.15**	1.29
		U	1.17	1.09	1.35	1.12	1.08	**1.01**
Total Import			6.36	6.20	6.59	5.47	**5.39**	5.47
EXPORT CONTAINERS								
Stack	Gantry	P	**0.73**	0.75	0.95	1.02	0.92	0.97
		U	**0.03**	**0.03**	0.04	0.10	0.07	0.07
	Trolley	P	0.01	0.01	0.01	0.01	0.01	0.01
		U	0.00	0.00	0.00	0.00	0.00	0.00
	Hoist	P	1.46	1.39	**1.38**	1.39	**1.38**	**1.38**
		U	0.00	0.00	0.00	0.00	0.00	0.00
Delivery	Gantry	P	1.40	1.32	1.08	**0.81**	0.97	0.97
		U	0.07	0.06	**0.02**	**0.02**	0.03	0.03
	Trolley	P	0.04	0.04	0.05	**0.02**	0.03	0.03
		U	0.00	0.00	0.00	0.00	0.00	0.01
	Hoist	P	3.90	3.50	4.03	**2.10**	2.43	0.26
		U	2.32	2.28	2.59	**1.54**	1.71	1.73
Total Export			9.98	9.41	9.49	**7.04**	7.59	7.73

Table 4. Average energy expenditure per container in kWh. 40% Occupancy. Best score for each indicator algorithm is highlighted. BOL = Block Occupancy Level. CET = Container Exit Time, WE: weight assigned to the energy criterion. WP: weight assigned to the productivity criterion. P = Productive, U = Unproductive.

Operation	Movement	Type	Algorithm					
						ESSA (W_E / W_P)		
			RSA	PRSA	LSA	0% / 100%	50% / 50%	100% / 0%
IMPORT CONTAINERS								
Stack	Gantry	P	0.89	0.88	0.89	0.86	0.85	**0.82**
		U	0.01	0.01	0.01	**0.01**	0.00	0.00
	Trolley	P	0.01	0.01	0.01	0.01	0.01	**0.01**
		U	0.00	0.00	0.00	0.00	0.00	**0.00**
	Hoist	P	1.37	1.40	1.38	1.36	1.34	**1.33**
		U	0.00	0.00	0.00	**0.00**	0.00	0.00
Delivery	Gantry	P	0.91	0.90	**0.78**	0.91	0.93	0.94
		U	0.04	0.04	0.01	**0.06**	0.06	0.06
	Trolley	P	0.01	0.01	0.01	**0.01**	0.01	0.01
		U	0.01	0.01	0.02	**0.02**	0.02	0.02
	Hoist	P	1.15	1.21	1.12	**1.11**	1.12	1.22
		U	1.01	1.00	**0.98**	1.08	1.10	1.14
Total Import			5.42	5.48	**5.22**	5.44	5.45	5.55
EXPORT CONTAINERS								
Stack	Gantry	P	0.71	**0.70**	0.74	0.81	0.77	0.70
		U	0.02	0.01	**0.00**	0.03	0.03	0.02
	Trolley	P	0.01	0.01	0.01	0.01	0.01	0.01
		U	0.00	0.00	0.00	0.00	0.00	0.00
	Hoist	P	1.40	**1.35**	1.38	1.37	1.38	1.37
		U	0.00	0.00	0.00	**0.00**	0.00	0.00
Delivery	Gantry	P	1.34	1.34	1.21	**1.07**	1.17	1.22
		U	0.04	0.05	**0.02**	**0.02**	0.03	0.03
	Trolley	P	0.03	0.03	0.04	0.03	0.03	**0.02**
		U	0.03	0.03	0.05	0.03	**0.02**	**0.02**
	Hoist	P	2.90	3.23	3.00	**2.06**	2.27	2.43
		U	2.33	2.25	2.81	**1.79**	**1.79**	1.86
Total Export			8.80	9.00	9.27	**7.02**	7.51	7.71

Table 5. Average energy expenditure per container in kWh. 60% Occupancy.

- Comparing the results of the ESSA algorithm according to the three weight combinations for W_E and W_P, it is worth noticing that favoring efficiency or productivity through the assignment of weights may not conduce to better efficiency or productivity respectively; the fact the results are not aligned with the criteria for weight assignment underscores the complexity and the dependence of the handling processes in the block: when benefiting dependence of the different processes characterizing inside the block.
- With respect to crane movements, hoist consumes the majority of the energy in the average duty cycle regardless the stacking algorithm or the inventory size. To this respect, it is worth noticing that *unproductive hoist* is of the same order of magnitude as *productive hoist*, which is the reason why optimization of stacking strategies focus mostly on rehandling operations.

- Regarding gantry movements, relatively less energy is consumed, but on the other hand they consume relatively more time than hoisting. The incidence of unproductive gantry is small compared to productive gantry.

4.5.1.3 Rehandling

With respect to rehandling, they include the operations needed to recover a container buried in the pile when the container is being delivered for final departure, and also when the ASC is carrying out a housekeeping movement. Figure 46 illustrates the PDFs of retrieval of an import container depending on the number of containers on top (or equivalently, the number of induced rehandles), and also with respect to the duration of its own stay at the terminal.

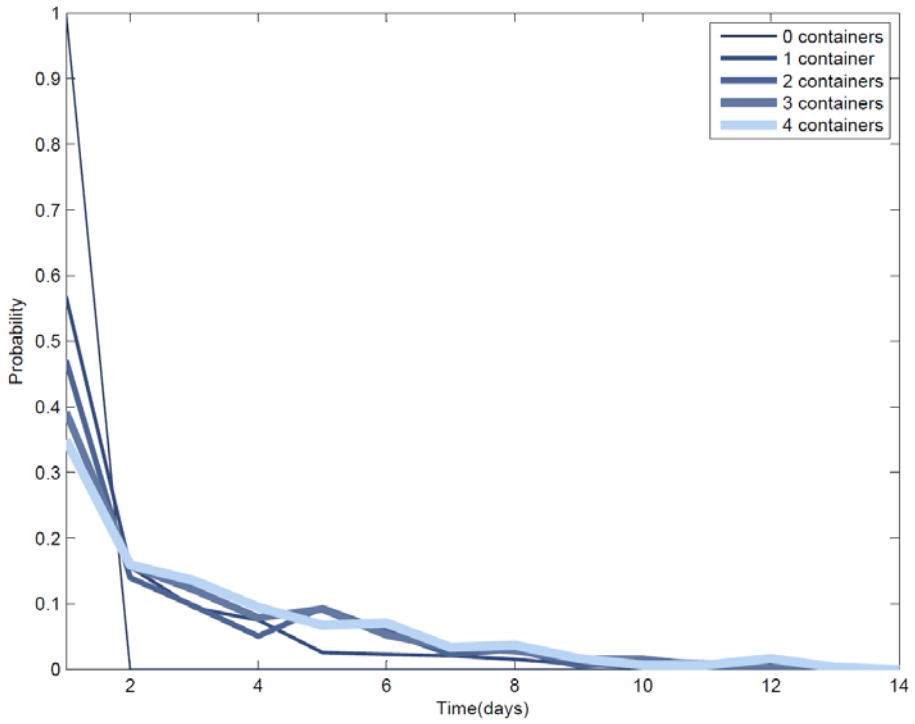


Figure 49. CDF of the probability of an import container being relocated with respect to the position it occupies in the stack and the time. X = time in days. Each curve corresponds to the number of containers placed on top of the pile being retrieved.

The figure is descriptive of the development of import container rehandling; upon container arrival, it is stacked in the top of a pile, and therefore containers retrieved short after are more likely to occupy a higher position in the pile. As time passes, containers tend to suffer rehandling; after day two, the probability of being retrieved is approximately the same regardless the number of containers on top, or equivalently, the position the container occupies in the pile.

This information is utilized by the stacking algorithm at the time of placing a container in the block to evaluate the probability of all the containers of a candidate pile to be retrieved from the stack, and adjusts well to Equation **35**.

$$p = 2E-05 \cdot d^4 - 0{,}0006 \cdot d^3 + 0{,}0093 \cdot d^2 - 0{,}0702 \cdot d + 0{,}2197 \quad \textbf{(Equation 35)}$$

Where d is the duration of container stay in the block (in days).

Table 8 provides an insight on the incidence of rehandling obtained in the numerical simulations. As expected, export containers experience greater rehandling import containers due to housekeeping operations. When considering the total rehandling for each strategy, the ESSA (W_E / W_P = 100% /0%) outperforms the benchmark algorithms; performance is especially superior for export operations. Results also indicate that when the occupancy increases, the amount of rehandling increases in a greater proportion, as indicated by Kim (1997).

With respect to the amount of secondary rehandling, to the best of the author's knowledge, this question has not been thoroughly addressed in the literature, and so it is worth the analysis. Secondary rehandling measures the number of unproductive movements suffered by a container that has been relocated more than once. While some authors like Imai et al. (2002) and Imai et al. (2006) assume that rehandled containers are moved back to their original stack once the target container is retrieved, in practice that is not the case, and therefore rehandling will likely disrupt the uniformity of containers in the block piles stacked under the container grouping principle. Kim et al. (2000) assume that export containers are relocated no more than once to simplify the problem, expecting that the amount of secondary rehandling is negligible.

Average Performance indicators	**BOL (%)**	**Container Type**	**Algorithm**					
			RSA	PRSA	LSA	ESSA (W_E / W_P)		
						0% / 100%	50% / 50%	100% / 0%
Total Rehandling (#)	40%	IMP	2,148	1,967	2,451	2,021	1,999	**1,421**
		EXP	5,219	4,521	6,490	3,033	3,407	**1,926**
	60%	IMP	2,824	2,692	2,834	2,915	2,988	**2,549**
		EXP	6,084	4,756	8,083	3,873	4,190	**3,589**
Secondary Rehandling (%)	40%	IMP	25.09	24.15	25.01	25.04	25.66	**22.87**
		EXP	21.82	22.76	24.17	**25.59**	25.33	28.95
	60%	IMP	24.72	27.09	22.97	25.01	25.67	**25.11**
		EXP	20.99	26.77	18.20	**24.17**	24.94	29.17

Table 6.Container rehandling. BOL = Block Occupancy Level.

For export containers, the stacking strategy and the nature (static or dynamic) of the setting constitute important factors affecting the incidence of secondary rehandling. Recall that in this model setup the arrival of containers is not known in advance, and so the implementation of optimization methods such as Kim et al. (2000) to reduce rehandling is not feasible. Another source of secondary rehandling is the uncertainty in the elaboration of vessel uploading plans: at the time of retrieving containers to be uploaded on the vessel, the loading plan may have experience changes from that at the time of stacking those containers. This phenomenon is taken

into account by the model by selecting 80% of export containers in a random way among all the candidates bound to that containership.

However, results clearly indicate that the proportion of secondary rehandling represents around 20-30% depending on the case; in conclusion, secondary rehandling is significant enough so as to be considered in any analysis.

4.5.2 ASC performance

In addition to energy efficiency, Table 8 presents the experimental results regarding block performance. In this case, the KPIs *container exit time* (CET) and *vessel service time* (VST), as described in Section 1.3.4, are used to characterize block productivity, and results discern import and export containers.

Recall CET measures the time needed for the land ASC to place the container on the external truck for departure relative to the moment the container is requested, and so it characterizes the retrieval operation of containers individually. Contrarily, VST performance indicator is global, as it accounts for the time needed to complete the uploading of all export containers bound to a vessel. Although in reality containers bound to a vessel are usually distributed over several blocks, we can assume in this case that results from a single block are representative of the whole uploading process.

Average Performance indicators	**BOL (%)**	**Container Type**	**Operation**	**Algorithm**					
				RSA	PRSA	LSA	ESSA (W_E / W_P)		
							0% / 100%	50% / 50%	100% / 0%
Vessel Upload Service Time (h)	40%	EXP	S	6.80	6.30	6.59	6.25	**5.39**	6.64
	60%			9.03	8.67	10.80	**8.40**	8.58	8.76
CET (min)	40%	IMP	R	7.59	7.56	8.30	8.02	7.59	**7.08**
	60%			19.02	16.27	15.38	21.66	16.83	**14.60**
ASC time per operation (sec)	40%	IMP	S	145.21	145.95	160.76	126.55	120.45	**117.09**
			R/H	263.91	248.33	244.10	**198.21**	203.98	198.06
		EXP	S	134.49	127.41	128.88	131.75	125.14	**123.39**
			R/H	497.34	478.49	449.17	**306.66**	346.27	350.05
	60%	IMP	S	152.04	151.93	172.18	138.54	132.82	**125.06**
			R/H	220.78	219.63	215.14	**216.33**	219.31	219.52
		EXP	S	136.10	124.71	131.78	125.39	132.23	**114.76**
			R/H	452.97	446.61	478.04	**349.80**	373.78	370.73

Table 7. Summary of results. Best score for each indicator algorithm is highlighted. BOL = Block Occupancy Level. CET = Container Exit Time, W_E: weight assigned to the energy criterion. W_P: weight assigned to the productivity criterion. S = Stack, R = Retrieval, H = Housekeeping.

Table 8 also includes ASC time per type of operation as an indirect measure of the ASC performance. They account for the average time needed by the ASC to carry out the stacking (S) and retrieval/housekeeping (R/H) operations. As R/H operations are very similar (both may produce rehandling containers when retrieving another that is buried in the pile), they are

considered together in the table for simplicity. As it is obvious, the duration of retrieval and housekeeping are much longer than stacking operations.

The numerical experiments confirm that the ESSA outperforms the benchmarking algorithms when delivering import and export containers, although again results from this algorithm are not always consistent with the weight assigned to the decision criteria. Conclusions drawn from the results are summarized in the following statements:

- Increasing occupancy level produces greater VST and CET times, as expected.
- Regarding average ASC operation times, stacking times increase with the occupancy level; however when considering retrieval and housekeeping operations, that is not the case, mainly because less occupancy level allows for a greater number of housekeeping operations.
- Housekeeping rules have a strong random component during the selection of candidate containers; thus, these operations seem to introduce disturbance in the results; optimization of the housekeeping operations is an interesting topic for future research as it may lead to further improvement in the performance of the algorithms.
- Regarding the comparison of the stacking algorithms, the ESSA performs better in terms of productivity than the benchmarking counterparts, especially when considering the VST and CET at the same time. This tendency is preserved regardless the volume of traffic, although greater differences are observed as the size of the inventory increases, but the behavior is not consistent when considering the weight assigned to W_E and W_P. The combination $W_E = 100\%$ / $W_P = 0\%$ produced the best results from the point of view of the import trucks. Considering the vessels, $W_E = W_P = 50\%$ gives better VST for the low occupancy level, and $W_E = 0\%$ / $W_P = 100\%$ for the high occupancy level. Considering the overall processes, the ESSA seems capable of saving a significant amount of time, especially under high traffic conditions.
- In terms of ASC times, the ESSA with $W_E = 100\%$ / $W_P = 0\%$ outperforms the benchmarking algorithms for stacking operations, it indicates that further improvement could be achieved in the upstream process. The ESSA $W_E = 0\%$ / $W_P = 100\%$ outperforms the benchmarking algorithms for all the retrieval and housekeeping operations,
- As expected, worse results are achieved by the algorithms with a random basis compared to the ESSA.
- It is also worth noticing the similar behavior observed between the reference stacking algorithms used in the terminal, the RSA and the LSA, which is consistent with the terminal observations.

- VST is slightly better when utilizing the PRSA instead of the LSA, but when considering the import CET, the algorithms exhibit just the opposite behavior, with the LSA being more efficient in the low traffic tests, whereas the PRSA is more capable when the inventories are of larger size.

4.5.3 Summary of results

Table 9 provides a visual summary of the KPIs in the analysis. In general, the ESSA with W_E = 100% / W_P = 0% provides better results for Export containers, while W_E = 100% / W_P = 0% is preferred for import containers.

Average Performance indicators	BOL (%)	Container Type	Operation	Algorithm					
				RSA	PRSA	LSA	ESSA (W_E / W_P) 0% / 100%	ESSA (W_E / W_P) 50% / 50%	ESSA (W_E / W_P) 100% / 0%
Energy consumption per container	40%	EXP					✓		
	60%						✓		
	40%	IMP						✓	
	60%					✓			
VST	40%	EXP	S					✓	
	60%						✓		
CET	40%	IMP	R						✓
	60%								✓
ASC time per operation	40%	IMP	S						✓
			R/H				✓		
		EXP	S						✓
			R/H				✓		
	60%	IMP	S						✓
			R/H				✓		
		EXP	S						✓
			R/H				✓		

Table 8. Summary of best scores results.

4.6 Conclusions

This work introduced an efficient stacking algorithm (ESSA) to address the location assignment problem with the goal to minimize energy consumption and maximize ASC productivity in yard handling operations. A sensibility analysis was conducted to understand the relationship between the volume of traffic and the performance of the ESSA. When compared to other benchmark algorithms, the ESSA produces significant better results not only in terms of vessel turn-around and container exit times, but also crane productivity and energy expenditure. These results indicate that the ESSA has potential for improving terminal handling policies even under congested situations.

The analysis of the energy consumption confirms that hoist movements consume the majority of the energy in the cycle, whereas gantry account for a greater proportion of the ASC cycle times.

Further research may focus on the optimization of the housekeeping operations in order to reduce amount of gantry need to handle the containers.

Chapter 5

Optimization of the ASC Block Dimensions

5.1 Introduction

As explained in the literature review, the design of container terminals in early stages mainly depends on the desired level of automation, which in turns determines the type of MHE to be used. At the same time, designers must rely on estimations of the traffic forecasts (volumes and type of containers, etc.) and the desired terminal throughput, which determine the numbers and type of MHE selected, the capital investment, and the operational costs.

Once established the strategic basis of design, it is time to address design detailed aspects such as the yard layout and configuration. The resulting number of blocks, rows and bays and stacking height needs to be established and the type and amount of MHE need to be decided simultaneously. I.e., given the availability of yard space (equivalently the number of ground slots) and the stack height, designers can estimate the amount of rehandling during retrieval operations; the number of YCs will determine the travel distances and the time needed to stack and retrieve the containers, and ultimately the terminal throughput. As it is obvious, availability of abundant space is always desirable in terms of productivity and operational costs (less rehandling), but on the other hand space has a significant impact on the amount of investment needed to construct the yard as well as on other running costs. In summary, trade-offs are frequently found between design related decisions, productivity and terminal costs.

5.2 Overview

Kim and Kim (2002) focused on the amount of space required in a terminal for import containers, but the yard layout is not provided. They proposed a method of determining the optimal amount of storage space and the optimal number of transfer cranes for handling import containers. The cost model considers the space cost, the investment cost of transfer cranes, and the operating cost of transfer cranes and trucks.

Kim and Park (2008) proposed a design method to determine the layout type, the outline of the yard, and the numbers of vertical and horizontal aisles. Transfer cranes and yard trucks are used for handling the containers. They compare the performance of parallel versus perpendicular layouts with a static, equation-based approach and a simulation methodology, concluding that the parallel layout is more productive.

Petering et al (2009), and Petering (2009) conducted simulation study to analyze the influence of the block dimensions in a multiple-berth parallel terminal using a complete DES model that considers the detailed movement of individual containers passing through a vessel-to-vessel transshipment terminal over a several week period. Experiments consider four container terminal scenarios and several different yard crane deployment systems. Quay and yard performance indicators are evaluated (quay crane rate, YC rate, etc.) the trade-offs between the block dimensions and the terminal throughput. The authors found that a block length between 56 and 72 (20-ft) slots and 6 to 12 rows yields the highest quay crane work rates. In addition, they concluded that higher QC throughput rates are obtained when the deployment system restricts YCs movements instead of allowing greater yard crane mobility.

The first study that addresses the influence of the block size on the performance of perpendicular automated terminals is given by Kemme (2012). A simulation study was carried to assess the effects of four rail-mounted-gantry-crane systems and 385 yard block layouts that differ in block length, width, and height. In this study, yard performance is evaluated only in terms of SC and ET waiting times, but the cranes throughput is not considered. Results show that the performance of the RMG yard blocks depends greatly on the layout. They concluded that high stacking has to be avoided, and also that wider blocks are preferable. The DES model, as described in Kemme (2010), can basically implement two stacking strategies: a random stacking strategy (RaS) and a greatly parametrizable combined cost function stacking strategy (CCFS), which is based on the ideas of the category, the retrieval time and the positional stacking concepts. In their work, category stacking of export containers is preferred, and so it does not provides a sensibility analysis on the RMG performance with respect to the stacking algorithms.

As in Kemme (2012), the present work also deals with the study of the ASC block dimensions in relationship with the cranes performance. In this case, the analysis of the block productivity does

not take into consideration the waiting times of ETs and SCs. Instead, performance is measured though indicators that characterize the ASC productivity directly. Besides, the efficiency of the operations is measured as a function of the energy consumption of the ASCs.

The reminder of this chapter is organized as follows. Section 5.3 introduces the problem, section 5.4 describes the experimental setup; later, section 5.5 presents the results from the simulation and provides a discussion, and finally section 5.6 brings together the main conclusions from this work.

5.3 Block size optimization

The goal of the present study is to evaluate a strategic decision, the layout of the yard block, in the long-run performance of ASC storage yards of an automated container terminal. Although the influence of the stacking height is well known and has been subject of exhaustive evaluation in the literature, the performance of a twin ASC system is also highly influenced by the floor plan dimensions. Comparing containers blocks of the same total capacity, a block of greater length (and therefore, smaller number of stacks per bay) will experience longer gantry distances for the cranes (Figure 52). Reduced crane productivity and efficiency may be expected as more time is required to execute the stacking or retrieval operations, and more energy is required to move the crane.

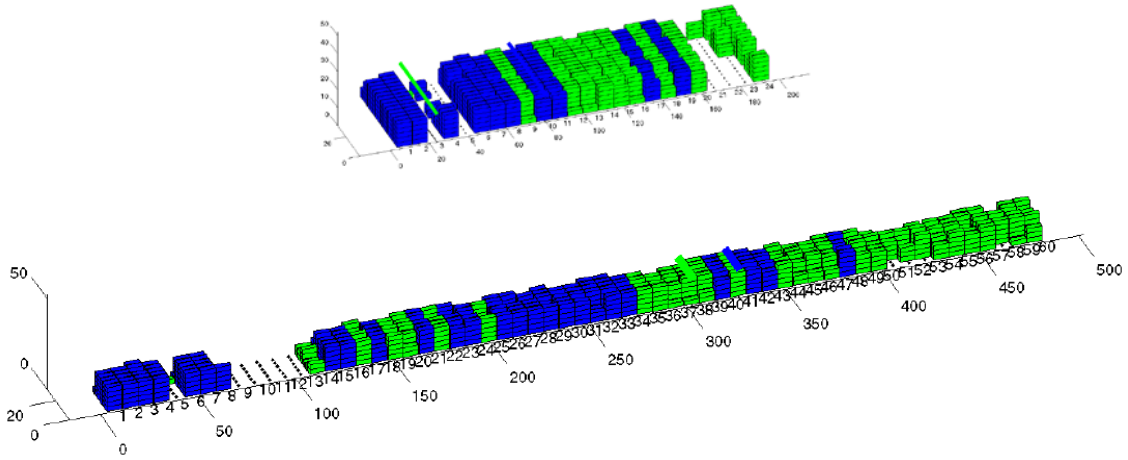


Figure 50. Example of two ASC blocks of different length and width: 24x15 (up) and 60x6 (down).

Increasing block length has also an effect on rehandling operations. As depicted in Figure 53, the gross capacity of a bay of stacks s and height h is $C^{gross} = s \bullet h$. However, not all the stacks can be fully occupied as a minimum amount of space (empty slots) is needed to relocate containers when buried at the bottom of a full pile. That minimum handling space is equal to h-1 slots, and thus independent of the number of stacks per bay, and the net capacity is $C^{net} = s \bullet h - (h - 1) = (s-1) \bullet h+1$. Now, considering two blocks of equivalent gross capacity but different lengths, as the number of stacks s decreases, C^{net}, and so the net block capacity diminishes. As a consequence,

for any given volume of containers in the block, the average stacking height will be higher for the longer block, resulting in greater amounts of rehandling despite the shorter trolley distances for such operations.

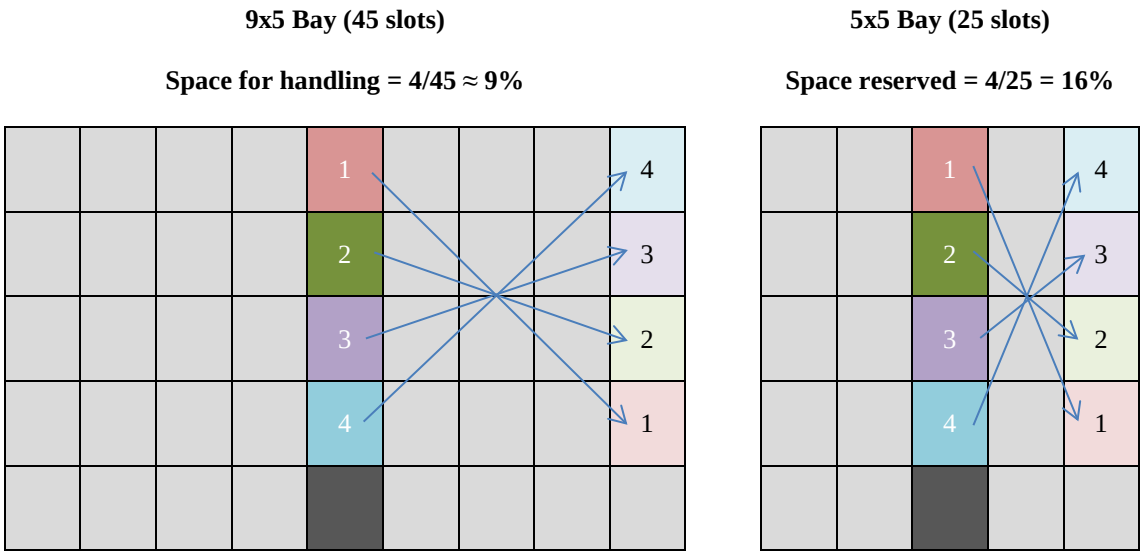


Figure 51. Minimum number of empty slots (white) required to relocate other containers (grey) on top of a container buried in the bottom of a full pile (dark grey). Light gray indicate containers present in the bay.

Conversely, reducing block length will result in greater crane interference, mainly because a shorter block will force the two ASC to work closer to each other. The fact that more time will be required by the ASCs to perform rehandling operations may be counterbalanced by the fact that less rehandling operations are expected due to the smaller average stacking height.

In summary, there is an economic trade-off between reducing block length to minimize gantry travel and reduce rehandling incidence, and accept more crane interference and thus greater waiting time, which reduces the ASC throughput.

A new simulation study is conducted in this work by means of a discrete event simulation model that is capable of reproducing the multi-objective, stochastic, real-time environment of an ASC yard block at a multiple-berth facility.

5.4 Experimental setup

In this section, the simulation model that was presented in Section 4.4 is customized to reproduce a freely scalable container yard block, along with the corresponding seaside and landside TPs, over a user-defined period of time. The length of the simulation period is set to 28 days.

5.4.1 Experimental cases

As the experimental setup is almost identical to that proposed in Chapter 4, we refer to that chapter for details on the simulation parameters, while the differences are described next.

First of all, block dimensions are variable in this case. While the stacking height is constant, the number of bays and stacks per bay is modified as to maintain a gross block capacity of 1,800 TEUs (see Table 10). The number of TPs varies accordingly to the number of stacks, as more space is available on both sides of the block.

Case	Length (20"slots)	Stacks (20"slots)	TP lanes
1	24	15	12
2	30	12	10
3	36	10	8
4	40	9	7
5	45	8	6
6	60	6	5
7	72	5	4

Table 9. Summary of experimental cases with respect to the block dimensions.

A second difference in the model setup is the energy model deployed in this analysis. In this case the Electric Consumption Model will be used in the calculations.

5.4.2 ASC modelization

Crane modelization and crane operational procedures used in the present work are the same as those explained in Section 5.4; otherwise they will be explicitly detailed in this chapter. ASC specifications utilized in the experiments are provided in Table 11. In addition to the dimensions and weight of the moving parts, crane motor characteristics are given in order to fulfill the parameters needed for the Electric Energy Consumption model. As indicated in the table, the movement is sensitive to the container weight, and therefore the crane will travel at a lower speed in laden condition. Under this assumption, speed will vary linearly between the maximum speed at empty condition and the minimum speed when laden at maximum capacity.

		Units	Gantry	Trolley	Hoist
Mass		kg	185,000	25,000	10,000
Acceleration		m/s^2	0.4	0.3	0.6
Speed	Laden	m/min	240	70	45
	Empty		270	70	90
Motor rpm	Laden		200	140	180
	Empty	Revs/min	225	140	360
Moment of inertia		$kg\ m^2$	2.0	8.0	46.0
Motor efficiency		-	0.95	0.9	0.85
Friction coefficient		-	0.005	0.006	1.0

Table 10. Summary of ASC specifications needed to feed the electrical model.

Another important difference in this case is that the Logic Stacking Algorithm used in this work introduces a restriction in the number of bays that can be assigned to import and export flows. This limit is imposed as the stacking algorithm by itself leads to situations in which the unequal share of space between import and export flows make difficult the stacking of new containers.

The limit is imposed so that the total number of bays devoted to import or export flows compared to the total number of bays is ranges between 40% - 60%.

5.5 Results and discussion

This section summarizes the main results obtained in the numerical experiments. Experiments are carried out using a single-core 2.7 gigahertz personal computer with 8 gigabytes of RAM. Each run consumes 6 hrs. typically. As in the previous chapter, optimal block dimensions are sought based on two main criteria. First, *efficiency* is examined by computing the average energy expenditure per container; second, *productivity* is evaluated by looking at the time needed by the ASCs to carry out the handling operations.

Regarding energy, results indicate that efficiency increases as blocks get shorter and wider. With respect to productivity, as block length increases a trade-off is observed between the travel times of the cranes along the block and the container delivery times. As a consequence, service levels show an optimum around 30-36 bays. Overall, for the parameters employed in the simulation, such interval of block length would be considered as the optimal solution. A comprehensive description of the results is provided in the next sections.

5.5.1 Incidence of rehandling

An analysis of the incidence of rehandling is provided here, as it will help explaining some of the results obtained later on. As is known, rehandling is considered an unproductive type of movement executed by the ASCs to retrieve a container buried in a pile. Two main sources of container rehandling are observed in the simulations: regular retrieval operations to place the container in the TP for final departure, and intermediate relocations due to housekeeping. Two important factors have a significant effect on the incidence of rehandling: the size of the inventory and the slenderness of the block. First, *inventory size* plays a double and opposite role with respect to rehandling: on the one hand, more containers increase the average stacking height, augmenting the number of rehandling operations; on the other hand, more containers reduce the amount time available for the ASCs to carry out housekeeping movements; hence the amount rehandling operations associated to such movements tends to be lower. Second, *block slenderness* also has a double effect on rehandling. Firstly, as indicated before, longer blocks have less net capacity and therefore higher stacks, increasing the incidence of rehandling. Secondly, considering that much of the rehandling is directly caused by housekeeping operations, longer blocks have longer ASC cycles, therefore leaving less time to carry out housekeeping operations; consequently container grouping is more likely to be preserved, reducing the amount of rehandling when retrieving the containers.

The effect of these trade-offs is observed in the somehow complex behavior seen in the results summarized in Table 12, which underscores the complexity of the housekeeping operations

depending on the amount of traffic and the type of flow. In the table, rehandling operations exhibit a maximum value at some intermediate value of the block slenderness, with the exception of the export containers under low occupancy of the block, for which the amount of relocation movements due to housekeeping reveals as the hegemonic effect.

Block slenderness (L/W)	1.60	2.50	3.60	4.44	5.63	10.00	14.40
Length (20" containers) L	24	30	36	40	45	60	72
Width (20" containers) W	15	12	10	9	8	6	5
Minimum handling space (%)	**5.3**	**6.7**	**8.0**	**8.9**	**10**	**13.3**	**16.0**
40% OCCUPANCY							
Import	1.36	1.45	1.57	**1.59**	1.48	1.45	1.42
Export	3.63	3.83	3.91	3.93	3.99	4.20	**4.24**
60% OCCUPANCY							
Import	1.39	1.42	**1.48**	1.41	1.43	1.34	-
Export	3.71	**3.95**	3.90	3.88	3.71	3.13	-

Table 11. Average number of rehandling operations per container with respect the block dimensions.

A summary of housekeeping operations is provided in Table 13. To this matter, the amount of time that the ASC cranes can dedicate to housekeeping (t_{HK} hereinafter) and also the distribution of containers in the block play a very important role. As expected, housekeeping operations are mainly affected by several interrelated factors: block occupancy, amount of rehandling, and block length. Higher block occupancy increases the rate at which containers arrive to the TPs; as a consequence t_{HK} decreases because of greater ASCs workloads. With respect to the amount of rehandling, a similar effect is observed: greater incidence of rehandling necessarily reduces t_{HK}. Finally, larger block length increases gantry times while reducing trolley times in a smaller proportion, subsequent longer duty cycles times per container will also reduce t_{HK}.

Block slenderness	1.60	2.50	3.60	4.44	5.63	10.00	14.40
40% OCCUPANCY							
Import	0.28	0.26	0.24	0.22	0.20	0.19	0.08
Export	2.23	2.26	2.40	2.41	2.53	2.60	2.65
60% OCCUPANCY							
Import	0.00	0.00	0.00	0.00	0.00	0.00	-
Export	1.91	1.83	1.77	1.67	1.52	1.12	-

Table 12. Average number of housekeeping operations per container with respect to block dimensions.

From the results, the following conclusions can be drawn:

- As expected, the amount of export housekeeping operations is greater than import's; priority is always given to O/B containers over I/Bs' to speed up the vessel loading operations.
- As pointed out above, under high level of occupancy t_{HK} is smaller; as a consequence, more housekeeping operations per container are undertaken when the block inventory is smaller. When the block occupancy increases, priority is first given to housekeeping of export containers, and there is little or no time to carry out such operations for import containers.
- With respect to block length, less housekeeping is generally observed as the block length increases, with the exception of export containers under low block occupancy. This result underscores the complexity of housekeeping procedures and their influence on other processes.
- To this extent, it is also worth noticing that shorter blocks tend to produce a greater degree of segregation of import and export containers in the yard. As a consequence, the geometrical center containers of both types is located closer to the final TPs, which in turn reduces the travel times of ASCs during housekeeping.

5.5.2 Energy expenditure

Again, average energy consumption per container ($\overline{E_C}$) is the KPI used to evaluate the efficiency of the ASCs. As before, the container transit along the block typically comprises one stacking operation, a number of housekeeping operations, and one retrieval operation. In addition, the container may be relocated within the same bay in order to retrieve a container buried underneath.

Table 23 and Table 24 summarize the $\overline{E_C}$ with respect to the block occupancy levels. Several tendencies can be inferred from these results:

- Export containers require more energy than import containers due to the fact that export housekeeping has priority over import.
- Gantry movements consume as much energy as hoist movements. This result is relevant because optimization usually focuses on the avoidance of rehandling, which is responsible of the unproductive hoist movements, which are comparatively larger than productive hoist. Hence, at the light of these results, it may be reasonable to attempt to investigate the optimization of productive gantry. To the best of the author knowledge, no literature addresses this particular topic, although the work by Speer et al. (2011) provides indirect consideration to gantry travel as it is intimately related to the optimization of the duration of RMG cycle times. In addition, they do not include housekeeping operations in their analysis.

- With respect to hoist, the unproductive component exceeds the productive component, which indicates a significant incidence of rehandling. The block length has little influence on the amount of energy expenditure.
- Trolley movements are of little significance.
- As a general rule, less energy consumption is observed as blocks get shorter and wider. However, housekeeping operations can introduce a significant degree of distortion to this rule. As indicated before, the incidence of housekeeping depends on the block occupancy and block length, and consequently on the efficiency of the operations. I.e., fewer housekeeping events observed for the import operations under low block occupancy and export operations under high block occupancy produce a negative slope in the curve for larger blocks. To this extent, it is also worth noticing that, as no import housekeeping operations are observed under high block occupancy, the $\overline{E_C}$ also follows a positive slope, and the same happens for obvious reasons for export containers when the size of the inventory is small.

5.5.3 ASC performance

Table 16 summarizes ASC performance results from the experimental setup. As in section 5.5.2, in addition to the average time per ASC operation (stacking, retrieval and translation), two overall performance indicators are considered regarding the quality of service provided: *container exit time* (CET) for import containers and *vessel service time* (VST) for export containers.

The effect of block length on the ASC cycle duty times is twofold: whereas shorter blocks reduce gantry times, trolley and translation (unproductive) movements increase significantly. As a consequence of this trade-off, ASC operation times are not always minimal for the smallest block length considered in the experiments, which in turn has a reflect on the CET and VST as well.

Operation	Movement	Type	Block dimensions (L x W) in 20” slots						
			24x15	30x12	36x10	40x9	45x8	60x6	72x5
IMPORT CONTAINERS									
Stack	Gantry	P	3.6	3.9	4.1	4.2	4.4	4.7	5.0
		U	0.2	0.2	0.2	0.2	0.1	0.1	0.1
	Trolley	P	0.1	0.1	0.1	0.1	0.1	0.1	0.1
		U	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Hoist	P	1.8	1.8	1.8	1.8	1.8	1.8	1.8
		U	0.0	0.0	0.0	0.0	0.0	0.0	0.2
Delivery	Gantry	P	3.9	4.0	4.9	4.7	4.6	4.5	4.3
		U	0.2	0.2	0.2	0.2	0.2	0.1	0.1
	Trolley	P	0.1	0.1	0.1	0.1	0.1	0.1	0.1
		U	0.1	0.1	0.2	0.1	0.1	0.1	0.1
	Hoist	P	2.1	2.0	2.4	2.3	2.1	2.1	1.8
		U	2.5	2.5	2.9	2.6	2.5	2.4	2.2
Total IMP			**14.5**	**16.6**	**16.8**	**16.3**	**16.0**	**15.9**	**15.5**
EXPORT CONTAINERS									
Stack	Gantry	P	3.6	3.7	4.0	4.0	4.1	4.3	4.5
		U	0.3	0.3	0.3	0.2	0.2	0.1	0.1
	Trolley	P	0.1	0.1	0.1	0.1	0.1	0.1	0.1
		U	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Hoist	P	1.8	1.8	1.8	1.8	1.8	1.8	1.8
		U	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Delivery	Gantry	P	8.4	8.9	9.2	9.5	9.9	10.5	11.0
		U	0.3	0.3	0.4	0.4	0.4	0.4	0.4
	Trolley	P	0.3	0.3	0.2	0.2	0.2	0.2	0.2
		U	0.3	0.3	0.3	0.3	0.3	0.3	0.3
	Hoist	P	4.4	4.6	4.7	4.8	4.9	5.0	5.1
		U	5.6	6.1	6.2	6.1	6.3	6.4	6.6
Total EXP			**25.1**	**26.3**	**26.7**	**27.4**	**28.1**	**29.1**	**29.9**

Table 13. $\overline{EC}$ in kWh. 40% Occupancy.

Operation	Movement	Type	Block dimensions (L x W) in 20” slots					
			24x15	30x12	36x10	40x9	45x8	60x6
IMPORT CONTAINERS								
Stack	Gantry	P	3.8	4.0	4.0	4.2	4.3	4.7
		U	0.1	0.1	0.1	0.1	0.1	0.0
	Trolley	P	0.1	0.1	0.1	0.1	0.1	0.1
		U	0.0	0.0	0.0	0.0	0.0	0.0
	Hoist	P	1.8	1.8	1.8	1.8	1.8	1.8
		U	0.0	0.0	0.0	0.0	0.0	0.0
Delivery	Gantry	P	3.1	3.1	3.3	3.4	3.4	3.6
		U	0.0	0.0	0.1	0.1	0.0	0.1
	Trolley	P	0.1	0.1	0.1	0.1	0.1	0.1
		U	0.1	0.1	0.1	0.1	0.1	0.1
	Hoist	P	1.5	1.5	1.5	1.5	1.5	1.5
		U	2.1	2.1	2.1	2.2	2.1	2.1
Total IMP			**12.8**	**13.0**	**13.2**	**13.4**	**13.4**	**14.0**
EXPORT CONTAINERS								
Stack	Gantry	P	3.4	3.5	3.6	3.6	3.7	4.0
		U	0.0	0.0	0.0	0.0	0.0	0.0
	Trolley	P	0.1	0.1	0.1	0.1	0.1	0.1
		U	0.0	0.0	0.0	0.0	0.0	0.0
	Hoist	P	1.8	1.8	1.8	1.8	1.8	1.7
		U	0.0	0.0	0.0	0.0	0.0	0.0
Delivery	Gantry	P	7.6	7.8	8.2	8.4	8.1	7.3
		U	0.2	0.2	0.3	0.3	0.2	0.2
	Trolley	P	0.2	0.2	0.2	0.2	0.2	0.1
		U	0.4	0.4	0.3	0.3	0.3	0.2
	Hoist	P	3.8	3.7	3.8	3.9	3.7	3.1
		U	5.9	5.9	5.8	5.9	5.8	5.0
Total EXP			**23.4**	**23.6**	**24.0**	**24.4**	**23.8**	**22.6**

Table 14. $\overline{EC}$ in kWh. 60% Occupancy.

Flow	Movement	Block dimensions (L x W) in 20" slots						
		24x15	30x12	36x10	40x9	45x8	60x6	72x5
40 % OCCUPANCY								
Import	Stack	130.0	154.1	156.9	160.0	179.4	223.1	234.7
	Delivery	210.6	**191.4**	209.4	227.2	226.9	234.7	253.1
	Translation	9.0	8.2	6.2	9.9	6.2	4.2	6.1
Export	Stack	148.8	135.9	148.9	159.2	163.8	190.7	192.3
	Delivery	497.9	496.8	**487.0**	492.7	506.3	517.4	541.8
	Translation	11.1	9.7	11.2	12.0	10.7	12.4	11.6
CET (min)		12.0	8.7	**7.9**	8.3	9.1	13.1	17.5
VST (hrs)		7.5	7.3	**6.7**	6.8	6.9	7.0	7.2
60 % OCCUPANCY								
Import	Stack	152.5	157.3	162.7	171.1	180.9	181.7	-
	Delivery	211.0	**196.1**	212.8	228.2	232.2	256.8	-
	Translation	2.8	2.6	3.1	2.7	2.5	3.3	-
Export	Stack	120.9	113.4	122.7	126.7	135.9	144.0	-
	Delivery	479.2	478.3	**474.3**	477.7	480.5	483.5	-
	Translation	4.1	4.7	4.9	5.9	5.3	4.7	-
CET (min)		14.3	**12.5**	13.6	15.2	20.8	32.6	-
VST (hrs)		9.8	9.0	**8.6**	9.2	9.4	10.9	-

Table 15. Average time per operation in seconds. Minimum values related to delivery operations and block quality of service are highlighted.

From the results, a number of conclusions can be drawn:

- With respect to block occupancy level, higher traffics lead to longer CET and VST, as expected.
- With respect to ASC operation times, stacking is very sensitive to housekeeping, hence longer stacking times per container are observed under low volumes of inventory. Retrievals entail slight longer times for the low occupancy scenarios, mainly due to hoisting. The incidence of rehandling plays an important role to this matter.
- Small differences in ASC retrieval times indicate that the distribution of export containers along the block also plays an important role on the time required to complete the vessel upload operations.

5.6 Conclusions

This work analyzes the effect of block dimensions in a perpendicular terminal with the focus on efficiency and productivity of the twin ASCs. From the discussion, several conclusions can be drawn:

- Both energy expenditure and productivity are sensitive to block occupancy. While energy expenditure due to gantry movements is proportional to the block length, hoist associated consumption is dominated by the incidence of housekeeping operations, which introduces

a significant amount of distortion in the results. With respect to productivity, shorter blocks reduce gantry times and thus the overall duration of the duty cycle; on the other hand, shorter blocks increase the frequency of crane interference, and also the duration of trolley movements required to retrieve containers. For the particular settings of the experiments, 30-36 bay-long block is optimal from the productivity point of view, providing the best results for ASC cycle duration, CET and VST. These results seem coherent with those obtained by Kemme (2012), who indicates the productivity of the block increases as the number of stacks increases. However, the maximum block width in his work is 12 slots, and the SC and ET waiting times do not reveal a trade-off with respect to this parameter.

- The behavior of the ASC block is complex for many reasons. In addition to simultaneous yard processes, results are also sensitive to the experimental setup: crane specifications (i.e. weight, motor rpm, hoist speed, or acceleration), friction coefficients, etc. Changes in input variables may lead to different results in terms of both efficiency and productivity. As the number of tunable parameters of the model is quite large, sensibility analyses are too extensive for the scope of this Thesis. In this context, although results are coherent with the literature, no general statements about the optimal values of the block are given; tailored models reveal as the most convenient solutions to study each particular case.
- In the same line, the stacking algorithm plays an important role in the distribution of containers in the block, and therefore the optimization of the block layout must take into account the operational procedures, which are usually not defined at the time of design.
- As observed in reality, the amount of housekeeping plays an important role on the overall performance of the block; future research may focus on imaginative solutions to further optimize the efficiency and productivity of the block housekeeping operations with the help of heuristic rules or even optimization methods when possible. Such techniques may be of help reducing the aleatory search of containers in the block, or to find a global solution for a subset of containers (i.e. those in the ASCs workload list) that represents a local or absolute optimum of the cost function.
- The somehow complex system behavior supports the fact that energy consumption depends on a multitude of factors in constant interaction (feedback) with each other; therefore, besides the general tendencies inferred from this study, the optimization of energy consumption in a more realistic scenario will require a tailored analysis considering the specific characteristics of the ASCs and the composition of container traffic (weight distribution, relative amount of import/export units, etc.).

Chapter 6

Allocation strategies for export containers as a function of space reservation time in parallel terminals

6.1 Introduction

As indicated in previous chapters, container handling activities in a terminal happen in a sequence. Each terminal subsystem in the transport chain can be characterized by the size of its buffer (capacity) and the rate at which containers are interchanged (throughput). To this respect, it is important to remark that every link in the chain depends on the others; as a consequence, the performance of any given link has a downstream effect (i.e. a crane delay is likely to produce congestion in the TP), but also an upstream effect (increase waiting time of trucks delivering containers to the terminal).

When considering the yard of a parallel terminal itself, it entails several of such links in the transport chain, and so its storage capacity (volume of import/export containers) and throughput (number of containers transiting through the quays and the terminal gates) depend on a large number of parameters: QC throughput, container dwell time, vessel inter-arrival time, etc. Consequently, yard performance is conditioned by the aggregate performance of all these subsystems determines and, correspondingly, yard management strategies have an effect besides the inventory's size and fluctuations (peaks) over time: the overall terminal performance.

Space reservation in storage yards for containers bound to a vessel is a common exercise among terminal operators to ensure efficient yard management. *Reservation* can be considered as a key

practice when dealing with the so called Block Allocation Problem (BAP), for which the operator needs to calculate first both the amount of space needed and the location of bays for containers in advance to the vessel arrival. The objective of this approach is threefold: balance the workload among YCs, minimize road congestion within the terminal, and minimize ET and IT delays while interchanging containers with the YCs.

6.1.1 Complexity of the BAP

The BAP is a multifactorial problem for which feasible solutions must consider not only problem constraints (terminal configuration, availability of handling equipment), but also container traffic in the terminal, which is the primary driving forcing of the logistic system.

The complex nature of traffic plays a fundamental role on the development of terminal operations. That complexity can be summarized in a number of factors listed next:

a) Flow type

Depending on the destination, three types of flows coexist simultaneously: *export* (EXP or outbound), *import* (IMP or inbound), and *transshipment* (although the latest can be assimilated to outbound containers); the stacking procedures differ in each case,

b) Nature of the cargo

Containers can be of different sizes (20”, 40”, 45”, etc.), type (reefer, dangerous goods, etc.), and so the relative share of each type may also change greatly the way each terminal is operated.

c) Uncertainty of traffic

The uncertain nature of traffic is double: not only volumes are variable, but also arrivals and departures of inbound and outbound containers (with a dissimilar degree of the uncertainty). On the sea side, vessels arrive on a different time basis, and they load and discharge different volumes of containers every time; in addition, loading plans are not always met, or last hour changes are frequent. On the land side, External Trucks, arrivals vary on an hourly, daily, monthly, and seasonally basis, and appointments are not always made, so their characterization is far from being deterministic.

The consequences of traffic variability on space reservation are readily apparent: both the size of the space reserved and the period of time for which space is reserved are variable. Therefore, terminal operators must decide whether reserve space for just few hours or for several days. For export operations, after the vessel notification of arrival (T_N) to the port, space may be reserved in the yard at any time (T_R). Outbound containers arrive to the terminal in a stochastic manner for several days in advance to the actual vessel arrival (T_A), typically 3 to 5days on average (Zhang et al., 2003), until the so-called cut-off time (usually coincident with the gate closure of the day

before departure, T_G). The opposite pattern applies for import containers; although the arrival date is known, the time of delivery is hardly foreseeable, and so containers to be retrieved early are often stacked underneath other containers, generating rehandling movements. In addition, dwell time of import containers (5-7 days on average) is usually longer than that of export containers.

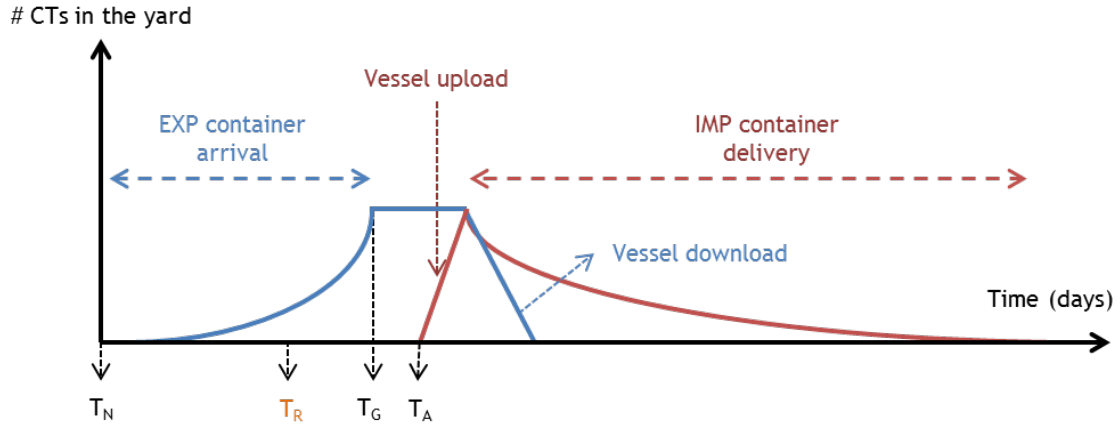


Figure 52. Scheme of the yard inventory associated to a vessel considering export (blue) and import (red) containers.

The distribution of containers in the yard and the total yard occupancy at any given instant results from all the above-mentioned factors. In turn, container distribution will determine the efficiency and productivity of the MHE (i.e. a greater number of relocations due to the higher average stacking height according to Kim, 1997), associated operational costs, etc. A review of the usual rules for distributing containers over the storage yard is given next.

6.1.2 Allocation principles

As indicated in Section 2.3.1, in general container yards are managed according to several widespread principles to ensure future containers will be placed in optimal positions in the yard (Woo and Kim, 2010). In the long run, along with the characteristics of the traffic, these principles will determine the distribution of containers, which can be characterized by the number of clusters and their size in terms of bays and number of containers per cluster.

Ideally, containers bound to a ship are allocated in bays as close as possible to its berth (NL principle). This sole practice enforces compliance with both the CL and NL principles at the same time. However, under congested situations, concentration may not be always possible. If yard occupation is high, the amount of space available may be smaller than the number of containers discharged by the ship, and thus reservation will be possible for a limited number of containers.

On the other hand, when space for a vessel is reserved, new containers arriving to the terminal are prevented from using that space for a significant amount of time (≈ O(days)), which may be

seen as wasteful. As a consequence, terminal capacity becomes "smaller" and current operations are somehow penalized. In the end, there is a tradeoff between reserving space to make future operations more efficient at the expense of punishing ongoing operations; conversely, when no space is reserved, the layout is less optimal, but the terminal capacity increases.

6.1.3 Objective

The objective of this chapter is to characterize the distribution of containers in the yard layout of a parallel terminal resulting from the use of a stacking strategy that makes use of space reservation, which is a well investigated practice in the literature under certain scenarios. However, little is known about the consequences of deploying such strategy in the distribution of containers in the yard. This study also introduces, for the sake of comparison, one online stacking strategy .Although online stacking is a common practice, to the best of the author knowledge literature has only addressed problems in which departure information is known, and so space could be reserved for whatever the length of the "planning horizon". In such cases, the allocation problem can be solved to near optimality, including the YC scheduling, and regardless the complexity of the container traffic within the considered planning periods. In addition to online (no reservation) and offline (reservation) stacking strategies, the study proposes the use of *mixed* strategies that make use of both online and offline stacking by modifying the value assigned to T_R. The strategies are analyzed so as to determine the productivity of YCs and their efficiency in terms of electric consumption.

One additional contribution of this work is the consideration of both types of traffic at the same time. In contrast to other studies that focus on pure transshipment terminals, in which the arrival of containers by vessel, this work takes into consideration the continuous arrival of External Trucks (ETs) for both import and export flows, introducing an uncertainty that arises from the truck arrival and departure times. This assumption poses a new constraint to the utilization of yard space and, thus, on the resulting yard template. In summary, the simulation setup of the present analysis, although common for many container terminals of medium size, and the problem approach, have not been dealt with in the literature.

The remainder of the chapter is organized as follows: Section 6.2.1 presents a literature review on works related to reservation strategies and allocation strategies, Section 6.2.2 introduces the proposal of reservation strategies and operating procedures utilized of this work. Section 6.3 describes the numerical setup for the DES model, as well as the traffic modelization and generation of traffic inputs. Section 6.4 presents the experimental setup, and Section 6.5 provides a summary of results of the numerical cases. Lastly, conclusions are drawn in Section 6.6.

6.2 Allocation strategies

6.2.1 Overview

As indicated in the literature review, many authors decompose the allocation problem into a two-stage problem (Kim and Park, 2003). The first stage is referred to as the *Block Allocation*, seeking the determination of both the number of bays (or stacks) and their exact location within the terminal yard. The second stage is the so-called Slot Allocation, in which the exact slot in a sub-block for a container is found.

Regarding the BAP, Steenken et al. (2004) distinguished between two different strategies to allocate containers bound to a vessel that differ on whether space is reserved before the specific ship's arrival. First, *Storage Planning* a reserves areas of the yard, with containers are grouped according to their port of destination (POD) and size. Heavier containers are piled on top of lighter ones assuming they are loaded earlier to ensure ship stability. Conversely, other terminals make use of *Scattered Planning*, an online procedure that requires no space reservation. Upon arrival of a new container, the system selects the berthing place of the ship from the ships schedule and automatically searches for a good stack location within the area assigned to the berth. The position of an arriving container in the yard is determined in real time, stacking the container on top of other containers of the same category. Scattered planning leads not only to greater container scattering over the yard, but also to larger ground occupation, which reduces the number of reshuffles.

Zhang et al. (2003) developed a rolling-horizon approach in order to solve the BAP for import and export containers in a RTGCs-operated parallel terminal of 10 blocks with 6-stack, 5-tier bays. A mathematical programming model is proposed to solve each 3 day long planning stage: the first stage seeks to determine the number of containers to be placed in each storage block by balancing the workloads among blocks, and the second stage aims to determine the number of containers associated with each vessel that constitutes the total number of containers in each block. The numerical simulations show the workload imbalance in the yard is reduced, hence helping avoid possible bottlenecks in terminal operations.

More recent studies focus on the organization of the yard template for pure transshipment terminals considering reservation strategies where space is consigned in entire sub-blocks. This way, containers are stored according to their destination vessel, reducing the number of reshuffles and enhancing terminal productivity by using an optimization method to solve an objective function with multipurpose constraints. While Lee et al. (2006) minimize the number of yard

cranes to deploy; Jiang et al. (2012) propose a space-sharing yard template to reduce the under-utilization of space while ensuring the efficiency of yard operations at the planning level. Zhen (2014) developed a model for yard template planning considering a random number of containers to be downloaded and loaded onto vessels, which makes the number of sub-blocks bounded to each vessel uncertain.

With respect to the *Slot allocation*, Kim et al. (2000) analyzed the SAP by considering weight information. Dynamic programming[7] was used to solve the problem with the objective to minimize the number of rehandling movements that occur during the ship loading operations.

When no departure information is known, category stacking is used by Dekker et al. (2006) to pile containers of the same category on top of each other. If departure information is provided, then residence time strategy allows stacking a container on a pile if its departure time is earlier than that of all containers below.

Wan et al. (2009) also studied the allocation problem considering an entire sub-block composed of several bays. The static version of the problem is solved with an integer program formulation, and then they propose a heuristic method to reduce the computational time, with variants of the IP model embedded and run in the rolling horizon fashion. Then they consider a dynamic version of the problem, with containers being stacked and retrieved from the block, concluding that the heuristic approach is capable of reducing reshuffles within a reasonable computation time.

Summarizing, to the best of the author's knowledge, no literature investigates trade-offs existing between online vs offline procedures.

6.2.2 Reservation strategies

In this section, reservation strategies for export containers are proposed in order to evaluate the influence of T_R in the yard distribution of containers. Attention must be paid first to the flow of import and export containers bound to one vessel. Inbound and outbound containers do not only have different dwell times, but also different arrival and departure rates. For the case of outbound containers, the arrival rate during the *delivery period* (Figure 50) can be adjusted to an exponential

[7] (Wikipedia) Dynamic programming (also known as dynamic optimization) is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions. The next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time at the expense of a (hopefully) modest expenditure in computer storage space.

function (Zhang et al., 2003). According to this, fewer containers will arrive at the terminal at the beginning of the delivery period compared to those at the end. As a consequence, fewer clusters may be obtained when reserving yard space in early stages of the delivery period, but a greater amount of empty space will remain untapped. As less space will be available in the terminal, more rehandling may be experienced in the remainder of yard operations. On the other hand, when T_R approaches the ship arrival time T_A, more dispersion in the yard clusters may be expected, hence incurring in different YC travel distances to retrieve the containers and greater rehandling effort. Hence, three main types of strategies are proposed depending on the time at which space reservation is made with respect to the delivery period or, equivalently, the number of containers stacked in an online or offline mode: (1) *Early Reservation*, when space for a vessel is reserved prior to the arrival of the first export container and thus all containers are stacked in an offline mode; (2) *No Reservation*, when no space is reserved for the ship and thus all the containers are stacked online; and finally (3) *Intermediate Reservation*, when some containers are stacked online, and after some time space is reserved for the ship and thus the remainder of containers are stacked offline.

All the reservation strategies used in the analysis observe the following planning principles commonly found in the literature:

- *Nearest (berth) Location Principle* (NL) prioritizes the bays closer to the target vessel berth, thus minimizing the travel time of Yard Trucks (YTs).
- *Concentrated (bays) Location Principle* (CL) enforces containers belonging to a group to be placed on bays or stacks located as near as possible, in order to decrease the gantry travel of YCs during the ship loading operation.
- *Least (YC) Congestion Principle* (LC) favors the dispersion the groups of containers among the blocks, since excessive concentration may lead to interference among yard cranes during ship loading. Thus, the number of blocks among which the work is distributed depends not only on the existing yard layout, but also on the number and type of available handling equipment.
- *Least Relocation (rehandling) Principle* (LR) prevents containers belonging to different groups from mixing in the same stack, thus avoiding reshuffle movements needed to retrieve containers.

Although the model makes use of these principles, their application is not always straight away (some principles may interfere with each other), and sometimes it depends on the type of flow (import/export). I.e., when space is reserved early (equivalently, small T_R), the NL principle is observed by selecting bays or stacks for export containers as near to the target berth as possible, which also enhances bay concentration (CL principle). However, if yard occupation is high,

vacant locations may be widely spread over the terminal, and so selecting bays near to the berth (NL) may be contrary to CL principle if fewer but greater vacancies are available at further distances.

With respect to the LC principle, the model takes into consideration the workload of RMG cranes to evenly distribute the containers in the yard. For export containers, however, at the time of reserving space, the future YC workload upon arrival of a new export container it is hardly foreseeable. Hence the LC will only be considered when stacking import containers. In this case, when the number of containers assigned to a YC exceeds that threshold (25 containers per hour), the systems considers that the cluster is overloading that yard crane, and then searches for a new cluster that can be attended by a different yard crane. In some cases, however, it may not be possible to find a better candidate, and then the system will produce an overload. Finally, the LR principle is observed when dealing with the Slot Allocation Problem. In this case, export containers are stacked by taking into consideration its weight, as described in Section 3.3.2.2.

The case of import containers differs from the export counterpart. First, the system will reserve space for import containers upon vessel arrival. Import containers do not need to be categorized in groups according to port destination or weight, and therefore the CL does not apply to this case. Although stacking import containers also obeys the NL and LC principles, under the assumption that no departure information is known at the time of stacking, the system can only enforce containers from the same vessel to be stacked close to each other, and take enforce newer containers not to be stacked on top of older containers.

6.2.3 Reservation and Stacking algorithms

As previously indicated, three Reservation Strategies are tested in the model for export stacking: *Early Reservation*, *No Reservation*, and *Intermediate Reservation*. Since the delivery period is set to three days in the simulations, *Intermediate Reservation* will be tested by reserving the space at the beginning of the second and the third day. In such scenarios, the yard space needed for a ship will calculated as the outbound containers in the stowage plan minus the number of containers that have already arrived at the terminal.

The space allocation policy is based on entire bays (not stacks). The priority is given first to bays with containers belonging to the same group, second, to empty bays, and third, stacks within bays assigned to different groups. Note that therefore, bay mixing is allowed in high occupation situations, although mixing is not permitted to containers bound to different vessels in the same stack.

6.3 Experimental setup

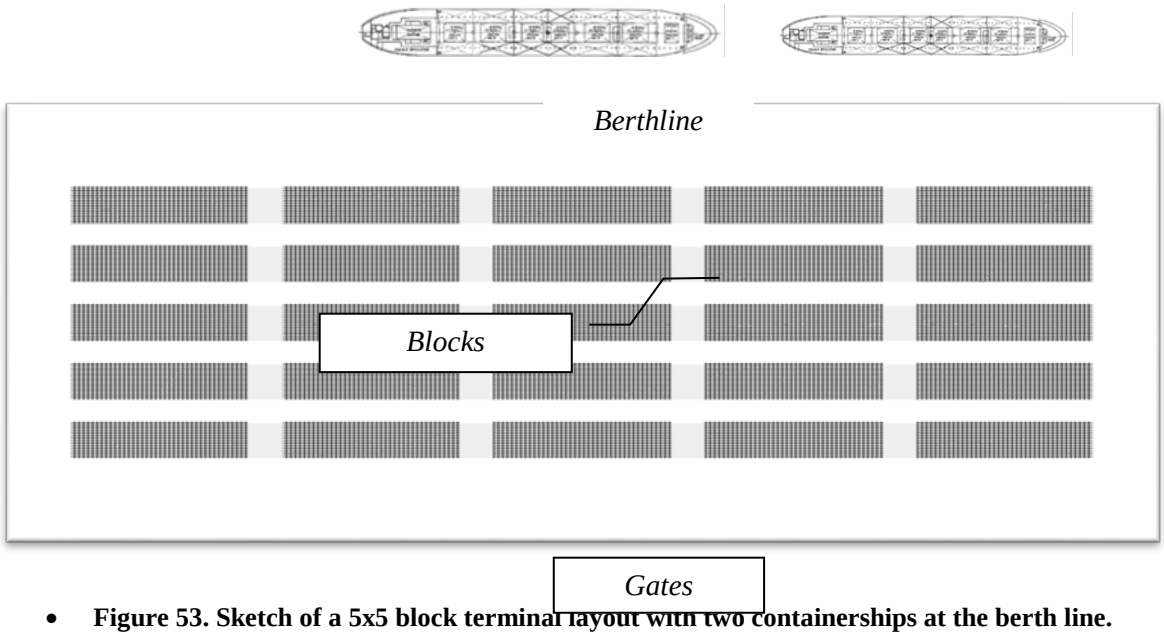
Reservation strategies proposed in the previous section are tested by means of numerical simulations conducted on a parallel terminal model described in Section 3.3. The simulation reproduces the distribution of container groups over the yard according to the stacking strategies and operational rules defined in previous sections, which in turn determine the operational costs associated to the yard. The simulation also allows evaluating the trade-off between the efficient use of space under each strategy and the operational costs associated to YC productive and unproductive movements.

Specific characteristics of the model setup are described in subsequent sections. A flow diagram of the container operations is given in Figure 52.

6.3.1 Terminal layout

The layout of the container terminal analyzed in this work (see Figure 51) is of rectangular shape, with a quay length of approximately 1350m and a width of 360m. Blocks are distributed in five rows and five columns, and they are composed of standard 20feet slots. A summary of block module characteristics is given next:

- Typical length: 32 TEUs (20-foot containers)
- Width: 6 stacks
- Height: 4 tiers
- Span: 25.25 m for 6 container rows
- Typical container spacing: 500 mm end-to-end, 400 mm side-to-side



- **Figure 53. Sketch of a 5x5 block terminal layout with two containerships at the berth line.**

As for the terminal layout itself,

- Number of berths: 3 (centers at thirds of the quay length)
- Aisles:
    - Vertical aisles:
        - Terminal sides: 60 m
        - Intermediate aisles: 50 m
    - Horizontal aisles:
        - Top: 80 m
        - Intermediate: 15 m
        - Bottom: 35 m
- Gates are located in the center of the landward side of the yard.

6.3.2 Container retrieval procedures

Whenever a container is about to be retrieved, one YC needs to be selected is selected according to the two following criteria:

- YC workload measured as the number of containers in the look ahead horizon WL_i
- Distance from the target container (C^T) and the container that occupies the last position of crane workload (C^{WL}_{yc}), denoted as $d_i(C^T, C^{WL}_{yc})$.

Before evaluating the candidate YCs, both criteria are nondimensionalized and the scoring formula is applied according to the next equation:

$$\boldsymbol{Q_{yc} = \frac{WL_{yc}}{max(WL_{yc})} + \frac{d_{yc}(C^T, C^{WL}_{yc})}{max(d_{yc})}} \qquad \textbf{(Equation 36)}$$

In the equation, YCs are overloaded when they surpass a threshold of 25 containers per hour, and so $\max(WL_{YC}) = 25$. The YC with the minimum score will add the target container to the last position of its workload.

6.3.2.1 Export containers

After the vessel download operation has ended, export containers bound to that ship are removed sequentially from the yard. When a YC is recovering containers from a bay, retrieval is accomplished in descending order of weight class (from 9 to 1), and so containers from the sea (left) side of the bay are retrieved first (Figure 51). Rehandling movements will be needed if a container of superior weight is piled underneath lighter ones.

6.3.2.2 Import containers

On the other hand, upon arrival of an import ET, one inbound container will be randomly retrieved. The probability of a container being picked up is directly proportional to the duration

of the container's stay in the yard. Therefore, the longer the duration of the stay, the higher the container's probability of departure will be.

6.3.3 Container traffic

Traffic is generated based on the total volume of containers in the year. The annual number of container is set to 1.2, 1.6, and 1.8 million TEUS depending on the experiment. This variability is used to analyze the effect of traffic on yard operations. Import and export containers are bound to a determined ship, meaning that the arrival and departure of containers over time is triggered by vessel arrival events.

6.3.4 Vessel arrivals

Vessels operate on a regular schedule characterized by an inter-arrival time (T_{IA}) that is generated according to a Poisson distribution. The mean T_{IA} results from dividing the annual volume of traffic of each scenario by the number of containers bound to the vessel for the import and export operations, which is set to an average of 1000 (with a variation of ±10%) for each ship. Upon arrival, vessels occupy one of the three berthing positions of the quay alternatively, therefore shifting the center of gravity of the container distribution in the yard for the containers bound to that vessel.

6.3.5 External trucks arrivals

External truck arrivals to the terminal take place the days before the vessel arrival event. Trucks bringing export containers follow an exponential distribution during the so called *delivery period*. The *delivery period* starts several days before the ship arrival (a value of 3 days is used in the simulation) and extends until desired by the terminal operator, at the so called *cut-off time*, T_c, set to the gate closure time on the day prior to the vessel arrival.

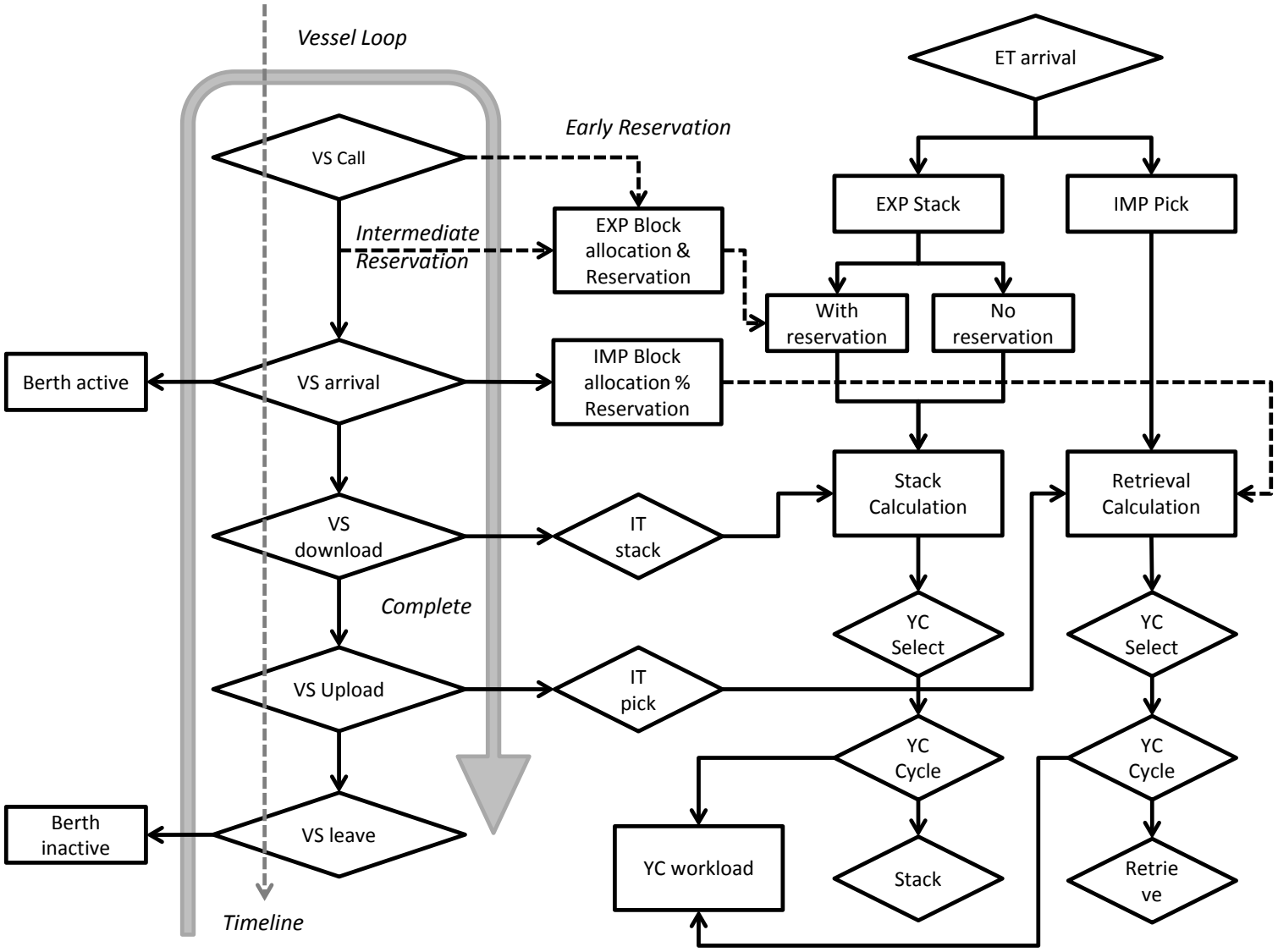


Figure 54. Flow diagram of the Parallel Terminal model.

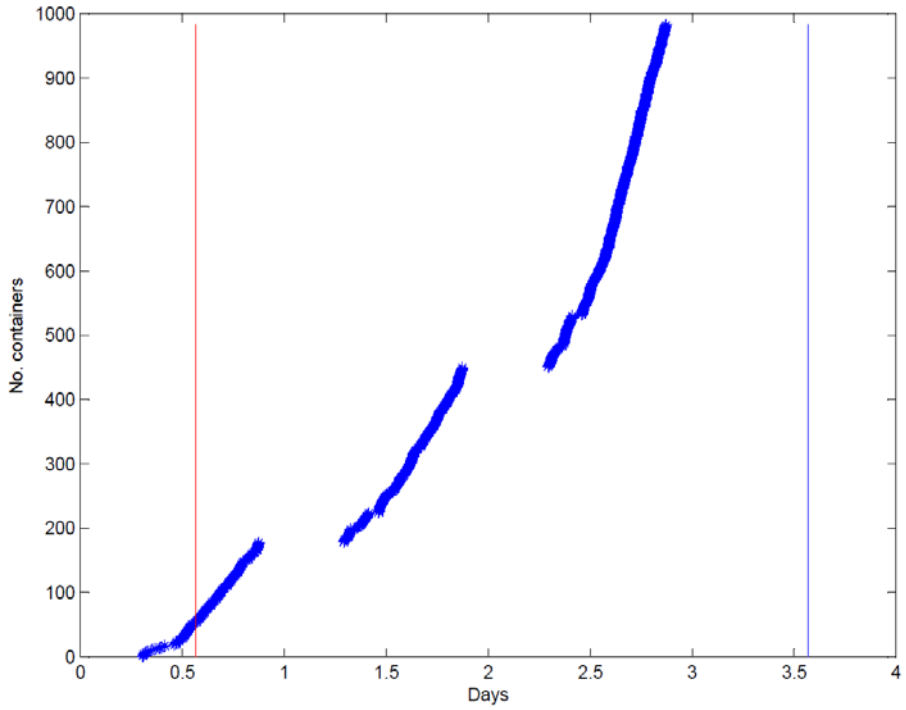


Figure 55. ET arrivals with respect to time. Vertical lines indicate vessel arrival time (blue) and vessel call to port (red).

On the other hand, inbound operations commence with the ship downloading operations, after which external trucks retrieve containers from the terminal following a decaying exponential arrival pattern that usually lasts longer than the delivery period (a limit value of 9 days is used in the experiments). The short term distribution of ET arrival is obtained from real data and stretches over the open gate period from 7:00 to 21:00 hours, observing two traffic peaks around 11:00 and 16:00 hours.

6.4 Numerical experiments

The total number of experimental cases arises from the combination of values adopted by two variables: traffic volume and reservation strategy. Whereas traffic volumes can be “low”, “medium” or “high”, the so called reservation time T_R adopt values of entire days, and cover uniformly the *delivery period*, which is set to 3 days. Therefore, reservation times T_R range ranging from 0 to 3 days: Early Reservation (T_R = 3, and so space is reserved before the arrival any EXP containers to the terminal), Intermediate Reservation (T_R = 2 and 1 days), and No Reservation (T_R =0).

As summarized in Table 17, the combination of inputs for simulation results in twelve (12) basic scenarios. Each simulation case is repeated five and average results of the simulations provided.

The duration of the simulation period is two weeks. The size of the inventory stabilizes after one week, after which results are collected for evaluation. Simulations are run on a CPU @ 2.7GHz and 8GB RAM, typically lasting about 4-8 hrs.

Reservation Strategy	Traffic volume (million TEUS /year)		
Early reservation	1.2	1.6	1.8
Intermediate (1 day delay)	1	5	9
Intermediate (2 days delay)	2	6	10
No reservation	3	7	11
Early reservation	4	8	12

Table 16. Experimental cases setup.

6.5 Results and discussion

This section summarizes the main results obtained in the numerical experiments. Experiments are carried out using a single-core 2.7 gigahertz personal computer with 8 gigabytes of RAM. Each run typically consumes 4-6 hrs.

The proposed stacking strategy with different reservation times is evaluated with regard to the distribution of containers in the yard, and also with respect to the *efficiency* and *productivity* of the MHE.

In general, results indicate that reservation strategies improve the distribution of containers in the terminal yard compared to online strategies, as the layout exhibits a greater degree of clusterization. As a consequence, the efficiency in the use of energy is improved. With respect to productivity, YCs also benefit from higher degrees of clusterization as the overload time is reduced.

A summary of the results and an in-depth analysis is provided in the next sections.

6.5.1 Distribution of containers in the layout

The performance of the algorithms in terms of yard space utilization is shown in Table 20. It is noticeable that under Early Reservation (ER) the terminal has not enough capacity to reserve space when the annual traffic is equal or exceeds 2.0 million TEUs, and so the traffic ranges from 1.2 million TEUs to 1.8 million TEUs.

Results confirm that the distribution of containers in the yard greatly depends on the moment at which the reservation is made (T_R). A clear tendency reveals that, as T_R is deferred, the amount of container spreading (measured as the number of groups of containers bound to the vessels) increases. As more clusters of bays are needed when delaying the reservation, the size of the average cluster (considering both the number of bays or containers in the group) tends to be smaller.

The volume of traffic in the terminal also has a significant influence on the amount container clustering in the yard template. Greater traffic not only introduces larger numbers of containers in the yard, but also increases the number of vessels to which the clusters are bound to. This way, when the traffic increases from 1.2 M TEUs to 1.8 M TEUs, the number of clusters increases roughly between ≈24% for the NR strategy to ≈57% in the ER strategy.

With respect to the amount of space blocked for export containers (relative to the gross terminal capacity) results indicate that it is very significant when utilizing Early Reservation, up to 16% in the 1.8M TEUs scenario. To this extent, it is worth noticing that, as the volume of export and import containers is the same in the experiments, the amount of inventory devoted to export containers must be approximately 50%, and so the net capacity of the inventory for export containers is 34%. In addition, a minimum of 12,5% of the space must remain empty to allow rehandling. On the other hand, $\overline{\text{RS}}$ decreases as T_R increases and more online stacking is performed by the algorithms, until the amount of space reserved is null under the No Reservation strategy.

Reservation Strategy	$\overline{TU}$ (%)	$\overline{RS}$ (%)	$\overline{NC}$	$\overline{NB}$	$\overline{\text{NCTs}}$
1.2 Million TEUS					
Early	46.0	10.0	8.04	3.91	23.92
Intermediate (1 day delay)		5.6	9.92	2.02	12.77
Intermediate (2 days delay)		1.7	12.77	1.43	10.11
No reservation		0.0	14.42	1.22	9.13
1.6 million TEUS					
Early	54.4	14.2	10.81	2.70	22.02
Intermediate (1 day delay)		5.7	13.97	1.72	13.37
Intermediate (2 days delay)		1.9	16.66	1.48	9.97
No reservation		0.0	17.57	1.34	8.77
1.8 million TEUS					
Early	69.3	16.5	12.63	2.05	15.31
Intermediate (1 day delay)		5.8	15.53	1.57	10.32
Intermediate (2 days delay)		2.1	17.63	1.44	9.13
No reservation		0.0	17.92	1.20	8.48

Table 17. Clustering of containers as a function of the volume of traffic. $\overline{\text{TU}}$: Average Terminal Utilization, $\overline{\text{RS}}$: Reserved Space for export containers relative to the Terminal Gross Capacity, $\overline{\text{NC}}$: Number of Clusters, $\overline{\text{NB}}$: Number of Bays per cluster, $\overline{\text{NCTs}}$: Number of Containers per cluster.

The amount of space reserved under ER is similar to that from Taleb-Ibrahimi et al. (1993), although consideration must be given to the fact that the total amount of spaced blocked for reservation ultimately depends on the combination of parameters that characterize the distribution functions of the ET arrivals and the vessel inter-arrival times (Saurí and Martín, 2011).

With respect to the operational costs, Table 18 shows the average distances travelled by the YTs from the berths and the block TLs for the vessel discharge and unloading operations. Several conclusions are readily evident:

- As expected, greater distances are traveled by the YTs as traffic volume increases. As expected, the average distance per container increases as more containers in the yard force the occupation of bays located further away from the target vessel. Also greater distances are traveled for import operations in general, as export containers are arranged in clusters following the nearest location principle.
- A greater degree of clustering observed as TR is deferred produces longer travel distances for export containers; the opposite effect is observed for import (download) operations.
- Overall, deferring T_R produces higher travel distances, between ≈4% and ≈6% for the 1.8 M TEUs and 1.2 M TEUs respectively, which may also have an effect on the productivity of the yard.
- The cost associated to YTs can be easily estimated. Considering a YT consumption of around 30l/100km and a fuel cost of ≈ 1.0€/l for the diesel, and the average distances to carry out a container from the berth to the block and back in Table 18 (811.1m for the shortest and 1554.1m for the longest trip), the total cost per trip ranges from around 0.24 € and 0.47 €. This cost is of the same order of magnitude than that of the RMGs when handling a container, as analyzed later on.

Algorithm	**Traffic volume: million moves/year**					
	1.2		1.6		1.8	
Flow type	IMP	EXP	IMP	EXP	IMP	EXP
Early reservation	1,471.3	811.1	1.582,1	914.9	1,554.1	864.1
1 day delay	1,465.4	867.7	1.491,8	871.5	1,369.0	994.5
2 days delay	1,334.0	1,057.3	1,368.3	1,069.8	1,242.7	1,148.9
No reservation	1,246.0	1,171.9	1,338.2	1,258.0	1,204.5	1,306.0

Table 18. Operational costs associated with the Yard Truck (traveled distances, in meters).

Table 20 shows the experimental results relative to the YCs operational costs and productivity. A number of conclusions can be inferred from the table:

- The number of reshuffles per import or export container is almost the same regardless the volume of traffic, as the height of the stacks is approximately the same regardless the reservation strategy.
- Conversely, earlier TR produces a smaller number of reshuffles per export container. The reason is that as TR decreases, the number of candidate bays among which the individual slot allocation of the container can be made is larger, and the higher the probability to find a suitable slot according to the weight classification indicated in Figure 40. This availability ultimately reduces the number of reshuffles as the order in which export containers are retrieved depends on their weight.

- With respect to rehandling, online stacking (equivalently, NR) is less restrictive as it does not impose as many constraints to the stacking operation. In contrast, later reservation worsens the individual slot allocation problem as indicated by the greater the number of rehandling jobs incurred when retrieving export containers.
- In average, expenditure associated to gantry represents around ≈20% of the total consumed by the RTG, while hoisting accounts for ≈73% and trolley the remainder ≈7%. Although small variations around these percentages are obtained among the experimental cases, this outcome indicates that YCs require relatively less gantry travels to handle containers in parallel terminals than in perpendicular terminals. As a consequence, YCs in parallel layouts also spend less energy per container. On the other hand, the contribution of rehandling to the efficiency of the operations is much greater.
- The energy expenditure reveals that more efficient use of the YCs (under the FIFO rule for dispatching) is achieved as more container clustering is present in the yard. Greater efficiency is achieved for two main reasons: first, strategies that favor container grouping will experience fewer crane movements when retrieving containers from the stacks to be uploaded to the vessel; and second, less rehandling effort is required for export operations.

Reservation Strategy	Nº Reshuffles/ CT		$\overline{E_C}$ (kWh)		Overload (% of time)
	IMP	EXP	IMP	EXP	
Low traffic volume: 1.2 million moves/year					
Early reservation	0.86	0.03	2.35	1.14	12.2
1 day delay	0.82	0.04	2.36	1.16	12.9
2 days delay	0.84	0.10	2.37	1.21	16.0
No reservation	0.83	0.13	2.39	1.27	18.2
High traffic volume: 1.6 million moves/year					
Early reservation	0.88	0.03	2.41	1.15	18.5
1 day delay	0.82	0.05	2.42	1.20	19.1
2 days delay	0.88	0.10	2.45	1.27	21.8
No reservation	0.89	0.14	2.46	1.34	27.8
High traffic volume: 1.8 million moves/year					
Early reservation	0.87	0.03	2.43	1.17	21.7
1 day delay	0.87	0.06	2.44	1.21	25.0
2 days delay	0.87	0.10	2.46	1.28	31.6
No reservation	0.86	0.13	2.47	1.35	37.1

Table 19. Operational costs associated to the Yard Cranes: IMP/EXP container reshuffles. $\overline{E_C}$ = Average Energy Consumption in kWh.

Finally, YC overload is provided as a manner to evaluate the compliance of the reservation strategies with the *Least Congestion* principle. As indicated earlier, YC overload is indirectly

measured as the amount of time the workload surpasses a threshold of 25 moves/hour. The following statements summarize the conclusions drawn from the YC overload results:

- With respect to the volume of traffic, YCs overload increases as traffic increases in all cases, as expected.
- Regarding the reservation strategy, greater YC overload is also observed as T_R is deferred. The variation in crane overload is very significant within each traffic scenario, revealing the influence of T_R on the number of YCs required. In reality, the amount overload a terminal considers assumable is set in advance, and the number of MHE is set accordingly. In this work, for comparison purposes, one YC per block is considered.
- As no YC control is implemented in this case, further research may contribute to providing deeper insight into the effect of the yard template on the YCs when deployed by a real YC control.

6.6 Conclusions and future research

This work analyzes the Bay Allocation Problem considering the role of T_R, defined the time at which space in a container terminal is reserved for the arrival of export containers. Resulting yard templates are compared when stacking containers under Early Reservation and No Reservation strategies, and also with a proposed Intermediate Strategy in which some containers bound to a vessel are stacked on an online basis, after which space is reserved for the remainder of containers. Reservation strategies are evaluated under three traffic scenarios representing low, medium and high volumes of yard inventory. The objective of this setup is to assess the performance of the strategies in producing a more efficient yard template measured not only in terms of space utilization and organization, but also in terms of operational costs associated to YTs and YCs. Productivity is also evaluated indirectly in terms of YC overloading under the assumption that, in this case, crane interference is not significant.

Several experiments using a simplified DES model were conducted in an import/export parallel terminal. As expected, as T_R is deferred results reveal a tradeoff between the amount of space unused for reservation and the productivity of the terminal: the earlier the reservation, the more significant the amount of terminal space is steadily unused, an unwanted effect due to the usually high CAPEX and OPEX related to the terminal yard. The empty space required by offline stacking strategies represents a physical constraint that could only be assumed in terminals operating under particular circumstances, or under a small level of occupancy. In addition, small T_R produces longer average YTs displacements for import operations (during vessel downloading). On the other hand, the earlier the reservation, the shorter YT travel distances (vessel uploading operations), and less YCs energy expenditure when moving along the blocks to deliver the

containers. Smaller T_R also induces greater terminal productivity in terms of rehandling effort of export containers and YCs throughput (lower overload).

Summarizing, although reservation is a frequently ignored practice for a number of reasons, it may be desirable when possible to enhance terminal productivity. Even in congested situations, when the amount of space available for reservation is relatively small, reservation with a certain amount of time delay may be worth considering in practice for the abovementioned reasons.

Overall, the results reveal a strong influence of T_R in the yard template, and so it may be an important factor to consider in the short term planning of terminal operations. The limitations of the simplified DES model in the simulation of YTs and YCs may provide a more comprehensive understanding upon the preliminary evaluation of the associated costs.

Chapter 7

Conclusions and future research

7.1 Overview

Tendencies in container transportation observed in the last years reveal that the sustained growth in the amount containerized cargo will continue in the near future. Despite the foreseeable overcapacity in the containership fleet, the average size of the carriers continues to increase and, as a consequence, more pressure is put on the terminal operators, who must find new ways to handle operations efficiently and, if possible, increase terminal productivity.

In such context, the first objective of this thesis is to provide two low cost simulation tools for the analysis and evaluation of terminal yard operations. The tools are very flexible and easily customizable, and so they can be utilized to reproduce any type of parallel or perpendicular yard layouts. In addition, the YCs are simulated in detail, and therefore both models allow testing solutions and optimization techniques to improve the efficiency and productivity of yard handling processes. The second objective of this Thesis is to develop and test a model for YCs energy consumption, with the aim to improve the efficiency of YC operations. Letting alone the consequences of climate change, this topic is relevant as more and more terminals turn to electrification as a way to reduce emissions and costs. As a consequence, the demand for electric power is becoming a concern for the terminal operators, as they represent a significant port of the running costs. The third objective is to propose and analyze several operational problems related to the allocation of storage space in the container yard. In these analyses the energy cost is utilized to evaluate the efficiency of the operations. The final goal is to provide some insight into practices that may help reduce operating costs while improving the yard productivity.

In order to achieve these general objectives, this Thesis addressed the following particular issues:

1) The slot allocation problem for export containers in a perpendicular terminal in order to improve the efficiency of the operations and, at the same time, improve the ASC throughput; and finally
2) Optimize the dimensions of a perpendicular block with regard to energy costs.
3) The reservation of space in the yard as a design/operational problem, in which several trade-offs are identified and evaluated;

The structure of this Thesis corresponds to the abovementioned topics, in which the proposed problems are analyzed separately. The main findings and conclusions of this research derive from contributions of each chapter, as described next.

7.2 Main findings and conclusions

A summary of the most important conclusions from the analyses of this Thesis is presented here.

Appendix A describes the duty cycle of YCs, and provides specifications two common types of cranes: RMGs and ASC, which are used in parallel and perpendicular terminals respectively. More importantly, the chapter includes the mathematical formulation of two YC energy consumption models. Both models give consideration to individual characteristics of hoist, trolley and gantry crane movements. Weights of both container and crane moving parts and are also considered. Chronologically speaking, a simpler model was developed first based on the potential energy required to mobilize a weight in the horizontal and vertical directions, where coefficients account for the friction and efficiency of the mechanisms. Later on, a more sophisticated approach was adopted taking into account both the motor characteristics and the duration of the different resistances encountered by the gantry, trolley and hoist movements of the crane. To the best of the author knowledge, this is the first time that such a model have been considered to evaluate the energy consumption of yard cranes.

Moreover, this chapter also includes a simple example with the aim to compare the two energy consumption models. Results indicate that energy consumption from both models is of the same order of magnitude. This is important since, as the particular characteristics of the cranes and their motors may be difficult to obtain for a particular case, a simpler model can produce practical results that can be useful in different applications. However, it must be noted that the more sophisticated electric model produces higher absolute values of electric consumption than the potential model.

More importantly, regardless of the model, energy consumption of gantry and hoist movements are of the same order of magnitude; therefore, the relative importance of the energy consumption associated to gantry and hoist movements required to handle containers will depend on the

characteristics of each particular setup (crane duty cycle, such as the distances traveled by the cranes to pick the container, the height of the target container in the stack, the amount of rehandling needed, etc.). As a consequence, the operational strategies (i.e. housekeeping operations) utilized to manage the terminal yard will have a strong influence on energy consumption.

Chapter 3 is devoted to the description of two DES models utilized to analyze two types of terminal yard layouts: parallel and perpendicular. Both models are fully configurable and can be used to simulate yard handling operations with a high level of detail; therefore they provide the capability to be used with both research and industrial purposes.

In Chapter 4 an Efficient Stacking Storage Algorithm is proposed and evaluated. This algorithm is applied to the stacking operations of export containers in a single block of a perpendicular container terminal with two non-crossable cranes. Contrarily to other algorithms, the ESSA estimates the total energy consumption and travel time required for a single container in its transit along the block; thus, it makes use of the system state information at the time of stacking the container, but it also estimates information regarding the future container leaps along the block based on a probabilistic analysis. This way, the ESSA takes into account both the efficiency of the operations (in terms of energy consumption) and the productivity of the ASC (measured as the amount of time needed to execute a stacking operation). Each stacking operation is calculated in advance so as to take into account the additional energy expenditure and delays deriving from crane interferences when the two ASC operate in the block. Results indicate that, for the experimental setup and two levels of average block occupancy, the ESSA outperforms several benchmarking algorithms (including the stacking algorithm utilized in a real container terminal) not only in terms of an efficient use of the energy, but also in terms of productivity. Block performance is also evaluated as the average time needed by the land ASC to retrieve an import container from the moment it is requested, and also as the amount of time needed by the sea ASC to retrieve all the containers bound to a vessel. In these cases, the ESSA is also the most efficient algorithm.

Chapter 5, Optimization of the ASC block dimensions deals with a decision at a tactical level, which is the determination of the optimal length and width of a perpendicular block with two non-crossable ASCs. While the gross block capacity is kept constant, block length is varied from 24 to 72 bays of 20” length. A sensibility analysis is carried out with consideration to the block occupancy level. While the simulation setup is similar to that of Chapter 5, the Logic Stacking Algorithm is used in this case to analyze the energy expenditure and crane productivity, as the algorithm is used in reality. Results indicate that shorter blocks benefit from the reduction in energy consumption and increasing productivity; however, a trade-off is found with respect to

productivity: wider blocks induce longer trolley travel times, which in turn may reduce crane productivity, despite the shorter gantry displacements. As a consequence, the optimal block dimensions resulting from the experimental case are in the 30-36 bays range. The analysis also reveals that results are strongly influenced by housekeeping operations; to this extent, the need for optimization of such procedures, such the limitation of housekeeping movements during night shifts to benefit from lower prices of the electricity, may be an interesting topic for further research.

In Chapter 6, the block allocation problem in a parallel terminal is analyzed by means of a DES model. The objective is to analyze the distribution of containers in the yard depending on the moment at which space is reserved in the yard to allocate export containers arriving several days after. Such container distributions have an impact on the operational costs in terms of travel distances and amount of unproductive movements. The study provides a sensibility analysis with respect to both the volume of traffic in the terminal and the moment in which space for containers is reserved in the yard relative to the 3 day period in which export containers arrive to the terminal. The stacking strategies are primarily evaluated with respect to the amount and size of the clusters of containers bound to each vessel. In addition, the model calculates the operational costs incurred under each strategy not only for the YCs (in terms of energy consumption) but also for the YTs and ETs (in terms of distances traveled). In addition, various indicators of yard performance are given such as the amount of rehandling and the YC overload.

Results from the numerical experiments indicate that a higher degree of clustering is obtained as terminal traffic increases, as expected. The initial hypothesis is also confirmed: the amount of dispersion increases when the decision of reserving space is delayed relative to the moment at which containers bound to a vessel start arriving to the terminal. As anticipated, a greater degree of clustering (or equivalently, the amount of dispersion of containers bound to a vessel over the yard) has an impact on the operability of the terminal, which unveils several trade-offs when considering the deployment of YCs and YTs. First, YTs experience lower travel distances as T_R is differed, the average distance travelled increases when the growth in terminal traffic volumes. Conversely, when considering the YCs, the overall energy consumed increases as T_R is differed; however, as the terminal traffic increases, the increase of energy expenditure is only evident for export operations.

Interestingly, the order of magnitude of the energy expenditure of electric YCs and diesel powered YTs are similar; therefore the best strategy may also depend on the fuel and electricity prices at each particular location. Again, the analysis indicates that complex systems require taylor-made simulations and the use of stochasticity in the definition of the simulation inputs to account for the real world variability. Finally, as expected, crane overload follows an analogous trend to

energy consumption: it increases when the reservation is delayed, and also when the volume of container traffic increases. Fom this point of view, strategies that reserve space in the yard for export containers are preferred over online stacking. However, online stacking is a common practice in terminals with high occupancy levels; in these cases, delaying reservation whenever possible may still be a good practice in order to reduce the amount of rehandling and thus crane overload.

7.3 Future research

From current trends observed in the literature review and the experience acquired from the results of the work and the collaboration with a container terminal, it is clear that DES models are a worthwhile tool to investigate operational practices, including a wide number of topics related to the yard management. Hence the following lines for future research are suggested:

- As the electric powered YCs have the capability to regenerate energy during the deceleration phase of each type of movement (gantry, hoist and trolley), this energy can be exerted back in the electric network to feed other terminal's MHE, which will depend on factors as the amount of energy that could be harnessed by the YC during deceleration periods, or the simultaneity factor of YC duty cycles. To this matter, the DES simulation can be used to characterize the aggregated demand and the amount of energy recovered with respect to time. Energy recovery may came not only from YCs, but also from other electric MHE such as QCs. This would help dimensioning the equipment needed to manage the storage and distribution of electricity within a terminal.
- As indicated in the conclusions, the incidence of housekeeping operations depends on the logic of the stacking algorithm and the amount of traffic experienced in the terminal. Although housekeeping operations are not subject to the same priority rules as stacking or retrieval operations, they constitute a significant portion of the energy consumption, and therefore heuristics and optimization techniques can be can be applied to improve not only the efficiency but also the productivity of the ASCs.
- Regarding housekeeping operations, they are based on heuristic rules that could be subject of extensive analysis to elucidate, i.e., whether such operations can be delayed until nighttime hours and benefit from reduced price of electricity. In addition, the characterization of the energy and productivity of the ESSA at the time of stacking a container could be further improved by considering housekeeping operations in a probabilistic way.
- More sophisticated YC control and deployment systems can be proposed and utilized to evaluate their effect in the different analyses realized in the present Thesis. In the

case of perpendicular terminals, the YC control is usually oriented to the scheduling of the ASCs so as to minimize the time required to complete the jobs in the workload including waiting times. This approach could be also extended to account for the efficiency of the stacking operations and future housekeeping or retrieval operations. This way, the crane interference can be minimized according to the proposed efficiency and productivity criteria. With regards the parallel terminal model, the YC control will help determining the effect of crane interference and crane overload in the distribution of containers in the yard.

References

Angeloudis, P. and Bell, M. G. H. (2011) A review of container terminal simulation models. Maritime Policy and Management, 38(5): 523-540.

Alarcón, J.A., Sung-Woo, C. and Myong-Sop, J. (2012) Fuel Consumption within Cargo Operations at the Port Industry. The Asian Journal of Shipping and Logistics. Volume 28, Number 2. August 2012 pp. 227-254.

Ashar, A. (1991) On selectivity and accessibility. Cargo Systems, June: 44-45.

Aydin, C. and Ünlüyurt, T. (2007) Optimizing the container retrieval process via rehandle strategies. Master thesis from the Faculty of Engineering and Natural Sciences, Sabanci University. Istanbul.

Bazzazi, M., Safaei, N., Javadian, N. (2009) A genetic algorithm to solve the storage space allocation problem in a container terminal. Computers and Industrial Engineering, 56: 44-52.

Bennathan, E. and Walters, A. A. (1979) Port pricing and investment policies for development countries. Oxford University Press for the World Bank, Oxford, New York

Boll, C. (2004) Capacity Planning of container terminals by way of example of the new Jade Weser Port. In Port and Terminal Technology. Berkshire: MCI Media Limited, Amsterdam.

Borgman, B., van Asperen, E. and Dekker, R. (2010) Online rules for container stacking. OR Spectrum, 32: 687-716.

Box, G.E.P., Jenkins, G.M. and Reinsel, G.C. (1994) Time series analysis-forecasting and control. 4th edition. John Wiley &Sons, San Francisco.

Brinkmann B. (2005) Seehäfen‐planung und entwurf. Springer, Berlin.

Brinkmann, B., J. W. Böse (ed.), Handbook of Terminal Planning, Chapter 2, Operations Research/Computer Science Interfaces Series 49, DOI 10.1007/978-1-4419-8408-1_2, © Springer Science+Business Media, LLC 2011.

Cao Z, Lee D-H, Meng Q (2008) Deployment strategies of double-rail-mounted gantry crane systems for loading outbound containers in container terminals. Int J Prod Econ 115:221–228.

Carlo, H.J., Vis, I.F.A. and Roodbergen, K.J. (2013) Storage yard operations in container terminals: Literature overview, trends, and research directions. European Journal of Operational Research (in press).

Cartenì, A., de Luca, S. 2009. Simulation of a container terminal through a discrete event approach: literature review and guidelines for application. *Association for European Transport and contributors.*

Chang, *D., Jiang, Z., Yan, W., He, J.* (2010) Integrating berth allocation and quay crane assignments. - *Transportation Research Part E, 2010.*

Chapman, I., The end of Peak Oil? Why this topic is still relevant despite recent denials. Energy Policy (2013), http://dx.doi.org/10.1016/j.enpol.2013.05.010i

Chen, C., Hsu, W-J. and Huang, S-Y. (2003) Simulation and optimization of container yard operations: A survey. Proceedings of 2nd International Conference on Port and Maritime R&D and Technology: 23-29. Singapore.

Chen L., and Langevin, A. (2009). Determining the storage location for outbound containers in a maritime terminal. Proceedings of the 2009 IEEE. 543-547.

Chen, L. and Lu, Z. (2012) The storage location assignment problem for outbound containers in a maritime terminal. International Journal of Production Economics, 135:73-80.

Chen, T. (1999) Yard operations in the container terminal‐a study in the "unproductive moves." Maritime Policy and Management, 26(1): 27-38.

Chen, T., Lin, K. and Juang, Y-C. (2000) Empirical studies on yard operations. Part 2: Quantifying unproductive moves undertaken in quay transfer operations. Maritime. Policy and Management, 27(2): 191-207.

Cheung, Raymond K., Chung-Lun Li, and Wuqin Lin. "Interblock Crane Deployment in Container Terminals." Transportation Science 36, no. 1 (2002): 79-93.

Choe R, Park T, Seung MO, Kwang RR (2007) Real-time scheduling for non-crossing stacking cranes in an automated container terminal. In: AI 2007: advances in artificial intelligence, Number 4830/2007 in Lecture Notes in Computer Science, Springer, Berlin, pp 625–631.

Choo Chung, K. (1993) Port performance indicators. The World Bank, Transportation, Water and Urban Development Department, No. PS-6.

Chu, C-Y and Huang, W-C. (2005) Determining container terminal capacity on the basis of an adopted yard handling system. Transport Reviews, 25(2): 181-199.

CMA-CGM. (2012) Terminal storage and demurrage. Available online: http://ebusiness.cmacgm.com. Retrieved and accessed in June 29, 2012.

De Castilho, B. and Daganzo, C.F. (1991) Optimal pricing policies for temporary storage at ports. Transportation Research Record, 1313: 66-74.

De Castilho, B. and Daganzo, C.F. (1993) Handling strategies for import containers at marine terminals. Transportation Research. B, 27B(2): 151-166.

De Haan, L. and Zhou, C. (2009) Evaluation of the statistical methods currently used to determine boundary conditions. Report to the Centre for Water Management (Rijkswaterstaat), Deltares, The Netherlands.

Dekker, S. (2005) Port Investment ‐ Towards an integrated planning of port capacity, The Netherlands TRAIL Research School, Delft, TRAIL Thesis Series, no. T2005/5.

Dekker, R., Voogd, P. and Van Asperen, E. (2006) Advanced methods for container stacking. OR Spectrum, 28: 563-586.

Duinkerken, MB., Evers, JJM. and Ottjes, JA. (2001) A simulation model for integrating quay transport and stacking policies in automated terminals. Proceedings of the 15th European Simulation Multiconference (ESM 2001), SCS, Prague.

Duinkerken, MB., Dekker, R., Kurstjens, S.T.G.L, Ottjes, J.A. and Dellaert, N.P. (2006) Comparing transportation systems for inter-terminal transport at the Maasvlakte container terminals. OR Spectrum, 28:469-493.

Electrical equipment, Federation Europeenne de la Manutention. Rules for the design of hoisting appliances. Booklet 5. 3rd edition revised 1998.10.01.

Fan, L., Low, M.Y.H., Ying, H.S., Jing, H.W., Min, Z. and Aye, W.C. (2010) Stowage planning of large containership with tradeoff between crane workload balance and ship stability.

Proceedings of the 2010 IAENG International Conference on Industrial Engineering. 1537-1543, Hong Kong.

Gkonis, K. G., Psaraftis, H. N. Some key variables affecting liner shipping costs. Laboratory for Maritime Transport, School of Naval Architecture and Marine Engineering. National Technical University of Athens, 2009.

Goss, R. and Stevens, H. (2001) Marginal cost pricing in seaports. International Journal of Maritime Economics, 3: 128-138.

Günther, H.O. and Kim.K.H. (2006) Container terminals and terminal operations. OR Spectrum, 28: 437-445.

Han, Y., Lee, L.H., Chew, E.P. and Tan, K.C. (2008) A yard storage strategy for minimizing traffic congestion in marine container transshipment hub. OR Spectrum, 30: 697-720.

He, J, Daofang C, Weijian M, and Wei, Y. "A Hybrid Parallel Genetic Algorithm for Yard Crane Scheduling." Transportation Research Part E: Logistics and Transportation Review 46, no. 1 (2010): 136-55.

Heggie, I. (1974) Charging for port facilities. Journal of Transportation Economics and Policy, 8: 3-25.

Henesey, L., Wernstedt, F. and Davidsson, P. (2002) A market-based approach to container port terminal management. Proceedings of the 15th European Conference on Artificial Intelligence, Workshop-Agent Technologies in Logistics, Lyon.

Henesey, L. (2004) Enhancing container terminal performance: a multi agent systems approach. Licentiate thesis in Department of systems and software engineering. Blekinge Institute of Technology, Karlshamn, Sweden.

Hirashima, Y., Ishikawa, N., Takeda, K. (2006) A new reinforcement learning for group-based marshaling plan considering desired layout of containers in port terminals. Proceedings of the 2006 IEEE International Conference. 670-675, Beijing, China.

Holguín-Veras, J. and Jara-Díaz, S. (1999) Optimal pricing for priority services and space allocation in container ports. Transportation Research B, 33(2): 81-106.

Holguín-Veras, J. and Jara-Díaz, S. (2006) Preliminary insights into optimal pricing and space allocation at intermodal terminals with elastic arrivals and capacity constraint. Network and Spatial Economics, 6: 25-38.

Hussein, M and Petering, Mathew E.H. Genetic Algorithm-Based Simulation Optimization of Stacking Algorithms for Yard Cranes to Reduce Fuel Consumption at Seaport Container

Transshipment Terminals, WCCI IEEE World Congress on Computational Intelligence (2012).

Huynh, N. (2008) Analysis of container dwell time on marine terminal throughput and rehandling productivity. Journal of International Logistics and Trade, 6 (2): 69-89.

Imai, A., Nishimura, E., Papadimitriou, S. and Sasaki, K. (2002) The containership loading problem. International Journal of Maritime Economics, 4: 126-148.

Imai, A., Sasaki, K., Nishimura, E. and Papadimitriou, S. (2006) Multi-objective simultaneous stowage and load planning for a container ship with container rehandle in yard stacks. European Journal of Operational Research, 171: 373-389.

International Transport Forum. Statistics Brief Infrastructure Investment. July 2015.

Jiang, X., Lee, L.H., Chew, E.P., Han, Y. and Tan, K.C. (2012) A container yard storage strategy for improving land utilization and operation efficiency in a transshipment hub port. European Journal of Operational Research, 221:64-73.

JOC Group Inc. Berth productivity. The Trends, Outlook and Market Forces Impacting Ship Turnaround Times, Whitepaper 2014.

Kang, J., Ryu, K.R. and Kim, K.H. (2006) Deriving stacking strategies for export containers with uncertain weight information. Journal of Intelligent Manufacturing, 17:399-410.

Kemme N (2010) Documentation of a simulation model for an RMG yard block at seaport container terminals.

Kemme, N., 2012. Effects of storage block layout and automated yard crane systems on the performance of seaport container terminals. OR Spectrum 34:563–591.

Kemme, N., 2013. Design and Operation of Automated Container Storage Systems, Physica-Verlag Heidelberg. ISBN 978-3-7908-2885-6.

Kiefer, N. (1988) Economic duration data and hazard functions. Journal of Economic Literature, 26(2): 646-679.

K.H. Kim, D.Y. Kim, Group storage methods at container port terminals, The Material Handling Engineering Division 75th Anniversary Commemorative Volume, ASME 1994 MH-Vol. 2, 1994, pp. 15±20.

Kim, K.H. (1997) Evaluation of the number of rehandles in container yards. Computers and Industrial Engineering, 32(4): 701-711.

Kim, K.H. and Bae, J.W. (1998) Re-marshaling export containers in port container terminals. Computers and Industrial Engineering, 35: 655-658.

Kim, K.H. and Kim, H.B. (1999) Segregating space allocation models for container inventories in port container terminals. International Journal of Production Economics, 59: 415-423.

Kim, K.H., Park, Y.M. and Ryu, K-R. (2000) Deriving decision rules to locate export containers in container yards. European Journal of Operational Research, 124:89-101.

Kim, K.H. and Kim, H.B. (2002) The optimal sizing of the storage space and handling facilities for import containers. Transportation Research B, 36: 821-835.

Kim, K.H. and Park, K.T. (2003) A note on a dynamic space-allocation method for outbound containers. European Journal of Operations Research, 148(1): 92-101.

Kim, K.H. and Hong, G.P. (2006) A heuristic rule for relocating blocks. Computers and Operations Research, 33: 940-954.

Kim, K.H. and Kim, K.Y. (2007) Optimal price schedules for storage of inbound containers. Transportation Research B, 41: 892-905.

Kim, K.H., Park, Y-M. and Jin, M-J. (2008) An optimal layout of container yards. OR Spectrum , 30: 675-695.

Kim, K.H., Keung M.L., and Hark H. "Sequencing Delivery and Receiving Operations for Yard Cranes in Port Container Terminals." International Journal of Production Economics 84, no. 3 (2003): 283-92.

Kozan, E. and Preston, P. (2006) Mathematical modelling of container transfers and storage locations at seaport terminals. OR Spectrum 28:519–537. DOI 10.1007/s00291-006-0048-1.

Ku, L.P., Lee, L.H., Chew, E.P. and Tan, K.C. (2010) An optimization framework for yard planning in a container terminal: case with automated rail-mounted gantry cranes. OR Spectrum, 32: 519-541.

Le-Griphin, H.D. and Murphy, M. (2006) Container terminal productivity: experiences at the ports of Los Angeles and Long Beach. Container terminal productivity, Feb/1/2006.

Lee, B.K. and Kim, K.H. (2010a) Comparison and evaluation of various cycle-time models for yard cranes in container terminals. International Journal of Production Economics, 126: 350-360.

Lee, B.K. and Kim, K.H. (2010b) Optimizing the block size in container yards. Transportation Research E, 46: 120-135.

Lee, B.K. and Kim, K.H. (2013) Optimizing the yard layout in container terminals. OR Spectrum, 35: 363-398.

Lee, C-Y. and Yu, M. (2012) Inbound container storage price competition between the container terminal and a remote container yard. Flexible Service and Manufacturing, 24: 320-348.

Lee, L.H., Chew, E.P., Tan, K.C. and Han, Y. (2006) An optimization model for storage yard management in transshipment hubs. OR Spectrum, 28: 539-561.

Lee, Y. and Hsu, N.Y. (2007) An optimization model for the container pre-marshalling problem. Computers and Operations Research, 34: 3295-3313.

Lee, Y. and Chao, S-L. (2009) A neighborhood search heuristic for pre-marshalling export containers. European Journal of Operational Research, 196: 468-475.

Levinson, M. (2006) How the shipping container made the world smaller and the world economy bigger. Princeton University Press, Princeton, New Jersey.

Lim, A. and Xu, Z. (2006) A critical-shaking neighborhood search for the yard allocation problem. European Journal of Operational Research, 174: 1247-1259

Linn, Richard, Ji-yin Liu, Yat-wah W, Chuqian Z, and Katta G. Murty. "Rubber Tired Gantry Crane Deployment for Container Yard Operation." Computers & Industrial Engineering 45, no. 3 (2003): 429-42.

Liu, C.I., Jula, H. and Ioannou, P.A. (2002) Design, simulation and evaluation of automated container terminals. IEEE Transactions on intelligent transportation systems, 3: 12-26.

Liu, C.I., Jula, H., Vukadinovic, K. and Ioannou, P.A. (2004) Automated guided vehicle system for two container yard layouts. Transportation Research C, 12: 349-368.

Maggio, G., Cacciola, G., 2009. A variant of the Hibbert curve for world oil production forecasts. Energy Policy 37, 4761–4770.

Meersmans, P.J.M. and Dekker, R. (2001) Operations research supports container handling. Econometric Institute Research Papers, No EI 2001-22, Erasmus University Rotterdam, Erasmus School of Economics (ESE), Econometric Institute.

Moorthy, R., Teo, C.P. 2006. Berth management in container terminal: the template design problem. OR Spectrum 28, 495–518.

Murty, K.G. (1997) Storage yard planning in container shipping terminals. Technical Report, Department of IEEM, Hong Kong University of Science and Technology.

Murty, K.G., Liu, J., Wan, Y.W. and Linn, R. (2005) A decision support system for operations in a container terminal, Decision Support Systems, 39(3): 309-332.

Nam, K-C. and Ha, W-I. (2001) Evaluation of handling systems for container terminals. Journal of Waterway, Port, Coastal, and Ocean Engineering, 127: 171-175.

Ng, A.K.Y., Padilha, F. and Pallis, A.A. (2013) Institutions, bureaucratic and logistical roles of dry ports: The Brazilian case. Journal of Transport Geography, 27: 46-55.

Ng, W. C., and K. L. Mak. "Yard Crane Scheduling in Port Container Terminals." Applied Mathematical Modelling 29, no. 3 (2005): 263-76.

Ng, W. C. "Crane Scheduling in Container Yards with Inter-Crane Interference." *European Journal of Operational Research* 164, no. 1 (2005): 64-78.

Nishimura, E., Imai, A., Janssens, G.K. and Papadimitriou, S. (2009) Container storage and transshipment marine terminals. Transportation Research E, 45:771-786.

Park, T., Kwang, M.S. and Ryu, K.R. (2010) Optimizing Stacking Policies Using an MOEA for an Automated Container Terminal.

Park, T., Choe, R., Ok, S. M. , and Ryu, K. R., (2010b) Real-time scheduling for twin RMGs in an automated container yard, OR Spectrum, OR Spectrum (2010) 32:593–615. DOI 10.1007/s00291-010-0209-0.

Park, T., Choe, R., Kim, Y.H. and Ryu, K.R. (2011) Dynamic adjustment of container stacking policy in an automated container terminal. International Journal of Production Economics, 133: 385-392.

Petering, M.E.H. (2008) Parallel versus perpendicular yard layouts for seaport container transshipment terminals: an extensive simulation analysis. Proceedings of the International trade and freight transportation conference, Ayia Napa, Cyprus.

Petering, M.E.H. (2009) Effect of block width and storage yard layout on marine container terminal performance. Transportation Research E, 45: 591-610.

Petering, M.E.H. and Murty, K.G. (2009) Effect of block length and yard crane deployment systems on overall performance at a seaport container transshipment terminal. Computers and Operations Research, 36(5): 1711-1725.

Pickands, J. (1975) Statistical inference using extreme order statistics. The Annals of Statistics, 3: 119-131.

Rodrigue, J-P. and Notteboom, T. (2008) The terminalization of supply chains. Proceedings of the 2008 International Association of Maritime Economists (IAME) Conference, Dalian, China.

Rijsenbrij, 2011; Electric consumption in terminals.

Scarrott, C. and MacDonald, A. (2012) A review of extreme value threshold estimation and uncertainty quantification. Statistical Journal, 10: 33-60.

Saanen, Y.A., Valkengoed, M.V. 2006. Comparison of three automated stacking alternatives by means of simulation. Proceedings of the 2005 Winter Simulation Conference.

Saanen, Y.A. (2009) Trafalquar manual v4.0. Software Manual, TBA b.v., Delft.

Saanen, Y.A. (2011) Modeling techniques in planning of terminals: the quantitative approach. Ensuring planning becomes reality. Handbook of Terminal planning, chapter 5, pp. 83-102, Springer, New York.

Saurí, S., Martín, E. 2011. Space allocating strategies for improving import yard performance at marine terminals. Transportation Research Part E 47: 1038–1057.

Schütt, H. (2011) Simulation technology in planning, implementation and operation of container terminals. Handbook of Terminal planning, chapter 6, pp. 103-116, Springer, New York.

Schmitt, A.J., Snyder, L.V. and Shen, Z-J.M. (2010) Inventory systems with stochastic demand and supply: Properties and approximations. European Journal of Operational Research, 206: 313-328.

Sculli, D. and Hui, C.F. (1988) Three-dimensional stacking of containers. Omega, 16(6): 585-594.

Sgouridis, S., Makris, D. and Angelides, D. (2003) Simulation analysis for midterm yard planning in container terminal. Journal of waterway, port, coastal and ocean engineering, 129(4): 178-187.

Sharif, (2011) Yard crane scheduling at seaport container terminals: a 2 comparative study of centralized and decentralized approaches. TRB 2015.

Speer, U., John, G., Fischer, K. Scheduling Yard Cranes Considering Crane Interference. Computational Logistics. Lecture Notes in Computer Science Volume 6971 (2011) pp 321-340.

Stahlbock, R. and Vo β , S. (2008) Operations research at container terminals: a literature update. OR Spectrum, 30: 1‑52.

StahlbockR,Voss S (2010) Efficiency considerations for sequencing and scheduling of double-rail-mounted gantry cranes at maritime container terminals. Int J Shipping Transp Logist 2:95–123.

Steenken, D., Vo β , S. and Stahlbock, R. (2004) Container terminal operation and operations research ‑ a classification and literature review. OR Spectrum, 26: 3-49.

Taleb-Ibrahimi, M. 1989, Modeling and analysis of container storage in ports, Ph.D. Thesis, University of California, Berkeley.

Taleb-Ibrahimi, M., De Castilho, B. and Daganzo, C. (1993) Storage space vs handling work in container terminals. Transportation Research B, 27B(1): 13-32.

Thanh (2012) Electric consumption in terminals. PhD Thesis. Deaking University.

Thorensen, C.A. 2003. Port Designer's Handbook, Thomas Telford Books, ISBN: 978-07217-3228-6.

UNCTAD, (2015) Review of maritime transport, United Nations Publication, New York, Geneva.

U.S. Energy Information Administration, "Impacts of the Kyoto Protocol on U.S. Energy Markets and Economic Activity" (Washington, DC: U.S. Department of Energy, 1998).

Valkengoed, M.P.V. (2004). How passing cranes influence stack operations in a container terminal: a simulation study. Diploma thesis, University of Amsterdam.

Veeke, H. P.M. Ottjes, J.A., Verbraeck, A., Saanen, Y., 2003. A simulation architecture for complex design projects. Proceedings of the 14th European Simulation Symposium.

Vis, I.F.A. and de Koster, R. (2003) Transshipment of containers at a container terminal: An overview. European Journal of Operational Research, 147:1-16.

Vis, I.F.A. and Harika, I. (2004) Comparison of vehicle types at an automated container terminal. OR Spectrum, 26: 117-143.

Vis, I.F.A. (2006) A comparative analysis of storage and retrieval equipment at a container terminal. International Journal of Production Economics, 103: 680-693.

Vis, I.F.A., Carlo, H.J. (2010), Sequencing two cooperating automated stacking cranes in a container terminal, Transportation Science, 44(2), 169-182.

Wan, Y-W., Liu, J. and Tsai, P-C. (2009), The assignment of storage locations to containers for a container stack. Naval Research Logistics, 56: 699-713.

Watanabe, I. (1991) Characteristics and analysis method of efficiencies of container terminal – an approach to the optimal loading/unloading method. Container Age March, 36-47.

Watanabe, I. (2001) Container terminal planning—A theoretical approach, World Cargo News Publishing, UK.

Wenkai, L., Wu, Y. Petering, M.E.H. Goh, M. and de Souza, R. "Discrete Time Model and Algorithms for Container Yard Crane Scheduling." European Journal of Operational Research 198, no. 1 (2009): 165-72.

Wiese, J., Kliewer, N. and Suhl, L. (2009) A survey of container terminal characteristics and equipment types. Technical report 0901, DS&OR Lab, University of Paderborn, Paderborn.

Wiese, J., Suhl, L. and Kliewer, N. (2011) Planning container terminal layouts considering equipment types and storage block design. Handbook of terminal planning. Springer, New York.

Wijlhuizen, E.TH.J. and Julialei, F.M. (2008), "The Sustainable Container Terminal," 2nd International Conference on Harbours, Air Quality and Climate Change, Rotterdam.

Won, S.H., Zhang, X. and Kim, K.H. (2012) Workload-based yard-planning system in container terminals. Journal of Intelligent Manufacturing, 23: 2193-2206.

Woo, Y.J. and Kim, K.H. (2011) Estimating the space requirement for outbound container inventories in port container terminals. International Journal of Production Economics, 133, 293-301.

Word Energy Outlook 2014. International Energy Agency.

Xin, J., Negenborn, R.R., Lodewijks, G. (2013a). Hybrid model predictive control for equipment in an automated container terminal, in: Proceedings of the 2013 IEEE International Conference on Networking, Sensing and Control, Evry, France. pp. 746‑752.

Xin, J., Negenborn, R.R., Lodewijks, G. (2013b). Hybrid MPC for balancing throughput and energy consumption in an automated container terminal, in: Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems, The Hague, The Netherlands. pp. 1238‑1244.

Xin, J., Negenborn, R.R., Lodewijks, G. (2015). Energy-efficient container handling using hybrid model predictive control. International Journal of Control. DOI: 10.1080/00207179.2015.1043350.

Yang, C., Choi, Y. and Ha, T. (2004) Simulation-based performance evaluation of transport vehicles at automated container terminals. OR Spectrum, 26: 149-170.

Zhang, C., Liu, J., Wan, Y., Murty, K. and Linn, R. (2003) Storage space allocation in container terminals. Transportation Research B, 37: 883-903.

Zhang, C, Yat-wah W., Jiyin L., and Richard J.L. "Dynamic Crane Deployment in Container Storage Yards." Transportation Research Part B: Methodological 36, no. 6 (2002): 537-55.

Zhen, L. 2014. Container yard template planning under uncertain maritime Market. Transportation Research Part E 69 199–217.

Zyngiridis I (2005) Optimizing container movements using one and two automated stacking cranes. Master thesis, Naval Postgraduate School Monterrey.

Zittel, W., Zerhusen, J., Zerta, M. (2013) Fossil and Nuclear Fuels – the Supply Outlook. Energy Watch Group, Berlin.

Zondag, B., Bucci, P., Gützkow, P., De Jong, G. (2010) Port competition modeling including maritime, port and hinterland characteristics. Maritime Policy and Management, 37(3): 179-194.

Websites, textual citations

http://www.joc.com/maritime-news/ships-shipbuilding/average-size-container-ship-order-rise-13-percent-2020_20150707.html

http://www.hellenicshippingnews.com/newbuilding-order-book-shows-overcapacity-in-container-shipping/

http://www.cargobusinessnews.com/featured_stories/The-big-ship-rate-volatility-conundrum.html

http://maritime-executive.com/editorials/megaships-economics-nearing-its-limit

Annex A

Crane Duty Cycle and Energy Consumption

This annex is dedicated to the description and analysis of the cranes used in the DES models, as well as the two different energy models utilized to estimate their consumption. The annex is structured as follows: first, a description of yard cranes is provided; second, are duty cycle of a real crane is

A.1. Description of a yard crane

Yard cranes are gantry robots over rails with a rigid guiding beam and fixed legs that perform high-precision storage and retrieval of containers (Figure 27). Several types of yard cranes can be found in the market depending on the type of wheels (rails or tires), the power source, or the level of automation. As previously indicated, this thesis considers two types of fully electrified cranes: ASC and RMG. These type of cranes and are becoming predominant in the market as they have several advantages over gasoil powered cranes:

- Lower noise, contaminant emissions, or spills
- Lower power consumption, with the possibility of regenerating power through gantry braking or hoist lowering.
- Lower maintenance costs,

Main power supply and data transmission of electric cranes are managed by motor driven cable reels. These cranes are also provided with automated functionalities and onboard monitoring equipment. Energy and data transmission to the trolley is managed by guiding chain systems.

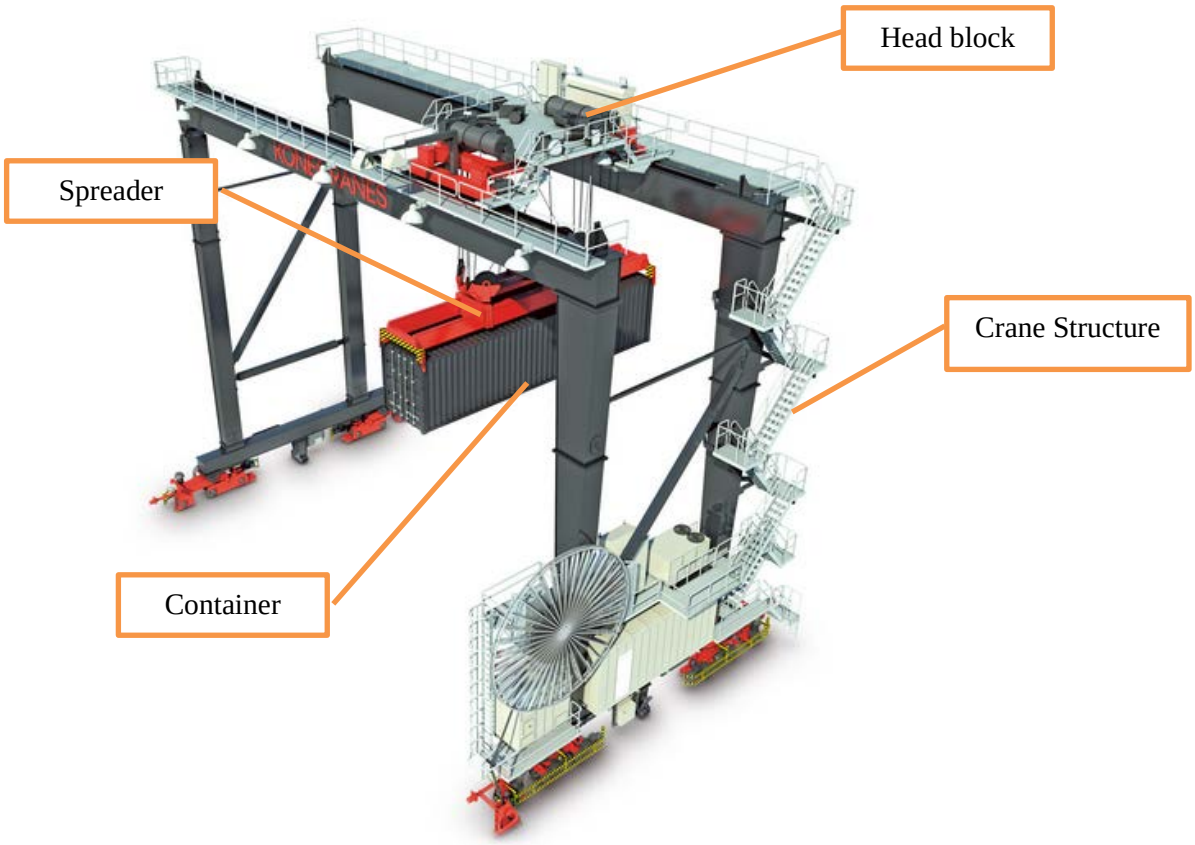


Figure 56. Automatic Stacking Crane (Courtesy of Kone Cranes).

There are several types of mechanisms that carry out the hoist and trolley movements. Figure 54 illustrates an RMG bridge on which the trolley movement takes place.

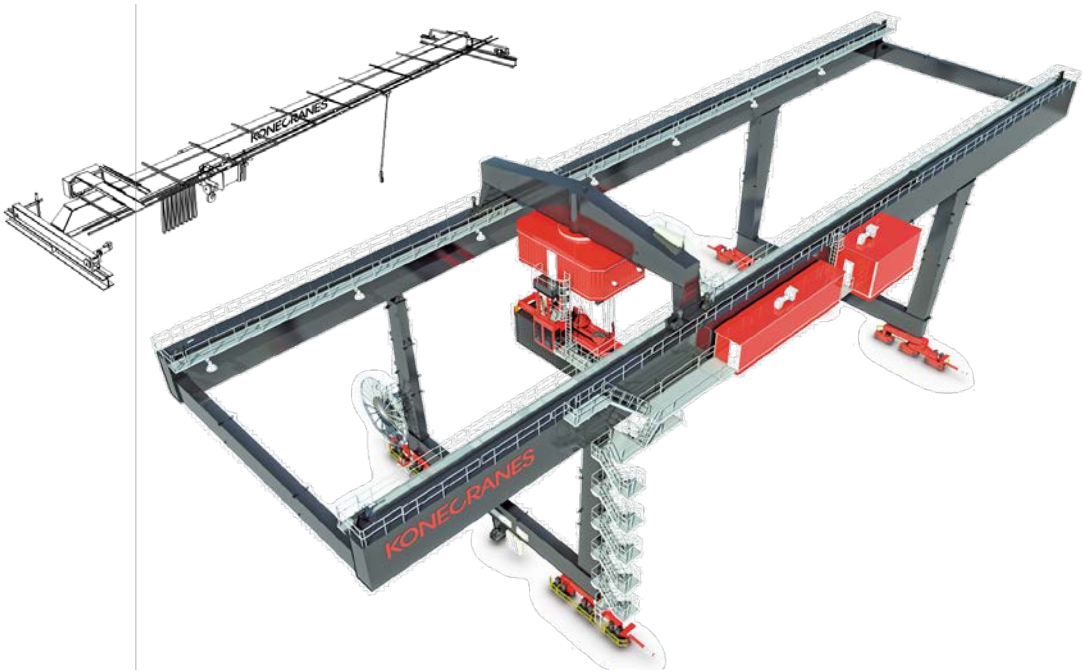


Figure 57. Crane bridge (left) RMG crane (right) and (Courtesy of Kone Cranes).

A.1.1. ASC Specifications

In this section, we provide a summary of the crane characteristics required to feed the energy consumption models.

A.1.1.1. Dimensions

The ASC dimensions are provided next:

- Working height: 1-over-5 high-cube containers
- Working span: 9 containers
- Track gauge: 28 m
- Length: 18.5 m
- Weight: 180 tons

A.1.1.2. Crane kinematics

The next kinematic characteristics are used for the calculation of the crane movements.

- Gantry travel:
  - Speed: 240 m/min
  - Acceleration: 0.4 m/s²
- Cross travel
  - Speed: 60 m/min
  - Acceleration: 0.4 m/s²
- Hoisting/lowering speed (winds up to force 10 Bft):
  - Speed: 39 – 72 m/min (full – empty)
  - Acceleration: 0.35 m/s²

A.1.2. RMG Specifications

The next crane specifications are utilized in the parallel terminal model.

A.1.2.1. Dimensions

The RMG dimensions are provided next:

- Working height: 1-over-4 high-cube containers
- Working span: 6 + 1 containers
- Track gauge: 22 m
- Length: 16.5 m
- Weight:
  - Whole crane: 116 tons
  - Trolley cabin: 16 tons
  - Spreader. 2 tons

A.1.2.2. Crane kinematics

The next kinematics are used for the calculations

- Gantry travel
  - Speed (empty - laden): 135 - 90 m/min
  - Acceleration: 0.35 m/s²
- Cross travel:
  - Speed: 70 m/min
  - Acceleration: 0.4 m/s²
- Hoisting/lowering (empty - laden), winds up to force 10 Bft:
  - Speed: 35 - 72 m/min
  - Acceleration: 0.35 m/s²

A.2. Crane Duty cycle

The crane characteristics described above are utilized to calculate the individual gantry, trolley and hoist movements that compose the representative crane duty cycle.

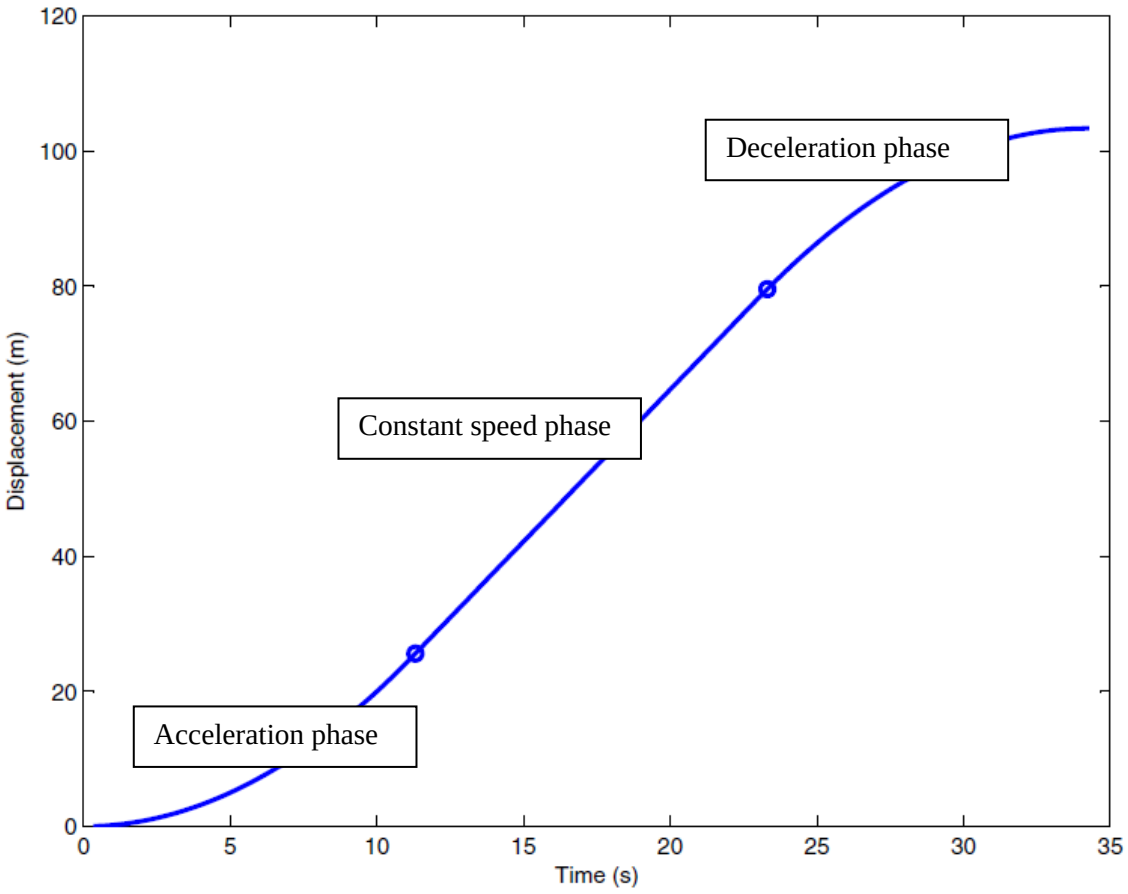


Figure 58. Kinematics of the ASC gantry movement.

A.2.1. Characterization of the crane individual movements

ASC or RTG perform three main types of movements, as indicated in the next table:

Movement	Direction	Crane moving parts
Gantry	X (Longitudinal)	Whole crane
Trolley	Y (Transversal)	Spreader + headblock
Hoist/lower	Z (vertical)	Spreader

Table 20. Types of individual crane movements.

The kinematics of each movement comprise a phase of acceleration, a phase of constant speed, and a phase of deceleration, as illustrated in Figure 8.

A.2.2. ASC cycle

A typical duty cycle of an ASC can be represented by the following steps:

1. Gantry to the bay position where the target container is located.
2. Start of duty cycle with the spreader in the default position and trolleying towards target container.
3. Lower hoist to pick up first container.
4. Lift the first container.
5. Trolley the first container to the end of the stack.
6. Lower hoist to land first container within the stack.
7. Lift the spreader only
8. Trolley toward the target container location.
9. Lower the spreader towards the target container.
10. Lift the target container.
11. Gantry to the next bay position awaiting the container.
12. Trolley the target container to the target lane.
13. Lower the target container onto the trailer of the truck/AGV/SC.
14. Lift the spreader to the default position.

A.2.3. RMG cycle

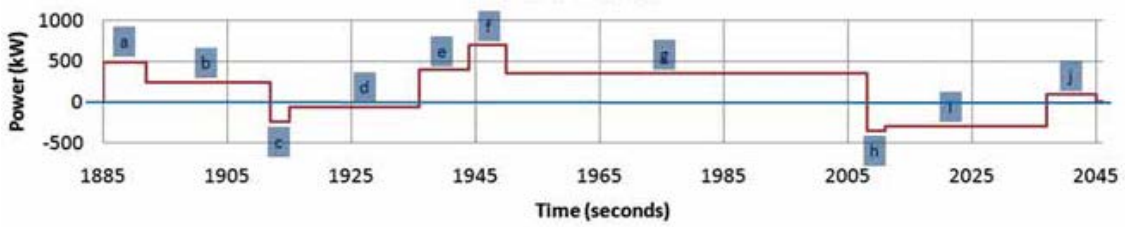
A typical duty cycle of an RMG can be represented by the following steps:

1. Gantry to the bay position where the target container is located.
2. Start of duty cycle with the spreader in the default position and trolleying towards target container.
3. Lower hoist to pick up first container.
4. Lift the first container.
5. Trolley the first container to the end of the stack.

6. Lower hoist to land first container within the stack.
7. Lift the spreader only
8. Trolley toward the target container location.
9. Lower the spreader towards the target container.
10. Lift the target container.
11. Trolley the target container to the truck lane.
12. Lower the target container onto the trailer of the truck.
13. Lift the spreader to the default position.
14. Gantry to the next bay position awaiting the container.

A.3. Electric consumption of YCs and energy consumption models

As the industry is shifting towards electrically powered YCs, operators are becoming increasingly concerned by the growth in demand for electric power of their container terminals. For instance, according to Le (2012) a single YC can have a peak demand of around 700 kW (Figure 55), which is important to determine the overall electric demand of the terminal.



ID	Movement	Consumption	Generation
a	Acceleration to maximum gantry without container	500 kW	
b	Gantry without container	250 kW	
c	Deceleration without container		200 kW
d	Lower spreader down without container		50 kW
e	Hoist spreader up with container	400 kW	
f	Acceleration to maximum gantry with container	50 kW	
g	Gantry with container	300 kW	
h	Deceleration with container		350 kW
i	Lower spreader down with container		300 kW
j	Hoist spreader up without container	100 kW	

Figure 59. Electric power demand YC duty cycle (Le, 2012) with notation.

An approximate idea of the YCs overall consumption can be inferred when considering that an average crane duty cycle requires ≈10 kWh per container. This value can be easily used to estimate the order of magnitude of the relatively large amount energy required by the yard cranes in a terminal. For example, the power consumption of a single YC in a mid-size terminal is in the

order of magnitude of 100.000 kWh. It is also worth noticing that electric powered YCs have the capability to recover energy during the deceleration part of the cycle, which amounts to ≈3 kWh.

Two energy consumption models are utilized throughout the analyses of this Thesis. Chronologically, a simple potential based model was developed first to study the parallel terminal DES model; later on, a more sophisticated electric model was provided to analyze the perpendicular block DES model.

For both models, the energy consumption associated with each movement is formulated with consideration to the weights of the container being carried and the moving parts of the crane.

A comparison between the outputs from both models is given in section 3.6.9.

A.4. Potential model

The energy consumed by the ASC crane on each gantry, trolley or hoist/lower movement is calculated according the nominal resistance to the movement, which is described by equation (1):

$$\boldsymbol{E_m = \frac{(M_m + M_C) \cdot g \cdot X_m \cdot \mu_m}{\eta_m}} \qquad \textbf{(Equation 37)}$$

Where:

- m denotes the type of individual movement (gantry, trolley or hoist);
- M_m is the mass associated to the moving parts of the crane,
- M_C is the mass of the container,
- g is the gravitational acceleration,
- Xm is the distance traveled by each movement, and finally
- μ_m and η_m are coefficients that account for the friction between surfaces and the engines efficiency, respectively. Values utilized are given in Table 2.

Coefficient	Gantry	Trolley	Hoist
$\boldsymbol{\alpha_m}$	0.005	0.006	1.0
$\boldsymbol{\beta_m}$	0.95	0.90	0.85

Table 21. Friction and efficiency of the potential model.

The estimation of the energy spent not only depends on the distance traveled by the container on the three axes (Δx, Δy, Δz) but also on the combined weight of the container and the moving parts of the YC (m_{cont} and m_s respectively), according to the flowing equations:

Gantry (g): **(Equation 38)**

$$e_g(Ws) = \frac{\Delta E_{mec}}{\eta_g} = \frac{\mu_g (m_g + m_{cont}) g \Delta x}{\eta_g}$$

Spreader (s): $$e_s(Ws) = \frac{\Delta E_{mec}}{\eta_s} = \frac{\mu_s (m_s + m_{cont}) g \Delta y}{\eta_s}$$ **(Equation 39)**

Hoist/Lift (h/l): $$e_{h/l}(Ws) = \frac{\Delta E_{pot}}{\eta_{h/l}} = \frac{(m_s + m_{cont}) g \Delta z}{\eta_{h/l}}$$ **(Equation 40)**

A.5. Electric model

The electric consumption model is more sophisticated as it takes into account the kinematics of the crane individual movements for the various crane components involved in the execution of the movement.

A.5.1. Types of crane moving systems and resistances

Next, crane movements in the three axes (hoist, trolley and gantry) are described. As the cranes usually have different types of hoist and trolley mechanisms, which are described next.

A.5.1.1. Hoist

Several hoisting mechanisms can be selected in the model:

a) Hoisting winch on the trolley (Number of rope sheaves: minimum),
b) Hoisting winch fixed on the bridge (Number of rope sheaves: depending on wire rope layout), or
c) Stacking crane with 'rope tower' (Number of sheaves: depends on rope system in trolley).

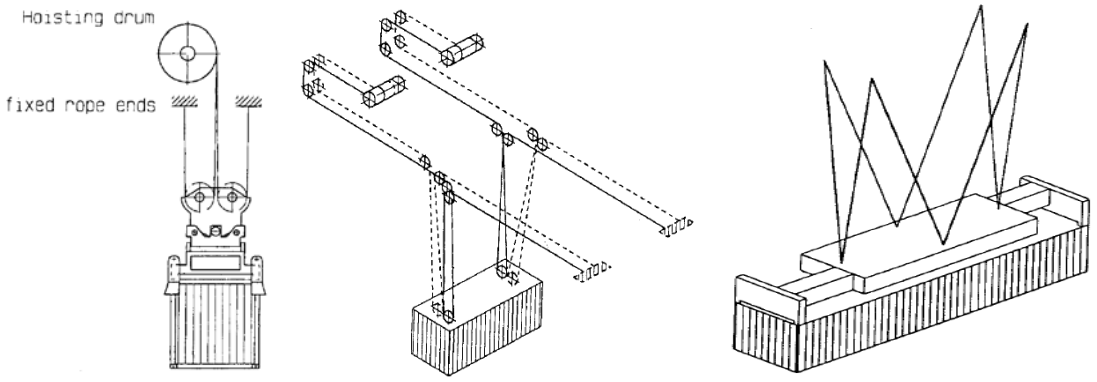


Figure 60. Hoist mechanisms a (left), b (center) and c (right). Source: Cranes, Design Practice and Maintenance.

From now on, the preferred hoist mechanism is the winch on the trolley.

A.5.1.2. Trolley

There are three general types of trolley systems, each of them with a specific type of calculation:

a) Direct driven trolleys or motor trolleys
b) Trolleys, which are pulled by wire ropes
c) Rope driven trolleys for grab-unloaders with a main- and an auxiliary trolley

From this moment on, we will assume that the ASCs utilize the first option “direct driven trolleys or motor trolleys; wheel slip control”.

A.5.2. Components of the crane movements

For each individual movement, the electric model of energy consumption comprises several types of resistances that must be overcome during the duty cycle of the crane, namely:

- Resistance due to normal travel of the crane or its parts,
- Resistance due to the current supply / festoon systems,
- Resistance due to wind,
- Resistance due to acceleration of the rotating masses, and
- Resistance due to acceleration of the linear moving masses.

Formulations for each resistance will be provided in subsequent sections. For each gantry/trolley/hoist movement, not only the values of the variables in the formulation are different, but also the type of resistance to be considered, as indicated in Table 2.

Type of resistance	Gantry	Trolley	Hoist
Normal travel	✓	✓	✓
Current supply		✓	
Wind	✓	✓	
Rotating masses	✓	✓	✓
Acceleration of linear moving masses	✓	✓	✓

Table 22. Types of resistances to consider in each movement.

A.5.3. Resistance due to nominal traveling

The required motor torque M (in Nm) to raise the maximum nominal load is:

$$\mathbf{M_1} = \frac{\boldsymbol{P_1 \cdot 9,550}}{\boldsymbol{n}}. \qquad \textbf{(Equation 41)}$$

Where:

- n = rotating speed of the motor (in revolutions per minute, rpm)

The power (in kW) required to overcome the resistance due to nominal travel can be generally calculated according to the following expression:

$$\boldsymbol{P_1 = \frac{F_1 \cdot v}{\eta}} \qquad \textbf{(Equation 42)}$$

Where:

- P_1 = required power in kW.
- F_1 = Force or resistance due to nominal travel in kN, which is different for each movement, as described later on.
- v = Speed of the crane or its moving parts.
- η = the efficiency of the system mechanism, which can be different for each case, as indicated later on.

A.5.3.1. Nominal force F1

The calculation of the nominal force F_1 is different in each type of movement

1) Hoisting

The nominal force F_1 (in kN) is given by:

$$\boldsymbol{F_1 = N = m \cdot g} \qquad \textbf{(Equation 43)}$$

- N = maximum nominal permissible lifting force (in N)

For the hoisting mechanism shown in Figure 10 the influence of the angles α have to be taken into account, as the forces and the motor power are multiplied in this wire rope system, with

$$f = \frac{1}{cos\alpha}$$

Where:

- α = half of the biggest angle between the wire ropes when the load is in the highest position.

2) Trolley and gantry

$$\boldsymbol{F_1 = W \cdot f} \qquad \textbf{(Equation 44)}$$

Where:

- f = wheel resistance of the trolley wheels (in kN/t): 5 kg/t = 0.05 kN/t
- W = weight of the moving crane structure of the trolley system (kg)

A.5.3.2. Efficiency of the mechanism

Again, the efficiency of each mechanism is different, as described next

3) Hoist

Only the efficiency of gearings and rope sheaves need to be considered, therefore η = 0.9.

4) Trolley

a) For a full *rope* trolley or semi-rope trolley of a container crane with the hoisting mechanism on the bridge, it is given by:

$$\eta = \eta_{sh} + \eta_{gearings} = 0.85 \; to \; 0.87 \quad \textbf{(Equation 45)}$$

b) Full *motor* trolley of a crane with the hoisting winch on the trolley:

$$\eta = \eta_{gearings} = 0.90 \quad \textbf{(Equation 46)}$$

5) Gantry

In this case, the efficiency of gearings is η = 0.90.

A.5.4. Resistance due to current supply or festoon system

For this resistance, the following values are adopted:

- F_2 = 1.5 kN for a rope driven trolley
- F_2 = 3–4 kN for a motor trolley

As before, the required power is:

$$P_2 = \frac{F_2 \cdot v}{\eta} \quad \textbf{(Equation 47)}$$

The formula can use the same notation as before

A.5.5. Resistance due to wind

The force required to overcome the wind is:

$$F_3 = F_w = \sum_{i=1}^{N} A_i \cdot c_i \cdot q_i \quad \textbf{(Equation 48)}$$

Where:

- F is the wind load in N,
- A_i = effective frontal area of the part under consideration (in m^2)
- c_i = shape coefficient in the direction of the wind for the part under consideration
- q_i = wind pressure corresponding to the appropriate design condition (in N/m^2)

Height above ground level (m)	Out of service design wind pressure (N/m^2)	Approximate equivalent out of service design wind speed (m/s)
0 to 20	800	36
20 to 100	1100	42
More than 100	1300	46

Table 23. Out of service wind.

The torque needed on the motor shaft to drive the crane against the wind is:

$$M_3 = \frac{F_3 \cdot \frac{D}{2}}{i \cdot \eta} \quad \textbf{(Equation 49)}$$

Where:

- D = diameter of the driven wheel
- i = reduction of the gearings between the driven wheel and the driving motor, given by:

$$i = \frac{n_{motor}}{n_{wheel}} \quad \textbf{(Equation 50)}$$

- n_{motor} = revolutions per minute of the motor shaft
- n_{motor} = revolutions per minute of the driven wheel, given by

$$n_{wheel} = \frac{v}{\pi \cdot D} \quad \textbf{(Equation 51)}$$

- v = trolley travelling speed, given in m/min to make it coherent
- η = efficiency of the mechanism

Again the power (in kW):

$$N_3 = \frac{F_3 \cdot v}{\eta} \quad \textbf{(Equation 52)}$$

A.5.6. Resistance due to accelerating the rotating masses

In this case, the expression for the motor torque (in Nm) becomes:

$$\mathbf{M_4} = \frac{J_{rot} \cdot \omega}{t_a} \cdot \quad \textbf{(Equation 53)}$$

Where:

- J_{rot} = moment of inertia of the rotating masses (in kg m^2), including:
  - Motors
  - Break sheaves and couplings
  - Gearbox, reduced
- ω = angular speed (in radians/sec)

$$\omega = \frac{2 \cdot \pi \cdot n}{60} \cdot \quad \textbf{(Equation 54)}$$

- t_a = acceleration time (secs)
- n = motor revolutions per minute

From here, we can evaluate the power in kW:

$$\mathbf{N_4} = \frac{\boldsymbol{M_4} \cdot \mathbf{n}}{\mathbf{9550}} \quad \textbf{(Equation 55)}$$

A.5.7. Resistance due to accelerating the linear masses

The force needed to mobilize the body is:

$$\boldsymbol{F_3} = \frac{\boldsymbol{W} \cdot \boldsymbol{v}}{\boldsymbol{t_a}} \quad \textbf{(Equation 56)}$$

Where:

- W = weight of the linear moving mass, which can be a sum of the following depending on the movement (kg)
- W_L = rated lifting load
- W_{SH} = weight of spreader and the headblock (hoist)
- W_B = weight of the bridge (trolley)
- W_C = weight of the crane (gantry)
- v = movement speed with or without load (m/s)
- G = gravitational acceleration
- t_a = acceleration time (secs)

And the power:

$$\boldsymbol{P_3} = \frac{\boldsymbol{F_3} \cdot \boldsymbol{v}}{\boldsymbol{\eta}} \quad \textbf{(Equation 57)}$$

In this case, the torque M_3 can be calculated according to Equation 2.

A.6. Calculation example

For illustration purposes, if we considering a crane duty cycle composed of on gantry, one trolley and one hoist (lower) movements of a 65 ton container as part of a stacking cycle. The calculation will be then repeated considering the crane does not carry a container to illustrate the influence of the weight in the relative importance of the consumption associated to each type of movement.

A calculation example is conducted in the next subsection according to the two consumption models to illustrate the differences between them.

Movement	Gantry	Trolley	Hoist
Mass (kg)	185,000	25,000	10,000
Acceleration (m/s^2)	±0.40	±0.30	±0.6 full
Speed (m/min)	240 full	70	45 full
	270 empty		90 empty
Motor rpm	200 full	140	180 full
	225 empty		360 empty
Moment of inertia (kg m^2)	0.16	0.06	2.30
Efficiency	0.95	0.9	0.85

Table 24. Crane characteristics.

The next table summarizes the kinematics of the crane movements:

Kinematics	Gantry	Trolley	Hoist
Distances travelled (20” slots)	25	4	4
Length of the 20” slot + margin (m)	6.058 + 0,4	2.438 + 0.4	2.591 + 0.0
Acceleration time (s)	10.0 full	3.89	1.25 full
Total movement time (s)	60.36 full	13.62	15.07 full

Table 25. Characteristics of the example movement.

A.6.1. Potential model

The next table summarizes the results from the example:

Resistance	Gantry	Trolley	Hoist
Loaded movements	E_{1G} = 0.522 kWh	E_{1T} = 0.015 kWh	E_{1H} = 1.800 kWh
Relative consumption	22.3%	0.6%	77.0%
Unloaded movements	E_1 = 0.387 kWh	E_1 = 0.004 kWh	E_{1H} = 0.240 kWh
Relative consumption	61.3%	0.7%	38.1%

Table 26. Energy consumption calculation.

As shown in the table, hoist movements consume comparatively more energy than trolley or gantry when the crane is loaded. On the other hand, when no container is being moved, gantry can be proportionally higher than hoist, although in the end the total consumption will depend on both the distances traveled by the crane and the number of movements of each kind.

A.6.2. Electric model

Results for the electric model are provided next.

A.6.2.1. Laden crane

Resistance	Gantry	Trolley	Hoist
Nominal hoisting (full load at maximum speed)	P_{1G} = 51.6 kW M_{1G} = 2,462.9 Nm E_{1G} = 0.736 kWh	P_{1T} = 5.5 kW M_{1T} = 375.8 Nm E_{1T} = 0.022 kWh	P_{1H} = 605.9 kW M_{1H} = 32,147.0 Nm E_{1H} = 2.717 kWh
Current supply	-	P_{2T} = 3.9 kW M_{2T} = 265.3 Nm E_{2T} = 0.015 kWh	-
Wind	-	-	-
Accelerating the rotating masses	ω = 20.94 rads/s M_{4G} = 4.2 Nm P_{4G} = 0.1 kW E_{4G} = 0.000 kWh	ω = 14.66 rads/s M_{4T} = 30.2 Nm P_{4T} = 0.4 kW E_{4T} = 0.000 kWh	ω = 18.85 rads/s M_{4H} = 693.7 Nm P_{4H} = 13.1 kW E_{4H} = 0.005 kWh
Accelerating the linear masses	F_{5G} = 98.0 kN P_{5G} = 412.6 kW M_{5G} = 19,703.2 Nm E_{5G} = 1.170 kWh	F_{5T} = 25.5 kN P_{5T} = 33,1 kW M_{5T} = 2,254.9 Nm E_{5T} = 0,038 kWh	F_{5H} = 42.0 kN P_{5H} = 37.1 kW M_{5H} = 1,966.2 Nm E_{5H} = 0.014 kWh
Total energy consumption	**E_{TG} = 1.906 kWh**	**E_{TT} = 0.075 kWh**	**E_{TH} = 2.736 kWh**
Relative consumption	40.4%	1.6%	58.0%

Table 27. Hoist movement power calculation for a loaded crane.

A.6.2.2. Unladen crane

The same calculation can be readily done for the crane with no container load.

Resistance	Gantry	Trolley	Hoist
Total energy consumption	**E_{TG} = 1.517 kWh**	**E_{TT} = 0.032 kWh**	**E_{TH} = 0.354 kWh**
Relative consumption	79.7%	1.7%	18.6%

Table 28. Hoist movement power calculation for the crane unloaded.

A.7. Conclusions

A brief comparison between both models and their results lead to some immediate conclusions:

- The electric model is more sophisticated than the potential model, as it takes into consideration physical aspects of the mechanisms and the duration of each resistance to the movement.
- Compared to the potential model, the electric model produces higher values of electric consumption, as expected.

- For both models, when the crane is loaded, the consumption associated to hoist is more important than that to the gantry; conversely, when no container is present, the gantry consumption is more important than that to the hoist. As a consequence, when considering the energy consumption associated to a container, the proportion associated to gantry and hoist movements will depend on each particular duty cycle.
- The electric model produces greater values for gantry movements compared to hoist movements.

Annex B

Parallel Terminal model code

The Matlab code for the Parallel Terminal DES Model used throughout this Thesis is provided here.

```
function [P,n] = addposition(P,move,nP)

% copy previous position
%keyboard
m = length(P.bay); n = m+1;
P.bay(n) = P.bay(m);
P.stack(n) = P.stack(m);
P.tier(n) = P.tier(m);
P.time(n) = P.time(m);

switch char(move)
    case 'gantry'
        P.bay(n) = nP;
    case 'trolley'
        P.stack(n) = nP;
    case 'hoist'
        P.tier(n) = nP;
end
```

```
function [asc] = ASC_move(asc,newposition,ct,writing)
% This function changes the position of the YC

global ASC

[pos]= length(asc.position.time);
%t_act = asc.tasks.time(asc.tasks.no);
t_act = asc.position.time(end);
t_min = min(asc.position.time(2:end));
if t_min ==0
   disp('Position Error')
   %keyboard
elseif newposition.time == 0
   disp('Position Error')
   %keyboard
elseif t_act > newposition.time
   disp('Position Error')
   t_act - newposition.time
   plot_ASC_trajectories(10)
   %keyboard
end

asc.position.bay(pos+1) = newposition.bay;
asc.position.stack(pos+1) = newposition.stack;
asc.position.tier(pos+1) = newposition.tier;
asc.position.time(pos+1) = newposition.time;
asc.position.ct(pos+1) = ct;

if writing ==1
   %keyboard
   switch char(asc.id)
      case 'sea'
         ASC.sea.position.bay(pos+1) = newposition.bay;
         ASC.sea.position.stack(pos+1) = newposition.stack;
         ASC.sea.position.tier(pos+1) = newposition.tier;
         ASC.sea.position.time(pos+1) = newposition.time;
         ASC.sea.position.ct(pos+1) = ct;
      case 'land'
         ASC.land.position.bay(pos+1) = newposition.bay;
         ASC.land.position.stack(pos+1) = newposition.stack;
         ASC.land.position.tier(pos+1) = newposition.tier;
         ASC.land.position.time(pos+1) = newposition.time;
         ASC.land.position.ct(pos+1) = ct;
   end
end

check_pos =1;
if check_pos ==1
   LP = ASC_act_pos(ASC.land);
   SP = ASC_act_pos(ASC.sea);

   if LP.bay <= SP.bay
      disp('Warning: there may be crane intersection')
```

```
        %plot_ASC_trajectories(25)
        %keyboard
        %old_ASC_interference()
    end
end
```

```
function BAY_change_reservation(bay,port,no_cts)
% This function adds or remove reservations in a bay

global BAYS  BL

%keyboard

if no_cts>0

      BAYS(bay).R.slots = BAYS(bay).R.slots + no_cts;
      oldports = BAYS(bay).R.ports;
      newports = ones(1,no_cts)*port;

      if oldports == 0
         ports = newports;
      else
         ports = sort([oldports newports]);
      end
      BAYS(bay).R.ports = ports;
      if BAYS(bay).R.slots > BL.capacity -BL.tiers
         disp(['Bay(' num2str(bay) ') Overloaded'])
         BAYS(bay).R
         keyboard
      end
elseif no_cts<0
   pos = find(BAYS(bay).R.ports == port);
   if isempty(pos)
      disp('Warning: Trying to remove a reservation not present in the bay')
      %keyboard
   else
      BAYS(bay).R.slots = BAYS(bay).R.slots + no_cts;
      BAYS(bay).R.ports(pos(end)) = [];
   end
end

if abs(BAYS(bay).R.slots -length(BAYS(bay).R.ports)) >0
  'Error when reservating slots'
  keyboard
end

if sum(sum(BAYS(bay).matriz)) > BAYS(bay).R.slots
   disp([BAYS(bay).id 'BAY(' num2str(bay) 'has more cts than reservations'])
   keyboard
end
```

```
function [c_bay_found] = BAY_check_id(BAY,mix_id,VS)

global BL
c_bay_found = 'Y';
switch mix_id
   case 'U'
      disp('error')
      keyboard
   case 'N' % Bays with a single port
      [positions, empty_ports] = Port_empty_slots(BAY); % return the # of Port empty slots and positions i,j
      port_id = sum(sum(BAY.port))/(BL.capacity - empty_ports);
      if port_id ~= VS.port
         c_bay_found = 'N';
      end
   case 'S' % Bays with port CTs arranged in stacks
      if BAY.mixing ~= 'S'
         c_bay_found = 'N';
      end

   case 'Y' % bays with
      if BAY.mixing ~= 'Y'
         c_bay_found = 'N';
      end
end
```

```
function [fs,bs] = BAY_ES(bay,ct_stack)
% ths function calculates the number of free slots and reserved slots

global BAYS BL

% Calculate the number of free slots
%keyboard
fs = zeros(1,BL.stacks);
os = zeros(1,BL.stacks);
bs = zeros(1,BL.stacks);
for stack = 1: BL.stacks
   if stack ~= ct_stack
      for tier = 1:BL.tiers
         if BAYS(bay).port(tier,stack) > 0
            if BAYS(bay).matriz(tier,stack)  == 0
               bs(stack) = bs(stack) + 1;
            else
               os(stack) = os(stack) + 1;
            end
         else
            fs(stack) = fs(stack) + 1;
         end
      end
   end
end
```

```
function [cts_cols,e_cols] = BAY_find_port_col(BAY,port)
% find empty slots in columns with port reserved
global BL

e_cols = 0; cts_cols =0;

for col = 1: BL.stacks
   sum_ports = sum(BAY.port(:,col));
   sum_cts = sum(BAY.matriz(:,col));
   if and(sum_ports > 0, sum_cts < BL.tiers)
      heights = 0;
      for tier = 1:BL.tiers
         if BAY.port(tier,col) > 0
            heights = heights +1;
         end
      end
      column_port = sum_ports/heights;

      if column_port == port
         e_cols= e_cols + 1;
         cts_cols(e_cols) = col;
      end
   end
end
```

```
function [col_ids,no_cols] = BAY_find_Port_empty_col(BAY)
% This function calculates the empty colums

[rows,cols]= size(BAY.port);

no_cols = 0; col_ids =0;

for col = 1: cols
    if sum(BAY.port(:,col)) == 0
        no_cols= no_cols + 1;
        col_ids(no_cols) = col;
    end
end

if no_cols == 0
   disp('BAY_find_empty_col Error: no empty staks')
   BAY
end
```

```
function [ports] = BAY_find_Port_reserved(BAY)
% This funciton calculates the number of reserved spaces for all the ports
% we search for slots reserved and but not having a CT
global BL TRF
ports = zeros(1,TRF.PARAM.no_ports);

for tier = 1:BL.tiers
   for stack = 1:BL.stacks
      bay_port = (BAY.port(tier,stack))*(1 - BAY.matriz(tier,stack));
      if bay_port > 0
         ports(bay_port) = ports(bay_port) + 1;
      end
   end
end
```

```
function [ports] = BAY_find_reservations(bay,bos)
% This function counts the number of slots with containers assigned to each port

global BAYS BL CT TRF

% Search bay reservations
ports =zeros(1,TRF.PARAM.no_ports);

if strcmp(bos,'B') == 1
   for p=1:BAYS(bay).R.slots
      bay_port = BAYS(bay).R.ports(p);
      if bay_port >0
         ports(bay_port) = ports(bay_port) + 1;
      end
   end
elseif strcmp(bos,'S')  == 1
   for s=1:BL.stacks
      for t=1:BL.tiers
         ct = BAYS(bay).R.S.cts(t,s);
         if ct >0
            p = CT(ct).vs;
            ports(p)= ports(p)+1;
         end
      end
   end
elseif strcmp(bos,'C')  == 1
   for s=1:BL.stacks
      for t=1:BL.tiers
         ct = BAYS(bay).ct_id(t,s);
         if ct >0
            p = CT(ct).vs;
            ports(p)= ports(p)+1;
         end
      end
   end
end
```

```
function [positions,no_ceros] = BAY_find_slots(BAY,flow)
% This function gives the number of empty ports

global BL T

positions = 0;
no_ceros = 0;

if strcmp(flow,'EXP') == 1
   max_h = T.stack.exp;
elseif strcmp(flow,'IMP') == 1
   max_h = T.stack.imp;
end

for stack = 1:BL.stacks
   pile = BAY.matriz(:,stack);
   if sum(pile) < max_h
      for tier = 1:BL.tiers %max_h
         if BAY.port(tier,stack) == 0
            no_ceros = no_ceros + 1;
            positions(no_ceros,1) = tier;
            positions(no_ceros,2) = stack;
         end
      end
   end
end
%keyboard
no_ceros = min(no_ceros,20);
positions = positions(1:no_ceros,:);
```

```
function [list, i] = BAY_get_port(bay,vs)
% This function gives a list of the ports of the containers within it

global BL CT BAYS

list = 0;
% pos = find(BAYS(bay).ct_id>0);
% ctlist = BAYS(bay).ct_id(pos);
i = 0;
for s = 1:BL.stacks
   for t = BL.tiers:-1:1
      ct = BAYS(bay).ct_id(t,s);
      if ct  >0
         if CT(ct).vs == vs
            i = i +1;
            list(i) = ct;
            wlist(i) = CT(ct).class;
         end
      end
   end
end

% for i = 1:length(ctlist)
%     if CT(ctlist(i)).vs == vs
%         ncvs = ncvs+1;
%         list(ncvs) = ctlist(i);
%     end
% end
%
% for i = 1:length(list)
%     wlist(i) = CT(list(i)).class;
% end
if i > 0
   [wlist,ord] = sort(wlist,'descend');
   list = list(ord);
end
```

```
function [tier,stack] = BAY_individual_allocation(ct)
% This function finds a slot within the matrix to place a CT
% It will be replaced with an algorithm to take into account CT weights

global BAYS BL CT TIME VS

%keyboard

stack = 0; tier = 0; bay = CT(ct).P.bay; vs = CT(ct).vs;

old_bay = BAYS(bay);
check_bay_es(bay);

for s = 1:BL.stacks
   %altura_pila(s) = Pile_height(bay,s,'M');
   a = find(BAYS(bay).ct_id(:,s)>0 );
   h(s)=length(a);

   if strcmp(CT(ct).type,'IMP') == 1
      if h(s) == BL.tiers
         peso(s) = TIME.simul*1.1*BL.tiers;
      else
         % NOT mixing
         if strcmp(BAYS(bay).mixing, 'N')==1
            nc = length(find(BAYS(bay).ct_id(:,s)>0));
            peso(s) = nc*TIME.t - sum(BAYS(bay).ct_arrival(:,s));
         % STACK MIXING
         else
            low_ct = BAYS(bay).ct_id(1,s);
            if low_ct == 0
               peso(s) = 0;
            else
               if CT(low_ct).vs ~= vs
                  peso(s) = 2*(h(s)*TIME.t-sum(BAYS(bay).ct_arrival(:,s))); %TIME.simul*1.1*BL.tiers;
               else
                  peso(s) = h(s)*TIME.t-sum(BAYS(bay).ct_arrival(:,s));
               end
            end
         end
      end
   elseif strcmp(CT(ct).type,'EXP') == 1
      if h(s) == BL.tiers
         peso(s) = 10000000;
      else
         % NOT mixing
         if strcmp(BAYS(bay).mixing, 'N')==1
            peso(s) = abs(BL.idealbay(h(s)+1,s)- CT(ct).class);
         % STACK MIXING
         else
            low_ct = BAYS(bay).ct_id(1,s);
            if low_ct == 0
               peso(s) = abs(BL.idealbay(h(s)+1,s)- CT(ct).class);
            else
```

```
                if CT(low_ct).vs ~= vs
                   peso(s) = 1000000;
                else
                   peso(s) = abs(BL.idealbay(h(s)+1,s)- CT(ct).class);
                end
             end
          end
       end
    end
end

[weight_dif,stack] = min(peso);
if weight_dif == 10000000
    keyboard
else
    tier = h(stack)+1;
    %tier = Pile_height(bay,stack,'P')+1;
    if tier > BL.tiers
       BAYS(bay).ct_id
       keyboard
    elseif tier==0
       keyboard
    end
end

% INDIVIDUAL SLOT ALLOCATION
disp([char(CT(ct).type) ' CT(' num2str(ct) ') Allocated in BAY(' num2str(bay) ')'])
CT(ct).P.tier = tier; CT(ct).P.stack = stack;

bay = CT(ct).P.bay;
BAYS(bay).matriz(tier,stack) = 1;
BAYS(bay).ct_id(tier,stack) = ct;
BAYS(bay).ct_arrival(tier,stack) = TIME.t;
% Empty slots are accounted for when the crane calculates the cycle for the CT
BAYS(bay).empty_slots = BAYS(bay).empty_slots - 1;

CT_addevent(ct,'stacked',0);

if strcmp(BAYS(bay).id,'NAS') == 1
    BAYS(bay).id = CT(ct).type;
end

% Check bay
%--------------------------------------------------------------------------
[ports] = BAY_find_reservations(bay,'C');
if BAYS(bay).R.slots < ports(vs)>0
    keyboard
end

if strcmp(BAYS(bay).mixing,'U') == 1
    keyboard
end

check_bay_es(CT(ct).P.bay);
```

```
if BAYS(bay).empty_slots < BL.tiers
    terminal_state
    if strcmp(CT(ct).type,'IMP')==1
        imp_bays= zeros(1,length(VS(vs).plan.IMP.bays));
        for i = 1:length(VS(vs).plan.IMP.bays)
            b = VS(vs).plan.IMP.usedbays(i);
            imp_bays(b) = imp_bays(b) +1;
        end
    end
    disp(['Too much occupation in ' BAYS(bay).id ' Bay(' num2str(bay) ')']); keyboard
end

check_port_occupation(bay);
```

```
function [iscomplete,C_slots,CB_ISD] = BAY_mixing_stack(bay_id,VS,slots_needed)
% This function searches for emtpy stacks in the terminal bays
% CB means Candidate Bays

global BAYS BL BT T

% if strcmp(bay_id,'IMP')== 1
%     keyboard
% end
nb = 0; CB_ISD = zeros(1,3);
max_ocup = BL.capacity - BL.tiers;
% Search bays with empty stacks
for bay=1:T.bays
   if BAYS(bay).R.slots == max_ocup
      continue
   end
   if BAYS(bay).id == bay_id % EXP, IMP or NAS
      % If there are emtpy stacks
      e_cols = length(find(sum(BAYS(bay).ct_id(:,:)) == 0));

      %free_slots = floor((max_ocup - BAYS(bay).R.slots)/BL.tiers)*BL.tiers;
      free_slots = max_ocup - BAYS(bay).R.slots;
      if free_slots > 0 %and(isempty(e_cols) == 0, free_slots >0)
         nb = nb + 1;
         CB_ISD(nb,1) = bay;
         CB_ISD(nb,2) = free_slots; %min(length(e_cols)* BL.tiers,free_slots);
         CB_ISD(nb,3) = distance_calculator(BT(VS.berth).position,BAYS(bay).position);
      end
   end
end

available_slots = sum(CB_ISD(:,2));
C_slots = min(slots_needed, available_slots);

if available_slots >= slots_needed
   iscomplete = true;
else
   iscomplete = false;
end
```

```
function [iscomplete,C_slots,CB_ISD] = BAY_mixing_total(bay_id,VS,slots_needed)
% This function calculates the bays that can have mixing in them
% There is only one criteria: port matrix has to have zeros
% CB means Candidate Bays

global BAYS BL BT T

nb = 0;

ids = 0; empty_slots = 0; distances = 0;
C_slots =0; CB_ISD = [0 0 1000000]; available_slots =0;

% Search bays with empty slots
for bay = 1:T.bays
   % Compare bay ids to search for export bays
   if BAYS(bay).id == bay_id
      % This calculates the bays with zeros in port matrix
      %[positions,no_ceros] = Port_empty_slots(BAYS(bay));
      no_ceros = BL.tiers*BL.stacks - BAYS(bay).R.slots;
      if no_ceros > BL.tiers
         nb = nb + 1;
         CB_ISD(nb,1) = bay;
         CB_ISD(nb,2) = no_ceros;
         CB_ISD(nb,3) = distance_calculator(BT(VS.berth).position,BAYS(bay).position);
      end
   end
end

if nb > 0
   available_slots = sum(CB_ISD(:,2));
   C_slots = min(slots_needed, available_slots);
end
if available_slots >= slots_needed
   iscomplete = true;
else
   iscomplete = false;
end
```

```
function BAY_remarshall(target_bay,s_needed)

global BAYS BL COST CT S YC

% 1. Find the YC to get the asssignment

slots_liberated = 0;
i_stack = 0;
old_bay = BAYS(target_bay);

% YC initial position and update the workload of the YC
[target_yc] = YC_select(target_bay);
xo = YC(target_yc).P.bay;
yo = YC(target_yc).P.stack;

% Initialize the work adding the initial gantry movement
%work = zeros(1,BL.stacks) + (xo - BAYS(target_bay).position(1)) * S.l * YC_consumption('G','U',0);
work = zeros(BL.stacks,2);
ycwork = zeros(BL.stacks,1);

% Calculate the number of stacks that will be moved.
no_cts = BL.capacity - BAYS(target_bay).empty_slots;
work(:,1) =  ST_work(BAYS(target_bay));
work(:,2) = 1:BL.stacks;
sortrows(work,1);

while i_stack < s_needed
   keyboard
   stack_cts = sum(BAYS(target_bay).matriz(:,:));
   i_stack = i_stack + 1;
   % evaluate the work
   work =  ST_work(target_bay);
   [min_work,g_stack] = min(work);
   % Sort the YC required work in descending order
   % work = sortrows(work,1);
   % Perform the redistribution operation

   cstack_cts = stack_cts;
   for h = 1:stack_cts(g_stack)
      % Identify the best position for the CT
      ct_h = stack_cts(g_stack) - h +1;
      %g_tier = BL.tiers - ct_h +1;
      g_tier = ct_h;
      ct_id = BAYS(target_bay).ct_id(g_tier,g_stack);
      c_wc = 100*ones(1,BL.stacks);

      for r_stack = 1:BL.stacks
         %top_tier = BL.tiers - cstack_cts(r_stack);
         top_tier = cstack_cts(r_stack);
         if and(BAYS(target_bay).matriz(top_tier,r_stack) == 0, r_stack ~= g_stack)
            c_wc(r_stack) = abs(BL.idealbay(top_tier,r_stack) - CT(ct_id).class);
         end
      end
```

```
      % The best position is the one with less difference
      % Among the possible candidates, choose the one with higher
      [wmin,r_stack] = min(c_wc);
      if r_stack < BL.stacks
         for s = r_stack+1:BL.stacks
            if wmin == c_wc(s)
               if and(cstack_cts(s) > cstack_cts(r_stack), cstack_cts(s)<BL.tiers)
                  r_stack = s;
               end
            end
         end
      end
      r_tier = BL.tiers - cstack_cts(r_stack);

      % Compute the effort
      port = BAYS(target_bay).port(g_tier,g_stack);

      CT_drop(ct_id);
      %target_bay,g_tier,g_stack
      CT_remove(ct_id);
      % Update stacks
      cstack_cts = sum(BAYS(target_bay).matriz(:,:));
      % go to slot
      ycwork(g_stack) = ycwork(g_stack) + YC_consumption('H','L',ct_id)*(BL.tiers+1-g_tier)*S.h +
YC_consumption('S','L',ct_id)*abs(r_stack-g_stack)*S.w;
   end
   % need to reevaluate the work
end

disp('Remarshalling ended')

COST.remarshall = COST.remarshall + slots_liberated;
```

```
function [cts_reserved,t_cts,VS] = BAY_reservation(cts_left,t_cts,bay,mix_id,VS,label,free_cols)
% This function puts label and port on a given bay

global BAYS BL T
%keyboard

if strcmp(label,'EXP') == 1
   max_h = T.stack.exp;
elseif strcmp(label,'IMP') == 1
   max_h = T.stack.imp;
end

% 1. Calculate the number of bay slots that will be reserved for a port
% ---------------------------------------------------------------------
max_ocup = BL.capacity - BL.tiers;
free_slots =  max_ocup - BAYS(bay).R.slots;
%if strcmp(mix_id, 'S') == 1
   %available_slots = free_cols*BL.tiers;
   %cts_reservable = min(free_slots, available_slots);
   %used_cols = ceil(cts_reserved/max_h);
%else % Strategy 'N' or 'Y'
   %[positions,no_ceros] = Port_empty_slots(BAYS(target_bay));
   %[positions,no_ceros] = BAY_find_slots(BAYS(target_bay),label);
   cts_reservable = min(free_slots,max_ocup);
%end

cts_reserved = min(cts_left,cts_reservable);
t_cts = t_cts + cts_reserved;

% 2. Port assignation to the slots
% --------------------------------------------------------------------
if BAYS(bay).mixing == 'S'
   disp(['Overwriting Stack reserved matrix. Bay ' num2str(BAYS(bay).no)])
end
BAYS(bay).mixing = mix_id;

BAY_change_reservation(bay,VS.no,cts_reserved);

BAYS(bay).id = label;
```

```
function [R,BAY,solution] = BAY_reshuffles(yc,ct)
% Given a bay "BAY" and a ct "ct", this function calculates the reshuffles
% needed to move the containers on top of ct whithin the same bay
global BAYS CT

pos = CT(ct).P;
bay =  CT(ct).P.bay;

R = 0;
%R.time = 0; R.E = 0; R.moves = ''; R.bay = 0; R.stack = 0; R.tier = 0; R.ct = 0;
solution = 0;
%pos = CT_act_pos(t_ct);
no_reshuffles = sum(BAYS(bay).matriz(:,pos.stack))-CT(ct).P.tier;

%keyboard
% Calculates the emtpy and stack reserved slots in the bay
%[fslots,rslots] =  BAY_ES(pos.bay,pos.stack);
%keyboard
fslots = BAYS(bay).empty_slots;
h = sum(BAYS(bay).matriz(:,pos.stack)); % no cts on the target stack
%keyboard
BAY=BAYS(bay).ct_id;
if sum(fslots) >= no_reshuffles
   %keyboard
   CT(ct).R.provocked.no = no_reshuffles;
   for r = 1:no_reshuffles
      tier = h - r + 1;
      rct = BAY(tier,pos.stack);
      CT(rct).R.suffered.no = CT(rct).R.suffered.no + 1;
%        if CT(ct).cnr > 4
%            keyboard
%        end
      %[BAY,ETM] = CT_reshuffle(tier,pos.stack,BAY,yc,W);
      if BAYS(bay).ct_id(tier,pos.stack) == 0
         keyboard
      end
      [BAY,ETM,T] = CT_reshuffle1(tier,pos.stack,BAY,yc);
      CT(ct).R.provocked.T(r).tier = T.tier;
      CT(ct).R.provocked.T(r).stack = T.stack;
      CT(ct).R.provocked.T(r).ct = rct;
      if r ==1
         R=ETM;
      else
         R.time = [R.time,ETM.time];
         R.E = [R.E,ETM.E];
         R.moves = [R.moves,ETM.moves];
         R.bay = [R.bay,ETM.bay];
         R.stack = [R.stack,ETM.stack];
         R.tier = [R.tier,ETM.tier];
         R.ct = [R.ct,ETM.ct];
         R.ctmove= [R.ctmove,ETM.ctmove];
      end
   end
```

```
    solution = 1;
else
    disp(num2str(BAYS(bay).ct_id))
    disp('BAY Reshuffles warning: No space in bay for Reshuffles. CTs should be moved to other bays');
    keyboard
end
```

```
function [col,row]= select_slot(bay,idealbay,c_class,c_weight)

% This function takes a vector of weightclass and places it on the best
% possible row and column

% Initialize

[no_tiers,no_rows]=size(idealbay);

diference=zeros(1,no_rows);
toptier=zeros(1,no_rows);
for row=no_rows:-1:1 % buscar la col en la que mejor encaja
   %row;
   tier=no_tiers;

   while bay(tier,row)>0 && tier ~= 1
      if tier>1
         tier=tier-1;
      end
   end
   if tier==1 && bay(tier,row) >0
      diference(row)=100;
   else
      diference(row)=idealbay(tier,row)-c_class(cont);
   end
   toptier(row)=tier;
end
%now choose the position with the smallest difference
% this function will be further improved
[j,min_row]=min(abs(diference));
j=sort(abs(diference));
i=2;    equals=1;
while i<no_rows
   if j(i)==j(1)
      equals=equals+1;
   end
   i=i+1;
end
if equals>1
   'There is more than one option'
   % Now find the alternative with lower tier'
end
```

```
function [CB_slots]=BAY_search_esl_strategy(BAY,mix_id,VS)
% This function calculates empty slots of a bay depending on the mix strategy
global BL

CB_slots = 0;

if or (mix_id == 'N', mix_id == 'Y')
    for col = 1:BL.stacks
        for row = 1:BL.tiers
            if BAY.port(row,col) == 0
                CB_slots = CB_slots + 1;
            end
        end
    end
elseif 'S'
    % Find an empty stacks in the BAY
    [target_col,e_cols] = Port_empty_cols(BAY);
    % Change the bay port
    CB_slots = e_cols * BL.tiers;
end
```

```
function [slots]=BAY_search_ports_strategy(BAY,mix_id,port)
% This function searches port and empty slots at the same time
global BL

slots = 0;

if  or(mix_id=='N',mix_id=='Y')
      for tier =1:BL.tiers
         for stack = 1:BL.stacks
            if and(BAY.port(tier,stack) == port,BAY.matriz(tier,stack) == 0)
               slots = slots + 1;
            end
         end
      end
elseif mix_id == 'S'
      % Find an empty stack in the BAY
      [target_col,e_cols] = BAY_find_port_col(BAY,port);
      % Change the bay port
      slots = e_cols * BL.tiers;
end
```

```
function BAY_selection_NR(ct,CB,stackmode)
% This function selects a bay among a list with two criteria:
%NO RESERVATION
global BAYS BL COUNT CT T TRF VS

arrived_cts = COUNT.inventory.exp(CT(ct).vs);
b_slots = 0;
nb = length(CB);
vs = CT(ct).vs;

if strcmp(CT(ct).type,'IMP') == 1
   ocup_limit = T.limits.bay.imp;
elseif strcmp(CT(ct).type,'EXP') == 1
   ocup_limit = T.limits.bay.exp;
end

% 1. EVALUATE THE BAYS
for i_bay = 1:nb
   bay = CB(i_bay);
   slots = 0;
   peso = ones(1,BL.stacks)*100;
   if strcmp(stackmode,'S') == 1
      %keyboard
      ocup = sum(BAYS(bay).matriz);
      positions = find(ocup==0);
      if isempty(positions)==1
         slots = 0;
      else
         %lp = length(positions);
         slots = length(positions)*BL.tiers;
         available_slots = BAYS(bay).R.slots-BL.tiers;
         if available_slots > 0
            slots = min(slots,available_slots);
         else
            slots = 0;
         end
         peso(positions) = BL.idealbay(1,positions);
      end

   else %if strcmp(stackmode,'N')==1
      % Find the ports already there
      %[ports] = BAY_find_reservations(bay,'C');
      ocup = BL.capacity - BAYS(bay).empty_slots;
      %rp = find(BAYS(bay).R.S.cts > 0); r = length(rp);
      %if ocup < ocup_limit
      %if BAYS(bay).R.B.slots <= ocup_limit
         if BL.capacity - BAYS(bay).empty_slots < ocup_limit
            for stack = 1:BL.stacks
               altura_pila = Pile_height(bay,stack);
               if altura_pila < BL.tiers
                  peso(stack) = BL.idealbay(altura_pila+1,stack);
                  slots = slots + BL.tiers - altura_pila;
               end
```

```
            end
          end
        %end
    end
    slots(i_bay) = min(slots, BL.capacity-BL.tiers);
    [weight_dif(i_bay),position(i_bay)] = min(abs(peso - CT(ct).class));
end

% 2. SELECT THE BAY WITH MINIMUM COEFICIENT
if strcmp(TRF.PARAM.idealbay_option,'MIN_OCCUPATION') == 1
    [empty_slots,bay_index] = max(b_slots);
     target_bay = CB(bay_index);
elseif strcmp(TRF.PARAM.idealbay_option,'PESO') == 1
    [mincoef,t_bay] = min(weight_dif);
    bay = CB(t_bay);
end

% 3. MARK THE BAY AND THE CT
CT(ct).P.bay = bay;

if BAYS(bay).empty_slots<=BL.tiers
    keyboard
end

if strcmp(BAYS(bay).id,'NAS')==1
    BAYS(bay).id = 'EXP';
end

% Change the reservation status of the bay

if strcmp(stackmode,'S') == 1
    BAYS(bay).mixing = 'S';
end

BAY_change_reservation(bay,CT(ct).vs,1);

disp([char(CT(ct).type) ' CT(' num2str(ct) ') BAY(' num2str(bay) ') Selected. # Bay reservations '
num2str(BAYS(bay).R.slots)])

check_bay_es(bay);

check_port_occupation(bay);
```

```
function [bay] = BAY_selection_R(plan,ct)
% This function searches for a candidate bay among the list
global BAYS BL CT TIME YC T


CB = plan.bays; ncb = length(CB);
vs = CT(ct).vs; vsocup=zeros(1,ncb);
m1 = 10000; WLP = m1*ones(1,ncb);
m2 = 1000000000; TT = m2*ones(1,ncb);

for i_bay = 1:ncb
   bay = CB(i_bay);
   R(i_bay) = BAYS(bay).R.slots;
   O(i_bay) = BL.capacity - BAYS(bay).empty_slots;

   % a) Now check if slots reserved have been occupied
   pos = find(BAYS(bay).ct_id>0); ctlist = BAYS(bay).ct_id(pos);

   vslist = 0;
   for i = 1: length(ctlist)
      vslist(i) = CT(ctlist(i)).vs;
   end
   vsocup(i_bay) = length(find(vslist == vs));
   othervsocup(i_bay) = length(vslist)-vsocup(i_bay);
   if O(i_bay) > BL.capacity - BL.tiers
      keyboard
   end
   if vsocup(i_bay) >= plan.cts(i_bay);
      continue
   end

   % b)If too much occupation, disregad
   if O(i_bay) >= BL.capacity - BL.tiers
      continue
   end

   % c) else: Calculate the workload
   [target_yc] = YC_assign_ct(bay);
   crane(i_bay) = target_yc;
   WLP(i_bay) = YC(target_yc).WL.cwl;
   %WLN(i_bay) = YC(target_yc).WL.normal.n;
   % Calculate the number of time
   ctsinbay = length(find(BAYS(bay).ct_arrival>0));
   TT(i_bay) = TIME.t*ctsinbay-sum(sum(BAYS(bay).ct_arrival));
end

% 3. Select a bay according to the given method
metodo = 2;
if metodo == 1     % Option 1) randomly
   keyboard
   i_bay = 1 + fix(random('unif',0,1) * bay);
   bay = CB(i_bay);
elseif metodo == 2 % Option 2) Usin weight criteria
```

```
   %
   if sum(TT) == 0
      mtt = 1;%keyboard
   else
      mtt = max(TT);
   end
   baycoefs = 0.5*WLP/25 +0.5*TT/mtt;
   maxval   = 0.5*m1/25  +0.5*m2/mtt;
   if max(baycoefs) == min(baycoefs)
      if max(baycoefs) == maxval
         figure; subplot(2,1,1); stem(plan.bays,R,'bo'); hold on; plot(plan.bays,O,'go');
         axis([0 T.bays 0 BL.capacity]);title('Occupation vs. total reservation')
         subplot(2,1,2); stem(plan.bays,plan.cts,'b.'); hold on; plot(plan.bays,vsocup,'ro');
         axis([0 T.bays 0 BL.capacity]); title('Vessel Occupation vs. Vessel reservation')
         disp([CT(ct).type ' BAYs select: All the coeficients are the same'])
         keyboard
      end
   end
   [peso_min,i_bay] = min(baycoefs);
   bay = CB(i_bay);
   if bay > 0
      CT(ct).P.bay = bay;
   else
      keyboard
   end
end

disp([char(CT(ct).type) ' CT(' num2str(ct) ') BAY(' num2str(bay) ')'])

% Check the number of reservations is not surpassed
   if vsocup(i_bay) >= plan.cts(i_bay);
      keyboard
   end

if BAYS(bay).R.slots <= sum(sum(BAYS(bay).matriz))
   keyboard
end

% Check that the reservation is ok
if BAYS(bay).R.slots < BL.capacity - BAYS(bay).empty_slots
   for i=1:length(CB)
      d(i)=BAYS(CB(i)).R.slots;
   end
   disp('Warning searching for bay: highly occupated bay')
end

% Check that occupation is ok
if sum(sum(BAYS(bay).matriz)) > BL.capacity - BL.tiers
   disp(['Choosing bay ' num2str(bay) ' with too ocupied']); keyboard
end
```

```
function BAY_unreserve(port)

global BAYS BL T
for bay = 1:T.bays
   for tier = 1:BL.tiers
      for stack = 1:BL.stacks
         if BAYS(bay).port(tier,stack) == port
            BAYS(bay).port(tier,stack) =  BAYS(bay).port(tier,stack) .* BAYS(bay).matriz(tier,stack);
            BAYS(bay).vs(tier,stack) =  BAYS(bay).vs(tier,stack) .* BAYS(bay).matriz(tier,stack);
         end
      end
   end
end
```

```
function [cts] = BAY_wlist(bay,vs)

global BAYS BL CT

clist=zeros(1,3);
i = 0;
for s=1:BL.stacks
   for t=BL.tiers:-1:1
      ct = BAYS(bay).ct_id(t,s);
      if ct >0
         if CT(ct).vs == vs
            i = i+1;
            clist(i,1) = BAYS(bay).ct_id(t,s);
            clist(i,2) = s;
            clist(i,3) = t;
            clist(i,4) = CT(ct).class;
         end
      end
   end
end

clist = sortrows(clist,-4);
cts = clist(:,1);
```

```
function BAYS_check_empty(bay)
% This function checks whether a bay is emtpy

global BAYS BL


if BAYS(bay).R.slots == 0
   %keyboard
   if BAYS(bay).empty_slots > BL.capacity
      disp('Uncomplete bay empty')
      keyboard
   end
   if sum(sum(BAYS(bay).ct_id(:,:)))>0
      disp('Uncomplete bay empty')
      keyboard
   end
%    if sum(sum(BAYS(bay).weightc(:,:))) >0
%       disp('Uncomplete bay empty')
%       keyboard
%    end
   if sum(sum(BAYS(bay).ct_arrival(:,:))) > 0
      disp('Uncomplete bay empty')
      keyboard
   end
   if sum(sum(BAYS(bay).matriz(:,:))) > 0
      disp('Uncomplete bay empty')
      keyboard
   end

   disp(['The ' num2str(BAYS(bay).id) ' bay ' num2str(BAYS(bay).no) ' has been emptied'])

   BAYS(bay).id = 'NAS';
   BAYS(bay).mixing = 'N';
   BAYS(bay).R.slots = 0;
elseif BAYS(bay).empty_slots > BL.capacity
   keyboard
end
```

```
function BAYS_check_repeated(ids_vect)
% This function checks wether bays are used twice for a VS Plan
ids_vect=sort(ids_vect);
for i=2:length(ids_vect)
   if ids_vect(i)==ids_vect(i-1)
      figure; plot(ids_vect(3,:),'.')
      disp('BAYS_check_repeated Error: a bay has been used two times')
      keyboard
      %close
   end
end
```

```
function [iscomplete,C_slots,newCB_NSD] = BAYS_find_slots(bay_id,mix_id,VS,operation,slots_needed)
% This function analyzes the bays and returns a vector with the identity of
% those bays that have a number of emtpy slots less or equal to the given
% value

global BAYS BL BT T TRF YC

% initial values
CB_NSD = [0; 0; 1000000];
WL = 0;
i_bay = 0;

for bay = 1:T.bays
   % Initial values
   cts_found = 'Y';
   % Bay Occupation: if bay has more CTs than allowed, bay not valid
   bay_occup = BL.capacity - BAYS(bay).empty_slots;
   if strcmp(BAYS(bay).id,'NAS') == 0
      if strcmp(BAYS(bay).id,'IMP') == 1
         ocup_limit = T.limits.bay.imp;
      elseif strcmp(BAYS(bay).id,'EXP') == 1
         ocup_limit = T.limits.bay.exp;
      end
      if bay_occup >= ocup_limit
         cts_found = 'N'; continue
      end
   end
   % Check bay reservation
   if BAYS(bay).R.slots >= BL.capacity - BL.tiers
      continue
   end
   % Bay ID Comparison
   if strcmp(BAYS(bay).id,bay_id) == 0
      cts_found = 'N'; continue
   end

   % Port Comparison:
   % If we look for spaces, we must pay attention to the allocation strategy
   slots = 0;
   if strcmp(BAYS(bay).id, 'NAS') == 1
      slots = BL.capacity;
   else

      switch operation

         case 'VSplan' % Strategy: Yes, BAY.Matriz: No, BAY.Port: Yes
            % 1) Are there containers of the same port within the bay?
            if strcmp(BAYS(bay).mixing,mix_id) == 0
               %keyboard
               continue
            end
            [nports] = ports_howmanyofeach(BAYS(bay).R.ports);
```

```
        if nports(VS.no) > 0
            if BAYS(bay).R.slots < BL.tiers
                continue
            else
                if sum(nports) > nports(VS.no)
                    disp('This is a nas bay)'); keyboard
                else
                    reservables = BL.capacity -BL.tiers - BAYS(bay).R.slots;
                    slots = min(reservables, BL.capacity-BL.tiers);
                end
            end
        else
            continue
        end

    case 'ETdrop' % Strategy: yes, BAY.matriz: 0, BAY.Port: yes
        [cts_found] = BAY_check_id(BAYS(bay),mix_id,VS);
        if cts_found == 'Y'
            [slots] = BAY_search_ports_strategy(BAYS(bay),mix_id,VS.port);
        end
    case 'ETdroprand' % Strategy: yes, BAY.matriz: 0, BAY.Port: yes
        %keyboard
        if strcmp(BAYS(bay).mixing,mix_id) == 0
            %keyboard
            continue
        end

        % Two requisites: there are vts of the same vessel
        % a)Number of slots for each port
        if BL.capacity - BL.tiers <= BAYS(bay).R.slots
            continue
        end
        %keyboard
        ports = BAY_find_reservations(bay,'B');
        reserved_slots = ports(VS.no);
        if sum(ports)-reserved_slots == 0
            cts_found = 'Y';
            slots = BL.capacity - reserved_slots;
        end

    case 'ETdroprandstacks' % Strategy: yes, BAY.matriz: 0
        if strcmp(BAYS(bay).mixing,mix_id) == 0
            continue
        end
        if BL.capacity - BL.tiers <= BAYS(bay).R.slots
            continue
        end
        %keyboard
        ports = BAY_find_reservations(bay,'B');
        if sum(ports) < BL.capacity - BL.tiers
            cts_found = 'Y';
            slots = BL.capacity -BL.tiers- sum(ports);
        end
%           ocup = sum(BAYS(bay).matriz);
```

```
%               positions = find(ocup==0);
%               if isempty(positions)==1
%                   slots = 0;
%               else
%                   slots = slots + length(positions)*BL.tiers;
%                   available_slots = BAYS(bay).empty_slots-BL.tiers;
%                   if available_slots > 0
%                       slots = min(slots,available_slots);
%                   else
%                       slots = 0;
%                   end
%               end
        case 'VSupload' % Strategy: no, BAY.matriz: 1, BAY.port: yes
            for tier =1:BL.tiers
                for stack = 1:BL.stacks
                    if and(BAYS(bay).port(tier,stack) == VS.port, BAYS(bay).matriz(tier,stack) == 1)
                        slots = slots + 1;
                    end
                end
            end
        case 'VSdownload' % Strategy: No, BAY.matriz: 1, BAY.port: No
            %[positions,slots] = Matriz_empty_slots(BAYS(bay),VS.no);
            a = find(BAYS(bay).R.ports == VS.no);
            if isempty(a) == 0
                slots = BAYS(bay).R.slots-sum(sum(BAYS(bay).matriz)); %before: BAYS(bay).empty_slots;
            end
        case 'ETpick' %  Strategy: No, BAY.matriz: 1, BAY.port: no
            for tier =1:BL.tiers
                for stack = 1:BL.stacks
                    if BAYS(bay).matriz(tier,stack) == 1
                        slots = slots + 1;
                    end
                end
            end
    end % switch operation
  end
  if and(cts_found == 'Y', slots > 0)
    % We have candidate slots but we need to check the YC Wl
    [yc] = YC_assign_ct(bay);

    % If this condition is passed, then evaluate the candidate bay
    i_bay = i_bay + 1;
    CB_NSD(1,i_bay) = bay;
    CB_NSD(2,i_bay) = min(slots,BL.capacity-BL.tiers);
    if strcmp(operation,'VSdownload') == 1
        CB_NSD(3,i_bay) = distance_calculator(BAYS(bay).position, T.gate.position);
    else %if or(strcmp(operation,'ETdrop') == 1, strcmp(operation,'ETdroprand') == 1) % ALSO ETDroprandstack
        CB_NSD(3,i_bay) = distance_calculator(BAYS(bay).position, BT(VS.berth).position);
    end
    %CRANES(i_bay) = yc;
    WL(i_bay) = YC(yc).WL.n;
  end
end
%keyboard
```

```
baycoefs = 0.5*CB_NSD(3,:)/max(CB_NSD(3,:)) + 0.5*WL/TRF.PARAM.overload;


[a_sorted, a_order] = sort(baycoefs);
newCB_NSD = CB_NSD(:,a_order);

available_slots = sum(CB_NSD(2,:));
C_slots = min(slots_needed,available_slots);

if available_slots >= slots_needed
   iscomplete = true;
else
   iscomplete = false;
end

% Final step, evaluate the bays according to the distance and yc wl
```

```
function [type]=BAYS_impexp(initial_emtpy_bays)
% THis function generates a type of operation to be associated to a CT, an
% ET, etc.

port=0;

p1 = initial_emtpy_bays;
p2 = initial_emtpy_bays+(1-initial_emtpy_bays)/2;
%p3=1-p1-p2;

aux=random('unif',0,1);
if aux<p1
   type = 'NAS';
elseif aux<p2
   type = 'EXP';
else
   type = 'IMP';
end
```

```
function [ct] = BAYS_init()
% This fucntion generates the terminal initial situation

global BAYS BL CT S T TRF
% 0. Parameters

num_cts = T.slots*T.initial_occupation;
num_bays = T.bays*(1-T.initial_emtpy_bays);
target_occupation = fix(num_cts/num_bays);

% 1. Create bays features
b = 0; % for bay
bl = 0; bb = zeros(T.blocks,BL.bays); % for block
for t_col = 1:T.cols
   for t_row = 1:T.rows
      bl = bl + 1;
      for b_bay = 1:BL.bays
         b = b + 1;
         % UNVARIABLE bay features
         BAYS(b).no = b;
         BAYS(b).block = bl;
         BAYS(b).row = t_row;
         BAYS(b).col = t_col;
         % BAY Position
         BAYS(b).b_bay = b_bay;
         BAYS(b).T_col = t_col;
         BAYS(b).T_row = t_row;
         BAYS(b).position(1) = T.aisles.sides.width+(BL.length+T.aisles.vertical.width)*(t_col-1)+(b_bay-1)*S.l;
         BAYS(b).position(2) = T.aisles.bottom.width-T.aisles.horizontal.width+
(BL.width+T.aisles.horizontal.width)*t_row;

         % Locatio at block
         bb(bl) = bb(bl) + 1;
         BL.baylist(bl,bb(bl)) = b;
         BL.rowlist(t_row,t_col) = bl;

         % VARIABLE bay features
         BAYS(b).id = 'NAS';
         BAYS(b).mixing = 'N'; % Can be N (None), U (Uniform), S (Stack), Y (Yes)
         BAYS(b).empty_slots = BL.capacity;
         % BAY Matrices
         BAYS(b).matriz = zeros(BL.tiers,BL.stacks);
         BAYS(b).ct_id = zeros(BL.tiers,BL.stacks);
         BAYS(b).ct_arrival = zeros(BL.tiers,BL.stacks);
         %BAYS(b).port = zeros(BL.tiers,BL.stacks);
         %BAYS(b).weightc = zeros(BL.tiers,BL.stacks);
         BAYS(b).R.slots = 0;
         BAYS(b).R.ports = 0;%cts = zeros(BL.tiers,BL.stacks);
%           BAYS(b).R.IMP.slots = 0;
%           BAYS(b).R.IMP.ports = 0;
      end
   end
end
```

```
% Assign an IMP or EXP ID to the bays
% for bay=1:T.bays
%     % BAYS(b).port(:,:) = 1 + fix(random('unif',0,1)*TRF_PARAM.no_ports); % No vessel associated yet
%     [BAYS(bay).id] = BAYS_impexp(T.initial_emtpy_bays);
% end

% 2. Create the slot_exist vector and fill it with containers

ct=0;
max_no_ct=fix(T.slots*T.initial_occupation);


% 2.1 Create a list of IMP and EXP bays
no_bays=0;
for bay=1:T.bays
  if BAYS(bay).id ~= 'NAS'
     no_bays = no_bays+1;
     bays_vector(no_bays) = bay;
  end
end

while ct <max_no_ct

   % a) Generate a random bay
   bay = bays_vector(fix(random('unif',1,no_bays+1)));
   %bay = fix(random('unif',1,T.bays+1));
   % b) Check the bay occupation
   bay_occupation = BL.capacity - BAYS(bay).empty_slots;
   if bay_occupation < ceil(target_occupation*1.2)
      slot_found='N';
      while slot_found=='N'
         stack = fix(random('unif',1,BL.stacks+1));
         cts_in_row = sum(BAYS(bay).matriz(:,stack));
         if cts_in_row < BL.tiers
            slot_found='Y';
            % Then find the position of the empty slot
            tier_found='N';
            %tier = BL.tiers+1;
            tier = 0;
            while tier_found=='N';
               % tier=tier-1;
               tier = tier +1;
               if BAYS(bay).matriz(tier,stack)==0
                  ct = ct + 1;
                  CT(ct).weight = CT_weight(random('unif',TRF.CT.cdf(1),1));
                  CT(ct).class = CT_class(CT(ct).weight);
                  BAYS(bay).matriz(tier,stack) = 1;
                  BAYS(bay).ct_id(tier,stack) = ct;
                  BAYS(bay).weightc(tier,stack) = CT(ct).class;
                  % generate the CT arrival time in days
                  if BAYS(bay).id == 'EXP'
                     ct_arrival = 1 + fix(random('unif',0,100)*TRF.PARAM.daysinadvance)/100;
                  elseif BAYS(bay).id == 'IMP'
```

```
                ct_arrival = 1 + fix(random('unif',0,100)*TRF.PARAM.daysofdischarge)/100;
            end
            BAYS(bay).ct_arrival(tier,stack)= ct_arrival/100*24*60*60; % In seconds
            BAYS(bay).empty_slots=BAYS(bay).empty_slots-1;
            tier_found='Y';
        end
      end
    end
  end
end

% Assign a Port to the bays
for bay=1:T.bays
  if BAYS(bay).id ~= 'NAS'
    % Generate a port
    port = 1+fix(random('unif',0,1)*TRF.PARAM.no_ports);
    for tier = 1: BL.tiers
      for stack = 1:BL.stacks
        if BAYS(bay).matriz(tier,stack) ~= 0
          [BAYS(bay).port(tier,stack)] = port;
        end
      end
    end
  end
end

% 3. Check again for empty bays
% for bay=1:T.bays
%   if BAYS(bay).empty_slots == BL.capacity
%     BAYS(bay).id = 'NAS';
%   end
% end

ploting = 0;
if ploting >0
  figure(40)

  disp('Initial bay configuration plot')

  plot_bays
  pause(1)
end
```

```
function [no_reshuffles,h_reshuffles,no_ct_col] = BAYS_reshuffles(ct)
% This function calculates the number of reshuffles and the height that the
% containers must be elevated. The dimension is therefore [CT*heights]

global BAYS BL CT

%keyboard
% initialize outputs
no_reshuffles = 0; h_reshuffles = 0;

bay = CT(ct).P.bay;
tier = CT(ct).P.tier;
stack = CT(ct).P.stack;

% Number of containers in that pile
no_ct_col = sum(BAYS(bay).matriz(:,stack));

% The number of CT to be reshuffled will be:
no_reshuffles = no_ct_col - tier;

% The height to which CT will be elevated:
h = BL.tiers + 1;

% The magnitude of resuffles is
for r = 1:no_reshuffles
    ct_height = no_ct_col - r + 1;
    ct_elevation = h - ct_height;
    h_reshuffles = h_reshuffles + ct_elevation;
end
```

```
function [esl,b_esl] = block_analyze()
% This function analyzes the terminal block to see the empty slots and

global BAYS BL T

esl=zeros(1,T.bays);
b_esl = zeros(1,T.blocks);
b_res = zeros(1,T.blocks);
b_ocup = zeros(1,T.blocks);

for bay = 1:T.bays
   if strcmp(BAYS(bay).id,'EXP') ==1
      block = BAYS(bay).block;
      for tier = 1: BL.tiers
         for stack = 1:BL.stacks
            % Empty slots and slots per block
            if BAYS(bay).matriz(tier,stack) ==0
               if BAYS(bay).port(tier,stack) ==0
                  esl(bay) = esl(bay) + 1;
                  b_esl(block) = b_esl(block) + 1;
               elseif BAYS(bay).port(tier,stack)> 0
                  b_res(block) = b_res(block) + 1;
               end
            else
               b_ocup(block) = b_ocup(block) + 1;
            end
         end
      end
   end
end

figure;
plot(b_esl,'b.'); hold on
%plot(b_res,'g.');
%plot(b_ocup,'m.');
y = b_esl + b_res;
plot(y,'r.');
```

```
function BT_initialization()

global BT T

for bt = 1:T.berth_no
    BT(bt).position(1) = bt * T.length/(T.berth_no+1);
    BT(bt).position(2) = T.width;
    BT(bt).status = 'empty';
    BT(bt).QC_no = 4;
    BT(bt).active = 0;
    for j = 1:BT(bt).QC_no
        BT(bt).QC_id(j) = (bt - 1) * BT(bt).QC_no + j;
    end
    BT(bt).vessel = 0;
    BT(bt).active = 0;
end
```

```
function [target_ct]= cdf_stay(vs)
% This function calculates the cumulative density function of the CT stay
% time

global BAYS BL CT TIME VS
%keyboard
CB = VS(vs).plan.IMP.bays;
nb = length(CB);

ct_no=0;
ct_list = zeros(1,2);
for i_bay = 1:nb
   bay = CB(i_bay);
   for stack = 1:BL.stacks
      for tier = 1:BL.tiers
         ct = BAYS(bay).ct_id(tier,stack);
         if ct >0
            if CT(ct).vs == vs
               ct_no = ct_no + 1;
               ct_list(ct_no,1) = BAYS(bay).ct_id(tier,stack);
               ct_list(ct_no,2) = TIME.t + TIME.delt - BAYS(bay).ct_arrival(tier,stack);
            end
         end
      end
   end
end

% 1.3 Sort the list according to the time (coordinate 2
%ct_list = ct_list';
ct_list = sortrows(ct_list,2);
%ct_list = ct_list';

% 1.4 Make a vector of accumulated times:
if ct_no >1
   ac_time(1) = ct_list(1,2);
   ct_list(:,2) = ct_list(:,2) - ac_time(1);
   ac_time(1) = ct_list(1,2);
   for i=2:ct_no
      ac_time(i) = ac_time(i-1) + ct_list(i,2);
   end
   sum_times = sum(ct_list(:,2));

   % 1.5 Generate a random number and find the position of the target CT
   target_time = 1 + fix(random('unif',0,1)*sum_times); target_time = min(target_time,sum_times);
   ict = 1;
   while target_time > ac_time(ict)
      ict = ict + 1;
   end

   target_ct = ct_list(ict,1);
else
   target_ct =  ct_list(1,1);
end
```

```
if target_ct == 0
   keyboard
else
   if CT(target_ct).R.readytogo >0
      %keyboard
   else
      CT(target_ct).R.readytogo = CT(target_ct).R.readytogo+1;
   end
end

if CT(target_ct).vs ~= vs
   disp('error in CT pick')
   it = 0;
   while CT(ct).vs ~= vs
      it = it+1;
      pos = find(CB == CT(ct).P.bay);
      CB(pos) = [];
      VS(vs).plan.IMP(pos) = [];
      [ct] = cdf_stay(CB);
      if it >12
         keyboard
      end
   end
else
   CT(target_ct).vspick = vs;
end
```

```
function [xf,top_xf,cdf]=cdf_x2(no_wc,wcl,min_weight,max_weight)

no_c=length(wcl);
swcl=sort(wcl); % sorted weight container levels

% Calculate the new cdf
ndelx=(max_weight-min_weight)/no_wc; % Intervals at x axis (weight)
top_xf=(ndelx:ndelx:ndelx*no_wc)+min_weight;
xf=top_xf-ndelx/2;

cont=1;
i=1; % index for the whole lenght of weight list
while i<=no_c && cont<no_wc
   if swcl(i)<=top_xf(cont)
      i=i+1;
   else
      cdf(cont)=i-1;
      cont=cont+1;
      i=i+1;
   end
end
cdf(no_wc)=no_c;
cdf=cdf/no_c;
```

```
function check_bay_es(bay)

global BAYS BL

a = find(BAYS(bay).ct_id >0);
if abs(BL.capacity -BAYS(bay).empty_slots- length(a)) >0
    disp([num2str(BAYS(bay).empty_slots) ' Empty slots and ' num2str(length(a)) ' CTs'])
    BAYS(bay).ct_id

    keyboard
    BAYS(bay).empty_slots = BL.capacity - length(a);
end
```

```
function check_bay_types()
% This function checks the yard inventory, bays and cts

global BAYS BL T

no_nas=0; no_imp=0; no_exp=0;
ct_imp=0; ct_exp=0;

for bay=1:T.bays
   % Not assigned bays
   if BAYS(bay).id == 'NAS'
      no_nas = no_nas + 1;
   elseif BAYS(bay).id == 'EXP'
      no_exp = no_exp +1;
      ct_exp = ct_exp + BL.capacity - BAYS(bay).empty_slots;
   elseif BAYS(bay).id == 'IMP'
      no_imp = no_imp + 1;
      ct_imp = ct_imp + BL.capacity - BAYS(bay).empty_slots;
   end
end

occup_imp = ct_imp/no_imp;
occup_exp = ct_exp/no_exp;

disp(['NAS bays: ' num2str(no_nas) ])
disp(['IMP bays: ' num2str(no_imp) ])
disp(['EXP bays: ' num2str(no_exp) ])
disp(['Imp CTs: ' num2str(ct_imp) ])
disp(['Exp CTs: ' num2str(ct_exp) ])
disp(['Cts per Imp bay: ' num2str(occup_imp) ])
disp(['Cts per Exp bay: ' num2str(occup_exp) ])
```

```
function [diff,row,tier]= check_class_diff(comp_bay,idealbay,c_class)

% This function takes a vector of weightclass and places it on the best
% possible row and column

% Initialize

[no_tiers,no_rows]=size(idealbay);
%comp_bay=flipud(comp_bay);
diference=zeros(1,no_rows);
toptier=zeros(1,no_rows);
for row=no_rows:-1:1 % buscar la col en la que mejor encaja
   %row;
   tier=no_tiers;

   while comp_bay(tier,row)>0 && tier ~= 1
      if tier>1
         tier=tier-1;
      end
   end
   if tier==1 && comp_bay(tier,row) >0
      diference(row)=100;
   else
      diference(row)=idealbay(tier,row)-c_class;
   end
   toptier(row)=tier;
end
%now choose the position with the smallest difference
% this function will be further improved
[diff,row]=min(abs(diference));
tier=toptier(row);
```

```
function check_cts_vs()

global BAYS BL BT CT COUNT MAC T VS YC


for bt=1:3
   vs = BT(bt).vessel;
   if vs>0
      ci = 0;ce=0;
      ctlisttheory=0; yclisttheory=0; cts_theory=0;
      pbays=zeros(1,T.bays);nyc_bays=zeros(1,MAC.YC.n);yc_bays=zeros(T.bays,MAC.YC.n);
      % Analyze the list of cts and see imp and exp cts
      for ct=1:COUNT.ct
         if CT(ct).vs==vs
            yc = CT(ct).P.yc; b = CT(ct).P.bay;
            if strcmp(CT(ct).type,'IMP')==1
               ci=ci+1;
            else
               ce = ce+1;
               pbays(b) = pbays(b)+1;
               ctlisttheory(ce) = ct;
               yclisttheory(ce) = yc;
               nyc_bays(yc) = nyc_bays(yc)+1;
               yc_bays(nyc_bays(yc),yc) = b;
            end
         end
      end

      bay_ids= find(pbays>0);
      nb=length(bay_ids);
      ctsfound = 0;
      pct= 0;
      for b = 1:nb
         bay = bay_ids(b);
         bc=0;
         for s=1:BL.stacks
            for t=1:BL.tiers
               ct = BAYS(bay).ct_id(t,s);
               if ct>0
                  if CT(ct).vs ==vs
                     bc = bc +1;
                     ctime(bc)=CT(ct).events.time(1);
                     pct = pct+1;
                     cts_theory(pct)=ct;
                  end
               end
            end
         end
         ctsfound = ctsfound +bc;
      end

      % Mirar elementos repetidos
      absentct = 0; ec = 0; ec2 = 0; nec = 0; lc = 0; lostcts = 0; yclostcts = 0;
```

```
    for i=1:ce
       cct = ctlisttheory(i);
       if isempty(find(cts_theory == cct)) == 1
          nec = nec+1;
          absentct(nec) = cct;
          yc = CT(cct).P.yc;
          if isempty(find(YC(yc).WL.prior.cts == cct))== 0
             lc = lc +1;
             lostcts(lc)=cct;
             yclostcts(lc)=yc;
          elseif isempty(find(YC(yc).WL.normal.cts == cct)) ==0
             lc = lc +1;
             lostcts(lc)=cct;
             yclostcts(lc)=yc;
          end
       else
          ec = ec + 1;
          presentct(ec) = cct;
          if strcmp(CT(cct).events.event(end),'stacked')==1
             ec2 = ec2+1;
             a(ec2) = CT(cct).events.time(1)/3600/24;
             f(ec2) = CT(cct).events.time(end)/3600/24;
          end
       end
    end
    keyboard
    figure(vs+1000); plot(presentct,a,'.b'); hold on
    figure(vs+1000); plot(presentct,f,'.m');
    figure(vs+1000); plot(VS(vs).arrival.time/3600/24,'*r');
    title(['VS ' num2str(vs) 'EXP cts'])
    %keyboard
    yclist=unique(yclisttheory); c_yc = 0;
    % List of present containers
%       for i = 1:length(yclist)
%          yc = yclist(i);
%          c_yc(i)=length(find(yclisttheory == yc));
%          if and(YC(yc).WL.prior.n + YC(yc).WL.normal.n > 0, YC(yc).active==0)
%             disp(['Error the YC(' num2str(yc) ') should be active' ])
%          end
%       end
    disp([num2str(ci) '/' num2str(ce) ' IMP/EXP han llegado a la terminal para el VS(' num2str(vs) ') out of '
num2str(VS(vs).IC) '/' num2str(VS(vs).OC)])
    disp(['Distributed in ' num2str(nb) ' bays'])
    disp([num2str(ctsfound) ' cts are present and ' num2str(lc) ' cts are not assigned'])
    disp(['The YCs with assigned cts are ' num2str(yclist)])
    disp(['The WL of those cranes is ' num2str(c_yc)])
    % List of lost containers
    yclostlist=unique(yclostcts);
    %keyboard
    for i=1:length(yclostlist)
       lc_yc(i)=length(find(yclostcts == yclostlist(i)));
    end
    disp(['The YCs with cts pending is ' num2str(yclostlist)])
    disp(['The WL of those cranes is ' num2str(lc_yc)])
```

```
        %figure; plot(ctime/3600/24)
        %keyboard
        t_yc = 7;
        bdyc=unique(yc_bays(:,t_yc));bdyc(1)=[];
        plot_bays_candidates(bdyc,'g*');

        %keyboard
    end
end
```

```
function [no_slots,distance] = check_port_bays(bay,port)
% This function checks if there are slots in a bay that belong to a port

global BAYS BL BT CT VS

no_slots = 0; distance = 0; calc = 1;
for stack = 1: BL.stacks
   for tier = 1:BL.tiers
      ct = BAYS(bay).ct_id(tier,stack);
      if ct >0
         if CT(ct).vs == port
            no_slots = no_slots + 1;
            vs = CT(ct).vs;
            if vs >0
               berth = VS(vs).berth;
               if calc == 1
                  distance = distance_calculator(BAYS(bay).position,BT(berth).position);
                  calc = 0;
               end
            end
         end
      end
   end
end

if length(BAYS(bay).R.slots) > 1
   disp('too long slots')
   keyboard
end
```

```
function check_port_occupation(bay)

global BL BAYS

if BAYS(bay).R.slots > BL.capacity - BL.tiers
    disp('Excessive port occupation')
    keyboard
end
```

```
function check_port_reservation(bay)
% This function checks if a cero is present in a position where it should
% not exist.
global BAYS BL

for stack = 1:BL.stacks
   for tier = 1:BL.tiers-1
      if and(BAYS(bay).port(tier,stack) == 0,BAYS(bay).port(tier+1,stack) > 0)
         disp('Error @ port reservation matrix')
         keyboard
      end
%        if strcmp(BAYS(bay).id,'EXP') == 1
%           if and(BAYS(bay).port(tier,stack) > 0,BAYS(bay).matriz(tier,stack) == 0)
%              disp('Error 2')
%              keyboard
%           end
%        end
   end
end
```

```
function [pure_ports,stack_ports,mixed_ports] = check_reserve()
% This function calculates the number of slots resereved for the different
% types of mixing strategies and ports

global TRF T BAYS

pure_ports = zeros(1,TRF.PARAM.no_ports); pp = 0; pbays = 0;
stack_ports = zeros(1,TRF.PARAM.no_ports); sp = 0; sbays = 0;
mixed_ports = zeros(1,TRF.PARAM.no_ports); mp = 0; mbays = 0;

for bay = 1:T.bays
   if strcmp(BAYS(bay).id, 'EXP') == 1
      %[ports] = BAY_find_Port_reserved(BAYS(bay));

      ports = ports_howmanyofeach(BAYS(bay).R.ports);
      if strcmp(BAYS(bay).mixing, 'N') == 1
         pure_ports = pure_ports + ports;
         pp = pp +1; pbays(pp) = bay;
      elseif strcmp(BAYS(bay).mixing, 'S') == 1
         stack_ports = stack_ports + ports;
         sp = sp +1; sbays(sp) = bay;
      elseif    strcmp(BAYS(bay).mixing, 'Y') == 1
         mixed_ports = mixed_ports + ports;
         mp = mp +1; mbays(mp) = bay;
      end
   end
end

%keyboard
```

```
function check_terminal_ocupation(label,VS)

global BAYS BL T

plot_bays

% Choose the plot
if strcmp(label,'EXP') == 1
   sp = 1;
   no_ct_vs = VS.OC;
   texto='EXP CT from VS going to ports';
elseif strcmp(label,'IMP') == 1
   sp = 2 ;
   no_ct_vs = VS.IC;
   texto='IMP CT Download: Cts will be distributed among all IMP bays';
end
figure(30); subplot(2,1,sp); hold on
plot(VS.port,no_ct_vs,'r.'); title(texto)
```

```
function [no_ct]=checkbays(T,bays,id)
no_ct=0;
no_bays=0;
for i=1:T.bays
    if bays(i).vs_plan==id
        no_bays=no_bays+1;
        no_ct=no_ct+sum(sum(bays(i).matriz));
    end
end
no_bays;
```

```
function close_figure_ifexists(fig_number)

[fig_exist]=figure_ckeck(fig_number);
if fig_exist==1 % there is a figure, close it
   close(fig_number)
end
```

```
function closure()

global BL COST COUNT GROUPS MAC R T TIME TRF YC

clc
close all
disp('------------------------------------------------------------')
terminal_state();
% Calculate fix costs
% ----------------------------------------------------
% 1. of YC
COST.YC.total.fix = COST.YC.fix * MAC.YC.ycsprow * TIME.t/24/3600;
% Bays cost
COST.BAY = T.bays * COST.bay * TIME.t/24/3600;
disp(['Total cost bay: ' num2str(COST.BAY)])
R.occupation = (T.state.EXP.no.cts + T.state.IMP.no.cts)/T.bays/BL.capacity;
disp(['Final occupation (%): ' num2str(100*R.occupation)])

TIME.end = clock;

TIME.duration = TIME.start - TIME.end;

ct = COUNT.ct;
plot_evolution()
disp(['Total IMP/EXP # CTs ' num2str(COUNT.IMP.ct) '/' num2str(COUNT.EXP.ct)])
disp(['Mean IMP ET Cost ' num2str(mean(COST.ET.IMP.total))])
disp(['Mean EXP ET Cost ' num2str(mean(COST.ET.EXP.total))])
disp('------------------------------------------------')
disp(['Mean IMP YT Cost ' num2str(mean(COST.YT.IMP.total))])
disp(['Mean EXP YT Cost ' num2str(mean(COST.YT.EXP.total))])
% disp(['Total IMP ET Cost ' num2str(COST.ET.IMP.total)])
% disp(['Total EXP ET Cost ' num2str(COST.ET.EXP.total)])
% disp('------------------------------------------------')
% disp(['Total IMP YT Cost ' num2str(COST.YT.IMP.total)])
% disp(['Total EXP YT Cost ' num2str(COST.YT.EXP.total)])
disp('------------------------------------------------')
disp(['IMP Reshuffles ' num2str(sum(COUNT.R.IMP.nr))])
disp(['EXP Reshuffles ' num2str(sum(COUNT.R.EXP.nr))])
disp('------------------------------------------------')
disp(['IMP R height ' num2str(sum(COUNT.R.IMP.sheight))])
disp(['EXP R height ' num2str(sum(COUNT.R.EXP.sheight))])


for yc = 1:MAC.YC.n
   n = length(find(YC(yc).WL.time<7*3600*24));
   startover = 1; oe = 0; O =0;
   wlt = YC(yc).WL.time; wlt = sort(wlt);
   while n < YC(yc).WL.n-1
      n = n+1;
      delt(n) = wlt(n) - wlt(n-1);

      if delt(n) < 60/TRF.PARAM.overload
        if startover == 1
```

```
            oe = oe+1; O(oe) = 0;
          end
          O(oe) = O(oe) + delt(n); startover = 1;
        end
    end
    yco(yc) = sum(O);
end

disp(['Cranes overload (%time): ' num2str((yco)/7/24*100)])
disp(['Mean crane overload (%time): ' num2str(mean(yco)/7/24*100)])
disp('----------------------------------------------------------------')
disp('IMP OPERATIONS')
disp('----------------------------------------------------------------')
disp('Stack')
disp('----------------------------------------------------------------')
disp(['IMP YC Gantry ' num2str(COST.YC.IMP.stack.gantry/COUNT.IMP.ct)])
disp(['IMP YC Trolley ' num2str(COST.YC.IMP.stack.trolley/COUNT.IMP.ct)])
disp(['IMP YC Hoist ' num2str(COST.YC.IMP.stack.hoist/COUNT.IMP.ct)])
disp('Delivery')
disp('----------------------------------------------------------------')
disp(['IMP YC Gantry ' num2str(COST.YC.IMP.deliver.gantry/COUNT.IMP.ct)])
disp(['IMP YC Trolley ' num2str(COST.YC.IMP.deliver.trolley/COUNT.IMP.ct)])
disp(['IMP YC Hoist ' num2str(COST.YC.IMP.deliver.hoist/COUNT.IMP.ct)])
disp('----------------------------------------------------------------')
disp('EXP OPERATIONS')
disp('----------------------------------------------------------------')
disp('Stack')
disp('----------------------------------------------------------------')
disp(['EXP YC Gantry ' num2str(COST.YC.EXP.stack.gantry/COUNT.EXP.ct)])
disp(['EXP YC Trolley ' num2str(COST.YC.EXP.stack.trolley/COUNT.EXP.ct)])
disp(['EXP YC Hoist ' num2str(COST.YC.EXP.stack.hoist/COUNT.EXP.ct)])
disp('Delivery')
disp('----------------------------------------------------------------')
disp(['EXP YC Gantry ' num2str(COST.YC.EXP.deliver.gantry/COUNT.EXP.ct)])
disp(['EXP YC Trolley ' num2str(COST.YC.EXP.deliver.trolley/COUNT.EXP.ct)])
disp(['EXP YC Hoist ' num2str(COST.YC.EXP.deliver.hoist/COUNT.EXP.ct)])


for i =1: length(GROUPS)
    nog = 0; nob= 0; nog = 0;
    for j = 1:TRF.PARAM.VS_no
        %nog(i,j) = GROUPS(i).CT(j).no;
        nog(j) = GROUPS(i).CT(j).no;
        nob(j) = mean(GROUPS(i).CT(j).bahias(:));
        noc(j) = mean(GROUPS(i).CT(j).cts(:));
    end
    mediag(i)= mean(nog(nog>0));
    mediab(i)= mean(nob(nob>0));
    mediac(i)= mean(noc(noc>0));
    not(i)= GROUPS(i).time;
end

figure; subplot(3,1,1);plot(not/3600/24,mediag); title('# Groups')
subplot(3,1,2);plot(not/3600/24,mediab); title('# Bays per group')
```

```
subplot(3,1,3);plot(not/3600/24,mediac); title('# Containers per group')
disp(['Number of groups ' num2str(mediag(end))])
disp(['Number of bays ' num2str(mediab(end))])
disp(['Number of cts ' num2str(mediac(end))])
```

```
function [vect_cont]=container_vector(ndf,no_cont)

% first create the vector of classes
vect=round(ndf*no_cont);
diference=vect-round(vect);
vect_class=round(vect);
if min(vect_class)==0
   'Number of classes is not appropriate for the exising distribution of containers';
   % vect_class
   keyboard
end

while sum(vect_class)~=no_cont
   if sum(vect_class)<no_cont
      [j,max_row]=max(abs(diference));
      vect_class(max_row)=vect_class(max_row)+1;
   elseif sum(vect_class)>no_cont
      [j,max_row]=max(abs(diference));
      vect_class(max_row)=vect_class(max_row)-1;
   end
   diference(max_row)=0;
end

% now the vector of containers
cont=0;
for class=1:no_wc
   for j=1:vect_class(class)
      cont=cont+1;
      vect_cont(cont)=class;
   end
end
```

```
function [] = COST_init()

global COST

COST.bay = 0.01;          % (€/day) Space cost of bay per day
COST.YC.travel = 0.02;    % (€/s) YC travel cost per unit time
COST.YC.fix = 1000;       % (€/day) YC fixed cost per day

total.fix = 0;
total.gantry = 0;
total.hoist = 0;
total.trolley = 0;

COST.YC.IMP.stack = total;
COST.YC.IMP.deliver = total;
COST.YC.EXP.stack = total;
COST.YC.EXP.deliver = total;

COST.YT.travel = 0.004; % Amount for 40l/100km = 0.0004€/m % Before: 0.02;
COST.ET.travel = 0.015;

% Initialize Total Costs:

COST.YT.IMP.total = [];
COST.YT.EXP.total = [];
COST.reshuffle.EXP = 0;
COST.reshuffle.IMP = 0;
COST.r_heights.EXP = 0;
COST.r_heights.IMP = 0;
COST.mixing = 0;
COST.remarshall = 0;
COST.ET.EXP.total = [];
COST.ET.IMP.total = [];
%COST.ET.travel = 0;
```

```
function COUNT_init()

global COUNT MAC T TRF

COUNT.ct = 0;
COUNT.IMP.ct = 0;
COUNT.EXP.ct = 0;
COUNT.vs = 0;
COUNT.vs_p = 0;
COUNT.et = 0;
COUNT.stat = 0;
COUNT.inventory.exp = zeros(1,TRF.PARAM.VS_no+1);
COUNT.inventory.imp = zeros(1,TRF.PARAM.VS_no+1);
COUNT.deliveries.exp = zeros(1,TRF.PARAM.VS_no+1);
COUNT.deliveries.imp = zeros(1,TRF.PARAM.VS_no+1);

COUNT.stacks.exp = zeros(1,TRF.PARAM.VS_no+1);
COUNT.stacks.imp = zeros(1,TRF.PARAM.VS_no+1);

COUNT.vsarrivals.imp = zeros(1,TRF.PARAM.VS_no+1);
COUNT.vsarrivals.exp = zeros(1,TRF.PARAM.VS_no+1);

X.events = 0;
X.nr = 0;
X.revent = 0;
X.sheight = 0;

COUNT.R.EXP = X;
COUNT.R.IMP = X;

for i = 1: T.rows*MAC.YC.ycsprow
   COUNT.YC.overload(i).no = 0;
   COUNT.YC.overload(i).time = 0;
end
```

```
function CT_addevent(ct,evento,delay)

global CT TIME

no = CT(ct).events.no +1;
CT(ct).events.no = no;
CT(ct).events.time(no) = TIME.t+delay;
CT(ct).events.event{no} = evento;
```

```
function [class] = CT_class(weight)

global TRF
xf = TRF.CT.top_xf;
class = 1;
for i = 1:length(xf)-1
    if weight > xf(i)
        class = i + 1;
    end
end
```

```
function [vect_cont]=CT_distribution(pdf,no_cont)

% This function makes a vector of categories of containers,
% given a probability density distribution. This vector will
% be used later on to fill the ideal bay.

% first create the vector of classes
vect=round(pdf*no_cont);
diference=sum(vect)-no_cont;  %vect-round(vect);

while abs(diference)~= 0
   vdiference=pdf*no_cont-vect;
   [i,j]=max(abs(vdiference));
   if diference >0
      vect(j)=vect(j)-1;
   else
      vect(j)=vect(j)+1;
   end
   diference=sum(vect)-no_cont;
end
% now the vector of containers
cont=0;
for class=1:length(vect)
   for j=1:vect(class)
      cont=cont+1;
      vect_cont(cont)=class;
   end
end
```

```
function CT_erase(ct)

global BAYS BL CT

bay = CT(ct).P.bay;
tier = CT(ct).P.tier;
stack = CT(ct).P.stack;

if BAYS(bay).ct_id(tier,stack) ~= ct
   disp('Searching for wrong ct')
   keyboard
end

hstack = sum(BAYS(bay).matriz(:,stack));
% if tier < hstack
%     disp('we have reshuffles');
%     keyboard
% end

for t = tier:hstack-1
   mct = BAYS(bay).ct_id(t+1,stack);
   CT(mct).P.tier = CT(mct).P.tier -1;
      BAYS(bay).matriz(t,stack) = BAYS(bay).matriz(t+1,stack);
      BAYS(bay).ct_id(t,stack) = BAYS(bay).ct_id(t+1,stack);
      BAYS(bay).ct_arrival(t,stack) = BAYS(bay).ct_arrival(t+1,stack);
      %BAYS(bay).R.S.cts(t,stack)= BAYS(bay).R.S.cts(t+1,stack);
end
BAYS(bay).matriz(hstack,stack) = 0;
BAYS(bay).ct_id(hstack,stack) = 0;
BAYS(bay).ct_arrival(hstack,stack) = 0;
%BAYS(bay).R.S.cts(hstack,stack) = 0;

BAYS(bay).empty_slots = BAYS(bay).empty_slots + 1;
BAY_change_reservation(bay,CT(ct).vs,-1);

BAYS_check_empty(CT(ct).P.bay);
%check_port_reservation(bay);
```

```
function CT_generate(flow,vs)

global CT COUNT TRF TIME VS
%keyboard
ct = COUNT.ct + 1;

COUNT.ct = ct;
CT(ct).weight = CT_weight(random('unif',TRF.CT.cdf(1),1));
CT(ct).class = CT_class(CT(ct).weight);
CT(ct).type = flow;
CT(ct).events.arrival = TIME.t;
CT(ct).ciclo.stack = 0;
CT(ct).ciclo.delivery = 0;
target_port = VS(vs).port;
CT(ct).vs = vs;
%CT(ct).port = VS(vs).port;
CT(ct).R.provocked.no = 0;
CT(ct).R.provocked.T.tier(1) = 0;
CT(ct).R.provocked.T.stack(1) = 0;
CT(ct).R.suffered.no = 0;
CT(ct).R.readytogo = 0;
CT(ct).events.no = 1;
CT(ct).events.time(1) = TIME.t;

if strcmp(flow,'IMP') == 1
   COUNT.IMP.ct = COUNT.IMP.ct + 1;
   COUNT.inventory.imp(target_port) = COUNT.inventory.imp(target_port) + 1;
   CT(ct).events.event{1} = 'vesselD';
elseif strcmp(flow,'EXP') ==1
   COUNT.EXP.ct = COUNT.EXP.ct + 1;
   COUNT.inventory.exp(target_port) = COUNT.inventory.exp(target_port) + 1;
   CT(ct).events.event{1} = 'gateIN';
end
```

```
function []=CT_get(bays,T)
```

```
function CT_init()

global CT

CT.weight = 0;
CT.class = 0;
CT.type = 0;
```

```
function [ET]=CT_order(ET)
% Takes the last ET and places it in the correct position

n=length(ET); % minum 2
arrival_time = ET(n).arrival_time;

pos=1;

for j=1:n-1
   if arrival_time>ET(j).arrival_time
      pos=pos+1;
   end
end

if pos ~= n % If the container needs to be moved to pos
   % Save a copy
   aux=ET(n);
   % Move the containers
   for j=n:-1:pos+1
      ET(j)=ET(j-1);
   end
   ET(pos)=aux;
end
```

```
function CT_remove(ct)
% This function takes a CT from the Bay. All the containers on top of that
% CT will go down one position, according to the following Scheme

global BAYS BL BT COST CT SPEED T VS YC
keyboard
bay = CT(ct).P.bay;
tier = CT(ct).P.tier;
stack = CT(ct).P.stack;
vs = CT(ct).vs;
yc = CT(ct).P.yc;

% Remove CT from crane
YC(yc).WL.ct = 0;
YC(yc).WL.move = '';
%  Compute the cost of the operation for YC and YT
%keyboard
if strcmp(CT(ct).type,'IMP') == 1
   distance = ET_dist_calculator(T.gate.position, BAYS(bay).position);
   traveltime = distance / SPEED.ET.travel;
   COST.ET.IMP.total = COST.ET.IMP.total + distance * COST.ET.travel;
elseif strcmp(CT(ct).type,'EXP') == 1
   distance = distance_calculator(BAYS(bay).position, BT(VS(vs).berth).position);
   traveltime = distance / SPEED.YT.travel;
   %distance = 2*distance_calculator(BAYS(target_bay).position,BT(VS.berth).position);
   COST.YT.EXP.total = COST.YT.EXP.total + distance * COST.YT.travel;
end

CT_addevent(ct,'picked',0);
CT_addevent(ct,'exit',traveltime);

old_bay = BAYS(bay);
% Inv_Tier
%  1       | | This is an empty stack
%  2       |x| This is the "top CT"
%  3       |x| Normal CT
%  4       |O| This is the "target CT" = tier
%  5       |x| Normal CT
%------------------------------

top_ct_no = sum(BAYS(bay).matriz(:,stack));

% We want to go from "Target CT" to "top_CT"
top_ct = BL.tiers;
if tier < top_ct_no

   vect_ct_m = BAYS(bay).matriz(tier+1:top_ct,stack);
   vect_ct_id = BAYS(bay).ct_id(tier+1:top_ct,stack);
   vect_ct_at = BAYS(bay).ct_arrival(tier+1:top_ct,stack);
   vect_port = BAYS(bay).port(tier+1:top_ct,stack); % Port
   vect_wc = BAYS(bay).weightc(tier+1:top_ct,stack);
   vect_vs = BAYS(bay).vs(tier+1:top_ct,stack);
   %
```

```
    BAYS(bay).matriz(tier:top_ct-1,stack) = vect_ct_m;
    BAYS(bay).ct_id(tier:top_ct-1,stack) = vect_ct_id;
    BAYS(bay).ct_arrival(tier:top_ct-1,stack) = vect_ct_at;
    BAYS(bay).port(tier:top_ct-1,stack) = vect_port;
    BAYS(bay).weightc(tier:top_ct-1,stack) = vect_wc;
    BAYS(bay).vs(tier:top_ct-1,stack) = vect_vs;
end

BAYS(bay).matriz(top_ct,stack) = 0;
BAYS(bay).ct_id(top_ct,stack) = 0;
BAYS(bay).ct_arrival(top_ct,stack) = 0;
BAYS(bay).port(top_ct,stack) = 0;
BAYS(bay).weightc(top_ct,stack) = 0;
BAYS(bay).vs(top_ct,stack) = 0;

BAY_change_reservation(bay,vs,-1);

% d) Update the number of empty slots
BAYS(bay).empty_slots = BAYS(bay).empty_slots + 1;

% e) Check emtpy bays
BAYS_check_empty(bay);

check_port_reservation(bay);

YC(yc).active = 0;
YC(yc).nextevent = 1000000;

check_port_occupation(bay)
```

```
function CT_remove(bay,tier,stack)
% This function takes a CT from the Bay. All the containers on top of that
% CT will go down one position, according to the following Scheme

global BAYS BL

keyboard

%  1       |x| This is the "top CT"
%  2       |O| This is the "target CT" = tier
%  3       |x| Normal CT
%  4       |x| Normal CT
%  5       | | This is an empty stack
%------------------------------

top_ct = sum(BAYS(bay).matriz(:,stack));

vect_ct_m = BAYS(bay).matriz(tier+1:BL.tiers,stack);
vect_ct_id = BAYS(bay).ct_id(tier+1:BL.tiers,stack);
vect_ct_at = BAYS(bay).ct_arrival(tier+1:BL.tiers,stack);
vect_port = BAYS(bay).port(tier+1:BL.tiers,stack); % Port
vect_wc = BAYS(bay).weightc(tier+1:BL.tiers,stack);
vect_vs = BAYS(bay).vs(tier+1:BL.tiers,stack);

BAYS(bay).matriz(tier:BL.tiers-1,stack) = vect_ct_m;
BAYS(bay).ct_id(tier:BL.tiers-1,stack) = vect_ct_id;
BAYS(bay).ct_arrival(tier:BL.tiers-1,stack) = vect_ct_at;
BAYS(bay).port(tier:BL.tiers-1,stack) = vect_port;
BAYS(bay).weightc(tier:BL.tiers-1,stack) = vect_wc;
BAYS(bay).vs(tier:BL.tiers-1,stack) = vect_vs;

BAYS(bay).matriz(BL.tiers,stack) = 0;
BAYS(bay).ct_id(BL.tiers,stack) = 0;
BAYS(bay).ct_arrival(BL.tiers,stack) = 0;
BAYS(bay).port(BL.tiers,stack) = 0;
BAYS(bay).weightc(BL.tiers,stack) = 0;
BAYS(bay).vs(BL.tiers,stack) = 0;

% d) Update the number of empty slots
BAYS(bay).empty_slots = BAYS(bay).empty_slots+1;

% e) Check emtpy bays
BAYS_check_empty(bay);

check_port_reservation(bay);
```

```
function CT_remove2(ct)
% This function takes a CT from the Bay. All the containers on top of that
% CT will go down one position, according to the following Scheme

global BAYS BT COST COUNT CT R SPEED T TIME VS
%keyboard

bay = CT(ct).P.bay;
old_bay = BAYS(bay);
check_bay_es(CT(ct).P.bay);
tier = CT(ct).P.tier;
stack = CT(ct).P.stack;
vs = CT(ct).vs;
% Check whether the vessel has left
if strcmp(CT(ct).type,'EXP')==1
   for i=1:3
      btvs(i)=BT(i).vessel;
   end
   isthere = find(btvs == vs);
   if isempty(isthere)
      disp('Error the vessel is not there')
      keyboard
   end
end

yc = CT(ct).P.yc;

%  Compute the cost of the operation for YC and YT
%keyboard
if strcmp(CT(ct).type,'IMP') == 1
   distance = ET_dist_calculator(T.gate.position, BAYS(bay).position);
   traveltime = distance / SPEED.ET.travel;
   COST.ET.IMP.total = COST.ET.IMP.total + distance * COST.ET.travel;
   COUNT.deliveries.imp(vs)=COUNT.deliveries.imp(vs)+1;
   m = COUNT.deliveries.imp(vs);
   n = COUNT.inventory.imp(vs);
   if COUNT.deliveries.imp(vs)>COUNT.inventory.imp(vs)
      keyboard
   end
elseif strcmp(CT(ct).type,'EXP') == 1
   distance = distance_calculator(BAYS(bay).position, BT(VS(vs).berth).position);
   traveltime = distance / SPEED.YT.travel;
   %distance = 2*distance_calculator(BAYS(target_bay).position,BT(VS.berth).position);
   COST.YT.EXP.total = COST.YT.EXP.total + distance * COST.YT.travel;
   COUNT.deliveries.exp(vs)=COUNT.deliveries.exp(vs)+1;
   m = COUNT.deliveries.exp(vs);
   n = COUNT.inventory.exp(vs);
end

disp([char(CT(ct).type) ' CT(' num2str(ct) ') VS(' num2str(vs) ') Delivery YC(' num2str(yc) ') ' num2str(m) ' out of '
num2str(n)])

CT_addevent(ct,'picked',0);
```

```
CT_addevent(ct,'exit',traveltime);

top_tier = sum(BAYS(bay).matriz(:,stack));

% We want to go from "Target CT" to "top_CT"
BAY= BAYS(bay).ct_id;

for t = top_tier:-1:tier+1
   %keyboard
   %transct = BAYS(bay).ct_id(t,stack);
   transct = BAY(t,stack);
   [BAY,ETM,P] = CT_reshuffle1(t,stack,BAY,yc);
   CT(transct).R.suffered.no = CT(transct).R.suffered.no + 1;
   CT_erase(transct);
   CT_write(transct,P);
end

CT_erase(ct);
YC_add_event(ct);

BAY_change_reservation(bay,vs,-1);

% d) Update the number of empty slots
BAYS(bay).empty_slots = BAYS(bay).empty_slots + 1;

% e) Check emtpy bays
BAYS_check_empty(bay);
check_bay_es(bay);
%check_port_reservation(bay);
check_port_occupation(bay)

if strcmp(CT(ct).type,'EXP')==1
   VS(vs).oct = VS(vs).oct + 1;
   COUNT.vsarrivals.exp(vs) = COUNT.vsarrivals.exp(vs) +1;
   R.vsct.exp(VS(vs).port,COUNT.vsarrivals.exp(VS(vs).port)) = TIME.t + TIME.delt;
   % Last CT
   if VS(vs).oct == VS(vs).OC
      disp([ num2str(COUNT.vsarrivals.exp(VS(vs).port)) 'CTs unloaded out of ' num2str(VS(vs).OC)])
      %BAY_unreserve(VS(vs).port)
      plot_bays
      plot_evolution()
      disp(['End of VS ' num2str(vs) ' upload operation'])
      BT(VS(vs).berth).active = 0;
      BT(VS(vs).berth).vessel = 0;
      [T.state.IMP,T.state.EXP,T.state.NAS] = terminal_state();
      disp(['# EXP CTs' num2str(T.state.EXP.no.cts)])
      disp(['# IMP CTs' num2str(T.state.IMP.no.cts)])
   end
end
```

```
function [BAY,R] = CT_reshuffle(tier,stack,BAY,yc,W)
% This function takes a CT in one position and moves it to another position
% within the bay
% The ASC is already located at the bay but it has not made the trolley
% movement yet

global BAYS BL TIME TRF YC

% Positions: P1 is the crane postion, P2 is the CT position, P3 is the
% target position hat will be evaluated

%keyboard
P2.tier = tier; P2.stack= stack;

%h = sum(BAY.slots);

max_E = 0; max_t = 0; %keyboard
ct = BAY.cts(P2.tier,P2.stack);

DELT = ones(1,BL.stacks)*1000;
E = ones(1,BL.stacks)*100000000;

for stack = 1:BL.stacks

   % The original positions shall be refreshed in each loop

   P1 = YC(yc).P;

   % Check wether the stack can have the ct
   alturas = find(BAY.cts(:,stack)>0);
   if isempty(alturas)
      h(stack) = 0;
   else
      h(stack) = alturas(end);
      if h(stack) == BL.tiers
         continue
      end
   end
   if stack == P2.stack
      continue
%     elseif BAYS(P1.bay).portres.no > 0
%         continue
   end

   m=0;

   r(stack).bay = P1.bay; % initially the crane is at the bay
   r(stack).stack = P1.stack;
   r(stack).tier = P1.tier;
   %r(stack).time = P1.time;
   r(stack).time = 0;
   r(stack).e = 0;
   r(stack).moves = '';
```

```
    r(stack).ct = 0;
    r(stack).ctmove = 0;

    % Determine the upper position
    tier = h(stack)+1;
    % Determine the ernergy and time
    m = m+1;  r(stack).bay(m) = P1.bay; r(stack).stack(m)= P2.stack; r(stack).tier = BL.tiers + 1; % O.tier;
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(P1.stack,P2.stack,ct,'utrolley','empty');
    P1.stack = P2.stack;   %[asc] = ASC_move(asc,P1,0,W);
    r(stack).ct(m) = 0; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack) = addposition(r(stack),'hoist',P2.tier);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(BL.tiers+1,P2.tier,ct,'ulower','empty');
    P1.tier = P2.tier;    %[asc] = ASC_move(asc,P1,0,W);
    r(stack).ct(m) = 0; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack) = addposition(r(stack),'hoist',BL.tiers+1);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(P2.tier,BL.tiers+1,ct,'upickbl','full');
    P1.tier = BL.tiers+1; %[asc] = ASC_move(asc,P1,ct,W);
    r(stack).ct(m) = ct; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack) = addposition(r(stack),'trolley',stack);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(P2.stack,stack,ct,'utrolley','full');
    P1.stack = stack;     %[asc] = ASC_move(asc,P1,ct,W);
    r(stack).ct(m) = ct; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack)=addposition(r(stack),'hoist',tier);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(BL.tiers+1,tier,ct,'udropbl','full');
    P1.tier = tier;        %[asc] = ASC_move(asc,P1,ct,W);
    r(stack).ct(m) = ct; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack)=addposition(r(stack),'hoist',BL.tiers+1);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(tier,BL.tiers+1,ct,'uraise','empty');
    P1.tier = BL.tiers+1;  %[asc] = ASC_move(asc,P1,0,W);
    r(stack).ct(m) = 0; r(stack).ctmove(m) = ct;

    DELT(stack) = sum(r(stack).time); E(stack) = sum(r(stack).e);

end
%keyboard

coefs = DELT/max(DELT)*TRF.PARAM.weight.t + E/max(E)*TRF.PARAM.weight.E;

[c_min,T.stack] = min(coefs);
T.tier = h(T.stack) +1;

energy = E(T.stack);
time = DELT(T.stack);

BAY.cts(T.tier,T.stack) = ct;
BAY.slots(T.tier,T.stack) = 1;
BAY.cts(P2.tier,P2.stack) = 0;
BAY.slots(P2.tier,P2.stack) = 0;
%BAY.time = BAY.time + time;
```

```
%T.bay = BAY.cts;
%T.time = TIME.t + time; %BAY.time;

% Write the results
R.time = r(T.stack).time;
R.E = r(T.stack).e;
R.moves = r(T.stack).moves;
R.bay = r(T.stack).bay;
R.stack = r(T.stack).stack;
R.tier = r(T.stack).tier;
R.ct = r(T.stack).ct;
R.ctmove = r(T.stack).ctmove;
%[asc] = ASC_move(asc,T,ct,0);
```

```
function [BAY,R,T] = CT_reshuffle1(tier,stack,BAY,yc)
% This function takes a CT in one position and moves it to another position
% within the bay
% The ASC is already located at the bay but it has not made the trolley
% movement yet

global BAYS BL TRF YC

% Positions: P1 is the crane postion, P2 is the CT position, P3 is the
% target position hat will be evaluated

%keyboard
P2.tier = tier; P2.stack= stack; O = P2;

%h = sum(BAY.slots);

max_E = 0; max_t = 0; %keyboard
ct = BAY(P2.tier,P2.stack);

DELT = ones(1,BL.stacks)*10000;
E = ones(1,BL.stacks)*100000000;

for stack = 1:BL.stacks

   % The original positions shall be refreshed in each loop

   P1 = YC(yc).P;

   % Check wether the stack can have the ct
   alturas = find(BAY(:,stack)>0);
   if isempty(alturas)
      h(stack) = 0;
   else
      h(stack) = alturas(end);
      if h(stack) == BL.tiers
         continue
      end
   end
   if stack == P2.stack
      continue
%     elseif BAYS(P1.bay).portres.no > 0
%         continue
   end

   m=0;

   r(stack).bay = P1.bay; % initially the crane is at the bay
   r(stack).stack = P1.stack;
   r(stack).tier = P1.tier;
   %r(stack).time = P1.time;
   r(stack).time = 0;
   r(stack).e = 0;
   r(stack).moves = '';
```

```
    r(stack).ct = 0;
    r(stack).ctmove = 0;

    % Determine the upper position
    tier = h(stack)+1;
    % Determine the ernergy and time
    m = m+1;  r(stack).bay(m) = P1.bay; r(stack).stack(m)= P2.stack; r(stack).tier = BL.tiers + 1; % O.tier;
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(P1.stack,P2.stack,ct,'utrolley','empty');
    P1.stack = P2.stack;   %[asc] = ASC_move(asc,P1,0,W);
    r(stack).ct(m) = 0; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack) = addposition(r(stack),'hoist',P2.tier);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(BL.tiers+1,P2.tier,ct,'ulower','empty');
    P1.tier = P2.tier;    %[asc] = ASC_move(asc,P1,0,W);
    r(stack).ct(m) = 0; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack) = addposition(r(stack),'hoist',BL.tiers+1);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(P2.tier,BL.tiers+1,ct,'upickbl','full');
    P1.tier = BL.tiers+1; %[asc] = ASC_move(asc,P1,ct,W);
    r(stack).ct(m) = ct; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack) = addposition(r(stack),'trolley',stack);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(P2.stack,stack,ct,'utrolley','full');
    P1.stack = stack;     %[asc] = ASC_move(asc,P1,ct,W);
    r(stack).ct(m) = ct; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack)=addposition(r(stack),'hoist',tier);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(BL.tiers+1,tier,ct,'udropbl','full');
    P1.tier = tier;        %[asc] = ASC_move(asc,P1,ct,W);
    r(stack).ct(m) = ct; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack)=addposition(r(stack),'hoist',BL.tiers+1);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = YC_energy(tier,BL.tiers+1,ct,'uraise','empty');
    P1.tier = BL.tiers+1;  %[asc] = ASC_move(asc,P1,0,W);
    r(stack).ct(m) = 0; r(stack).ctmove(m) = ct;

    DELT(stack) = sum(r(stack).time); E(stack) = sum(r(stack).e);

end
%keyboard

coefs = DELT/max(DELT)*TRF.PARAM.weight.t + E/max(E)*TRF.PARAM.weight.E;

[c_min,T.stack] = min(coefs);
if T.stack == P2.stack
   disp('CT can not go to the same stack')
   keyboard
end
T.tier = h(T.stack) +1;

energy = E(T.stack);
time = DELT(T.stack);

BAY(T.tier,T.stack) = ct;
```

```
% BAY.slots(T.tier,T.stack) = 1;
BAY(P2.tier,P2.stack) = 0;
% BAY.slots(P2.tier,P2.stack) = 0;
% BAY.time = BAY.time + time;
% T.bay = BAY.cts;
% T.time = TIME.t + time; %BAY.time;

% Write the results
R.time = r(T.stack).time;
R.E = r(T.stack).e;
R.moves = r(T.stack).moves;
R.bay = r(T.stack).bay;
R.stack = r(T.stack).stack;
R.tier = r(T.stack).tier;
R.ct = r(T.stack).ct;
R.ctmove = r(T.stack).ctmove;
%[asc] = ASC_move(asc,T,ct,0);

% % place CT
% BAYS(bay).matriz(T.tier,T.stack) = BAYS(bay).matriz(O.tier,O.stack);
% BAYS(bay).ct_id(T.tier,T.stack) = BAYS(bay).ct_id(O.tier,O.stack);
% BAYS(bay).ct_arrival(T.tier,T.stack) = BAYS(bay).ct_arrival(O.tier,O.stack);
% BAYS(bay).port(T.tier,T.stack) = BAYS(bay).port(O.tier,O.stack);
% BAYS(bay).weightc(T.tier,T.stack) = BAYS(bay).weightc(O.tier,O.stack);
% %BAYS(bay).vs(T.tier,T.stack) = BAYS(bay).vs(O.tier,O.stack);
% % remove CT
% BAYS(bay).matriz(O.tier,O.stack) = 0;
% BAYS(bay).ct_id(O.tier,O.stack) = 0;
% BAYS(bay).ct_arrival(O.tier,O.stack) = 0;
% BAYS(bay).port(O.tier,O.stack) = 0;
% BAYS(bay).weightc(O.tier,O.stack) = 0;
% %BAYS(bay).vs(O.tier,O.stack) = 0;
```

```
function CT_stack(ct)

global BAYS BL BT COUNT CT T TIME YC VS

%keyboard

bay = CT(ct).P.bay;
vs= CT(ct).vs;
check_bay_es(CT(ct).P.bay);


if strcmp(CT(ct).type,'EXP')==1
   COUNT.stacks.exp(vs)=COUNT.stacks.exp(vs)+1;
%    c = COUNT.inventory.exp(vs);
%    d = VS(vs).OC;
%    if COUNT.inventory.exp(vs) == VS(vs).OC;
%        VS(vs).seed=1;
%        keyboard
%    end
   m= COUNT.stacks.exp(vs); n = VS(vs).OC;
elseif strcmp(CT(ct).type,'IMP')==1
   COUNT.stacks.imp(vs)=COUNT.stacks.imp(vs)+1;
%    c= VS(vs).ict;
%    d= VS(vs).IC;
   m= COUNT.stacks.imp(vs); n = VS(vs).IC;
end

disp([char(CT(ct).type) ' CT(' num2str(ct) ') VS(' num2str(CT(ct).vs) ') Stacked on BAY(' num2str(bay) ') ' num2str(m) '
out of ' num2str(n)])
% num2str(c) ' cts stacked out of ' num2str(d)

tier = CT(ct).P.tier;
stack = CT(ct).P.stack;
if BAYS(bay).ct_id(tier,stack) >0
   disp('Overwriting CT')
   keyboard
end

% First, check whether the pile has a different height due to the arrival
% of CTs prior to the stacking of this in its reserved position
CT_addevent(ct,'stacked',0);
YC_add_event(ct);

%arrived_cts = COUNT.inventory.exp(port);
old_bay = BAYS(bay);


if BAYS(bay).R.S.cts(tier,stack) == 0
   disp('We need to change the Port reservation')
   keyboard
end

BAYS(bay).matriz(tier,stack) = 1;
```

```
BAYS(bay).ct_id(tier,stack) = ct;
BAYS(bay).ct_arrival(tier,stack) = TIME.t;
% Empty slots are accounted for when the crane calculates the cycle for the CT
BAYS(bay).empty_slots = BAYS(bay).empty_slots - 1;
check_bay_es(CT(ct).P.bay);
if BAYS(bay).empty_slots < BL.tiers
   keyboard
end

%BAYS(bay).port(tier,stack) = port;
%BAYS(bay).weightc(tier,stack) = CT(ct).class;

if strcmp(BAYS(bay).id,'NAS') == 1
   BAYS(bay).id = CT(ct).type;
end

if strcmp(BAYS(bay).mixing,'U') == 1
   keyboard
end

%check_port_reservation(bay);
check_bay_es(bay);
check_port_occupation(bay);
```

```
function [] = CT_translate(receiving_bay,giving_bay,ct)
% this function takes a CT from initial bay to final bay

global BAYS BL YC

% Determine bay dimensions
[rows,cols] = size(BAYS(1).matriz);

row_found = 'N'; cts_row = 0;
g_row = 0;

% Find container in the giving bay going from top to bottom
while row_found == 'N'
   g_row = g_row + 1;
   if g_row > BL.tiers
      disp('CT translate Error')
      keyboard
   end
   cts_row = sum(BAYS(giving_bay).matriz(g_row,:));
   if cts_row > 0
      row_found = 'Y';
      col_found = 'N';
      while col_found == 'N'
         g_col = fix(random('Uniform',1,cols+1));
         if BAYS(giving_bay).matriz(g_row,g_col) == 1
            col_found = 'Y';
         end
      end
   end
end

% Find CT in the receiving bay

col_found = 'N';

while col_found == 'N'
   r_col = fix(random('Uniform',1,cols + 1));
   cts_stack = sum(BAYS(receiving_bay).matriz(:,r_col));
   if  cts_stack < rows
      col_found = 'Y';
   end
end

r_row = rows-cts_stack;

% Move CT to receiving bay
BAYS(receiving_bay).matriz(r_row,r_col) = BAYS(giving_bay).matriz(g_row,g_col);
BAYS(receiving_bay).ct_id(r_row,r_col) = BAYS(giving_bay).ct_id(g_row,g_col);
BAYS(receiving_bay).ct_arrival(r_row,r_col) = BAYS(giving_bay).ct_arrival(g_row,g_col);
BAYS(receiving_bay).empty_slots = BAYS(receiving_bay).empty_slots - 1;

% Remove CT from giving bay
BAYS(giving_bay).matriz(g_row,g_col) = 0;
```

```
BAYS(giving_bay).ct_id(g_row,g_col) = 0;
BAYS(giving_bay).ct_arrival(g_row,g_col) = 0;
BAYS(giving_bay).empty_slots = BAYS(giving_bay).empty_slots +1;
% Check whether bay is empty
if BAYS(giving_bay).empty_slots == BL.capacity
    BAYS(giving_bay).id = 'NAS';
    BAYS(giving_bay).port = 0;
end

% Compute the cost of the operation
% Find the closest YC
[target_yc] = YC_select(giving_bay);
ct =  BAYS(giving_bay).ct_id(g_row,g_col);
[YC(target_yc)] =
YC_assign_ct(YC(target_yc),ct,BAYS(giving_bay),g_row,g_col,'trans',15000,BAYS(receiving_bay),r_row,r_col);
```

```
function [weight] = CT_weight(seed)

global TRF

if seed == 1
   weight = 32500;
else
   cdf  = TRF.CT.cdf;
   psup = 1;
   while seed >  TRF.CT.cdf(psup)
      psup = psup+1;
   end
   pinf = psup - 1;
   % linear interpolation between values
   delp = TRF.CT.cdf(psup)-TRF.CT.cdf(pinf);
   delw = TRF.CT.top_xf(psup)-TRF.CT.top_xf(pinf);

   incp =  seed - TRF.CT.cdf(pinf);
   incw = incp*delw/delp;

   weight = TRF.CT.top_xf(pinf) + incw;
end
```

```
function CT_write(ct,P)

global BAYS CT

bay = CT(ct).P.bay;

if BAYS(bay).ct_id(P.tier,P.stack) > 0
   'A ct has been placed before due to previous reshuffles'
   keyboard
end

BAYS(bay).matriz(P.tier,P.stack) = 1;
BAYS(bay).ct_id(P.tier,P.stack) = ct;
BAYS(bay).ct_arrival(P.tier,P.stack) = CT(ct).events.time(1);
%BAYS(bay).port(P.tier,P.stack) =CT(ct).port;
%check_port_reservation(bay);
%BAYS(bay).weightc(P.tier,P.stack) = CT(ct).weight;
BAYS(bay).R.S.cts(P.tier,P.stack) = ct;

CT(ct).P.tier = P.tier;
CT(ct).P.stack = P.stack;
CT_addevent(ct,'reshuffle',0);
```

```
function [C,R] = cycle_ini()

R.time = 0;
R.E = 0;
R.moves = '';
R.bay=0;
R.stack = 0;
R.tier =0;
R.ct = 0;

C.bay = 0;
C.stack = 0;
C.tier = 0;
C.time = 0;
C.E = 0;
C.moves = '';
C.go = 0;
%C.back =0;
```

```
function [distance] = distance_calculator(A,B)

% This function calculates the distance between point A and B
% Consideration shall be given to the relative position of A and B to know
% the direction of travel.

global BL T
%keyboard
inter_x = BL.length + T.aisles.vertical.width;

dely=A(2)-B(2);

if dely<0
   aux = B;
   B = A;
   A = aux;
end

dely = A(2)-B(2);

% Find the aisle closest to bay on the left
x_min = min(A(1),B(1));
x_max = max(A(1),B(1));

% Initialize leftmost distance: start in the middle of the left side aisle
x_izq = T.aisles.sides.width/2;
laisle=0;

while x_min-x_izq > inter_x
   laisle = laisle+1;
   x_izq = T.aisles.sides.width+laisle*inter_x-T.aisles.vertical.width/2;
end

% Find the aisle closest to the berth (position A) on the right
% Initialize rightmost distance: start in the middle of the right side aisle
x_dch = T.length - T.aisles.sides.width/2;
raisle=0;
while x_dch - x_max > inter_x
   raisle = raisle+1;
   x_dch = T.length-T.aisles.sides.width-raisle*inter_x+T.aisles.vertical.width/2;
end

% Calculate distance
distance = 2*(dely+x_dch-x_izq);
if distance<0
   'hey, negative distance'
   keyboard
elseif distance > 2*(T.length+T.width)
end
```

```
function [G,T,H,UG,UT,UH,UW] = Energybreakdown(C)

G.no = 0; T.no = 0; H.no = 0;
G.e = 0; T.e = 0; H.e = 0;
G.t = 0; T.t = 0; H.t = 0;
UG.no = 0; UT.no = 0; UH.no = 0;
UG.e = 0; UT.e = 0; UH.e = 0;
UG.t = 0; UT.t = 0; UH.t = 0;
UW.no = 0; UW.t = 0;

movements = C.moves;
energia = C.E;
tiempo = C.time;

le = length(energia);
lm = length(movements);

if le-lm> 0
   %keyboard
   for i = 1:le-lm
      a{i} = 'trans';
   end
   a{i+1} = 'trans';
   a = [a, movements(2:end)];
   movements = a;
end

for m = 1:le
   switch char(movements(m))
      % Productive moves

      case 'gantry'
         G.no = G.no + 1;
         G.e = G.e + energia(m);
         G.t = G.t + tiempo(m);
      case 'trolley'
         T.no = T.no + 1;
         T.e = T.e + energia(m);
         T.t = T.t + tiempo(m);
      case 'pickbf'
         H.no = H.no + 1;
         H.e = H.e + energia(m);
         H.t = H.t + tiempo(m);
      case 'raise'
         H.no = H.no + 1;
         H.e = H.e + energia(m);
         H.t = H.t + tiempo(m);
      case 'pickbl'
         H.no = H.no + 1;
         H.e = H.e + energia(m);
         H.t = H.t + tiempo(m);
      % Unproductive moves
      case 'transtranstrans'
```

```
        UG.no = UG.no + 1;
        UG.e = UG.e + energia(m);
        UG.t = UG.t + tiempo(m);
    case 'transtrans'
        UG.no = UG.no + 1;
        UG.e = UG.e + energia(m);
        UG.t = UG.t + tiempo(m);
    case 'trans'
        UG.no = UG.no + 1;
        UG.e = UG.e + energia(m);
        UG.t = UG.t + tiempo(m);
    case 'utrolley'
        UT.no = UT.no + 1;
        UT.e = UT.e + energia(m);
        UT.t = UT.t + tiempo(m);
    case 'upickbl'
        UH.no = UH.no + 1;
        UH.e = UH.e + energia(m);
        UH.t = UH.t + tiempo(m);
    case 'uraise'
        UH.no = UH.no + 1;
        UH.e = UH.e + energia(m);
        UH.t = UH.t + tiempo(m);
    case 'wait'
        UW.no = UW.no +1;
        UW.t = UW.t + tiempo(m);
  end
end
```

```
close all
no=60

h=2/no;
del=h/no
a(1)=h-del/2
for vs=2:no
    a(vs)=a(vs-1)-del
end
plot(a)
sum(a)
```

```
function [arrivals] = ET_arrivals(VS)

global TRF

et = 0; ec = 0; ic = 0;
et_arrivals = zeros(VS.OC,4); it_arrivals = zeros(VS.IC,4);

% 1. LIST of ET EXP ARRIVALS
% ------------------------------------------------------------------------
% The EXP ETs come from a few days before the  vessel arrival

exp_daily_arrivals = pyramid(TRF.PARAM.daysinadvance,VS.OC);
disp(['Generate ' num2str(exp_daily_arrivals) ' export arrivals'])
% The vessel can come at any time, but ET come fom 8 to 24. Then, find out
% when the vessel is planned and start moving ET the next day

% 1.1. Distribute the containers during the day
%keyboard
for day = 1:TRF.PARAM.daysinadvance
   t_ini = (VS.arrival.day -TRF.PARAM.daysinadvance -2 + day )*3600*24; %Before:
(vessel_plan_day+day)*3600*24;

   if day == TRF.PARAM.daysinadvance
      t_fin = VS.arrival.time- 3600;
   else
      t_fin = Inf;
   end
   % Generate a random number and use it to feed the daily arrival distribution
   for i = 1:exp_daily_arrivals(day)
      et = et + 1; ec = ec+1;
      % Set the arrival limit one hour before
      arrival_time = t_ini + et_daily_pdf(random('unif',0,1)); cont = 0;
      while arrival_time > t_fin %) or(arrival_time < VS.plan.time,
         arrival_time = t_ini + et_daily_pdf(random('unif',0,1));
         cont = cont +1;
         if cont >10000
            keyboard
         end
      end
      if arrival_time == 0
         keyboard
      end
      et_arrivals(et,1) = arrival_time;
      et_arrivals(et,2) = 1;
      et_arrivals(et,3) = VS.no;
      et_arrivals(et,4) = VS.port;
   end
end

% 2. LIST of IMP ET ARRIVALS
% ------------------------------------------------------------------------
% The IMP ETs start coming from the moment the vessel finishes downloading
et = 0;
```

```
%keyboard
% 2.1. Make the list of IMP containers coming every day
imp_daily_arrivals = pyramid(TRF.PARAM.daysofdischarge,VS.IC);
disp(['Generate ' num2str(imp_daily_arrivals) ' import arrivals'])
for day = 1:TRF.PARAM.daysofdischarge
   % Time of the vessel arrival
%    if day == 1
%       t_ini = ready2load;
%    else
%       t_ini = (vessel_arrival_day+day-1)*3600*24;
%    end
   t_ini = (VS.arrival.day+day-1)*3600*24;
   mintime = max(VS.operation.switch,(t_ini/3600+8)*3600);
   maxtime = (VS.arrival.day+day)*24*3600;
   for i = 1:imp_daily_arrivals(day)
      et = et + 1; ic=ic+1;
      departure_time = 0; cont = 0;
      while or(departure_time <= mintime, departure_time >= maxtime)
         departure_time = t_ini + et_daily_pdf(random('unif',0,1));
         cont = cont +1;
         if cont >10000
            keyboard
         end
      end
      if departure_time == 0
         keyboard
      end
      it_arrivals(et,1) = departure_time;
      it_arrivals(et,2) = 2;
      it_arrivals(et,3) = VS.no;
      it_arrivals(et,4) = VS.port;
   end
end
%keyboard
arrivals = [et_arrivals;it_arrivals];

% figure(1000); plot(VS.arrival.time/3600/24,'g>'); hold on; plot(VS.plan.time/3600/24,'r>');
% plot(sort(et_arrivals(:,1))/3600/24,'g.'); plot(sort(it_arrivals(:,1))/3600/24,'b.');

disp(['ET generated for VS ' num2str(VS.no) ' with ' num2str(ec) ' EXP CT and ' num2str(ic) ' IMP CT'])
disp('------------------------------------------------------------')
```

```
function [arrivals] = ET_arrivals(VS)

global TRF

et = 0; ec = 0; ic = 0;
et_arrivals = zeros(VS.OC,4); it_arrivals = zeros(VS.IC,4);

% 1. LIST of ET EXP ARRIVALS
% ------------------------------------------------------------------------
% The EXP ETs come from a few days before the  vessel arrival

exp_daily_arrivals = pyramid(3,VS.OC); TRF.PARAM.daysinadvance
disp(['Generate ' num2str(exp_daily_arrivals) ' export arrivals'])
% The vessel can come at any time, but ET come fom 8 to 24. Then, find out
% when the vessel is planned and start moving ET the next day

% 1.1. Distribute the containers during the day
keyboard
for day = 1:TRF.PARAM.daysinadvance
   t_ini = (VS.arrival.day -1 + day - 1)*3600*24; %Before: (vessel_plan_day+day)*3600*24;

   % Generate a random number and use it to feed the daily arrival distribution
   for i = 1:exp_daily_arrivals(day)
      et = et + 1; ec = ec+1;
      % Set the arrival limit one hour before
      arrival_time = 0; cont = 0;
      while or(arrival_time < VS.plan.time, arrival_time > VS.arrival.time- 3600)
         arrival_time = t_ini + et_daily_pdf(random('unif',0,1));
         cont = cont +1;
         if cont >10000
            keyboard
         end
      end
      if arrival_time == 0
         keyboard
      end
      et_arrivals(et,1) = arrival_time;
      et_arrivals(et,2) = 1;
      et_arrivals(et,3) = VS.no;
      et_arrivals(et,4) = VS.port;
   end
end

% 2. LIST of IMP ET ARRIVALS
% ------------------------------------------------------------------------
% The IMP ETs start coming from the moment the vessel finishes downloading
et = 0;
%keyboard
% 2.1. Make the list of IMP containers coming every day
imp_daily_arrivals = pyramid(TRF.PARAM.daysofdischarge,VS.IC);
disp(['Generate ' num2str(imp_daily_arrivals) ' import arrivals'])
for day = 1:TRF.PARAM.daysofdischarge
   % Time of the vessel arrival
```

```
%     if day == 1
%         t_ini = ready2load;
%     else
%         t_ini = (vessel_arrival_day+day)*3600*24;
%     end
    t_ini = (VS.arrival.day+day-1)*3600*24;
    mintime = max(VS.operation.switch,(t_ini/3600+8)*3600);
    maxtime = (VS.arrival.day+day)*24*3600;
    for i = 1:imp_daily_arrivals(day)
        et = et + 1; ic=ic+1;
        departure_time = 0; cont = 0;
        while or(departure_time <= mintime, departure_time >= maxtime)
            departure_time = t_ini + et_daily_pdf(random('unif',0,1));
            cont = cont +1;
            if cont >10000
                keyboard
            end
        end
        if departure_time == 0
            keyboard
        end
        it_arrivals(et,1) = departure_time;
        it_arrivals(et,2) = 2;
        it_arrivals(et,3) = VS.no;
        it_arrivals(et,4) = VS.port;
    end
end
%keyboard
arrivals = [et_arrivals;it_arrivals];

disp(['ET generated for VS ' num2str(VS.no) ' with ' num2str(ec) ' EXP CT and ' num2str(ic) ' IMP CT'])
disp('------------------------------------------------------------')
```

```
function [delt] = ET_cycle(operation,BAY,YC,tier,stack)

global BL S SPEED T

t=0;
switch operation
   case 'load' % pick CT from truck and drop it
      % Gantry
      i=i+1; t(i) = abs(YC.position-BAY.position(1))*SPEED.YC.gantry;
      % Spreader to truck
      i=i+1; t(i) = YC.spreader*S.w * SPEED.YC.spreader.empty
      % Hoist down to pick
      i=i+1; t(i) = BL.tiers * S.h * SPEED.YC.hoist.empty;
      % Hoist up
      i=i+1; t(i) = BL.tiers * S.h * SPEED.YC.hoist.loaded;
      % Spreader
      i=i+1; (i) = stack * S.w * SPEED.YC.spreader.loaded;
      % Hoist down loaded
      i=i+1; t(i) = (BL.tiers - tier) * S.h * SPPED.YC.hoist.loaded;
      % Hoist up again
      i=i+1; t(i) = (BL.tiers - tier) * S.h * SPPED.YC.hoist.empty;
   case 'unload'
      % Gantry to location
      i=i+1; t(i) = abs(YC.position-BAY.position(1))*SPEED.YC.gantry;
      % Spreader emtpy to stack
      i=i+1; t(i) = abs(YC.spreader - stack)*S.w * SPEED.YC.spreader.empty;
      % Hoist down to pick
      i=i+1; t(i) = BL.tiers * S.h * SPEED.YC.hoist.empty;
      % Hoist up
      i=i+1; t(i) = BL.tiers * S.h * SPEED.YC.hoist.loaded;
      % Spreader
      i=i+1; t(i) = stack * S.w * SPEED.YC.spreader.loaded;
      % Hoist down loaded
      i=i+1; t(i) = (BL.tiers - tier) * S.h * SPPED.YC.hoist.loaded;
      % Hoist up again
      i=i+1; t(i) = (BL.tiers - tier) * S.h * SPPED.YC.hoist.empty;
   case 'move'
      distance = ET_dist_calculator(T.gate.position,BAY.position);
      t = distance * SPEED.ET.travel;

end

delt= sum(t);
```

```
function [t]=et_daily_pdf(n)

% n is between 0 and 1

if n<0.15
   t0 = 7;  t = t0 + n*3/0.15;
elseif n<0.375
   t0 = 11; t = t0 + (n-0.15)/0.225*3;
elseif n<0.54
   t0 = 14; t = t0 + (n-0.375)/0.165*2;
else
   t0 = 16; t = t0 +(n-0.54)/0.46*(21 -t0);
end

if t == 0
   keyboard
end
t=t*3600;
```

```
function [distance] = ET_dist_calculator(A,B)

global T

delx=T.length-T.aisles.sides.width; %(A(1)-B(1));
dely=(A(2)-B(2));
distance = 2*(delx^2+dely^2)^0.5;
```

```
function ET_pick_CT(et)
% This function selects randomly a container from an
% IMP bay so that a a ET can take it

global BAYS COST COUNT CT ET T VS


vs = ET(et).target_VS;

% 1. Find a CT based on the time they arrived to the terminal
% 1.1 Search the time among the list
[ct] = cdf_stay(vs);

% 2. Compute reshuffles and swap containers
% [reshuffles,r_heights] = BAYS_reshuffles(ct);
% COST.reshuffle.IMP = COST.reshuffle.IMP + reshuffles;
% COST.r_heights.IMP = COST.r_heights.IMP + r_heights;

CT(ct).P.yc = YC_assign_ct(CT(ct).P.bay);

disp(['ET pick IMP CT(' num2str(ct) ') from Bay (' num2str(CT(ct).P.yc) ') Tier(' num2str(CT(ct).P.tier)  ') Stack('
num2str(CT(ct).P.stack) ')' ])

[C] = YC_calc_delivery(ct);
[G,Tr,H,UG,UT,UH,UW] = Energybreakdown(C);
COST.YC.IMP.deliver.gantry = COST.YC.IMP.deliver.gantry + G.e + UG.e;
COST.YC.IMP.deliver.trolley = COST.YC.IMP.deliver.trolley + Tr.e + UT.e;
COST.YC.IMP.deliver.hoist = COST.YC.IMP.deliver.hoist + H.e + UH.e;

%check_port_reservation(CT(ct).P.bay)
COUNT.deliveries.imp(vs) = COUNT.deliveries.imp(vs) + 1;

CT_erase(ct); %desreservation is here
```

```
function ET_stack(et)

global BAYS BL COST CT ET COUNT T TIME TRF VS

%keyboard
vs = ET(et).target_VS;

CT_generate('EXP',vs);

ct = COUNT.ct;

VS(vs).OC_arrived = [VS(vs).OC_arrived, ct];
% Stack CT with reservation
% ------------------------------------------------------------------------
if TRF.PARAM.reservation == 1
   if COUNT.vs_p < vs
      ET_stack_NR(ct,VS(vs));%
   else
      if length(VS(vs).OC_arrived) <= sum(VS(vs).plan.EXP.cts)
         ET_stack_R3(ct,VS(vs));
      else
         ET_stack_NR(ct,VS(vs));
      end
   end

% Stack CT with NO reservation
% ------------------------------------------------------------------------
else
   ET_stack_NR(ct,VS(vs));
end


% Assign a crane
CT(ct).P.yc = YC_assign_ct(CT(ct).P.bay);

% Allocate ct in the BAY
%-------------------------------------------------------------------------
BAY_individual_allocation(ct);

% Assign to YC
%-------------------------------------------------------------------------
[C] = YC_calc_stack(ct,0);
[G,Tr,H,UG,UT,UH,UW] = Energybreakdown(C);
COST.YC.EXP.stack.gantry = COST.YC.EXP.stack.gantry + G.e + UG.e;
COST.YC.EXP.stack.trolley = COST.YC.EXP.stack.trolley + Tr.e + UT.e;
COST.YC.EXP.stack.hoist = COST.YC.EXP.stack.hoist + H.e + UH.e;

% Check wether all the EXP cts associated to a vessel have come
% ------------------------------------------------------------------------
if COUNT.inventory.exp(CT(ct).vs) == VS(vs).OC
   disp(['All the CT for VS ' num2str(vs) ' have arrived to the Terminal'])
   disp('-----------------------------------------------')
   plot_evolution()
```

```
    plot_bays
end
```

```
function ET_stack_NR(ct,VS)

global CT

% 1 Search for bays with cts of the same group
%------------------------------------------------------------------------
[iscomplete,C_slots,CBAYS1] = BAYS_find_slots('EXP','N',VS,'ETdroprand',1);
if C_slots > 0
   BAY_selection_NR(ct,CBAYS1(1,:),'N');
end

% 2. Second, try NAS Bays
%------------------------------------------------------------------------
if iscomplete == false %and(available_slots > 0, available_slots < VS.OC)
   disp(['ET drop requests NAS bay for CT(' num2str(ct) ')'])
   [iscomplete,C_slots2,CBAYS2] = BAYS_find_slots('NAS','N',VS,'ETdroprand',1);
   if C_slots2 > 0
      BAY_selection_NR(ct,CBAYS2(1,:),'N');
   end
end

% 3. Third, try bays with mixing
%------------------------------------------------------------------------
if iscomplete == false
   [iscomplete,C_slots3,CBAYS3] = BAYS_find_slots('EXP','S',VS,'ETdroprandstacks',1); %'ETdroprandstacks'
   if C_slots3 > 0
      BAY_selection_NR(ct,CBAYS3(1,:),'S');
   else
      disp('Error') ; terminal_state;
      VS_inventory(VS.no);
      keyboard
   end
end

CT(ct).nr = 1;
```

```
function [target_bay] = ET_stack_R3(ct,VS)
% This function places a CT in a bay in the yard that has previously
% reserved

global BAYS BL COUNT CT T TRF

% 0. Check the reserve
[pure_ports,stack_ports,mixed_ports] = check_reserve();
slots_reserved = pure_ports(VS.port) + stack_ports(VS.port) + mixed_ports(VS.port);

arrived_cts = COUNT.inventory.exp(VS.port);
if arrived_cts + slots_reserved - VS.OC < 0
   disp('ET_drop Warning: not enough reserved slots in the yard for this vessel')
   %keyboard
end

% if strcmp(CT(ct).type,'IMP') == 1
%     ocup_limit = T.limits.bay.imp;
% elseif strcmp(CT(ct).type,'EXP') == 1
%     ocup_limit = T.limits.bay.exp;
% end

target_bay = 0; %keyboard

CB = VS.plan.EXP.bays;  Ccts = VS.plan.EXP.cts; nb = length(CB);

if nb > 0
   % 1. IDENTIFY THE SLOTS
   %keyboard
   b_slots= zeros(1,nb);     w_classes = ones(nb,BL.stacks)*1000;
   for i_bay = 1:nb
      bay = CB(i_bay);
      slots = 0;    peso = ones(1,BL.stacks)*1000;
      % Compare the bay capacity for that port with the port occupation
      [list,nctsvs] = BAY_get_port(bay,VS.no);
      %ocup_per_vs = length(find(BAYS(bay).R.ports == VS.no));
      if nctsvs >= Ccts(i_bay)
         continue
      end
      % Check total occupation
      if BAYS(bay).empty_slots <= BL.tiers
         continue
      end

      for stack = 1:BL.stacks
         altura_pila = Pile_height(bay,stack);
         if altura_pila < BL.tiers
            peso(stack) = BL.idealbay(altura_pila+1,stack);
            slots = slots + BL.tiers - altura_pila;
         end

      end
      b_slots(i_bay) = slots;
```

```
      w_classes(i_bay,:) = peso;
   end

   % 2. ASSIGN THE CT TO A SPECIFIC SLOT

   % 2.1 SELECT TARGET BAY
   %keyboard
   if strcmp(TRF.PARAM.idealbay_option,'MIN_OCCUPATION') == 1
      [empty_slots,bay_index] = max(b_slots);
      target_bay = CB(bay_index);
   elseif strcmp(TRF.PARAM.idealbay_option,'PESO') == 1
      for i_bay = 1: nb
         [weight_dif(i_bay),position(i_bay)] = min(abs(w_classes(i_bay,:) - CT(ct).class));
      end
      [wdif,t_bay] = min(weight_dif);
      target_bay = CB(t_bay);
      CT(ct).P.bay = target_bay;
%        CT(ct).P.stack = position(t_bay);
%        %tier = BL.tiers - sum(BAYS(target_bay).matriz(:,stack));
%        CT(ct).P.tier = sum(BAYS(target_bay).matriz(:,CT(ct).P.stack)) + 1;
%        if CT(ct).P.tier ==0
%           keyboard
%           disp('Etdrop3 warning: look for additional space'); keyboard
%           ET_drop_NR(ct,arrived_cts,VS);
%        end
   end

   disp(['EXP CT(' num2str(ct) ') VS(' num2str(VS.no) ') Stack: Bay(' num2str(target_bay) ') ' num2str(arrived_cts) '
CTs arrived out of ' num2str(VS.OC) ' /' num2str(slots_reserved) ' reservations left' ])
else
   disp(['ET_Drop_CT3: Error: not enough bays to accomodate EXP CT destined to vessel:' num2str(VS.no)])
   %[space_reserved] = plot_bays_reserved
   [T.STATE.IMP,T.STATE.EXP] = terminal_state();
   occup = 100*(T.state.EXP.no.cts + T.state.IMP.no.cts)/T.bays/BL.capacity
   disp(['Total Occupation = ' num2str(occup)])
   closure(ct)
   keyboard
end

[pure_ports2,stack_ports2,mixed_ports2] = check_reserve();
slots_reserved2 = pure_ports2(VS.port) + stack_ports2(VS.port) + mixed_ports2(VS.port);

% Check the reservation is ok
if slots_reserved2 - slots_reserved ~= 0
   'ETDrop 3 Reservation Error'
   pure_ports
   pure_ports2
   keyboard
end

CT(ct).wr = 1;
```

```
function [ET,CT]=ET_task(ET,T,B,CT,QC,bays)

% This function resolves stages I and II of the assignment problem of a
% container. Stage I resolves what bay the container goes to, and stage II
% solves the slot of that bay to which the container goes to.

% 1. Stage I
%-------------------------------------------------------------------------
% Several criteria can be used.
% - Equal bay occupation: choose the bay with less containers
% - Best container fit: choose the bay that has an empty slot of the same
% weigth class
% - Best QC fit: bay that has the optimum distance to the QC regarding the
% QC schedule.
% - Best YC fit: bay that show the best operative for YC. Very difficult
row=0;tier=0;

for bay=1:T.bays
   if bays(bay).port==CT.port % Compare class
      bay_min_C(bay)=sum(sum(bays(bay).matriz));
      [class_diff(bay),row,tier]=check_class_diff(bays(bay).matriz,B.idealbay,CT.class); %ideal bay type
      distance(bay)=distance_calculator(QC(CT.berth).position,bays(bay).position);
   else
      bay_min_C(bay)=30;
      class_diff(bay)=30;
      distance(bay)=T.length*2;
   end
end

% Three criteria have been calculated. Now select the best bay target for
% the container. We will use the less ocupation for now:

[bay_min_occup,ET.target_bay]=min(bay_min_C); % The ET has a bay assigned

% 2. Stage II
%-------------------------------------------------------------------------
% Now in stage II we must select the best slot in the bay
aux=1;CT.target_row=1;CT.target_tier=1;
[aux,CT.target_row,CT.target_tier]=check_class_diff(bays(ET.target_bay).matriz,B.idealbay,CT.class);

CT.target_bay=ET.target_bay;
ET.target_T_col=bays(ET.target_bay).T_col;
ET.target_T_row=bays(ET.target_bay).T_row;
ET.target_position(1)=bays(ET.target_bay).position(1);
ET.target_position(2)=bays(ET.target_bay).position(2);

%.........................................................................
% maybe not a bad idea to introduce a marker fot the bay slot to receive
% container so it is not used for anything else. For improvement!
%.........................................................................
```

```
function [traveltime] = ETYT_cost_calc(ct,move)

global BAYS BT COST CT SPEED T VS

vs = CT(ct).vs;
bay = CT(ct).P.bay;

if strcmp(CT(ct).type,'EXP') == 1
   if strcmp(move,'stack') == 1
      distance = ET_dist_calculator(T.gate.position, BAYS(bay).position);
      traveltime = distance / SPEED.ET.travel;
      COST.ET.EXP.total(end+1) = distance;% * COST.ET.travel;
   elseif strcmp(move,'delivery') == 1
      distance = distance_calculator(BAYS(bay).position, BT(VS(vs).berth).position);
      traveltime = distance / SPEED.YT.travel;
      %distance = 2*distance_calculator(BAYS(target_bay).position,BT(VS.berth).position);
      COST.YT.EXP.total(end+1) = distance;% * COST.YT.travel;
   end
elseif strcmp(CT(ct).type,'IMP') == 1
   if strcmp(move,'stack') == 1
      distance = distance_calculator(BAYS(bay).position, BT(VS(vs).berth).position);
      traveltime = distance / SPEED.YT.travel;
      %distance = 2*distance_calculator(BAYS(target_bay).position,BT(VS.berth).position);
      COST.YT.IMP.total(end+1) = distance;% * COST.YT.travel;
   elseif strcmp(move,'delivery') == 1
      distance = ET_dist_calculator(T.gate.position, BAYS(bay).position);
      traveltime = distance / SPEED.ET.travel;
      COST.ET.IMP.total(end+1) = distance;% * COST.ET.travel;
   end
end
```

```
clear all
clc
close all

%profile on

%matlabpool open 4

%trflevel = [1.2 1.6 1.8]*1000000;
trflevel = [2.0 ]*1000000;
delaydays = 2:-1:0;
%delaydays = [0];
for trf = 1: length(trflevel)

    % Whith no reservation
    TERMINALmainv2(0,0,trflevel(trf),1);

    % Whith reservation
    for d = 1:length(delaydays)
        TERMINALmainv2(1,delaydays(d),trflevel(trf),1);
    end
end
```

```
clear all
clc
close all

%profile on

%matlabpool open 4

%trflevel = [1.2 1.6 1.8]*1000000;
trflevel = [1.2 1.5 1.8]*1000000;
delaydays = 2:-1:0;

for trf = 1: length(trflevel)

    % Whith no reservation
    TERMINALmainv2(0,0,trflevel(trf),1);

    % Whith reservation
    for d = 1:3
        TERMINALmainv2(1,delaydays(d),trflevel(trf),1);
    end
end
```

```
function [yes_no]=figure_ckeck(fig_number)

list_figures=findobj('type','figure')';

yes_no=0;
for i=1:length(list_figures)
    if list_figures(i)==fig_number
        yes_no=1;
    end
end
```

```
clear
% Get all PDF files in the current folder
files = dir('*.m');
% Loop through each
for id = 1:length(files)
    % Get the file name (minus the extension)
    fn = files(id).name;
    if strcmp(fn(end-1:end),'.m')==1
    [~, g] = fileparts(fn);
      % Convert to number
      num = str2double(g);
      sourcef =  strcat(g ,'.m');
      destf = strcat(g , '.txt');
      copyfile(sourcef,destf,'f');
      %if ~isnan(num)
         % If numeric, rename
         %movefile(files(id).name, sprintf('%03d.pdf', num));

      end
end
```

```
function [berth,BT]=find_berth4vessel(BT,T,VS)

busy_berths=0;
for berth=1:T.berth_no
   if BT(berth).active==1
      busy_berths=busy_berths+1;
   end
end

empty_berths=T.berth_no-busy_berths;

if empty_berths>0
   target_berth_found='N';
   while target_berth_found=='N'
      berth=1+fix(random('unif',0,1))*T.berth_no;
      if BT(berth).active==0
         target_berth_found='Y';
         BT(berth).active=1;
         BT(berth).vessel=VS.no;
      end
   end
else
   'error: there are no empty berths'
   keyboard
end
```

```
function [CT]=generate_container(CT)


CT(CT_no).type='EXP';
CT(CT_no).assign='N';
CT(CT_no).port=fix(random('unif',0.5,no_ports+0.5));
%.....................................................................
CT(CT_no).berth=round(random('unif',1,T.berth_no)); % FIX
%CT(CT_no).quay=1; % FIX

%.....................................................................
CT(CT_no).owner='ET';
CT(CT_no).size=40;
CT(CT_no).weight=fix(maxw*random('unif',0,1));
CT(CT_no).class=C_class(top_xf,CT(CT_no).weight);
CT(CT_no).position(1)=T.length/2; % X coordinate of the gate
CT(CT_no).position(2)=0;          % Y coordinate of the gate
CT(CT_no).target_bay=0;
CT(CT_no).target_row=0;
CT(CT_no).target_tier=0;
```

```
function [B]=ideal_bay()

calc = 1;
if calc == 1
   % 5.1 Container weight type to be analyzed
   cont_category=40;
   if cont_category==40
      owl=load('tw40.dat');
      minw=4400;
      maxw=32500;
   elseif cont_category==20
      owl=load('tw20.dat');
      minw=2200;
      maxw=30480;
   end

   % 5.2 length of the list
   ol_w=length(owl);

   % 5.3 Remove the empty containers, if there are any
   l_w=0;
   for cont=1:ol_w
      if owl(cont)>minw;
         l_w=l_w+1;
         wl(l_w)=owl(cont);
      end
   end

   % 5.4 Probability density function
   no_wc=9;
   [xf,top_xf,cdf]=cdf_x2(no_wc,wl,minw,maxw);
   ndelx=(maxw-minw)/no_wc;
   low_xf=top_xf-ndelx;
   ndf(1)=cdf(1);
   for j=2:no_wc
      ndf(j)= cdf(j)-cdf(j-1);
   end

   % 5.5 Make ideal bay
   no_cont=round(T.initial_occupation*B.rows*B.tiers);
   vect_cont=container_vector2(ndf,no_cont);
   B.idealbay=midealbay_d(B.tiers,B.rows,vect_cont);
   %--------------------------------------------------------------------
end

%figure(1)
```

```
function [ct] = INIT(gsiono,rsiono,rdelay,trf_level)

global TRF VS

% 1. TEMPORAL PARAMETERS OF THE SIMULATION
%--------------------------------------------------------------------------
TIME_init()

% 2. Load or create traffic file
%--------------------------------------------------------------------------
TRF.PARAM.ct_traffic = trf_level; % Measured in CT (movements) per year
TRF.PARAM.average_no_IC = 1000; %average_no of IMP cont (download);
TRF.PARAM.average_no_OC = TRF.PARAM.average_no_IC; %average_no of EXP cont (upload);
TRF.PARAM.vs_traffic = fix(TRF.PARAM.ct_traffic/(TRF.PARAM.average_no_IC+TRF.PARAM.average_no_OC));
% number of vessels per year
TRF.PARAM.lambda_vs = 1/(TRF.PARAM.vs_traffic/365/24/60/60);
TRF.PARAM.idealbay_option = 'PESO'; %'MIN_OCCUPATION';

TRF.PARAM.daysinadvance = 3; % for EXP trucks
TRF.PARAM.daysofdischarge = 8;
TRF.PARAM.qcthroughput = 30; % Movements per hour
TRF.PARAM.qcdelt = 3600/TRF.PARAM.qcthroughput;
TRF.PARAM.generate = gsiono;
TRF.PARAM.reservation = rsiono;

trf_file_name = ['TRF ' num2str(TRF.PARAM.ct_traffic/1000000) 'MCTs.mat'];

if gsiono == 1 %GENERATE
   TRF_init(trf_file_name);
else
   load(trf_file_name);
   disp('Traffic file loaded')
end

% 1.7 Delay the reservation
% --------------------------------------------------------------------
TRF.PARAM.reservation = rsiono;
TRF.PARAM.delayreservation = rdelay; % In days
% --------------------------------------------------------------------
if and(TRF.PARAM.reservation == 1, TRF.PARAM.delayreservation > 0)
   for  vs = 1: TRF.PARAM.VS_no
      VS(vs).plan.time = VS(vs).plan.time + TRF.PARAM.delayreservation*3600*24;
   end
end

plot_TRF1()
%plot_trf(TRF,0)

CT_init()

COST_init()

T_initialization()
```

```
SPEED_init()

BT_initialization();

[ct] = BAYS_init();

YC_init();

COUNT_init();

Results_init();
```

```
function [bays]=marshalling[bays,T,target_bay,required_space]

for bay=1.T.bays
   if bays(bays).id=target_bay
      i=i+1;
      candidate_bays(i)=bay;
      candidate_bay_space(i)=bays(bay).empty_slots;
   end
end

% Sort the bays
```

```
function [col_ids,no_cols] = Matriz_empty_cols(BAY)
% This function calculates the empty colums

[rows,cols]= size(BAY.matriz);

no_cols = 0; col_ids =0;

for col = 1: cols
    if sum(BAY.matriz(:,col)) == 0
        no_cols= no_cols + 1;
        col_ids(no_cols) = col;
    end
end

if no_cols == 0
   disp('BAY_find_empty_col Error: no empty staks')
   BAY
end
```

```
function [positions,no_ceros] = Matriz_empty_slots(BAY,vs)
% this function searches for empty slots that are not reserved so that they
% can be reserved for IMP CTs
global BL
positions = 0;

no_ceros= 0;

% Calculate the port of comparison
[x,y] = find(BAY.port>0);
for i=1:length(x)
   portofcomp(i) = BAY.port(x(i),y(i));
end
portofcomp = unique(portofcomp);

if ismember(portofcomp ,vs)
   for stack = 1:BL.stacks
      for tier = 1:BL.tiers
         if and(BAY.matriz(tier,stack) == 0, BAY.port(tier,stack) == 0)
            no_ceros = no_ceros + 1;
            positions(no_ceros,1) = tier;
            positions(no_ceros,2) = stack;
         end
      end
   end
end

no_ceros = min(no_ceros, BL.capacity-BL.tiers);
```

```
function [ports] = Matriz_ports_reserved(BAY)
% This funciton calculates the number of reserved spaces for all the ports
% we search for slots reserved and but not having a CT
global BL TRF
ports = zeros(1,TRF.PARAM.no_ports);

for tier = 1:BL.tiers
   for stack = 1:BL.stacks
      bay_port = BAY.port(tier,stack)*BAY.matriz(tier,stack);
      if bay_port > 0
         ports(bay_port) = ports(bay_port) + 1;
      end
   end
end
```

```
function [idealbay] = midealbay_d(vect_cont)

clear idealbay

global BL
%keyboard
idealbay = zeros(BL.tiers,BL.stacks);

no_cont = length(vect_cont);
no_slots = BL.tiers * BL.stacks;
no_wholes = no_slots - no_cont;
no_class = BL.tiers + BL.stacks - 1;

% Generate the orders matrix
ordermatrix(1,1) = 1; ordermatrix(BL.tiers,BL.stacks) = no_slots;
cont=1;
for diagonal=2:no_class-1
   tier_fin=diagonal;
   tier_ini=1;
   if diagonal>BL.tiers
      tier_fin=BL.tiers;
   end
   if diagonal>no_class-3
      %tier_fin=no_class-3;
      tier_ini=BL.tiers-(no_class-diagonal);
   end

   for tier=tier_ini:tier_fin
      row=diagonal-tier+1;
      if row>BL.stacks
         row=BL.stacks;
      end
      cont=cont+1;
      ordermatrix(tier,row)=cont;
   end
end

% Invert the ordermatrix
invordermatrix=zeros(BL.tiers,BL.stacks);
for row=1:BL.stacks
   for tier=1:BL.tiers
      invordermatrix(tier,row)=no_slots+1-ordermatrix(tier,row);
   end
end

% Modify inverordermatrix to flip upside down
auxmatrix=zeros(BL.tiers,BL.stacks);
for i = 1:BL.tiers
   auxmatrix(i,:) = invordermatrix(BL.tiers+1-i,:);
end

invordermatrix = auxmatrix;
%Now identify the empty slots
```

```
if no_wholes>0
   cont=0;
   for slot=no_slots:-1:no_slots-no_wholes+1
      tier=BL.tiers-fix(slot/(BL.stacks+0.1));
      row=BL.stacks+1-rem(slot,BL.stacks);
      if row==BL.stacks+1
         row=1;
      end
      cont=cont+1;
      wholes_vector(cont)= invordermatrix(tier,row);%tier+(row-1)*BL.tiers; %corregir
   end
   wholes_vector=sort(wholes_vector);
end
idealbay=zeros(BL.tiers,BL.stacks);

% Fill the matrix with the vector of containers

cont=0;
for slot=1:no_slots
   %find the postition of the matrix
   %for tier=BL.tiers:-1:1
   for tier=1:BL.tiers
      for row=BL.stacks:-1:1
         if slot==invordermatrix(tier,row)
            %check wheter it is an empty slot
            emptyslot='n';
            for empty=1:no_wholes
               if slot==wholes_vector(empty)
                  idealbay(tier,row)=100;
                  emptyslot='y';
               end
            end
            if emptyslot=='n'
               cont=cont+1;
               idealbay(tier,row)=vect_cont(cont);
            end
         end
      end
   end
end
```

```
function [idealbay]=midealbay_dc(no_tiers,no_rows,vect_cont)


no_cont=length(vect_cont);
no_wholes=no_rows*no_tiers-no_cont;
no_slots=no_tiers*no_rows;

if no_wholes>0

   for slot=1:no_wholes
      tier=no_tiers-fix(slot/(no_rows+0.1));
      row=no_rows+1-rem(slot,no_rows);
      if row==no_rows+1
         row=1;
      end
      wholes_vector(slot)=tier+(row-1)*no_tiers; %corregir
   end
   wholes_vector=sort(wholes_vector);
end

idealbay=zeros(no_tiers,no_rows);

for tier=1:no_tiers
   for row=1:no_rows
       idealbay(tier,row)=no_rows+1-row+(no_tiers-tier);
    end
end

whole=1; slot=1;
while slot<=no_slots
   tier=no_tiers+1-rem(slot,no_tiers);
   if tier==no_tiers+1
      tier=1;
   end
   row=no_rows-fix(slot/(no_tiers+.1));

   if no_wholes>0
      if slot==wholes_vector(whole)
         idealbay(tier,row)=100;
         whole=whole+1;
      end
   end
   slot=slot+1;
end
```

```
function [idealbay]=midealbay_h(no_tiers,no_rows,vect_cont)

% Detect the slots with no containers in the ideal bay configuration
no_cont=length(vect_cont);
no_wholes=no_rows*no_tiers-no_cont;

for slot=1:no_wholes
   tier=no_tiers-fix(slot/(no_rows+0.1));
   row=no_rows+1-rem(slot,no_rows);
   if row==no_rows+1
      row=1;
   end
   wholes_vector(slot)=tier+(row-1)*no_tiers; %corregir
end
wholes_vector=sort(wholes_vector);

no_slots=no_tiers*no_rows;
idealbay=zeros(no_tiers,no_rows);
cont=1; whole=1; slot=1;
while slot<=no_slots
   tier=no_tiers+1-rem(slot,no_tiers);
   if tier==no_tiers+1
      tier=1;
   end
   row=no_rows-fix(slot/(no_tiers+.1));

   if slot==wholes_vector(whole)
      idealbay(tier,row)=100;
      whole=whole+1;
   else
      idealbay(tier,row)=vect_cont(cont);
      cont=cont+1;
   end
   slot=slot+1;
end
```

```
function [idealbay]=midealbay_v(no_tiers,no_rows,vect_cont)

% Detect the slots with no containers in the ideal bay configuration
no_cont=length(vect_cont);
no_wholes=no_rows*no_tiers-no_cont;
no_slots=no_tiers*no_rows;

for slot=1:no_wholes
   wholes_vector(slot)=no_slots+1-slot; %corregir
end
wholes_vector=sort(wholes_vector);

idealbay=zeros(no_tiers,no_rows);
cont=1; whole=1; slot=1;
while slot<=no_slots
   tier=no_tiers-fix(slot/(no_rows+.1));

   row=no_rows+1-rem(slot,no_rows);
   if row==no_rows+1
      row=1;
   end
   if slot==wholes_vector(whole)
      idealbay(tier,row)=100;
      whole=whole+1;
   else
      idealbay(tier,row)=vect_cont(cont);
      cont=cont+1;
   end
   slot=slot+1;
end
```

```
old_energy

global COST

% initialize indeces
ge = 0; gl = 0; se = 0; sl = 0; he = 0; hl = 0;
% Initialize the distances
gantry_e = 0; gantry_l = 0; spreader_e = 0; spreader_l = 0; hoist_e = 0; hoist_l = 0;

% Consumptions per meter

% Gantry
gec = (0.15*(240000+0)*9.81)/0.7;
glc = (0.15*(240000+ct_weight)*9.81)/0.7;
% Spreader empty/loaded
sec = (0.25*(13000+0)*9.81)/0.75;
slc = (0.25*(13000+ct_weight)*9.8)/0.75;
% Hoist loaded/empty
hlc = ((13000+ct_weight)*9.81)/0.6;
hec = ((13000+0)*9.81)/0.6;


if strcmp(operation,'deliver')  == 1
    ge = ge + 1; gantry_e(ge) = abs(YC(yc).P.bay - BAY.position(1));
    se = se + 1; spreader_e(se) = abs(YC(yc).P.stack - stack)*S.w;
    %he = he + 1; hoist_e(he) = (BL.tiers + 1 - tier) * S.h ;
    hl = hl + 1; hoist_l(hl) = (BL.tiers - tier) * S.h ; %UP
    sl = sl + 1; spreader_l(sl) = abs(0 - stack)*S.w; % to truck
    % Hoist loaded down
    he = he + 1; hoist_e(he) = (BL.tiers) * S.h ; % UP again
elseif strcmp(operation , 'stack') == 1
    ge = ge + 1; gantry_e(ge) = abs(YC(yc).P.bay - BAY.position(1));
    se = se + 1; spreader_e(se) = abs(YC(yc).P.stack)*S.w;
    % he = he + 1; hoist_e(he) = (BL.tiers) * S.h; %DOWN
    hl = hl + 1; hoist_l(hl) = (BL.tiers) * S.h;
    sl = sl + 1; spreader_l(sl) = abs(YC(yc).P.stack - stack)*S.w;
    %hl = hl + 1; hoist_l(hl) = (BL.tiers + 1) * S.h; % DOWN
    he = he + 1; hoist_e(he) = (BL.tiers - tier) * S.h; % UP again
elseif strcmp(operation ,'trans') == 1
    ge = ge + 1; gantry_e(ge) = abs(YC(yc).P.bay - BAY.position(1));
    se = se + 1; spreader_e(se) = abs(YC(yc).P.stack - stack)*S.w;
    %he = he + 1; hoist_e(he) = (BL.tiers - tier + 1) * S.h;
    hl = hl + 1; hoist_l(hl) = (BL.tiers - tier) * S.h;
    gl = gl + 1; gantry_l(gl) = abs(RBAY.position(1) - BAY.position(1));
    sl = sl + 1; spreader_l(sl) = abs(stack - r_stack) * S.w;
    % hoist down
    he = he + 1; hoist_e(he) = (BL.tiers - r_tier) * S.h;
end

GAN = sum(gantry_e)*gec + sum(gantry_l)*glc;
HOI = sum(hoist_e)*hec + sum(hoist_l)*hlc;
SPR = sum(spreader_e)*sec + sum(spreader_l)*slc;
if strcmp(CT(ct).type,'EXP') == 1
```

```
    if strcmp(operation , 'deliver') == 1
        COST.YC.EXP.deliver.gantry =COST.YC.EXP.deliver.gantry + GAN;
        COST.YC.EXP.deliver.hoist = COST.YC.EXP.deliver.hoist + HOI;
        COST.YC.EXP.deliver.spreader = COST.YC.EXP.deliver.spreader + SPR;
    elseif strcmp(operation , 'stack') == 1
        COST.YC.EXP.stack.gantry = COST.YC.EXP.stack.gantry + GAN;
        COST.YC.EXP.stack.hoist = COST.YC.EXP.stack.hoist + HOI;
        COST.YC.EXP.stack.spreader = COST.YC.EXP.stack.spreader + SPR;
    else
        keyboard
    end
elseif strcmp(CT(ct).type,'IMP') == 1
    if strcmp(operation , 'deliver') == 1
        COST.YC.IMP.deliver.gantry = COST.YC.IMP.deliver.gantry + GAN;
        COST.YC.IMP.deliver.hoist = COST.YC.IMP.deliver.hoist + HOI;
        COST.YC.IMP.deliver.spreader = COST.YC.IMP.deliver.spreader + SPR;
    elseif strcmp(operation , 'stack') == 1
        COST.YC.IMP.stack.gantry = COST.YC.IMP.stack.gantry + GAN;
        COST.YC.IMP.stack.hoist = COST.YC.IMP.stack.hoist + HOI;
        COST.YC.IMP.stack.spreader = COST.YC.IMP.stack.spreader + SPR;
    else
        keyboard
    end
end
keyboard
```

```
function [ET]=ordenar(ET)

n=length(ET);
for i=1:n
    i
   for j=1:n-1
      if ET(j).arrival_time>ET(j+1).arrival_time
         aux=ET(j);
         ET(j)=ET(j+1);
         ET(j+1)=aux;
      end
   end
end
```

```
Tmean=2; % in days
Tmean=Tmean*24*3600; % in secs

Tmax=10; %in days

Tmax=Tmean*24*3600; % in secs

delt=Tmax-Tmean;

Po=(2-delt)/Tmax;
```

```
function [ndf,cdf,top_xf] = pdfcdf(no_wc)
% This function calculates both the PDF and CDF of a
calc = 1;
if calc == 1
   % 1 Container weight type to be analyzed
   cont_category=40;
   if cont_category==40
      owl=load('tw40.dat');
      minw=4400;
      maxw=32500;
   elseif cont_category==20
      owl=load('tw20.dat');
      minw=2200;
      maxw=30480;
   end

   % 2 length of the list
   ol_w=length(owl);

   % 3 Remove the empty containers, if there are any
   l_w=0;
   for cont=1:ol_w
      if owl(cont)>minw;
         l_w=l_w+1;
         wl(l_w)=owl(cont);
      end
   end

   % 4 Probability density function NDF
   %no_wc=9;
   [xf,top_xf,cdf]=cdf_x2(no_wc,wl,minw,maxw);
   ndelx=(maxw-minw)/no_wc;
   low_xf=top_xf-ndelx;
   ndf(1)=cdf(1);
   for j=2:no_wc
      ndf(j)= cdf(j)-cdf(j-1);
   end
end

save('pdf.mat','ndf')
save('cdf.mat','cdf')
```

```
function [tier] = Pile_height(bay,stack)

global BAYS BL

tier = 0;
for t = 1: BL.tiers
    if BAYS(bay).ct_id(t,stack)>0
        tier = tier +1;
    end
end
```

```
function plot_bay(bay)

global BAYS

x = BAYS(bay).position(1);
y = BAYS(bay).position(2);
plot(x,y,'bO')
```

```
function plot_bays

global BAYS  BL T TRF

% Check ct
[T.state.IMP,T.state.EXP,T.state.NAS] = terminal_state();
%no_exp_ct,no_imp_ct,exp_es,imp_es,nas_es,no_exp_bays,no_imp_bays,puertos_exp,puertos_imp

% Figure with number of available bays
close_figure_ifexists(30); figure(30)
mnb=100;
subplot(2,2,1); hold on
plot(T.state.EXP.ports.cts,'.'); title('Terminal Ocupation per Port. EXP CTs'); axis([0 TRF.PARAM.no_ports 0 2000])
subplot(2,2,2); hold on
plot(T.state.IMP.ports.cts,'.'); title('Terminal Ocupation per Port. IMP CTs'); axis([0 TRF.PARAM.no_ports 0 2000])
subplot(2,2,3); hold on
plot(T.state.EXP.ports.bays,'.'); title('Terminal Ocupation per Port. EXP Bays'); axis([0 TRF.PARAM.no_ports 0 mnb])
subplot(2,2,4); hold on
plot(T.state.IMP.ports.bays,'.'); title('Terminal Ocupation per Port. IMP Bays'); axis([0 TRF.PARAM.no_ports 0 mnb])

nas_vector = 0; exp_vector = 0; imp_vector = 0;

% Figure of Emtpy slots per bay
close_figure_ifexists(40); figure(40); hold on

nas_b = 0; imp_b = 0; exp_b = 0;
nas_vector = 0; imp_vector = 0; exp_vector = 0;
vector = zeros(T.bays,3);

for bay = 1:T.bays
   vector(bay,1) = BAYS(bay).position(1);
   vector(bay,2) = BAYS(bay).position(2);
   vector(bay,3) = 0;

   switch BAYS(bay).id
      case 'NAS'
         nas_b = nas_b + 1;
         nas_vector(nas_b,1) = BAYS(bay).position(1);
         nas_vector(nas_b,2) = BAYS(bay).position(2);
         nas_vector(nas_b,3) = T.state.NAS.bays.esl(bay);
      case 'EXP'
         exp_b = exp_b + 1;
         exp_vector(exp_b,1) = BAYS(bay).position(1);
         exp_vector(exp_b,2) = BAYS(bay).position(2);
         exp_vector(exp_b,3) = T.state.EXP.bays.cts(bay);
      case 'IMP'
         imp_b = imp_b + 1;
         imp_vector(imp_b,1) = BAYS(bay).position(1);
         imp_vector(imp_b,2) = BAYS(bay).position(2);
         imp_vector(imp_b,3) = T.state.IMP.bays.cts(bay);
   end
end
```

```
AZ =  -37.5; EL = 80;
subplot(3,1,1);
if sum(nas_vector(:,1)) > 0
   stem3(vector(:,1),vector(:,2),vector(:,3),'k.'); view(AZ,EL); hold on
   stem3(nas_vector(:,1),nas_vector(:,2),nas_vector(:,3),'.');
   %stem3(nas_vector(:,1),nas_vector(:,2),nas_vector(:,3)*0,'m.');
end
title('Not assigned bays');
axis([0 T.length 0 T.width 0 BL.capacity])

subplot(3,1,2);
if sum(exp_vector(:,1)) > 0
   stem3(vector(:,1),vector(:,2),vector(:,3),'k.'); view(AZ,EL); hold on
      stem3(exp_vector(:,1),exp_vector(:,2),exp_vector(:,3),'.');
   %stem3(exp_vector(:,1),exp_vector(:,2),exp_vector(:,3)*0,'m.');
end
title([num2str(sum(T.state.EXP.no.bays)) ' / ' num2str(sum(T.state.EXP.no.cts)) ' EXP bays/CT. CT distribution']);
axis([0 T.length 0 T.width 0 BL.capacity])

subplot(3,1,3);
if sum(imp_vector(:,1)) > 0
   stem3(vector(:,1),vector(:,2),vector(:,3),'k.');  view(AZ,EL); hold on
   stem3(imp_vector(:,1),imp_vector(:,2),imp_vector(:,3),'.');
   %stem3(imp_vector(:,1),imp_vector(:,2),imp_vector(:,3)*0,'m.');
end
title([num2str(sum(T.state.IMP.no.bays)) ' / ' num2str(sum(T.state.IMP.no.cts)) ' IMP bays/CT. CT distribution']);
axis([0 T.length 0 T.width 0 BL.capacity])

% Figure Exp/Imp bays
% close_figure_ifexists(50); figure(50); hold on
% %subplot(3,1,1); plot(nas_port,'.'); title('Not assigned bays. Port ID');
% subplot(2,1,1); plot(exp_es,'.'); title('Export bays. Port ID');
% subplot(2,1,2); plot(imp_es,'.'); title('Import bays. Port ID');

pause(1)
```

```
function plot_bays_candidates(CB_ids,col)
% This function plots the candidate bays with the desired color COL
global BAYS BL T

%keyboard

plot_bays()
if length(CB_ids) > 0

   %figure(40);
   k = 0;   figure(40)

   for i = 1:T.bays
      plot3(BAYS(i).position(1),BAYS(i).position(2),0,'k.'); hold on
   end

   for i = 1:length(CB_ids)
      k = k +1;
      bay_no = CB_ids(i);
      x=BAYS(bay_no).position(1); y = BAYS(bay_no).position(2);
      if BAYS(bay_no).id == 'NAS'
         subplot(3,1,1); hold on
         colo = '.r';
      elseif BAYS(bay_no).id == 'EXP'
         subplot(3,1,2); hold on
         colo = '.r';
      elseif BAYS(bay_no).id == 'IMP'
         subplot(3,1,3); hold on
         colo = '.r';
      else
         disp('VS Plan Error: a bay with no identifier!')
         keyboard
      end
      %plot(BAYS(bay_no).no, BAYS(bay_no).empty_slots,col);
      plot3(x,y,BL.capacity-BAYS(bay_no).empty_slots,col);
      %plot3(x,y,0,'k*');
      if k == 1
        hold on
      end
   end
   grid on
end
```

```
function [space_reserved] = plot_bays_reserved

global TRF

[pure_ports,stack_ports,mixed_ports] = check_reserve();

figure(100)
subplot(3,1,1); plot(pure_ports,'.'); axis([0 TRF.PARAM.no_ports 0 2000])
subplot(3,1,2); plot(stack_ports,'.'); axis([0 TRF.PARAM.no_ports 0 2000])
subplot(3,1,3); plot(mixed_ports,'.'); axis([0 TRF.PARAM.no_ports 0 2000])
```

```
function plot_evolution()

global BL R T TIME TRF

b = length(R.tiempo); %R.tiempo has the time in days

figure(21);
subplot(3,1,1); hold on
%axis([0 b 0 t])

plot(R.tiempo/3600/24,R.ics,'r.'); ylabel('IMP')
plot(R.tiempo/3600/24,R.ocs,'g.'); ylabel('EXP')
plot(R.tiempo/3600/24,R.tcs,'b.'); ylabel('TOTAL')
% j = +R.tcs;
plot(R.tiempo/3600/24,R.res,'m.'); ylabel('OCC & RES')
axis([0 TIME.simul/3600/24 0 T.bays*BL.capacity])

x=[R.tiempo(b)/3600/24; R.tiempo(b)/3600/24];
y=[0; 2000];

subplot(3,1,2); hold on; plot(x,y,'r')
subplot(3,1,3); hold on; plot(x,y,'r')

%plot_vs_traffic()
```

```
function plot_ics(bays)

figure(30)
for i=1:length(ics) %ict_list
    if bays(i).id=='EXP'
        spaces(i)=bays(i).empty_slots;
    else
        spaces(i)=BL.tiers*BL.rows+1;
    end
    ids(i)=bays(i).port;
end
```

```
function plot_terminal (block)

global BAYS BL S T

figure(120)

b=0;
for bay = 1:T.bays
   if BAYS(bay).block == block
   % if strcmp(BAYS(bay).id,'EXP')==1
      %bay
      x = BAYS(bay).position(1);
      for stack = BL.stacks:-1:1
         y = BAYS(bay).position(2) - S.w*stack;
         for tier = 1:BL.tiers
            z = (BL.tiers-tier+1) * S.h/2;
            %ct_id = BAYS(bay).matriz(tier,stack)*BAYS(bay).port(tier,stack);
            ct_id = BAYS(bay).port(tier,stack);
            if BAYS(bay).matriz(tier,stack) == 1
               switch BAYS(bay).id
                  case 'EXP'
                     col='r.';
                  case 'IMP'
                     col='r*';
                  case 'NAS'
                     col='r-';
               end
            else
               switch BAYS(bay).id
                  case 'EXP'
                     col='b.';
                  case 'IMP'
                     col='b*';
                  case 'NAS'
                     col='b-';
               end
            end
            plot3(x,y,z,col); hold on
            text(x,y,z,num2str(ct_id))
         end
      end
   end
end

axis equal
```

```
function plot_trf(TRF,plot_option)
% This function displays the traffic generation

disp('Plot traffic')

if plot_option == 1
   close_figure_ifexists(60); figure(60); hold on
   subplot(2,1,1); hold on; title('EXP ET'); axis([0 24 0 TRF.PARAM.no_ports])
   subplot(2,1,2); hold on; title('IMP ET'); axis([0 24 0 TRF.PARAM.no_ports])

   for et=1:length(TRF.ET)
      if strcmp(TRF.ET(et).id,'EXP') == 1
         subplot(2,1,1);
      elseif strcmp(TRF.ET(et).id,'IMP') == 1
         subplot(2,1,2);
      end
      plot(TRF.ET(et).arrival_time/24/3600,TRF.ET(et).target_VS,'b.')
   end

   for vs=1:length(TRF.VS)
      subplot(2,1,1);hold on;
      plot(TRF.VS(vs).arrival_time/24/3600,TRF.VS(vs).port,'r.')
      subplot(2,1,2); hold on;
      plot(TRF.VS(vs).arrival_time/24/3600,TRF.VS(vs).port,'r.')
   end
elseif plot_option == 2

   % Display the arrival of CT to the Terminal
   exp_ct_no = zeros(1,TRF.PARAM.no_ports);
   imp_ct_no = zeros(1,TRF.PARAM.no_ports);
   close_figure_ifexists(90); figure(90)

   for ct=1:length(TRF.ET)
      port=TRF.ET(ct).port;
      if strcmp(TRF.ET(ct).id,'IMP') == 1
         imp_ct_no(port) = imp_ct_no(port) +1;
         subplot(2,1,2); hold on; plot(TRF.ET(ct).arrival_time,imp_ct_no(port))
      elseif strcmp(TRF.ET(ct).id,'EXP') == 1
         exp_ct_no(port) = exp_ct_no(port) +1;
         subplot(2,1,1); hold on; plot(TRF.ET(ct).arrival_time,exp_ct_no(port))
      end
   end
end

%keyboard
```

```
function plot_TRF1()

global ET TIME TRF VS

figure(20); hold on
exp_ct = zeros(1,TRF.PARAM.no_ports);
imp_ct = zeros(1,TRF.PARAM.no_ports);

for ct = 1:length(ET)
   port = ET(ct).port;
   flow = ET(ct).id;
   switch flow
      case 'IMP'
         imp_ct(port) = imp_ct(port) + 1;
         imp_ct_no(imp_ct(port),port)=imp_ct(port);
         imp_time(imp_ct(port),port) = ET(ct).arrival_time;
      case 'EXP'
         exp_ct(port) = exp_ct(port) + 1;
         exp_ct_no(exp_ct(port),port) = exp_ct(port);
         exp_time(exp_ct(port),port) = ET(ct).arrival_time;
   end
end
%keyboard
for p=1:TRF.PARAM.no_ports
   subplot(3,1,2); hold on
   plot(imp_time(:,p)/3600/24,imp_ct_no(:,p),'r.')
   subplot(3,1,3); hold on
   plot(exp_time(:,p)/3600/24,exp_ct_no(:,p),'g.')
end

time_span = TIME.simul/3600/24;

subplot(3,1,2); title('IMP ET Traffic'); axis([0 time_span 0 2000])
subplot(3,1,3); title('EXP ET Traffic'); xlabel('Time');axis([0 time_span 0 2000])
%keyboard
for vs = 1: length(VS)
   vs_arrival_time = VS(vs).arrival.time/24/3600;
   vs_plan_time = VS(vs).plan.time/24/3600;
   subplot(3,1,2); plot(vs_arrival_time,vs*100,'bo')
   subplot(3,1,3); plot(vs_arrival_time,vs*100,'bo')
   subplot(3,1,2); plot(vs_plan_time,vs*100,'g*')
   subplot(3,1,3); plot(vs_plan_time,vs*100,'g*')
end
```

```
function plot_VS_arrival(vs)

global VS ET

for i = 1:length(VS(vs).OC_arrived)
    et = VS(vs).OC_arrived(i);
    etarrivals(i) = ET(et).arrival_time;
end
y = 1:length(VS(vs).OC_arrived);
figure; plot(etarrivals/3600/24,y,'*'); hold on
xt = [VS(vs).plan.time/3600/24 VS(vs).plan.time/3600/24];
yt= [0 length(VS(vs).OC_arrived)];
plot(xt,yt,'r');
xe = [VS(vs).arrival.time/3600/24 VS(vs).arrival.time/3600/24];
plot(xe,yt)
    xlabel('Days'); ylabel('No. containers')
```

```
function plot_VS_plan(vs)

global BAYS BL CT T VS

plan = VS(vs).plan.EXP;

ncb = length(plan.bays); vsocup=zeros(1,ncb);

for i_bay = 1:ncb
   bay = plan.bays(i_bay);
   R(i_bay) = BAYS(bay).R.slots;
   O(i_bay) = BL.capacity - BAYS(bay).empty_slots;

   % a) Now check if slots reserved have been occupied
   pos = find(BAYS(bay).ct_id>0); ctlist = BAYS(bay).ct_id(pos);

   vslist = 0;
   for i = 1: length(ctlist)
      vslist(i) = CT(ctlist(i)).vs;
   end
   vsocup(i_bay) = length(find(vslist == vs));
end
figure;
subplot(2,1,1);
stem(plan.bays,R,'bo'); hold on;
plot(plan.bays,O,'go');
axis([0 T.bays 0 BL.capacity]); title('Occupation vs. total reservation')

subplot(2,1,2);
stem(plan.bays,plan.cts,'b.'); hold on;
plot(plan.bays,vsocup,'ro');
axis([0 T.bays 0 BL.capacity]); title('Vessel Occupation vs. Vessel reservation')
```

```
function plot_vs_traffic()

global R TRF

figure(20);

subplot(3,1,2);
for p = 1:TRF.PARAM.no_ports
   plot(R.vsct.imp(p,:),1:length(R.vsct.imp))
end

subplot(3,1,3);
for p = 1:TRF.PARAM.no_ports
   plot(R.vsct.exp(p,:),1:length(R.vsct.exp))
end
```

```
function plot_yc()

global BAYS MAC YC T

close_figure_ifexists(344);figure(344); hold on
for bay = 1:T.bays
   xb(bay) = BAYS(bay).position(1);
   yb(bay) =  BAYS(bay).position(2);
end

plot(xb,yb,'r.')

for yc = 1:MAC.YC.ycsprow*T.rows
   bay = YC(yc).P.bay;
   x = BAYS(bay).position(1);
   y =  BAYS(bay).position(2);
   if rem(yc,2) == 0
      text(x,y+7,num2str(yc));
   else
      text(x,y-7,num2str(yc));
   end

   plot(x,y,'ro')
end

Y =T.width+20;
X = T.length+20;
axis([0 X 0 Y])
```

```
function plot_yc_wl()

global BAYS T MAC YC

close_figure_ifexists(343);figure(343)
for row = 1:T.rows
   for iyc=1:MAC.YC.ycsprow
      yc = (row-1)*MAC.YC.ycsprow + iyc;
      bay=YC(yc).P.bay;
      x = BAYS(bay).position(1);
      y = BAYS(bay).position(2);
      zp= YC(yc).WL.cwl;

      stem3(x,y,0,'kx');hold on
      text(x,y,-0.5,num2str(yc),'FontWeight','bold');
      if zp >0
         m = [ num2str(zp)];
         text(x,y,-1,m,'Color','r');
      end
      stem3(x,y,zp,'.-r');
   end
end
```

```
function plot_ycs()


global BAYS MAC YC T

for row = 1:T.rows
   for i=1:MAC.YC.ycsprow
      yc = (row-1)*MAC.YC.ycsprow + i;
      %Y(yc) = row;
      bay = YC(yc).P.bay;
      X(yc) = BAYS(bay).position(1);
      Y(yc) = BAYS(bay).position(2);
   end
end

figure
plot(X,Y,'*')
```

```
function plotvector(vector)

figure;

n=length(vector);
x=1:n;
plot(x,vector,'.');
```

```
V=0;
for i= 1:10000
    V(i)=poissrnd(7);
end
[a,x] = hist(V,50);
figure; plot(x,a)

std(V)
```

```
function [col_ids,no_cols] = Port_empty_cols(BAY)
% This function calculates the empty colums

[rows,cols]= size(BAY.matriz);

no_cols = 0; col_ids =0;

for col = 1: cols
    if sum(BAY.port(:,col)) == 0
        no_cols= no_cols + 1;
        col_ids(no_cols) = col;
    end
end

if no_cols == 0
   disp('BAY_find_empty_col Error: no empty staks')
   BAY
end
```

```
function [positions,no_ceros] = Port_empty_slots(BAY)
% This function gives the number of empty ports

global BL

positions = 0;
no_ceros = 0;
for tier = 1:BL.tiers
   for stack = 1:BL.stacks
      if BAY.port(tier,stack) == 0
         no_ceros = no_ceros +1;
         positions(no_ceros,1) = tier;
         positions(no_ceros,2) = stack;
      end
   end
end
```

```
function [nports] = ports_howmanyofeach(portslist)

global TRF
nports = 0;

ports = unique(portslist);
if sum(ports) > 0
    nports = zeros(1,TRF.PARAM.VS_no);
    for i =1: length(ports)
        ppos = find(portslist == ports(i));
        nports(ports(i)) = length(ppos);
    end
end
```

```
function [ct_day]=pyramid(timeperiod,no_tot_ct)
% this function generates the container list for the number of days
% specified, and for a given vessel.
global TRF
option = 1; % Exponential function

if option ==1
   if timeperiod == 3 %TRF.PARAM.daysinadvance
      B = -0.419401381636162;
      A = no_tot_ct/1.560652615;
   elseif timeperiod == TRF.PARAM.daysofdischarge
      B = -0.68368083;
      A = no_tot_ct/1.017037602;
   end
   for day=1:timeperiod
      ct_day(day) = fix(A*exp(B*day));
   end
   dif = no_tot_ct - sum(ct_day);
   if dif ~= 0
      ct_day(1) = ct_day(1) + dif;
   end
   if timeperiod == TRF.PARAM.daysinadvance
      ct_day = ct_day(end:-1:1);
   end
else
   no_days=fix(timeperiod/2);
   extraday=timeperiod-2*no_days;
   no_deltas=0;
   for day=2:no_days
      no_deltas=no_deltas+2*(day-1);
   end
   no_deltas=no_deltas+day*extraday;
   P=1/timeperiod*.8;
   delta=(1-timeperiod*P)/no_deltas;
   no_ct=0;
   for day=1:no_days
      p(day)=P+(day-1)*delta;
      p(timeperiod-day+1)=p(day);
      ct_day(day)=fix(p(day)*no_tot_ct);
      ct_day(timeperiod-day+1)=fix(p(day)*no_tot_ct);
      no_ct=no_ct+ct_day(day)+ct_day(timeperiod-day+1);
   end
   if extraday==1
      %p(day+1)=p(day)+delta;
      ct_day(day+1)=no_tot_ct-no_ct;
   end
end
```

```
function [QC]=QC_cycle(QC,VS,YT)
% This function describes the cycle of a yard crane

% 1. Speeds based on a Doosan QC brochure
t_mean=(25/60)*60*60;
gantry_speed=45/60; % 45m/min
hoist_speed_fulload=70/60; %70m/min
hoist_speed_empty=170/60; %70m/min
trolley_speed=240/60;


% 2. Now analyze the cycle

% 2.1 Change from wait to move to the vessel bay target
switch QC.status
   case 'w4un'
      if VS.status=='w4qc'
         QC.status='move';


      end

% 2.2. Go from wait for load to load operation
   case 'w4lo' & YT.status=='w4qc'
      QC.status='load';
      QC.t4completion=poissrnd(t_mean); % calculate the time of operation

% 2.3. Go from load for load to wait again for load another container
   case QC.status=='load';
      QC.status='w4un';
end
```

```
function [QC]=QC_move(QC)

% Criteria 1: Find the vessel bay that is closer to the curren position of the QC
% Criteria 2: Find the leftmost vessel bay. Not used for the moment
limit=100000;
for bay=1:QC.vs_bays_no
   if sum(QC.CperVS_bay(:))==0 % then the bay is empty
      distance(bay)=limit;
   else
      distance(bay)=distance_calculator(QC.position,QC.bay_position(bay,:));
   end
end


[min_distance,target_bay]=min(distance);

if min_distance==limit
   'error: the QC is trying to move to a bay but they are all empty. start loading'
   keyboard
else
   QC.target_position(1)=QC.bay_position(target_bay,1);
end
```

```
function remarshalling(slots2liberate,VS,label)
% This function makes remarshalling on a certain number of bays with little
% occupation in order to minimize the rehandling effort

global BAYS BL COST T

slots_liberated = 0;

% See what EXP bays in blocks have less occupation
% ------------------------------------------------------------------------

[esl,b_esl] = block_analyze()

if sum(b_esl) < slots2liberate
   disp('Remarshanlling error: not enough slots to place CTs')
   figure; plot(b_esl,'.')
else
   % Start remarshalling
   disp('Start Remarshalling')
   while  slots_liberated < slots2liberate
      [T.state.IMP,T.state.EXP,T.state.NAS] = terminal_state();
      % Find target block, the one having more slots
      [tb_available_slots, target_block] = max(b_esl);
      disp(['Housekeeping containers of block: ' num2str(target_block)])
      % Inside each bay, search those with less
      i = 0; clear tb_cts_and_ids
      ini_bay = BL.baylist(target_block,1);
      fin_bay = BL.baylist(target_block,BL.bays);
      for bay = ini_bay:fin_bay
         if strcmp(BAYS(bay).id,label) == 1
            [positions,no_ceros] = Port_empty_slots(BAYS(bay));
            if no_ceros >= BL.tiers
               tiers2lib(bay) = fix(no_ceros/BL.tiers);
            end
         end
      end

      % Time to move CTs. There are two possibilities
      % First, all the block CTs need to be moved
      if sum(tiers2lib(ini_bay:fin_bay)*BL.tiers <= slots2liberate)
         baylist = ini_bay:1:fin_bay;
      % The second chance is the block has more slots free than needed
      else
         % need to determine the minimum movement for the YC
         baylist = BL_analysis(target_block);
      end
      % Now go through the list and reorganize the CTs
      for i_bay = 1:length(baylist)
         bay = baylist(i_bay);
         BAY_remarshall(bay);
         slots_liberated = slots_liberated + tiers2lib(bay) * BL.tiers;
      end
   end
```

end

```
function Results_init()

global R TRF

R.progreso = 10;
R.res = 0;
R.vsct.imp = zeros(TRF.PARAM.no_ports,1);
R.vsct.exp = zeros(TRF.PARAM.no_ports,1);
R.ev = 0;
R.tiempo = 0;
R.ocs = 0; R.ics = 0; R.tcs = 0;
```

```
function savestate

global BAYS BL BT COST COUNT CT ET GROUPS MAC R S SPEED T TIME TRF VS YC

%if exist('pstate.mat')
%   movefile('pstate.mat','apstate.mat');
%end
%save('pstate.mat','BAYS','BL','BT','COST','COUNT','CT','ET','GROUPS','MAC','R','S','SPEED','T','TIME','TRF','VS','
YC');
```

```
function [i,j]=search_bay(bay,port)
% This function search for the empty slot located more down-right in the
% bay

[a,b]=size(bay.matriz);

if bay.empty_slots ~= 0
    for tier=1:a
        for row=1:b
            if and(bay.matriz(tier,row)==0, bay.port(tier,row) == port)
                i=tier; j=row;
            end
        end
    end
end
```

```
function [ports,es] = Search_Port_Matriz(bay)
% This function counts the number of slots assigned to each port

global BAYS BL TRF
ports = zeros(1,TRF.PARAM.no_ports);
es = 0;
for tier = 1:BL.tiers
   for stack = 1:BL.stacks
      bay_port = BAYS(bay).port(tier,stack)*BAYS(bay).matriz(tier,stack);
      if bay_port >0
         ports(bay_port) = ports(bay_port) + 1;
      end
      empty_port = BAYS(bay).port(tier,stack) + BAYS(bay).matriz(tier,stack);
      if empty_port == 0
         es = es + 1;
      end
   end
end
```

```
function [slot_col,row]= select_slot(comp_bay,idealbay,c_class)

% This function takes a vector of weightclass and places it on the best
% possible row and column

% Initialize

[no_tiers,no_rows]=size(idealbay);
comp_bay=flipud(comp_bay);
diference=zeros(1,no_rows);
toptier=zeros(1,no_rows);
for row=no_rows:-1:1 % buscar la col en la que mejor encaja
   %row;
   tier=no_tiers;

   while comp_bay(tier,row)>0 && tier ~= 1
      if tier>1
         tier=tier-1;
      end
   end
   if tier==1 && comp_bay(tier,row) >0
      diference(row)=100;
   else
      diference(row)=idealbay(tier,row)-c_class;
   end
   toptier(row)=tier;
end
%now choose the position with the smallest difference
% this function will be further improved
[j,slot_row]=min(abs(diference));
j=sort(abs(diference));
i=2;    equals=1;
while i<no_rows
   if j(i)==j(1)
      equals=equals+1;
   end
   i=i+1;
end
if equals>1
   'There is more than one option'
   % Now find the alternative with lower tier'
end

slot_col=toptier(slot_row);
```

```
function [distance]=simple_dist_calculator(A,B)

delx=(A(1)-B(1));
dely=(A(2)-B(2));
distance = (delx^2+dely^2)^0.5;
```

```
function SPEED_init()

global SPEED

% 1. YT
SPEED.YT.travel = 40 / 3.6;         % From km/h to m/s

% 2. YC
SPEED.YC.gantry = 130/60; %1.5;         % (m/s)
SPEED.YC.hoist.empty = 56/60;
SPEED.YC.hoist.loaded = 30/60;
SPEED.YC.spreader.empty = 70/60;
SPEED.YC.spreader.loaded = 70/60;

% 3. ET
SPEED.ET.travel = 20/3.6;       % (m/s)
```

```
function [work,destination] = ST_work(bay,stacks_needed)
% THis function calculates the work done by a YC to move CTs within a bay

global BAYS BL CT S YC

keyboard
% 1. Initialize
used_stacks = BL.stacks;
work = zeros(1,BL.stacks) ;
stack_cts = sum(BAYS(bay).matriz(:,:));

i_stack = 0;

while i_stack < stacks_needed
   i_stack = i_stack + 1;

   for ini_stack = 1 : BL.stacks

      % Compute the work to remove the ini_stack CTs to fin_Stack
      cstack_cts = stack_cts;
      if stack_cts(ini_stack) == 0
         work(ini_stack) = 10000000000;
      else
         for h = 1:stack_cts(ini_stack)
            % Identify the best position for the CT
            ct_h = stack_cts(ini_stack) - h +1;
            %ini_tier = BL.tiers - ct_h +1;
            ini_tier = ct_h;
            ct_id = BAYS(bay).ct_id(ini_tier,ini_stack);
            c_wc = 100*ones(1,BL.stacks);

            for tstack = 1:BL.stacks
               %top_tier = BL.tiers - cstack_cts(tstack);
               top_tier = cstack_cts(tstack)+1;
               if and(BAYS(bay).matriz(top_tier,tstack) == 0, tstack ~= ini_stack)
                  c_wc(tstack) = abs(BL.idealbay(top_tier,tstack) - CT(ct_id).class);
               end
            end
            % The best position is the one with less difference
            % Among the possible candidates, choose the one with higher
            [wmin,target_stack] = min(c_wc);
            if target_stack < BL.stacks
               for s = target_stack+1:BL.stacks
                  if wmin == c_wc(s)
                     if and(cstack_cts(s) > cstack_cts(target_stack), cstack_cts(s)<BL.tiers)
                        target_stack = s;
                     end
                  end
               end
            end
            %target_tier = BL.tiers - cstack_cts(target_stack);
            target_tier = cstack_cts(target_stack)+1;
            % Compute the effort
```

```
            port = BAY.port(ini_tier,ini_stack);

            CT_drop(ct_id);
            %bay,ini_tier,ini_stack
            CT_remove(ct_id);
            % Update stacks
            cstack_cts = sum(BAYS(bay).matriz(:,:));
            % go to slot
            work(ini_stack) = work(ini_stack) + YC_consumption('H','L',ct_id)*(BL.tiers+1-ini_tier)*S.h +
YC_consumption('S','L',ct_id)*abs(ini_stack-target_stack)*S.w;
          end
       end
    end
end
```

```
function [ctplan] = T_bay_search(vs)
% This function analyzes where the CTs to be uploaded on a vs are located

global BAYS BL BT MAC T VS

%keyboard
nb=0; nc = 0;
CB = 0; YCS = 0; maxwl = 0;

for yc = 1:MAC.YC.n
   YCWL(yc).cts = 0;
   YCWL(yc).D = 0;
end
for bay = 1:T.bays
   if strcmp(BAYS(bay).id,'EXP') ==1
      % check if the bay still has CTs for that destination
      %[ports] = BAY_find_reservations(bay,'C');
      [list, ncvs] = BAY_get_port(bay,vs);
      if sum(list) > 0
         nb = nb+1;
         yc = YC_assign_ct(bay); YCS(nb) = yc;
         dist = ones(1,length(list))*simple_dist_calculator(BAYS(bay).position,BT(VS(vs).berth).position);
         if YCWL(yc).cts == 0
            YCWL(yc).cts = list;
            YCWL(yc).D = dist;
         else
            YCWL(yc).cts = [YCWL(yc).cts, list];
            YCWL(yc).D = [YCWL(yc).D, dist];
         end
         maxwl = max(maxwl,length(YCWL(yc).cts));
      end
   end
end


if nb > 0
   %keyboard
   YCS = unique(YCS);
   % Sort the workload of YCS
   for i = 1:length(YCS)
      yc = YCS(i);
      [dist,ord] = sort(YCWL(yc).D);
      YCWL(yc).cts = YCWL(yc).cts(ord);
   end
   for i = 1:maxwl
      for y = 1:length(YCS)
         yc = YCS(y);
         if i <= length(YCWL(yc).cts)
            nc = nc+1;
            ctplan(nc) =  YCWL(yc).cts(i);
         end
      end
   end
```

```
   if abs(length(ctplan)-VS(vs).OC) >0
      disp(['VS(' num2str(vs) ') has ' num2str(VS(vs).OC) ' to upload. ' num2str(length(ctplan)) ' found in yard'])
      difcts = setdiff(VS(vs).OC_arrived, ctplan)
      disp([' The different containers are: ' num2tr(difcts)])
      check_cts_vs();
      keyboard
   end
else
   keyboard
end

%keyboard
```

```
function [grupo] = T_eval()

global BAYS BL COUNT GROUPS T TRF TIME

for port = 1:TRF.PARAM.no_ports

   grupo(port).no = 0;
   grupo(port).bahias = 0;
   grupo(port).cts = 0;
   grupo(port).dist = 0;

   bay=0; lastbay = 1;

   for t_col = 1:T.cols % respect the order of cols and rows
      for t_row = 1:T.rows
         for b_bay = 1:BL.bays
            bay = bay + 1;
            if BAYS(bay).T_row ~= BAYS(lastbay).T_row
               start = 1;
            end
            [no_slots,distance] = check_port_bays(bay,port);
            if no_slots > 0 % Add bay to existing group
               if start == 1 % If the bay is 1, start a the group
                  grupo(port).no = grupo(port).no + 1;
                  grupo(port).bahias(grupo(port).no) = 1;
                  grupo(port).cts(grupo(port).no) = no_slots;
                  grupo(port).dist(grupo(port).no) = distance*no_slots;
                  start = 0;
               else % Add bay to existing group
                  grupo(port).bahias(grupo(port).no) = grupo(port).bahias(grupo(port).no) + 1;
                  grupo(port).cts(grupo(port).no) = grupo(port).cts(grupo(port).no) + no_slots;
                  grupo(port).dist(grupo(port).no) = grupo(port).dist(grupo(port).no) + distance*no_slots;
               end
            else % Start a new group
               start = 1;
            end
            lastbay = bay;
         end
      end
   end
end

COUNT.stat = COUNT.stat +1;
GROUPS(COUNT.stat).CT = grupo;
GROUPS(COUNT.stat).time = TIME.t;
```

```
function T_initialization()

global BL S T TRF

% TERMINAL GEOMETRY
% - Dimensions in meters ----------------------------------------------

% 1 Slots
S.w = 2.438 + 0.40; %17/6; % Width of individual CT slot
S.l = 6.058 + 0.40; %6.058; % 20" size
S.h = 2.59; % Standard 40"

% 2 Blocks
BL.tiers = 4; BL.stacks = 6; BL.bays = 32;
BL.capacity = BL.tiers*BL.stacks; BL.slots = BL.bays * BL.capacity;
BL.width = (BL.stacks + 1.5) * S.w;
BL.length = BL.bays * S.l;

% 2.2. Terminal layout
T.rows = 5; T.cols = 5;  T.blocks = T.rows * T.cols;
T.YC_no = T.blocks * 2;

% Aisles
% horizontal
T.aisles.horizontal.no = T.rows - 1;
T.aisles.top.width = 80;
T.aisles.bottom.width = 35;
T.aisles.horizontal.width = 15;

T.aisles.vertical.no = T.cols - 1;

% Terminal Lengths
T.aisles.sides.width = 60;
T.aisles.vertical.width = 50;

% Total Terminal size
T.length = 2 * T.aisles.sides.width + T.aisles.vertical.width * T.aisles.vertical.no + T.cols * BL.bays * S.l;

T.width = T.aisles.top.width + T.aisles.bottom.width + T.rows * (BL.stacks+1.5) * S.w + T.aisles.horizontal.no *
T.aisles.horizontal.width;
disp(['Terminal dimensions: length ' num2str(T.length) ' m. Width: ' num2str(T.width) ' m'])

% Total Terminal berths
T.berth_no = 3;
for berth = 1:T.berth_no
   T.berth(berth).position(1) = T.length * (berth/(T.berth_no + 1));
   T.berth(berth).position(2) = T.width;
end

% Gates
T.gate.position(1) = T.length/2;
T.gate.position(2) = 0;
```

```
% Terminal Ocupation
T.initial_occupation = 0.0;
T.max_occupation = 0.7;
T.initial_emtpy_bays = 0.2;
T.stack.imp = BL.tiers - 1;
T.stack.exp = BL.tiers;
T.limits.bay.imp = BL.capacity - BL.tiers;
T.limits.bay.exp = BL.capacity - BL.tiers;

% Mixed features
T.slots = BL.slots * T.blocks;
T.bays = T.cols * T.rows * BL.bays;

% ------------------------------------------------------------------------
% 10. DEFINITION OF THE IDEAL BAY CONFIGURATION
% ------------------------------------------------------------------------
% It is based on the existing Container distribution
vect_cont = CT_distribution(TRF.CT.pdf,BL.capacity);
BL.idealbay = midealbay_d(vect_cont);
```

```
function [R_bays,R_slots,VS] = T_reservation(CB_NSD,needed_slots,mix_id,VS,label)
% This function makes the reservation of slots making use of the BL.idealbay

global BAYS BL

target_bay =0; R_bays = 0; R_slots = 0; % Reserved bays and slots
cts_left = needed_slots;
 % This is used to control the # cts to be "assigned"
no_bays = length(CB_NSD(:,1));

% Start marking bays for this vessel
while and( cts_left >0 , R_bays < no_bays )
   R_bays = R_bays + 1; cts_reserved(R_bays) = 0;
   target_bay = CB_NSD(R_bays,1);
   if isempty(target_bay)
      keyboard
   end
   if BAYS(target_bay).empty_slots > BL.tiers
      %BAYS(target_bay).R.B.slotsb = 0;
      % Once identified the bay, fill it to limit
%        if strcmp (mix_id,'S') == 1
%           no_cols = CB_NSD(R_bays,2)/BL.tiers;
%           [cts_reserved(R_bays),R_slots,VS] = BAY_reservation(cts_left,R_slots,target_bay,mix_id,VS,label,no_cols);
%        else
         [cts_reserved(R_bays),R_slots,VS] = BAY_reservation(cts_left,R_slots,target_bay,mix_id,VS,label);
%       end
      cts_left = cts_left - cts_reserved(R_bays);
      %BAY_change_reservation(target_bay,VS.no,cts_reserved(R_bays));
   end
end
%check_port_reservation(target_bay)
if target_bay ==0
   'Bays assign error'
   keyboard
end
label = BAYS(CB_NSD(1,1)).id;

% Add the bays already in the plan to the new bays
switch label
   case 'EXP'
      [VS.plan.EXP] = VS_plan_generation(VS.plan.EXP,CB_NSD(1:R_bays,1),cts_reserved);
   case 'IMP'
      %keyboard
      [VS.plan.IMP] = VS_plan_generation(VS.plan.IMP,CB_NSD(1:R_bays,1),cts_reserved);
end
% 1.3 Plot
if mix_id == 'U'
   col = 'k.';
elseif mix_id == 'N'
%    if BAYS(target_bay).id == 'EXP'
%       col = 'g.';
%    elseif BAYS(target_bay).id == 'NAS'
%       col = 'c.';
```

```
%     end
      col = 'g.';
elseif mix_id == 'S'
   col = 'm.';
elseif mix_id == 'Y'
   col = 'r.';
end

%plot_bays_candidates(CB_NSD(1:R_bays,1),col)

if abs(R_slots-needed_slots) > 0
   disp('Terminal reservation warning: more space needed')
   %keyboard
end
```

```
function [IMP,EXP,NAS] = terminal_state()
% This function calcultes the state of the terminal in a given time
% no_exp_ct: total number of exp containers
% exp_es: emtpy slots per exp bay

%keyboard

global BL BAYS T TRF

% Initialization
% Per total numbers
IMP.no.bays = 0; EXP.no.bays = 0; NAS.no.bays = 0;
IMP.no.cts = 0; EXP.no.cts = 0; NAS.no.cts = 0;
IMP.reservations = 0; IMP.cts = 0; IMP.unusedcts = 0;
EXP.reservations = 0; EXP.cts = 0; EXP.unusedcts = 0;
IMP.empty = 0; EXP.empty = 0;
% Per bay
IMP.bays.cts = zeros(1,T.bays);
EXP.bays.cts = zeros(1,T.bays);

IMP.bays.esl = zeros(1,T.bays);
EXP.bays.esl = zeros(1,T.bays);
NAS.bays.esl = zeros(1,T.bays);

IMP.bays.unused=zeros(1,T.bays);
EXP.bays.unused=zeros(1,T.bays);

% Ports
% Per CTs
IMP.ports.cts = zeros(1,TRF.PARAM.no_ports);
EXP.ports.cts = zeros(1,TRF.PARAM.no_ports);
%NAS.ports.cts = zeros(1,no_ports);
IMP.ports.bays = zeros(1,TRF.PARAM.no_ports);
EXP.ports.bays = zeros(1,TRF.PARAM.no_ports);
%NAS.ports.bays = zeros(1,no_ports);

% Bay loop
for bay = 1:T.bays
   % See the destination port of the bay
   %port = BAYS(bay).port;
   bayocup = sum(sum(BAYS(bay).matriz));

   switch BAYS(bay).id
      % Number of bays, # Empty slots per bay, and Existing containers
      case 'IMP'
         IMP.no.bays = IMP.no.bays + 1;
         IMP.bays.esl(bay) = BAYS(bay).empty_slots;
         IMP.bays.cts(bay) = bayocup ; %BL.capacity - BAYS(bay).empty_slots;
         [ports] = BAY_find_reservations(bay,'B');
         if and(bayocup == 0, sum(ports)==0)
            keyboard
         end
         IMP.ports.cts = IMP.ports.cts + ports;
```

```
        res = BAYS(bay).R.slots-sum(sum(BAYS(bay).matriz));
        IMP.unusedcts = IMP.unusedcts + BL.capacity - BAYS(bay).R.slots;
        IMP.reservations = IMP.reservations + res;
        IMP.empty = IMP.empty + IMP.bays.cts(bay);
        IMP.unused(bay) = BL.capacity - BL.tiers - BAYS(bay).R.slots;
        IMP.cts = IMP.cts + sum(sum(BAYS(bay).matriz));
    case 'EXP'
        EXP.no.bays = EXP.no.bays + 1;
        EXP.bays.esl(bay) = BAYS(bay).empty_slots;
        EXP.bays.cts(bay) = bayocup; %BL.capacity - BAYS(bay).empty_slots;
        [ports] = BAY_find_reservations(bay,'B');
        EXP.ports.cts = EXP.ports.cts + ports;
        res = BAYS(bay).R.slots-sum(sum(BAYS(bay).matriz));
        EXP.unusedcts = EXP.unusedcts + BL.capacity - BAYS(bay).R.slots;
        EXP.reservations = EXP.reservations + res;
        EXP.unused(bay) = BL.capacity - BL.tiers - BAYS(bay).R.slots;
        EXP.cts = EXP.cts + sum(sum(BAYS(bay).matriz));
    case 'NAS'
        NAS.no.bays = NAS.no.bays + 1;
        NAS.bays.esl(bay) = BAYS(bay).empty_slots;
        % NAS.bays(bay).cts = NAS.bays(bay).cts + BL.capacity - bays(bay).empty_slots;

        %nas_ct(i)=BL.capacity-bays(i).empty_slots;
        % NAS.ports.bays(port) = NAS.ports.bays(port) + 1;
  end
end

IMP.no.cts = sum(IMP.bays.cts);
EXP.no.cts = sum(EXP.bays.cts);

disp('------------------------------------------------------------')
% disp(['Terminal state: % of IMP ct: ' num2str(IMP.no.cts/T.slots*100)])
% disp(['Terminal state: % of EXP ct: ' num2str(EXP.no.cts/T.slots*100)])

% disp('Terminal state: Bay distribution for each port: ')
disp(['% IMP BAYS: ' num2str(sum(IMP.no.bays/T.bays*100)) ])
disp(['% EXP BAYS: ' num2str(sum(EXP.no.bays/T.bays*100)) ])
disp(['% NAS BAYS: ' num2str(sum(NAS.no.bays/T.bays*100)) ])

disp(['% of IMP occupation ' num2str(IMP.cts/T.slots*100)])
disp(['% of EXP occupation ' num2str(EXP.cts/T.slots*100)])

disp(['% of IMP reservation ' num2str(IMP.reservations/T.slots*100)])
disp(['% of EXP reservation ' num2str(EXP.reservations/T.slots*100)])

disp(['Total % of Utilization ' num2str((IMP.cts+EXP.cts+IMP.reservations+EXP.reservations)/T.slots*100)])

disp(['% of IMP unused ' num2str(IMP.unusedcts/T.slots*100)])
disp(['% of EXP unused ' num2str(EXP.unusedcts/T.slots*100)])

disp(['% of IMP bays unused ' num2str(sum(IMP.bays.unused)/T.slots*100)])
disp(['% of EXP bays unused ' num2str(sum(EXP.bays.unused)/T.slots*100)])

disp(['Total % NOT Utilized '
```

```
num2str((IMP.unusedcts+EXP.unusedcts+NAS.no.bays*BL.tiers*BL.stacks)/T.slots*100)])
disp('------------------------------------------------------------')
```

```
% MAIN PROGRAM ~=

function TERMINALmainv2(rsiono,rdelay,trf_level,hotstart)

clear global

global BAYS BL BT COST COUNT CT ET EXEC GROUPS MAC R S T TIME TRF VS YC

EXEC.energy_model = 2;
caso = 1;
% 1. Reference parameters

if hotstart == 1
   rng(caso);
   ct = INIT(1,rsiono,rdelay,trf_level); % Generate, reserve space
   vs_no = 0;  % VS list
   et = 0;  % ET list
   ev = 0;
   TRF.PARAM.overload = 25;
   TRF.PARAM.weight.E = 1;
   TRF.PARAM.weight.t = 1 - TRF.PARAM.weight.E;
elseif hotstart == 2
   rng(caso);
   load('pstate.mat');
   et = COUNT.et;
   vs_no = COUNT.vs;
   ev = R.ev;
   ct = COUNT.ct;
end

TRF.PARAM.averias = 1;
TRF.PARAM.overload = 25;
TIME.super = TIME.simul*1.2;
% 3. START OF THE MAIN LOOP
% ------------------------------------------------------------------------
while TIME.t <= TIME.simul % && ct<TRF.PARAM.max_no_ct

   event = TIME_next();

   % 6.3. Calculate event

   switch event
      % ------------------------------------------------------------------
      case 1 % A new vessel arrives
      % ------------------------------------------------------------------
         %savestate()
         vs_no = vs_no + 1; COUNT.vs = vs_no;
         % activate the vessel berth
         t_bt = VS(vs_no).berth;
         if BT(t_bt).active == 1
            disp('Error: berth occupied')
            keyboard
            %check_cts_vs();
```

```
        else
            BT(t_bt).active = 1; % Maybe this parameter is redundant
            BT(t_bt).vessel = vs_no;
        end
        T_eval();
    % ---------------------------------------------------------------
    case 2 % The external truck comes
    % ---------------------------------------------------------------
        et = COUNT.et + 1; COUNT.et = et;
        switch char(ET(et).id)
            case 'EXP'
                ET_Stack(et);
            case 'IMP'
                ET_pick_CT(et);

        end % end of ET(et).id
    % ---------------------------------------------------------------
    case 3  % EVENT TYPE 3: VESSEL PLAN GENERATION
    % ---------------------------------------------------------------
        savestate()
        COUNT.vs_p = COUNT.vs_p + 1;

        if TRF.PARAM.reservation == 1
            VS_EXP_reservation(COUNT.vs_p);
        end

        %keyboard
        T_eval();
  end
  % ---------------------------------------------------------------------
  % 7. VESSEL DOWNLOAD AND UPLOAD OPERATIONS
  % ---------------------------------------------------------------------
  for berth = 1:T.berth_no
    % Check whether the berth corresponds to the container to be loaded
    if event == 3 + berth

        % a) See what vessel is at berth and its port
        vs = BT(berth).vessel;     port = VS(vs).port;

        % 7.1. VESSEL DOWNLOADING OPERATION OF IMP CT
        if VS(vs).ict < VS(vs).IC % Then we have IMP containers to be downloaded
            VS_download_ct(vs);

        % 7.2. EXP CTs VESSEL LOADING OPERATION
        elseif VS(vs).oct < VS(vs).OC % We have expot containers to be uploaded onto vessel
            VS_upload_ct(vs);
        end
    end
  end

  % ---------------------------------------------------------------------
  % 8. YC OPERATIONS
  % ---------------------------------------------------------------------
%     for row = 1:T.rows
```

```
%        for iyc=1:MAC.YC.ycsprow
%            yc = (row-1)*MAC.YC.ycsprow + iyc;
%            YC_cycle(yc);
%        end
%    end
end

closure()

   wsname = ['C0' num2str(caso) ' TRF' num2str(TRF.PARAM.ct_traffic/1000000) ' R'
num2str(TRF.PARAM.reservation) ' D' num2str(TRF.PARAM.delayreservation) '.mat'];
   disp(['Saving ' wsname ' workspace'])
   save(wsname);

profile off
```

```
% 1 Search for bay stacks with little occupation
% ------------------------------------------------------------------------

if false
   for bay = 1:T.bays
      if and(strcmp(BAYS(bay).id,label) == 1, BAYS(bay).empty_slots > 0)
         for stack = 1:BL.stacks
            if min(BAYS(bay).port(:,stack)) == 0 % There are slots left
               s = s + 1;
               stack_candidates(s,1) = bay;
               esl=0;
               for tier = 1:BL.tiers
                  if BAYS(bay).port(tier,stack) == 0
                     esl = esl + 1;
                  end
               end
               stack_candidates(s,2) = esl;
               %candidates(s,3) =
            end
         end
      end
   end
% Sort the candidates
stack_candidates = sortrows(stack_candidates,2);
end

% 1 Compute the port ocupation
if strcmp(label,'EXP') == 1
%     no_ports = length(T.state.EXP.ports.cts);
%     %cts2move = VS.OC - available_slots;
%     for port = 1:no_ports
%        port_occupation(port) = T.state.EXP.ports.cts(port) / T.state.EXP.ports.bays(port);
%     end
   slots_per_port = T.state.EXP.ports.cts; %(T.state.EXP.ports.bays*BL.capacity) - T.state.EXP.ports.cts;
else
%     no_ports = length(T.state.IMP.ports.cts)
%     for port = 1:no_ports
%        port_occupation(port) = T.state.IMP.ports.cts(port) / T.state.IMP.ports.bays(port);
%     end
   %cts2move = VS.IC - available_slots;
   slots_per_port = T.state.IMP.ports.cts; % (T.state.IMP.ports.bays*BL.capacity) - T.state.IMP.ports.cts;
end
```

```
CB_distances = ones(1,nb)*1000000000;
for i_bay = 1:nb

    bay = CB(i_bay);
    % check if the bay still has CTs for that destination
    [ports] = Matriz_ports_reserved(BAYS(bay));
    if ports(TRF.VS(vs).port) > 0
        CB_distances(i_bay) = distance_calculator(BAYS(bay).position, BT(TRF.VS(vs).berth).position);
    end
end
```

```
function test_etarrival(port)

global TIME TRF

j = 0;
for i = 1:length(TRF.ET)
   if TRF.ET(i).port == port
      if TRF.ET(i).arrival_time < TIME.t
         j=j+1; arrival(j) = TRF.ET(i).arrival_time;
      end
   end
end

figure; plot(arrival/3600/24); hold on
plot(TRF.VS(port).plan.time/3600/24, 'or')
plot(TRF.VS(port).arrival.time/3600/24, 'o')
```

```
function TIME_init()

global TIME

%  Initialize the time, by defining the first time step
TIME.t = 0; %VS(1).plan.time;
TIME.delt = 0;

TIME.start = clock;
TIME.simul = 4;                 % In weeks
TIME.simul = TIME.simul * 7 * 24 * 60 * 60; % In secs
TIME.super = TIME.simul*1.2;
% 3.2. CT TIME parameters
TIME.CT.stay = 2 * 24 * 60 * 60; % (s) Duration of the stay

% 3.3 YC TIME Parameters associated to the YC
% TIME.YC.start = 20;             % (s) Engine start up
% TIME.YC.rehandle = 180;         % (s) Rehandling time per CT
% TIME.YC.load = 185;             % (s) CT loading operation
```

```
function [event] = TIME_next()

global BT COUNT ET MAC R T TIME VS YC

TIME.t = TIME.t + TIME.delt;

% Compute the state of the terminal

if 100 * TIME.t/TIME.simul > R.progreso
   TIME_update();
end

% 6.1. Check the list of events to take place
% 6.1.1. Arrival of vessels
t_del(1) = VS(COUNT.vs + 1).arrival.time - TIME.t;

% 6.1.2. Arrival of ETs
t_del(2) = ET(COUNT.et + 1).arrival_time - TIME.t;

% 6.1.3. Generación de la lista del buque unos días antes
if COUNT.vs_p < length(VS)
   t_del(3) = VS(COUNT.vs_p + 1).plan.time - TIME.t;
else
   t_del(3) = TIME.super;
end

% 6.1.4. VS CT unloading/loading
for berth = 1:T.berth_no
   if BT(berth).active == 1
      vs = BT(berth).vessel; % Maybe this instruction is not necessary
      if VS(vs).ict < VS(vs).IC % DOWNLOAD IMP CTs
         current_ct = VS(vs).ict;
         next_ct = current_ct + 1;
         t_del(3 + berth) = VS(vs).icts(next_ct).time - TIME.t;
      elseif VS(vs).oct < VS(vs).OC % UPLOAD EXP CTs
         current_ct = VS(vs).oct;
         next_ct = current_ct + 1;
         t_del(3 + berth) =VS(vs).octs(next_ct).time - TIME.t; % ;TIME.super
      else
         t_del(3 + berth) = TIME.super;
      end
   else
      t_del(3 + berth) = TIME.super;
   end
end

% for row = 1:T.rows
%    for i=1:MAC.YC.ycsprow
%       yc = (row-1)*MAC.YC.ycsprow + i;
%       t_del(6 + yc) = YC(yc).nextevent;
%    end
% end
```

```
% 6.2. Find the event that will take place
[TIME.delt,event] = min(t_del);
```

```
function TIME_update()

global R T TIME VS

if 100 * TIME.t/TIME.simul > R.progreso
   disp(['Progreso = ' num2str(100 * TIME.t/TIME.simul)])
   disp('------------------------------------------------------------')
   R.progreso = R.progreso + 5;

   [T.state.IMP,T.state.EXP,T.state.NAS] = terminal_state();
   R.ev = R.ev + 1;
      %VS(1).icts(VS(1).IC).time;
   R.tiempo(R.ev) = TIME.t; % Show time in days
   R.ocs(R.ev) = T.state.EXP.no.cts;
   R.ics(R.ev) = T.state.IMP.no.cts;
   R.tcs(R.ev) = R.ocs(R.ev) + R.ics(R.ev);
   R.res(R.ev) = R.tcs(R.ev) + T.state.IMP.reservations + T.state.EXP.reservations;
   plot_evolution()
   savestate()
end
```

```
function [x,t,full_speed,ms] = travel_time(X,s,a,d)

% 1.1 Time
t.ac = s/a;
t.dc = s/d;
t.fs = 0;

% 1.2 Distance needed for acceleration
x.ac = 0.5*a*t.ac^2;
x.dc = s*t.dc-0.5*d*t.dc^2;

% 2 Full speed
% 2.1 Distance traveled at full speed
x.fs = X - x.ac - x.dc;

% 2.2 Time at full speed
if x.fs > 0 % Full speed achieved
    full_speed = 1;
    % time at full speed
    t.fs = x.fs/s;
    ms = s;
else % Full speed not achieved
    full_speed = 0; x.fs = 0;
    t.dc = ( 2*X*a / (d*(a+d)) )^0.5;
    t.ac = ( 2*X*d / (a*(a+d)) )^0.5;
    x.ac = a*t.ac^2/2;
    x.dc = X-x.ac;
    ms = t.ac*a;
end

t.total = t.fs + t.ac + t.dc;
```

```
function TRF_init(trf_file_name)
% This function generates the VS list with atributes:

global ET TIME TRF VS

ET = 0; VS = 0;

disp('Initializing traffic')
disp('---------------------------------------------------------------------')

% 0. Parameters

average_no_QC = 4;

% 1. Initialization
% 1.1. Initialize the Vessels
%VS.et = 0;
VS.IC = 0;
VS.OC = 0;
VS.OC_arrived = [];
VS.port = 0;
VS.plan.time = 0;
VS.plan.cts = [];
VS.arrival.time = 0;
VS.no = 0;

% 1.2. Initialize the trucks
ET.arrival_time = 0;
ET.target_VS = 0;

% 2. Vessel attributes
time = 0; vs = 0; et_arrivals = []; %keyboard
while time < TIME.simul*1.1
   vs = vs + 1;
   % 2.1. vessel identity and features
   VS(vs).no = vs;
   VS(vs).strategy = 'N';
   VS(vs).IC = poissrnd(TRF.PARAM.average_no_IC);
   VS(vs).OC = poissrnd(TRF.PARAM.average_no_OC);
   VS(vs).port = vs; %1 + mod(vs,TRF.PARAM.no_ports);

   % TIME PARAMETERS
   % 2.2. Time series of arrivals
   f = 0.001;  delvs(vs) = poissrnd(f*TRF.PARAM.lambda_vs)/f;

   % MAKE THE PLAN
   plan_time = time + delvs(vs);
   VS(vs).plan.time = plan_time;
   plan_day = floor(plan_time/3600/24)+1;
   VS(vs).plan.day = plan_day;
   VS(vs).strategy = 'N';
   plan.bays = 0; plan.cts =0;
   VS(vs).plan.EXP = plan;
```

```
VS(vs).plan.IMP = plan;

% MAKE THE ARRIVAL
arrival_time = plan_time + (TRF.PARAM.daysinadvance)*3600*24;
VS(vs).arrival.time = arrival_time;
remday = rem(arrival_time,3600*24);
rem_hour = floor(remday/3600);
VS(vs).plan.hour = rem_hour;
arrival_day = floor(arrival_time/3600/24)+1;
VS(vs).arrival.day = arrival_day;

if rem_hour > 24
   keyboard
else
   VS(vs).arrival.hour = rem_hour;
end

VS(vs).operation.start = arrival_time + 0.5*3600;

% 2.3. Generate the time lists of containers
% 2.3.1. indecees for tracking the containers
VS(vs).ict = 0; VS(vs).oct = 0;

disp(['Generate Vessel ' num2str(vs) '  with ' num2str(VS(vs).OC) ' EXP CT and ' num2str(VS(vs).IC) ' IMP CT'])

% 2.3.2. IC list (download)
t0 = VS(vs).operation.start;
for ct = 1:VS(vs).IC
   delt = poissrnd(TRF.PARAM.qcdelt)/average_no_QC;
   VS(vs).icts(ct).time = t0 + delt;
   t0 = t0 + delt;
end
VS(vs).operation.switch = t0;

% 2.3.3. OC list (upload)
for ct = 1:VS(vs).OC
   delt = poissrnd(TRF.PARAM.qcdelt)/average_no_QC;
   VS(vs).octs(ct).time = t0 + delt;
   t0 = t0 + delt;
end

% 2.3.4. End of loading
VS(vs).operation.end = t0;

% 2.4. Assign berth
if rem(vs,3)==0
   berth=3;
else
   berth= rem(vs,3);
end

VS(vs).berth = berth;

% 2.5. Generate trucks associated to a vessel
```

```
    %keyboard
    [arrivals] = ET_arrivals(VS(vs));
    et_arrivals = [et_arrivals; arrivals];

    time = time + delvs(vs);
end

TRF.PARAM.VS_no = vs;
TRF.PARAM.no_ports = vs;
TRF.PARAM.overload = 125;

% 2. ET ATTRIBUTES
% ------------------------------------------------------------------------
et_arrivals = sortrows(et_arrivals,1);
et = length(et_arrivals);

for i = 1:length(et_arrivals)
    ET(i).arrival_time = et_arrivals(i,1);
    if et_arrivals(i,2)== 1
        ET(i).id = 'EXP';
    elseif et_arrivals(i,2)== 2
        ET(i).id = 'IMP';
    end
    ET(i).target_VS = et_arrivals(i,3);
    ET(i).port = et_arrivals(i,4);
end

% 3. Weight cathegories
% ------------------------------------------------------------------------
no_wc = 10;
[TRF.CT.pdf,TRF.CT.cdf,TRF.CT.top_xf]= pdfcdf(10);

% Save traffic
%filename = ['TRF ' num2str(TRF.PARAM.ct_traffic/1000000) 'MTEU ' num2str(TRF.PARAM.no_ports) 'ports '
num2str(average_no_IC/1000) ' kVCTs.mat'];
%trf_file_name = ['TRF ' num2str(TRF.PARAM.ct_traffic/1000000) 'MCTs ' num2str(TRF.PARAM.no_ports) 'Ports '
num2str(TRF.PARAM.average_no_IC/1000)  'kVCTs.mat'];
save(trf_file_name,'ET','TRF','VS');

disp('End of traffic generation')
```

```
function TRF_init(trf_file_name)
% This function generates the VS list with atributes:

global ET TIME TRF VS

ET = 0; VS = 0;

disp('Initializing traffic')
disp('---------------------------------------------------------------------')

% 0. Parameters

average_no_QC = 4;

% 1. Initialization
% 1.1. Initialize the Vessels
%VS.et = 0;
VS.IC = 0;
VS.OC = 0;
VS.OC_arrived = [];
VS.port = 0;
VS.plan.time = 0;
VS.arrival.time = 0;
VS.no = 0;

% 1.2. Initialize the trucks
ET.arrival_time = 0;
ET.target_VS = 0;

% 2. Vessel attributes
initime = 3*24*3600;
time = initime; vs = 0; et_arrivals = []; %keyboard
while time < initime + TIME.simul*1.1
   vs = vs + 1;
   % 2.1. vessel identity and features
   VS(vs).no = vs;
   VS(vs).strategy = 'N';
   VS(vs).IC = poissrnd(TRF.PARAM.average_no_IC);
   VS(vs).OC = poissrnd(TRF.PARAM.average_no_OC);
   VS(vs).port = vs; %1 + mod(vs,TRF.PARAM.no_ports);

   % TIME PARAMETERS
   % 2.2. Time series of arrivals
   f = 0.001;  delvs(vs) = poissrnd(f*TRF.PARAM.lambda_vs)/f;

   % MAKE THE PLAN
   VS(vs).arrival.time = time + delvs(vs);
   VS(vs).arrival.day = floor(VS(vs).arrival.time/3600/24)+1;
   remday = rem(VS(vs).arrival.time,3600*24);
   rem_hour = floor(remday/3600);
   if rem_hour > 24
      keyboard
   else
```

```
        VS(vs).arrival.hour = rem_hour;
    end

    VS(vs).strategy = 'N';
    VS(vs).plan.EXP = 0;
    VS(vs).plan.IMP = 0;

    VS(vs).operation.start = VS(vs).arrival.time + 0.5*3600;

    % 2.3. Generate the time lists of containers
    % 2.3.1. indecees for tracking the containers
    VS(vs).ict = 0; VS(vs).oct = 0;

    disp(['Generate Vessel ' num2str(vs) '  with ' num2str(VS(vs).OC) ' EXP CT and ' num2str(VS(vs).IC) ' IMP CT'])

    % 2.3.2. IC list (download)
    t0 = VS(vs).operation.start;
    for ct = 1:VS(vs).IC
        delt = poissrnd(TRF.PARAM.qcdelt)/average_no_QC;
        VS(vs).icts(ct).time = t0 + delt;
        t0 = t0 + delt;
    end
    VS(vs).operation.switch = t0;

    % 2.3.3. OC list (upload)
    for ct = 1:VS(vs).OC
        delt = poissrnd(TRF.PARAM.qcdelt)/average_no_QC;
        VS(vs).octs(ct).time = t0 + delt;
        t0 = t0 + delt;
    end

    % 2.3.4. End of loading
    VS(vs).operation.end = t0;

    % 2.4. Assign berth
    if rem(vs,3)==0
        berth=3;
    else
        berth= rem(vs,3);
    end

    VS(vs).berth = berth;

    % 2.5. Generate trucks associated to a vessel
    %keyboard
    [arrivals] = ET_arrivals2(VS(vs));
    et_arrivals = [et_arrivals; arrivals];

    time = time + delvs(vs);
end

TRF.PARAM.VS_no = vs;
TRF.PARAM.no_ports = vs;
TRF.PARAM.overload = 125;
```

```
% 2. ET ATTRIBUTES
% --------------------------------------------------------------------
et_arrivals = sortrows(et_arrivals,1);
et = length(et_arrivals);

for i = 1:length(et_arrivals)
   ET(i).arrival_time = et_arrivals(i,1);
   if et_arrivals(i,2)== 1
      ET(i).id = 'EXP';
   elseif et_arrivals(i,2)== 2
      ET(i).id = 'IMP';
   end
   ET(i).target_VS = et_arrivals(i,3);
   ET(i).port = et_arrivals(i,4);
end

% 3. Weight cathegories
% --------------------------------------------------------------------
no_wc = 10;
[TRF.CT.pdf,TRF.CT.cdf,TRF.CT.top_xf]= pdfcdf(10);

% Save traffic
%filename = ['TRF ' num2str(TRF.PARAM.ct_traffic/1000000) 'MTEU ' num2str(TRF.PARAM.no_ports) 'ports '
num2str(average_no_IC/1000) ' kVCTs.mat'];
%trf_file_name = ['TRF ' num2str(TRF.PARAM.ct_traffic/1000000) 'MCTs ' num2str(TRF.PARAM.no_ports) 'Ports '
num2str(TRF.PARAM.average_no_IC/1000)  'kVCTs.mat'];
save(trf_file_name,'ET','TRF','VS');

disp('End of traffic generation')
```

```
function [pos,TRF.ET]=place_ct(TRF.ET,arrival_time)

n=length(TRF.ET);
pos=1;
while arrival_time>ET(pos).arrival_time
   pos=pos+1;
end
% Move the containers
for j=n+1:-1:pos+1
   ET(j)=ET(j-1);
end
ET(pos).arrival_time=arrival_time;
```

```
function [vector,stacks] = vector_orientation(vector)
% This function simply rotates the vector to put it vertical?

[tiers,stacks] = size(vector);

if tiers + stacks > 2 % The vector is of size greater than 1
    if tiers> stacks
        vector = vector';
        stacks = tiers;
    end
end
```

```
function [QC,VS]=vessel_qc_split(VS,QC,BT,T)
% This function generates the bays of the vessel that arrives and assigns
% the workload of the QCs based on how many containers per bay there are

% 0. Variables needed for later
central_vs_bay=fix(VS.bays_no/2);
delx_bay=15; % lets start with 15 m between bays in average

% 1. Stage I: Assign berth to the vessel
% 1.2. First check whether all berths are occupied
index=0;
for bt=1:T.berth_no
  if BT(bt).status=='empty'
     index=index+1;
  end
end

% 1.3. Find the best berthing option based on QC ocupation.
   % Another criteria could be the proximity of the containers to serve.
   % .....................................................................
if index~=0 % if not all the berths are occupied
   VS.status=='atbt';
   empty_quays=zeros(1,T.berth_no);
   for bt=1:T.berth_no
      for qc=1:BT(bt).QC_no
         qc_id=BT(bt).QC_id(qc) % Get id of the  QC to assign to a VBay
         if QC(qc_id).berth==bt
            if QC(qc_id).status~='ocup' % Check if the QC is busy
            empty_quays(bt)=empty_quays(bt)+1;
            end
         end
      end
   end

   % The first berth with more empty quays gets the assignment
   [value,berth]=min(empty_quays(bt));
   VS.target_berth=berth;

% 1.4. Find the quays ids for the berth and the vessel
% berth is QC(qc).berth, and the QC number is BT(berth).QC_id
   qc_ini=BT(berth).QC_id(1); qc_fin=BT(berth).QC_id(4);

% 2. Stage II: assign QCs to the vessel positions
% 2.1. Target bays need to be assigned to the QCs
% 2.1.1. Lets make it easier to track the vector of container bays
   for bay=1:VS.bays_no
      CperB(bay)=VS.bays(bay).no_C;
   end

   av_no_cont=sum(CperB)/BT(berth).QC_no;

   quay=0; bay=0; qc=qc_ini-1;lastbay=1;
   isok='N';
```

```
while bay<VS.bays_no
   qc=qc+1;
   if qc>qc_fin
      'Number of qc per berth exceeded'
      keyboard
      qc=qc_fin;
      for i=bay+1:VS.bays_no
         CperQ=CperQ+CperB(i);
         VS.bays(i).target_QC=qc;
         bay=i;
      end
      isok='Y';
   end
   CperQ=0; dif=av_no_cont-CperQ;
   while dif >0 && bay<VS.bays_no
      bay=bay+1;
      CperQ=CperQ+CperB(bay);
      VS.bays(bay).target_QC=qc;  % assign a QC to the vessel bay
      VS.bays(bay).status='load'; % The vessel bay is loaded
      VS.bays(bay).position(1)=BT(berth).position(1)+(bay-central_vs_bay)*delx_bay;
      VS.bays(bay).position(2)=T.width;
      dif=av_no_cont-CperQ;
   end

   % Lets see whether the next bay improves the share of C between bays
   if isok=='N'
      CperQ=CperQ-CperB(bay);
      dif=av_no_cont-CperQ;
      bay=bay-1;
      dif2=CperQ+CperB(bay+1)-av_no_cont;
      if dif2<abs(dif) && bay<=VS.bays_no
         CperQ=CperQ+CperB(bay+1);
         VS.bays(bay+1).target_QC=qc;
         bay=bay+1;
      end
   end

   % 2.2. Assign the total number of vessel bays to each QC. The IDs are assigned later on
   if isok=='N'
      if qc==1
         QC(qc).vs_bays_no=bay;
      else
         QC(qc).vs_bays_no=bay-lastbay;
      end
   else
     QC(qc).vs_bays_no=bay-lastbay;
   end
   lastbay=bay;
   % It should be implemented a way to check wether the load per QC is
   % adequate. Otherwise the QC would be empty for other tasks
   %.....................................................................
end
```

```
% 3. Assign to each QC the number of vessel bays and the number of
% containers in each bay
bay=0;



% 3.2. Find the target bay
for qc=qc_ini:qc_fin

    % a) Assign the containers per bay to the QC andCalculate the absolute position of the VS BAYS
    for j=1:QC(qc).vs_bays_no
        bay=bay+1; %(qc-1)*4+j
        QC(qc).Bay(j).id=bay;
        QC(qc).Bay(j).no_VS_C=CperB(bay);
        % 3.1. Find the target position of the QC.
        QC(qc).Bay(j).position(1)=VS.bays(bay).position(1);
        QC(qc).Bay(j).position(2)=T.width;
    end

    % b) Among the VS BAYs, find which one is closer to the quay
    bay_ini=QC(qc).Bay(1).id
    bay_fin=QC(qc).Bay(1)+QC(qc).vs_bays_no-1;
    for bay=bay_ini:bay_fin
        if QC(qc).Bay(bay).no_VS_C~=0
            dist(bay)=distance_calculator(QC(qc).position,QC(qc).bay(bay).position);
        else
            dist(bay)=10000;
        end
    end

    % b) Check the target bay is not empty
    if sum(QC(qc).Bay(:).no_VS_C)==0
        QC(qc).status='w4lo'; % The quay can start loading
    else % if there are BAYs with containers to download
        % Find the closer bay
        [QC(qc).target_distance,target_bay]=min(dist);
        QC(qc).target_bay=target_bay;
        % Assign a target position to each crane
        QC(qc).target_position(2)=T.width;
        QC(qc).target_position(1)=QC(qc).Bay(target_bay).position(1);
        gantry_speed=45/60; % 45m/min CHECK IF THIS HAS TO GO IN CYCLE
        QC(qc).t4completion=QC(qc).target_distance/gantry_speed;
    end

    %------------------
    QC(qc).VS_id=VS.id;
%       target_distance=distance_calculator(QC(qc).position,QC(qc).target_position);
%       QC(qc).t4completion=target_distance
  end


end % of the if
```

```
function VS_download_ct(vs)
% This function takes an IMP CT from the VS and places it in a bay at the Terminal.
% Criteria for bay allocation:
% - YC availability
% - Distance has been previoulsy considered through the VS.IMP reservation plan

global BAYS BL BT COUNT COST CT R T TIME YC VS

%keyboard

if VS(vs).ict < VS(vs).IC
   CT_generate('IMP',vs);
   ct = COUNT.ct;
end

% Upon Download of the first container
if VS(vs).ict == 0
   VS(vs).seed=0; %keyboard
   % MAKE IMP RESERVATION
   VS_IMP_reservation(vs);
   %vs_imp_cts(VS(vs).port) = 0;
   terminal_state();
   disp('-------------------------------------------------')
   disp(['Start downloading ' num2str(VS(vs).IC) ' CTs for VS ' num2str(VS(vs).no)])
   disp('-------------------------------------------------')

   plot_evolution()
end

% 1. BAY ALLOCATION
%-------------------------------------------------------------------
BAY_selection_R(VS(vs).plan.IMP,ct);

% 3. Place the container on the bay
%-------------------------------------------------------------------------
BAY_individual_allocation(ct);

CT(ct).P.yc = YC_assign_ct(CT(ct).P.bay);

% 3.1 Remove container from the vessel
VS(vs).ict = VS(vs).ict + 1;
VS(vs).plan.IMP.usedcts(VS(vs).ict) = ct;
VS(vs).plan.IMP.usedbays(VS(vs).ict) = CT(ct).P.bay;

disp(['VS(' num2str(VS(vs).no) ') IMP CT(' num2str(ct) '). ' num2str(VS(vs).ict + 1) ' out of ' num2str(VS(vs).IC) ' to
BAY(' num2str(CT(ct).P.bay) ')'])
%disp(['Block ' num2str(BL.capacity - BAYS(target_bay).empty_slots) ' out of ' num2str(BAYS(target_bay).R.slots)])

% Update vsarrival
COUNT.vsarrivals.imp(vs) = COUNT.vsarrivals.imp(vs) + 1;
R.vsct.imp(VS(vs).no,COUNT.vsarrivals.imp(VS(vs).no)) = TIME.t + TIME.delt;

[C] = YC_calc_stack(ct,0);
```

```
[G,Tr,H,UG,UT,UH,UW] = Energybreakdown(C);
COST.YC.IMP.stack.gantry = COST.YC.IMP.stack.gantry + G.e + UG.e;
COST.YC.IMP.stack.trolley = COST.YC.IMP.stack.trolley + Tr.e + UT.e;
COST.YC.IMP.stack.hoist = COST.YC.IMP.stack.hoist + H.e + UH.e;

% END OF VS DOWNLOAD OPERATION
%-----------------------------------------------------------
if VS(vs).ict == VS(vs).IC
   %keyboard
   disp([ num2str(COUNT.vsarrivals.imp(VS(vs).no)) ' CTs downloaded out of '  num2str(VS(vs).IC)])
   % check_terminal_ocupation(bays,'EXP',BL,T,TRF.PARAM.no_ports,VS(vs))
   plot_bays
   plot_evolution()
end
```

```
function VS_EXP_reservation(vs)
% This function calculates the bays candidate to receive the IMP and EXP
% containers from VS

global BL COUNT T TRF VS

%disp('El error esta en que las bahias reservadas se duplican')
%keyboard
C_slots0 = 0; C_slots1 = 0; C_slots2 = 0; C_slots3 = 0; C_slots4 = 0; iscomplete = false;

% Stage I: Calcualte the distance and number of slots of the candidate
% bays. There are three requisites for candidate bays:
% 1) Bays shall be export EXP, or NAS.
% 2) The bay shall not be associated to any other plan: .vs_plan=0 (Or .port?)
% 3) Bays shall have empty slots
% Stage II: Assign the bays to that particular vessel.
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
% 0. DISPLAY SUBROUTINE
%--------------------------------------------------------------------------
VS(vs).strategy = 'N';

VS(vs).oc_list(1:VS(vs).OC) = 0; % the list needs to be initialized outside the next routine
disp('------------------------------------------------------------')
disp(['VS(' num2str(vs) ') Yard Storage Reserve calcualtion' ])

CTs_arrived = COUNT.inventory.exp(VS(vs).port);
slots_needed = VS(vs).OC - CTs_arrived;
disp([ num2str(CTs_arrived) ' CTs arrived for a Vessel ' num2str(VS(vs).OC) ' inventory ' num2str(slots_needed) ' are
needed' ])

%--------------------------------------------------------------------------
% 1. CHECK CONTAINERS STACKED WITH NO RESERVATION
%--------------------------------------------------------------------------

if TRF.PARAM.reservation > 0

   [CB0(:,1),CB0(:,2),CB0(:,3)] = VS_initial_reservation(vs); % Add bays non reserved to the vessel reservation plan
   C_slots0 = sum(CB0(:,2)); %keyboard

   if C_slots0 > 0
      disp([ num2str(length(VS(vs).plan.EXP.bays)) ' Bays are already in the plan from previous arrivals with '
num2str(C_slots0) ' Empty Slots'])
      sCB0 = sortrows(CB0,3);
      slots2res = min(C_slots0,slots_needed);
      if slots2res == 0
         %keyboard
         return
      end
      [reserved_bays,cts_reserved,VS(vs)] = T_reservation(sCB0,slots2res,'N',VS(vs),'EXP');
      slots_needed = VS(vs).OC - CTs_arrived - cts_reserved;
      if slots_needed == 0
```

```
        iscomplete = true;
      end
   end
end

disp('------------------------------------------------------------')

[pure_ports,stack_ports,mixed_ports] = check_reserve();
tot_space = pure_ports + stack_ports + mixed_ports;

%-------------------------------------------------------------------------
% STAGE I.Calcualte the distance and number of slots of the candidate
%-------------------------------------------------------------------------
% 1. FIRST, EXP Bays of the same group
%-------------------------------------------------------------------------
VS(vs).strategy = 'Uniform';
% 1.1 Search for bay candidates ----------------------------------------
% ----------------------------------------------------------------------
if iscomplete == false
   [iscomplete,C_slots1,CB1(:,1)] = BAYS_find_slots('EXP','N',VS(vs),'VSplan',slots_needed);
   if C_slots1 > 0
      if length(CB1(1,:))>1
         sCB1 = sortrows(CB1,3); % Sort according to distance
      else
         sCB1 = CB1;
      end
      % Make reservations ----------------------------------------------
      [reserved_bays,cts_reserved,VS(vs)] = T_reservation(sCB1,C_slots1,'N',VS(vs),'EXP');
      % ----------------------------------------------------------------
      slots_needed = slots_needed - cts_reserved;
   end
end

%-------------------------------------------------------------------------
% 2. Second, try NAS Bays
%-------------------------------------------------------------------------
if iscomplete == false %and(available_slots > 0, available_slots < VS.OC)
   disp(['VS_Plan: NAS bays requested to download Vessel ' num2str(VS(vs).no)])

   % 2.1 Search for bay candidates
   VS(vs).strategy = 'Not mixing with NAS bays';
   %---------------------------------------------------------------------
   [iscomplete,C_slots2,CB2] = BAYS_find_slots('NAS','N',VS(vs),'VSplan',slots_needed);
   %---------------------------------------------------------------------

   if C_slots2 > 0
      sCB2 = sortrows(CB2',3); %keyboard
      % 2.3 Make reservations ---------------------------------------
      [reserved_bays2,cts_reserved2,VS(vs)] = T_reservation(sCB2,C_slots2,'N',VS(vs),'EXP');
      % -------------------------------------------------------------
      slots_needed = slots_needed - cts_reserved2;
   end
end
```

```
%-------------------------------------------------------------------------
% 3. Second, try allocation strategies
%-------------------------------------------------------------------------
if iscomplete == false
   disp(['VS_Plan: MIX or REMARSHALL needed for VS: ' num2str(VS(vs).no)])
   disp(['Stil ' num2str(slots_needed) ' slots needed for ' num2str(VS(vs).OC) ' CTs'])
   occup = 100*(T.state.EXP.no.cts+T.state.IMP.no.cts)/T.bays/BL.capacity;
   disp(['Stack mixing required when occupation is (%): ' num2str(occup)]);
   % A) Mixing in stacks
   VS(vs).strategy = 'Stack mixing';

   %-----------------------------------------------------------------
   [iscomplete,C_slots3,CB3] = BAY_mixing_stack('EXP',VS(vs),slots_needed);
   %-----------------------------------------------------------------

   if C_slots3 > 0
      % Make reservations ---------------------------------------------
      sCB3 = sortrows(CB3,3); %keyboard
      [reserved_bays3,cts_reserved3,VS(vs)] = T_reservation(sCB3,C_slots3,'S',VS(vs),'EXP');
      %-----------------------------------------------------------------
      slots_needed = slots_needed - cts_reserved3;
      % Check the reservation made
      [pure_ports2,stack_ports2,mixed_ports2] = check_reserve();
      tot_space2 = pure_ports2 + stack_ports2 + mixed_ports2;
      tot_reserve = tot_space2 - tot_space;
   end
end

% B) NOW CHOOSE BETWEEN TOTAL MIXING OR MARSHALLING
%-----------------------------------------------------------------
if iscomplete == false

   option='Mix';

   if strcmp(option,'Remarshall') == 1
      % 3.1. Move CTs to other locations--------------------------------
      remarshalling(slots_needed,VS,'EXP');
      % 3.2. Identify candidate bays ----------------------------------
      [iscomplete,C_slots4,CB4(:,1),CB4(:,2),CB4(:,3)] =
BAYS_find_slots('NAS','N',BL.capacity,VS(vs),'VSplan',slots_needed);
      %-----------------------------------------------------------------
      sCB4 = sortrows(CB4',3);
      if C_slots4 > 0
         % 3.3. Make reservations ---------------------------------------
         [reserved_bays4,cts_reserved4,VS(vs)] = T_reservation(sCB4,C_slots4,'EXP',VS(vs),'EXP');
         % -------------------------------------------------------------
         slots_needed = slots_needed - cts_reserved4;
      end
   elseif strcmp(option,'Mix') == 1

      occup = 100*(T.state.EXP.no.cts+T.state.IMP.no.cts)/T.bays/BL.capacity;
      disp(['Total mixing required when occupation is (%): ' num2str(occup)]);
      VS(vs).strategy = 'Total mixing';
      % Find candidate bays ---------------------------------------
```

```
      [iscomplete,C_slots4,CB4] = BAY_mixing_total('EXP',VS(vs),slots_needed);
      %-------------------------------------------------------------
      sCB4 = sortrows(CB4,3); %keyboard

      if C_slots4 > 0
         % Make reservations ----------------------------------------
         %keyboard
         [reserved_bays4,cts_reserved4,VS(vs)] = T_reservation(sCB4,C_slots4,'Y',VS(vs),'EXP');
         %-------------------------------------------------------------
         slots_needed = slots_needed - cts_reserved4;

         [pure_ports2,stack_ports2,mixed_ports2] = check_reserve();
         tot_space2 = pure_ports2 + stack_ports2 + mixed_ports2;
         tot_reserve = tot_space2 - tot_space;
         if min(tot_reserve)<0
            'Error'
            keyboard
         end
      end
   end
end

if slots_needed > 0
   disp(['VS Plan warning: not enough slots to accomodate CTs. ' num2str(slots_needed) ' slots are needed'])
   %terminal_state();
   %keyboard
else
   [pure_ports2,stack_ports2,mixed_ports2] = check_reserve();
   space_reserved = pure_ports2(VS(vs).port) + stack_ports2(VS(vs).port) + mixed_ports2(VS(vs).port);
   %plot_bays_reserved
   disp(['VS(' num2str(VS(vs).no) ') Yard storage reserve plan completed with ' num2str(space_reserved) ' CTs using
strategy: ' VS(vs).strategy])
   tot_space2 = pure_ports2 + stack_ports2 + mixed_ports2;
   tot_reserve = tot_space2 - tot_space;
   disp(['Total reserve: ' num2str(tot_reserve)])
end

if COUNT.vs_p > 1
   plot_evolution()
end
```

```
function VS_IMP_reservation(vs)
% This function makes the IMP CT reservation for a given VS.

global BL T VS

VS(vs).plan.IMP.bays = 0;  VS(vs).plan.IMP.cts = 0;  vessel = VS(vs);

disp(['Generating VS ' num2str(vs) ' IMP plan for ' num2str(VS(vs).IC) ' CTs'])

% 1. Parameters

not_assigned_slots = VS(vs).IC;

[IMP,EXP,NAS] = terminal_state();
t_ocup = (IMP.cts + EXP.cts)/T.slots*100;

% First check if the terminal is too crowded
if IMP.unusedcts > 0.5*IMP.cts %and(IMP.no.bays/EXP.no.bays > 50/50, t_ocup > 30)
   [iscomplete,C_slots3,CB3] = BAY_mixing_stack('IMP',VS(vs),not_assigned_slots);
   if C_slots3 > 0
      % Make reservations ---------------------------------------------
      sCB3 = sortrows(CB3,3); %keyboard
      [reserved_bays3,cts_reserved3,vessel] = T_reservation(sCB3,C_slots3,'S',vessel,'IMP');
      %--------------------------------------------------------------------
      not_assigned_slots = not_assigned_slots - cts_reserved3;
      % Check the reservation made
      [pure_ports2,stack_ports2,mixed_ports2] = check_reserve();
   end
   % 2.1 Try IMP bay with at least some empty slots
   if iscomplete == false
      [iscomplete,C_slots,CB] = BAYS_find_slots('IMP','N',vessel,'VSdownload',not_assigned_slots);
      %%keyboard
      if C_slots > 0
         % Make reservations ------------------------------------------
         [reserved_bays,cts_reserved,vessel] = T_reservation(CB',C_slots,'N',vessel,'IMP');
         % ------------------------------------------------------------
         not_assigned_slots = not_assigned_slots - cts_reserved;
      end
   end
      if iscomplete == false
      [iscomplete,C_slots2,CB2] = BAYS_find_slots('NAS','N',vessel,'VSdownload',not_assigned_slots);
      if C_slots2 > 0
         % 2.3 Make reservations ------------------------------------
         %keyboard
         [reserved_bays2,cts_reserved2,vessel] = T_reservation(CB2',C_slots2,'N',vessel,'IMP');
         % ------------------------------------------------------------
         not_assigned_slots = not_assigned_slots - C_slots2;
      end
   end
else

   % 2. Find candidate bays
   %--------------------------------------------------------------------
```

```
   % 2.1 Try IMP bay with at least some empty slots
   [iscomplete,C_slots,CB] = BAYS_find_slots('IMP','N',vessel,'VSdownload',not_assigned_slots);

   if C_slots > 0
      % Make reservations ---------------------------------------------
      [reserved_bays,cts_reserved,vessel] = T_reservation(CB',C_slots,'N',vessel,'IMP');
      % -------------------------------------------------------------
      not_assigned_slots = not_assigned_slots - cts_reserved;
   end

   % 2.2 If there aren't, check the not assigned bays
   if iscomplete == false
      [iscomplete,C_slots2,CB2] = BAYS_find_slots('NAS','N',vessel,'VSdownload',not_assigned_slots);
      if C_slots2 > 0
         % 2.3 Make reservations ------------------------------------
         %keyboard
         [reserved_bays2,cts_reserved2,vessel] = T_reservation(CB2',C_slots2,'N',vessel,'IMP');
         % ---------------------------------------------------------
         not_assigned_slots = not_assigned_slots - C_slots2;
      end
   end

   if iscomplete == false
   %-----------------------------------------------------------------------
      [iscomplete,C_slots3,CB3] = BAY_mixing_stack('IMP',VS(vs),not_assigned_slots);
      %-----------------------------------------------------------------

      if C_slots3 > 0
         % Make reservations ----------------------------------------
         sCB3 = sortrows(CB3,3);
         [reserved_bays3,cts_reserved3,vessel] = T_reservation(sCB3,C_slots3,'S',vessel,'IMP');
         %-------------------------------------------------------------
         not_assigned_slots = not_assigned_slots - cts_reserved3;
         % Check the reservation made
         [pure_ports2,stack_ports2,mixed_ports2] = check_reserve();
      end
   end
end

if iscomplete == false
   disp(['Vs IMP Reservation Error: there is no space available to download vessel: ' num2str(vessel.no)])
   plot_bays
   keyboard
   check_bay_types()
end

VS(vs) = vessel;
```

```
function [CB,CB_slots,CB_distances] = VS_initial_reservation(vs)
% This function checks the bays with containers placed without reservation
% to add them to the vessel plan

global BAYS BL BT T VS

%plot_bays
i_bay = 0;
CB=0; CB_slots=0; CB_distances=0;
%keyboard
for bay = 1:T.bays
   if strcmp(BAYS(bay).id,'EXP') == 1
      %[ports,es]  = Search_Port_Matriz(bay);
      %[nports] = ports_howmanyofeach(BAYS(bay).R.ports);
      savailable = max(0,BAYS(bay).empty_slots-BL.tiers); %BAYS(bay).empty_slots; %
%        if savailable == 0
%           continue
%        end
      nports = BAY_find_reservations(bay,'C');
      if nports(vs) > 0
         i_bay = i_bay+1;
         CB(i_bay) = bay;
         CB_cts(i_bay) = nports(vs);
         CB_slots(i_bay) = savailable;
         CB_distances(i_bay) = distance_calculator(BAYS(bay).position, BT(VS(vs).berth).position);
      end
   end
end

if i_bay >0
   VS(vs).plan.EXP.bays  = CB';
   VS(vs).plan.EXP.cts  = CB_cts';
end
```

```
function VS_inventory(vs)

global BAYS VS
keyboard
ects = 0;
if length(VS(vs).plan.EXP.bays) == 0
   keyboard
else
   for i = 1:length(VS(vs).plan.EXP.bays)
      bay = VS(vs).plan.EXP.bays(i);
      ects = ects + length(find(BAYS(bay).R.ports==vs));
   end
end
disp('-----------------------------------------------------------------')
disp(['Situation of the VS(' num2str(vs) ') inventory']);
disp([num2str(ects) ' slots reserved in the yard'])
disp([num2str(VS(vs).OC) ' cts to be loaded'])
disp([num2str(length(VS(vs).OC_arrived)) ' cts have arrived'])

plot_VS_plan(vs)
```

```
function [plan] = VS_plan_generation(oldplan,newbays,newcts)
% This function takes the vessel IMP/EXP bay list and adds new bays

global BL

%keyboard
if length(newbays) ~= length(newcts)
   keyboard
end
% Orient the vectors before adding them together
if sum(oldplan.bays) == 0
   plan.bays = newbays;   plan.cts = newcts;
else
   %keyboard
   % Bays
   [oldplan.bays,B] = vector_orientation(oldplan.bays);
   [newbays,NB] = vector_orientation(newbays);
   plan.bays = [oldplan.bays,newbays];
   plan.bays = unique(plan.bays); % Unique also sorts
   % Cts
   newind = find(ismember(plan.bays, newbays)>0);
   lb = length(plan.bays);%plan.cts(lc+1:lb) = 0;lc = length(plan.cts);
   plan.cts = zeros(1,lb);
   % copy old containers
   for i = 1:length(oldplan.bays)
      index = find(plan.bays == oldplan.bays(i));
      plan.cts(index) = oldplan.cts(i);
   end
   % copy new containers in the new positions
   if length(newbays) ~= length(newcts)
      keyboard
   end
   for i = 1:length(newbays)
      index = find(plan.bays == newbays(i));
      plan.cts(index) = plan.cts(index) + newcts(i);
   end
end
```

```
function VS_upload_ct(vs)
% This function takes containers from the yard and places them in the
% vessel. It searches within the reserved bays
global BAYS BT COST COUNT CT VS

% Initialization of the operation: make loading plan
if VS(vs).oct == 0
   disp('-------------------------------------------------')
   disp(['Start uploading ' num2str(VS(vs).OC) ' CTs to VS ' num2str(VS(vs).no)])
   disp('-------------------------------------------------')
   %keyboard
   VS(vs).plan.EXP = 0;
   if sum(VS(vs).plan.EXP) == 0
      VS(vs).plan.EXP = T_bay_search(vs);
   end
end

% Now put all those CTs in the YC WLs
oct = VS(vs).oct +1;
VS(vs).oct = oct;
ct = VS(vs).plan.EXP(oct);

% verify that the ct hasn't moved
bay = CT(ct).P.bay;

%oldres = BAYS(bay).R.slots;
if ct ~= BAYS(bay).ct_id(CT(ct).P.tier,CT(ct).P.stack)
   keyboard
   [CT(ct).P.tier,CT(ct).P.stack] = find(BAYS(bay).ct_id==ct);
end

CT(ct).P.yc = YC_assign_ct(bay);

% [reshuffles,r_heights] = BAYS_reshuffles(ct);
%
% COST.reshuffle.EXP = COST.reshuffle.EXP + reshuffles;
% COST.r_heights.EXP = COST.r_heights.EXP + r_heights;
% COUNT.deliveries.exp(vs) = COUNT.deliveries.exp(vs)+1;

[C] = YC_calc_delivery(ct);

[G,Tr,H,UG,UT,UH,UW] = Energybreakdown(C);
COST.YC.EXP.deliver.gantry = COST.YC.EXP.deliver.gantry + G.e + UG.e;
COST.YC.EXP.deliver.trolley = COST.YC.EXP.deliver.trolley + Tr.e + UT.e;
COST.YC.EXP.deliver.hoist = COST.YC.EXP.deliver.hoist + H.e + UH.e;

CT_erase(ct);
%
% if bay == 257
%    oldres
%    newres = BAYS(bay).R.slots
%    %keyboard
% end
```

```
% if oldres == newres
%     keyboard
% end

if VS(vs).oct == VS(vs).OC
    bt = VS(vs).berth;
    BT(bt).active = 0;
    BT(bt).vessel = 0;
end
%plot_yc_wl()
%disp('------------------------------------------------------------')
disp([char(CT(ct).type) ' CT(' num2str(ct) ') Uploaded by VS(' num2str(vs) ') ' num2str(oct) ' out of '
num2str(VS(vs).OC)])
```

```
function YC_add_event(ct,E,t,move)

global CT COUNT YC TIME TRF

%keyboard

yc = CT(ct).P.yc;

e = YC(yc).WL.n +1; YC(yc).WL.n = e;
YC(yc).WL.cts(e) = ct;
YC(yc).WL.time(e) = TIME.t+TIME.delt;
YC(yc).WL.move{e} = move;
YC(yc).WL.E(e) = E;
YC(yc).WL.duration(e) = t;

YC(yc).WL.cwl = length(find(YC(yc).WL.time > TIME.t - 60*60));

if YC(yc).WL.cwl > TRF.PARAM.overload
   COUNT.YC.overload(yc).no = COUNT.YC.overload(yc).no + 1;
   COUNT.YC.overload(yc).type{COUNT.YC.overload(yc).no} = move;
   COUNT.YC.overload(yc).time(COUNT.YC.overload(yc).no) = TIME.t;
end
```

```
function YC_addtask(ct,time,move)

global COUNT CT TIME TRF YC

yc = CT(ct).P.yc;

n = YC(yc).WL.n+1;

YC(yc).WL.n = n;
YC(yc).WL.cts(n) = ct;
YC(yc).WL.time(n) = TIME.t + time;
YC(yc).WL.move{n} = move;

%keyboard
YC(yc).WL.cwl = length(find(YC(yc).WL.time > TIME.t - 60*60));

if YC(yc).WL.cwl > TRF.PARAM.overload
   COUNT.YC.overload(yc).no = COUNT.YC.overload(yc).no + 1;
   COUNT.YC.overload(yc).type{COUNT.YC.overload(yc).no} = move;
   COUNT.YC.overload(yc).time(COUNT.YC.overload(yc).no) = TIME.t;
end
```

```
function [yc] = YC_assign_ct(tbay)
% This function calculates the YC to be assigned to a CT

global BAYS CT MAC TRF YC

% Find the YC that will take the CT
% ---------------------------------------------------------------------
trow = BAYS(tbay).row;

for iyc=1:MAC.YC.ycsprow
   % Identify the YC
   yc = (trow-1)*MAC.YC.ycsprow + iyc;
   % Calculate distances to the target
   % First, look at the position of the crane in its last assignment
   if YC(yc).WL.cwl == 0
      bay = YC(yc).P.bay;
   else
      l_ct = YC(yc).WL.cts(end); bay = CT(l_ct).P.bay;
   end
   yc_pos(1) = BAYS(bay).position(1);
   yc_pos(2) = BAYS(bay).position(2);

   distreal(iyc) = abs(BAYS(tbay).position(1)-yc_pos(1));
   wl(iyc) = YC(yc).WL.cwl;
end

% Compute the coeficients of each YC and choose the one with smaller value
if max(wl) == 0
   denom = TRF.PARAM.overload;
else
   denom = max(wl);
end
yccoefs = 0.5*distreal/max(distreal) + 0.5*wl/denom; %TRF.PARAM.overload;
[mincoef,iyc] = min(yccoefs); %keyboard
yc = (trow-1)*MAC.YC.ycsprow + iyc;
```

```
function YC_assign_ct2(ct,operation)

global BAYS BL CT S TIME YC

%keyboard

% 1. First, determine the yc responsible of the CT
YC_select(ct);

% Make a copy of the CT position
yc = CT(ct).P.yc;
bay = CT(ct).P.bay;
BAY = BAYS(bay);
stack = CT(ct).P.stack;
tier = CT(ct).P.tier;
ct_weight = CT(ct).weight;

if strcmp(operation,'stack') == 1
   ciclo = YC_calc_stack(yc,ct,0); % = is not to calculate potential reshuffle
   CT(ct).events.stack = TIME.t + sum(ciclo.time);
   if CT(ct).events.arrival/3600/24 > 7 % Then we record the cycle to measure
      CT(ct).ciclo.stack = ciclo;
   end
elseif strcmp(operation,'deliver') == 1
   %keyboard
   ciclo = YC_calc_delivery(yc,ct);
   CT(ct).events.delivery = TIME.t + sum(ciclo.time);
   if CT(ct).events.arrival/3600/24 > 7 % Then we record the cycle to measure
      CT(ct).ciclo.delivery = ciclo; keyboard
   end
end

% Assign CT to the YC list
%keyboard
YC_addtask(yc,ct,sum(ciclo.time));

% Update the YC position
YC(yc).P.bay = CT(ct).P.bay;
YC(yc).P.stack = CT(ct).P.stack;
YC(yc).P.tier = BL.tiers +1;
```

```
function [C] = YC_calc_delivery(ct)
% This function calculates the energy and time of an ASC to make a move

global BL COST COUNT CT TIME YC

C = cycle_ini();
%keyboard
bay = CT(ct).P.bay;
tier = CT(ct).P.tier;
stack = CT(ct).P.stack;
yc = CT(ct).P.yc;

CT_addevent(ct,'requestD',0);

% 1. Crane current position is the start of the cycle
C.bay = YC(yc).P.bay; C.stack = YC(yc).P.stack; C.tier = YC(yc).P.tier;

% PART 1: CT PICKUP: Gantry to bay, Trolley to the stack, Lower to pickup and Lift CT
% ------------------------------------------------------------------------
C.bay = CT(ct).P.bay;
C.ct(1) = 0;  C.ctmove(1) = ct;
[C.time(1),C.E(1),C.moves{1},C.go] = YC_energy(YC(yc).P.bay,CT(ct).P.bay,0,'gantry','empty');

% calculate reshuffles and heights
[nr,nh,h] = BAYS_reshuffles(ct);
% COST.reshuffle.EXP = COST.reshuffle.EXP + nr;
% COST.r_heights.EXP = COST.r_heights.EXP + nh;

if nr > 0
   %keyboard
   if strcmp(CT(ct).type,'EXP')==1
      e = COUNT.R.EXP.events + 1;
      COUNT.R.EXP.events = e;
      COUNT.R.EXP.nr(e) = nr;
      COUNT.R.EXP.revent(e) = TIME.t;
      COUNT.R.EXP.sheight(e) = h;
   elseif strcmp(CT(ct).type,'IMP')==1
      e = COUNT.R.IMP.events + 1;
      COUNT.R.IMP.events = e;
      COUNT.R.IMP.nr(e) = nr;
      COUNT.R.IMP.revent(e) = TIME.t;
      COUNT.R.IMP.sheight(e) = h;
   end

   W = 0; % Set W = 1 to write the asc position changes

   [ETM] = BAY_reshuffles(yc,ct);

   C.time = [C.time,ETM.time];  C.E = [C.E,ETM.E];  C.moves = [C.moves,ETM.moves];
   C.bay = [C.bay,ETM.bay];  C.stack = [C.stack,ETM.stack];  C.tier = [C.tier,ETM.tier];
   C.ct = [C.ct, ETM.ct];  C.ctmove = [C.ctmove, ETM.ctmove];
end
```

```
[C,n] = addposition(C,'trolley',CT(ct).P.stack);  C.ct(n) = 0;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(YC(yc).P.stack,CT(ct).P.stack,0,'trolley','empty');

[C,n] = addposition(C,'hoist',CT(ct).P.tier); C.ct(n) = 0;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(BL.tiers+1,CT(ct).P.tier,0,'lower','empty');

[C,n] = addposition(C,'hoist',BL.tiers+1);  C.ct(n) = ct;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(CT(ct).P.tier,BL.tiers+1,ct,'pickbl','full');

% PART 2. CT DELIVERY
% ------------------------------------------------------------------------
[C,n] = addposition(C,'trolley',0); C.ct(n) = ct;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(CT(ct).P.stack,0,ct,'trolley','full');

[C,n] = addposition(C,'hoist',1); C.ct(n) = ct;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(BL.tiers+1,1,ct,'dropbf','full');

[C,n] = addposition(C,'hoist',BL.tiers+1); C.ct(n) = 0;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(1,BL.tiers+1,0,'raise','empty');

%  Compute the cost of the operation for ET and YT
ETYT_cost_calc(ct,'delivery');

% The last position of the YC
YC(yc).P.bay = CT(ct).P.bay; YC(yc).P.stack = 0; YC(yc).P.tier = BL.tiers + 1;


%disp(['YC(' num2str(yc) ') CT(' num2str(ct) ') Delivery cycle calculation' ])
%disp(['CTS: ' num2str(BL.GS(yc.ciclo.gs).cts) ' Stack/Delivery (Sea/Land)Reservations ' num2str(sr) ' / ' num2str(Sdr)
' / ' num2str(Ldr)])

YC_add_event(ct,sum(C.E),sum(C.time),'delivery');

YC(yc).active = 1;
```

```
function [C,R] = YC_calc_stack(ct,siono)
% This function calculates the energy and time of an ASC to make a move

global BAYS BL BT COST CT SPEED T TIME VS YC

[C,R] = cycle_ini();
%keyboard

bay = CT(ct).P.bay;
old_bay=BAYS(bay);
stack = CT(ct).P.stack;
tier = CT(ct).P.tier;
yc = CT(ct).P.yc;
vs = CT(ct).vs;

%  Compute the cost of the operation for YC and YT
traveltime = ETYT_cost_calc(ct,'stack');

% Before arriving to the bay where it will be stacked, it has to get there:
% If the ct is import, it will go to the bay inmediately
% if the ct is export, it will stay at the gate buffer
CT_addevent(ct,'bay',-traveltime);
CT_addevent(ct,'requestS',0);

%[t_tier] = Pile_height(bay,stack,'M')+1;
a = find(BAYS(bay).ct_id(:,stack)>0);
t_tier= length(a);

if t_tier ~= tier % The ct is not on the tier is should be
   keyboard
   if strcmp(CT(ct).type,'IMP')==1
      keyboard
   end
   disp(['CTs have arrived prior to CT(' num2str(ct) '). Recalculate position within the bay'])
   %disp(['YC(' num2str(yc) ') Workload: ' num2str(YC(yc).WL.n)])
   for s = 1:BL.stacks
      a = find(BAYS(bay).ct_id(:,s)>0);  altura_pila(s)= length(a);
      %a = find(BAYS(bay).port(:,s)>0);  h(s)= length(a);
      if altura_pila(s) == BL.tiers
         peso(s) = 100;
      else
         peso(s) = BL.idealbay(altura_pila(s)+1,s);
      end
   end
   [weight_dif,new_stack] = min(abs(peso - CT(ct).class));
   new_tier = Pile_height(bay,new_stack,'M') +1;
   %keyboard
   % Change port reservation if we are in a different stack
   if abs(new_stack -stack)> 0
      if altura_pila(stack)==0
         keyboard
      end
      BAYS(bay).port(h(stack),stack) = 0;
```

```
      BAYS(bay).port(h(new_stack)+1,new_stack) = 1;
      BAYS(bay).res.cts(new_tier,stack) = 0;
   else
      if BAYS(bay).res.cts(new_tier,stack) > 0
         BAYS(bay).res.cts(tier,stack) = BAYS(bay).res.cts(new_tier,stack);
      end
   end
   check_port_reservation(bay);
   CT(ct).P.stack = new_stack; CT(ct).P.tier = new_tier;
   BAYS(bay).res.cts(new_tier,new_stack) = ct;
end

% 1. Start cycle at crane position, but goes to CT position
C.bay = YC(yc).P.bay; C.stack = YC(yc).P.stack; C.tier = BL.tiers+1;

% PART 1: CT PICKUP: Gantry to bay, Trolley to the stack, Lower to pickup and Lift CT
% ------------------------------------------------------------------------
C.bay = CT(ct).P.bay; C.ct(1) = 0; C.ctmove(1) = ct;
[C.time(1),C.E(1),C.moves{1},C.go] = YC_energy(YC(yc).P.bay,CT(ct).P.bay,0,'gantry','empty');

[C,n] = addposition(C,'trolley',0);  C.ct(n) = 0; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(YC(yc).P.stack,0,0,'trolley','empty');
% Combine gantry and trolley
if C.time(n) < C.time(n-1)
   %keyboard
   C.time(n) = 0;
else
   C.time(n) = C.time(n) - C.time(n-1);
end

[C,n] = addposition(C,'hoist',1); C.ct(n) = 0; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(BL.tiers+1,1,0,'lower','empty');

[C,n] = addposition(C,'hoist',BL.tiers+1);  C.ct(n) = ct; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(1,BL.tiers+1,ct,'pick','full');

% PART 2. CT DELIVERY TO STACK

% ------------------------------------------------------------------------
[C,n] = addposition(C,'trolley',CT(ct).P.stack); C.ct(n) = ct; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(0,CT(ct).P.stack,ct,'trolley','full');

[C,n] = addposition(C,'hoist',CT(ct).P.tier); C.ct(n) = ct; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(BL.tiers+1,CT(ct).P.tier,ct,'dropbl','full');

[C,n] = addposition(C,'hoist',BL.tiers+1); C.ct(n) = 0; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = YC_energy(CT(ct).P.tier,BL.tiers+1,0,'raise','empty');

% 1.3. Calculate the potential reshuffles of containers underneath
if siono == 1
   %keyboard
   nr = CT(ct).P.tier - 1;
   if nr > 0
      % bay = BAY_copy(destiny.bay);
```

```
      keyboard
      [BAY] = BAY_copy(destiny.bay);
      BAY.time = max(TIME.t,asc.position.time(end));
      BAY.cts(destiny.tier,destiny.stack) = ct;
      BAY.slots(destiny.tier,destiny.stack) = 1;
      for r = 1:nr
         h = destiny.tier - r;
         nct = BL.GS(destiny.gs).cts(h);
         if h == 5
            keyboard
         end
         dift = (CT(ct).events.time(1)-CT(nct).events.time(1))/3600/24;
         dift = fix(dift)+1;    dift = min(dift,8);
         p(r) = PT(h,dift);
         [ETM,newBAY,sol] = BAY_reshuffles(asc,nct,BAY,0);
         if sol == 1
            ETM.E = ETM.E*p(r);
            if r == 1
               R = ETM;
            else
               %keyboard
               R.time = [R.time,ETM.time];  R.E = [R.E,ETM.E];  R.moves = [R.moves,ETM.moves];
            end
         else
            R.time = 1000000000;
            R.E = 1000000000000;
            %r = nr;
         end
      end
   end
end

YC(yc).P.bay = CT(ct).P.bay; YC(yc).P.stack = 0; YC(yc).P.tier = BL.tiers + 1;

YC_add_event(ct,sum(C.E),sum(C.time),'stack');

YC(yc).active = 1;

%disp(['CT(' num2str(ct) ') YC(' num2str(yc) ') Stack calculation with duration: ' num2str(sum(C.time)/60) ' min'])

% if sum(C.time)/60 > 3
%     disp('long cycle!')
%     plot_yc
%     xb = BAYS(bay).position(1); yb = BAYS(bay).position(2);
%     xc = BAYS(YC(yc).P.bay).position(1); yc = BAYS(YC(yc).P.bay).position(2);
%     plot(xb,yb,'bO')
%     plot(xc,yc,'b>')
%     pause(1)
% end

check_port_occupation(bay)
```

```
function [consumption,m] = YC_consumption(movement,load,ct)

global CT MAC

consumption = 0;
% Choose the weight of the cargo depending on the loading condition
if strcmp(load,'L') == 1
   ct_weight = CT(ct).weight;
else
   ct_weight = 0;
end

% Initilalize consumption coeficients
%cte_hoist = 2.591; cte_spreader = 2.438; cte_gantry = 6.058;

switch movement
   % Gantry
   case 'G'
      m = MAC.YC.crane.W + ct_weight;
      consumption = (MAC.YC.gantry.friction/10*m*9.81)/MAC.YC.gantry.motor.efficiency;
   % Spreader empty/loaded
   case 'T'
      m = MAC.YC.trolley.W + ct_weight;
      consumption = (MAC.YC.trolley.friction/10*m*9.81)/MAC.YC.trolley.motor.efficiency;
   % Hoist loaded/empty
   case 'H'
      m = MAC.YC.spreader.W + ct_weight;
      consumption = (m*9.81)/MAC.YC.hoist.motor.efficiency;
end
```

```
function YC_cycle(yc)

% This function analyzes the YC cycle and

global CT YC TIME

if YC(yc).active == 1
   YC(yc).nextevent = YC(yc).nextevent - TIME.delt;
   if YC(yc).nextevent <= 0
      ct = YC(yc).WL.ct; move = YC(yc).WL.move;
      if strcmp(move,'delivery') == 1
         CT_remove2(ct);
      elseif strcmp(move,'stack') == 1
         CT_stack(ct);
      end
      YC_wl_remove(ct);
   end
end

% The YC is inactive, check if there are CTs in the WL
if YC(yc).active == 0
   ct = 0;
   if YC(yc).WL.prior.n >0
      [ct,move] = YC_next_task(yc,YC(yc).WL.prior);
   end
   if and (ct == 0, YC(yc).WL.normal.n >0)
      %keyboard
      [ct,move] = YC_next_task(yc,YC(yc).WL.normal);
   end

   if ct>0
      %keyboard
      disp(['Next ct in the WL is ' char(move) ' CT(' num2str(ct) ')'])
      if strcmp(move,'stack') == 1
         %keyboard
         BAY_individual_allocation(ct);
         [C] = YC_calc_stack(ct,0); % 1 or 0 to calculate or not potential reshuffles
      elseif strcmp(move,'delivery') == 1
         %keyboard
         [C] = YC_calc_delivery(ct);
      end
      YC(yc).nextevent = sum(C.time);
      check_bay_es(CT(ct).P.bay);
   end
end
```

```
function [travelt,E1,move,C] = YC_energy(O,D,ct,type,load)
% This function calculates the time it takes to make a move from A to B
% A and B must be given in container indecees
global BAYS CT MAC S

E = 0;
move = type; C = 0; factor = 1;

if strcmp(type,'gantry') == 1
   movetype = 1;
elseif or(strcmp(type,'raise') == 1, strcmp(type,'uraise') == 1)
   movetype = 2;
elseif or(strcmp(type,'lower') == 1, strcmp(type,'ulower') == 1)
   movetype = 2; factor = 0;
elseif strcmp(type,'pick') == 1
   movetype = 2;
elseif or(strcmp(type,'pickbl') == 1, strcmp(type,'upickbl') == 1)
   movetype = 2;
elseif strcmp(type,'dropbf') == 1
   movetype = 2; factor = 0;
elseif or(strcmp(type,'dropbl') == 1, strcmp(type,'udropbl') == 1)
   movetype = 2; factor = 0;
elseif or(strcmp(type,'trolley') == 1, strcmp(type,'utrolley') == 1)
   movetype = 3;
end

if ct == 0
   ctw = 0;
else
   %keyboard
   ctw = CT(ct).weight;
end
switch movetype
   case 1 % GANTRY
      a = MAC.YC.move.gantry.accel;
      vmax = MAC.YC.move.gantry.speed.empty;
      vmin = MAC.YC.move.gantry.speed.full;
      s = vmax + ctw/65000*(vmin-vmax);
      d = abs(BAYS(D).position(1) - BAYS(O).position(1));   %abs(O-D)*S.l;
      [E,m] = YC_consumption('G',load,ct);
      motorrpm = MAC.YC.gantry.motor.rpm;
      %motorinertia = 91.2*m*(s/motorrpm)^2; %MAC.YC.gantry.motor.inertia;
      motorinertia = MAC.YC.gantry.motor.inertia;
      motoref = MAC.YC.gantry.motor.efficiency;
      no = MAC.YC.gantry.motor.no;
      xo = O*S.l;
   case 2 % SPREADER
      a = MAC.YC.move.hoist.accel;
      vmax = MAC.YC.move.hoist.speed.empty;
      vmin = MAC.YC.move.hoist.speed.full;
      s = vmax + ctw/65000*(vmin-vmax);
      d = abs(O-D)*S.h;
      [E,m] = YC_consumption('H',load,ct);
```

```
        % MODEL 2
        motorrpm = MAC.YC.spreader.motor.rpm;
        motorinertia = MAC.YC.spreader.motor.inertia;
        motoref = MAC.YC.spreader.motor.efficiency;
        no = MAC.YC.spreader.motor.no;

    case 3 % TROLLEY
        a = MAC.YC.move.troll.accel;
        s = MAC.YC.move.troll.speed;
        d = abs(O-D)*S.w;
        [E,m] = YC_consumption('T',load,ct);
        % MODEL 2
        motorrpm = MAC.YC.trolley.motor.rpm;
        motorinertia = MAC.YC.trolley.motor.inertia;
        motoref = MAC.YC.trolley.motor.efficiency;
        %keyboard
        c = MAC.YC.trolley.friction; % Friction coef
        v = MAC.YC.trolley.sheave;
        friction = m/1000*c;
        rope = (1-v^3)*(m-MAC.YC.W.cabin)/2;
        m = friction + rope;
        no = MAC.YC.trolley.motor.no;
end

% MODEL 1: POTENTIAL ENERGY
E1 = d*E*factor;

if a == 0
    t_ac = 0;
else
    t_ac= s/a;          % Acceleration period
end
d_ac = 0.5*a*t_ac^2;  % Distance needed for acceleration
d_fs = d -2*d_ac;     % Distance traveled at full speed
full_speed = 0;
if d_fs > 0 % Full speed achieved
    full_speed =1;
    % time at full speed
    t_fs = d_fs/s;
    travelt = t_fs + 2*t_ac;
else % Full speed not achieved
    d_ac = d/2;
    t_ac = (d_ac*2/a)^0.5;
    travelt = 2*t_ac;
end

% MODEL 2: CRANES
if factor == 1
    E_ct = 0.5*m*9.81*a^2*travelt^2/motoref;
    E_ac = 0.5*m*a^2*t_ac^2/motoref; %Accelerate and decelerate
else % Going down: only deceleration is considered
    %E_ct = 0.5*m*9.81*a^2*travelt^2/efficiency;
    E_ct = 0;
```

```
    E_ac = 0.5*m*a^2*t_ac^2/motoref;
end
E_mot = motorinertia*(2*3.141592*motorrpm)^2/t_ac;
E2 = (E_ac + E_ct + E_mot); %no*


% Generate gantry movement curve
travelt = ceil(travelt);
% keyboard
if travelt > 0
    if strcmp(type,'gantry') == 1
        C = YC_trajectory(a,s,travelt,full_speed);
    else
        C = 0;
    end
end
```

```
function [travelt,E,move,C] = YC_energy(O,D,ct,type,load)
% This function calculates the time it takes to make a move from A to B
% A and B must be given in container indecees
global BAYS CT EXEC S MAC

%keyboard

move = type; C = 0; factor = 1;

if or(strcmp(type,'gantry') == 1, strcmp(type,'ugantry') == 1)
   movetype = 1;
elseif strcmp(type,'trans') == 1
   movetype = 1;
elseif or(strcmp(type,'raise') == 1, strcmp(type,'uraise') == 1)
   movetype = 2;
elseif or(strcmp(type,'lower') == 1, strcmp(type,'ulower') == 1)
   movetype = 2; factor = -1;
elseif or(strcmp(type,'pickbf') == 1, strcmp(type,'pick') == 1)
   movetype = 2;
elseif or(strcmp(type,'pickbl') == 1, strcmp(type,'upickbl') == 1)
   movetype = 2;
elseif strcmp(type,'dropbf') == 1
   movetype = 2; factor = -1;
elseif or(strcmp(type,'dropbl') == 1, strcmp(type,'udropbl') == 1)
   movetype = 2; factor = -1;
elseif or(strcmp(type,'trolley') == 1, strcmp(type,'utrolley') == 1)
   movetype = 3;
else
   keyboard
end

if ct == 0
   ctw = 0;
else
   ctw = CT(ct).weight;
end


switch movetype
   case 1 % GANTRY
      a = MAC.YC.move.gantry.accel;
      d = MAC.YC.move.gantry.decel;
      vmax = MAC.YC.move.gantry.speed.empty;
      vmin = MAC.YC.move.gantry.speed.full;
      s = vmax + ctw/65000*(vmin-vmax);
      X = abs(BAYS(D).position(1) - BAYS(O).position(1));   %abs(O-D)*S.l; %
      [C,m] = YC_consumption('G',load,ct);
      xo = O*S.l;

      % MODEL 2 electric energy model
      W = MAC.YC.crane.W + ctw;
      rpm = MAC.YC.gantry.motor.rpm;
      inertia = MAC.YC.gantry.motor.inertia;
```

```
        efficiency = MAC.YC.gantry.motor.efficiency;
        F1 = W*MAC.YC.gantry.friction/1000; % in kN
        F2 = 0;%keyboard
    case 2 % HOIST
        a = MAC.YC.move.hoist.accel; d = a;
        vmax = MAC.YC.move.hoist.speed.empty;
        vmin = MAC.YC.move.hoist.speed.full;
        s = vmax + ctw/65000*(vmin-vmax);
        X = abs(O-D)*S.h;
        [C,m] = YC_consumption('H',load,ct);

        % MODEL 2
        W = MAC.YC.spreader.W + ctw;
        maxrpm = MAC.YC.hoist.motor.loaded.rpm;
        minrpm = MAC.YC.hoist.motor.unloaded.rpm;
        rpm = maxrpm + ctw/65000*(minrpm-maxrpm);
        inertia = MAC.YC.hoist.motor.inertia;
        efficiency = MAC.YC.hoist.motor.efficiency;
        F1 = W*9.81/1000; % in kN
        F2 = 0; %keyboard
    case 3 % TROLLEY
        a = MAC.YC.move.troll.accel; d = a;
        s = MAC.YC.move.troll.speed;
        X = abs(O-D)*S.w;
        [C,m] = YC_consumption('T',load,ct);

        % MODEL 2
        W = MAC.YC.trolley.W + ctw;
        rpm = MAC.YC.trolley.motor.rpm;
        inertia = MAC.YC.trolley.motor.inertia;
        efficiency = MAC.YC.trolley.motor.efficiency;
        F1 = W*MAC.YC.trolley.friction/1000;
        F2 = 3;%keyboard
%         c = YC.trolley.friction; % Friction coef
%         v = YC.trolley.sheave;
%         friction = m/1000*c;
%         % Main rope rigid load
%         rope = (1-v^3)*(m+YC.trolley.W)/2; % antes YC.W.cabin
%         m = friction + rope;
        %no = YC.trolley.motor.no;

end

% Time analysis of the movement
[x,t,full_speed,maxs] = travel_time(X,s,a,d);
travelt = ceil(t.total);

% Energy model
if t.total == 0
    E = 0;
else
    % MODEL 1: POTENTIAL + KINETIC
    % ------------------------------------------------------------------------
    if EXEC.energy_model == 1
```

```
    %keyboard
    E = X*C*factor/3600000; % From J to kWh
% MODEL 2: CRANE MOTOR BASED
% ------------------------------------------------------------------------
elseif EXEC.energy_model == 2

    P1 = F1 * s/efficiency;
    E1 = P1*t.total;

    P2 = F2 * s/efficiency;
    E2 = P2*t.total;

    E3 = 0;

    w = 2*3.141592*rpm/60;
    M4 = inertia*w/t.ac;
    P4 = M4*rpm/9550;
    E4 = P4*t.ac;

    F5 = W*s/t.ac/1000;
    P5 = F5 * s/efficiency;
    E5 = P5*t.ac;

    E = (E1+E2+E3+E4+E5)*factor/3600;

%     efficiency = 0.9;
%     if factor == 1
%         E_ct = 0.5*m*9.81*a^2*t.total^2/efficiency;
%         E_ac = 0.5*m*a^2*t.ac^2/efficiency; % Accelerate only
%     else % Going down: only deceleration is considered
%         %E_ct = 0.5*m*9.81*a^2*travelt^2/efficiency;
%         E_ct = 0;
%         E_ac = 0.5*m*a^2*t.ac^2/efficiency;
%     end
%     if t.ac > 0
%         E_mot = inertia*(2*3.141592*rpm)^2/t.ac;
%     else
%         E_mot = 0;
%     end
%    E2 = E_ac + E_ct + E_mot;
    %disp(['Movement ' type ' Relationship between E models ' num2str(E1/E2) ])
    % Generate gantry movement curve

    %keyboard
    if t.total > 0
        if strcmp(type,'gantry') == 1
            C = YC_trajectory(a,s,t.total,full_speed);
        else
            C = 0;
        end
    end

    %E = E2;
else
```

```
        disp('Wrong energy model'); keyboard
    end
end
```

```
function YC_init()

global BAYS BL MAC S T TRF YC

% 2. MACHINERY PARAMETERS
MAC.YC.ycsprow = 7;

MAC.YC.move.gantry.speed.full = 90/60; %240 before
MAC.YC.move.gantry.speed.empty = 135/60; %240 before
MAC.YC.move.gantry.accel = 0.40;
MAC.YC.move.gantry.decel = 0.40;

MAC.YC.move.hoist.speed.full = 31/60;
MAC.YC.move.hoist.speed.empty = 62/60;
MAC.YC.move.hoist.accel = 0.35;

MAC.YC.move.troll.speed = 70/60;
MAC.YC.move.troll.accel = 0.40;

MAC.YC.W.crane = 116000-16000;
MAC.YC.W.spreader = 2000;
MAC.YC.W.cabin = 16000;

MAC.YC.crane.W = 116000;
MAC.YC.spreader.W = 2000;
MAC.YC.trolley.W = MAC.YC.W.spreader + 16000;

MAC.YC.gantry.motor.no = 6;
MAC.YC.gantry.motor.rpm = 1467/60;
MAC.YC.gantry.motor.inertia = 0.5;
MAC.YC.gantry.motor.efficiency = 0.95;
MAC.YC.gantry.friction = 0.05; % Friction coef in kg/ton

MAC.YC.trolley.motor.no = 2;
MAC.YC.trolley.motor.rpm = 1350/60; % rpm to rps
MAC.YC.trolley.motor.inertia = 20.66; % From catalogue, pedro vila gives 12.48; % kgm^2
MAC.YC.trolley.friction = 0.06; % Friction coef in (kg/t)
MAC.YC.trolley.sheave = 0.985;
MAC.YC.trolley.motor.efficiency = 0.90; %

MAC.YC.hoist.motor.no = 1;
MAC.YC.hoist.motor.loaded.rpm = 900/60; % rpm to rps
MAC.YC.hoist.motor.unloaded.rpm = 1800/60; % rpm to rps
MAC.YC.hoist.motor.inertia = 110; % kgm^2
MAC.YC.hoist.motor.efficiency = 0.85; %

ycsprow = MAC.YC.ycsprow;
MAC.YC.n = ycsprow*T.rows;

for row = 1:T.rows

    bloques = BL.rowlist(row,:);
    b=0;
```

```
  for j = 1:length(bloques)
     blo = bloques(j);
     for k = 1:BL.bays
        b=b+1;
        bahias(b) = BL.baylist(blo,k);
     end
  end

  x=zeros(1,ycsprow);
  for i=1:ycsprow
     yc = (row-1)*ycsprow + i;
     YC(yc).id = yc;
     YC(yc).row = row;
     YC(yc).col = i;
     %YC(yc).list_time = 0;
     YC(yc).spreader = 0;
     % Generate a random block
     col = round(random('unif',1,T.cols));
     % Generate a random bay within the block
     %keyboard
     ind = rem(i,T.rows);
     if ind == 0
        ind = T.rows;
     end

     bbay = round(random('unif',1,BL.bays));
     %bay = BL.baylist(bloque,bbay);
     x(i) = i*T.length/(ycsprow+1);
     md = 100000;

     for j=1:length(bahias)
        d=abs(BAYS(bahias(j)).position(1)-x(i));
        if abs(d)<md
           md = d;
           t_bay(i) = bahias(j);
        end
     end
     %keyboard
     % Generate initial position for the YC
     YC(yc).P.bay = t_bay(i); %T.aisles.sides.width+(BL.length+T.aisles.vertical.width)*(col-1)+(bay-1)*S.l;
     YC(yc).P.tier = BL.tiers + 1;
     YC(yc).P.stack = 0;
     %=T.aisles.bottom.width-T.aisles.horizontal.width+(BL.width+T.aisles.horizontal.width)*row;
     WL.cts = [];
     WL.time = [];
     WL.n = 0;
     WL.move = '';
     WL.cwl = 0;
     %WL.delivery=zeros(1,TRF.PARAM.VS_no);
     YC(yc).WL = WL;
%       YC(yc).WL.prior = WL;
%       YC(yc).WL.normal = WL;
     YC(yc).nextevent = 1000000000;
     YC(yc).events.n = 0;
```

```
        YC(yc).events.cts = 0;
        YC(yc).events.time = 0;
        YC(yc).events.E = 0;
        YC(yc).events.duration = 0;
        YC(yc).active = 0;
    end
end
```

```
function [l_x,r_x] = YC_limit_bays(yc_o)

global BAYS BL CT MAC YC

%keyboard
bay = YC(yc_o).P.bay;
trow = BAYS(bay).row;

if YC(yc_o).WL.prior.n> 0
   t_ct = YC(yc_o).WL.prior.cts(1);
   t_ct_bay = CT(t_ct).P.bay;
   t_ct_x = BAYS(t_ct_bay).position(1);
elseif YC(yc_o).WL.normal.n> 0
   t_ct = YC(yc_o).WL.normal.cts(1);
   t_ct_bay = CT(t_ct).P.bay;
   t_ct_x = BAYS(t_ct_bay).position(1);
else
   t_ct_x = 1000000;
end

if YC(yc_o).col < MAC.YC.ycsprow
   r_yc = yc_o+1;
   % The Limiting Right position is the min between the RC bay and the next ct
   % in the workload
   if YC(r_yc).WL.prior.n> 0
      r_ct = YC(r_yc).WL.prior.cts(1);
      r_ct_bay = CT(r_ct).P.bay;
      r_ct_x = BAYS(r_ct_bay).position(1);
   elseif YC(r_yc).WL.normal.n> 0
      r_ct = YC(r_yc).WL.normal.cts(1);
      r_ct_bay = CT(r_ct).P.bay;
      r_ct_x = BAYS(r_ct_bay).position(1);
   else
      r_ct_x = 1000000;
   end
   r_yc_bay = YC(r_yc).P.bay;
   r_yc_x = BAYS(r_yc_bay).position(1);
else
   r_yc_x = 1000000;
   r_ct_x = 1000000; %t_ct_x;
end

r_x = min(r_ct_x,r_yc_x);

if YC(yc_o).col > 1
   l_yc = yc_o-1;
   if YC(l_yc).WL.prior.n> 0
      l_ct = YC(l_yc).WL.prior.cts(1);
      l_ct_bay = CT(l_ct).P.bay;
      l_ct_x = BAYS(l_ct_bay).position(1);
   elseif YC(l_yc).WL.normal.n> 0
      l_ct = YC(l_yc).WL.normal.cts(1);
      l_ct_bay = CT(l_ct).P.bay;
```

```
        l_ct_x = BAYS(l_ct_bay).position(1);
    else
        l_ct_x = 0;
    end
    l_yc_bay = YC(l_yc).P.bay;
    l_yc_x = BAYS(l_yc_bay).position(1);
else
    l_yc_x = 0;
    l_ct_x = 0; %t_ct_x;
end

l_x = max(l_ct_x,l_yc_x);
```

```
function [ct,move] = YC_next_task(yc,WL)

global BAYS BT CT MAC YC

ct = 0; move='';
strategy = 'WIPE';
ycbay = YC(yc).P.bay;
% if yc ==1
%     keyboard
% end
[l_x,r_x] = YC_limit_bays(yc);

if strcmp(strategy,'FIFO') == 1
    ct = WL.cts(1);
    time = WL.time(1);
    %YC(yc).WL.bay(no) = BAYS(bay).position(1);
elseif strcmp(strategy,'WIPE') == 1
    c = 0; ctlist = 0; x=0;
    % check the active vessels
    for i=1:3
        btvs(i)=BT(i).vessel;
    end
    % See the list of cts of the yc
    for i = 1:WL.n
        ict = WL.cts(i);
        if strcmp(CT(ict).type,'EXP')==1
            if strcmp(move,'delivery')==1
                if isempty(find(btvs == CT(ict).vs))
                    continue % Disregard stacking ct to non present vessels yet
                end
            end
        end
        ctbay = CT(ict).P.bay;
        x(i) = BAYS(ctbay).position(1);
        if and(l_x<=x(i), x(i)<=r_x)
            c = c+1;
            ctlist(c) = BAYS(ctbay).position(1)-BAYS(ycbay).position(1);
        end
    end
    % If there are cts in the list
    if c==0
         %keyboard
    else
        poslist = 0 ; neglist = 0;
        poslist = find(ctlist >=0);
        neglist = find(ctlist <0);

        % now we can have several possibilities
        % 1. Only CTs to the right:
        if and(isempty(poslist) == 0, isempty(neglist) == 1)
            YC(yc).WL.dir = 1;
            % 2. Only CTs to the left: move to the left
        elseif and(isempty(poslist) == 1, isempty(neglist) == 0)
```

```
        YC(yc).WL.dir = -1;
        % 3. CTs on both sides
    elseif and(isempty(poslist) == 0, isempty(neglist) == 0)
        % The crane will keep moving on the previous direction, do nothing
        if YC(yc).WL.dir == 0
            [mindist,pos] = min(abs(ctlist));
            ct = WL.cts(pos); ctbay = CT(ct).P.bay;
            if BAYS(ctbay).position(1)-BAYS(ycbay).position(1)>=0
                YC(yc).WL.dir = 1;
            else
                YC(yc).WL.dir = -1;
            end
        end
        % 4. No CTs on any side
    else
        disp('Crane selection error')
        keyboard
    end

    % Once determined the direction of movement, pick the next ct in the list
    if YC(yc).WL.dir == 0
        disp('Crane has no CTs in the WL list')
        YC(yc).active = 0;
        keyboard
    else
        if YC(yc).WL.dir == 1
            [mindist,ict] = min(poslist);
        elseif YC(yc).WL.dir == -1
            [mindist,ict] = min(neglist);
        end
        ct = WL.cts(ict);
        move = WL.move(ict);
        YC(yc).active = 1;
        YC(yc).WL.ct = ct;
        YC(yc).WL.move = move;
    end
    check_bay_es(CT(ct).P.bay);
  end
end
```

```
function [YC]=YC_task(YC)


% 2.1 Assign YC
for yc=1:no_YC
    % Criteria 1: Measure the distance to the slot
    %--------------------------------------------
    yc_dist(yc)=dist_slot_yc(YC,slot);


    % Criteria 2: Queue of works of each the YC

    %--------------------------------------------
end
% Evaluate the YC with weights
```

```
function [C] = ASC_trajectory(a,s,travelt,full_speed)
% This function calculates the trajectory of a crane

NP = 100;
C.t = travelt/NP:travelt/NP:travelt;

if full_speed == 1
   i = 1; v = 0;
   C.x(1) = 0.5*a*C.t(i)^2;

   while v < s
      i = i+1;
      C.x(i) = 0.5*a*C.t(i)^2;
      v = a*C.t(i);
      k = NP-i+1;
   end
   for j = i+1:NP-i
      C.x(j) = C.x(i) + s*(C.t(j)-C.t(i));
   end

   for i = j +1: NP
      t = C.t(i)-C.t(j);
      C.x(i) = C.x(j)+s*t-0.5*a*t^2;
   end
else
   for i=1:NP/2
      C.x(i) = 0.5*a*C.t(i)^2;
      C.x(NP-i+1) = C.x(i);
   end
end

%keyboard
% figure; plot(C.t,C.x)
```

```
function YC_wl_remove(ct)

global BAYS BL CT YC

keyboard
yc = CT(ct).P.yc;
bay = CT(ct).P.bay;
check_bay_es(bay);

pos1 = find(YC(yc).WL.prior.cts == ct);
pos2 = find(YC(yc).WL.normal.cts == ct);
if isempty(pos1) == 0
   %keyboard
   YC(yc).WL.prior.cts(pos1)  = [];
   YC(yc).WL.prior.time(pos1) = [];
   YC(yc).WL.prior.move(pos1) = [];
   YC(yc).WL.prior.n = YC(yc).WL.prior.n -1;
elseif isempty(pos2) == 0
   %keyboard
   YC(yc).WL.normal.cts(pos2)  = [];
   YC(yc).WL.normal.time(pos2) = [];
   YC(yc).WL.normal.move(pos2) = [];
   YC(yc).WL.normal.n = YC(yc).WL.normal.n -1;
else
   disp(['YC(' num2str(yc) ') WL removal error CT(' num2str(ct) ')'])
   keyboard
end

% Remove CT from crane
YC(yc).WL.ct = 0;
YC(yc).WL.move = '';
YC(yc).active = 0;
YC(yc).nextevent = 1000000;
```

Annex C

Perpendicular Terminal model code

The Matlab code for the Perpendicular Terminal DES Model used throughout this Thesis is provided here.

```
function [P,n] = addposition(P,move,nP)

% copy previous position

m = length(P.bay); n = m+1;
P.bay(n) = P.bay(m);
P.stack(n) = P.stack(m);
P.tier(n) = P.tier(m);
P.time(n) = P.time(m);


if strcmp(move,'gantry') == 1
   P.bay(n) = nP;
elseif strcmp(move,'ugantry') == 1
   P.bay(n) = nP;
elseif strcmp(move,'trolley') == 1
   P.stack(n) = nP;
elseif strcmp(move,'utrolley') == 1
   P.stack(n) = nP;
elseif strcmp(move,'hoist') == 1
   P.tier(n) = nP;
elseif strcmp(move,'uhoist') == 1
   P.tier(n) = nP;
else
   keyboard
end
```

```
function [pos]= ASC_act_pos(asc)
% This funciton returns the las position of the ASC

pos.bay = asc.position.bay(end);
pos.stack =asc.position.stack(end);
pos.tier =asc.position.tier(end);
pos.time = asc.position.time(end);
```

```
function [no_cts] = ASC_add_list_housekeeping(ctlist,flow,priority)

global ASC BL COUNT CT TIME

% Determine the amount of CTs to remove

%keyboard

no_cts = length(ctlist);

disp('------------------------------------------------------------------')
disp([flow  ' HOUSEKEEPING: ' num2str(no_cts)  ' cts added'])
disp('------------------------------------------------------------------')

% plot_ASC_tasks(ASC.sea);

if  no_cts >0
   for ict = 1: no_cts
      ct = ctlist(ict);
      %ASC_wl(asc,ct);
      t_gs = CT(ct).position.gs(end);
      %last_event = length(CT(ct).events.time);
      v_ct(ict) = ct;              % asc.tasks.ct(current_task + ict)
      v_ct_id{ict} = CT(ct).id;     % asc.tasks.ct_id{current_task + ict}
      v_time(ict) = TIME.t + ict; % asc.tasks.time(current_task + ict)
      v_action{ict} = 'housekeeping';   % asc.tasks.action{current_task + ict}
   end

   % Add the queue
   %keyboard
   ASC.hktasks.ct = v_ct;
   ASC.hktasks.ct_id = v_ct_id;
   ASC.hktasks.time = v_time;
   ASC.hktasks.action = v_action;
   ASC.hktasks.priority = priority;
   %ASC.housekeeping = 1;
else
   disp('Add list housekeeping error'); keyboard
end
```

```
function [no_exp_ct] = ASC_add_list_tasks(ctlist)

global ASC BL COUNT CT TIME TRF SEA_DELIVERY

% Determine the amount of CTs to remove

factor = TRF.PARAM.perc_imp; %0.8; %TRF.PARAM.rand(COUNT.VS+1);
no_exp_ct = fix(factor*length(ctlist));

disp('----------------------------------------------------------------------')
disp(['SEA DELIVERY: ' num2str(no_exp_ct) ' cts added to ASC sea WL'])
disp('----------------------------------------------------------------------')

plot_ASC_tasks(ASC.sea);

keyboard
% need to redo the list of tasks is made

current_task = 1;
final_task = length(ASC.sea.tasks.ct);

% Trozo 1
v_ct1 =     ASC.sea.tasks.ct(1:current_task);
v_ct_id1 =  ASC.sea.tasks.ct_id(1:current_task);
v_time1 =   ASC.sea.tasks.time(1:current_task);
v_action1 = ASC.sea.tasks.action(1:current_task);

% Trozo 2
queue = final_task - current_task;
if  and(queue> 0,current_task >0)
   %keyboard
   v_ct2 =     ASC.sea.tasks.ct(current_task+1:final_task);
   %v_executed =     ASC.sea.tasks.executed(current_task+1:final_task);
   v_ct_id2 =  ASC.sea.tasks.ct_id(current_task+1:final_task);
   v_time2 =   ASC.sea.tasks.time(current_task+1:final_task);
   v_action2 = ASC.sea.tasks.action(current_task+1:final_task);
end

if current_task>0
   new_time = ASC.sea.tasks.time(current_task)+1;
else
   new_time= TIME.t;
end

% Trozo 3
cont=0;
for ict = 1: no_exp_ct
   cont= cont+1;
   ct = ctlist(ict);
   CT(ct).position.vs = COUNT.VS+1;
   SEA_DELIVERY(COUNT.VS+1).ct(cont)=ct;

   [res,listapos] = ASC_wl(ASC.sea,ct);
```

```
    if res > 1
        CT(ct).events
        keyboard
        %plot_BL
    end
    t_gs = CT(ct).position.gs(end);
    BL.GS(t_gs).Sdelreservations = BL.GS(t_gs).Sdelreservations + 1; % Delivery reservation
    %last_event = length(CT(ct).events.time);
    v_ct3(cont) = ct;               % ASC.sea.tasks.ct(current_task + ict)
    v_ct_id3{cont} = CT(ct).id;     % ASC.sea.tasks.ct_id{current_task + ict}
    v_time3(cont) = new_time + ict; % ASC.sea.tasks.time(current_task + ict)
    v_action3{cont} = 'delivery';   % ASC.sea.tasks.action{current_task + ict}
end

% Add the queue
%keyboard
if  and(queue> 0,current_task >0)
    ASC.sea.tasks.ct = [v_ct1, v_ct2, v_ct3];
    ASC.sea.tasks.ct_id = [v_ct_id1, v_ct_id2, v_ct_id3];
    ASC.sea.tasks.time =[v_time1,v_time2,v_time3 ];
    ASC.sea.tasks.action = [v_action1, v_action2, v_action3];
else
    ASC.sea.tasks.ct = [v_ct1, v_ct3];
    ASC.sea.tasks.ct_id = [v_ct_id1, v_ct_id3];
    ASC.sea.tasks.time =[v_time1, v_time3 ];
    ASC.sea.tasks.action = [v_action1, v_action3];
end

%keyboard
plot_ASC_tasks(ASC.sea);

if length(ASC.sea.tasks.ct_id) ~=  length(ASC.sea.tasks.ct)
    keyboard
    disp('Error in the task')
end
```

```
function [asc] = ASC_addcycle(asc,NP,delay,ct)
% This function ads a new translation cycle to the idle asc

global TIME

%keyboard

asc.status = 'trans';
asc.nextevent = ceil(delay);

[asc.ciclo] = ASC_cycle_ini();

asc.ciclo.bay = NP.bay;
asc.ciclo.stack = NP.stack;
asc.ciclo.tier = NP.tier;
asc.ciclo.time = delay;
asc.ciclo.basetime = TIME.t;
move = 'trans';
asc.ciclo.moves{1} = move;
asc.ciclo.ctmove = ct;
asc.ciclo.originaltime = delay;

asc.ciclo.origin = ASC_act_pos(asc);
asc.ciclo.destiny = NP;
```

```
function [asc] = ASC_adddelay(asc,task,delay,baytarget,guilty_ct)
% This function adds a delay to the crane cycle
global BL TIME

%keyboard
if abs(length(asc.ciclo.ctmove) - length(asc.ciclo.moves)) > 0
   keyboard
end
t = task;
t_fin = length(asc.ciclo.bay);
found = 0;
while found == 0
   if t==t_fin
      itask=t_fin;found = 1;
   end
   if strcmp(asc.ciclo.moves(t),'gantry') == 1
      itask = t; found = 1;
   end
   t = t+1;
end

gantrycommenced = 0;
if asc.ciclo.originaltime(itask)-asc.ciclo.time(itask) > 0
   gantrycommenced = 1;
   %keyboard
end

ct = asc.ciclo.ct(itask);
tier = asc.ciclo.tier(itask);
stack = asc.ciclo.stack(itask);
if itask > length(asc.ciclo.ctmove)
   keyboard
else
   move = asc.ciclo.ctmove(itask);
end

% Crane position
ascpos = ASC_act_pos(asc);

if baytarget > 0
   %keyboard
   bay = baytarget;
   itask = asc.ciclo.c_task;
   [delay,E] = ASC_energy(ascpos.bay,baytarget,0,'gantry','empty');
else % baytarget = 0
   if or(ascpos.bay > BL.bays, ascpos.bay < 0)
      bay = ascpos.bay;
      itask = asc.ciclo.c_task;
   else
      if itask ==1
         putafter=0; putbefore=0;
         % if the cycle has started
         dir =  asc.ciclo.bay(itask)- ascpos.bay;
```

```
        % if the cranes goes upwards, after the first gantry
        if strcmp(asc.id,'land') == 1
            a= 1;
        elseif strcmp(asc.id,'sea') == 1
            a=-1;
        end
        if a*dir >0
            putafter = 1;
        else
            putbefore=1;
        end
        if putafter == 1;
            bay = asc.ciclo.bay(itask);
            itask = itask +1;
            gantrycommenced = 0;
        elseif putbefore == 1 % crane has to stop here
            bay = ascpos.bay;
        end
    else % itask > 1
        ascpos = ASC_act_pos(asc);
        bay = ascpos.bay;
    end
  end
end

if gantrycommenced == 0
  asc.ciclo.bay = [asc.ciclo.bay(1:itask-1), bay, asc.ciclo.bay(itask:end)];
  asc.ciclo.stack = [asc.ciclo.stack(1:itask-1), stack, asc.ciclo.stack(itask:end)];
  asc.ciclo.tier = [asc.ciclo.tier(1:itask-1), tier, asc.ciclo.tier(itask:end)];
  asc.ciclo.time = [asc.ciclo.time(1:itask-1), delay,asc.ciclo.time(itask:end)];
  asc.ciclo.E = [asc.ciclo.E(1:itask-1), 0, asc.ciclo.E(itask:end)];
  asc.ciclo.moves = [asc.ciclo.moves(1:itask-1), 'wait', asc.ciclo.moves(itask:end)];
  asc.ciclo.ctmove = [asc.ciclo.ctmove(1:itask-1), guilty_ct, asc.ciclo.ctmove(itask:end)];
  asc.ciclo.ct = [asc.ciclo.ct(1:itask-1), ct, asc.ciclo.ct(itask:end)];
  asc.ciclo.originaltime = [asc.ciclo.originaltime(1:itask-1), delay, asc.ciclo.originaltime(itask:end)];
else
%    if asc.ciclo.nr >0
%        keyboard
%    end
  % Modify the gantry in progress
%    if itask ==1
%        keyboard
%    end

%    if asc.ciclo.originaltime(itask)-asc.ciclo.time(itask) > 0
%        if and(strcmp(asc.id,'land')==1,asc.ciclo.bay(itask)<BL.bays+3)
%            itask = itask +1;
%        elseif and(strcmp(asc.id,'sea')==1,asc.ciclo.bay(itask)>-2)
%            itask = itask +1;
%        end
%    end
  timeleft = asc.ciclo.time(itask);
  timepassed = asc.ciclo.originaltime(itask)-asc.ciclo.time(itask);
  t_bay = asc.ciclo.bay(itask);
```

```
    asc.ciclo.bay (itask) = bay;
    asc.ciclo.time(itask) = 0;
    asc.ciclo.originaltime(itask) = timepassed;

    % Add the wait
    itask = itask+1;
    asc.ciclo.bay = [asc.ciclo.bay(1:itask-1), bay, asc.ciclo.bay(itask:end)];
    asc.ciclo.stack = [asc.ciclo.stack(1:itask-1), stack, asc.ciclo.stack(itask:end)];
    asc.ciclo.tier = [asc.ciclo.tier(1:itask-1), tier, asc.ciclo.tier(itask:end)];
    asc.ciclo.time = [asc.ciclo.time(1:itask-1), delay,asc.ciclo.time(itask:end)];
    asc.ciclo.E = [asc.ciclo.E(1:itask-1), 0, asc.ciclo.E(itask:end)];
    asc.ciclo.moves = [asc.ciclo.moves(1:itask-1), 'wait', asc.ciclo.moves(itask:end)];
    asc.ciclo.ctmove = [asc.ciclo.ctmove(1:itask-1), guilty_ct, asc.ciclo.ctmove(itask:end)];
    asc.ciclo.ct = [asc.ciclo.ct(1:itask-1), ct, asc.ciclo.ct(itask:end)];
    asc.ciclo.originaltime = [asc.ciclo.originaltime(1:itask-1), delay, asc.ciclo.originaltime(itask:end)];
    % Add a new gantry to target
    itask = itask +1;
    asc.ciclo.bay = [asc.ciclo.bay(1:itask-1), t_bay, asc.ciclo.bay(itask:end)];
    asc.ciclo.stack = [asc.ciclo.stack(1:itask-1), stack, asc.ciclo.stack(itask:end)];
    asc.ciclo.tier = [asc.ciclo.tier(1:itask-1), tier, asc.ciclo.tier(itask:end)];
    asc.ciclo.time = [asc.ciclo.time(1:itask-1), timeleft,asc.ciclo.time(itask:end)];
    asc.ciclo.E = [asc.ciclo.E(1:itask-1), 0, asc.ciclo.E(itask:end)];
    asc.ciclo.moves = [asc.ciclo.moves(1:itask-1), 'gantry', asc.ciclo.moves(itask:end)];
    asc.ciclo.ctmove = [asc.ciclo.ctmove(1:itask-1), ct, asc.ciclo.ctmove(itask:end)];
    asc.ciclo.ct = [asc.ciclo.ct(1:itask-1), ct, asc.ciclo.ct(itask:end)];
    asc.ciclo.originaltime = [asc.ciclo.originaltime(1:itask-1), timeleft, asc.ciclo.originaltime(itask:end)];

end
v=AUX_vect_ac(asc.ciclo.time) + TIME.t;
plot(v/3600/24,asc.ciclo.bay,'.m');
asc.nextevent = asc.nextevent + delay;
if abs(length(asc.ciclo.ctmove) - length(asc.ciclo.moves)) > 0
    keyboard
end
```

```
function [asc] = ASC_addtask(asc,ct,action,priority)
% This function places the ct on the position of the asc WL list
% - pos = 0; Put in in the last position
% - pos = number, put it in that position

global COUNT CT TIME

if length(action) == 0
   keyboard
end

if strcmp(asc.id,'sea')
   COUNT.ASC.sea.tasks = COUNT.ASC.sea.tasks + 1;
elseif strcmp(asc.id,'land')
   COUNT.ASC.land.tasks = COUNT.ASC.land.tasks + 1;
end

% Determine the position of the task
if priority <= 0; % The new task has no priority
   task_no = length(asc.tasks.ct) + 1;
else % The new task has priority
   task_no = priority;
end

% Determine the id of the ct in case of traslation due to crane interf
if ct == 0
   newid = 'TRS'; %keyboard
   %disp('Watch addpriortask, criteria has changed to include prior task when ASC has no tasks')
else
   newid = CT(ct).id;
end


% Check workload for repeated CTs
if ct > 0
   ASC_wl(asc,ct);
end

% Place the task on its position
asc.tasks.action = vect_insert_char(asc.tasks.action, action, task_no);
asc.tasks.ct_id  = vect_insert_char(asc.tasks.ct_id, newid,task_no);
asc.tasks.ct     = vect_insert(asc.tasks.ct,ct,task_no);

if priority == -1 % sea delivery
   delt = 0;
else
   delt = TIME.t + TIME.delt;
end
asc.tasks.time   = vect_insert(asc.tasks.time, delt, task_no);

% TRANS has always priority over the remaninder of tasks
if strcmp(action,'trans') == 1
   if asc.tasks.current > 1
```

```
        keyboard
    else
        asc.tasks.current = 1;
    end
else
    if task_no <= asc.tasks.current
        asc.tasks.current = asc.tasks.current + 1; %keyboard
    end
end

%disp(['ASC ' num2str(asc.id) ': adding ct ' num2str(ct) ' to WL task ' num2str(task_no)])
```

```
function [asc] = ASC_advance(asc,task)
% This function makes a partial progress the yard crane along the block

global TIME

%keyboard

ct = asc.ciclo.ct(task);

% Compute how much cycle has not been completed yet
% in time
t0 = asc.ciclo.originaltime(task); % Time complete
t1 = asc.ciclo.time(task);    % Time left
%keyboard
tconsumed = t0-t1;
if TIME.delt >= tconsumed
   timeleft = t0;
   tadv = tconsumed;
else
   timeleft = TIME.delt + t1;
   tadv = TIME.delt;
end

prog = (tadv)/timeleft;
ascpos= ASC_act_pos(asc);

% In space (bays)
leftdelbay = asc.ciclo.bay(task) - ascpos.bay; % Final - initial
delbay = fix(100*leftdelbay*prog)/100;

newpos = ascpos;
newpos.bay = newpos.bay + delbay;
newpos.time = TIME.t + TIME.delt ; %newpos.time + (t0- t1);

[asc] = ASC_move(asc,newpos,ct,1);
```

```
function [btime] = ASC_basetime(asc)
% this function calculates the base time of a cycle

global TIME


% 1. No active cycle
if strcmp(asc.status,'wait') == 1
   btime = TIME.t; % +inct;

% 2. Cycle is active
else
   % Check if more time hs passed til the beginning of the cycle
   if asc.ciclo.no == 0
      btime = TIME.t; %+inct;
   else
      btime = asc.ciclo.basetime(1);
   end
end
```

```
function [ini_gs, end_gs, del_gs] = ASC_bay_direction(asc)
% This function gives the direction of the ASC
global BAYS BL CT EXEC TIME TRF

if strcmp(TRF.PARAM.stackmode, 'tercat') ==1
   seabay = BL.hklimit.sea; %10; % BL.baylimit.sea; % = 7;
   landbay = BL.hklimit.land;  %31; % BL.baylimit.land; % = 34;
elseif strcmp(TRF.PARAM.stackmode, 'psrandom') ==1
   seabay = 1; %10; % BL.baylimit.sea; % = 7;
   landbay = BL.bays;  %31; % BL.baylimit.land; % = 34;
else
   seabay = 1; %seabay = BL.hklimit.sea; %
   landbay = BL.bays; %BL.hklimit.land; %
end

sea_gs = BAYS(seabay).GS(1);
land_gs = BAYS(landbay).GS(end);
%keyboard

if TIME.t < EXEC.cutofftime
   if strcmp(asc.id,'land') == 1 % From sea to land
      ini_gs = 1; end_gs = land_gs; del_gs = 1;
   elseif strcmp(asc.id,'sea') == 1 % From land to sea
      ini_gs = BL.no_gs; end_gs = sea_gs; del_gs = -1;
   end
elseif asc.housekeeping == 1
   ct = asc.hkct; %keyboard
   if strcmp(CT(ct).id,'EXP') == 1
      del_gs = 1;
      if strcmp(TRF.PARAM.stackmode, 'essa') == 1
         bayjump = 6; %keyboard
      else
         bayjump = 1; %keyboard
      end
      end_gs = min(land_gs,BAYS(CT(ct).position.bay(end)-bayjump).GS(end));
      if strcmp(asc.id,'land')
         ini_gs = sea_gs;
      elseif strcmp(asc.id,'sea')
         ini_gs = 1;
      end
   elseif strcmp(CT(ct).id,'IMP') == 1
      del_gs = -1;
      if strcmp(TRF.PARAM.stackmode, 'essa') == 1
         bayjump = 6; %keyboard
      else
         bayjump = 1; %keyboard
      end
      end_gs = max(sea_gs,BAYS(CT(ct).position.bay(end)+bayjump).GS(1)); %keyboard
      if strcmp(asc.id,'land')
         ini_gs = BL.no_gs;
      elseif strcmp(asc.id,'sea')
         ini_gs = land_gs;
      end
```

```
    end
elseif strcmp(asc.id,'land') == 1
    %caso = 'S2L';
    del_gs = 1; ini_gs = sea_gs; end_gs = BL.no_gs;%end_gs = land_gs; %
elseif strcmp(asc.id,'sea') == 1
    %caso = 'L2S';
    del_gs = -1;
%     if strcmp(TRF.PARAM.stackmode,'tercat') == 1
%         ini_gs = land_gs+3; %BL.bays; %
%     else
        ini_gs = land_gs;
%     end
    end_gs = sea_gs;
     %end_gs = 1;
end

% if strcmp(caso,'S2L') == 1 % From sea to land
%     ini_gs = 1; end_gs = BL.no_gs; del_gs = 1;
% elseif strcmp(caso,'L2S') == 1 % From land to sea
%     ini_gs = BL.no_gs; end_gs = 1; del_gs = -1;
% end
```

```
function [asc] = ASC_calc_delivery(asc,ct)
% This function calculates the energy and time of an ASC to make a move

global BL COUNT CT LIMITS TIME

[C,R] = ASC_cycle_ini();

% original CT
C.originalct = ct;
% 1. Crane position
P = ASC_act_pos(asc); C.ascposition =  ASC_act_pos(asc);

% PART 1: CT PICKUP: Gantry to bay, Trolley to the stack, Lower to pickup and Lift CT
% ---------------------------------------------------------------------
origin = CT_act_pos(ct);    C.origin = origin;

pile = PILE_copy(asc.id,CT(ct).id);

[destiny] = PILE_LOCATION(pile); C.destiny = destiny;

C.bay = P.bay; C.stack = P.stack; C.tier = P.tier;
C.bay = origin.bay; C.ct(1) = 0;  C.ctmove(1) = ct;
[C.time(1),C.E(1),C.moves{1},C.go] = ASC_energy(P.bay,origin.bay,0,'gantry','empty',0);

% 1.3. Calculate the potential reshuffles
if origin.gs ==0
   keyboard
end

%keyboard
% Calculates the emtpy and stack reserved slots in the bay
nr = BL.GS(origin.gs).ocup - origin.tier;

%if strcmp(asc.id,'land') == 1
if nr > 0
   [BAY] = BAY_copy(origin.bay);
   pos = CT_act_pos(ct);
   [fslots,rslots] =  BAY_ES(pos.bay,pos.stack);

   if sum(fslots) < nr
      %keyboard
      BAY.cts
      disp('BAY Reshuffles warning: No space in bay for Reshuffles. CTs should be moved to other bays');
      seed = 1; keyboard
   else
   % Count the number of reshuffles
      if strcmp(CT(ct).id,'EXP') == 1
         e = COUNT.RE.events + 1;
         COUNT.RE.events = e;
         COUNT.RE.nr(e) = nr;
         COUNT.RE.revent(e) = TIME.t;
         COUNT.RE.sheight(e) = BL.GS(origin.gs).ocup;
      else
```

```
        e = COUNT.RI.events + 1;
        COUNT.RI.events = e;
        COUNT.RI.nr(e) = nr;
        COUNT.RI.revent(e) = TIME.t;
        COUNT.RI.sheight(e) = BL.GS(origin.gs).ocup;
      end

      BAY.time = max(TIME.t,asc.position.time(end));
      W = 1; % Set W = 1 to account for the number of reshuffles
      %keyboard
      [ETM,newBAY] = BAY_reshuffles(asc,ct,BAY,W);

      C.time = [C.time,ETM.time];  C.E = [C.E,ETM.E];  C.moves = [C.moves,ETM.moves];
      C.bay = [C.bay,ETM.bay];  C.stack = [C.stack,ETM.stack];  C.tier = [C.tier,ETM.tier];
      C.ct = [C.ct, ETM.ct];  C.ctmove = [C.ctmove, ETM.ctmove];
   end
end

P.bay = C.bay(end); P.stack = C.stack(end); P.tier = C.tier(end);

[C,n] = addposition(C,'trolley',origin.stack);  C.ct(n) = 0;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(P.stack,origin.stack,0,'trolley','empty',0);

[C,n] = addposition(C,'hoist',origin.tier); C.ct(n) = 0;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(BL.tiers+1,origin.tier,0,'lower','empty',0);

[C,n] = addposition(C,'hoist',BL.tiers+1);  C.ct(n) = ct;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(origin.tier,BL.tiers+1,ct,'pickbl','full',0);

% PART 2. CT DELIVERY TO Buffer
% ------------------------------------------------------------------------

[C,n] = addposition(C,'gantry',destiny.bay); C.ct(n) = ct;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n},C.back] = ASC_energy(origin.bay,destiny.bay,ct,'gantry','full',0);

[C,n] = addposition(C,'trolley',destiny.stack); C.ct(n) = ct;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(origin.stack,destiny.stack,ct,'trolley','full',0);

[C,n] = addposition(C,'hoist',destiny.tier); C.ct(n) = ct;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(BL.tiers+1,destiny.tier,ct,'dropbf','full',0);

[C,n] = addposition(C,'hoist',BL.tiers+1); C.ct(n) = 0;  C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(destiny.tier,BL.tiers+1,0,'raise','empty',0);

%keyboard
asc.ciclo = C;
asc.ciclo.nr = nr;
asc.ciclo.origin = origin;
asc.ciclo.destiny = destiny;
asc.nextevent = sum(asc.ciclo.time);
asc.status = 'delivery';
asc.ciclo.originaltime = C.time;

CT(ct).delivery.moves = C.moves;
```

```
% Establish limits to the other crane
if strcmp(asc.id,'sea') == 1
   LIMITS.land.bay = destiny.bay;
   LIMITS.land.ct = ct;
end

%keyboard
Ldr = BL.GS(origin.gs).Ldelreservations;
Sdr = BL.GS(origin.gs).Sdelreservations;
if strcmp(asc.id,'land') == 1
   BL.GS(origin.gs).Ldelreservations = BL.GS(origin.gs).Ldelreservations + 1;
   Ldr = BL.GS(origin.gs).Ldelreservations;
   if Sdr > 0
      keyboard
   end
elseif  strcmp(asc.id,'sea') == 1
   BL.GS(origin.gs).Sdelreservations = BL.GS(origin.gs).Sdelreservations + 1;
   Sdr = BL.GS(origin.gs).Sdelreservations;
   if Ldr > 0
      keyboard
   end
end

sr = BL.GS(origin.gs).sreservations;
disp(['ASC(' asc.id ') CT(' num2str(ct) ') Delivery calculation. Time2complete: ' num2str(asc.nextevent)])
disp(['Origin: Bay/GS (' num2str(origin.bay) '/' num2str(origin.gs) ') Destiny: Bay/GS (' num2str(destiny.bay) '/'
num2str(destiny.gs) ')'])
disp(['Pile cts [' num2str(BL.GS(asc.ciclo.origin.gs).cts) '] Stack/Delivery (Sea/Land)Reservations ' num2str(sr) ' / '
num2str(Sdr) ' / ' num2str(Ldr)])

% if and(strcmp(asc.id,'sea')==1, origin.tier < BL.GS(origin.gs).ocup)
%     disp('There are export reshuffles')
%     keyboard
% end

%keyboard
[asc.tasks] = ASC_task_remove(asc.tasks,asc.tasks.current);

%ASC_check_bay(asc)
```

```
function [asc,found] = ASC_calc_stack(asc,ct,tasktype)
% This function calculates the energy and time of an ASC to make a move

global ASC CT TRF TIME

%stopatct(ct,5706)

% 1. Search
% Try to find a solution with same group of containers
%if strcmp(CT(ct).id,'IMP') == 1
   CT_dwell();
%end

% ------------------- TERCAT SEARCH -------------------------------------
if strcmp(TRF.PARAM.stackmode,'tercat') == 1
% -----------------------------------------------------------------------
   if TIME.t/3600/24 > 10
      [t_gs,asc,found] = TERCAT_search_drop(ct,asc,'calm');
%        if and(found == 0, asc.housekeeping == 0)
%            [t_gs,asc,found] = PSRANDOM_search_drop(ct,asc);
%        end
   else
      %keyboard
       [t_gs,asc,found] = PSRANDOM_search_drop(ct,asc);
   end
% ------------------- ESSA ----------------------------------------------
elseif strcmp(TRF.PARAM.stackmode,'essa') == 1
% -----------------------------------------------------------------------

   if TIME.t/3600/24 > 10
      [t_gs,asc,found] = ESSA_search_drop(ct,asc,'calm');
%        if and(found == 0, asc.housekeeping == 0)
%            keyboard
%            [t_gs,asc,found] = PSRANDOM_search_drop(ct,asc);
%        end
   else
      [t_gs,asc,found] = PSRANDOM_search_drop(ct,asc);
   end
% ------------------- RANDOM --------------------------------------------
elseif strcmp(TRF.PARAM.stackmode,'psrandom') == 1
   [t_gs,asc,found] = PSRANDOM_search_drop(ct,asc);
elseif strcmp(TRF.PARAM.stackmode,'crandom') == 1
   if TIME.t/3600/24 > 10
      [t_gs,asc,found] = CRANDOM_search_drop(ct,asc);
   else %if asc.housekeeping == 0
      [t_gs,asc,found] = PSRANDOM_search_drop(ct,asc);
   end
else
   disp('Error: wrong stackmode name')
   keyboard
end

%keyboard
```

```
if found == 0
   disp(['Warning: no slot found to ' asc.status ' '  CT(ct).id  ' CT(' num2str(ct) ')'])
   if and(strcmp(CT(ct).id,'EXP') == 1, asc.housekeeping ==0)
      plot_BL
      keyboard
   end
else
   % Write the ASC
   [asc] = ASC_write_cycle(ct,asc,t_gs,tasktype);
   CT(ct).hk_asc = asc.id; %keyboard
   if strcmp(asc.status,'housekeeping') == 1
      %[ASC.hktasks] = ASC_task_remove(ASC.hktasks,ASC.hktasks.current);%
      %keyboard
%        if CT(ct).priority == 2
%           if strcmp(CT(ct).id,'IMP') == 1
%              i = find(ASC.hktasks.prior.IMP == ct);
%              ASC.hktasks.prior.IMP(i) = [];
%           elseif strcmp(CT(ct).id,'EXP') == 1
%              i = find(ASC.hktasks.prior.EXP == ct);
%              ASC.hktasks.prior.EXP(i) = [];
%           end
%        elseif CT(ct).priority == 1
%           if strcmp(CT(ct).id,'IMP') == 1
%              i = find(ASC.hktasks.nonprior.IMP == ct);
%              ASC.hktasks.nonprior.IMP(i) = [];
%           elseif strcmp(CT(ct).id,'EXP') == 1
%              i = find(ASC.hktasks.nonprior.EXP == ct);
%              ASC.hktasks.nonprior.EXP(i) = [];
%           end
%        end
   else
      [asc.tasks] = ASC_task_remove(asc.tasks,asc.tasks.current);
   end
end
```

```
function [asc] = ASC_catch(asc)

global TIME

%keyboard

Pos = ASC_act_pos(asc);

if and(strcmp(asc.status,'wait') == 1, Pos.time < TIME.t + TIME.delt)
    Pos.time = TIME.t+TIME.delt;
    [asc] = ASC_move(asc,Pos,0,1);
    %asc.status = 'wait';
end
```

```
function ASC_check_bay(asc)

global BF

if min(asc.ciclo.bay) < BF.sea.exp.bay
   disp('Bay calculation error')
   keyboard
end

if max(asc.ciclo.bay) > BF.land.imp.bay
   disp('Bay calculation error')
   keyboard
end
```

```
function [rep] = ASC_check_repeated_wl(asc,Cslots)


L = length(asc.tasks.ct);

for i = 1:length(Cslots)
    ct = Cslots(i);
    rep(1,i) = 0; rep(2,i) = 0;
    if ct > 0
        for ict = 1:L
            if asc.tasks.ct(ict) == ct;
                rep(1,i) = ct;
                rep(2,i) = rep(2,i) + 1;
            end
        end
    end
    if rep(2,i) > 1
        disp(['ASC workload error: CT: ' num2str(ct) ' repeated '])
        keyboard
    end
end

figure; plot(rep(1,:),rep(2,:),'.')
```

```
function [asc] = ASC_check_wl(asc)
% This function checks the Workload of the ASC to provide the next task

global ASC BL CT EXEC TIME TRF

% Check wether there is a list of tasks yet (que) to be done
if strcmp(asc.status,'wait') == 1
   %keyboard
   queue = length(asc.tasks.ct);

   if queue == 0
      do_hk = 1;
      exp_list = length(ASC.hktasks.prior.EXP);
      imp_list = length(ASC.hktasks.prior.IMP);
      hkqueue = exp_list + imp_list;
   elseif queue > 0 % There are tasks left, so we check if they are feasible
      % Determine the first task that are due from this moment on
      found = 0; asc.tasks.current = 0; its = 0; hkqueue = 0;

      while and(found == 0,asc.tasks.current < length(asc.tasks.ct))
         its = its + 1;
         [asc,ct,action,solution] = ASC_next_ct(asc,asc.tasks.current+1);

         if solution == 0
            break
         elseif solution == 1
            asc.housekeeping = 0;
            switch char(action)
               case 'delivery'
                  [asc] = ASC_calc_delivery(asc,ct); found = 1;
               case 'stack'
                  [asc,found] = ASC_calc_stack(asc,ct,'stack');
            end
         end
      end

      % if we carry out a task, reset housekeeping
      if found == 1
         do_hk = 0;
         ASC_tasks_init();
      else
         do_hk = 1;
      end
   end

   if and(do_hk ==1, TIME.day > 10)
      %keyboard
      % a) If there are no hktasks from before, calculate them
      % -----------------------------------------------------
      if or(hkqueue == 0, TIME.t - EXEC.lasthksearch >30*60)
         %keyboard
         SEARCH_Housekeeping3(); %keyboard
      end
```

```
% b) If there are tasks to do
% -----------------------------------------------------
% Now get the list depending on the hour of the day
ct_list = HK_get_list();

% For tercat, order the list randomly
if strcmp(TRF.PARAM.stackmode, 'tercat') == 1
   indlist = randperm(length(ct_list));  % generate a random sequence
   ct_list = ct_list(indlist);
% elseif strcmp(TRF.PARAM.stackmode, 'essa') == 1
%    keyboard
end

% c) Analyze the HK workload to find a feasible movement
% -----------------------------------------------------
ctask = 0; t_found = 0;
while and(ctask < length(ct_list), t_found == 0)
   ctask = ctask + 1;
   hkct = ct_list(ctask);
   % Verify this ct does not belong to the other crane
   if strcmp(asc.id,'land')==1
      if ASC.sea.hkct == hkct
         continue
      end
   elseif strcmp(asc.id,'sea')==1
      if ASC.land.hkct == hkct
         continue
      end
   end
   conflict = BAY_prevent_delivery_conflict(hkct,asc);
   if conflict == 1
      %keyboard
      continue
   end
   gs = CT(hkct).position.gs(end);
   if gs == 0
      CT(hkct).position
      disp(['Error in retrieving ' CT(hkct).id ' CT(' num2str(hkct) ') with priority ' num2str(CT(hkct).priority)]);
      keyboard
   elseif BL.GS(gs).sreservations == 0
      %disp(['length of the HK task list: ' num2str(hkqueue) ' list ' num2str(asc.hktasks.ct)])
      asc.hkct = hkct;
      asc.housekeeping = 1;
      [asc, t_found] = ASC_calc_stack(asc,hkct,'housekeeping');
      if t_found == 0
         asc.hkct = 0;
         asc.housekeeping = 0; %keyboard
      end

   end
   if ctask > 9
      disp('terminate search for cts')
      break
```

```
        end
      end

   end

   if  and(queue ==0, hkqueue == 0)
      asc.nextevent = 1000000;
      asc.status = 'wait'; keyboard
   end
end
```

```
function [consumption,m] = ASC_consumption(movement,loading,ct)
% this function calculates the consumption per meter of advance for the
% potential model. Output units are kg m/s^2

global ASC CT

G = 9.81;

% Choose the weight of the cargo depending on the loading condition
if ct  == 0
   ct_weight = 0;
else
   if strcmp(loading,'full') == 1
      ct_weight = CT(ct).weight;
   elseif strcmp(loading,'empty') == 1
      ct_weight = 0;
   else
      disp('CT weight error')
      keyboard
   end
end

% Initilalize consumption coeficients
%cte_hoist = 2.591; cte_spreader = 2.438; cte_gantry = 6.058;

switch char(movement)
   % Gantry
   case 'G'
      consumption = (ASC.gantry.friction*(ASC.crane.W + ct_weight)*G)/ASC.gantry.motor.efficiency/10;
      m = ASC.crane.W + ct_weight;
   % Spreader empty/loaded
   case 'T'
      consumption = (ASC.trolley.friction*(ASC.trolley.W + ct_weight)*G)/ASC.trolley.motor.efficiency/10;
      m = ASC.trolley.W + ct_weight;
   % Hoist loaded/empty
   case 'H'
      consumption = ((ASC.spreader.W + ct_weight)*G)/ASC.hoist.motor.efficiency;
      m = ASC.spreader.W + ct_weight;
end
```

```
function [asc] = ASC_copy(crane)

global ASC

switch char(crane)
    case 'sea'
        asc = ASC.sea;
    case 'land'
        asc = ASC.land;
end
```

```
function [asc] = ASC_cycle(asc)

global  BAYS BL CT TIME

% First of all, check whether reservations have been made in the meanwhile
% ------------------------------------------------------------------------
o_ct = asc.ciclo.originalct;
% if strcmp(asc.status, 'delivery')==1
% stopatct(o_ct,1529)
% end
% if strcmp(asc.status, 'housekeeping')==1
% stopatct(o_ct,2286)
% end
% Determine the position of the CT
% ------------------------------------------------------------------------
asc.ciclo.no = asc.ciclo.no + 1;
%disp(['ASC(' asc.id ') ' asc.status ' cycle CT( ' num2str(nct) ') step ' num2str(asc.ciclo.no) ' Current task '
num2str(asc.tasks.current)])

% Check ct noASC_main
ASC_wl(asc,o_ct); % Check for repeated WL

% Check whether the GS have been occupied due to reshuffles
% ------------------------------------------------------------------------
if or(strcmp(asc.status, 'stack') == 1, strcmp(asc.status, 'housekeeping') == 1)
   t_gs = asc.ciclo.destiny.gs;
   if and(t_gs > 0, t_gs <= BL.no_gs)
      if BL.GS(t_gs).ocup == BL.tiers
         if asc.ciclo.no == 1
            keyboard
            % Search another slot
            [found,asc,t_gs] = TERCAT_search_drop(o_ct,asc);
            % Calculate the cycle again
            [asc] = ASC_calc_stack(asc,ct);
            asc.ciclo.no = asc.ciclo.no + 1;
            task_no = asc.tasks.current;
         end
      end
   end
end
% ------------------------------------------------------------------------
t_tasks = asc.ciclo.time;
t_tasks_ac = AUX_vect_ac(t_tasks);
t_spent = 0;

asc.ciclo.basetime(asc.ciclo.no) = TIME.t;

% 1. Determine the initial and final task of the cycle
% ------------------------------------------------------------------------
a = find(t_tasks_ac > 0); i_task = a(1); %asc.ciclo.c_task;
%a = find(t_left_ac <= TIME.delt); f_task = a(end);

% 2. Start the movements cycle
```

```
% ------------------------------------------------------------------------
task = i_task; go = 1;
while and(t_spent < TIME.delt, go == 1)
   % A task is partially carried out
   t_left = TIME.delt - t_tasks_ac(task);
   ct = asc.ciclo.ct(task);
   movement = asc.ciclo.moves(task);
   if t_left < 0% Not enough time to complete the task
      t_pass = t_tasks(task) + t_left; %keyboard
      t_spent = t_spent + t_pass;
      asc.ciclo.time(task) = asc.ciclo.time(task) - t_pass;
      if strcmp(movement,'gantry') == 1
         seed = 1;
      elseif strcmp(movement,'ugantry') == 1
         seed = 1;
      elseif strcmp(movement,'trans') == 1
         seed = 1;
      else
         seed = 0;
      end
      if seed == 1
         prog = t_pass/t_tasks(task);
         if prog >1
            keyboard
         end
         leftdelbay = asc.ciclo.bay(task) - asc.position.bay(end); % Final - initial
         delbay = fix(10000*leftdelbay*prog)/10000;
         ascPos.bay = asc.position.bay(end) + delbay;
      else
         ascPos.bay = asc.ciclo.bay(task);
      end
   else % There is time to complete the task

      t_spent = t_tasks_ac(task);
      if strcmp(movement, 'pickbf') == 1 % ASC removes the CT
         BF_removeCT(ct,asc.id);
      elseif or(strcmp(movement, 'pickbl') == 1, strcmp(movement, 'upickbl') == 1)
         octpos.bay = asc.ciclo.bay(task-1);
         octpos.stack = asc.ciclo.stack(task-1);
         octpos.tier = asc.ciclo.tier(task-1);
         if octpos.tier > BL.tiers
            keyboard
         end
         octpos.gs = BAYS(octpos.bay).GS(octpos.stack);
         if o_ct == ct
            desreserve = 1;
            asc.ciclo.ctpicked = 1;
            %stopatct(ct,47)
            HK_ct_priority(ct);
         else
            desreserve = 0;
         end
         BL_removeCT(octpos,ct,desreserve,asc.status);
         %[errores] = AUX_check_bays(CT(nct).id);
```

```
      elseif or(strcmp(movement, 'dropbf') == 1, strcmp(movement, 'udropbf') == 1)
         [pile] = PILE_copy(asc.id,CT(ct).id);
         BF_put_ct(pile,ct);
         BF_removeCT(ct,asc.id);
         VS_check_complete(ct,sum(asc.ciclo.time));
         %keyboard
         HK_ct_priority(ct);
      elseif or(strcmp(movement, 'dropbl') == 1, strcmp(movement, 'udropbl') == 1)
         tstack = asc.ciclo.stack(task);
         tbay = asc.ciclo.bay(task);
         tgs = BAYS(tbay).GS(tstack);
         if and(o_ct == ct, strcmp(movement, 'dropbl') == 1)
            desreserve = 1;
         else
            desreserve = 0;
         end
         BL_dropCT(tgs,ct,desreserve,asc);
         if desreserve == 1 %and(, strcmp(CT(ct).id,'IMP') == 1)
            %keyboard
            HK_ct_priority(ct); % THe priority might have changed
         end
      end
      ascPos.bay = asc.ciclo.bay(task);
      asc.ciclo.time(task) = 0;
   end

   % find the new position of the crane

   ascPos.tier = asc.ciclo.tier(task);
   ascPos.stack = asc.ciclo.stack(task);
   if ct > 0
      ctpos = CT_act_pos(ct);
      ascPos.gs = ctpos.gs;
   end
   ascPos.time = TIME.t + t_spent; %asc.ciclo.basetime auxtime
   [asc] = ASC_move(asc,ascPos,ct,1);

   task = task + 1;
   if strcmp(asc.status,'trans') == 1
      go = 0;
   elseif task > length(asc.ciclo.ct)
      go = 0; %keyboard
   end
end

t_restante = sum(asc.ciclo.time);

% 3. Check the completeness of the cycle
% ---------------------------------------------------------------------
% 3.1. Cycle finished
if t_restante <= 0
   %keyboard
   % disp(['ASC ' num2str(asc.id) ' Stack CT ' num2str(nct) ' in GS: ' num2str(destiny.gs)])
```

```
% Check the bay occupation
% ---------------------------------------------------------------------
tbay = asc.ciclo.destiny.bay;
if and(tbay > 0, tbay < BL.bays + 1)
   bocup = BAY_occupation(tbay);
   if and(bocup == 0, o_ct > 0)
      if strcmp(asc.ciclo.moves(1),'trans')==0
         keyboard
      end
   end
end

% Write the CT cycle
% ---------------------------------------------------------------------
if o_ct >0 % This is needed to avoid empty moves (trans)
   %CT_energy_sum(asc,nct);

   for t = 1:length(asc.ciclo.ctmove)
      f_ct = asc.ciclo.ctmove(t);
      time = asc.ciclo.originaltime(t);
      e = asc.ciclo.E(t);
      move = asc.ciclo.moves{t};
      if f_ct > 0
         if strcmp(move,'wait') == 1
            %keyboard
            CT_write_cycle(o_ct,e,move,time);
         else
            %keyboard
            CT_write_cycle(f_ct,e,move,time);
         end
      else
         CT_write_cycle(o_ct,e,move,time);
      end

   end
   %CT_check_UnMoves(nct,asc.ciclo.moves)
end

% Write the executed tasks to the list
% ---------------------------------------------------------------------
asc.extasks.ct(end+1) = o_ct;
asc.extasks.action{end+1} = asc.status;
asc.extasks.time(end+1) = sum(asc.ciclo.originaltime);
asc.extasks.type{end +1} = asc.status;

% Reset the ASC (task = 0 indicates removing the current task)
% ---------------------------------------------------------------------
%    if strcmp(asc.status,'housekeeping') == 1
%       keyboard
%       [ASC.hktasks] = ASC_task_remove(ASC.hktasks,0);
%       asc.housekeeping = 0;
%    else
%       [asc.tasks] = ASC_task_remove(asc.tasks,asc.tasks.current);
%    end
```

```
   disp(['ASC(' asc.id ') CT (' num2str(o_ct) ') Iteration (' num2str(asc.ciclo.no) ') Task completed in ' num2str(t_spent) '
secs' ]);

   % Reset the ASC cicle and status
   % ---------------------------------------------------------------------
   asc.nextevent = 1000000;
   asc.status = 'wait';
   asc.hkct = 0;
   %asc.housekeeping = 0;
   asc.ciclo = ASC_cycle_ini();

% 3.2. Cycle not finished
%--------------------------------------------------------------------------
elseif t_restante > 0

   % 3.2.1 One one task to be partially completed
%    if f_task == 0
%       asc.ciclo.time(i_task) = asc.ciclo.time(i_task) - TIME.delt;
%
%       %if or(strcmp(move,'gantry') == 1,strcmp(move,'trans') == 1)
%       [asc] = ASC_advance(asc,i_task);
%       %end
%    else
%       asc.ciclo.c_task = task+1;
%       asc.ciclo.time(task+1) = t_left_ac(task+1) - TIME.delt;
%       %keyboard
%       [asc] = ASC_advance(asc,task+1);
%    end

   tc_prog = t_tasks_ac(end) - t_restante;
   %inc_t = tc_prog - t_spent;

   disp(['ASC(' asc.id ') CT(' num2str(o_ct) ') Iteration (' num2str(asc.ciclo.no) ') Ini/Progress/Time left: '
num2str(asc.nextevent) ' / ' num2str(t_spent) ' / ' num2str(t_restante) ' secs']);
   asc.nextevent = asc.nextevent - TIME.delt;

   if abs(sum(asc.ciclo.time)-asc.nextevent) > 0.01
      keyboard
      asc.nextevent = sum(asc.ciclo.time);
   elseif asc.nextevent == 0
      keyboard
   end

      % Reset the ASC. task = 0 removes the current task
%    if abs(asc.tasks.ct(asc.tasks.current) - asc.ciclo.ctmove(end)) >0
%       if strcmp(asc.tasks.action,'trans')== 0
%          asc.tasks
%          keyboard
%       end
%    end
end
```

```
function [asc] = ASC_cycle_check(asc)

global ASC BL

%keyboard
if strcmp(asc.status,'trans')== 0
   t_gs = asc.ciclo.destiny.gs;
%     if or(t_gs == 0, t_gs > BL.no_gs)
%         keyboard
%     end
   task_no = asc.tasks.current;
   if isempty(ASC.hktasks.action)
      hktask_no = 0;
   else
      hktask_no = 1; %asc.hktasks.current;
   end

   if task_no> 0
      nct = asc.tasks.ct(task_no);
   elseif hktask_no > 0
      nct = ASC.hktasks.ct(hktask_no);
   else
      nct = -1;
   end

   if nct >= 0
      if strcmp(asc.id,'sea')== 1
         othercrane = 'land';
      else
         othercrane = 'sea';
      end
      if strcmp(asc.status,'delivery') == 1
         if BL.GS(asc.ciclo.origin.gs).sreservations > 0
            %keyboard
            disp(['The ' char(asc.id) ' crane wants to deliver CT(' num2str(nct) ') @ GS(' num2str(asc.ciclo.origin.gs) ')
reserved for the ' char(othercrane) ' ASC'])
            %keyboard
            %[asc] = ASC_calc_delivery(asc,nct);
         end
      elseif or(strcmp(asc.status,'stack') == 1, strcmp(asc.status,'housekeeping') == 1)
         if or(BL.GS(t_gs).Sdelreservations > 0, BL.GS(t_gs).Ldelreservations > 0)
            disp(['The ' char(asc.id) ' crane ' char(asc.status) ' wants to stack a CT and the ' char(othercrane) ' has
reservation'])
            %keyboard
            %[asc] = ASC_calc_stack(asc,nct);
         end
      end
   else
      disp('ASC cycle check error'); keyboard
   end
end
```

```
function [C,R] = ASC_cycle_ini()
% This fucntion initializes the ASC cycle
global TIME

% Reshuffle part
R.time = 0;
R.E = 0;
R.moves = '';
R.bay=0;
R.stack = 0;
R.tier =0;
R.ct = 0;
R.ctmove = 0;

% Cycle part
C.bay = 0; C.stack = 0; C.tier = 0;
C.ct = 0; C.originalct = 0; C.ctmove = 0; C.ctpicked = 0;
C.E = 0; C.moves = '';
C.time = 0; C.originaltime = 0; C.basetime = TIME.t;
p.gs = []; p.bay = []; p.tier = []; p.stack = [];
C.origin = p; C.destiny = p;
C.go = 0; C.back =0;
C.nr = 0; C.no = 0; C.gs = 0;
C.c_task = 1;
C.ascposition = 0;
```

```
function [asc] = ASC_cycle_insert(asc,bay,stack,tier,delay,E,move,guilty_ct,ct,priority)
% This fucntion inserts a unit in the ASC cicle

%keyboard

% Determine the position of the task
if priority <= 0; % The new task has no priority
   task_no = length(asc.tasks.ct) +1;
else % The new task has priority
   task_no = priority;
end

asc.ciclo.bay   = vect_insert(asc.ciclo.bay, bay, task_no);
asc.ciclo.stack = vect_insert(asc.ciclo.stack, stack, task_no);
asc.ciclo.tier  = vect_insert(asc.ciclo.tier, tier, task_no);
asc.ciclo.time  = vect_insert(asc.ciclo.time, delay, task_no);
asc.ciclo.E     = vect_insert(asc.ciclo.E, E, task_no);
asc.ciclo.moves = vect_insert_char(asc.ciclo.moves, move, task_no);
if strcmp(move,'wait')
   newct = guilty_ct; % This works
elseif asc.tasks.current == 0
   newct = asc.ciclo.originalct; % keyboard
else
   newct = asc.tasks.ct(asc.tasks.current); %keyboard
end
asc.ciclo.ctmove = vect_insert(asc.ciclo.ctmove, newct , task_no);
asc.ciclo.ct    = vect_insert(asc.ciclo.ct, ct , task_no);
asc.ciclo.originaltime = vect_insert(asc.ciclo.originaltime, delay, task_no);
```

```
function [travelt,E,move,C] = ASC_energy(O,D,ct,type,load,calcsiono)
% This function calculates the time it takes to make a move from A to B
% A and B must be given in container indecees
global ASC CT EXEC S

%keyboard

move = type; C = 0; factor = 1; inct = 0;

if or(strcmp(type,'gantry') == 1, strcmp(type,'ugantry') == 1)
   movetype = 1;
elseif strcmp(type,'trans') == 1
   movetype = 1;
elseif or(strcmp(type,'raise') == 1, strcmp(type,'uraise') == 1)
   movetype = 2;
elseif or(strcmp(type,'lower') == 1, strcmp(type,'ulower') == 1)
   movetype = 2; factor = -1;
   if calcsiono == 1 % then we are estimating ESSA algorithm
      inct = 10;
   else
      inct = poissrnd(10*10)/10; % sea buffer
   end
elseif strcmp(type,'pickbf') == 1
   movetype = 2;
   if strcmp(CT(ct).id,'IMP')==1 % SEA side Transfer area
      if calcsiono == 1 % then we are estimating ESSA algorithm
         inct = 10;
      else
         inct = poissrnd(10*10)/10; % sea buffer
      end
   elseif strcmp(CT(ct).id,'EXP')==1 % LAND side Transfer Area
      if calcsiono == 1 % then we are estimating ESSA algorithm
         inct = 30;
      else
         inct = poissrnd(30*10)/10; % land buffer
      end
   end
elseif or(strcmp(type,'pickbl') == 1, strcmp(type,'upickbl') == 1)
   movetype = 2;
elseif strcmp(type,'dropbf') == 1
   movetype = 2; factor = -1;
   if strcmp(CT(ct).id,'IMP')==1
      if calcsiono == 1 % then we are estimating ESSA algorithm
         inct = 30;
      else
         inct = poissrnd(30*10)/10; % land buffer
      end
   elseif strcmp(CT(ct).id,'EXP')==1
      if calcsiono == 1 % then we are estimating ESSA algorithm
         inct = 10;
      else
         inct = poissrnd(10*10)/10; % sea buffer
      end
```

```
   end
elseif or(strcmp(type,'dropbl') == 1, strcmp(type,'udropbl') == 1)
   movetype = 2; factor = -1;
   if calcsiono == 1 % then we are estimating ESSA algorithm
      inct = 7;
   else
      inct = poissrnd(7*10)/10;
   end
elseif or(strcmp(type,'trolley') == 1, strcmp(type,'utrolley') == 1)
   movetype = 3;
else
   keyboard
end

if ct == 0
   ctw = 0;
else
   ctw = CT(ct).weight;
end


switch movetype
   case 1 % GANTRY
      a = ASC.move.gantry.accel;
      d = ASC.move.gantry.decel;
      vmax = ASC.move.gantry.speed.empty;
      vmin = ASC.move.gantry.speed.full;
      s = vmax + ctw/65000*(vmin-vmax);
      X = abs(O-D)*S.l; %abs(BAYS(D).position(1) - BAYS(O).position(1));   %
      [E,m] = ASC_consumption('G',load,ct);
      xo = O*S.l;

      % MODEL 2 electric energy model
      W = ASC.crane.W + ctw;
      rpm = ASC.gantry.motor.rpm;
      inertia = ASC.gantry.motor.inertia;
      efficiency = ASC.gantry.motor.efficiency;
      F1 = W*ASC.gantry.friction/1000; % F1 in kN, W in kg, frict coef in kN/ton
      F2 = 0;%keyboard
   case 2 % HOIST
      a = ASC.move.hoist.accel; d = a;
      vmax = ASC.move.hoist.speed.empty;
      vmin = ASC.move.hoist.speed.full;
      s = vmax + ctw/65000*(vmin-vmax);
      X = abs(O-D)*S.h;
      [E,m] = ASC_consumption('H',load,ct);

      % MODEL 2
      W = ASC.spreader.W + ctw;
      maxrpm = ASC.hoist.motor.loaded.rpm;
      minrpm = ASC.hoist.motor.unloaded.rpm;
      rpm = maxrpm + ctw/65000*(minrpm-maxrpm);
      inertia = ASC.hoist.motor.inertia;
      efficiency = ASC.hoist.motor.efficiency;
```

```
        F1 = W*9.81/1000; % in kN
        F2 = 0; %keyboard
    case 3 % TROLLEY
        a = ASC.move.troll.accel; d = a;
        s = ASC.move.troll.speed;
        X = abs(O-D)*S.w;
        [E,m] = ASC_consumption('T',load,ct);

        % MODEL 2
        W = ASC.trolley.W + ctw;
        rpm = ASC.trolley.motor.rpm;
        inertia = ASC.trolley.motor.inertia;
        efficiency = ASC.trolley.motor.efficiency;
        F1 = W*ASC.trolley.friction/1000; % in kN
        F2 = 3; % in kN
%         c = ASC.trolley.friction; % Friction coef
%         v = ASC.trolley.sheave;
%         friction = m/1000*c;
%         % Main rope rigid load
%         rope = (1-v^3)*(m+ASC.trolley.W)/2; % antes ASC.W.cabin
%         m = friction + rope;
        %no = ASC.trolley.motor.no;

end

% Time analysis of the movement
[x,t,full_speed,maxs] = travel_time(X,s,a,d);
t.total = t.total + inct;
travelt = ceil(t.total); % inct is the additional time to position spreader

% Energy model
if t.total == 0
    E = 0;
else
    % MODEL 1: POTENTIAL + KINETIC
    % ---------------------------------------------------------------------
    if EXEC.energy_model == 1
        E = X*E*factor/3600000;
    % MODEL 2: CRANE MOTOR BASED
    % ---------------------------------------------------------------------
    elseif EXEC.energy_model == 2

        P1 = F1 * s/efficiency;
        E1 = P1*(t.total-inct);

        P2 = F2 * s/efficiency;
        E2 = P2*(t.total-inct);

        E3 = 0;

        w = 2*3.141592*rpm/60;
        M4 = inertia*w/t.ac;
        P4 = M4*rpm/9550;
        E4 = P4*t.ac;
```

```
        F5 = W*s/t.ac/1000;
        P5 = F5 * s/efficiency;
        E5 = P5*t.ac;
%keyboard
        E = (E1+E2+E3+E4+E5)*factor/3600; % 3600 is to go to kwh

    %     efficiency = 0.9;
    %     if factor == 1
    %         E_ct = 0.5*m*9.81*a^2*t.total^2/efficiency;
    %         E_ac = 0.5*m*a^2*t.ac^2/efficiency; % Accelerate only
    %     else % Going down: only deceleration is considered
    %         %E_ct = 0.5*m*9.81*a^2*travelt^2/efficiency;
    %         E_ct = 0;
    %         E_ac = 0.5*m*a^2*t.ac^2/efficiency;
    %     end
    %     if t.ac > 0
    %         E_mot = inertia*(2*3.141592*rpm)^2/t.ac;
    %     else
    %         E_mot = 0;
    %     end
    %    E2 = E_ac + E_ct + E_mot;
        %disp(['Movement ' type ' Relationship between E models ' num2str(E1/E2) ])
        % Generate gantry movement curve

        %keyboard
        if t.total > 0
            if strcmp(type,'gantry') == 1
                C = ASC_trajectory(a,s,t.total,full_speed);
            else
                C = 0;
            end
        end

        %E = E2;
    else
        disp('Wrong energy model'); keyboard
    end
end
```

```
function [C,R] = ASC_ETM(asc,ct,origin,destiny,estim_or_calc)
% This function calculates the energy and time of an ASC to make a move

global BL COUNT CT TIME

[C,R] = ASC_cycle_ini();

C.originalct = ct; C.origin = origin; C.gs = origin.gs;

% Crane position
P = ASC_act_pos(asc); C.bay = P.bay; C.stack = P.stack; C.tier = P.tier;

% PART 1: CT PICKUP: Gantry to bay, Trolley to the stack, Lower to pickup and Lift CT
% ------------------------------------------------------------------------
C.bay = origin.bay; C.ct(1) = 0; C.ctmove(1) = ct;
[C.time(1),C.E(1),C.moves{1},C.go] = ASC_energy(P.bay,origin.bay,0,'gantry','empty',estim_or_calc);

% CT being picked. There are two possiblities:
% (1) from block to do housekeeping
% (2) from transfer area to stack

nr = 0;

if asc.housekeeping == 1 % There may be reshuffles
   nr = BL.GS(origin.gs).ocup - origin.tier;  % Calculate always to write in cycle
   if nr > 0
      [BAY] = BAY_copy(origin.bay);
      pos = CT_act_pos(ct);
      [fslots,rslots] =  BAY_ES(pos.bay,pos.stack);

      if sum(fslots) < nr
         BAY.cts
         disp('BAY Reshuffles warning: No space in bay for Reshuffles. CTs should be moved to other bays');
         keyboard
      else
      % Count the number of reshuffles
         if estim_or_calc == 0
            if strcmp(CT(ct).id,'EXP') == 1
               e = COUNT.RhkE.events + 1;
               COUNT.RhkE.events = e;
               COUNT.RhkE.nr(e) = nr;
               COUNT.RhkE.revent(e) = TIME.t;
               COUNT.RhkE.sheight(e) = BL.GS(origin.gs).ocup;
            elseif strcmp(CT(ct).id,'IMP') == 1
               i = COUNT.RhkI.events + 1;
               COUNT.RhkI.events = i;
               COUNT.RhkI.nr(i) = nr;
               COUNT.RhkI.revent(i) = TIME.t;
               COUNT.RhkI.sheight(i) = BL.GS(origin.gs).ocup;
            end
         end

         BAY.time = max(TIME.t,asc.position.time(end));
```

```
        W = 1; % Set W = 1 to account for the number of reshuffles
        %keyboard
        [ETM,newBAY,sol] = BAY_reshuffles(asc,ct,BAY,W);

        % Check if the trolley can be simultaneous to gantry
        if ETM.time(1) < C.time(end)
           ETM.time(1) = 0;
        else
           ETM.time(1) = ETM.time(1) - C.time(end);
        end

        C.time = [C.time,ETM.time];  C.E = [C.E,ETM.E];  C.moves = [C.moves,ETM.moves];
        C.bay = [C.bay,ETM.bay];  C.stack = [C.stack,ETM.stack];  C.tier = [C.tier,ETM.tier];
        C.ct = [C.ct, ETM.ct];  C.ctmove = [C.ctmove, ETM.ctmove];
      end
   end
else % Pick from transfer area to stack
   if origin.tier > 2
      origin.tier = 1;
   end
end

[C,n] = addposition(C,'trolley',origin.stack);  C.ct(n) = 0; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(P.stack,origin.stack,0,'trolley','empty',estim_or_calc);

% If no housekeeping, see if the trolley can be simultaneous to gantry
if asc.housekeeping == 0
   if C.time(n) < C.time(n-1)
      C.time(n) = 0;
   else
      C.time(n) = C.time(n) - C.time(n-1);
   end
end

[C,n] = addposition(C,'hoist',origin.tier); C.ct(n) = 0; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(BL.tiers+1,origin.tier,0,'lower','empty',estim_or_calc);

[C,n] = addposition(C,'hoist',BL.tiers+1);  C.ct(n) = ct; C.ctmove(n) = ct;

%keyboard
if asc.housekeeping == 1
   [C.time(n),C.E(n),C.moves{n}] = ASC_energy(origin.tier,BL.tiers+1,ct,'pickbl','full',estim_or_calc);
else
   [C.time(n),C.E(n),C.moves{n}] = ASC_energy(origin.tier,BL.tiers+1,ct,'pickbf','full',estim_or_calc);
end

% PART 2. CT DELIVERY TO STACK
% ------------------------------------------------------------------------
[C,n] = addposition(C,'trolley',destiny.stack); C.ct(n) = ct; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(origin.stack,destiny.stack,ct,'trolley','full',estim_or_calc);

[C,n] = addposition(C,'gantry',destiny.bay); C.ct(n) = ct; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n},C.back] = ASC_energy(origin.bay,destiny.bay,ct,'gantry','full',estim_or_calc);
```

```
if C.time(n) > C.time(n-1)
   C.time(n-1) = 0;
else
   C.time(n-1) = C.time(n-1) - C.time(n);
end

[C,n] = addposition(C,'hoist',destiny.tier); C.ct(n) = ct; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(BL.tiers+1,destiny.tier,ct,'dropbl','full',estim_or_calc);

[C,n] = addposition(C,'hoist',BL.tiers+1); C.ct(n) = 0; C.ctmove(n) = ct;
[C.time(n),C.E(n),C.moves{n}] = ASC_energy(destiny.tier,BL.tiers+1,0,'raise','empty',estim_or_calc);

C.originaltime = C.time;
C.nr = nr;

% PART 3. Calculate the potential reshuffles of CTs underneath the CT being stacked
% ------------------------------------------------------------------------------
if estim_or_calc == 1 %and(strcmp(asc.id,'sea') == 1 , )
   nr = destiny.tier - 1;
   if nr > 0
      [BAY] = BAY_copy(destiny.bay);
      BAY.time = max(TIME.t,asc.position.time(end));
      BAY.cts(destiny.tier,destiny.stack) = ct;
      BAY.slots(destiny.tier,destiny.stack) = 1;
      for r = 1:nr
         h = destiny.tier - r;
         nct = BL.GS(destiny.gs).cts(h);
         if h == BL.tiers
            keyboard
         end
         % dift = (CT(ct).events.time(2)-CT(nct).events.time(2))/3600/24;
         % dift = fix(dift)+1;    dift = min(dift,8);
         % p(r) = PT(h,dift)/max(max(PT));
         [ETM,newBAY,sol] = BAY_reshuffles(asc,nct,BAY,0); %W = 0;
         if sol == 1
            ETM.E = ETM.E*CT(nct).pickp;
            R = cycle_add(ETM,R,r);
         else
            R.time = 1000000000;
            R.E = 1000000000000;
         end
      end
   end
end
```

```
function [slopes] = ASC_gantryslope(asc)

%keyboard
fin_task = length(asc.ciclo.bay);
ini_task = asc.ciclo.c_task;
s = 0; slopes = 0;
if asc.ciclo.no >0
   for t = ini_task:fin_task
      if strcmp(asc.ciclo.moves(t),'gantry')
         s = s+1;
         if t == 1;
            ascpos = ASC_act_pos(asc);
            delbay = asc.ciclo.bay(t)-ascpos.bay;
         else
            delbay = asc.ciclo.bay(t)-asc.ciclo.bay(t-1);
         end
         deltime = asc.ciclo.originaltime(t);
         slopes(s) = delbay/deltime;
      end
   end
end
```

```
function [itask,t_end] = ASC_get_gantry_task(asc,t_ref)
% THis function finds the gantry position in the cycle that is closer to
% the intersection point.
global TIME

if strcmp(asc.status,'wait') == 1
   itask = 0; t_end = 0; %keyboard
else
   % See the tasks to be executed later than the intersection
   %keyboard
   gantrypositions = find(strcmp(asc.ciclo.moves,'gantry'));
   % Compute the vector of accumulated time and place the origin in the
   % current moment
   t_passed = sum(asc.ciclo.originaltime) - sum(asc.ciclo.time);
   v_time_m = AUX_vect_ac(asc.ciclo.originaltime) - t_passed - (t_ref-TIME.t);
   time_dif = v_time_m(gantrypositions);
   [val] = find(time_dif>0); % min(abs(time_dif));
   if isempty(val)
      itask = gantrypositions(end);%keyboard
   else
      itask = gantrypositions(val(1));
   end
   v_time_m2 = AUX_vect_ac(asc.ciclo.time);
   t_end = v_time_m2(itask);
end
```

```
function [task_pos] = ASC_get_pointed(gs,asc)
% this function find the cts in a GS included in the asc tasks list

global BL CT

%keyboard

list_p_cts = 0; j = 0;
for i=1:BL.GS(gs).ocup
   cct = BL.GS(gs).cts(i);
   if CT(cct).pointed == 1
      tskposit = find(asc.tasks.ct == cct);
      if length(tskposit) > 0
         j = j+1;
         list_p_cts(j) = tskposit;
      end
   end
end

if j > 0
   list_p_cts = sort(list_p_cts);
   task_pos = list_p_cts(end)+1;
else
   task_pos = 0; % No priority
end
```

```
function [basetime] = ASC_getbasetime(asc)

global TIME

if asc.ciclo.no > 0
   basetime = asc.ciclo.basetime(1);
else
   basetime = TIME.t;
end
```

```
function ASC_init()

global ASC

t.action = {};
t.ct = [];
t.ct_id = {};
t.time = [];

crane.tasks = t;
crane.tasks.current = 0;

crane.extasks = t;
crane.extasks.type = {};

crane.housekeeping = 0;

crane.status = 'wait';
crane.plusevent=0;
crane.nextevent = 1000000;
crane.position.bay = 0;
crane.position.stack = 9;
crane.position.tier = 6;
crane.position.time = 0;
crane.position.ct = 0;
crane.speed = 0;

crane.ciclo = 0;
% crane.ciclo.no = 0;
% crane.ciclo.basetime(1) = 0;
crane.ciclo.c_task = 0;

ASC.sea = crane;
ASC.land = crane;

ASC.hktasks = t;
ASC.hktasks.current = 0;
ASC.hktasks.priority = 0;

% ASC.hktasks.IMP = [];
% ASC.hktasks.EXP = [];
ASC.hktasks.prior.IMP = [];
ASC.hktasks.prior.EXP = [];
ASC.hktasks.nonprior.IMP = [];
ASC.hktasks.nonprior.EXP = [];

ASC.sea.position.bay= 5;
ASC.land.position.bay = 35;

ASC.sea.id = 'sea';
ASC.land.id = 'land';

ASC.sea.ciclo = ASC_cycle_ini();
```

```
ASC.land.ciclo = ASC_cycle_ini();


ASC.move.gantry.speed.full = 240/60; % 240 before
ASC.move.gantry.speed.empty = 270/60; % (BEST). 240 before
ASC.move.gantry.accel = 0.40;
ASC.move.gantry.decel = 0.40;

ASC.move.troll.speed = 70/60;  % (BEST). Before: 60/60;
ASC.move.troll.accel = 0.30; % (BEST)

ASC.move.hoist.speed.full = 0.75; % (BEST) Before: 31/60;  % 39/60;
ASC.move.hoist.speed.empty = 1.5; % (BEST) Before: 62/60; % 72/60;
ASC.move.hoist.accel = 0.6; %(BEST). Before: 0.35;

% Weights
% ASC.W.crane = 116000-16000;
% ASC.W.spreader = 2000;
% ASC.W.cabin = 16000;

ASC.crane.W = 185000; % 240000;
ASC.spreader.W = 10000; % 2000;
ASC.trolley.W = 25000; % ASC.W.spreader + 35000;

ASC.gantry.motor.no = 12;
ASC.gantry.motor.rpm = 200/60; %1467/60;
ASC.gantry.motor.inertia = 0.16; %0.5;
ASC.gantry.motor.efficiency = 0.95;

ASC.trolley.motor.no = 4;
ASC.trolley.motor.rpm = 140/60; %1350/60; % rpm to rps
ASC.trolley.motor.inertia = 0.06; %20.66; % From catalogue, pedro vila gives 12.48; % kgm^2
ASC.trolley.friction = 6; % Friction coef in (kg/t)
ASC.trolley.sheave = 0.985;
ASC.trolley.motor.efficiency = 0.95; %

ASC.hoist.motor.no = 2;
ASC.hoist.motor.loaded.rpm = 180; %900/60; % rpm to rps
ASC.hoist.motor.unloaded.rpm = 360;
ASC.hoist.motor.inertia = 2.3; %110; % kgm^2
ASC.hoist.motor.efficiency = 0.95; %
```

```
function ASC_init()

global ASC BL

t.action = {};
t.ct = [];
t.ct_id = {};
t.time = [];

crane.tasks = t;
crane.tasks.current = 0;


crane.extasks = t;
crane.extasks.type = {};

crane.housekeeping = 0;

crane.status = 'wait';
crane.plusevent=0;
crane.nextevent = 1000000;
crane.position.bay = 0;
crane.position.stack = 1;
crane.position.tier = 6;
crane.position.time = 0;
crane.position.ct = 0;
crane.speed = 0;

crane.ciclo = 0;
% crane.ciclo.no = 0;
% crane.ciclo.basetime(1) = 0;
crane.ciclo.c_task = 0;

ASC.sea = crane;
ASC.land = crane;

ASC.hktasks = t;
ASC.hktasks.current = 0;
ASC.hktasks.priority = 0;

% ASC.hktasks.IMP = [];
% ASC.hktasks.EXP = [];
ASC.hktasks.prior.IMP = [];
ASC.hktasks.prior.EXP = [];
ASC.hktasks.nonprior.IMP = [];
ASC.hktasks.nonprior.EXP = [];

ASC.sea.position.bay= 1;
ASC.land.position.bay = BL.bays;

ASC.sea.id = 'sea';
ASC.land.id = 'land';
```

```
ASC.sea.ciclo = ASC_cycle_ini();
ASC.land.ciclo = ASC_cycle_ini();


ASC.move.gantry.speed.full = 240/60; % 240 before
ASC.move.gantry.speed.empty = 270/60; % (BEST). 240 before
ASC.move.gantry.accel = 0.40;
ASC.move.gantry.decel = 0.40;

ASC.move.troll.speed = 70/60;  % (BEST). Before: 60/60;
ASC.move.troll.accel = 0.30; % (BEST)

ASC.move.hoist.speed.full = 0.75; % (BEST) Before: 31/60;  % 39/60;
ASC.move.hoist.speed.empty = 1.5; % (BEST) Before: 62/60; % 72/60;
ASC.move.hoist.accel = 0.6; %(BEST). Before: 0.35;

% Weights
% ASC.W.crane = 116000-16000;
% ASC.W.spreader = 2000;
% ASC.W.cabin = 16000;

ASC.crane.W = 185000; % 240000;
ASC.spreader.W = 10000; % 2000;
ASC.trolley.W = 25000; % ASC.W.spreader + 35000;

ASC.gantry.motor.no = 12;
ASC.gantry.motor.rpm = 200; %1467/60;
ASC.gantry.motor.inertia = 2; %0.16; %0.5;
ASC.gantry.motor.efficiency = 0.95;
ASC.gantry.friction = 0.05; % Friction coef in (kg/t)

ASC.trolley.motor.no = 4;
ASC.trolley.motor.rpm = 140; %1350/60; % rpm to rps
ASC.trolley.motor.inertia = 8; %0.06; %20.66; % From catalogue, pedro vila gives 12.48; % kgm^2
ASC.trolley.friction = 0.06; % Friction coef in (kg/t)
ASC.trolley.sheave = 0.985;
ASC.trolley.motor.efficiency = 0.90; %

ASC.hoist.motor.no = 2;
ASC.hoist.motor.loaded.rpm = 180; %900/60; % rpm to rps
ASC.hoist.motor.unloaded.rpm = 360;
ASC.hoist.motor.inertia = 46; %2.3; %110; % kgm^2
ASC.hoist.motor.efficiency = 0.85; %

ASC.collisions = [];
```

```
function [orden] = ASC_interferencev()
% This funciton checks the interference of land crane with respect to sea
% crane.

global ASC BL S TIME

delay =0; Sgs =0; Lgs = 0;

% For later, identify the tasklist being used by the ascs
if strcmp(ASC.sea.status,'housekeeping') == 1
   seatasks = ASC.sea.hktasks;
else
   seatasks = ASC.sea.tasks;
end
if strcmp(ASC.land.status,'housekeeping') == 1
   landtasks = ASC.land.hktasks;
else
   landtasks = ASC.land.tasks;
end


% First identify the GSs where the ascs want to go

if ASC.land.ciclo.c_task > 0
   Lgs = ASC.land.ciclo.gs;
   if Lgs > 0
      Lbay = BL.GS(Lgs).bay;
   else
      Lbay = ASC.land.ciclo.bay;
   end
end

if ASC.sea.ciclo.c_task > 0
   Sgs = ASC.sea.ciclo.gs;
   if Sgs > 0
      Sbay = BL.GS(Sgs).bay;
   else
      Sbay = ASC.sea.ciclo.bay;
   end
end

if Lgs*Sgs > 0
   %keyboard
   if Lbay == Sbay % The two cranes have the same bay target
      %keyboard
      if strcmp(ASC.sea.status,'delivery')==1
         orden = 'S';
      elseif strcmp(ASC.land.status,'delivery')==1
         orden = 'L';
      elseif and(strcmp(ASC.sea.status,'stack')==1,strcmp(ASC.land.status,'stack')==1)
         if ASC.land.nextevent < ASC.sea.nextevent
            orden = 'L';
         else
```

```
            orden = 'S';
         end
         %keyboard
      else
         orden = 'S';
      end
   else
      orden = 'S';
   end
elseif or(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.sea.status,'housekeeping') == 1)
   orden = 'S'; %keyboard
elseif or(strcmp(ASC.land.status,'wait') == 1, strcmp(ASC.land.status,'housekeeping') == 1)
   orden = 'L'; %keyboard
else
   orden = 'S';
end


if and(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.land.status,'wait') == 1)
   % There is no intersection
   delay = 0;
else
   % First point of the cycle trajectory:
   xs = ASC_act_pos(ASC.sea);  XS(1) = xs.bay; TS(1) = xs.time;
   xl = ASC_act_pos(ASC.land); XL(1) = xl.bay; TL(1) = xl.time;

   delayfactor = 1.0;

   % There are several posibilities depending on the cranes status
   if and(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.land.status,'wait') ~= 1)
      % SEA is idle, land is not
      Ltask = ASC.land.ciclo.c_task;
      LCbays = ASC.land.ciclo.bay(Ltask:end);
      LCtime = ASC.land.ciclo.time(Ltask:end);
      XL = [XL,LCbays];
      tl = AUX_vect_ac(LCtime) + TIME.t;
      TL = [TL,tl];
      XS = ones(1,length(XL))*xs.bay;
      TS = [TS,TL(2:end)];
      %figure(22); plot(TS,XS,TL,XL); axis([min(TS) max(TS) -1 42 ]);
   elseif and(strcmp(ASC.sea.status,'wait') ~= 1, strcmp(ASC.land.status,'wait') == 1)
      % LAND is idle, land is not
      %keyboard
      Stask = ASC.sea.ciclo.c_task;
      SCbays = ASC.sea.ciclo.bay(Stask:end);
      SCtime = ASC.sea.ciclo.time(Stask:end);
      XS = [XS,SCbays];
      ts = AUX_vect_ac(SCtime) + TIME.t;
      TS = [TS,ts];
      XL = ones(1,length(XS))*xl.bay;
      TL = [TL,TS(2:end)];
      %figure(22); plot(TS,XS,TL,XL); axis([min(TL) max(TL) -1 42 ]);
   elseif and(strcmp(ASC.sea.status,'wait') ~= 1, strcmp(ASC.land.status,'wait') ~= 1)
      % Both cranes are busy
```

```
    %keyboard
    Ltask = ASC.land.ciclo.c_task;
    LCbays = ASC.land.ciclo.bay(Ltask:end);
    LCtime = ASC.land.ciclo.time(Ltask:end);
    XL = [XL,LCbays];
    tl = AUX_vect_ac(LCtime) + TIME.t;
    TL = [TL,tl];

    Stask = ASC.sea.ciclo.c_task;
    SCbays = ASC.sea.ciclo.bay(Stask:end);
    SCtime = ASC.sea.ciclo.time(Stask:end);
    XS = [XS,SCbays];
    ts = AUX_vect_ac(SCtime) + TIME.t;
    TS = [TS,ts];
    %figure(22); plot(TS,XS,TL,XL); axis([min(min(TS),min(TL)) max(max(TS),max(TL)) -1 42 ])
  end

  % 2. CHECK WHETHER THERE IS INTERSECTION
  % ----------------------------------------------------------------
  % Filter the vectors for intersection%keyboard
  [XSo,TSo] = AUX_filter_repeated(XS,TS);
  [XLo,TLo] = AUX_filter_repeated(XL,TL);
  XSi = XSo + S.baymargin+0.0001;
  XLi = XLo; % - S.baymargin; %1.4999;
  if isempty(XSi) == 0
    %keyboard
    Ri = 0; Fi = 0; % Set real and ficticious intersections

    [xRi,yRi,iout,jout] = intersections(XSo,TSo,XLo,TLo);
    [xFi,yFi,iout,jout] = intersections(XSi,TSo,XLi,TLo);
    if length(yFi) > 2
      disp('Complex intersection'); %keyboard
    end
    % Analize the real intersection
    [xRi0,yRi0,xFi0,yFi0,Ri,Fi,itype,Lintslope,Sintslope] =
INTERSECTION_yesno_init(xRi,yRi,xFi,yFi,0,XSo,TSo,XLo,TLo,XSi,XLi);
  else
    keyboard
  end

  % 3. CALCULATE THE TYPE OF INTERSECTION
  % -----------------------------------------

  if strcmp(itype,'N') == 0

    % 3.1 Get some intersection particulars for later
    Stask = ASC.sea.ciclo.c_task;
    Ltask = ASC.land.ciclo.c_task;

    % 3.2 Type of intersection
    % ----------------------------------------------------------------
    [escenario,matrizcaso,orden]= INTERS_type(Sintslope,Lintslope,itype,XL,XS,TL,TS,xRi0,yRi0,xFi0,yFi0);

    % 4 CALCULATION of intersection
```

```
% -------------------------------------------------------------
addSdelay = 0; addLdelay = 0; bayshift = 0;
switch escenario
   % CASES 1 and 2: INTRODUCE A DELAY
   case 1    % The LAND CRANE HAS PRIORITY and ascSEA C is waiting
   % -------------------------------------------------------------
      [ASC.land,ASC.sea] = ASC_trans_unprod(ASC.land,ASC.sea,TS);

   case 2    % The SEA CRANE HAS PRIORITY and ascLAND is waiting
   % -------------------------------------------------------------
      [ASC.sea,ASC.land] = ASC_trans_unprod(ASC.sea,ASC.land,TL); % active, idle

   case 3    % The LAND crane has priority
   % -------------------------------------------------------------
      % 1. Determine the final point of the delay
      lowbay = min(ASC.land.ciclo.bay);
      if matrizcaso == 9 % consider change lowbay
         lowbay = min(XL);
            keyboard
      end
      [t_task] = AUX_find_vector_pos(LCbays,lowbay); t_fin = TL(t_task+1);

      plot(t_fin/3600/24,lowbay,'<r'); keyboard

      % 2. Determine the INITIAL point of the delay
      % 2.1 No real intersection
      % -------------------------------------------------------
      if Ri == 0
         t_ini = yFi0;

      % 2.2 Plane intersection
      % -------------------------------------------------------
      elseif length(xRi) > 2
         t_ini = yFi0;

      % 2.3 Normal intersection
      % -------------------------------------------------------
      else
         [bayant,baypost] = AUX_pointinvector(XS,lowbay); %SCbays
         % Simple interserctions
         if size(bayant) == 1
            % None intersection case
            if bayant + baypost == 0
               t_ini = yFi0; inters = 0; %keyboard % Step(709) wrong
            else
               inters = 1; a = bayant(1); b = baypost(1);
            end
         % Multiple intersections
         else
            inters = 1;
            if strcmp(ASC.sea.status,'stack') == 1 % Case 4 8 land stack
               a = bayant(2);
               b = baypost(2);
            else
```

```
            %keyboard
            a = bayant(1); % Case 8
            b = baypost(1);
        end
    end
    if inters == 1
        VX(1) = XS(a); VX(2) = XS(b); % +1
        VT(1) = TS(a); VT(2) = TS(b); %+1
        plot(VT/3600/24,VX,'-.m')
        if VX(1) == VX(2)
            t_ini= VT(1);
        else
            t_ini = interp1(VX,VT,lowbay);
        end
    end
end
keyboard
plot(t_ini/3600/24,lowbay,'>r')

% Calculate the delay
delay = ceil(( t_fin - t_ini) * delayfactor); % We give a margin of 20 sec
% keyboard
delay = max(delay, 15);
bayshift = 0;

Dlowbay = lowbay; it = 0;
% Calculate the final distance of the solution
while lowbay - Dlowbay < S.baymargin %2
    it = it+1;
    % Registry of cases:
    % 8: land: delivery, Sea: stack. Real inter, same bay
    % 9: Land: trans, Sea: Stack Ficticious, ok
    PTS = TS + delay;
    plot(PTS/3600/24,XS,'-.r'); %keyboard
    [pant,ppost] = AUX_pointinvector(PTS,t_fin);
    if length(pant) >1
        % in time, there can only exist one intersection
        pant = pant(1); ppost = ppost(1);
    end
    if pant+ppost > 0
        %keyboard
        VX(1) = XS(pant); VX(2) = XS(ppost);
        VT(1) = PTS(pant); VT(2) = PTS(ppost);
        if VX(1) == VX(2)
            %keyboard
            Dlowbay = VX(1);
        elseif VT(1) == VT(2)
            keyboard;
            VT(2) = PTS(ppost+1); Dlowbay = interp1(VT,VX,t_fin);
        else
            Dlowbay = interp1(VT,VX,t_fin); % ojo aqui con la parada anterior
        end
        plot(VT/3600/24,VX,'.-k')
    else
```

```
            % Case 8 9
            Dlowbay = XS(1);
            %keyboard
        end
        plot(t_fin/3600/24,Dlowbay,'*k');
        if lowbay - Dlowbay <0
            a=1; %keyboard
        elseif lowbay - Dlowbay <= S.baymargin; %2
            delay = delay + 5;
        end
        if it > 10
            %keyboard
            delay = TL(end)-TIME.t + 5;
            Dlowbay = lowbay - S.baymargin;
            if XS(1) > Dlowbay
                bayshift = Dlowbay;
            else
                bayshift = 0;
            end
            PTS = TS + delay;
            plot(PTS/3600/24,XS,'-.m');
        end
    end

    %plot((TS+delay)/3600/24,XS,'.-r');
    disp(['ASC(L&S) Interference. LAND crane priority. ASC(sea) Delay: ' num2str(delay)])
    % keyboard
    if bayshift ~= 0
        %keyboard
        endtime = TL(end);
        disp(['ASC retard until ' num2str((endtime-TIME.t))])
        guilty_ct = landtasks.ct(landtasks.current); %keyboard
        [ASC.sea] = ASC_retard(ASC.sea,bayshift,endtime,guilty_ct,xRi0,yRi0,xFi0,yFi0);
    else
        addSdelay = 1; %[ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,bayshift);
        orden = 'L';
    end

% ------------------------------------------------------------
case 4    % The SEA CRANE has priority. Introduce delay the land crane
% ------------------------------------------------------------
    upbay = max(ASC.sea.ciclo.bay);
    [t_task] = AUX_find_vector_pos(SCbays,upbay); t_fin = TS(t_task+1);
    plot(t_fin/3600/24,upbay,'<r');
    %keyboard

    % Find the intersection of the cranes
    if Ri == 0
        t_ini = yFi0;
    elseif Ri == 1
        %keyboard
        [bayant,baypost] = AUX_pointinvector(XL,upbay); % LCbays
        % Simple intersection
        if size(bayant) == 1
```

```
            if bayant+baypost == 0
               t_ini = yFi0; inters = 0; %keyboard % Case 1
            else
               a = bayant(1); b = baypost(1); inters = 1;
            end
         else
            % Multiple intersections
            inters = 1;
            % Case 4 land stack
            if strcmp(ASC.land.status,'stack') == 1
               a = bayant(2);
               b = baypost(2);
            else
               % Case 4 failed.
               % Case 7 failed did not add the delay to the
               % task
               %keyboard
               a = bayant(1);
               b = baypost(1);
            end
         end

         if inters == 1
            VX(1) = XL(a); VX(2) = XL(b); % +1
            VT(1) = TL(a); VT(2) = TL(b); % +1
            plot(VT/3600/24,VX,'-.m')

            if VX(1) == VX(2)
               t_ini= VT(1);
            else
               t_ini = interp1(VX,VT,upbay);
            end
         end
      end

      plot(t_ini/3600/24,upbay,'>r')
      delay = ceil((t_fin -t_ini) * delayfactor); % We give a margin of 15 sec
      delay = max(delay, 15); Pupbay = upbay;
      bayshift = 0;
      %keyboard
      if matrizcaso ~= 1
         % Registry of Cases:
%           if and(matrizcaso == 7, strcmp(ASC.land.status,'delivery')==1)
%              keyboard
%           end
         % 7 Stack-Stack, ok
         % 4 Stack-Stack, ok
         %keyboard
         it = 0;
         while Pupbay - upbay < S.baymargin
            it = it +1;
            PTL = TL + delay;
            plot(PTL/3600/24,XL,'-.r');
            [pant,ppost] = AUX_pointinvector(PTL,t_fin);
```

```
            if length(pant) >1
               % In time, only one intersection is possible
               pant = pant(1); ppost = ppost(1);
            end
%              if matrizcaso == 4
%                 if pant>length(XL)
%                    keyboard % problems with VX(1) = XL(pant);
%                 end
%              end
            if pant + ppost >0 % Normal intersection
               VX(1) = XL(pant);
               VX(2) = XL(ppost);
               VT(1) = PTL(pant); VT(2) = PTL(ppost);
               if VX(1)==VX(2)
                  Pupbay = VX(1);
               else
                  Pupbay = interp1(VT,VX,t_fin);
               end
               plot(VT/3600/24,VX,'.-k')
            else % There is no time intersection
               %keyboard
               Pupbay = XL(1); % Case 9
            end
            plot(t_fin/3600/24,Pupbay,'*k')

            if Pupbay - upbay <= S.baymargin
               delay = delay + 5;
            end
            if it>40
               %keyboard
               delay = TS(end) - TIME.t + 5;
               Pupbay = upbay + S.baymargin;
               if XL(1) < Pupbay
                  bayshift = Pupbay;
               else
                  bayshift = 0;
               end
               PTL = TL + delay;
               plot(PTL/3600/24,XL,'-.r');
            end
         end
      else % Case 1 going downwards
         a = 1;
      end

      disp(['ASC(L&S) Interference. SEA crane priority. ASC(land) Delay: ' num2str(delay)])
      if bayshift ~= 0
         %keyboard
         endtime = TS(end);
         disp(['ASC land retard until ' num2str((endtime-TIME.t))])

         guilty_ct = seatasks.ct(seatasks.current);
         [ASC.land] = ASC_retard(ASC.land,bayshift,endtime,guilty_ct,xRi0,yRi0,xFi0,yFi0);
      else
```

```
        addLdelay = 1;  %[ASC.land] = ASC_adddelay(ASC.land,Ltask,delay,bayshift);
        orden = 'S';
      end
    % ------------------------------------------------------------
    case 5 % LAND CRANE PRIORITY
    % ------------------------------------------------------------
      %keyboard
      delay = 5;
      it = 0;
      PXS = XSo+3;
      while xFi > 0
        it = it+1;
        PTS = TSo + delay; plot(PTS/3600/24,PXS,'.-r')
        [xFi,yFi,iout,jout] = intersections(PXS,PTS,XLo,TLo);
        if xFi >0
          delay = delay + 5;
        end
        if it > 100
          %keyboard
          xFi = 0;  delay = 10; % to leave the
        end
      end
      addSdelay = 1; bayshift = 0; %[ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,0);
    % CASES 6 & 7: RETARD A CRANE
    case 6 % SEA CRANE PRIORITY. Retard the ASC land crane
      %keyboard % Step (1196, 1200, 1208)
      %ascpos = ASC_act_pos(ASC.sea);
      [sea_bay] = ASC_target_bay(ASC.sea); keyboard
      endtime = TS(end);
      disp(['ASC land case 6 retard'])
      guilty_ct = ASC.sea.ciclo.ctmove(end);
      [ASC.land] = ASC_retard(ASC.land,sea_bay+3,endtime,guilty_ct,xRi0,yRi0,xFi0,yFi0);
      orden = 'L';
    case 7 % LAND CRANE PRIORITY. Retard the ASC sea crane
      %ascpos = ASC_act_pos(ASC.land);
      [land_bay] = ASC_target_bay(ASC.land);
      endtime = TL(end);
      disp(['ASC sea case 7 retard' num2str((endtime-TIME.t))]);
      guilty_ct = ASC.land.ciclo.ctmove(end);
      [ASC.sea] = ASC_retard(ASC.sea,land_bay-3,endtime,guilty_ct,xRi0,yRi0,xFi0,yFi0);
      orden = 'S';
    case 10 % Introduce a delay in the land crane
      keyboard
      [land_bay] = ASC_target_bay(ASC.land);
      [sea_bay] = ASC_target_bay(ASC.sea); sea_bay = -1;
      i_delay = 10;
      [ASC.land] = ASC_adddelay(ASC.land,Ltask,i_delay,0);
      % Introduce a new task in the sea crane
      delay = i_delay + TL(end)
      [ASC.sea] = ASC_adddelay(ASC.sea,Ltask,delay,0);
      land_line = find(XL == land_bay);
      sea_line = find(XS == sea_bay);

end
```

```
    % Check if there is need to add a delay
    if addSdelay == 1
       guilty_ct = landtasks.ct(landtasks.current);
       [ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,bayshift,guilty_ct);
    end

    if addLdelay == 1
       guilty_ct = seatasks.ct(seatasks.current);
       [ASC.land] = ASC_adddelay(ASC.land,Ltask,delay,bayshift,guilty_ct);
    end

    disp(['/////////// Intersection case ' num2str(matrizcaso) ' ////////////////////////////////////'])
    title(['Intersection ' num2str(matrizcaso) ' escenario ' num2str(escenario)])
  end
end
```

```
function [orden] = ASC_interferencev2()
% This funciton checks the interference of land crane with respect to sea
% crane.

global ASC BL S

delay =0; Sgs =0; Lgs = 0; orden = 'S';

% For later, identify the tasklist being used by the ascs
if strcmp(ASC.sea.status,'housekeeping') == 1
   seatasks = ASC.hktasks;
else
   seatasks = ASC.sea.tasks;
end
if strcmp(ASC.land.status,'housekeeping') == 1
   landtasks = ASC.hktasks;
else
   landtasks = ASC.land.tasks;
end

if and(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.land.status,'wait') == 1)
   % There is no possibility for intersection
   delay = 0;
else
   % 1. Get the trajectories of the cranes
   [XS,TS,XL,TL] = INTERS_asc_trajectories();

   % 2. See if there is intersection and get the point
   [Ri,xi,yi] = INTERS_exist(XS,TS,XL,TL,S.baymargin);

   % 3. CALCULATE THE TYPE OF INTERSECTION
   % -----------------------------------------
   if Ri == 1
      % 3.1 Analize the intersection
      %keyboard
      % -----------------------------------------------------------------
      if strcmp(ASC.sea.status , 'wait') == 1
         escenario = 1; orden = 'L';
      elseif strcmp(ASC.land.status , 'wait') == 1
         escenario = 2; orden = 'S';
      else
         [escenario,orden] = INTERSECTION_set_escenario(XL,XS,TL,TS,xi,yi);
      end
      title(['Intersection escenario ' num2str(escenario)])

      % 4 CALCULATION of intersection
      % -----------------------------------------------------------------
      addSdelay = 0; addLdelay = 0; bayshift = 0;

      switch escenario
         % CASES 1 and 2: INTRODUCE A DELAY
         % ------------------------------------------------------------
         case 1    % The LAND CRANE HAS PRIORITY and ascSEA C is waiting
```

```
        % ------------------------------------------------------------
           [ASC.land,ASC.sea] = ASC_trans_unprod(ASC.land,ASC.sea,TS);
        % ------------------------------------------------------------
        case 2    % The SEA CRANE HAS PRIORITY and ascLAND is waiting
        % ------------------------------------------------------------
           [ASC.sea,ASC.land] = ASC_trans_unprod(ASC.sea,ASC.land,TL); % active, idle
        % ------------------------------------------------------------
        case 3    % The LAND crane has priority
        % ------------------------------------------------------------
           te = find(TL>yi); xe = XL(te);
           lowbay = min(floor(xi),min(xe)-S.baymargin-0);
           %lowbay = min(XL)-S.baymargin; keyboard
           endtime = TL(end);
           guilty_ct = ASC.land.ciclo.originalct; %keyboard
           [ASC.sea] = ASC_retard(ASC.sea,lowbay,endtime,guilty_ct,XL,XS,TL,TS,xi,yi);
           orden = 'S';
        % ------------------------------------------------------------
        case 4    % The SEA CRANE has priority. Introduce delay the land crane
        % ------------------------------------------------------------
           te = find(TS>yi); xe = XS(te);
           upbay = max(ceil(xi),max(xe)+S.baymargin+0);
           %upbay = max(XS)+S.baymargin; keyboard
           endtime = TS(end);
           guilty_ct = ASC.sea.ciclo.originalct; %keyboard
           [ASC.land] = ASC_retard(ASC.land,upbay,endtime,guilty_ct,XS,XL,TS,TL,xi,yi);
           orden = 'L';
     end

     % Check if there is need to add a delay
     if addSdelay == 1
        guilty_ct = landtasks.ct(landtasks.current);
        Stask = ASC.sea.ciclo.c_task;
        [ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,bayshift,guilty_ct);
     end

     if addLdelay == 1
        guilty_ct = seatasks.ct(seatasks.current);
        Ltask = ASC.land.ciclo.c_task;
        [ASC.land] = ASC_adddelay(ASC.land,Ltask,delay,bayshift,guilty_ct);
     end
   end
end

% % Final check
% [fXS,fTS,fXL,fTL] = INTERS_asc_trajectories();
% [Ri,xi,yi] = INTERS_exist(fXS,fTS,fXL,fTL,0);
% if Ri >0
%    figure; plot(fTS,fXS,fTL,fXL,yi,xi,'*')
%    disp('Intersection error: crane collision')
%    keyboard
% end
```

```
function [orden] = ASC_interferencev3(baymargin)
% This funciton checks the interference of land crane with respect to sea
% crane

global ASC BL TIME

delay =0; S.gs =0; L.gs = 0;
% First identify the GSs where the ascs want to go

if ASC.land.ciclo.c_task > 0
   L.gs = ASC.land.ciclo.gs;
   if L.gs > 0
      L.bay = BL.GS(L.gs).bay;
   else
      L.bay = ASC.land.ciclo.bay;
   end
end

if ASC.sea.ciclo.c_task > 0
   S.gs = ASC.sea.ciclo.gs;
   if S.gs > 0
      S.bay = BL.GS(S.gs).bay;
   else
      S.bay = ASC.sea.ciclo.bay;
   end
end

if L.gs*S.gs > 0
   %keyboard
   if L.bay == S.bay % The two cranes have the same bay target
      %keyboard
      if strcmp(ASC.sea.status,'delivery')==1
         orden = 'S';
      elseif strcmp(ASC.land.status,'delivery')==1
         orden = 'L';
      elseif and(strcmp(ASC.sea.status,'stack')==1,strcmp(ASC.land.status,'stack')==1)
         if ASC.land.nextevent < ASC.sea.nextevent
            orden = 'L';
         else
            orden = 'S';
         end
         %keyboard
      else
         orden = 'S';
      end
   else
      orden = 'S';
   end
elseif or(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.sea.status,'housekeeping') == 1)
   orden = 'S'; %keyboard
elseif or(strcmp(ASC.land.status,'wait') == 1, strcmp(ASC.land.status,'housekeeping') == 1)
   orden = 'L'; %keyboard
else
```

```
   orden = 'S';
end


if and(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.land.status,'wait') == 1)
   % There is no intersection
   delay = 0;
else
   % First point of the cycle trajectory:
   xs = ASC_act_pos(ASC.sea);  S.X(1) = xs.bay; S.T(1) = xs.time;
   xl = ASC_act_pos(ASC.land); L.X(1) = xl.bay; L.T(1) = xl.time;

   delayfactor = 1.0;

   % There are several posibilities depending on the cranes status
   if and(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.land.status,'wait') ~= 1)
      % SEA is idle, land is not
      L.task = ASC.land.ciclo.c_task;
      L.Cbays = ASC.land.ciclo.bay(L.task:end);
      L.Ctime = ASC.land.ciclo.time(L.task:end);
      L.X = [L.X,L.Cbays];
      tl = AUX_vect_ac(L.Ctime) + TIME.t;
      L.T = [L.T,tl];
      S.X = ones(1,length(L.X))*xs.bay;
      S.T = [S.T,L.T(2:end)];
      %figure(22); plot(S.T,S.X,L.T,L.X); axis([min(S.T) max(S.T) -1 42 ]);
   elseif and(strcmp(ASC.sea.status,'wait') ~= 1, strcmp(ASC.land.status,'wait') == 1)
      % LAND is idle, land is not
      %keyboard
      S.task = ASC.sea.ciclo.c_task;
      S.Cbays = ASC.sea.ciclo.bay(S.task:end);
      S.Ctime = ASC.sea.ciclo.time(S.task:end);
      S.X = [S.X,S.Cbays];
      ts = AUX_vect_ac(S.Ctime) + TIME.t;
      S.T = [S.T,ts];
      L.X = ones(1,length(S.X))*xl.bay;
      L.T = [L.T,S.T(2:end)];
      %figure(22); plot(S.T,S.X,L.T,L.X); axis([min(L.T) max(L.T) -1 42 ]);
   elseif and(strcmp(ASC.sea.status,'wait') ~= 1, strcmp(ASC.land.status,'wait') ~= 1)
      % Both cranes are busy
      %keyboard
      L.task = ASC.land.ciclo.c_task;
      L.Cbays = ASC.land.ciclo.bay(L.task:end);
      L.Ctime = ASC.land.ciclo.time(L.task:end);
      L.X = [L.X,L.Cbays];
      tl = AUX_vect_ac(L.Ctime) + TIME.t;
      L.T = [L.T,tl];

      S.task = ASC.sea.ciclo.c_task;
      S.Cbays = ASC.sea.ciclo.bay(S.task:end);
      S.Ctime = ASC.sea.ciclo.time(S.task:end);
      S.X = [S.X,S.Cbays];
      ts = AUX_vect_ac(S.Ctime) + TIME.t;
      S.T = [S.T,ts];
```

```
    %figure(22); plot(S.T,S.X,L.T,L.X); axis([min(min(S.T),min(L.T)) max(max(S.T),max(L.T)) -1 42 ])
  end

  % 2. CHECK WHETHER THERE IS INTERSECTION
  % ---------------------------------------------------------------------
  %[XSi,TSi] = AUX_time_filter(S.X,S.T,xs.time);
  %[L.Xi,L.Ti] = AUX_time_filter(L.X,L.T,xl.time);

  % Filter the vectors for intersection%keyboard
  [S.Xo,S.To] = AUX_filter_repeated(S.X,S.T);
  [L.Xo,L.To] = AUX_filter_repeated(L.X,L.T);
  S.Xi = S.Xo + 1.4999; L.Xi = L.Xo - 1.4999;

  if isempty(S.Xi) == 0
    %keyboard
    Ri = 0; Fi = 0; % Set real and ficticious intersections

    [xRi,yRi,iout,jout] = intersections(S.Xo,S.To,L.Xo,L.To);
    [xFi,yFi,iout,jout] = intersections(S.Xi,S.To,L.Xi,L.To);
    if length(yFi) > 2
      disp('Complex intersection'); %keyboard
    end
    % Analize the real intersection
    [xRi0,yRi0,xFi0,yFi0,Ri,Fi,itype] =
INTERSECTION_filter_points(xRi,yRi,xFi,yFi,0,S.Xo,S.To,L.Xo,L.To,S.Xi,L.Xi);
  end

  % 3. CALCULATE THE TYPE OF INTERSECTION
  % -----------------------------------------

  if yFi0 > 0

    % 3.1 Plot the intersection
    plot_intersection(S.X,S.T,L.X,L.T,Ri,xRi0,yRi0,Fi,xFi0,yFi0);

    % 3.2 Get some intersection particulars for later
    Stask = ASC.sea.ciclo.c_task;
    Ltask = ASC.land.ciclo.c_task;
    [Lintslope]= ASC_way(xFi0,yFi0,L.Xi,L.To); %(L.Xo(2)-L.Xo(1))/(L.To(2)-L.To(1));
    [Sintslope]= ASC_way(xFi0,yFi0,S.Xi,S.To); %(S.Xo(2)-S.Xo(1))/(S.To(2)-S.To(1));

    % 3.3 Check a particular case of error
    if length(yFi) == 2
      if Sintslope + Lintslope ~= 0
        if abs(yFi(1)-yFi(2))/10000 > 0.2
          keyboard
        end
      end
    end

    % 3.4 Type of intersection
    % ---------------------------------------------------------------
    [escenario,matrizcaso,orden]=
INTERS_type(Ri,Fi,Sintslope,Lintslope,itype,L.X,S.X,L.T,S.T,xRi0,yRi0,xFi0,yFi0);
```

```
% 4 CALCULATION of intersection
% --------------------------------------------------------------
addSdelay = 0; addLdelay = 0; bayshift = 0;
switch escenario
   % CASES 1 and 2: INTRODUCE A DELAY
   case 1    % The LAND CRANE HAS PRIORITY and ascSEA C is waiting
   % --------------------------------------------------------------
      [ASC.land,ASC.sea] = ASC_trans_unprod(ASC.land,ASC.sea,S.T);

   case 2    % The SEA CRANE HAS PRIORITY and ascLAND is waiting
   % --------------------------------------------------------------
      [ASC.sea,ASC.land] = ASC_trans_unprod(ASC.sea,ASC.land,L.T); % active, idle

   case 3    % The LAND crane has priority
   % --------------------------------------------------------------
      % 1. Determine the final point of the delay
      lowbay = min(ASC.land.ciclo.bay);
      if matrizcaso == 9 % consider change lowbay
         lowbay = min(L.X);
            keyboard
      end
      [t_task] = AUX_find_vector_pos(L.Cbays,lowbay); t_fin = L.T(t_task+1);
      plot(t_fin/3600/24,lowbay,'<r');

      % 2. Determine the INITIAL point of the delay
      % 2.1 No real intersection
      % ------------------------------------------------------
      if Ri == 0
         t_ini = yFi0;

      % 2.2 Plane intersection
      % ------------------------------------------------------
      elseif length(xRi) > 2
         t_ini = yFi0;

      % 2.3 Normal intersection
      % ------------------------------------------------------
      else
         [bayant,baypost] = AUX_pointinvector(S.X,lowbay); %S.Cbays
         % Simple interserctions
         if size(bayant) == 1
            % None intersection case
            if bayant + baypost == 0
               t_ini = yFi0; inters = 0; %keyboard % Step(709) wrong
            else
               inters = 1; a = bayant(1); b = baypost(1);
            end
         % Multiple intersections
         else
            inters = 1;
            if strcmp(ASC.sea.status,'stack') == 1 % Case 4 8 land stack
               a = bayant(2);
               b = baypost(2);
```

```
        else
            %keyboard
            a = bayant(1); % Case 8
            b = baypost(1);
        end
    end
    if inters == 1
        VX(1) = S.X(a); VX(2) = S.X(b); % +1
        VT(1) = S.T(a); VT(2) = S.T(b); %+1
        plot(VT/3600/24,VX,'-.m')
        if VX(1) == VX(2)
            t_ini= VT(1);
        else
            t_ini = interp1(VX,VT,lowbay);
        end
    end
end
plot(t_ini/3600/24,lowbay,'>r')

% Calculate the delay
delay = ceil(( t_fin - t_ini) * delayfactor); % We give a margin of 20 sec
% keyboard
delay = max(delay, 15);
bayshift = 0;

Dlowbay = lowbay; it = 0;
% Calculate the final distance of the solution
while lowbay - Dlowbay < baymargin %2
    it = it+1;
    % Registry of cases:
    % 8: land: delivery, Sea: stack. Real inter, same bay
    % 9: Land: trans, Sea: Stack Ficticious, ok
    S.PT = S.T + delay;
    plot(S.PT/3600/24,S.X,'-.r'); %keyboard
    [pant,ppost] = AUX_pointinvector(S.PT,t_fin);
    if length(pant) >1
        % in time, there can only exist one intersection
        pant = pant(1); ppost = ppost(1);
    end
    if pant+ppost > 0
        %keyboard
        VX(1) = S.X(pant); VX(2) = S.X(ppost);
        VT(1) = S.PT(pant); VT(2) = S.PT(ppost);
        if VX(1) == VX(2)
            %keyboard
            Dlowbay = VX(1);
        elseif VT(1) == VT(2)
            keyboard;
            VT(2) = S.PT(ppost+1); Dlowbay = interp1(VT,VX,t_fin);
        else
            Dlowbay = interp1(VT,VX,t_fin); % ojo aqui con la parada anterior
        end
        plot(VT/3600/24,VX,'.-k')
    else
```

```
            % Case 8 9
            Dlowbay = S.X(1);
            %keyboard
        end
        plot(t_fin/3600/24,Dlowbay,'*k');
        if lowbay - Dlowbay <0
            a=1; %keyboard
        elseif lowbay - Dlowbay <= baymargin; %2
            delay = delay + 5;
        end
        if it > 10
            %keyboard
            delay = L.T(end)-TIME.t + 5;
            Dlowbay = lowbay - baymargin;
            if S.X(1) > Dlowbay
                bayshift = Dlowbay;
            else
                bayshift = 0;
            end
            S.PT = S.T + delay;
            plot(S.PT/3600/24,S.X,'-.m');
        end
    end

    %plot((S.T+delay)/3600/24,S.X,'.-r');
    disp(['ASC(L&S) Interference. LAND crane priority. ASC(sea) Delay: ' num2str(delay)])
    % keyboard
    if bayshift ~= 0
        %keyboard
        endtime = L.T(end);
        disp(['ASC retard until ' num2str((endtime-TIME.t))])
        guilty_ct = ASC.land.tasks.ct(ASC.land.tasks.current);
        [ASC.sea] = ASC_retard(ASC.sea,bayshift,endtime,guilty_ct);
    else
        addSdelay = 1; %[ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,bayshift);
        orden = 'L';
    end

% ------------------------------------------------------------
case 4    % The SEA CRANE has priority. Introduce delay the land crane
% ------------------------------------------------------------
    upbay = max(ASC.sea.ciclo.bay);
    [t_task] = AUX_find_vector_pos(S.Cbays,upbay); t_fin = S.T(t_task+1);
    plot(t_fin/3600/24,upbay,'<r');
    %keyboard

    % Find the intersection of the cranes
    if Ri == 0
        t_ini = yFi0;
    elseif Ri == 1
        %keyboard
        [bayant,baypost] = AUX_pointinvector(L.X,upbay); % L.Cbays
        % Simple intersection
        if size(bayant) == 1
```

```
            if bayant+baypost == 0
               t_ini = yFi0; inters = 0; %keyboard % Case 1
            else
               a = bayant(1); b = baypost(1); inters = 1;
            end
         else
            % Multiple intersections
            inters = 1;
            % Case 4 land stack
            if strcmp(ASC.land.status,'stack') == 1
               a = bayant(2);
               b = baypost(2);
            else
               % Case 4 failed.
               % Case 7 failed did not add the delay to the
               % task
               %keyboard
               a = bayant(1);
               b = baypost(1);
            end
         end

         if inters == 1
            VX(1) = L.X(a); VX(2) = L.X(b); % +1
            VT(1) = L.T(a); VT(2) = L.T(b); % +1
            plot(VT/3600/24,VX,'-.m')

            if VX(1) == VX(2)
               t_ini= VT(1);
            else
               t_ini = interp1(VX,VT,upbay);
            end
         end
      end

      plot(t_ini/3600/24,upbay,'>r')
      delay = ceil((t_fin -t_ini) * delayfactor); % We give a margin of 15 sec
      delay = max(delay, 15); Pupbay = upbay;
      bayshift = 0;
      %keyboard
      if matrizcaso ~= 1
         % Registry of Cases:
%           if and(matrizcaso == 7, strcmp(ASC.land.status,'delivery')==1)
%              keyboard
%           end
         % 7 Stack-Stack, ok
         % 4 Stack-Stack, ok
         %keyboard
         it = 0;
         while Pupbay - upbay < baymargin
            it = it +1;
            PTL = L.T + delay;
            plot(PTL/3600/24,L.X,'-.r');
            [pant,ppost] = AUX_pointinvector(PTL,t_fin);
```

```
            if length(pant) >1
               % In time, only one intersection is possible
               pant = pant(1); ppost = ppost(1);
            end
%              if matrizcaso == 4
%                 if pant>length(L.X)
%                    keyboard % problems with VX(1) = L.X(pant);
%                 end
%              end
            if pant + ppost >0 % Normal intersection
               VX(1) = L.X(pant);
               VX(2) = L.X(ppost);
               VT(1) = PTL(pant); VT(2) = PTL(ppost);
               if VX(1)==VX(2)
                  Pupbay = VX(1);
               else
                  Pupbay = interp1(VT,VX,t_fin);
               end
               plot(VT/3600/24,VX,'.-k')
            else % There is no time intersection
               %keyboard
               Pupbay = L.X(1); % Case 9
            end
            plot(t_fin/3600/24,Pupbay,'*k')

            if Pupbay - upbay <= baymargin
               delay = delay + 5;
            end
            if it>40
               %keyboard
               delay = S.T(end) - TIME.t + 5;
               Pupbay = upbay + baymargin;
               if L.X(1) < Pupbay
                  bayshift = Pupbay;
               else
                  bayshift = 0;
               end
               PTL = L.T + delay;
               plot(PTL/3600/24,L.X,'-.r');
            end
         end
      else % Case 1 going downwards
         a = 1;
      end

      disp(['ASC(L&S) Interference. SEA crane priority. ASC(land) Delay: ' num2str(delay)])
      if bayshift ~= 0
         %keyboard
         endtime = S.T(end);
         disp(['ASC land retard until ' num2str((endtime-TIME.t))])
         guilty_ct = ASC.sea.tasks.ct(ASC.sea.tasks.current);
         [ASC.land] = ASC_retard(ASC.land,bayshift,endtime,guilty_ct);
      else
         addLdelay = 1;  %[ASC.land] = ASC_adddelay(ASC.land,Ltask,delay,bayshift);
```

```
        orden = 'S';
      end
    % ------------------------------------------------------------
    case 5 % LAND CRANE PRIORITY
    % ------------------------------------------------------------
      %keyboard
      delay = 5;
      it = 0;
      S.PX = S.Xo+3;
      while xFi > 0
        it = it+1;
        S.PT = TSo + delay; plot(S.PT/3600/24,S.PX,'.-r')
        [xFi,yFi,iout,jout] = intersections(S.PX,S.PT,L.Xo,L.To);
        if xFi >0
          delay = delay + 5;
        end
        if it > 100
          %keyboard
          xFi = 0;  delay = 10; % to leave the
        end
      end
      addSdelay = 1; bayshift = 0; %[ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,0);
    % CASES 6 & 7: RETARD A CRANE
    case 6 % SEA CRANE PRIORITY. Retard the ASC land crane
      %keyboard % Step (1196, 1200, 1208)
      %ascpos = ASC_act_pos(ASC.sea);
      [sea_bay] = ASC_target_bay(ASC.sea);
      endtime = S.T(end);
      disp(['ASC land case 6 retard until ' num2str((endtime-TIME.t))])
      guilty_ct = ASC.sea.ciclo.ctmove(end);
      keyboard; [ASC.land] = ASC_retard(ASC.land,sea_bay+3,endtime,guilty_ct);
      orden = 'L';
    case 7 % LAND CRANE PRIORITY. Retard the ASC sea crane
      %ascpos = ASC_act_pos(ASC.land);
      [land_bay] = ASC_target_bay(ASC.land);
      endtime = L.T(end);
      disp(['ASC sea case 7 retard until ' num2str((endtime-TIME.t))]);
      guilty_ct = ASC.land.ciclo.ctmove(end);
      [ASC.sea] = ASC_retard(ASC.sea,land_bay-3,endtime,guilty_ct);
      orden = 'S';
    case 10 % Introduce a delay in the land crane
      keyboard
      [land_bay] = ASC_target_bay(ASC.land);
      [sea_bay] = ASC_target_bay(ASC.sea); sea_bay = -1;
      i_delay = 10;
      [ASC.land] = ASC_adddelay(ASC.land,Ltask,i_delay,0);
      % Introduce a new task in the sea crane
      delay = i_delay + L.T(end)
      [ASC.sea] = ASC_adddelay(ASC.sea,Ltask,delay,0);
      land_line = find(L.X == land_bay);
      sea_line = find(S.X == sea_bay);

end
```

```
    % Check if there is need to add a delay
    if addSdelay == 1
       if matrizcaso > 5
          a=1;%keyboard
       end
       guilty_ct = ASC.land.tasks.ct(ASC.land.tasks.current);
       [ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,bayshift,guilty_ct);
    end

    if addLdelay == 1
       if matrizcaso > 5
          a=1;%keyboard
       end
       guilty_ct = ASC.sea.tasks.ct(ASC.sea.tasks.current);
       [ASC.land] = ASC_adddelay(ASC.land,Ltask,delay,bayshift,guilty_ct);
    end

    disp(['/////////// Intersection case ' num2str(matrizcaso) ' ////////////////////////////////////'])
    title(['Intersection ' num2str(matrizcaso) ' escenario ' num2str(escenario)])
  end
end
```

```
function [orden] = ASC_interferencev5(baymargin)
% This funciton checks the interference of land crane with respect to sea
% crane

global ASC BL TIME

delay =0; S.gs =0; L.gs = 0;
% First identify the GSs where the ascs want to go

if ASC.land.ciclo.c_task > 0
   L.gs = ASC.land.ciclo.gs;
   if L.gs > 0
      L.bay = BL.GS(L.gs).bay;
   else
      L.bay = ASC.land.ciclo.bay;
   end
end

if ASC.sea.ciclo.c_task > 0
   S.gs = ASC.sea.ciclo.gs;
   if S.gs > 0
      S.bay = BL.GS(S.gs).bay;
   else
      S.bay = ASC.sea.ciclo.bay;
   end
end

if L.gs*S.gs > 0
   %keyboard
   if L.bay == S.bay % The two cranes have the same bay target
      %keyboard
      if strcmp(ASC.sea.status,'delivery')==1
         orden = 'S';
      elseif strcmp(ASC.land.status,'delivery')==1
         orden = 'L';
      elseif and(strcmp(ASC.sea.status,'stack')==1,strcmp(ASC.land.status,'stack')==1)
         if ASC.land.nextevent < ASC.sea.nextevent
            orden = 'L';
         else
            orden = 'S';
         end
         %keyboard
      else
         orden = 'S';
      end
   else
      orden = 'S';
   end
elseif or(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.sea.status,'housekeeping') == 1)
   orden = 'S'; %keyboard
elseif or(strcmp(ASC.land.status,'wait') == 1, strcmp(ASC.land.status,'housekeeping') == 1)
   orden = 'L'; %keyboard
else
```

```
   orden = 'S';
end


if and(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.land.status,'wait') == 1)
   % There is no intersection
   delay = 0;
else
   % First point of the cycle trajectory:
   xs = ASC_act_pos(ASC.sea);  S.X(1) = xs.bay; S.T(1) = xs.time;
   xl = ASC_act_pos(ASC.land); L.X(1) = xl.bay; L.T(1) = xl.time;

   delayfactor = 1.0;

   % There are several posibilities depending on the cranes status
   if and(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.land.status,'wait') ~= 1)
      % SEA is idle, land is not
      L.task = ASC.land.ciclo.c_task;
      L.Cbays = ASC.land.ciclo.bay(L.task:end);
      L.Ctime = ASC.land.ciclo.time(L.task:end);
      L.X = [L.X,L.Cbays];
      tl = AUX_vect_ac(L.Ctime) + TIME.t;
      L.T = [L.T,tl];
      S.X = ones(1,length(L.X))*xs.bay;
      S.T = [S.T,L.T(2:end)];
      %figure(22); plot(S.T,S.X,L.T,L.X); axis([min(S.T) max(S.T) -1 42 ]);
   elseif and(strcmp(ASC.sea.status,'wait') ~= 1, strcmp(ASC.land.status,'wait') == 1)
      % LAND is idle, land is not
      %keyboard
      S.task = ASC.sea.ciclo.c_task;
      S.Cbays = ASC.sea.ciclo.bay(S.task:end);
      S.Ctime = ASC.sea.ciclo.time(S.task:end);
      S.X = [S.X,S.Cbays];
      ts = AUX_vect_ac(S.Ctime) + TIME.t;
      S.T = [S.T,ts];
      L.X = ones(1,length(S.X))*xl.bay;
      L.T = [L.T,S.T(2:end)];
      %figure(22); plot(S.T,S.X,L.T,L.X); axis([min(L.T) max(L.T) -1 42 ]);
   elseif and(strcmp(ASC.sea.status,'wait') ~= 1, strcmp(ASC.land.status,'wait') ~= 1)
      % Both cranes are busy
      %keyboard
      L.task = ASC.land.ciclo.c_task;
      L.Cbays = ASC.land.ciclo.bay(L.task:end);
      L.Ctime = ASC.land.ciclo.time(L.task:end);
      L.X = [L.X,L.Cbays];
      tl = AUX_vect_ac(L.Ctime) + TIME.t;
      L.T = [L.T,tl];

      S.task = ASC.sea.ciclo.c_task;
      S.Cbays = ASC.sea.ciclo.bay(S.task:end);
      S.Ctime = ASC.sea.ciclo.time(S.task:end);
      S.X = [S.X,S.Cbays];
      ts = AUX_vect_ac(S.Ctime) + TIME.t;
      S.T = [S.T,ts];
```

```
    %figure(22); plot(S.T,S.X,L.T,L.X); axis([min(min(S.T),min(L.T)) max(max(S.T),max(L.T)) -1 42 ])
  end

  % 2. CHECK WHETHER THERE IS INTERSECTION
  % ---------------------------------------------------------------------
  %[XSi,TSi] = AUX_time_filter(S.X,S.T,xs.time);
  %[L.Xi,L.Ti] = AUX_time_filter(L.X,L.T,xl.time);

  % Filter the vectors for intersection%keyboard
  [S.Xo,S.To] = AUX_filter_repeated(S.X,S.T);
  [L.Xo,L.To] = AUX_filter_repeated(L.X,L.T);
  S.Xi = S.Xo + 1.4999; L.Xi = L.Xo - 1.4999;

  if isempty(S.Xi) == 0
    %keyboard

    keyboard
    [xRi,yRi,iout,jout] = intersections(S.Xo,S.To,L.Xo,L.To);
    if isempty(xRi)
      [xFi,yFi,iout,jout] = intersections(S.Xi,S.To,L.Xi,L.To);
      itype = 'F';
    else
      itype = 'R';
    end
    if length(yFi) > 2
      disp('Complex intersection'); %keyboard
    end
    % Analize the real intersection
    [xRi0,yRi0,xFi0,yFi0,Ri,Fi,itype] =
INTERSECTION_filter_points(xRi,yRi,xFi,yFi,0,S.Xo,S.To,L.Xo,L.To,S.Xi,L.Xi);
  end

  % 3. CALCULATE THE TYPE OF INTERSECTION
  % -----------------------------------------

  if yFi0 > 0

    % 3.1 Plot the intersection
    plot_intersection(S.X,S.T,L.X,L.T,Ri,xRi0,yRi0,Fi,xFi0,yFi0);

    % 3.2 Get some intersection particulars for later
    Stask = ASC.sea.ciclo.c_task;
    Ltask = ASC.land.ciclo.c_task;
    [Lintslope]= ASC_way(xFi0,yFi0,L.Xi,L.To); %(L.Xo(2)-L.Xo(1))/(L.To(2)-L.To(1));
    [Sintslope]= ASC_way(xFi0,yFi0,S.Xi,S.To); %(S.Xo(2)-S.Xo(1))/(S.To(2)-S.To(1));

    % 3.3 Check a particular case of error
    if length(yFi) == 2
      if Sintslope + Lintslope ~= 0
        if abs(yFi(1)-yFi(2))/10000 > 0.2
          keyboard
        end
      end
    end
```

```
    % 3.4 Type of intersection
    % --------------------------------------------------------------
    [escenario,matrizcaso,orden]=
INTERS_type(Ri,Fi,Sintslope,Lintslope,itype,L.X,S.X,L.T,S.T,xRi0,yRi0,xFi0,yFi0);

    % 4 CALCULATION of intersection
    % --------------------------------------------------------------
    addSdelay = 0; addLdelay = 0; bayshift = 0;
    switch escenario
      % CASES 1 and 2: INTRODUCE A DELAY
      case 1    % The LAND CRANE HAS PRIORITY and ascSEA C is waiting
      % --------------------------------------------------------------
        [ASC.land,ASC.sea] = ASC_trans_unprod(ASC.land,ASC.sea,S.T);

      case 2    % The SEA CRANE HAS PRIORITY and ascLAND is waiting
      % --------------------------------------------------------------
        [ASC.sea,ASC.land] = ASC_trans_unprod(ASC.sea,ASC.land,L.T); % active, idle

      case 3    % The LAND crane has priority
      % --------------------------------------------------------------
        % 1. Determine the final point of the delay
        lowbay = min(ASC.land.ciclo.bay);
        if matrizcaso == 9 % consider change lowbay
          lowbay = min(L.X);
            keyboard
        end
        [t_task] = AUX_find_vector_pos(L.Cbays,lowbay); t_fin = L.T(t_task+1);
        plot(t_fin/3600/24,lowbay,'<r');

        % 2. Determine the INITIAL point of the delay
        % 2.1 No real intersection
        % ------------------------------------------------------
        if Ri == 0
          t_ini = yFi0;

        % 2.2 Plane intersection
        % ------------------------------------------------------
        elseif length(xRi) > 2
          t_ini = yFi0;

        % 2.3 Normal intersection
        % ------------------------------------------------------
        else
          [bayant,baypost] = AUX_pointinvector(S.X,lowbay); %S.Cbays
          % Simple interserctions
          if size(bayant) == 1
            % None intersection case
            if bayant + baypost == 0
              t_ini = yFi0; inters = 0; %keyboard % Step(709) wrong
            else
              inters = 1; a = bayant(1); b = baypost(1);
            end
          % Multiple intersections
```

```
    else
        inters = 1;
        if strcmp(ASC.sea.status,'stack') == 1 % Case 4 8 land stack
            a = bayant(2);
            b = baypost(2);
        else
            %keyboard
            a = bayant(1); % Case 8
            b = baypost(1);
        end
    end
    if inters == 1
        VX(1) = S.X(a); VX(2) = S.X(b); % +1
        VT(1) = S.T(a); VT(2) = S.T(b); %+1
        plot(VT/3600/24,VX,'-.m')
        if VX(1) == VX(2)
            t_ini= VT(1);
        else
            t_ini = interp1(VX,VT,lowbay);
        end
    end
end
plot(t_ini/3600/24,lowbay,'>r')

% Calculate the delay
delay = ceil(( t_fin - t_ini) * delayfactor); % We give a margin of 20 sec
% keyboard
delay = max(delay, 15);
bayshift = 0;

Dlowbay = lowbay; it = 0;
% Calculate the final distance of the solution
while lowbay - Dlowbay < baymargin %2
    it = it+1;
    % Registry of cases:
    % 8: land: delivery, Sea: stack. Real inter, same bay
    % 9: Land: trans, Sea: Stack Ficticious, ok
    S.PT = S.T + delay;
    plot(S.PT/3600/24,S.X,'-.r'); %keyboard
    [pant,ppost] = AUX_pointinvector(S.PT,t_fin);
    if length(pant) >1
        % in time, there can only exist one intersection
        pant = pant(1); ppost = ppost(1);
    end
    if pant+ppost > 0
        %keyboard
        VX(1) = S.X(pant); VX(2) = S.X(ppost);
        VT(1) = S.PT(pant); VT(2) = S.PT(ppost);
        if VX(1) == VX(2)
            %keyboard
            Dlowbay = VX(1);
        elseif VT(1) == VT(2)
            keyboard;
            VT(2) = S.PT(ppost+1); Dlowbay = interp1(VT,VX,t_fin);
```

```
            else
               Dlowbay = interp1(VT,VX,t_fin); % ojo aqui con la parada anterior
            end
            plot(VT/3600/24,VX,'.-k')
         else
            % Case 8 9
            Dlowbay = S.X(1);
            %keyboard
         end
         plot(t_fin/3600/24,Dlowbay,'*k');
         if lowbay - Dlowbay <0
            a=1; %keyboard
         elseif lowbay - Dlowbay <= baymargin; %2
            delay = delay + 5;
         end
         if it > 10
            %keyboard
            delay = L.T(end)-TIME.t + 5;
            Dlowbay = lowbay - baymargin;
            if S.X(1) > Dlowbay
               bayshift = Dlowbay;
            else
               bayshift = 0;
            end
            S.PT = S.T + delay;
            plot(S.PT/3600/24,S.X,'-.m');
         end
      end

      %plot((S.T+delay)/3600/24,S.X,'.-r');
      disp(['ASC(L&S) Interference. LAND crane priority. ASC(sea) Delay: ' num2str(delay)])
      % keyboard
      if bayshift ~= 0
         %keyboard
         endtime = L.T(end);
         disp(['ASC retard until ' num2str((endtime-TIME.t))])
         guilty_ct = ASC.land.tasks.ct(ASC.land.tasks.current);
         [ASC.sea] = ASC_retard(ASC.sea,bayshift,endtime,guilty_ct);
      else
         addSdelay = 1; %[ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,bayshift);
         orden = 'L';
      end

   % ------------------------------------------------------------
   case 4    % The SEA CRANE has priority. Introduce delay the land crane
   % ------------------------------------------------------------
      upbay = max(ASC.sea.ciclo.bay);
      [t_task] = AUX_find_vector_pos(S.Cbays,upbay); t_fin = S.T(t_task+1);
      plot(t_fin/3600/24,upbay,'<r');
      %keyboard

      % Find the intersection of the cranes
      if Ri == 0
         t_ini = yFi0;
```

```
elseif Ri == 1
   %keyboard
   [bayant,baypost] = AUX_pointinvector(L.X,upbay); % L.Cbays
   % Simple intersection
   if size(bayant) == 1
      if bayant+baypost == 0
         t_ini = yFi0; inters = 0; %keyboard % Case 1
      else
         a = bayant(1); b = baypost(1); inters = 1;
      end
   else
      % Multiple intersections
      inters = 1;
      % Case 4 land stack
      if strcmp(ASC.land.status,'stack') == 1
         a = bayant(2);
         b = baypost(2);
      else
         % Case 4 failed.
         % Case 7 failed did not add the delay to the
         % task
         %keyboard
         a = bayant(1);
         b = baypost(1);
      end
   end

   if inters == 1
      VX(1) = L.X(a); VX(2) = L.X(b); % +1
      VT(1) = L.T(a); VT(2) = L.T(b); % +1
      plot(VT/3600/24,VX,'-.m')

      if VX(1) == VX(2)
         t_ini= VT(1);
      else
         t_ini = interp1(VX,VT,upbay);
      end
   end
end

plot(t_ini/3600/24,upbay,'>r')
delay = ceil((t_fin -t_ini) * delayfactor); % We give a margin of 15 sec
delay = max(delay, 15); Pupbay = upbay;
bayshift = 0;
%keyboard
if matrizcaso ~= 1
   % Registry of Cases:
%     if and(matrizcaso == 7, strcmp(ASC.land.status,'delivery')==1)
%       keyboard
%     end
   % 7 Stack-Stack, ok
   % 4 Stack-Stack, ok
   %keyboard
   it = 0;
```

```
        while Pupbay - upbay < baymargin
            it = it +1;
            PTL = L.T + delay;
            plot(PTL/3600/24,L.X,'-.r');
            [pant,ppost] = AUX_pointinvector(PTL,t_fin);
            if length(pant) >1
                % In time, only one intersection is possible
                pant = pant(1); ppost = ppost(1);
            end
%               if matrizcaso == 4
%                   if pant>length(L.X)
%                       keyboard % problems with VX(1) = L.X(pant);
%                   end
%               end
            if pant + ppost >0 % Normal intersection
                VX(1) = L.X(pant);
                VX(2) = L.X(ppost);
                VT(1) = PTL(pant); VT(2) = PTL(ppost);
                if VX(1)==VX(2)
                    Pupbay = VX(1);
                else
                    Pupbay = interp1(VT,VX,t_fin);
                end
                plot(VT/3600/24,VX,'.-k')
            else % There is no time intersection
                %keyboard
                Pupbay = L.X(1); % Case 9
            end
            plot(t_fin/3600/24,Pupbay,'*k')

            if Pupbay - upbay <= baymargin
                delay = delay + 5;
            end
            if it>40
                %keyboard
                delay = S.T(end) - TIME.t + 5;
                Pupbay = upbay + baymargin;
                if L.X(1) < Pupbay
                    bayshift = Pupbay;
                else
                    bayshift = 0;
                end
                PTL = L.T + delay;
                plot(PTL/3600/24,L.X,'-.r');
            end
        end
    else % Case 1 going downwards
        a = 1;
    end

    disp(['ASC(L&S) Interference. SEA crane priority. ASC(land) Delay: ' num2str(delay)])
    if bayshift ~= 0
        %keyboard
        endtime = S.T(end);
```

```
        disp(['ASC land retard until ' num2str((endtime-TIME.t))])
        guilty_ct = ASC.sea.tasks.ct(ASC.sea.tasks.current);
        [ASC.land] = ASC_retard(ASC.land,bayshift,endtime,guilty_ct);
    else
        addLdelay = 1;  %[ASC.land] = ASC_adddelay(ASC.land,Ltask,delay,bayshift);
        orden = 'S';
    end
% ------------------------------------------------------------
case 5 % LAND CRANE PRIORITY
% ------------------------------------------------------------
    %keyboard
    delay = 5;
    it = 0;
    S.PX = S.Xo+3;
    while xFi > 0
        it = it+1;
        S.PT = TSo + delay; plot(S.PT/3600/24,S.PX,'.-r')
        [xFi,yFi,iout,jout] = intersections(S.PX,S.PT,L.Xo,L.To);
        if xFi >0
            delay = delay + 5;
        end
        if it > 100
            %keyboard
            xFi = 0;  delay = 10; % to leave the
        end
    end
    addSdelay = 1; bayshift = 0; %[ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,0);
% CASES 6 & 7: RETARD A CRANE
case 6 % SEA CRANE PRIORITY. Retard the ASC land crane
    %keyboard % Step (1196, 1200, 1208)
    %ascpos = ASC_act_pos(ASC.sea);
    [sea_bay] = ASC_target_bay(ASC.sea);
    endtime = S.T(end);
    disp(['ASC land case 6 retard until ' num2str((endtime-TIME.t))])
    guilty_ct = ASC.sea.ciclo.ctmove(end);
    keyboard; [ASC.land] = ASC_retard(ASC.land,sea_bay+3,endtime,guilty_ct);
    orden = 'L';
case 7 % LAND CRANE PRIORITY. Retard the ASC sea crane
    %ascpos = ASC_act_pos(ASC.land);
    [land_bay] = ASC_target_bay(ASC.land);
    endtime = L.T(end);
    disp(['ASC sea case 7 retard until ' num2str((endtime-TIME.t))]);
    guilty_ct = ASC.land.ciclo.ctmove(end);
    [ASC.sea] = ASC_retard(ASC.sea,land_bay-3,endtime,guilty_ct);
    orden = 'S';
case 10 % Introduce a delay in the land crane
    keyboard
    [land_bay] = ASC_target_bay(ASC.land);
    [sea_bay] = ASC_target_bay(ASC.sea); sea_bay = -1;
    i_delay = 10;
    [ASC.land] = ASC_adddelay(ASC.land,Ltask,i_delay,0);
    % Introduce a new task in the sea crane
    delay = i_delay + L.T(end)
    [ASC.sea] = ASC_adddelay(ASC.sea,Ltask,delay,0);
```

```
            land_line = find(L.X == land_bay);
            sea_line = find(S.X == sea_bay);

        end

        % Check if there is need to add a delay
        if addSdelay == 1
            if matrizcaso > 5
                a=1;%keyboard
            end
            guilty_ct = ASC.land.tasks.ct(ASC.land.tasks.current);
            [ASC.sea] = ASC_adddelay(ASC.sea,Stask,delay,bayshift,guilty_ct);
        end

        if addLdelay == 1
            if matrizcaso > 5
                a=1;%keyboard
            end
            guilty_ct = ASC.sea.tasks.ct(ASC.sea.tasks.current);
            [ASC.land] = ASC_adddelay(ASC.land,Ltask,delay,bayshift,guilty_ct);
        end

        disp(['/////////// Intersection case ' num2str(matrizcaso) ' ////////////////////////////////////'])
        title(['Intersection ' num2str(matrizcaso) ' escenario ' num2str(escenario)])
    end
end
```

```
% ASC MAIN PROGRAM

clear all
close all
clc
profile on

%matlabpool open 2

global ASC BAYS BF BL COUNT COST CT EXEC HK INTERS LIMITS PLOT PT  S SEA_DELIVERY TIME TRF

EXEC.cutofftime = 10*3600*24; % Change the value of days

caso = 1;  EXEC.caso = ['0' num2str(caso)];
TRF.PARAM.level = 40;

EXEC.statefilename = ['STATE CASE' char(EXEC.caso) 'TRF ' num2str(TRF.PARAM.level) '.mat'];
% Energy model: 1-Potential, 2-Electric
EXEC.energy_model = 2;
EXEC.plot = 0;
EXEC.lasthksearch =0;
% Choose hotstart among three options:
% -1: run until cutoff and save a new initial state
%  0: complete execution
%  1: execute from cutoff
%  2: load previoustate (averias)

hotstart = 1;

if hotstart <= 0
   rng(caso);
   INIT(0) % 1 generates traffic, 0 does not generate
   TRF.PARAM.stackmode = 'psrandom';
   TRF.PARAM.inifill = 0; % -1: fill, 0: no fill and simulate, 1: load fill and simulate
   TRF.PARAM.baymaxocup = 40;

   blfilename = ['BL400901' num2str(TRF.PARAM.level) '.mat'];

   % 1. Create inifill during a number of days
   % ------------------------------------------------------------------------
   if TRF.PARAM.inifill == -1
      TIME.simulation.fill = fix(TRF.PARAM.level/100*BL.tiers*BL.bays*BL.stacks/2-TRF.PARAM.no.cts_imp/2);
   % 2. Do not create initial fill
   % ------------------------------------------------------------------------
   else
      TIME.simulation.fill = 0;
      % 2.1 Start simulation with empty block
      if TRF.PARAM.inifill == 0
         a = 0;
      % Start simulation with the block already occupied
      elseif TRF.PARAM.inifill == 1
         load(blfilename);  ct = COUNT.CT.no;
      end
```

```
   end
elseif hotstart == 1
   load(EXEC.statefilename);
   TIME.parar = 0;
   TRF.PARAM.stackmode = 'essa'; % tercat, essa, psrandom, crandom
   %--------------------------------------------------------------------
elseif hotstart == 2
   load('previoustate.mat');
end

% ----------------------------------------
TRF.PARAM.averias = 1;
TRF.PARAM.maxh = BL.tiers;
TRF.PARAM.weight.E = 1;
TRF.PARAM.weight.t = 1 - TRF.PARAM.weight.E;
% ----------------------------------------

% CREATE INITIAL FILL
if TRF.PARAM.inifill == -1
   BL_inifill(blfilename);

% SIMULATE
else
   while and(TIME.t <= TIME.simulation.T, TIME.parar == 0)

      % EVENT CREATOR
      TIME_next();

      if hotstart == -1
         if and(TIME.t > EXEC.cutofftime, TIME.e > 5)
            TIME.parar = 1; TIME.delt = 0;
            TIME.cut = TIME.t;
            continue
         end
      end

      switch TIME.e

         % LAND
         % ------------------------------------------------------------
         case 1 % Arrival of EXP ET
            [ct] = CT_generate1('EXP');
            BF_dropCT('land',ct);

         case 2 % Arrival of IMP TRUCK
            BL_search_pick1();

         case 3 % Arrival of DUAL CT
            [ct] = CT_generate1('DUAL');
            BF_dropCT('land',ct);
            BL_search_pick1();

            % SEA
            % ------------------------------------------------------
```

```
            case 4 % Arrival of IMP CT
                [ct] = CT_generate1('IMP');
                BF_dropCT('sea',ct);

            case 5 % Delivery of EXP CT
                %Sea_delivery3();
                Sea_delivery_tasks();
        end

        % CRANES
        % ---------------------------------------------------------------
        [orden] = ASC_interferencev2();

        ASC_progress(orden);
    end
end

%keyboard
if hotstart == -1
    TIME.parar = 0;

save(EXEC.statefilename,'ASC','BAYS','BF','BL','COST','COUNT','CT','INTERS','LIMITS','PLOT','PT','S','SEA_DELI
VERY','TIME','TRF');
else
    RESULTS
end

profile off

% Save workspace
if hotstart >= 1
    disp('Saving workspace')
    wsname = ['C' num2str(EXEC.caso) ' ' TRF.PARAM.stackmode ' TRF' num2str(TRF.PARAM.level) ' '
num2str(BL.bays) 'x' num2str(BL.stacks) '.mat' ];
    save(wsname);
end
```

```
% ASC MAIN PROGRAM
function ASC_main4exec(trflevel,energy_crit,stack_alg,hotstart,trf_gen)

clear global
close all
global ASC BAYS BF BL COUNT COST CT EXEC HK INTERS LIMITS PLOT PT  S SEA_DELIVERY TIME TRF

EXEC.cutofftime = 10*3600*24; % Change the value of days

caso = 1;  EXEC.caso = ['0' num2str(caso)];
TRF.PARAM.level = trflevel;

EXEC.statefilename = ['STATE CASE' char(EXEC.caso) 'TRF ' num2str(TRF.PARAM.level) '.mat'];

% Energy model: 1-Potential, 2-Electric
EXEC.energy_model = 2;
EXEC.plot = 0;
EXEC.lasthksearch =0;
% Choose hotstart among three options:
% -1: run until cutoff and save a new initial state
%  0: complete execution
%  1: execute from cutoff
%  2: load previoustate (averias)

%hotstart = 1;

if hotstart <= 0
   rng(caso);
   INIT(trf_gen) % 1 generates traffic, 0 does not generate
   TRF.PARAM.stackmode = 'psrandom';
   TRF.PARAM.inifill = 0; % -1: fill, 0: no fill and simulate, 1: load fill and simulate
   TRF.PARAM.baymaxocup = 40;

   blfilename = ['BL400901' num2str(TRF.PARAM.level) '.mat'];

   % 1. Create inifill during a number of days
   % ------------------------------------------------------------------------
   if TRF.PARAM.inifill == -1
      TIME.simulation.fill = fix(TRF.PARAM.level/100*BL.tiers*BL.bays*BL.stacks/2-TRF.PARAM.no.cts_imp/2);
   % 2. Do not create initial fill
   % ------------------------------------------------------------------------
   else
      TIME.simulation.fill = 0;
      % 2.1 Start simulation with empty block
      if TRF.PARAM.inifill == 0
         a = 0;
      % Start simulation with the block already occupied
      elseif TRF.PARAM.inifill == 1
         load(blfilename);  ct = COUNT.CT.no;
      end
   end
elseif hotstart == 1
   load(EXEC.statefilename);
```

```
   TIME.parar = 0;
   TRF.PARAM.stackmode = stack_alg; % tercat, essa, psrandom, crandom
   %--------------------------------------------------------------------
elseif hotstart == 2
   load('previoustate.mat');
end

% ----------------------------------------
TRF.PARAM.averias = 1;
TRF.PARAM.maxh = BL.tiers;
TRF.PARAM.weight.E = energy_crit;
TRF.PARAM.weight.t = 1 - TRF.PARAM.weight.E;
% ----------------------------------------

wsname = ['C' num2str(EXEC.caso) ' ' TRF.PARAM.stackmode num2str(TRF.PARAM.weight.E)
num2str(TRF.PARAM.weight.t) ' TRF' num2str(TRF.PARAM.level) ' ' num2str(BL.bays) 'x' num2str(BL.stacks) '.mat'
];

% CREATE INITIAL FILL
if TRF.PARAM.inifill == -1
   BL_inifill(blfilename);

% SIMULATE
else
   while and(TIME.t <= TIME.simulation.T, TIME.parar == 0)

      % EVENT CREATOR
      TIME_next();

      if hotstart == -1
         if and(TIME.t > EXEC.cutofftime, TIME.e > 5)
            TIME.parar = 1; TIME.delt = 0;
            TIME.cut = TIME.t;
            continue
         end
      end

      switch TIME.e

         % LAND
         % ------------------------------------------------------------
         case 1 % Arrival of EXP ET
            [ct] = CT_generate1('EXP');
            BF_dropCT('land',ct);

         case 2 % Arrival of IMP TRUCK
            BL_search_pick1();

         case 3 % Arrival of DUAL CT
            [ct] = CT_generate1('DUAL');
            BF_dropCT('land',ct);
            BL_search_pick1();

            % SEA
```

```
            % ---------------------------------------------------------
        case 4 % Arrival of IMP CT
            [ct] = CT_generate1('IMP');
            BF_dropCT('sea',ct);

        case 5 % Delivery of EXP CT
            %Sea_delivery3();
            Sea_delivery_tasks();
    end

    % CRANES
    % ---------------------------------------------------------------
    [orden] = ASC_interferencev2();

    ASC_progress(orden);
  end
end

%keyboard
if hotstart == -1
  TIME.parar = 0;

save(EXEC.statefilename,'ASC','BAYS','BF','BL','COST','COUNT','CT','INTERS','LIMITS','PLOT','PT','S','SEA_DELI
VERY','TIME','TRF');
else
  RESULTS
end

profile off

% Save workspace
if hotstart >= 1
  disp('Saving workspace')

  save(wsname);
end
```

```
% ASC MAIN PROGRAM

clear all
close all
clc
profile on

%matlabpool open 2

global ASC BAYS BF BL COUNT COST CT EXEC HK INTERS LIMITS PLOT PT  S SEA_DELIVERY TIME TRF

EXEC.cutofftime = 10*3600*24; % Change the value of days

caso = 1;  EXEC.caso = ['0' num2str(caso)];
TRF.PARAM.level = 40;

EXEC.statefilename = ['STATE CASE' char(EXEC.caso) 'TRF ' num2str(TRF.PARAM.level) '.mat'];
% Energy model: 1-Potential, 2-Electric
EXEC.energy_model = 1;
EXEC.plot = 0;

% Choose hotstart among three options:
% -1: run until cutoff and save a new initial state
%  0: complete execution
%  1: execute from cutoff
%  2: load previoustate (averias)

hotstart = 1;

if hotstart <= 0
   rng(caso);
   INIT(0) % 1 generates traffic, 0 does not generate
   TRF.PARAM.stackmode = 'psrandom';
   TRF.PARAM.inifill = 0; % -1: fill, 0: no fill and simulate, 1: load fill and simulate
   TRF.PARAM.baymaxocup = 60;

   blfilename = ['BL400901' num2str(TRF.PARAM.level) '.mat'];

   % 1. Create inifill during a number of days
   % ------------------------------------------------------------------------
   if TRF.PARAM.inifill == -1
      TIME.simulation.fill = fix(TRF.PARAM.level/100*BL.tiers*BL.bays*BL.stacks/2-TRF.PARAM.no.cts_imp/2);
   % 2. Do not create initial fill
   % ------------------------------------------------------------------------
   else
      TIME.simulation.fill = 0;
      % 2.1 Start simulation with empty block
      if TRF.PARAM.inifill == 0
         a = 0;
      % Start simulation with the block already occupied
      elseif TRF.PARAM.inifill == 1
         load(blfilename);  ct = COUNT.CT.no;
      end
```

```
   end
elseif hotstart == 1
   load(EXEC.statefilename);
   TIME.parar = 0;
   TRF.PARAM.stackmode = 'essa'; % tercat, essa, psrandom, crandom
   %--------------------------------------------------------------------
elseif hotstart == 2
   load('previoustate.mat');
end

% ----------------------------------------
TRF.PARAM.averias = 1;
TRF.PARAM.maxh = BL.tiers;
TRF.PARAM.weight.E = 0.5;
TRF.PARAM.weight.t = 1 - TRF.PARAM.weight.E;
% ----------------------------------------

% CREATE INITIAL FILL
if TRF.PARAM.inifill == -1
   BL_inifill(blfilename);

% SIMULATE
else
   while and(TIME.t <= TIME.simulation.T, TIME.parar == 0)

      % EVENT CREATOR
      TIME_next();

      if hotstart == -1
         if and(TIME.t > EXEC.cutofftime, TIME.e > 5)
            TIME.parar = 1; TIME.delt = 0;
            TIME.cut = TIME.t;
            continue
         end
      end

      switch TIME.e

         % LAND
         % ------------------------------------------------------------
         case 1 % Arrival of EXP ET
            [ct] = CT_generate1('EXP');
            BF_dropCT('land',ct);

         case 2 % Arrival of IMP TRUCK
            BL_search_pick1();

         case 3 % Arrival of DUAL CT
            [ct] = CT_generate1('DUAL');
            BF_dropCT('land',ct);
            BL_search_pick1();

            % SEA
            % ------------------------------------------------------
```

```
            case 4 % Arrival of IMP CT
                [ct] = CT_generate1('IMP');
                BF_dropCT('sea',ct);

            case 5 % Delivery of EXP CT
                %Sea_delivery3();
                Sea_delivery_tasks();
        end

        % CRANES
        % ---------------------------------------------------------------
        [orden] = ASC_interferencev2();

        ASC_progress(orden);
    end
end

%keyboard
if hotstart == -1
    TIME.parar = 0;

save(EXEC.statefilename,'ASC','BAYS','BF','BL','COST','COUNT','CT','INTERS','LIMITS','PLOT','PT','S','SEA_DELI
VERY','TIME','TRF');
else
    RESULTS
end

profile off

% Save workspace
if hotstart >= 1
    disp('Saving workspace')
    wsname = ['C' num2str(EXEC.caso) ' ' TRF.PARAM.stackmode ' TRF' num2str(TRF.PARAM.level) ' '
num2str(BL.bays) 'x' num2str(BL.stacks) '.mat' ];
    save(wsname);
end
```

```
function [asc] = ASC_move(asc,newposition,ct,writing)
% This function changes the position of the YC

global ASC

[pos]= length(asc.position.time);
%t_act = asc.tasks.time(asc.tasks.current);
t_act = asc.position.time(end);
t_min = min(asc.position.time(2:end));
if t_min ==0
   disp('Position Error')
   %keyboard
elseif newposition.time == 0
   disp('Position Error')
   %keyboard
end

asc.position.bay(end+1) = newposition.bay;
asc.position.stack(end+1) = newposition.stack;
asc.position.tier(end+1) = newposition.tier;
asc.position.time(end+1) = newposition.time;
asc.position.ct(end+1) = ct;

if writing ==1
   %keyboard
   switch char(asc.id)
      case 'sea'
         ASC.sea.position.bay(end+1) = newposition.bay;
         ASC.sea.position.stack(end+1) = newposition.stack;
         ASC.sea.position.tier(end+1) = newposition.tier;
         ASC.sea.position.time(end+1) = newposition.time;
         ASC.sea.position.ct(end+1) = ct;
      case 'land'
         ASC.land.position.bay(end+1) = newposition.bay;
         ASC.land.position.stack(end+1) = newposition.stack;
         ASC.land.position.tier(end+1) = newposition.tier;
         ASC.land.position.time(end+1) = newposition.time;
         ASC.land.position.ct(end+1) = ct;
   end
end
```

```
function [asc,ct,action,solution] = ASC_next_ct(asc,task)
% This function calculates the next CT to be serviced by the crane

global ASC BAYS BL CT

%keyboard

solution = 0;  %This is so that the while loop works ok. before: it = 0;
task = task - 1;
lt = length(asc.tasks.ct);

% By default, the next task is the first
if lt > 0
   asc.tasks.current = 1;
end

while and(solution == 0, task < lt)
   task = task + 1;
   ct = asc.tasks.ct(task);
   action = asc.tasks.action(task);
   % Check ct and get a new
   % Get a different container if that task can generate problems
   if ct > 0
      % first check if the other crane is currently doing that ct
      if strcmp(asc.id,'sea')== 1
         other_ct = ASC.land.ciclo.originalct;
      elseif strcmp(asc.id,'land')== 1
         other_ct = ASC.sea.ciclo.originalct;
      end
      if other_ct == ct
        continue
      end
      gs = CT(ct).position.gs(end);
      if strcmp(action, 'delivery') == 1
         %stopatct(ct,2343)
         bay = BL.GS(gs).bay;
         % Check if there are stack reservations
         sres = 0;
         for s = 1:BL.stacks
            tgs = BAYS(bay).GS(s);
            sres = sres+BL.GS(tgs).sreservations;
         end
         % Check if the other crane is operating in that bay
         conflict = BAY_prevent_delivery_conflict(ct,asc);
         if or(sres > 0, conflict == 1)
            solution = 0;
         else
            solution = 1;
         end

      elseif strcmp(action, 'stack') == 1
         solution = 1;
         %               keyboard
```

```
        %           if BL.GS(gs).Ldelreservations > 0
        %              solution = 0;
        %           elseif BL.GS(gs).Sdelreservations > 0
        %              solution = 0;
        %           else
        %              solution = 1;
        %           end
      end
   else
      disp('ASC workload error')
      keyboard
   end
end

if solution == 1
   if task > length(asc.tasks.ct)
      keyboard
   end
   if task > 1 % Then swap cts
      %keyboard % Correct the following line
      asc.tasks.current = task;
      disp(['ASC(' char(asc.id) ') Switch workload to task(' num2str(asc.tasks.current) ') CT('
num2str(asc.tasks.ct(asc.tasks.current)) ')'])
      ct = asc.tasks.ct(asc.tasks.current);
      action = asc.tasks.action(asc.tasks.current);
   end
else
   %keyboard
   %if it > 1
      %disp(['ASC(' char(asc.id) ') cannot change the WL order']); % keyboard
   %else
      %disp(['ASC(' char(asc.id) ') A normal task has not been found for crane to execute']);
   %end

   asc.tasks.current = task;
   asc.nextevent = 1000000;
   asc.status = 'wait';
   ct = 0;
   action = 'none';
end
```

```
function [delay,E] = ASC_pot_inter(asc,ciclo)
% This funciton checks the interference of land crane with respect to sea
% crane

global ASC BAYS BL CT S TIME

delay =0; E = 0;

% First point of the cycle trajectory:

xl = ASC_act_pos(ASC.land); XL(1) = xl.bay; TL(1) = xl.time;
xs = ASC_act_pos(ASC.sea);  XS(1) = xs.bay; TS(1) = xs.time;

delayfactor = 1.2;

% 1) GET THE ASC TAHT CORRESPONDS TO CICLO
% ASC is SEA, ciclo is land
if strcmp(asc.id,'sea') == 1
   Ltask = 1; %ciclo.c_task;
   LCbays = ciclo.bay(Ltask:end);
   LCtime = ciclo.time(Ltask:end);
   ALCtime = AUX_vect_ac(LCtime)+ TIME.t + TIME.delt;
   XL = [XL,LCbays];
   TL = [TL,ALCtime];
% ASC is LAND, ciclo is sea
elseif strcmp(asc.id,'land') == 1
   Stask = 1; %ciclo.c_task;
   SCbays = ciclo.bay(Stask:end);
   SCtime = ciclo.time(Stask:end);
   XS = [XS,SCbays];
   ts = AUX_vect_ac(SCtime) + TIME.t + TIME.delt;
   TS = [TS,ts];
end

% 2) GET THE CICLO FOR THE ASC
% Asc is SEA and
if strcmp(asc.id,'sea') == 1 % ASC is waiting.
   if strcmp(asc.status,'wait') == 1
      caso = 1;
      XS = ones(1,length(XL))*xs.bay;
      TS = [TS,TL(2:end)];
   elseif strcmp(asc.status,'wait') ~= 1
      %keyboard
      caso = 2;
      Stask = ASC.sea.ciclo.c_task;
      SCbays = asc.ciclo.bay(Stask:end);
      SCtime = asc.ciclo.time(Stask:end);
      XS = [XS,SCbays];
      ts = AUX_vect_ac(SCtime) + TIME.t + TIME.delt;
      TS = [TS,ts];
   end
% asc is LAND
elseif strcmp(asc.id,'land') == 1
```

```
   if strcmp(asc.status,'wait') == 1 % ASC is waiting
      %keyboard
      caso = 3;
      XL = ones(1,length(XS))*xl.bay;
      TL = [TL,TS(2:end)];
   elseif strcmp(asc.status,'wait') ~= 1
      %keyboard
      caso = 4;
      Ltask = asc.ciclo.c_task;
      LCbays = asc.ciclo.bay(Ltask:end);
      LCtime = asc.ciclo.time(Ltask:end);
      XL = [XL,LCbays];
      tl = AUX_vect_ac(LCtime) + TIME.t + TIME.delt;
      TL = [TL,tl];
   end
end

% 2. CHECK WHETHER THERE IS INTERSECTION
% --------------------------------------------------------------------
%[XSi,TSi] = AUX_time_filter(XS,TS,xs.time);
%[XLi,TLi] = AUX_time_filter(XL,TL,xl.time);

% Filter the vectors for intersection%keyboard
[XSo,TSo] = AUX_filter_repeated(XS,TS);
if length(XL) == 1
   keyboard
end
[XLo,TLo] = AUX_filter_repeated(XL,TL);
XSi = XSo + 1; XLi = XLo - 1;

if isempty(XSi) == 0
   %keyboard
   Ri = 0; Fi = 0; % Set real and ficticious intersections

   [xRi,yRi,iout,jout] = intersections(XSo,TSo,XLo,TLo);
   [xFi,yFi,iout,jout] = intersections(XSi,TSo,XLi,TLo);

   % Analize the real intersection
   [xRi0,yRi0,xFi0,yFi0,Ri,Fi,itype] = INTERSECTION_filter_points(xRi,yRi,xFi,yFi,0,XSo,TSo,XLo,TLo,XSi,XLi);
end

% 3. CALCULATE THE TYPE OF INTERSECTION
% -----------------------------------------

if yFi0 > 0

   % 3.1 Plot the intersection
   plot_intersection(XS,TS,XL,TL,Ri,xRi0,yRi0,Fi,xFi0,yFi0);

   % 3.2 Get some intersection particulars for later

   [Lintslope]= ASC_way(xFi0,yFi0,XLi,TLo); %(XLo(2)-XLo(1))/(TLo(2)-TLo(1));
   [Sintslope]= ASC_way(xFi0,yFi0,XSi,TSo); %(XSo(2)-XSo(1))/(TSo(2)-TSo(1));
```

```
   % 3.3 Check a particular case of error
   if length(yFi) == 2
      if Sintslope + Lintslope ~= 0
         if abs(yFi(1)-yFi(2))/10000 > 0.2
            keyboard
         end
      end
   end

   % 3.4 Type of intersection
   % ----------------------------------------------------------------
   if strcmp(asc.status , 'wait') == 1 % THE CRANE IS WAITING
      if strcmp(asc.id,'sea') == 1 % SEA crane waiting
         escenario = 1; matrizcaso = 11; orden = 'L';
      elseif strcmp(asc.id,'land') == 1 % LAND crane waiting
         escenario = 2; matrizcaso = 12; orden = 'S';
      end
   else % THE CRANE IS ALREADY IN A CYCLE
      %keyboard
      if asc.ciclo.no > 0
         if strcmp(asc.id,'sea') == 1
            escenario = 4;
         elseif strcmp(asc.id,'land') == 1
            commenced = 0;
            for t = 1:asc.ciclo.c_task
               move = asc.ciclo.moves(t);
               if or(strcmp(move,'pickbf') == 1, strcmp(move,'pickbl') ==1)
                  commenced = 1;
               end
            end
            if commenced == 0
               escenario = 4; % SEA crane priority
            else
               escenario = 3; % LAND crane priority
            end
         end
      elseif asc.ciclo.no == 0 % None of the cranes has commenced cycle
         escenario = 4; % SEA crane priority
%        else
%           if ASC.sea.nextevent <= ASC.land.nextevent
%              escenario = 4;
%           else
%              escenario = 3;
%           end
      end
%        if Ri > 0
%           keyboard
%           [nLi,Lside] = INTERS_crosses(XL,TL,xRi0,yRi0);
%           [nSi,Sside] = INTERS_crosses(XS,TS,xRi0,yRi0);
%           [escenario,matrizcaso,orden] = INTERSECTION_escenario(Sintslope,Lintslope,XL,XS,nSi,nLi);
%        elseif Fi > 0
%           keyboard
%           [nLi,Lside] = INTERS_crosses(XL,TL,xFi0,yFi0);
%           [nSi,Sside] = INTERS_crosses(XS,TS,xFi0,yFi0);
```

```
%           [escenario,matrizcaso,orden] = INTERS_fict_escenario(itype,Sside,Lside,nSi,nLi);
%        else
%           keyboard
%        end
   end
   % 4 CALCULATION of intersection
   % ---------------------------------------------------------------
   addSdelay = 0; addLdelay = 0; bayshift = 0;
   switch escenario
      case 1    % SEA waits, The LAND CRANE HAS PRIORITY
         % ---------------------------------------------------------------
         %keyboard
         [PSea] = ASC_act_pos(ASC.sea);
         destiny.bay = min(ciclo.bay) - 3;
         destiny.stack = PSea.stack;
         destiny.tier = BL.tiers+1;
         if destiny.bay > 0
            destiny.gs = BAYS(destiny.bay).GS(destiny.stack);
         else
            destiny.gs = 0;
         end
         [delay,E] = ASC_energy(PSea.bay,destiny.bay,0,'gantry','empty');
         delay = 0;
         %disp(['ASC(sea) Potential Interference while idle. ASC(land) Delay: ' num2str(ceil(delay))])

      case 2    % The SEA CRANE HAS PRIORITY
         % ---------------------------------------------------------------
         %keyboard
         [PLand] = ASC_act_pos(ASC.land);
         destiny.bay = max(ciclo.bay) + 3;
         destiny.stack = PLand.stack;
         destiny.tier = BL.tiers+1;
         if destiny.bay <= BL.bays
            destiny.gs = BAYS(destiny.bay).GS(destiny.stack);
         else
            destiny.gs = 0;
         end
         [delay,E] = ASC_energy(PLand.bay,destiny.bay,0,'gantry','empty');
         delay = 0;
         %disp(['ASC(land) potentail Interference while idle.  ASC(sea) Delay: ' num2str(ceil(delay))])

      case 3    % The LAND crane has priority
         % ---------------------------------------------------------------
         % 1. Determine the final point of the delay
         % keyboard
         lowbay = min(asc.ciclo.bay);
         [t_task] = AUX_find_vector_pos(LCbays,lowbay); t_fin = TL(t_task+1);
         plot(t_fin/3600/24,lowbay,'<r');

         % 2. Determine the INITIAL point of the delay
         % 2.1 No real intersection
         % ---------------------------------------------------------
         if Ri == 0
            t_ini = yFi0;
```

```
    % 2.2 Plane intersection
    % ---------------------------------------------------------
    elseif length(xRi) > 2
        %keyboard
        t_ini = yFi0;

    % 2.3 Normal intersection
    % ---------------------------------------------------------
    else
        [bayant,baypost] = AUX_pointinvector(XS,lowbay); %SCbays
        % Simple interserctions
        if size(bayant) == 1
            % None intersection case
            if bayant + baypost == 0
                t_ini = yFi0; inters = 0; keyboard % Step(709) wrong
            else
                inters = 1; a = bayant(1); b = baypost(1);
            end
            % Multiple intersections
        else
            inters = 1;
            if strcmp(ASC.sea.status,'stack') == 1 % Case 4 8 land stack
                a = bayant(2);
                b = baypost(2);
            else
                %keyboard
                a = bayant(1); % Case 8
                b = baypost(1);
            end
        end
        if inters == 1
            VX(1) = XS(a); VX(2) = XS(b); % +1
            VT(1) = TS(a); VT(2) = TS(b); %+1
            plot(VT/3600/24,VX,'-.m')
            if VX(1) == VX(2)
                t_ini= VT(1);
            else
                t_ini = interp1(VX,VT,lowbay);
            end
        end
    end
    plot(t_ini/3600/24,lowbay,'>r')

    % Calculate the delay
    delay = ceil(( t_fin - t_ini) * delayfactor); % We give a margin of 20 sec
    % keyboard
    delay = max(delay, 20);

    % ---------------------------------------------------------
case 4    % The SEA CRANE has priority. Introduce delay the land crane
    % ---------------------------------------------------------
    %keyboard
    upbay = max(SCbays);
```

```
        [t_task] = AUX_find_vector_pos(SCbays,upbay); t_fin = TS(t_task+1);
        plot(t_fin/3600/24,upbay,'<r');
        %keyboard

        % Find the intersection of the cranes
        if Ri == 0
           t_ini = yFi0;
        elseif Ri == 1
           %keyboard
           [bayant,baypost] = AUX_pointinvector(XL,upbay); % LCbays
           % Simple intersection
           if size(bayant) == 1
              if bayant+baypost == 0
                 t_ini = yFi0; inters = 0; keyboard % Case 1
              else
                 a = bayant(1); b = baypost(1); inters = 1;
              end
           else
              % Multiple intersections
              inters = 1;
              % Case 4 land stack
              if strcmp(ASC.land.status,'stack') == 1
                 a = bayant(2);
                 b = baypost(2);
              else
                 % Case 4 failed.
                 % Case 7 failed did not add the delay to the
                 % task
                 %keyboard
                 a = bayant(1);
                 b = baypost(1);
              end
           end

           if inters == 1
              VX(1) = XL(a); VX(2) = XL(b); % +1
              VT(1) = TL(a); VT(2) = TL(b); % +1
              plot(VT/3600/24,VX,'-.m')

              if VX(1) == VX(2)
                 t_ini= VT(1);
              else
                 t_ini = interp1(VX,VT,upbay);
              end
           end
        end

        plot(t_ini/3600/24,upbay,'>r')
        delay = ceil((t_fin -t_ini) * delayfactor); % We give a margin of 15 sec
        delay = max(delay, 20); Pupbay = upbay;

     end
  end
```

```
function [delay,E] = ASC_pot_interv2(asc,ciclo)
% This funciton checks the interference of land crane with respect to sea
% crane

global ASC BAYS BL CT S TIME

delay =0; E = 0;

% First point of the cycle trajectory:

xl = ASC_act_pos(ASC.land); XL(1) = xl.bay; TL(1) = xl.time;
xs = ASC_act_pos(ASC.sea);  XS(1) = xs.bay; TS(1) = xs.time;

delayfactor = 1.2;

% 1) GET THE ASC TAHT CORRESPONDS TO CICLO
% ASC is SEA, ciclo is land
if strcmp(asc.id,'sea') == 1
   Ltask = 1; %ciclo.c_task;
   LCbays = ciclo.bay(Ltask:end);
   LCtime = ciclo.time(Ltask:end);
   ALCtime = AUX_vect_ac(LCtime)+ TIME.t + TIME.delt;
   XL = [XL,LCbays];
   TL = [TL,ALCtime];
% ASC is LAND, ciclo is sea
elseif strcmp(asc.id,'land') == 1
   Stask = 1; %ciclo.c_task;
   SCbays = ciclo.bay(Stask:end);
   SCtime = ciclo.time(Stask:end);
   XS = [XS,SCbays];
   ts = AUX_vect_ac(SCtime) + TIME.t + TIME.delt;
   TS = [TS,ts];
end

% 2) GET THE CICLO FOR THE ASC
% Asc is SEA and
if strcmp(asc.id,'sea') == 1 % ASC is waiting.
   if strcmp(asc.status,'wait') == 1
      caso = 1;
      XS = ones(1,length(XL))*xs.bay;
      TS = [TS,TL(2:end)];
   elseif strcmp(asc.status,'wait') ~= 1
      %keyboard
      caso = 2;
      Stask = ASC.sea.ciclo.c_task;
      SCbays = asc.ciclo.bay(Stask:end);
      SCtime = asc.ciclo.time(Stask:end);
      XS = [XS,SCbays];
      ts = AUX_vect_ac(SCtime) + TIME.t + TIME.delt;
      TS = [TS,ts];
   end
% asc is LAND
elseif strcmp(asc.id,'land') == 1
```

```
   if strcmp(asc.status,'wait') == 1 % ASC is waiting
      %keyboard
      caso = 3;
      XL = ones(1,length(XS))*xl.bay;
      TL = [TL,TS(2:end)];
   elseif strcmp(asc.status,'wait') ~= 1
      %keyboard
      caso = 4;
      Ltask = asc.ciclo.c_task;
      LCbays = asc.ciclo.bay(Ltask:end);
      LCtime = asc.ciclo.time(Ltask:end);
      XL = [XL,LCbays];
      tl = AUX_vect_ac(LCtime) + TIME.t + TIME.delt;
      TL = [TL,tl];
   end
end
% Filter the vectors for intersection
[XS,TS] = AUX_filter_repeated(XS,TS);
[XL,TL] = AUX_filter_repeated(XL,TL);

% 2. See if there is intersection and get the point
if length(XS) > 1
   [Ri,xi,yi] = INTERS_exist(XS,TS,XL,TL,S.baymargin);
else
  Ri = 0;
end

% 3. CALCULATE THE TYPE OF INTERSECTION
% -----------------------------------------
if Ri == 1
   %keyboard
   if strcmp(ASC.sea.status , 'wait') == 1
      escenario = 1; orden = 'L';
   elseif strcmp(ASC.land.status , 'wait') == 1
      escenario = 2; orden = 'S';
   else % THE CRANES ARE ALREADY IN A CYCLE
      [escenario,orden] = INTERSECTION_set_escenario(XL,XS,TL,TS,xi,yi);

      keyboard
      if asc.ciclo.no > 0
         if strcmp(asc.id,'sea') == 1
            escenario = 4;
         elseif strcmp(asc.id,'land') == 1
            commenced = 0;
            for t = 1:asc.ciclo.c_task
               move = asc.ciclo.moves(t);
               if or(strcmp(move,'pickbf') == 1, strcmp(move,'pickbl') ==1)
                  commenced = 1;
               end
            end
            if commenced == 0
               escenario = 4; % SEA crane priority
            else
               escenario = 3; % LAND crane priority
```

```
        end
      end
   elseif asc.ciclo.no == 0 % None of the cranes has commenced cycle
      escenario = 4; % SEA crane priority
   end
end

% 4 CALCULATION of intersection
% ---------------------------------------------------------
addSdelay = 0; addLdelay = 0; bayshift = 0;
switch escenario
   case 1    % SEA waits, The LAND CRANE HAS PRIORITY
      % ---------------------------------------------------------
      %keyboard
      [PSea] = ASC_act_pos(ASC.sea);
      destiny.bay = min(ciclo.bay) - 3;
      destiny.stack = PSea.stack;
      destiny.tier = BL.tiers+1;
      if destiny.bay > 0
         destiny.gs = BAYS(destiny.bay).GS(destiny.stack);
      else
         destiny.gs = 0;
      end
      [delay,E] = ASC_energy(PSea.bay,destiny.bay,0,'gantry','empty',1);
      delay = 0;
      %disp(['ASC(sea) Potential Interference while idle. ASC(land) Delay: ' num2str(ceil(delay))])

   case 2    % The SEA CRANE HAS PRIORITY
      % ---------------------------------------------------------
      %keyboard
      [PLand] = ASC_act_pos(ASC.land);
      destiny.bay = max(ciclo.bay) + 3;
      destiny.stack = PLand.stack;
      destiny.tier = BL.tiers+1;
      if destiny.bay <= BL.bays
         destiny.gs = BAYS(destiny.bay).GS(destiny.stack);
      else
         destiny.gs = 0;
      end
      [delay,E] = ASC_energy(PLand.bay,destiny.bay,0,'gantry','empty',1);
      delay = 0;
      %disp(['ASC(land) potentail Interference while idle.  ASC(sea) Delay: ' num2str(ceil(delay))])

   case 3    % The LAND crane has priority
      % ---------------------------------------------------------
      % 1. Determine the final point of the delay
      keyboard
      lowbay = min(asc.ciclo.bay);
      [t_task] = AUX_find_vector_pos(LCbays,lowbay); t_fin = TL(t_task+1);
      plot(t_fin/3600/24,lowbay,'<r');

      % 2. Determine the INITIAL point of the delay
      % 2.1 No real intersection
      % -------------------------------------------------------
```

```
    if Ri == 0
        t_ini = yFi0;

        % 2.2 Plane intersection
        % ---------------------------------------------------------
    elseif length(xRi) > 2
        %keyboard
        t_ini = yFi0;

        % 2.3 Normal intersection
        % ---------------------------------------------------------
    else
        [bayant,baypost] = AUX_pointinvector(XS,lowbay); %SCbays
        % Simple interserctions
        if size(bayant) == 1
            % None intersection case
            if bayant + baypost == 0
                t_ini = yFi0; inters = 0; keyboard % Step(709) wrong
            else
                inters = 1; a = bayant(1); b = baypost(1);
            end
            % Multiple intersections
        else
            inters = 1;
            if strcmp(ASC.sea.status,'stack') == 1 % Case 4 8 land stack
                a = bayant(2);
                b = baypost(2);
            else
                %keyboard
                a = bayant(1); % Case 8
                b = baypost(1);
            end
        end
        if inters == 1
            VX(1) = XS(a); VX(2) = XS(b); % +1
            VT(1) = TS(a); VT(2) = TS(b); %+1
            plot(VT/3600/24,VX,'-.m')
            if VX(1) == VX(2)
                t_ini= VT(1);
            else
                t_ini = interp1(VX,VT,lowbay);
            end
        end
    end
    plot(t_ini/3600/24,lowbay,'>r')

    % Calculate the delay
    delay = ceil(( t_fin - t_ini) * delayfactor); % We give a margin of 20 sec
    % keyboard
    delay = max(delay, 20);

    % ---------------------------------------------------------
case 4    % The SEA CRANE has priority. Introduce delay the land crane
    % ---------------------------------------------------------
```

```
        keyboard
        upbay = max(SCbays);
        [t_task] = AUX_find_vector_pos(SCbays,upbay); t_fin = TS(t_task+1);
        plot(t_fin/3600/24,upbay,'<r');
        %keyboard

        % Find the intersection of the cranes
        if Ri == 0
           t_ini = yFi0;
        elseif Ri == 1
           %keyboard
           [bayant,baypost] = AUX_pointinvector(XL,upbay); % LCbays
           % Simple intersection
           if size(bayant) == 1
              if bayant+baypost == 0
                 t_ini = yFi0; inters = 0; keyboard % Case 1
              else
                 a = bayant(1); b = baypost(1); inters = 1;
              end
           else
              % Multiple intersections
              inters = 1;
              % Case 4 land stack
              if strcmp(ASC.land.status,'stack') == 1
                 a = bayant(2);
                 b = baypost(2);
              else
                 % Case 4 failed.
                 % Case 7 failed did not add the delay to the
                 % task
                 %keyboard
                 a = bayant(1);
                 b = baypost(1);
              end
           end

           if inters == 1
              VX(1) = XL(a); VX(2) = XL(b); % +1
              VT(1) = TL(a); VT(2) = TL(b); % +1
              plot(VT/3600/24,VX,'-.m')

              if VX(1) == VX(2)
                 t_ini= VT(1);
              else
                 t_ini = interp1(VX,VT,upbay);
              end
           end
        end

        plot(t_ini/3600/24,upbay,'>r')
        delay = ceil((t_fin -t_ini) * delayfactor); % We give a margin of 15 sec
        delay = max(delay, 20); Pupbay = upbay;
     end
  end
```

```
function ASC_progress(orden)

% This function carries out the movement of the asc and the positioning of
% cts in the yard
% All the functions in this SR must not contain the copy/write ASC statements

global ASC COUNT S TIME

if strcmp(orden,'S') == 1 % THE SEA CRANE MOVES FIRST
   if strcmp(ASC.sea.status,'wait') == 0
      %[ASC.sea] = ASC_cycle_check(ASC.sea);
      [ASC.sea]= ASC_cycle(ASC.sea);
   end

   if strcmp(ASC.land.status,'wait') == 0
      %[ASC.land] = ASC_cycle_check(ASC.land);
      [ASC.land]= ASC_cycle(ASC.land);
   end
elseif strcmp(orden, 'L') == 1 % THE LAND CRANE MOVES FIRST
   if strcmp(ASC.land.status,'wait') == 0
      %[ASC.land] = ASC_cycle_check(ASC.land);
      [ASC.land]= ASC_cycle(ASC.land);
   end
   if strcmp(ASC.sea.status,'wait') == 0
      %[ASC.sea] = ASC_cycle_check(ASC.sea);
      [ASC.sea]= ASC_cycle(ASC.sea);
   end
end

% Update the position of the crane
if strcmp(ASC.sea.status,'wait') == 1
   % keyboard
   ASC_catch(ASC.sea);
end

if strcmp(ASC.land.status,'wait') == 1
   %keyboard
   ASC_catch(ASC.land);
end

Lpos = ASC_act_pos(ASC.land);
Spos = ASC_act_pos(ASC.sea);

if abs(TIME.t + TIME.delt - Lpos.time) > 0
   disp('Time problem')
   ASC.land.position.time(end) = TIME.t + TIME.delt;
   keyboard
end

if abs(TIME.t + TIME.delt - Spos.time) > 0
   disp('Time problem')
   ASC.sea.position.time(end) = TIME.t + TIME.delt;
   keyboard
```

```
end

if Lpos.bay <= Spos.bay
   disp('ASC progres Error: ASC colliding')
   plot_ASC_trajectories(20); title('ASC colliding')
   keyboard
elseif Lpos.bay - Spos.bay < S.baymargin
   disp('ASC progres Warning: ASC Approaching')
   %keyboard
end

ASC_wl(ASC.sea,COUNT.CT.no);
if TIME.delt < 0
   disp(['Time delt error' num2str(TIME.delt)])
   keyboard
end
```

```
function [compat] = ASC_reshuffle_prevent2(asc,t_bay)
% This function checks whether the other crane is making rehandles in a bay
% so we don't use that bay to place a ct

global BL CT

compat = 1;

if strcmp(asc.status,'wait') == 0
   if asc.ciclo.nr > 0
      if asc.ciclo.destiny.gs == 0
         keyboard
      end
      workbay = BL.GS(asc.ciclo.destiny.gs).bay;
      if workbay == t_bay
         compat = 0;
      end
   end
end

% Check the following task

if length(asc.tasks.ct) > 1
   %keyboard
   ft = asc.tasks.current + 1;
   if ft > length(asc.tasks.ct)
      %keyboard
      %disp('Change this to evaluate the previous tasks')
   else
      if strcmp(asc.tasks.action(ft),'delivery') == 1
         ct = asc.tasks.ct(ft);
         workbay2 = CT(ct).position.bay(end);
         if workbay2 == t_bay
            compat = 0;
         end
      end
   end
end
```

```
function [asc] = ASC_retard(asc,baystop,endtime,guilty_ct,Xpref,Xret,Tpref,Tret,xi,yi)
% This function adds a delay to the crane cycle
global S TIME

% Get the index of the nearest gantry task
int_t = yi; seed = 0;

[int_task] = ASC_get_gantry_task(asc,int_t);

% Find the position of the asc before it started gantrying
if int_task == 1
   inibay = asc.ciclo.ascposition.bay;
elseif int_task > 1
   inibay = asc.ciclo.bay(int_task-1); %keyboard
end

plot(endtime/3600/24,baystop,'<r'); %baystop-S.baymargin
% 1 Modify first cycle task
% 1.1 Make a copy of the gantry task
ct = asc.ciclo.ct(int_task);
tier = asc.ciclo.tier(int_task);
stack = asc.ciclo.stack(int_task);
tbay = asc.ciclo.bay(int_task);
time0 = asc.ciclo.time(int_task);
time1 = asc.ciclo.originaltime(int_task);
time1passed = time1 - time0;

% 1.2 We make the crane to go slower to the target
[time1b,E1] = ASC_energy(inibay,baystop,0,'gantry','empty'); %keyboard
[time3,E2] = ASC_energy(baystop,tbay,0,'gantry','empty');

% Find the positions where there is gantry
j = 0; %keyboard
for i=1:length(Xret)-1
   if abs(Xret(i)-Xret(i+1)) >0
      if and(Tret(i)<yi, yi<Tret(i+1))
         ix1 = i;
      end
   end
end

if isempty(ix1)
   keyboard
end
ix2= ix1+1; ix3 = ix1+2;

if time1passed >0
   %keyboard
   delbay = (time1passed/time1b)*(baystop-inibay); %keyboard
   asc.position.bay(end) = inibay+delbay;
   % Change the starting point of the asc trajectory
   Tret(ix1) = TIME.t+time1b-time1passed;
end
```

```
if ix1>1
   ix1 = ix1+1; ix2 = ix2+1; ix3 = ix3+1; %keyboard;
   Tret(ix1) = Tret(ix1-1) + time1b - time1passed;
else
   %keyboard
   Tret(ix1) = max(TIME.t+time1b-time1passed,TIME.t);
end
Xret(ix1) = baystop; plot(Tret(ix1)/3600/24,baystop,'o')

% Change the second point of the trajectory
Xret(ix2) = tbay;
Tret(ix3:end) = Tret(ix3:end)+time3-(Tret(ix2)-Tret(ix1));
Tret(ix2) = Tret(ix1)+time3;
if Tret(ix3) < Tret(ix2)
  keyboard
end

% Add a stop to the asc trajectory
Xret = vect_insert(Xret,baystop,ix2);
newt = Tret(ix1);
if newt < Tret(ix1)
   keyboard
end
Tret = vect_insert(Tret,newt,ix2);plot(Tret/3600/24,Xret,'c')
if strcmp(asc.id,'land') == 1
   delbay = - S.baymargin+0.001; checkasc= 6;
elseif strcmp(asc.id,'sea') == 1
   delbay = S.baymargin-0.001;  checkasc= 7;
end
[x,y,iout,jout] = intersections(Xret+delbay,Tret,Xpref,Tpref);
time2 = 0; deltime = 5; cont = 0;
oldx = x; oldy = y;
%keyboard
while length(y)>0
   cont = cont + 1;
   if time2 == 0
      Xret = vect_insert(Xret,baystop,ix3);
      Tret = vect_insert(Tret,newt,ix3);
      time2 = time2 + deltime;
   end
   Tret(ix3:end) = Tret(ix3:end) + deltime;
   [fTret,fXret] = AUX_filtrar(Tret,Xret) ;Xfict = fXret+delbay;
   [fTpref,fXpref] = AUX_filtrar(Tpref,Xpref);
   %figure(4);hold on; plot(fTret,Xfict,'.-',fTpref,fXpref,'.-')
   [x,y,jout,iout] = intersections(Xfict,fTret,fXpref,fTpref);
   plot(Tret/3600/24,Xret,'r'); plot(y/3600/24,x,'*r')

   % Check whether the intersection is the same despite the iteration
   if and(isequal(oldx,x), isequal(oldy,y))
      % keyboard
      if and(y >0, length(y) ==1)
         if length(asc.ciclo.originaltime) >1
            ntask = ASC_get_gantry_task(asc,y);
```

```
            if int_task == ntask
               y = [];
            end
         else
            keyboard
         end
      end
   end
   oldx = x; oldy = y; old_lengthy = length(y);

   time2 = time2 + deltime;
   if cont > 200
      disp('El cálculo de la intersección entra en bucle');
      figure; grid on; plot(Tret,Xret,fTret,Xfict,'g',Tpref,Xpref);
      hold on; plot(y,x,'*');
      seed = 1;
      y = []; % So there is intersection no more
   end
end

if seed == 1
   if TIME.e == checkasc
      if time2 + yi < TIME.t + TIME.delt
         %keyboard
         time2 = TIME.t + TIME.delt - yi;
      end
   end
end

tdir = tbay - inibay;   ndir = baystop-inibay;
if tdir*ndir >= 0
   gantry1 = 'gantry';
   gantry2 = 'gantry';
elseif tdir*ndir < 0
   gantry1 = 'ugantry';%keyboard % step 455:656: ok
   gantry2 = 'gantry';
end

% 1.3 Modify the first task of the cycle
% a) If the gantry has commenced, we need to modify it
asc.ciclo.bay(int_task) = baystop;
asc.ciclo.originaltime(int_task) = time1b;
asc.ciclo.moves{int_task} = gantry1;
asc.ciclo.E(int_task) = E1;
if time1passed > 0
   if time1passed > time1b % 1st gantry is over
      asc.ciclo.time(int_task) = 0; %keyboard
   else
      asc.ciclo.time(int_task) = time1b-time1passed;
   end
else
   asc.ciclo.time(int_task) = time1b-time1passed;
end
```

```
% 2. Modify the second position by adding wait
% --------------------------------------------------------------------
if time2 > 0
   int_task = int_task +1;
   time2 = ceil(time2);
   [asc] = ASC_cycle_insert(asc,baystop,stack,tier,time2,0,'wait',guilty_ct,ct,int_task);
end

% 3. Add a third gantry to the ct position
% --------------------------------------------------------------------
if time3> 0
   int_task = int_task +1;
   if int_task > length(asc.ciclo.bay)
      asc.ciclo
      keyboard
   end
   [asc] = ASC_cycle_insert(asc,tbay,stack,tier,time3,E2,gantry2,ct,ct,int_task);
end

v = AUX_vect_ac(asc.ciclo.time);
plot((v+TIME.t)/3600/24,asc.ciclo.bay,'.-m')
asc.nextevent = sum(asc.ciclo.time); %asc.nextevent -time1 + time1b + time2 + time3;
%    if abs(sum(asc.ciclo.time)-asc.nextevent) > 0
%       keyboard
%       asc.nextevent = sum(asc.ciclo.time);
%    end
```

```
function [destiny] = ASC_shift(asc,TL)

global ASC BAYS BL CT
ascpos = ASC_act_pos(asc);
if strcmp(asc.id,'land') == 1
   destiny.bay = max(ASC.sea.ciclo.bay) + 4;
elseif strcmp(asc.id,'sea') == 1
   destiny.bay = max(ASC.land.ciclo.bay) - 4;
end

destiny.stack = ascpos.stack;
destiny.tier = BL.tiers+1;
destiny.gs = BAYS(destiny.bay).GS(destiny.stack);
[delay,E] = ASC_energy(ascpos.bay,destiny.bay,0,'gantry','empty');
%keyboard
ct = ASC.sea.tasks.ct(end);
CT(ct).energy = CT(ct).energy + E/1000000;
delay = max(delay, 25);
destiny.time = TL(1) + ceil(delay);
```

```
function [bay] = ASC_target_bay(asc)


if or(strcmp(asc.status,'stack') == 1, strcmp(asc.status,'housekeeping') == 1)
    bay = asc.ciclo.destiny.bay;
elseif strcmp(asc.status,'delivery') == 1
    bay = asc.ciclo.origin.bay;
elseif strcmp(asc.status,'trans') == 1
    bay = asc.ciclo.bay(1); %keyboard
elseif strcmp(asc.id,'sea') == 1
    bay = -10; keyboard
elseif strcmp(asc.id,'land') == 1
    bay = 50; keyboard
else
    keyboard
end
```

```
function [tasks] = ASC_task_remove(tasks,current)

% This function removes the current task from the ASC workload

if current == 0
    keyboard
    ctask = tasks.current;
else
    ctask = current;
end
%keyboard
tasks.action(ctask) = [];
tasks.ct(ctask) = [];
tasks.ct_id(ctask) = [];
tasks.time(ctask) = [];
tasks.current = 0;
```

```
function ASC_tasks_init()
% this function resets the list of tasks
global ASC

ASC.hktasks.action = {};
ASC.hktasks.ct = [];
ASC.hktasks.ct_id = {};
ASC.hktasks.time = [];
ASC.hktasks.current = 0;
ASC.hktasks.priority = 0;
```

```
function ASC_traj_histograms()

global BL ASC

disct = 1;
% Sea
%-------------------------------------------------------------------------
[Bs,Ts,a] = AUX_filter_repeated(ASC.sea.position.bay,ASC.sea.position.time);
malos = find(ASC.sea.position.bay <-2);
if isempty(malos) == 0
   Bs(malos) = []; Ts(malos) = [];
end
% Discretize axis every "disct" seconds
nT = 0:disct:ASC.sea.position.time(end);
NBs = interp1(Ts,Bs,nT,'linear');
hs =zeros(1,BL.bays+6);
hs = hist(NBs,BL.bays+3); ps = -2:BL.bays;
hs = 100*hs/sum(hs);
% Land
%-------------------------------------------------------------------------
[Bl,Tl,a] = AUX_filter_repeated(ASC.land.position.bay,ASC.land.position.time);
malos = find(ASC.land.position.bay >43);
if isempty(malos) == 0
   Bl(malos) = []; Tl(malos) = [];
end

% Discretize axis every "disct" seconds
nT = 0:disct:ASC.land.position.time(end);

NBl = interp1(Tl,Bl,nT,'linear');
hl =zeros(1,BL.bays+6);
hl = hist(NBl,BL.bays+3); pl = 1:(BL.bays+3);
hl = 100*hl/sum(hl);
k = length(hl);
% Plot the histograms
figure; plot(ps(4:k),hs(4:k)); hold on; plot(pl(1:k-3),hl(1:k-3),'g');
title('ASC position histograms')
xlabel('Bays'); ylabel('Probability (%)')
```

```
function [C] = ASC_trajectory(a,s,travelt,full_speed)
% This function calculates the trajectory of a crane

NP = 100;
C.t = travelt/NP:travelt/NP:travelt;

if full_speed == 1
   i = 1; v = 0;
   C.x(1) = 0.5*a*C.t(i)^2;

   while v < s
      i = i+1;
      C.x(i) = 0.5*a*C.t(i)^2;
      v = a*C.t(i);
      k = NP-i+1;
   end
   for j = i+1:NP-i
      C.x(j) = C.x(i) + s*(C.t(j)-C.t(i));
   end

   for i = j +1: NP
      t = C.t(i)-C.t(j);
      C.x(i) = C.x(j)+s*t-0.5*a*t^2;
   end
else
   for i=1:NP/2
      C.x(i) = 0.5*a*C.t(i)^2;
      C.x(NP-i+1) = C.x(i);
   end
end

% figure; plot(C.t,C.x)
```

```
function [asc]= ASC_trans(asc)
% This function simply moves the crane to another location

keyboard

P = ASC_act_pos(asc);

[delt,E] = ASC_energy(asc.target.bay,P.bay,0,'gantry','empty');
P.time = P.time + delt(m); P.bay = origin.bay;
[asc] = ASC_move(asc,P,0,1);

asc.tasks.executed(task_no) = ct;
asc.nextevent = 1000000;
asc.status = 'done';
```

```
function [ACTIVE,IDLE] = ASC_trans_unprod(ACTIVE,IDLE,Tidle)

global BAYS BL S

% The LAND CRANE HAS PRIORITY (ACTIVE) and SEA CRANE is waiting (IDLE)
% The idle crane is moved (translated) to another position

% 1. Calculate the destination position for the idle crane
[Pidle] = ASC_act_pos(IDLE);
if strcmp(ACTIVE.id, 'land') == 1
   destiny.bay = min(ACTIVE.ciclo.bay) - S.baymargin;
else
   destiny.bay = max(ACTIVE.ciclo.bay) + S.baymargin;
end

destiny.stack = Pidle.stack;
destiny.tier = BL.tiers + 1;

if strcmp(ACTIVE.id, 'land') == 1
   if destiny.bay > 0
      destiny.gs = BAYS(destiny.bay).GS(destiny.stack);
   else
      destiny.gs = 0;
   end
else
   if destiny.bay < BL.bays
      destiny.gs = BAYS(destiny.bay).GS(destiny.stack);
   else
      destiny.gs = 0;
   end
end

% 2. Calculate the necessary energy and time for the idel crane to move
[delay,E] = ASC_energy(Pidle.bay,destiny.bay,0,'gantry','empty');

% Assign the unproductive energy consumption to the active ct
% if strcmp(ACTIVE.status,'housekeeping')==0
%     ctask = ACTIVE.tasks.current; ct = ACTIVE.tasks.ct(ctask);
% else
%     ctask = ACTIVE.hktasks.current; ct = ACTIVE.hktasks.ct(ctask); %keyboard
% end
ct = ACTIVE.ciclo.originalct;

CT_write_cycle(ct,E,'trans',delay);

destiny.time = Tidle(1) + ceil(delay); plot(destiny.time/3600/24,destiny.bay,'m*');
% generate a new cycle for the crane
if strcmp(IDLE.status,'wait') == 0
   keyboard
end

[IDLE] = ASC_addcycle(IDLE,destiny,delay,ct);
%[IDLE] = ASC_add_priortask(IDLE,0,'trans');
```

```
%keyboard
%[IDLE] = ASC_addtask(IDLE,0,'trans',1);

disp(['Idle ASC(' char(IDLE.id)  ') Interferes with active ASC(' char(ACTIVE.id)  ') CT(' num2str(ct) ') Duration of displacing: ' num2str(ceil(delay))])
```

```
function [slope] = ASC_way(x,t,X,T)

% get the position of x with respect to X

ini = 1; fin = 2; found = 0;

for p = 1: length(X)-1

    if and( T(p)<t , T(p+1)>=t )
        ini = p; fin = p+1;
        found = 1;
    elseif found == 0
        ini = p; fin = p +1;
    end
end

delx = X(fin) - X(ini);
delt = T(fin) - T(ini);
slope = delx/delt;
```

```
function [n,lista] = ASC_wl(asc,ct)
% This function checks for repeated cts in the WL

global ASC CT

n = 0; lista = 0;

if strcmp(asc.status,'housekeeping') == 1
   tasklist = ASC.hktasks;
else
   tasklist = asc.tasks;
end

if ct >0
   r_cts = find(tasklist.ct == ct);

   if isempty(r_cts) > 1
      disp(['ASC workload error: CT(' num2str(ct) ') repeated at positions: ' num2str(r_cts)])
      disp(['CT Movements ' tasklist.action(r_cts)])
      CT(ct).events
      CT(ct).position
      keyboard
   end
end
```

```
function ASC_write(asc)

global ASC

switch char(asc.id)
    case 'sea'
        ASC.sea = asc;
    case 'land'
        ASC.land = asc;
end
```

```
function [asc] = ASC_write_cycle(ct,asc,t_gs,tasktype)

global BAYS BL CT EXEC TRF


%keyboard
origin = CT_act_pos(ct);
destiny = BL_act_pos(t_gs);
[asc.ciclo] = ASC_ETM(asc,ct,origin,destiny,0);

%asc.ciclo.origin = t_gs;
asc.ciclo.destiny = BL_act_pos(t_gs);
asc.ciclo.ascposition =  ASC_act_pos(asc);
asc.nextevent = sum(asc.ciclo.time);
asc.status = tasktype;

% The next line needs correction
CT(ct).stackestim = asc.ciclo.E;
%keyboard
if strcmp(tasktype,'stack') == 1
   BL.GS(t_gs).sreservations = BL.GS(t_gs).sreservations + 1;
   sr = BL.GS(t_gs).sreservations;
elseif strcmp(tasktype,'housekeeping') == 1
   %keyboard
   BL.GS(t_gs).HKreservations = BL.GS(t_gs).HKreservations + 1; % Stack reservation
   sr = BL.GS(t_gs).HKreservations;
   t_bay= BL.GS(t_gs).bay;
   if strcmp(CT(ct).id,'IMP') == 1
      if origin.bay >= t_bay
         disp('Error calculating jump for ct when housekeeping'); keyboard
      end
   elseif strcmp(CT(ct).id,'EXP') == 1
      if origin.bay <= t_bay
         disp('Error calculating jump for ct when housekeeping'); keyboard
      end
   end
end

Sdr = BL.GS(t_gs).Sdelreservations;
Ldr = BL.GS(t_gs).Ldelreservations;

% Reservation of the slot and bay id
t_bay = BL.GS(t_gs).bay;
for s = 1:BL.stacks
   b_gs = BAYS(t_bay).GS(s);
   BL.GS(b_gs).id = CT(ct).id;
end

if EXEC.plot ==1
   disp(['ASC(' asc.id ') CT(' num2str(ct) ') ' asc.status ' Calculation. Time2complete: ' num2str(asc.nextevent)])
   disp(['Origin: Bay/GS (' num2str(origin.bay) '/' num2str(origin.gs) ') Destiny: Bay/GS (' num2str(BL.GS(t_gs).bay) '/'
num2str(t_gs) ')'])
   disp(['Pile cts [' num2str(BL.GS(asc.ciclo.destiny.gs).cts) '] Stack/Delivery (Sea/Land)Reservations ' num2str(sr) ' / '
```

```
num2str(Sdr) ' / ' num2str(Ldr)])
end

if Sdr + Ldr > 1
    disp([TRF.PARAM.stackmode ' reservation error'])
    keyboard
end
```

```
function [errores] = AUX_check_bays(flow)

global BAYS BL

errores =0; e = 0;
for bay = 1:BL.bays
   bay_ocup = BAY_occupation(bay);
   if bay_ocup ==0
      for s=1:BL.stacks
        gs = BAYS(bay).GS(s);
        if strcmp(BL.GS(gs).id,flow) == 1
           e = e+1;
           errores(e) = gs;
        end
      end
   end
end

if e > 0
   keyboard
end
```

```
global COUNT CT
c = 0;
for i = 1:COUNT.CT.no
   if strcmp(CT(i).id,'EXP') == 1
      c=c+1;
      %impcts(c)= i;
      b(c) = CT(i).events.time(1);
   end
end

%impcts
b
figure; plot(b,'.')
mean(b)
```

```
for task =1:length(delt)
   dif = TIME.delt - t_acumulado(task);
   if dif >= 0
      f_task = task;
   end
   if and(task == f_task,dif ~= 0)
      keyboard
   end
end
```

```
function [Xf,Tf,a] = AUX_filter_repeated(X,T)

r = 0; p = 1;

Xf(1) = X(1); Tf(1) = T(1);
for i = 2:length(T)
    if T(i) == T(i-1)
        if X(i) == X(i-1)
            r = r+1;
            a(r) = i;
        end
    else
        p = p+1;
        Tf(p) = T(i);
        Xf(p) = X(i);
    end
end
```

```
function [TL,PL] = AUX_filtrar(uTL,uPL)

n = length(uPL);
j = 1;

PL(1) = uPL(1);
TL(1) = uTL(1);

for i = 2:n-1
    inc_ant = abs(uPL(i-1)-uPL(i));
    inc_pos = abs(uPL(i)-uPL(i+1));
    if inc_ant+inc_pos > 0
        j = j+1;
        PL(j)=uPL(i);
        TL(j)=uTL(i);
    end
end

PL(j+1) = uPL(n);
TL(j+1) = uTL(n);
```

```
function [target] = AUX_find_vector_pos(X,pos)
% Find the position of a point in a vector

target = 0;
n= length(X);

for t = 1:n
   if X(t) <= pos
      target = t;
   end
end
```

```
function [target] = AUX_find_vector_pos_e(X,pos)
% Find the position of a point in a vector

target = 0; found = 'N';
n= length(X);

for t = 1:n
    dif = abs(X(t) - pos);
    if and(strcmp(found,'N') ==1, dif == 0)
        found = 'Y';
        target = t;
    end
end
```

```
function [ant,pos] = AUX_pointinvector(X,P)
% This function returns the points of X around P (first solution)

s = 0;
%keyboard
for t = 1:length(X)-1
   delbayant = P - X(t);
   delbaypos = X(t+1) - P;
   if or(delbayant * delbaypos > 0, delbayant == 0)
      s = s+1;
      ant(s)=t; pos(s)=t+1;
   end
end

% if s > 1
%    % Case 4
%    disp('Multiple intersections')
% end
if s == 0
   ant = 0;
   pos = 0;
   %keyboard % case 8 9 9 4(time)
end
```

```
g=0;
for i =1:360
    if strcmp(BL.GS(i).id,'EXP')==1
        g(i)=BL.GS(i).group;
    else
        g(i)=0;
    end
end
figure
plot(g,'.-')
```

```
function [Xf,Tf] = AUX_time_filter(X,T,time)

p = 0; Xf = 0; Tf = 0;
for i = 1: length(X)
    if T(i)>=time
        p = p+1;
        Xf(p)=X(i); Tf(p) = T(i);
    end
end
```

```
function [R] = AUX_topct(ct)

global BL CT
keyboard
ctpos = CT_act_pos(ct);
R = 0;
if ct.tier < BL.GS(ctpos.gs).ocup
    R =1;
    disp(['CT(' num2str(ct) ') Not on top of the pile'])
    keybaord
end
```

```
function [f_task] = AUX_v_task(t_acumulado,t_act,i_task)

f_task = 0;
nt = length(t_acumulado);
for task =1:nt
    dif = t_act - t_acumulado(task);
    if dif >= 0
        f_task = task;
    else % The difference is very small so we include the task
        if abs(dif) < 1
            f_task = task;
        end
    end
end

if f_task < i_task
   f_task = 0;
end
```

```
function [acumulado] = AUX_vect_ac(vector)


nt = length(vector);
acumulado = zeros(1,nt);
acumulado(1) = vector(1);
for t_task =2:nt
    acumulado(t_task) = acumulado(t_task-1) + vector(t_task);
end
```

```
function [conflict] = BAY_backwards(t_bay,ct)

global CT

o_bay = CT(ct).position.bay(end);
conflict = 0;
if strcmp(CT(ct).id,'EXP') == 1
    if t_bay >= o_bay
        conflict = 1;%keyboard
    end
elseif strcmp(CT(ct).id,'IMP') == 1
    if t_bay <= o_bay
        conflict = 1; %keyboard
    end
end
```

```
function [BAY] = BAY_copy(n_bay)
% This function takes the bay number and returns the bay slots and cts

global BAYS BL

BAY.bay = n_bay;
BAY.cts = zeros(BL.tiers,BL.stacks);
BAY.slots = zeros(BL.tiers,BL.stacks);
for stack = 1:BL.stacks
   gs = BAYS(n_bay).GS(stack);
   if BL.GS(gs).ocup > BL.tiers
      disp(['GS(' num2str(gs) ') Overload'])
      keyboard
   end
   BAY.cts(:,stack) = BL.GS(gs).cts;
   for tier = 1:BL.tiers
      if BL.GS(gs).cts(tier) > 0
         BAY.slots(tier,stack) = 1;
      end
   end
end
```

```
function [fs,rs] = BAY_ES(bay,ct_stack)

global BAYS BL


% Calculate the number of free slots
fs = zeros(1,BL.stacks);
rs = zeros(1,BL.stacks);
for stack = 1: BL.stacks
   if stack ~= ct_stack
      gs = BAYS(bay).GS(stack);
      if BL.GS(gs).sreservations > 0
         rs(stack) = BL.tiers - BL.GS(gs).ocup; % BL.GS(gs).sreservations; %
         %keyboard
      else
         fs(stack) = BL.tiers-BL.GS(gs).ocup;
      end
   end
end
```

```
function [minbay,maxbay] = BAY_limit(flow,operation,demand)

global BF BL

% Make sure the SEA crane is not operating in that ground slot

seabay = BL.hklimit.sea; %10; % BL.baylimit.sea; % = 7;
landbay = BL.hklimit.land; %31; % BL.baylimit.land; % = 34;

if strcmp(demand,'peak') == 1
   minbay = seabay + 1;  caso = 0; % 10
   maxbay = landbay - 1; % 25
elseif strcmp(demand,'calm') == 1
   if strcmp(flow,'IMP') == 1
      if strcmp(operation,'stack') ==1
         caso = 1;
      elseif strcmp(operation,'delivery') == 1
         caso = 2; keyboard
      elseif strcmp(operation,'hkstack') == 1
         caso = 3;
      end
   elseif strcmp(flow,'EXP') == 1
      if strcmp(operation,'stack') ==1
         caso = 2;
      elseif strcmp(operation,'delivery') ==1
         caso = 1; keyboard
      elseif strcmp(operation,'hkstack') == 1
         caso = 4;
      end
   end
else
   disp('BAY LIMIT error')
   keyboard
end

if caso == 1    % Import stack and export delivery
   % HIGH OCCUPATION case
   if and(strcmp(operation,'stack') == 1, BF.sea.imp.ocup >= BF.sea.ocuplimit)
      %keyboard
      minbay = 1; % seabay +1
      maxbay = landbay-1; %

   % NORMAL OCCUPATION CASE
   else
      minbay= landbay; % 26
      maxbay = BL.bays;
%        if strcmp(ASC.land.status,'wait') == 1
%           maxbay = landbay-1;
%        else
%           maxbay = min(landbay-1,LIMITS.sea.bay-2);
%        end
   end
elseif caso == 2 % Export stack and import delivery
```

```
    % High occupation
    if and(strcmp(operation,'stack') == 1, BF.land.exp.ocup >= BF.land.ocuplimit)
        % This case is when considering peaks at sea
        % minbay = seabay+1;  %
        % maxbay = BL.bays; % landbay -1
        % This case is for normal situation
        minbay = 1;
        maxbay = landbay; %seabay; %10
    else
        %keyboard
        minbay = 1;
        maxbay = landbay; %seabay; %10
    end
elseif caso == 3 % Import housekeeping stack
    %keyboard
    minbay = BL.hklimit.land;
    maxbay = BL.bays;
elseif caso == 4 % Export a housekeeping stack
    %keyboard
    minbay = 1;
    maxbay = BL.hklimit.sea;
end
```

```
function [ncts,nrsv] = BAY_occupation(bay)

global BAYS BL

ncts = 0;nrsv = 0;
for s = 1:BL.stacks
    gs = BAYS(bay).GS(s);
    nrsv = nrsv + BL.GS(gs).sreservations;
    for t = 1:BL.tiers
        if BL.GS(gs).cts(t)>0
            ncts = ncts+1;
        end
    end
end
```

```
function [conflict] = BAY_prevent_conflicts(Sf_gs,SEARCH,OTHER)
% This function checks, when SEARCH is looking for a place to stack or HK,
% whether the OTHER crane is doing something on the same bay

global BAYS BL CT

Sf_bay = BL.GS(Sf_gs).bay;         % Search crane, final bay
if SEARCH.housekeeping == 1
   t_ct = SEARCH.hkct; %keyboar
   Si_bay = CT(t_ct).position.bay(end);
   Si_gs = CT(t_ct).position.gs(end);
else
   Si_bay = SEARCH.ciclo.origin.bay; % Search crane, initial bay
   Si_gs = SEARCH.ciclo.origin.gs;   % Search crane, initial GS
end
conflict = 0;
sreservations = 0; Ldelreservations = 0; Sdelreservations = 0; HKreservations = 0;

for s = 1:BL.stacks
   tgs = BAYS(Sf_bay).GS(s);
   sreservations = sreservations + BL.GS(tgs).sreservations;
   Ldelreservations = Ldelreservations  + BL.GS(tgs).Ldelreservations;
   Sdelreservations = Sdelreservations + BL.GS(tgs).Sdelreservations;
   HKreservations = HKreservations + BL.GS(tgs).HKreservations;
end

% Then check whether the original bay position is

ot_ct = OTHER.ciclo.originalct;

if ot_ct >0
   Oi_gs = OTHER.ciclo.origin.gs; %CT(ot_ct).position.gs(end);
   Oi_bay = OTHER.ciclo.origin.bay; %CT(ot_ct).position.bay(end);
   Of_gs = OTHER.ciclo.destiny.gs;
   Of_bay = OTHER.ciclo.destiny.bay;

   % Stack: compare GS since no reshuffles can occur
   % OTHER wants to stack @Sf_gs. NO need to compare o_gs
   if strcmp(OTHER.status,'stack') == 1

      % SEARCH wants to do hosuekeeping from Sf_bay
      if SEARCH.housekeeping == 1
         % Cicle is not calculated yet. To determine hypot reshuffles
         if CT(t_ct).position.tier(end) < BL.GS(Si_gs).ocup %there will be reshuffles
            %keyboard
            if Si_bay == Of_bay
               conflict = 1;
            end
         else
            %keyboard
            if Si_gs == Of_gs
               conflict = 1; %keyboard
            end
```

```
            end
            if Sf_gs == Of_gs
               conflict = 1;
            end
         end

         if Of_gs == Sf_gs
            conflict = 1;
         end

      elseif strcmp(OTHER.status,'housekeeping') == 1
         % SEARCH wants to stack @t_gs and the other hasn't picked yet. At
         % the original GS, reshuffles can occur
         %keyboard
         if Sf_bay == Oi_bay
            conflict=1;
         elseif Si_bay == Oi_bay
            conflict =1 ;
         elseif Sf_bay == Of_bay
            conflict=1;
         elseif Si_bay == Of_bay
            conflict =1 ;
         end
      elseif strcmp(OTHER.status,'delivery') == 1
%                if and(Ldelreservations > 0, strcmp(SEARCH.id,'sea') == 1)
%                   conflict = 1; %keyboard
%                elseif and(Sdelreservations > 0, strcmp(SEARCH.id,'land') == 1)
%                   conflict = 1; keyboard
%                end
         % Conflicts will arise if ot_ct has not been picked yet, because
         % there may be reshuffles
         if OTHER.ciclo.ctpicked == 0
            % SEARCH wants to put ct at the s_bay
            if SEARCH.housekeeping == 0 % That is, SEARCH asc is stacking
               %keyboard
               if OTHER.ciclo.nr == 0 % No reshuffles, compare GSs
                  if Oi_gs == Sf_gs
                     conflict = 1;
                  end
               else % There are reshuffles, compare bays
                  if Oi_bay == Sf_bay
                     conflict = 1;
                  end
               end
               % SEARCH is HK and wants to pick ct from the s_bay
            elseif SEARCH.housekeeping == 1
               %keyboard
               if OTHER.ciclo.nr == 0 % We can compare GSs
                  if Oi_gs == Sf_gs
                     conflict = 1;
                  end
               else % Compare bays
                  if Oi_bay == Sf_bay
                     conflict = 1;
```

```
                end
            end
        end
    end
  end
end

if conflict == 0
  if BL.GS(Sf_gs).Sdelreservations > 0
    if OTHER.ciclo.ctpicked == 1
      keyboard
    end
  elseif BL.GS(Sf_gs).Ldelreservations > 0
    %keyboard % Step (X  : ok). Step (12659: ok)
  elseif BL.GS(Sf_gs).sreservations > 0
    keyboard
  elseif BL.GS(Sf_gs).HKreservations > 0
    %keyboard
  end
end
```

```
function [conflict_gs] = BAY_prevent_conflicts2(Sf_gs,SEARCH,OTHER)
% This function checks, when SEARCH is looking for a place to stack or HK,
% whether the OTHER crane is doing something on the same bay

global BAYS BL CT
conflict_gs = [];

Sf_bay = BL.GS(Sf_gs).bay;        % Search crane, final bay
if SEARCH.housekeeping == 1
   t_ct = SEARCH.hkct; %keyboar
   Si_bay = CT(t_ct).position.bay(end);
   Si_gs = CT(t_ct).position.gs(end);
else
   Si_bay = SEARCH.ciclo.origin.bay; % Search crane, initial bay
   Si_gs = SEARCH.ciclo.origin.gs;   % Search crane, initial GS
end

sreservations = 0; Ldelreservations = 0; Sdelreservations = 0; HKreservations = 0;

for s = 1:BL.stacks
   tgs = BAYS(Sf_bay).GS(s);
   sreservations = sreservations + BL.GS(tgs).sreservations;
   Ldelreservations = Ldelreservations  + BL.GS(tgs).Ldelreservations;
   Sdelreservations = Sdelreservations + BL.GS(tgs).Sdelreservations;
   HKreservations = HKreservations + BL.GS(tgs).HKreservations;
end

% Then check whether the original bay position is

ot_ct = OTHER.ciclo.originalct;

if ot_ct >0
   Oi_gs = OTHER.ciclo.origin.gs; %CT(ot_ct).position.gs(end);
   Oi_bay = OTHER.ciclo.origin.bay; %CT(ot_ct).position.bay(end);
   Of_gs = OTHER.ciclo.destiny.gs;
   Of_bay = OTHER.ciclo.destiny.bay;

   % Stack: compare GS since no reshuffles can occur
   % OTHER wants to stack @Sf_gs. NO need to compare o_gs
   if strcmp(OTHER.status,'stack') == 1

      % SEARCH wants to do hosuekeeping from Sf_bay
      if SEARCH.housekeeping == 1
         % Cicle is not calculated yet. To determine hypot reshuffles
         if CT(t_ct).position.tier(end) < BL.GS(Si_gs).ocup %there will be reshuffles
            %keyboard
            if Si_bay == Of_bay
               conflict_gs = BAYS(Si_bay).GS;
            end
         else
            %keyboard
            if Si_gs == Of_gs
               conflict_gs = Si_gs; %keyboard
```

```
            end
         end
         if Sf_gs == Of_gs
            conflict_gs = Sf_gs;
         end
      end

      if Of_gs == Sf_gs
         conflict_gs = Sf_gs;
      end

   elseif strcmp(OTHER.status,'housekeeping') == 1
      % SEARCH wants to stack @t_gs and the other hasn't picked yet. At
      % the original GS, reshuffles can occur
      %keyboard
      if Sf_bay == Oi_bay
         conflict_gs = BAYS(Sf_bay).GS;
      elseif Si_bay == Oi_bay
         conflict_gs = BAYS(Si_bay).GS ;
      elseif Sf_bay == Of_bay
         conflict_gs = BAYS(Sf_bay).GS;
      elseif Si_bay == Of_bay
         conflict_gs = BAYS(Si_bay).GS;
      end
   elseif strcmp(OTHER.status,'delivery') == 1
%                 if and(Ldelreservations > 0, strcmp(SEARCH.id,'sea') == 1)
%                    conflict = 1; %keyboard
%                 elseif and(Sdelreservations > 0, strcmp(SEARCH.id,'land') == 1)
%                    conflict = 1; keyboard
%                 end
      % Conflicts will arise if ot_ct has not been picked yet, because
      % there may be reshuffles
      if OTHER.ciclo.ctpicked == 0
         % SEARCH wants to put ct at the s_bay
         if SEARCH.housekeeping == 0 % That is, SEARCH asc is stacking
            %keyboard
            if OTHER.ciclo.nr == 0 % No reshuffles, compare GSs
               if Oi_gs == Sf_gs
                  conflict_gs = Sf_gs;
               end
            else % There are reshuffles, compare bays
               if Oi_bay == Sf_bay
                  conflict_gs = BAYS(Sf_bay).GS;
               end
            end
            % SEARCH is HK and wants to pick ct from the s_bay
         elseif SEARCH.housekeeping == 1
            %keyboard
            if OTHER.ciclo.nr == 0 % We can compare GSs
               if Oi_gs == Sf_gs
                  conflict_gs = Sf_gs;
               end
            else % Compare bays
               if Oi_bay == Sf_bay
```

```
                conflict_gs = BAYS(Sf_bay).GS;
            end
        end
    end
  end
end

if isempty(conflict_gs) == 0
  if BL.GS(Sf_gs).Sdelreservations > 0
    if OTHER.ciclo.ctpicked == 1
      keyboard
    end
  elseif BL.GS(Sf_gs).Ldelreservations > 0
    %keyboard % Step (X  : ok). Step (12659: ok)
  elseif BL.GS(Sf_gs).sreservations > 0
    keyboard
  elseif BL.GS(Sf_gs).HKreservations > 0
    %keyboard
  end
end
```

```
function [conflict] = BAY_prevent_delivery_conflict(ct,asc)

global ASC CT

check = 0; conflict = 0;
Oi_bay = CT(ct).position.bay(end);

if strcmp(asc.id,'land') == 1
   if or(strcmp(ASC.sea.status,'delivery') == 1, strcmp(ASC.sea.status,'housekeeping') == 1)
      Sf_bay = ASC.sea.ciclo.destiny.bay;
      Si_bay = ASC.sea.ciclo.origin.bay;
      check = 1;
   end
elseif strcmp(asc.id,'sea') == 1
   if or(strcmp(ASC.land.status,'delivery') == 1, strcmp(ASC.land.status,'housekeeping') == 1)
      Sf_bay = ASC.land.ciclo.destiny.bay;
      Si_bay = ASC.land.ciclo.origin.bay;
      check = 1;
   end
end

if check == 1
   if or(Oi_bay == Sf_bay, Oi_bay == Si_bay)
   % if Oi_bay == Si_bay
      conflict = 1;
   end
end
```

```
function [R,BAY,solution] = BAY_reshuffles(asc,t_ct,BAY,W)
% Given a bay "BAY" and a ct "ct", this function calculates the reshuffles
% needed to move the containers on top of ct whithin the same bay

global CT TIME

old_bay = BAY;

R = ASC_cycle_ini();   solution = 1;
pos = CT_act_pos(t_ct);
no_reshuffles = sum(BAY.slots(:,pos.stack)) - pos.tier;

% Calculates the emtpy and stack reserved slots in the bay
h = sum(BAY.slots(:,pos.stack)); % no cts on the target stack

% Calculate the difference in time

min_time = TIME.simulation.T; %keyboard
for tier = pos.tier+1:h
   cct = BAY.cts(tier,pos.stack);
%     for i = 2: length(CT(cct).events.type)
%       if strcmp(CT(cct).events.type(i),'BL Drop')==1
%          ind = i;
%       end
%     end
   ind = 2;
   min_time = min(min_time,CT(cct).events.time(ind));
end

if W == 1
   t_dif = abs((min_time - CT(t_ct).events.time(2))/24/3600);
   CT(t_ct).timedif = t_dif;
   CT(t_ct).R.induced = CT(t_ct).R.induced + no_reshuffles;
end

% Calculate the reshuffles
for r = 1:no_reshuffles
   tier = h - r + 1;
   ctpos.tier = tier; ctpos.stack = pos.stack; ctpos.bay = pos.bay;
   ct = BAY.cts(tier,pos.stack); % This is the ct to be moved

   if W > 0
      CT(ct).R.n = CT(ct).R.n + 1;
   end

   [BAY,asc,ETM,solution] = CT_reshuffle(ctpos,BAY,asc,W);
   R = cycle_add(ETM,R,r);
end

a = R.time;
if sum(a) == 0
   keyboard
end
```

```
function [buffer] = BF_copy(side)
% This funciton makes a copy of the SEA or LAND buffer to work with

global BF

if strcmp(side,'sea') == 1
    buffer = BF.sea;
elseif strcmp(side,'land') == 1
    buffer = BF.land;
end
```

```
function BF_dropCT(side,ct)

global ASC BF COUNT CT TIME

% Make a copy of the Buffer pile

[pile] = PILE_copy(side,CT(ct).id);

if pile.ocup == 5
   disp(['BF drop warning: ' num2str(CT(ct).id) ' BF half occupated. Time ' num2str(TIME.t/24/3600)])
end
if pile.ocup == 9
   disp(['BF drop warning: ' num2str(CT(ct).id) ' BF very occupated. Time ' num2str(TIME.t/24/3600)])
   %keyboard
end
if pile.ocup == BF.size
   disp([' BF(' side '/' CT(ct).id ') drop Error: full BF ' num2str(CT(ct).id) ' at time ' num2str(TIME.t/24/3600)])
   pile.cts
   plot_BL
   keyboard
else
   % PUT CONTAINER IN THE BUFFER
   % ------------------------------------------------------------------------

   BF_put_ct(pile,ct);

   % UPDATE THE ASC WORKLOAD
   % ------------------------------------------------------------------------
   if strcmp(CT(ct).id,'IMP') == 1    % IMP CT ARRIVAL
   % ------------------------------------------------------------------------
      CT_add_event(ct,'SBF');
      % Place it in the ASC list
      [ASC.sea] = ASC_addtask(ASC.sea,ct,'stack',0);
   % ------------------------------------------------------------------------
   else    % EXP OR DUAL CT ARRIVAL
   % ------------------------------------------------------------------------
      CT_add_event(ct,'LBF');
      % Place it in the ASC list
      [ASC.land] = ASC_addtask(ASC.land,ct,'stack',0);
   end
end

%disp(['Drop operation: ' num2str(op_num) ' CT: ' num2str(ct) ' dropped in ' side ' buffer'  ])
```

```
function BF_erase_CT(pile,ct)
% This  function puts a ct on the imp or exp buffer pile
global BF CT

target_tier =0; target_stack  =0;
for stack = 1:BF.stacks
    for tier = 1:BF.tiers
        if pile.cts(tier,stack) == ct
            target_tier= tier; target_stack = stack;
        end
    end
end

if target_stack + target_stack == 0
    keyboard
end

pile.slots(target_tier,target_stack) = 0;
pile.cts(target_tier,target_stack) = 0;
pile.ocup = pile.ocup - 1;

CT_add_event(ct,'BF Leave');

BF_write(pile,CT(ct).id);
```

```
function BF_init()

global BF BL

BF.stacks = BL.stacks; BF.tiers = 20;
BF.size = BF.tiers * BF.stacks;

buffer.slots = zeros(BF.tiers,BF.stacks);
buffer.cts = zeros(BF.tiers,BF.stacks);
buffer.arrivals = 0;
buffer.ocup = 0;

% SEA
BF.sea.exp = buffer;
BF.sea.imp = buffer;
BF.sea.side = 'sea';

% LAND
BF.land.exp = buffer;
BF.land.imp = buffer;
BF.land.side = 'land';

BF.land.exp.bay = BL.bays + 2;
BF.land.imp.bay = BL.bays + 3;
BF.sea.exp.bay = - 2;
BF.sea.imp.bay = -1;

BF.sea.exp.side = 'sea';
BF.sea.imp.side = 'sea';

BF.land.exp.side = 'land';
BF.land.imp.side = 'land';

BF.sea.ocuplimit = 6;
BF.land.ocuplimit = 8;
```

```
function [pile,flow] = BF_pile(buffer,ct)

global CT

if ct == 0
   disp('IMP/EXP Pile error')
else
   if strcmp(CT(ct).id,'EXP')==1
      pile = buffer.exp.cts;
      flow = 'EXP';
   elseif strcmp(CT(ct).id,'IMP')==1
      pile = buffer.imp.cts;
      flow = 'IMP';
   end
end
```

```
function BF_put_ct(pile,ct)
% This  function puts a ct on the imp or exp buffer pile
global BF CT

[pos] = PILE_LOCATION(pile);

if pile.ocup == BF.size
   plot_BL
   plot_ASC_trajectories(50)
   keyboard
end

pile.slots(pos.tier,pos.stack) = 1;
pile.cts(pos.tier,pos.stack) = ct;
pile.ocup = pile.ocup + 1;

if strcmp(pile.side,'land') == 1
   ocuplimit = BF.land.ocuplimit;
elseif strcmp(pile.side,'sea') == 1
   ocuplimit = BF.sea.ocuplimit;
end

CT_add_pos(ct,pos);

BF_write(pile,CT(ct).id);
```

```
function BF_removeCT(ct,side)
% This function removes a ct from the buffer

global CT

pile = PILE_copy(side,CT(ct).id);
pos = CT_act_pos(ct);

if pile.cts(pos.tier,pos.stack) == 0
   CT(ct).events
   CT(ct).position
   pile.cts
   keyboard
else
   if pile.cts(pos.tier,pos.stack) ~= ct
      disp('Trying to retrieve the wrong CT');keyboard
   end
   pile.cts(pos.tier,pos.stack) = 0;
   pile.slots(pos.tier,pos.stack) = 0;
   pile.ocup = pile.ocup - 1;

   BF_write(pile,CT(ct).id);

   %disp(['CT(' num2str(ct) ') removed from BF ' num2str(side)])

   CT_add_event(ct,'BF Leave');

   BL_height();
end
```

```
function BF_write(pile,flow)
% This function overwrites the pile in the BF

global BF

switch char(pile.side)
    case 'land'
        switch char(flow)
            case 'IMP'
                BF.land.imp = pile;
            case 'EXP'
                BF.land.exp = pile;
        end
    case 'sea'
        switch char(flow)
            case 'IMP'
                BF.sea.imp = pile;
            case 'EXP'
                BF.sea.exp = pile;
        end
end
```

```
function [position] = BL_act_pos(gs)


global BL

position.bay = BL.GS(gs).bay;
position.stack= BL.GS(gs).stack;
position.tier= BL.GS(gs).ocup + 1;
position.gs= gs;
```

```
function [imp_bays,exp_bays] = BL_bay_types()

global BAYS BL

imp_bays = 0; exp_bays = 0;

for bay = 1:BL.bays
    gs = BAYS(bay).GS(1);
    if strcmp(BL.GS(gs).id,'IMP') == 1
        imp_bays = imp_bays +1;
    elseif strcmp(BL.GS(gs).id,'EXP') == 1
        exp_bays = exp_bays +1;
    end
end
```

```
function BL_dropCT(gs,ct,desreserve,asc)
% This function puts the ct on the stack

global ASC BAYS BL COUNT CT

%stopatct(ct,47)

% Check wether a CT is being placed on a wrong type of GS
if strcmp(BL.GS(gs).id,CT(ct).id) == 0
   if strcmp(BL.GS(gs).id,'NAS') == 1
      bay = BL.GS(gs).bay;
      for s = 1:BL.stacks
         b_gs = BAYS(bay).GS(s);
         BL.GS(b_gs).id = CT(ct).id;
      end
   else
      keyboard
   end
end

h = BL.GS(gs).ocup;

if h == BL.tiers
   disp(['CT(' num2str(ct) ') BL drop error: the pile @ BAY/GS(' num2str(BL.GS(gs).bay) '/' num2str(gs) ') is full'])
   plot_ASC_trajectories(50)
   BL.GS(gs).cts
   keyboard
end

% We need to remove the STACK reservations
if desreserve ==1
   if strcmp(asc.status,'housekeeping') == 1
      if BL.GS(gs).HKreservations == 0
         keyboard
      end
   elseif strcmp(asc.status,'stack') == 1
      if BL.GS(gs).sreservations == 0
         keyboard
      end
   end
   if or(BL.GS(gs).Sdelreservations > 0, BL.GS(gs).Ldelreservations > 0)
      disp([CT(ct).id ' CT(' num2str(ct) ') Drop warning. GS(' num2str(gs) ') has delivery reservations'])
      plot_ASC_trajectories(25)
      keyboard
   end

   if strcmp(asc.status,'housekeeping')==1
      %keyboard
      BL.GS(gs).HKreservations = BL.GS(gs).HKreservations - 1;
   elseif strcmp(asc.status,'stack')==1
      BL.GS(gs).sreservations = BL.GS(gs).sreservations - 1;
   else
      disp('Error removeing stack reservations')
```

```
    end
    sr = BL.GS(gs).sreservations; hkr = BL.GS(gs).HKreservations;
    disp(['CT(' num2str(ct) ') Stacked @GS(' num2str(gs) '). Stack/HK Reservations left (' num2str(sr) '/' num2str(hkr)
')'])
end

pos.tier = h+1;
BL.GS(gs).ocup = pos.tier;
BL.GS(gs).cts(pos.tier) = ct;
BL.GS(gs).group = CT(ct).group;

pos.bay = BL.GS(gs).bay;
pos.stack = BL.GS(gs).stack;
pos.gs = gs;

CT_add_pos(ct,pos);

CT_add_event(ct,'BL Drop');

if asc.housekeeping == 0
    if strcmp(CT(ct).id,'IMP') == 1
        [COUNT.INVENTORY.imp] = CT_inventory(COUNT.INVENTORY.imp,'add');
    elseif strcmp(CT(ct).id,'EXP') == 1
        [COUNT.INVENTORY.exp] = CT_inventory(COUNT.INVENTORY.exp,'add');
    elseif strcmp(CT(ct).id,'DUAL') == 1
        keyboard
    end
end
```

```
function BL_height()
% This function calculates the state of the Block

global  BL COUNT TIME

i = 0; e = 0; n = 0;
I = 0; E = 0; N = 0;

for gs = 1: BL.no_gs
   if strcmp(BL.GS(gs).id,'IMP')== 1
      i = i + 1; I(i) = BL.GS(gs).ocup;
   elseif strcmp(BL.GS(gs).id,'EXP')== 1
      e = e + 1; E(e) = BL.GS(gs).ocup;
   elseif strcmp(BL.GS(gs).id,'NAS')== 1
      n = n + 1; N(n) = BL.GS(gs).ocup;
   end
end
% keyboard
s = COUNT.S.no + 1;
COUNT.S.no = s;
COUNT.S.time(s) = TIME.t;
COUNT.S.I.no(s) = i; COUNT.S.I.mean(s) = mean(I); COUNT.S.I.std(s) = std(I);
COUNT.S.E.no(s) = e; COUNT.S.E.mean(s) = mean(E); COUNT.S.E.std(s) = std(E);
COUNT.S.N.no(s) = n; COUNT.S.N.mean(s) = mean(N); COUNT.S.N.std(s) = std(N);
```

```
function BL_inifill(filename)

global ASC BL COUNT CT TIME

nc = 0;
% Start a simulation without moving the cranes

%while TIME.t <= TIME.simulation.fill*3600*24
while nc <= TIME.simulation.fill
   % EVENT CREATOR
   TIME_next();

   switch TIME.e
      case 1 % Arrival of EXP ET
         [ct] = CT_generate1('EXP');
         gs = RANDOM_search_drop(ct,ASC.land,'calm',36);
         BL_dropCT(gs,ct,0);
         nc = nc+1;
      case 2 % Arrival of IMP ET
         COUNT.CT.land.imp = COUNT.CT.land.imp + 1;

      case 3 % Arrival of DUAL ET
         [ct] = CT_generate1('DUAL');
         gs = RANDOM_search_drop(ct,ASC.land,'calm',36);
         BL_dropCT(gs,ct,0);
         %COUNT.CT.land.imp = COUNT.CT.land.imp + 1;
      case 4 % Arrival of IMP CT
         %[ct] = CT_generate1('IMP');
         %gs = RANDOM_search_drop(ct,ASC.sea,'calm',36);
         %BL_dropCT(gs,ct,0);
         COUNT.CT.sea.imp = COUNT.CT.sea.imp + 1;

      case 5
         %keyboard
         COUNT.CT.sea.exp = COUNT.CT.sea.exp +1;
   end

end

% Make a treatment to the block
% ---------------------------------------------------------------------
% a) remove the reservations in bays
for i =1:BL.no_gs
   if BL.GS(i).sreservations > 0
      BL.GS(i).sreservations = 0;
   end
end

% b) Change the time parameters of the containers
for c = 1:ct
   % Change events
   for e = 1:CT(c).no_events
      CT(c).events.time(e) = CT(c).events.time(e) - TIME.simulation.fill*3600*24;
```

```
    end
    % Change position
    for p = 1: length(CT(c).position)
        CT(c).position.time(p) = CT(c).position.time(p) - TIME.simulation.fill*3600*24;
    end
    % Change dwell
    if strcmp(CT(c).id, 'IMP') == 1
        CT(c).dwell = CT(c).dwell - TIME.simulation.fill*3600*24;
    end
end


% c) Change the COUNT
COUNT.CT.land.exp = 0;
COUNT.CT.land.imp = 0;
COUNT.CT.land.dual = 0;
COUNT.CT.sea.imp = 0;
COUNT.CT.sea.exp = 0;

save(filename,'BL','COUNT','CT');
plot_BL
disp('**************************************************************************')
disp('                    END OF INITIAL FILL                          ')
disp('**************************************************************************')
```

```
function BL_init()

global BAYS BL CS S

% BLOCK
BL.stacks = 9;
BL.tiers = 5;
BL.bays = 40;
BL.no_gs = BL.bays * BL.stacks;
%BL.baylimit.land = BL.bays - 6; %34;
%BL.baylimit.sea = 7;
BL.hklimit.sea = fix(12/50*BL.bays);%16
BL.hklimit.land = BL.bays - fix(8/50*BL.bays);

s = 0;
for bay = 1:BL.bays
   for stack = 1:BL.stacks
      s = s+1;
%        CS(s).bay = bay;
%        CS(s).stack = stack;
      BAYS(bay).GS(stack)=s;
      %BL.gs.no(bay,stack) = s;
      BL.GS(s).id = 'NAS';
      BL.GS(s).group = 0;
      BL.GS(s).bay = bay;
      BL.GS(s).stack = stack;
      BL.GS(s).ocup = 0;
      BL.GS(s).cts(1:BL.tiers) = 0;
      BL.GS(s).sreservations = 0;
      BL.GS(s).Ldelreservations = 0;
      BL.GS(s).Sdelreservations = 0;
      BL.GS(s).HKreservations = 0;
      BL.GS(s).ctspointed = 0;
   end
end

S.w = 3.5; %17/6; % Width of individual CT slot
S.l = 50/40*6.058+0.40; %6.058; % 20" size
S.h = 2.59; % Standard 40"
S.baymargin = 3;
```

```
function BL_removeCT(slot,ct,desreserve,ascstatus)
% this function removes a CT from the stack in the BL

global ASC BAYS BL COUNT CT

ildelr = BL.GS(slot.gs).Ldelreservations;

h_stack = BL.GS(slot.gs).ocup;
ict = ct;
ctpos = CT_act_pos(ct); gs = ctpos.gs;

%keyboard
if gs ~= slot.gs
   bahia = BAY_copy(slot.bay)
   bahia.cts
   disp(['BL remove CT(' num2str(ct) ') Assignment error'])
   keyboard
else

   %disp(['CT(' num2str(ct) ') removed from GS/BAY(' num2str(gs) '/' num2str(ctpos.bay) ')'])

   if BL.GS(gs).ctspointed > BL.GS(gs).ocup
      keyboard
   end

   % We need to remove the DELIVERY reservations
   if desreserve == 0
      %keyboard
   elseif desreserve == 1
      if BL.GS(gs).sreservations > 0
         disp(['CT(' num2str(ct) ') Warning Pick'])
         %keyboard
      end

      if strcmp(ascstatus, 'housekeeping') == 1
         %BL.GS(gs).HKreservations = BL.GS(gs).HKreservations - 1;
      elseif strcmp(ascstatus, 'delivery') == 1

         if strcmp(CT(ct).id,'IMP') == 1
            if BL.GS(gs).Ldelreservations <= 0
               keyboard
            end
            if BL.GS(gs).ctspointed > BL.GS(gs).ocup
               keyboard
            end

            BL.GS(gs).Ldelreservations = BL.GS(gs).Ldelreservations - 1;
            dr = BL.GS(gs).Ldelreservations;
         elseif strcmp(CT(ct).id,'EXP') == 1
            if BL.GS(gs).Sdelreservations == 0
               keyboard
            end
            BL.GS(gs).Sdelreservations = BL.GS(gs).Sdelreservations - 1;
```

```
            dr = BL.GS(gs).Sdelreservations;
        end
        %disp(['CT(' num2str(ct) ') Delivery reservations left: ' num2str(dr)])
    else
        disp('error when removing reservations on delivery of ct')
    end
end

% 0. Before removing the CT, compute the reshuffles
[reshuffles,r_heights] = Reshuffles(slot.gs,slot.tier);
if reshuffles > 0
    disp(['CT(' num2str(ct) ') Pickup Error @GS(' num2str(gs) ') with cts [' num2str(BL.GS(gs).cts) ']'])
    disp(['CT is really at bay(' num2str(CT(ct).position.bay(end)) ') and gs(' num2str(CT(ct).position.gs(end)) ')'])
    plot_ASC_trajectories(50)
    keyboard
else
    % 1. Remove CT from stack
    BL.GS(gs).ocup = BL.GS(gs).ocup - 1;
    BL.GS(gs).cts(slot.tier) = 0;
    if CT(ct).pointed == 1
        BL.GS(gs).ctspointed = BL.GS(gs).ctspointed - 1;
    end
    % Add event
    CT_add_event(ct,'Pickbl');

    % 2. Check if the stack is empty
    if BL.GS(gs).ocup == 0
        BL.GS(gs).group = 0;
        if BL.GS(gs).Ldelreservations > 0
            keyboard
            BL.GS(gs).Ldelreservations = 0;
        end
    end

    % Check if the Bay id (IMP/EXP) has changed
    bayocup = BAY_occupation(slot.bay);
    if bayocup == 0
        sres = 0; hkres = 0; %keyboard
        for s = 1:BL.stacks
            t_gs = BAYS(slot.bay).GS(s);
            sres = sres + BL.GS(s).sreservations;
            hkres = hkres + BL.GS(s).HKreservations;
            BL.GS(t_gs).id = 'NAS';
        end
        disp(['BAY(' num2str(slot.bay) ') emptied'])
        if sres + hkres > 0
            disp('BL remove warning: bay emptied with HK reservations'); %keyboard
        end
    end

    % 2. Update the position of the CT that have been moved
    pos=slot;
    for ict = slot.tier:h_stack - 1
        ct = BL.GS(gs).cts(ict); keyboard
```

```
        pos.tier = ict;
        CT_add_pos(ct,pos);
        CT_add_event(ct,'Drop_4_Rsf');
      end

      if strcmp(ascstatus,'housekeeping') == 0
        if strcmp(CT(ct).id,'IMP') == 1
          [COUNT.INVENTORY.imp] = CT_inventory(COUNT.INVENTORY.imp,'remove');
        elseif strcmp(CT(ct).id,'EXP') == 1
          [COUNT.INVENTORY.exp] = CT_inventory(COUNT.INVENTORY.exp,'remove');
        end
      end
    end

    if BL.GS(slot.gs).Ldelreservations > BL.GS(gs).ocup
      keyboard
    end
  end
end
```

```
function [t_ct] = BL_search_pick1()
% This function makes a block search for CTs to be picked up

global ASC BL COUNT LIMITS CT TIME

% Initialize
t_ct = 0; gs = 0; nsr = 0; no = 0; ndrs = 0; ndrl = 0; ni = 0;

n = 0; dwell = 0;
while gs < BL.no_gs
   gs = gs+1;
   if strcmp(BL.GS(gs).id,'IMP') == 0
      ni = ni + 1; continue
   elseif BL.GS(gs).ocup == 0
      no = no + 1;  continue
%    elseif BL.GS(gs).sreservations > 0
%       nsr = nsr + 1;  continue
%    elseif BL.GS(gs).Sdelreservations > 0
%       ndrs = ndrs + 1;  continue
%    elseif BL.GS(gs).Ldelreservations > 0
%       ndrl = ndrl + 1;  continue
   else
      for c = 1:BL.GS(gs).ocup
         ct = BL.GS(gs).cts(c);
         if ct ==0
            keyboard
         end
         if CT(ct).pointed == 0
            %plot(CT(ct).dwell,'xb'); hold on
            n = n+1;

            if strcmp(CT(ct).id,'EXP') == 1
               disp(['Retrieving wrong type of CT('  num2str(ct) ')'])
               keyboard
            end
            dwell(n,1) = CT(ct).estim_leave;
            dwell(n,2) = ct;
            %dwell(n,3) = CT(ct).position.time(1);
         end
      end
   end
end

%keyboard
if n > 0
   %keyboard
   [mindwell,i] = min(dwell(:,1));
   t_ct = dwell(i,2);
   CT_add_event(t_ct,'Requested');
   if CT(t_ct).pointed == 1
      keyboard
   else
      CT(t_ct).pointed = 1;
```

```
    end

    pos = CT_act_pos(t_ct);    t_gs = pos.gs;

    [task_pos] = ASC_get_pointed(t_gs,ASC.land);

    % 3. Add the task to the YC
    [ASC.land] = ASC_addtask(ASC.land,t_ct,'delivery',task_pos);

    BL.GS(t_gs).ctspointed = BL.GS(t_gs).ctspointed + 1;

    er = COUNT.ER.no + 1; COUNT.ER.no = er;
    COUNT.ER.ocup(er) = BL.GS(t_gs).ocup;
    COUNT.ER.height(er) = pos.tier;
    COUNT.CT.land.imp = COUNT.CT.land.imp + 1;

    t_bay = CT(t_ct).position.bay(end);
    %t_gs = CT(t_ct).position.gs(end);
    LIMITS.sea.bay = t_bay;
    LIMITS.sea.ct = t_ct;

    % 4. Display
    disp(['CT (' num2str(t_ct) ') GS(' num2str(t_gs) ') Bay (' num2str(t_bay) ') Selected for delivery among ' num2str(n) '
CTs'])
    disp(['Ground slots [' num2str(BL.GS(t_gs).cts) '] with ' num2str(BL.GS(t_gs).Ldelreservations) ' reservations'])

    %disp(['ET arrival operation: ' num2str(COUNT.CT.land.imp) ' Pick IMP CT: ' num2str(target_ct)])
    disp('**********************************************************************')
else
    disp(['ET arrival operation: ' num2str(COUNT.CT.land.imp) ' Pick IMP CT: ' num2str(0)])
    disp('Candidate slots search Warning: no valid candidate GSs to pick CT')
    plot_inventory
    keyboard
end
```

```
function [target_ct]= cdf_stay(ct_list)
% This function calculates the cumulative density function of the CT stay
% time

target_ct = 0;

[no_ct,b] =  size(ct_list);
% 1.3 Sort the list according to the time (coordinate 3
ct_list = sortrows(ct_list,3);

% 1.4 Make a vector of accumulated times:
ac_time(1) = ct_list(1,3);
ct_list(:,3) = ct_list(:,3) - ac_time(1);
ac_time(1) = ct_list(1,3);

for i=2:no_ct
   ac_time(i) = ac_time(i-1) + ct_list(i,3);
end

sum_times = sum(ct_list(:,3));

% 1.5 Generate a random number and find the position of the target CT
target_time = 1 + fix(random('unif',0,1)*sum_times);
ct_no = 1;
while target_time > ac_time(ct_no)
   if ct_no> no_ct
      disp('error')
      keyboard
   end
   ct_no = ct_no + 1;
end

target_ct = ct_list(ct_no,4);
```

```
function [xf,top_xf,cdf]=cdf_x2(no_wc,wcl,min_weight,max_weight)

no_c=length(wcl);
swcl=sort(wcl); % sorted weight container levels

% Calculate the new cdf
ndelx=(max_weight-min_weight)/no_wc; % Intervals at x axis (weight)
top_xf=(ndelx:ndelx:ndelx*no_wc)+min_weight;
xf=top_xf-ndelx/2;

cont=1;
i=1; % index for the whole lenght of weight list
while i<=no_c && cont<no_wc
   if swcl(i)<=top_xf(cont)
      i=i+1;
   else
      cdf(cont)=i-1;
      cont=cont+1;
      i=i+1;
   end
end
cdf(no_wc)=no_c;
cdf=cdf/no_c;
```

```
function check_reshuffles(asc,ct)

global CT

if nct > 0
   %CT(nct).nr = asc.ciclo.nr;
   if asc.ciclo.nr> 0
      %           if strcmp(asc.tasks.action(task_no),'delivery') == 1
      %              keyboard
      %           end
      if strcmp(CT(nct).id,'EXP') == 1
         keyboard
      end
   end
end
```

```
function [conflict] = checkbayreservations(gs,ascid)
% This function checks whether the other crane is doing something on the
% same bay as ascid

global ASC BAYS BL CT

bay = BL.GS(gs).bay;
conflict = 0;
sreservations = 0; Ldelreservations = 0; Sdelreservations = 0; HKreservations = 0;

for s = 1:BL.stacks
   tgs = BAYS(bay).GS(s);
   sreservations = sreservations + BL.GS(tgs).sreservations;
   Ldelreservations = Ldelreservations  + BL.GS(tgs).Ldelreservations;
   Sdelreservations = Sdelreservations + BL.GS(tgs).Sdelreservations;
   HKreservations = HKreservations + BL.GS(tgs).HKreservations;
end

if and(Ldelreservations > 0, strcmp(ascid,'sea') == 1)
   conflict = 1;
elseif and(Sdelreservations > 0, strcmp(ascid,'land') == 1)
   conflict = 1;
elseif sreservations == 1
   conflict = 1;
end

% Then check whether the original bay position is
if strcmp(ascid,'sea') == 1
   t_ct = ASC.land.ciclo.originalct;
   if t_ct >0
      if strcmp(ASC.land.status,'housekeeping') == 1
         pbay = CT(t_ct).position.bay(end);
         if bay == pbay
            if ASC.land.ciclo.ctpicked == 0
               conflict = 1; %keyboard
            elseif HKreservations >0
               conflict = 1;
            end
         end
      end
   end
elseif strcmp(ascid,'land') == 1
   t_ct = ASC.sea.ciclo.originalct;
   if t_ct >0
      if strcmp(ASC.sea.status,'housekeeping')==1
         pbay = CT(t_ct).position.bay(end);
         if bay == pbay
            if ASC.sea.ciclo.ctpicked == 0
               conflict = 1; %keyboard
            elseif HKreservations >0
               conflict = 1; keyboard
            end
         end
```

```
        end
    end
end
```

```
function COST_init()


global COST

COST.reshuffle.EXP = 0;
COST.r_heights.EXP = 0;

COST.reshuffle.IMP = 0;
COST.r_heights.IMP = 0;
```

```
function COUNT_init()

global COUNT

COUNT.CT.no = 0;
COUNT.CT.land.imp = 0;
COUNT.CT.land.exp = 0;
COUNT.CT.land.dual = 0;
COUNT.CT.sea.imp = 0;
COUNT.CT.sea.exp = 0;
COUNT.VS = 0;
COUNT.ASC.sea.tasks = 0;
COUNT.ASC.land.tasks = 0;
COUNT.ASC.sea.hktasks = 0;
COUNT.ASC.land.hktasks = 0;

COUNT.INVENTORY.imp.no_cts = 0;
COUNT.INVENTORY.imp.time = 0;
COUNT.INVENTORY.exp.no_cts = 0;
COUNT.INVENTORY.exp.time = 0;
% For calculating the reshuffles

R.events = 0;
R.nr(1) = 0;
R.revent(1) = 0;
R.sheight(1) = 0;

COUNT.RE = R;
COUNT.RhkE = R;
COUNT.RI = R;
COUNT.RhkI = R;

COUNT.ER.no = 0;
COUNT.ER.ocup = 0;
COUNT.ER.height = 0;
% For calculating the Block state (heights)
COUNT.S.no = 0;
COUNT.S.time = 0;
COUNT.S.I.no = 0; COUNT.S.I.mean = 0; COUNT.S.I.std = 0;
COUNT.S.E.no = 0; COUNT.S.E.mean = 0; COUNT.S.E.std = 0;
COUNT.S.N.no = 0; COUNT.S.N.mean = 0; COUNT.S.N.std = 0;
```

```
function [t_gs,asc,found] = CRANDOM_search_drop(ct,asc)
% This function makes a block search for candidate slots to drop a CT
global ASC BAYS BF BL CT S TRF

found = true; t_gs = 0;
acs = 0; AGS = 0;  bcs = 0; BGS = 0;
[origin] = CT_act_pos(ct);

%[minbay,maxbay] = BAY_limit(CT(ct).id,'stack',demand);

% 1 Determine the limiting bay for a crane
[ini_gs, end_gs, del_gs] = ASC_bay_direction(asc);
[imp_bays,exp_bays] = BL_bay_types();

% Satisfy the criteria to be a candidate slot
for gs = ini_gs:del_gs:end_gs
   bay = BL.GS(gs).bay;

   % a) GS and Bay occupation
   [bocup,bres] = BAY_occupation(bay);
   if bocup+bres > BL.tiers*BL.stacks - BL.tiers;
     continue
   end

   if BL.GS(gs).ocup == BL.tiers
      continue
   end

   % b) This is to prevent a worse position during housekeeping
   if asc.housekeeping == 1
      [conflict] = BAY_backwards(bay,ct);
      if conflict == 1
         continue
      end
   end

   % c) Special criteria: if the other asc is delivering, don't put
   if strcmp(asc.id,'land') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.sea);
   elseif strcmp(asc.id,'sea') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.land);
   end
   if conflict == 1 % and(, bay < BL.bays)
      continue
   end

   % d) Traffic flow.
   bayflow = BL.GS(gs).id;

   if strcmp(CT(ct).id,'IMP') == 1   % Import flow
      % a) Disregard exp bays
      if strcmp(bayflow,'EXP') == 1
         continue
```

```
      elseif bocup > TRF.PARAM.baymaxocup
         continue
      end
      if imp_bays > 22
         if bocup > 0
            bcs = bcs + 1; BGS(bcs) = gs; %ccs = ccs + 1; CGS(ccs) = gs;
         end
      else
         acs=acs+1;  AGS(acs) = gs;
      end
%        if and(BL.GS(gs).ocup > 0, BL.GS(gs).ocup < 3)
%           acs=acs+1;  AGS(acs) = gs;
%        else
%           bcs = bcs+1; BGS(bcs) = gs;
%        end
   elseif strcmp(CT(ct).id,'EXP') == 1 % Export flow. Piles can be EXP
      % a) Disregard imp bays
      if strcmp(bayflow,'IMP') == 1
         continue
      end
      if exp_bays > 22
         if bocup > 0
            bcs = bcs + 1; BGS(bcs) = gs; %ccs = ccs + 1; CGS(ccs) = gs;
         end
      else
         acs = acs+1;  AGS(acs) = gs;
      end
      % b) Group: disregard other groups
%        if BL.GS(gs).group > 0 % For bays with a group assigned
% %           if BL.GS(gs).group == CT(ct).group
%              acs = acs+1;  AGS(acs) = gs;
% %           else
% %              ccs = ccs+1; CGS(ccs) = gs;
% %           end
%        else
%           bcs = bcs+1; BGS(bcs) = gs;
%        end
      % c) Pile occupation
   end
end

if acs>0
   GS = AGS;     CT(ct).strategy = 'A';
elseif bcs>0
   GS = BGS;     CT(ct).strategy = 'B';
% elseif ccs >0
%    GS =CGS;     CT(ct).strategy = 'C';
else
   found = 0;
end

% Determine the coeficients of the candidate slots
if found
   %keyboard
```

```
    n = length(GS);    CT(ct).strategypossibilities = n;
    % Select ground slot randomly
    t_gs = GS(random('unid',n));
end
```

```
function [position] = CT_act_pos(ct)
% This function gets the position of the CT on the BL

global CT

pos = length(CT(ct).position.bay);
position.bay = CT(ct).position.bay(pos);
position.stack = CT(ct).position.stack(pos);
position.tier = CT(ct).position.tier(pos);
position.gs = CT(ct).position.gs(pos);
```

```
function CT_add_event(ct,event)

global CT TIME

e = CT(ct).no_events;
CT(ct).no_events = e + 1;

if e == 0
   i = 1;
else
   i = e+1;
end
CT(ct).events.time(i) = TIME.t;
CT(ct).events.type{i} = event;
```

```
function CT_add_pos(ct,pos)
% This function places a CT on the given position

global CT TIME

if CT(ct).position.gs + CT(ct).position.time == 0
    event = 0;
else
    event = length(CT(ct).position.bay);
end

CT(ct).position.stack(event+1) = pos.stack;
CT(ct).position.tier(event+1) = pos.tier;
CT(ct).position.bay(event+1) = pos.bay;
CT(ct).position.gs(event+1) = pos.gs;
CT(ct).position.time(event+1) = TIME.t;
```

```
function CT_add_time(ct,time,move)

global CT

if sum(CT(ct).timepermove) == 0
    CT(ct).timepermove = time;
    CT(ct).movings{1} = move;
else
    CT(ct).timepermove = [CT(ct).timepermove, time];
    CT(ct).movings = [CT(ct).movings, move];
end
```

```
function CT_check_UnMoves(ct,moves)

global CT

error = 0;
for j = 1: length(moves)
    a = char(moves(j));
    if strcmp(a(1),'u') == 1
        if CT(ct).nr == 0
            error = error +1;
        end
    end
end

if error > 0
    disp(['Error in CT(' num2str(ct) ')']);
    keyboard
end
```

```
function CT_destiny(ct,side)

global CT

% Search for stacks of the same group
[CS,E,found] = BL_search(ct,side);

if found
   disp('CTs found')
else
   % Search for empty stacks
   [CS,E,found] = BL_search(ct,side);
end

% Analyze the candidate stacks
```

```
function CT_dwell()

global BAYS BL COUNT CT TIME TRF
idwell = zeros(1,2); edwell = zeros(1,2); no_ict = 0; no_ect = 0;

for bay = 1: BL.bays
   for s = 1:BL.stacks
      t_gs = BAYS(bay).GS(s);
      if strcmp(BL.GS(t_gs).id,'IMP') == 1
         for t = 1:BL.GS(t_gs).ocup
            no_ict = no_ict +1;
            ct = BL.GS(t_gs).cts(t);
            idwell(no_ict,1) = CT(ct).estim_leave; % TIME.t - CT(ct).events.time(2);
            idwell(no_ict,2) = ct;
         end
      elseif strcmp(BL.GS(t_gs).id,'EXP') == 1
         for t = 1:BL.GS(t_gs).ocup
            no_ect = no_ect +1;
            ct = BL.GS(t_gs).cts(t);
            est_line = TRF.PARAM.CT.groups(CT(ct).group,3);
            edwell(no_ect,1) = 1;
%              if TRF.VS(COUNT.VS+1).line == est_line
%                  edwell(no_ect,1) = 1; %TRF.VS(vs).arrival-TIME.t;
%              else
%                  edwell(no_ect,1) = 0;
%              end
            edwell(no_ect,2) = ct;
         end
      end
   end
end

sum_times = sum(idwell(:,1));
for c = 1:no_ict;
   ct = idwell(c,2);
   CT(ct).pickp = idwell(c,1)/sum_times;
end
%keyboard
sum_times = sum(edwell(:,1));
for c = 1:no_ect;
   ct = edwell(c,2);
   CT(ct).pickp = edwell(c,1)/sum_times;
end
```

```
function CT_energy_sum(asc,ct)

global CT

%keyboard
le = length(asc.ciclo.E);
% initial check
orig_e = CT(ct).energy; orig_E = CT(ct).cicloe;
if abs(orig_e - sum(orig_E))> 0.1
   keyboard
end

c = 0;
%keyboard
for t = 1:le
   %if strcmp(asc.ciclo.moves(t),'trans') == 0 % If not trans
      c = c + 1;
      cicloe(c) = asc.ciclo.E(t);
      ciclomoves(c) = asc.ciclo.moves(t);
      %CT_add_time(c,asc.ciclo.originaltime(t));
   %end
end

CT_energy_add(ct,cicloe,ciclomoves);
```

```
function [ct] = CT_generate1(type)

global BF COUNT CT TIME
% keyboard

% Update the CTs count
COUNT.CT.no = COUNT.CT.no + 1;
ct = COUNT.CT.no;

CT(ct).id = type;
if strcmp(CT(ct).id,'IMP') == 1    % IMP CT ARRIVAL
   COUNT.CT.sea.imp = COUNT.CT.sea.imp + 1;
   CT(ct).dwell = BF.sea.imp.dwell(COUNT.CT.sea.imp);
   CT(ct).estim_leave = TIME.t + TIME.delt + CT(ct).dwell;
   if CT(ct).dwell == 0
      disp('Error generating ct. No dwell time'); keyboard
   end
   CT(ct).weight = BF.sea.imp.weight(COUNT.CT.sea.imp);
   CT(ct).w_class = BF.sea.imp.w_class(COUNT.CT.sea.imp);
   CT(ct).group = BF.sea.imp.group(COUNT.CT.sea.imp);
   location.bay = BF.sea.imp.bay;
else    % EXP OR DUAL CT ARRIVAL
   location.bay = BF.land.imp.bay;
   switch char(CT(ct).id)
      case 'EXP'
         COUNT.CT.land.exp = COUNT.CT.land.exp + 1;
         CT(ct).weight = BF.land.exp.weight(COUNT.CT.land.exp);
         CT(ct).w_class = BF.land.exp.w_class(COUNT.CT.land.exp);
         CT(ct).group = BF.land.exp.group(COUNT.CT.land.exp);
      case 'DUAL'
         COUNT.CT.land.dual = COUNT.CT.land.dual + 1; %keyboard
         CT(ct).id = 'EXP';
         CT(ct).weight = BF.land.dual.weight(COUNT.CT.land.dual);
         CT(ct).w_class = BF.land.dual.w_class(COUNT.CT.land.dual);
         CT(ct).group = BF.land.dual.group(COUNT.CT.land.dual);
   end
end

disp([num2str(type) ' CT(' num2str(ct) ') Arrival to terminal' ])% List of events
%disp('-----------------------------------------------------------------')
CT(COUNT.CT.no).no = ct;
CT(ct).no_events = 1;
CT(ct).events.time(1) = TIME.t+TIME.delt;
CT(ct).events.type{1} = 'Generate';
CT(ct).pointed = 0;
CT(ct).vs = 0;

% reshuffles
CT(ct).R.n = 0;
CT(ct).R.induced = 0;

location.stack = 0;
location.tier = 0;
```

```
location.gs = 0;
location.time = 0;
location.vs = 0;
CT(ct).position = location;
CT(ct).target = location;
CT(ct).energy = 0;
CT(ct).cicloe = 0;
CT(ct).priority = 0;
CT(ct).timedif = 0;
CT(ct).strategy = {};
CT(ct).strategypossibilities = [];


% weight
%[CT(ct).weight,CT(ct).w_class] = CT_weight(random('unif',TRF.CT.cdf(1),1));
% POD
%CT(ct).pod = fix(random('unif',1,TRF.PARAM.no.pod));
% Line
%CT(ct).line = fix(random('unif',1,TRF.PARAM.no.lines));
% Group
%CT(ct).group = CT_group_search(CT(ct).pod, CT(ct).line, CT(ct).w_class);
```

```
function [target_g] = CT_group_search(pod, line, w_class)

global TRF

for g = 1: TRF.PARAM.CT.no_groups
   if TRF.PARAM.CT.groups(g,2) == pod
      if TRF.PARAM.CT.groups(g,3) == line
         if TRF.PARAM.CT.groups(g,4) == w_class
            target_g = g;
         end
      end
   end
end
```

```
function CT_init()

global CT

CT.no = 0;
CT.no_events = 0;
CT.events.time = 0;
CT.events.type = '';
CT.id = '';
CT.weight = 0;
%CT.pod = 0;
%CT.line = 0;
CT.nr = 0;

%CT.buffer = '';
```

```
function [INVENTORY] = CT_inventory(INVENTORY,operation)

global TIME

if strcmp(operation,'add') == 1
   if INVENTORY.no_cts == 0
      INVENTORY.no_cts(1) = 1;event = 0;
   else
      event = length(INVENTORY.no_cts);
      INVENTORY.no_cts(event+1) = INVENTORY.no_cts(event) + 1;
   end
elseif strcmp(operation,'remove') == 1
   if INVENTORY.no_cts == 0
      disp('Error'); keyboard
   else
      event = length(INVENTORY.no_cts);
      INVENTORY.no_cts(event+1) = INVENTORY.no_cts(event) - 1;
   end
else
   disp('Error in the Inventory Operation')
end

INVENTORY.time(event+1) =  TIME.t;
```

```
function [leaps] = CT_leaps(ct)

global BL CT

leaps = zeros(1,BL.bays);
if strcmp(CT(ct).id,'EXP') == 1
   positions = wrev(unique(CT(ct).position.bay)); % "flip" in matlab 2014 % "wrev" in 2011
elseif  strcmp(CT(ct).id,'IMP') == 1
   positions = unique(CT(ct).position.bay);
end

for i = 2:length(positions)-2
   leaps(positions(i)) = abs(positions(i+1)-positions(i));
%     if leaps(positions(i)) >30
%         positions
%         keyboard
%     end
end

%figure; plot(leaps,'.')
```

```
function [BAY,asc,R,sol] = CT_reshuffle(P2,BAY,asc,W)
% This function takes a CT in one position and moves it to another position
% within the bay
% The ASC is already located at the bay but it has not made the trolley
% movement yet

global BAYS BL TRF

% Positions: P1 is the crane postion, P2 is the CT position, P3 is the
% target position hat will be evaluated

%keyboard
sol  = 0;
h = sum(BAY.slots);
if W == 1
   estim_or_calc = 0;
else
   estim_or_calc = 1;
end
ct = BAY.cts(P2.tier,P2.stack);
[ETM,R] = ASC_cycle_ini();
T.tier = 0; T.stack = 0; T.bay = 0; T.time =0;
DELT = ones(1,BL.stacks)*1000;
E = ones(1,BL.stacks)*100000000;

for stack = 1:BL.stacks

   % The original positions shall be refreshed in each loop
   P1 = ASC_act_pos(asc); P1.tier = BL.tiers+1; P1.bay = P2.bay;

   % Check wether the stack can have the ct
   gs = BAYS(BAY.bay).GS(stack);
   if stack == P2.stack
      continue
   elseif h(stack) == BL.tiers
      continue
   % IMPORTANT For the next line:
   % W = 0 when called by ASC_ETM to calculate potential reshuffles
   % W = 1 when called by ASC_calc_delivery to calculate delivery cycle
   elseif BL.GS(gs).sreservations
      if strcmp(asc.status, 'delivery') == 1
         keyboard
         continue
      end
   end

   m=0; sol = 1;

   r(stack).bay = P1.bay; % initially the crane is at the bay
   r(stack).stack = P1.stack;
   r(stack).tier = P1.tier;
   %r(stack).time = P1.time;
   r(stack).time = 0;
```

```
    r(stack).e = 0;
    r(stack).moves = '';
    r(stack).ct = 0;
    r(stack).ctmove = 0;

    % Determine the upper position
    tier = h(stack)+1;
    % Determine the energy and time
    m = m+1;  r(stack).bay(m) = P1.bay; r(stack).stack(m)= P2.stack; r(stack).tier = BL.tiers + 1; % O.tier;
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] =
ASC_energy(P1.stack,P2.stack,ct,'utrolley','empty',estim_or_calc);
    P1.stack = P2.stack;   [asc] = ASC_move(asc,P1,0,W); r(stack).ct(m) = 0; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack) = addposition(r(stack),'uhoist',P2.tier);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] =
ASC_energy(BL.tiers+1,P2.tier,ct,'ulower','empty',estim_or_calc);
    P1.tier = P2.tier;    [asc] = ASC_move(asc,P1,0,W); r(stack).ct(m) = 0; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack) = addposition(r(stack),'uhoist',BL.tiers+1);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] =
ASC_energy(P2.tier,BL.tiers+1,ct,'upickbl','full',estim_or_calc);
    P1.tier = BL.tiers+1; [asc] = ASC_move(asc,P1,ct,W); r(stack).ct(m) = ct; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack) = addposition(r(stack),'utrolley',stack);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = ASC_energy(P2.stack,stack,ct,'utrolley','full',estim_or_calc);
    P1.stack = stack;     [asc] = ASC_move(asc,P1,ct,W); r(stack).ct(m) = ct; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack)=addposition(r(stack),'uhoist',tier);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = ASC_energy(BL.tiers+1,tier,ct,'udropbl','full',estim_or_calc);
    P1.tier = tier;        [asc] = ASC_move(asc,P1,ct,W); r(stack).ct(m) = ct; r(stack).ctmove(m) = ct;

    m = m+1;  r(stack)=addposition(r(stack),'uhoist',BL.tiers+1);
    [r(stack).time(m),r(stack).e(m),r(stack).moves{m}] = ASC_energy(tier,BL.tiers+1,ct,'uraise','empty',estim_or_calc);
    P1.tier = BL.tiers+1;  [asc] = ASC_move(asc,P1,0,W); r(stack).ct(m) = 0; r(stack).ctmove(m) = ct;

    DELT(stack) = sum(r(stack).time); E(stack) = sum(r(stack).e);
end

if sol == 1
    coefs = DELT/max(DELT)*TRF.PARAM.weight.t + E/max(E)*TRF.PARAM.weight.E;

    [c_min,T.stack] = min(coefs);
    T.tier = h(T.stack) +1;

    time = DELT(T.stack);

    BAY.cts(T.tier,T.stack) = ct;
    BAY.slots(T.tier,T.stack) = 1;
    BAY.cts(P2.tier,P2.stack) = 0;
    BAY.slots(P2.tier,P2.stack) = 0;
    BAY.time = BAY.time + time;
    T.bay = BAY.bay;
    T.time = BAY.time;
```

```
    % Write the results
    R.time = r(T.stack).time;
    R.E = r(T.stack).e;
    R.moves = r(T.stack).moves;
    R.bay = r(T.stack).bay;
    R.stack = r(T.stack).stack;
    R.tier = r(T.stack).tier;
    R.ct = r(T.stack).ct;
    R.ctmove = r(T.stack).ctmove;
else
    disp('Error: No stack available for CT'); keyboard
end
```

```
function [weight,wc] = CT_weight(seed)

global TRF

distrib = 'random';

if strcmp(distrib,'random') == 1
   if seed == 1
      weight = 32500;
   elseif seed == 0
      weight = 2200;
   else
      weight = random('unif',2200,32500);
   end

else

   if seed == 1
      weight = 32500;
   else
      cdf  = TRF.CT.cdf;
      psup = 1;
      while seed >  TRF.CT.cdf(psup)
         psup = psup+1;
      end
      pinf = psup - 1;
      % linear interpolation between values
      delp = TRF.CT.cdf(psup)-TRF.CT.cdf(pinf);
      delw = TRF.CT.top_xf(psup)-TRF.CT.top_xf(pinf);

      incp =  seed - TRF.CT.cdf(pinf);
      incw = incp*delw/delp;

      weight = TRF.CT.top_xf(pinf) + incw;
   end

end

% now the container weigth class
delc = (32500-2200)/TRF.PARAM.no.ct_weights;
up_w = (1:TRF.PARAM.no.ct_weights)*delc+2200;
wc = 1;

while weight>up_w(wc)
   wc = wc+1;
end
```

```
function CT_write_cycle(ct,energia,move,time)

global CT

%keyboard
%disp(['Writing CT(' num2str(ct) ') cycle'])
% First calculate if there is an empty cycle
empt = 0;
if length(CT(ct).cicloe) == 1
   if CT(ct).energy == 0
      empt = 1;
   end
end

if empt == 1
   CT(ct).energy = sum(energia);
   CT(ct).cicloe = energia;
   CT(ct).ciclomoves{1} = move;
   CT(ct).ciclotimes = time;
else
   %keyboard
   CT(ct).energy = CT(ct).energy + sum(energia);
   CT(ct).cicloe = [CT(ct).cicloe,energia];
   CT(ct).ciclomoves = [CT(ct).ciclomoves,move];
   CT(ct).ciclotimes = [CT(ct).ciclotimes,time];
end

% if strcmp(move, 'upickbl') == 1
%     CT(ct).R.n = CT(ct).R.n + 1;
% end

if abs(CT(ct).energy - sum(CT(ct).cicloe)) > 0.1
   disp('energy error')
   keyboard
end
```

```
function [R] = cycle_add(ETM,R,r)

if r ==1
    R = ETM;
else
    R.time = [R.time,ETM.time];
    R.E = [R.E,ETM.E];
    R.moves = [R.moves,ETM.moves];
    R.bay = [R.bay,ETM.bay];
    R.stack = [R.stack,ETM.stack];
    R.tier = [R.tier,ETM.tier];
    R.ct = [R.ct,ETM.ct];
    R.ctmove= [R.ctmove,ETM.ctmove];
end
```

```
function [O] = decompose(R,n)

O.mE = mean(R.E);
disp(['# cts: ' num2str(n) ' mean energy (kWh): ' num2str(O.mE) ]); xlabel('CT class'); ylabel('Mean Energy (kWh)');
% S = Stack, D = Delivery
e_scale = 1.000000;
% PRODUCTIVE
% Gantry number
P.gl = R.S.P.G.eloa.no + R.D.P.G.eloa.no; P.gs = R.S.P.G.eloa.no + R.S.P.G.eunl.no;
P.gu = R.S.P.G.eunl.no + R.D.P.G.eunl.no; P.gd = R.D.P.G.eloa.no + R.D.P.G.eunl.no;
P.g = P.gl + P.gu;
% Gantry energy
P.GeL = sum(R.S.P.G.eloa.e) + sum(R.D.P.G.eloa.e); P.GeS = sum(R.S.P.G.eloa.e) + sum(R.S.P.G.eunl.e);
P.GeU = sum(R.S.P.G.eunl.e) + sum(R.D.P.G.eunl.e); P.GeD = sum(R.D.P.G.eloa.e) + sum(R.D.P.G.eunl.e);
P.G = P.GeL+P.GeU;
% Gantry time
P.GtS = sum(R.S.P.G.eloa.t) + sum(R.S.P.G.eunl.t);
P.GtD = sum(R.D.P.G.eloa.t) + sum(R.D.P.G.eunl.t);
P.Gt = P.GtS + P.GtD;
% trolley number
P.ts = R.S.P.T.no; P.td = R.D.P.T.no; P.t = P.ts + P.td;
%trolley energy
P.TeS = sum(R.S.P.T.e); P.TeD = sum(R.D.P.T.e); P.T = P.TeS + P.TeD;
%trolley time
P.TtS = sum(R.S.P.T.t); P.TtD = sum(R.D.P.T.t); P.Tt = P.TtS+P.TtD;
% hoist number
P.hs = R.S.P.H.no; P.hd = R.D.P.H.no; P.h = P.hs + P.hd;
% Hoist energy
P.HeS = sum(R.S.P.H.e); P.HeD = sum(R.D.P.H.e); P.He = P.HeS + P.HeD;
if R.S.P.H.e <0
   keyboard
end
% Hoist time
P.HtS = sum(R.S.P.H.t); P.HtD = sum(R.D.P.H.t); P.Ht = P.HtS + P.HtD;

% UNPRODUCTIVE
U.gs = R.S.U.G.no ; U.gd = R.D.U.G.no; U.g = U.gs + U.gd;
U.GeS = sum(R.S.U.G.e); U.GeD = sum(R.D.U.G.e); U.G = U.GeS + U.GeD;
U.GtS = sum(R.S.U.G.t) ;U.GtD = sum(R.D.U.G.t); U.GT = sum(R.S.U.G.t) + sum(R.D.U.G.t);

U.ts = R.S.U.T.no; U.td = R.D.U.T.no; U.t = U.ts + U.td;
U.TeS = sum(R.S.U.T.e); U.TeD = sum(R.D.U.T.e); U.Te = U.TeS + U.TeD;
U.TtS = sum(R.S.U.T.t); U.TtD =sum(R.D.U.T.t); U.Tt = U.TtS+ U.TtD;

U.hs = R.S.U.H.no; U.hd = R.D.U.H.no; U.h = U.hs + U.hd;
U.HeS = sum(R.S.U.H.e);  U.HeD = sum(R.D.U.H.e); U.He = U.HeS + U.HeD;
U.HtS = sum(R.S.U.H.t); U.HtD = sum(R.D.U.H.t); U.HT = U.HtS + U.HtD;

U.ws = R.S.U.W.no; U.wd = R.D.U.W.no; U.w = U.ws  + U.wd;
U.WtS = sum(R.S.U.W.t); U.WtD = sum(R.D.U.W.t); U.WT = U.WtS + U.WtD;
% iud = IMP.S.U.D.no+IMP.D.U.D.no;
% IUDT = sum(IMP.S.U.D.t) + sum(IMP.D.U.D.t);
```

```
% total time =
P.sttime = sum(R.S.P.G.eloa.t) + sum(R.S.P.G.eunl.t) + sum(R.S.P.T.t)+ sum(R.S.P.H.t);
U.sttime = sum(R.S.U.T.t) + sum(R.S.U.H.t) + sum(R.S.U.W.t);
P.dttime = sum(R.D.P.G.eloa.t) + sum(R.D.P.G.eunl.t) + sum(R.D.P.T.t)+ sum(R.D.P.H.t);
U.dttime = sum(R.D.U.T.t) + sum(R.D.U.H.t) + sum(R.D.U.W.t);

disp('- STACK ------------------------------------------------------------')
disp(['Gantry  [#/E/t] Prod: [' num2str(P.gs) '/' num2str(P.GeS/n*e_scale) '/' num2str(P.GtS/n) '] Unprod: ['
num2str(U.gs) '/' num2str(U.GeS/n*e_scale) '/' num2str(U.GtS/n) ']']);
disp(['Trolley [#/E/t] Prod: [' num2str(P.ts) '/' num2str(P.TeS/n*e_scale) '/' num2str(P.TtS/n) '] Unprod: [' num2str(U.ts)
'/' num2str(U.TeS/n*e_scale) '/' num2str(U.TtS/n) ']']);
disp(['Hoist   [#/E/t] Prod: [' num2str(P.hs) '/' num2str(P.HeS/n*e_scale) '/' num2str(P.HtS/n) '] Unprod: ['
num2str(U.hs) '/' num2str(U.HeS/n*e_scale) '/' num2str(U.HtS/n) ']']);
disp(['Wait    [#/E/t]                    Unpr: [' num2str(U.ws) '/' num2str(U.WtS/n) ']']);

disp('- DELIVERY ---------------------------------------------------------')
disp(['Gantry  [#/E/t] Prod: [' num2str(P.gd) '/' num2str(P.GeD/n*e_scale) '/' num2str(P.GtD/n) '] Unprod: ['
num2str(U.gd) '/' num2str(U.GeD/n*e_scale) '/' num2str(U.GtD/n) ']']);
disp(['Trolley [#/E/t] Prod: [' num2str(P.td) '/' num2str(P.TeD/n*e_scale) '/' num2str(P.TtD/n) '] Unprod: ['
num2str(U.td) '/' num2str(U.TeD/n*e_scale) '/' num2str(U.TtD/n) ']']);
disp(['Hoist   [#/E/t] Prod: [' num2str(P.hd) '/' num2str(P.HeD/n*e_scale) '/' num2str(P.HtD/n) '] Unprod: ['
num2str(U.hd) '/' num2str(U.HeD/n*e_scale) '/' num2str(U.HtD/n) ']']);
disp(['Wait    [#/E/t]                    Unpr: [' num2str(U.wd) '/' num2str(U.WtD/n) ']']);

disp('-------------------------------------------------------------------')
disp(['Stack num/time: '       num2str(n) '/' num2str((P.sttime+U.sttime)/n)])
disp(['Delivery num/time: '    num2str(n) '/' num2str((P.dttime+U.dttime)/n)])
disp(['Translation num/time: ' num2str(U.g) '/' num2str(U.GT/n)])
disp(['UNPRODUCTIVE STACK time (reshuffles/trans): ' num2str((U.TtS + U.HtS)/n) '/' num2str((U.WtS)/n)]);
disp(['UNPRODUCTIVE DELIVERY time (reshuffles/trans): ' num2str((U.TtD + U.HtD)/n) '/' num2str((U.WtD)/n)]);
disp('-------------------------------------------------------------------')

sep = sum(R.S.P.G.eunl.e)+sum(R.S.P.G.eloa.e)+sum(R.S.U.G.e)+...
   sum(R.S.P.T.e)+sum(R.S.U.T.e)+sum(R.S.P.H.e)+sum(R.S.U.H.e)+ ...
   sum(R.D.P.G.eunl.e)+sum(R.D.P.G.eloa.e)+sum(R.D.U.G.e)+...
   sum(R.D.P.T.e)+sum(R.D.U.T.e)+sum(R.D.P.H.e)+sum(R.D.U.H.e);

if abs(sep-sum(R.E)) >0.0001
   sep-sum(R.E)
   keyboard
end
```

```
function Energy_postprocess()

trf = 1;
if trf == 1 % EXP
   ini_bay = 40; fin_bay = 10; incbay = -1; delb = -5;
elseif trf == 2 % IMP
   ini_bay = 8; fin_bay = 32; incbay = 1; delb = 5;
end


for b1 = ini_bay:incbay:fin_bay
      % a) if rehandling
   Presh1(b1) = exponential(X2,Y2);
      Erehand1(b1) = ERehand(h);

   % a1) if retrieved
   Pretr1(b1) = exponential(X1,Y1);
      Eretrieved1(b1) = E_ASCsea(p50->b1)+E_ASCland(b1->b1+3)

      % a2) if not retrieved

      for b2 = b1+delb:fin_bay
      % jump
      L = (b2-b1);
      if trf == 1 % EXP
         Pjump = -0.0004*L^2+0.0018*L-0.0108;
         Pbay = 1.3281*exp(-0.18*b1);
         Pjb = Pjump*Pbay;
      else
         Pjb = 0.04;
      end
      % if rehandling
      Presh(b2,2) = exponential(X2,Y2);
      Erehand(b2,2) = ERehand(h);

      % if retrieved
      Pretr(b2,2) = exponential(X1,Y1);
      Eretrieved(b2,2) = E_ASCsea(p50->b2)+E_ASCland(b1->b2+3)
      % if not retrieved
      for b3 = b2+delb:fin_bay
         % if rehandling
         Presh(b3) = exponential(X2,Y2);
         Erehand3(b3) = ERehand(h);

         % if retrieved
         Pretr(b3) = exponential(X1,Y1);
         Eretrieved3(b3) = E_ASCsea(p50->b2)+back+E_ASCland(b1->b2+3);
         for b4 = b3+delb:fin_bay
            % if rehandling
            Presh(b4) = exponential(X2,Y2);
            Erehand4(b4) = ERehand(h);

            % if retrieved
```

```
            Pretr(b4) = exponential(X1,Y1);
            Eretrieved4(b4) = E_ASCsea(p50->b2)+back+E_ASCland(b1->b2+3);
            for b5 = b4+delb:fin_bay
               Eretrieved5(b5) = (E_ASCsea(p50->b5)+back+E_ASCland(b4->buffer));
            end
            Enotretrieved4(b4) = Phk(b4) * sum(Eretrieved5);
         end
         Enotretrieved3(b3) = Phk(b3) * (sum(Erehand4) + sum(Eretrieved4) + sum(Enotretrieved4));
      end
      Enotretrieved2(b2) = Phk(b2) * (sum(Erehand3) + sum(Eretrieved3) + sum(Enotretrieved3));
   end
   Enotretrieved1(b1) = Phk(b1) * (sum(Erehand2) + sum(Eretrieved2) + sum(Enotretrieved2));
end

Et = Erehand1 + Eretrieved1 + Enotretrieved1;
```

```
function [S,D,sumae] = Energybreakdown(S,D,movements,energia,tiempo)
% This function gets the stack S and delivery D characteristics for every
% container cycle

le = length(energia);
lm = length(movements);

raisepositions = find(strcmp(movements,'raise'));
dropblpositions = find(strcmp(movements,'dropbl'));

lastrise = raisepositions(end);
firstdropbl = dropblpositions(1);
if length(raisepositions) == 1
   movements
   keyboard
else
   deliveryposition = raisepositions(end-1)+1;
   deliveryposition = dropblpositions(1)+2;
   % found =0; m = 0;
   % while found == 0
   %     m = m+1;
   %     if isempty(movements) == 0
   %         if strcmp(movements(m),'raise') == 1
   %             splitmove = m;
   %             found = 1;
   %         end
   %     else
   %         found = 2; m = 0; keyboard
   %     end
   % end
   %
   % keyboard

   if le-lm> 0
      keyboard
      for i = 1:le-lm
         a{i} = 'trans';
      end
      a{i+1} = 'trans';
      a = [a, movements(2:end)];
      movements = a;
   end

   ng = 0; sumat = 0; t =0; sumae = 0; e = 0; gantrytype = 'U';
   for m = 1:le
      if strcmp(movements(m),'gantry') == 1
         ng = ng+1;
         if ng> 1
            change = 1;
            % By default, change, unless
            if strcmp(movements(m-1),'gantry') == 1
               change = 0;
            elseif strcmp(movements(m-1),'wait') == 1
```

```
            change = 0;
        end

        if change == 1
            if strcmp(gantrytype,'U') == 1
                gantrytype = 'L';
            elseif strcmp(gantrytype,'L') == 1
                gantrytype = 'U';
            end
        else
            %keyboard
        end
      end
    end
    %keyboard
    if m < deliveryposition
      [S.P,S.U,t(m),e(m)] = get_ET(S.P,S.U,movements(m),energia(m),tiempo(m),gantrytype);
      %disp([num2str(m) ' ' num2str(energia(m))])
      sumat=sumat + t(m); sumae=sumae + e(m);
    else
      [D.P,D.U,t(m),e(m)] = get_ET(D.P,D.U,movements(m),energia(m),tiempo(m),gantrytype);
      %disp([num2str(m) ' ' num2str(energia(m))])
      sumat=sumat + t(m); sumae=sumae + e(m);
    end
  end

%    p=find(energia>0); ep=sum(energia(p));
%    sep = sum(S.P.G.eunl.e)+sum(S.P.G.eloa.e)+sum(S.U.G.e)+...
%      sum(S.P.T.e)+sum(S.U.T.e)+sum(S.P.H.e)+sum(S.U.H.e)+ ...
%      sum(D.P.G.eunl.e)+sum(D.P.G.eloa.e)+sum(D.U.G.e)+...
%      sum(D.P.T.e)+sum(D.U.T.e)+sum(D.P.H.e)+sum(D.U.H.e);
%
%    if abs(sep-ep) >0
%       keyboard
%    end


  if abs(sumat-sum(tiempo))>0.001
    tiempo-t
    keyboard
  end
end
```

```
function [S,D,sumae] = Energybreakdown10(S,D,movements,energia,tiempo)
% This function gets the stack S and delivery D characteristics for every
% container cycle

le = length(energia);
lm = length(movements);

raisepositions = find(strcmp(movements,'raise'));
dropblpositions = find(strcmp(movements,'dropbl'));

lastrise = raisepositions(end);
firstdropbl = dropblpositions(1);
if length(raisepositions) == 1
   movements
   keyboard
else
   deliveryposition = raisepositions(end-1)+1;
   deliveryposition = dropblpositions(1)+2;
   % found =0; m = 0;
   % while found == 0
   %     m = m+1;
   %     if isempty(movements) == 0
   %         if strcmp(movements(m),'raise') == 1
   %             splitmove = m;
   %             found = 1;
   %         end
   %     else
   %         found = 2; m = 0; keyboard
   %     end
   % end
   %
   % keyboard

   if le-lm> 0
      keyboard
      for i = 1:le-lm
         a{i} = 'trans';
      end
      a{i+1} = 'trans';
      a = [a, movements(2:end)];
      movements = a;
   end

   ng = 0; sumat = 0; sumae = 0; t =0; gantrytype = 'U';
   for m = 1:le
      if strcmp(movements(m),'gantry') == 1
         ng = ng+1;
         if ng> 1
            change = 1;
            % By default, change, unless
            if strcmp(movements(m-1),'gantry') == 1
               change = 0;
            elseif strcmp(movements(m-1),'wait') == 1
```

```
            change = 0;
        end

        if change == 1
            if strcmp(gantrytype,'U') == 1
                gantrytype = 'L';
            elseif strcmp(gantrytype,'L') == 1
                gantrytype = 'U';
            end
        else
            %keyboard
        end
      end
    end
    %keyboard
    if m < deliveryposition
      [S.P,S.U,t(m),e(m)] = get_ET10(S.P,S.U,movements(m),energia(m),tiempo(m),gantrytype);
      %disp([num2str(m) ' ' num2str(energia(m))])
      sumat=sumat + t(m);
      if e(m)>0
        sumae = sumae+e(m);
      end
    else
      [D.P,D.U,t(m),e(m)] = get_ET10(D.P,D.U,movements(m),energia(m),tiempo(m),gantrytype);
      %disp([num2str(m) ' ' num2str(energia(m))])
      sumat=sumat + t(m);
      if e(m)>0
        sumae = sumae+e(m);
      end
    end
  end

  % sumPS=sum(S.P.G.eunl.e)+sum(S.P.G.eloa.e)+sum(S.P.H.e)+sum(S.P.T.e);
  % sumUS=S.U.G.e+S.U.H.e+S.U.T.e;
  % sumPD=sum(D.P.G.eunl.e)+sum(D.P.G.eloa.e)+sum(D.P.H.e)+sum(D.P.T.e);
  % sumUD=D.U.G.e+D.U.H.e+D.U.T.e;
  % sumae= sumPS+sumPD+sumUS+sumUD;
  if abs(sumat-sum(tiempo))>0.001
    tiempo-t
    keyboard
  end
end
```

```
function [t_gs,asc,found] = ESSA_search_drop(ct,asc,demand)
% This function makes a block search for candidate slots to drop a CT
global ASC BAYS BF BL CT S TRF

found = true;

AGS = 0; BGS = 0; CGS = 0; DGS = 0;
acs = 0; bcs = 0; ccs = 0; dcs = 0; t_gs = 0;
RT = 0; RE = 0; PT = 0; PE = 0; PD = 0; PEC = 0; D = 0; E=0;

[origin] = CT_act_pos(ct);

if asc.housekeeping == 1
   modo = 'hkstack';
else
   modo = asc.tasks.action(asc.tasks.current);
   if strcmp(modo,'delivery')==1
      keyboard
   end
end

[minbay,maxbay] = BAY_limit(CT(ct).id,modo,demand);

% 1 Determine the limiting bay for a crane
if strcmp(asc.id,'sea') == 1
   if strcmp(ASC.land.status,'wait')== 1
      ascpos = ASC_act_pos(ASC.land); limit_bay = ascpos.bay;
   else
      [limit_bay] = ASC_target_bay(ASC.land);
   end
elseif strcmp(asc.id,'land') == 1
   if strcmp(ASC.sea.status,'wait')== 1
      ascpos = ASC_act_pos(ASC.sea); limit_bay = ascpos.bay;
   else
      [limit_bay] = ASC_target_bay(ASC.sea);
   end
end
%keyboard

[ini_gs, end_gs, del_gs] = ASC_bay_direction(asc);

% if strcmp(asc.id,'land') == 1
%     [conflicts_gs] = BAY_prevent_conflicts2(gs,asc,ASC.sea);
% elseif strcmp(asc.id,'sea') == 1
%     [conflict_gs] = BAY_prevent_conflicts2(gs,asc,ASC.land);
% end

% Satisfy the criteria to be a candidate slot
for gs = ini_gs:del_gs:end_gs

   bay = BL.GS(gs).bay;

   [bocup,bres] = BAY_occupation(bay); % Bay occupation
```

```
   if bocup+bres > BL.tiers*BL.stacks - BL.tiers;
     continue
   end

   if BL.GS(gs).ocup == BL.tiers
      continue
   end

   % b) This is to prevent a worse position during housekeeping
   if asc.housekeeping == 1
      [conflict] = BAY_backwards(bay,ct);
      if conflict == 1
         continue
      end
   end

   % c) Special criteria: if the other asc is delivering, don't put
%     if isempty(find(conflict_gs == gs))
%        keyboard
%        continue
%     end

   if strcmp(asc.id,'land') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.sea);
   elseif strcmp(asc.id,'sea') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.land);
   end
   if conflict == 1
      continue
   end

   % f) Traffic flow.
   bayflow = BL.GS(gs).id;
   destiny = BL_act_pos(gs); % BL_act_pos gives the tier = ocup + 1

   % IMPORT CT
   % ---------------------------------------------------------------
   if strcmp(CT(ct).id,'IMP') == 1 % Import flow. NAS or IMP are ok
      if strcmp(bayflow,'EXP') == 1
         continue
      end
      if bocup < TRF.PARAM.baymaxocup

      % if and(bay < BL.hklimit.land, bay > BL.hklimit.sea)
         if and(BL.GS(gs).ocup > 0, BL.GS(gs).ocup < TRF.PARAM.maxh)
            acs=acs+1;  AGS(acs) = gs;
            ABAYS(acs) = bay;
            [AC(acs),AR(acs)] = ASC_ETM(asc,ct,origin,destiny,1);
            [ART(acs),APT(acs),ARE(acs),APE(acs)] = Stack_coefs(AC(acs),AR(acs));
         elseif BL.GS(gs).ocup == 0
            bcs = bcs+1; BGS(bcs) = gs;
            BBAYS(bcs) = bay;
            [BC(bcs),BR(bcs)] = ASC_ETM(asc,ct,origin,destiny,1);
            [BRT(bcs),BPT(bcs),BRE(bcs),BPE(bcs)] = Stack_coefs(BC(bcs),BR(bcs));
```

```
        else
            ccs = ccs+1; CGS(ccs) = gs;
            CBAYS(ccs) = bay;
            [CC(ccs),CR(ccs)] = ASC_ETM(asc,ct,origin,destiny,1);
            [CRT(ccs),CPT(ccs),CRE(ccs),CPE(ccs)] = Stack_coefs(CC(ccs),CR(ccs));
        end
    else
        dcs = dcs+1; DGS(dcs) = gs;
        DBAYS(dcs) = bay;
        [DC(dcs),DR(dcs)] = ASC_ETM(asc,ct,origin,destiny,1);
        [DRT(dcs),DPT(dcs),DRE(dcs),DPE(dcs)] = Stack_coefs(DC(dcs),DR(dcs));
    end
% ---------------------------------------------------------------------
elseif strcmp(CT(ct).id,'EXP') == 1 % Export flow. NAS or EXP are ok
    if strcmp(bayflow,'IMP') == 1
        continue
    end
    if bocup > BL.tiers*BL.stacks
        continue
    end
        % c) Bay constraint
    %if and(bay < BL.hklimit.land, bay > BL.hklimit.sea)

        if BL.GS(gs).group > 0 % For bays with a group assigned
            if BL.GS(gs).group == CT(ct).group
                %if TRF.VS(CT(ct).vs).line == TRF.PARAM.CT.groups(CT(ct).group);
                    acs = acs+1;  AGS(acs) = gs;
                    ABAYS(acs) = bay;
                    [AC(acs),AR(acs)] = ASC_ETM(asc,ct,origin,destiny,1);
                    [ART(acs),APT(acs),ARE(acs),APE(acs)] = Stack_coefs(AC(acs),AR(acs));
                else
                    ccs = ccs+1; CGS(ccs) = gs;
                    CBAYS(ccs) = bay;
                    [CC(ccs),CR(ccs)] = ASC_ETM(asc,ct,origin,destiny,1);
                    [CRT(ccs),CPT(ccs),CRE(ccs),CPE(ccs)] = Stack_coefs(CC(ccs),CR(ccs));
                end
%               else
%                   dcs = dcs+1;
%                   DGS(dcs) = gs; DBAYS(dcs) = bay;
%                   [DC(dcs),DR(dcs)] = ASC_ETM(asc,ct,origin,destiny,1);
%                   [DRT(dcs),DPT(dcs),DRE(dcs),DPE(dcs)] = Stack_coefs(DC(dcs),DR(dcs));
%               end
        else
            bcs = bcs+1; BGS(bcs) = gs;
            BBAYS(bcs) = bay;
            [BC(bcs),BR(bcs)] = ASC_ETM(asc,ct,origin,destiny,1);
            [BRT(bcs),BPT(bcs),BRE(bcs),BPE(bcs)] = Stack_coefs(BC(bcs),BR(bcs));
        end
%       else
%           dcs = dcs+1;
%           DGS(dcs) = gs; DBAYS(dcs) = bay;
%           [DC(dcs),DR(dcs)] = ASC_ETM(asc,ct,origin,destiny,1);
%           [DRT(dcs),DPT(dcs),DRE(dcs),DPE(dcs)] = Stack_coefs(DC(dcs),DR(dcs));
%       end
```

```
    end
end

if acs>0
    ns = acs;     GS = AGS; C = AC; R = AR;
    CT(ct).strategy{end+1} = 'A';
    TBAYS = ABAYS;
    RE = ARE;
    PE = APE;
    RT = ART;
    PT = APT;
elseif bcs>0
    ns = bcs;     GS = BGS; C = BC; R = BR;
    CT(ct).strategy{end+1} = 'B';
    TBAYS = BBAYS;
    RE = BRE;
    PE = BPE;
    RT = BRT;
    PT = BPT;
elseif and(ccs >0, asc.housekeeping == 0) % if HK, do not mix so much
    ns = ccs;    GS =CGS; C = CC; R = CR;
    CT(ct).strategy{end+1} = 'C';
    TBAYS=CBAYS;
    RE = CRE;
    PE = CPE;
    RT = CRT;
    PT = CPT;
elseif dcs >0
    ns = dcs;    GS =DGS; C = DC; R = DR;
    CT(ct).strategy{end+1} = 'D';
    TBAYS = DBAYS;
    RE = DRE;
    PE = DPE;
    RT = DRT;
    PT = DPT;
else
    found = false;
    if asc.housekeeping == 0
       disp('ESSA search warning: no slot found');
    end
end

% For non priority HK, avoid mixing in piles
% if asc.housekeeping == 1
%     if CT(ct).priority == 1
%         if and(acs == 0, bcs == 0)
%             found = false;
%         end
%     end
% end

% Determine the coeficients of the candidate slots
if found
    CT(ct).strategypossibilities(end+1) = ns;
```

```
   % Sort the bays BGS
   baylist = unique(TBAYS);

   % Potential energy due to crane interference
   if ns > 1
      for ibay = 1:length(baylist)
         bay = baylist(ibay);
         %keyboard
         lista = find(TBAYS == bay);
         ics = lista(1);
         D(bay)= 0; E(bay)=0;
         if strcmp(asc.id,'sea') == 1
            if bay > limit_bay - S.baymargin
               [D(bay),E(bay)] = ASC_pot_interv2(ASC.land,C(ics));
            end
         elseif strcmp(asc.id,'land') == 1
            if bay < limit_bay + S.baymargin
               [D(bay),E(bay)] = ASC_pot_interv2(ASC.sea,C(ics));
            end
         end
      end

      if sum(D+E) > 0
         for i = 1:ns
            bay = TBAYS(i);
            PD(i) = D(bay);
            PEC(i) = E(bay);
         end
      end

      % Assign the delay and energy to the candidate slots
      E_max = max(RE+PE+PEC); t_max = max(RT+PT+PD);

      % CRITERIA TO SELECT THE CONTAINERS
      if sum(D+E) > 0
         %keyboard
         COEFS = (RE+PE+PEC)/E_max*TRF.PARAM.weight.E + (RT+PT+PD)/t_max*TRF.PARAM.weight.t;
      else
         %keyboard
         COEFS = (RE+PE)/E_max*TRF.PARAM.weight.E + (RT+PT)/t_max*TRF.PARAM.weight.t;
      end
      [c_min,islot] = min(COEFS);
   else
      islot = 1;
   end

   % plot_COEFS(CGS,COEFS)

   % Select ground slot
   t_gs = GS(islot);
end
```

```
function ET_pick_CT()
% This function selects randomly a container from an
% IMP bay so that a a ET can take it

global ASC COUNT CT

% 1. Find a CT based on the time they arrived to the terminal

[target_ct] = BL_search_pick1();

if target_ct > 0

else

   %plot_BL
   %keyboard
end
```

```
function [yes_no]=figure_check(fig_number)

list_figures=findobj('type','figure')';

yes_no=0;
for i=1:length(list_figures)
    if list_figures(i)==fig_number
        yes_no=1;
    end
end
```

```
function figure_close_ifexists(fig_number)

[fig_exist] = figure_check(fig_number);
if fig_exist==1 % there is a figure, close it
   close(fig_number)
end
```

```
clear
% Get all PDF files in the current folder
files = dir('*.m');
% Loop through each
for id = 1:length(files)
    % Get the file name (minus the extension)
    fn = files(id).name;
    if strcmp(fn(end-1:end),'.m')==1
    [~, g] = fileparts(fn);
      % Convert to number
      num = str2double(g);
      sourcef =  strcat(g ,'.m');
      destf = strcat(g , '.txt');
      copyfile(sourcef,destf,'f');
      %if ~isnan(num)
         % If numeric, rename
         %movefile(files(id).name, sprintf('%03d.pdf', num));

      end
end
```

```
function FILL_init(siono)

global ASC CT
ncts = 720
for ct = 1:ncts
   seed = random('unif',0,1);
   if seed < 0.5
      strcmp(CT(ct).id, 'EXP')  == 1;
      gs = TERCAT_search_drop(ct,ASC.land,'stack');
   else
      CT(ct).id == 'IMP';
      CT(ct).dwell = exprnd(TRF.PARAM.av_dwell*3600*24);
      gs = TERCAT_search_drop(ct,ASC.sea,'stack');
   end
   BL_dropCT(gs,ct,0);

end
```

```
function [P,U,t,e] = get_ET(P,U,move,energia,tiempo,gantrytype)

t =0; e = 0;
% if energia < 0
%     keyboard
% end
% Productive moves
% ------------------------------------------------------------------------
if strcmp(move,'gantry') == 1
       if strcmp(gantrytype,'U') == 1
          P.G.eunl.no = P.G.eunl.no + 1; n = P.G.eunl.no;
          P.G.eunl.e(n) =  energia; e = energia;
          P.G.eunl.t(n) =  tiempo; t = tiempo;
       elseif strcmp(gantrytype,'L') == 1
          P.G.eloa.no = P.G.eloa.no + 1; n = P.G.eloa.no;
          P.G.eloa.e(n) =  energia; e = energia;
          P.G.eloa.t(n) =  tiempo; t = tiempo;
       end
elseif strcmp(move,'trolley') == 1
       P.T.no = P.T.no + 1; n = P.T.no;
       P.T.e(n) = energia; e = energia;
       P.T.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'lower') == 1
       P.L.no = P.L.no + 1; n = P.L.no;
       P.L.e(n) = energia; e = energia; if energia >0, keyboard, end
       P.L.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'pickbf') == 1
       P.H.no = P.H.no + 1; n = P.H.no;
       P.H.e(n) = energia; e = energia;
       P.H.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'dropbf') == 1
       P.L.no = P.L.no + 1; n = P.L.no;
       P.L.e(n) = energia; e = energia; if energia >0, keyboard, end
       P.L.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'dropbl') == 1
       P.L.no = P.L.no + 1; n = P.L.no;
       P.L.e(n) = energia; e = energia; if energia >0, keyboard, end
       P.L.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'raise') == 1
       P.H.no = P.H.no + 1; n = P.H.no;
       P.H.e(n) = energia; e = energia;
       P.H.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'pickbl') == 1
       P.H.no = P.H.no + 1; n = P.H.no;
       P.H.e(n) = energia; e = energia;
       P.H.t(n) = tiempo; t = tiempo;
% Unproductive moves
% ------------------------------------------------------------------------
elseif strcmp(move,'transtranstrans') == 1
       U.G.no = U.G.no + 1; n = U.G.no;
       U.G.e(n) =  energia; e = energia;
       U.G.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'transtrans') == 1
```

```
        U.G.no = U.G.no + 1; n = U.G.no;
        U.G.e(n) = energia; e = energia;
        U.G.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'trans') == 1
        U.G.no = U.G.no + 1; n = U.G.no;
        U.G.e(n) =  energia; e = energia;
        U.G.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'ugantry') == 1
        U.G.no = U.G.no + 1; n = U.G.no;
        U.G.e(n) =  energia; e = energia;
        U.G.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'utrolley') == 1
        U.T.no = U.T.no + 1; n = U.T.no;
        U.T.e(n) =  energia; e = energia;
        U.T.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'upickbl') == 1
        U.H.no = U.H.no + 1; n = U.H.no;
        U.H.e(n) =  energia; e = energia;
        U.H.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'uraise') == 1
        U.H.no = U.H.no + 1; n = U.H.no;
        U.H.e(n) = energia; e = energia;
        U.H.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'ulower') == 1
        U.L.no = U.L.no + 1; n = U.L.no;
        U.L.e(n) = energia; e = energia; if energia >0, keyboard, end
        U.L.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'udropbl') == 1
        U.L.no = U.L.no + 1; n = U.L.no;
        U.L.e(n) = energia; e = energia; if energia >0, keyboard, end
        U.L.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'wait') == 1
        U.W.no = U.W.no +1; n = U.W.no;
        U.W.t(n) = tiempo; t = tiempo;
else
   keyboard
end

if e < 0
   e = 0;
end
```

```
function [P,U,t,e] = get_ET10(P,U,move,energia,tiempo,gantrytype)

t =0; e = 0;

% Productive moves
% --------------------------------------------------------------------
if strcmp(move,'gantry') == 1
       if strcmp(gantrytype,'U') == 1
          P.G.eunl.no = P.G.eunl.no + 1; n = P.G.eunl.no;
          P.G.eunl.e(n) =  energia/10; e = energia/10;
          P.G.eunl.t(n) =  tiempo; t = tiempo;
       elseif strcmp(gantrytype,'L') == 1
          P.G.eloa.no = P.G.eloa.no + 1; n = P.G.eloa.no;
          P.G.eloa.e(n) =  energia/10; e = energia/10;
          P.G.eloa.t(n) =  tiempo; t = tiempo;
       end
elseif strcmp(move,'trolley') == 1
       P.T.no = P.T.no + 1; n = P.T.no;
       P.T.e(n) = energia/10; e = energia/10;
       P.T.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'lower') == 1
       P.L.no = P.L.no + 1; n = P.L.no;
       P.L.e(n) = energia; e = energia;
       P.L.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'pickbf') == 1
       P.H.no = P.H.no + 1; n = P.H.no;
       P.H.e(n) = energia; e = energia;
       P.H.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'dropbf') == 1
       P.L.no = P.L.no + 1; n = P.L.no;
       P.L.e(n) = energia;e = energia;
       P.L.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'dropbl') == 1
       P.L.no = P.L.no + 1; n = P.L.no;
       P.L.e(n) = energia; e = energia;
       P.L.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'raise') == 1
       P.H.no = P.H.no + 1; n = P.H.no;
       P.H.e(n) = energia; e = energia;
       P.H.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'pickbl') == 1
       P.H.no = P.H.no + 1; n = P.H.no;
       P.H.e(n) = energia; e = energia;
       P.H.t(n) = tiempo; t = tiempo;
% Unproductive moves
% --------------------------------------------------------------------
elseif strcmp(move,'transtranstrans') == 1
       U.G.no = U.G.no + 1; n = U.G.no;
       U.G.e(n) =  energia/10; e = energia/10;
       U.G.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'transtrans') == 1
       U.G.no = U.G.no + 1; n = U.G.no;
       U.G.e(n) = energia/10; e = energia/10;
```

```
        U.G.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'trans') == 1
        U.G.no = U.G.no + 1; n = U.G.no;
        U.G.e(n) =  energia/10; e = energia/10;
        U.G.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'ugantry') == 1
        U.G.no = U.G.no + 1; n = U.G.no;
        U.G.e(n) =  energia/10; e = energia/10;
        U.G.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'utrolley') == 1
        U.T.no = U.T.no + 1; n = U.T.no;
        U.T.e(n) =  energia/10; e = energia/10;
        U.T.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'upickbl') == 1
        U.H.no = U.H.no + 1; n = U.H.no;
        U.H.e(n) =  energia; e = energia;
        U.H.t(n) =  tiempo; t = tiempo;
elseif strcmp(move,'uraise') == 1
        U.H.no = U.H.no + 1; n = U.H.no;
        U.H.e(n) = energia; e = energia;
        U.H.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'ulower') == 1
        U.L.no = U.L.no + 1; n = U.L.no;
        U.L.e(n) = energia; e = energia;
        U.L.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'udropbl') == 1
        U.L.no = U.L.no + 1; n = U.L.no;
        U.L.e(n) = energia; e = energia;
        U.L.t(n) = tiempo; t = tiempo;
elseif strcmp(move,'wait') == 1
        U.W.no = U.W.no +1; n = U.W.no;
        U.W.t(n) = tiempo; t = tiempo;
else
   keyboard
end
```

```
function GROUP_init()

global TRF

%keyboard
no_groups = TRF.PARAM.no.pod*TRF.PARAM.no.lines*TRF.PARAM.no.ct_weights;
TRF.PARAM.CT.groups = zeros(no_groups,4);
g =0;

for pod = 1:TRF.PARAM.no.pod
   for line = 1:TRF.PARAM.no.lines
      for weight = 1:TRF.PARAM.no.ct_weights
         g = g+1;
         TRF.PARAM.CT.groups(g,1) = g;
         TRF.PARAM.CT.groups(g,2) = pod;
         TRF.PARAM.CT.groups(g,3) = line;
         TRF.PARAM.CT.groups(g,4) = weight;
      end
   end
end

TRF.PARAM.CT.no_groups = g;
```

```
function HK_check(ct)

global ASC CT

ctid = CT(ct).id;
%keyboard
if strcmp(ctid,'IMP') == 1
    if CT(ct).priority == 2
        pos = find(ASC.hktasks.prior.IMP == ct);
        if isempty(pos)
            keyboard
        end
    elseif CT(ct).priority == 1
        pos = find(ASC.hktasks.nonprior.IMP == ct);
        if isempty(pos)
            keyboard
        end
    end
elseif strcmp(ctid,'EXP') == 1
    if CT(ct).priority == 2
        pos = find(ASC.hktasks.prior.EXP == ct);
        if isempty(pos)
            keyboard
        end
    elseif CT(ct).priority == 1
        pos = find(ASC.hktasks.nonprior.EXP == ct);
        if isempty(pos)
            keyboard
        end
    end
end
```

```
function [priority] = HK_ct_priority(ct)
% This fucntion checks whether a ct has priority or not

global BAYS BL CT TIME TRF

%stopatct(ct,1523);
old_priority = CT(ct).priority;

seabay = BL.hklimit.sea;   seags = BAYS(seabay).GS(1);
landbay = BL.hklimit.land; landgs = BAYS(landbay).GS(end);

gs = CT(ct).position.gs(end);

if strcmp(CT(ct).id, 'EXP') == 1
   % Get the time
   if or(gs <=0, gs < seags)
      priority = 0;
   else
      avl = 0; ivl = 0; %keyboard
      for vs = 1:TIME.simulation.days + 1
         % Determine the number of vessels present at the facility
         if TRF.VS(vs).active == 1
            avl = avl +1;
            AVL(avl) = TRF.VS(vs).line;
         end
         % Vessels which will come in the next 24 hrs
         t_left = TRF.VS(vs).arrival - TIME.t;
         if and(t_left >0, t_left < 24*3600)
            ivl = ivl +1;              %keyboard
            IVL(ivl) = TRF.VS(vs).line;
         end
      end

      group = CT(ct).group;
      ctline = TRF.PARAM.CT.groups(group,3);
      % Check if ct belongs to a vessel coming in the next 24h
      priority = 1;
      if ivl > 0
         if isempty (find(IVL == ctline))==0 %ctline == vsline
            priority = 2;
         end
      end
      if avl > 0
         if isempty (find(AVL == ctline))==0 %ctline == vsline
            priority = 2;
         end
      end
   end
elseif strcmp(CT(ct).id, 'IMP') == 1
   if or(gs <= 0, gs > landgs)
      priority = 0;
   else
      if CT(ct).position.time(1) + CT(ct).dwell - TIME.t < 0.5*60*60 % hrs*mins*segs
```

```
            priority = 2; %keyboard
        else
            priority = 1;
        end
    end
end
add = 0; remove = 0;
if abs(priority-old_priority) > 0
    %keyboard
    if old_priority > 0
        remove = 1;
    end
    if priority > 0
        add = 1;
    elseif priority == 0
        remove = 1;
    end
elseif priority - old_priority < 0
    keyboard
end

if remove == 1
    %keyboard
    HK_list_modify(ct,-1);
    CT(ct).priority = priority;
end
if add == 1
    CT(ct).priority = priority;
    HK_list_modify(ct,1); % 1 to add, -1 to remove from list
end

HK_check(ct)
```

```
function [fctlist] = HK_explore_block(ctlist)

global ASC BAYS BL CT

fctlist = []; fctlist2 = [];
CGS=0;
if isempty(ctlist) == 0
   flow = CT(ctlist(1)).id;
   seabay = BL.hklimit.sea; sea_gs = BAYS(seabay).GS(1);
   landbay = BL.hklimit.land;  land_gs = BAYS(landbay).GS(end);

   if strcmp(flow,'IMP') == 1
      inigs = sea_gs; endgs = BL.no_gs; delgs = -1;
   elseif strcmp(flow,'EXP') == 1
      inigs = 1; endgs = land_gs; delgs = 1;
   end
   cgs=0;
   for gs = inigs:delgs:endgs
      if strcmp(BL.GS(gs).id,'flow') == 1
         continue
      end

      if BL.GS(gs).ocup < BL.tiers
         cgs = cgs+1;
         CGS(cgs,1) = gs;
         CGS(cgs,2) = BL.GS(gs).ocup;
         CGS(cgs,3) = BL.GS(gs).group;
         CGS(cgs,4) = BL.GS(gs).bay;
      end
   end
%    tgr = unique(CGS(:,3));
%     for i = 1:length(ctlist)
%        ctgroups(i) = CT(i).group;
%     end
%     grlist = unique(ctgroups);
   % Filter the container list
   j=0; j2 = 0;
   for i=1:length(ctlist)
      ct = ctlist(i);
      for k = 1:cgs
         % Compare groups
         if CGS(cgs,3) == CT(ct).group
            go = 0;
            % Compare bays
            if strcmp(flow,'EXP') == 1
               if CT(ct).position.bay(end)-CGS(cgs,4) > 5
                  go = 1;
               end
            elseif strcmp(flow,'IMP') == 1
               if CGS(cgs,4)-CT(ct).position.bay(end) > 5
                  go = 1;
               end
            end
```

```
            if go == 1
               j = j+1; fctlist(j) = ct;
               break
            end
         else
            go2 = 0;
            % Compare bays
            if strcmp(flow,'EXP') == 1
               if CT(ct).position.bay(end)-CGS(cgs,4) > 5
                  go2 = 1;
               end
            elseif strcmp(flow,'IMP') == 1
               if CGS(cgs,4)-CT(ct).position.bay(end) > 5
                  go2 = 1;
               end
            end
            if go2 == 1
               j2 = j2+1; fctlist2(j2) = ct;
               break
            end
         end
      end
   end
end

if isempty(fctlist)
   if isempty(fctlist2)
      %keyboard
   else
      fctlist = fctlist2;
   end
end
% ASC.hktasks.cgs = CGS(:,1);
```

```
function [fct_list] = HK_get_list()
% This function chooses the mosr prior HK list depending on the time of the
% day

global ASC

ct_list = []; ct_list2 = []; ct_list3 = []; ct_list4 = [];
imp_list = length(ASC.hktasks.prior.IMP);
exp_list = length(ASC.hktasks.prior.EXP);

[day,hour] = TIME_split();
% First, the prior HK is analyzed
if and(hour > 7, hour < 21) % Day: IMP priority
    if imp_list > 0
        ct_list = ASC.hktasks.prior.IMP;
        ct_list2 = ASC.hktasks.prior.EXP;
        ct_list3 = ASC.hktasks.nonprior.IMP;
        ct_list4 = ASC.hktasks.nonprior.EXP;
    elseif exp_list >0
        ct_list = ASC.hktasks.prior.EXP;
        ct_list2 = ASC.hktasks.prior.IMP;
        ct_list3 = ASC.hktasks.nonprior.EXP;
        ct_list4 = ASC.hktasks.nonprior.IMP;
    elseif length(ASC.hktasks.nonprior.IMP) > 0
        ct_list = ASC.hktasks.nonprior.IMP;
        ct_list2 = ASC.hktasks.nonprior.EXP;
    elseif length(ASC.hktasks.nonprior.EXP) > 0
        ct_list = ASC.hktasks.nonprior.EXP;
        ct_list2 = ASC.hktasks.nonprior.IMP;
    end
else % night. check EXP first
    if exp_list > 0
        ct_list = ASC.hktasks.prior.EXP;
        ct_list2 = ASC.hktasks.prior.IMP;
        ct_list3 = ASC.hktasks.nonprior.EXP;
        ct_list4 = ASC.hktasks.nonprior.IMP;
    elseif imp_list >0
        ct_list = ASC.hktasks.prior.IMP;
        ct_list2 = ASC.hktasks.prior.EXP;
        ct_list3 = ASC.hktasks.nonprior.IMP;
        ct_list4 = ASC.hktasks.nonprior.EXP;
    elseif length(ASC.hktasks.nonprior.EXP) > 0
        ct_list = ASC.hktasks.nonprior.EXP;
        ct_list2 = ASC.hktasks.nonprior.IMP;
    elseif length(ASC.hktasks.nonprior.IMP) > 0
        ct_list = ASC.hktasks.nonprior.IMP;
        ct_list2 = ASC.hktasks.nonprior.EXP;
    end
end
% In any case, the EXP list must preveal
if exp_list > 0
    ct_list = ASC.hktasks.prior.EXP;
    ct_list2 = ASC.hktasks.prior.IMP;
```

```
end
% Now filter the list to take into account the possiblities of each
% containers
[fct_list] = HK_explore_block(ct_list);

% use the other list if CTs cannot be housekeeped
if isempty(fct_list)
   [fct_list] = HK_explore_block(ct_list2);
end

if isempty(fct_list)
   [fct_list] = HK_explore_block(ct_list3);
end

if isempty(fct_list)
   [fct_list] = HK_explore_block(ct_list4);
end
```

```
function [fct_list,priority] = HK_list_analyze(flow,ct_list,IVL,AVL)
% This function takes a ct_list and checks whether the CTs are gonna suffer
% Pre-move or Pre-positioning

global CT TIME TRF

%keyboard
csp = 0; csn = 0; CSP = 0; CSN = 0;

if strcmp(flow,'EXP') == 1
   for i = 1:length(ct_list)
      ct = ct_list(i);
      group = CT(ct).group;
      ctline = TRF.PARAM.CT.groups(group,3);
      if isempty(IVL) == 0
         if isempty (find(IVL == ctline))==0 %ctline == vsline
            csp = csp +1;
            CSP(csp) = ct;
         else
            csn = csn + 1;
            CSN(csn) = ct;
         end
      elseif isempty(AVL) == 0
         if isempty (find(AVL == ctline))==0 %ctline == vsline
            csp = csp +1;
            CSP(csp) = ct;
         else
            csn = csn +1;
            CSN(csn) = ct;
         end
      else
         csn = csn +1;
         CSN(csn) = ct;
      end
   end

   % IMP CTS
elseif strcmp(flow,'IMP') == 1
   for i = 1:length(ct_list)
      ct = ct_list(i);
      if CT(ct).position.time(1) + CT(ct).dwell - TIME.t < 2*60*60 % hrs*mins*segs
         csp = csp +1;
         CSP(csp) = ct;
      else
         csn = csn +1;
         CSN(csn) = ct;
      end
   end
end

% Analyze the lists
if csp > 0
   priority = 1;
```

```
    fct_list = CSP;
elseif csn > 0
    priority = 2;
    fct_list = CSP;
else
    priority = 0;
    fct_list = 0;
end
```

```
function HK_list_modify(ct,add_or_remove)

global ASC CT

% add to the list
if add_or_remove == 1
   if strcmp(CT(ct).id,'IMP') == 1
      if CT(ct).priority == 2
         ASC.hktasks.prior.IMP(end+1) = ct; caso = 1;
      else
         ASC.hktasks.nonprior.IMP(end+1) = ct; caso = 2;
      end
   elseif strcmp(CT(ct).id,'EXP') == 1
      if CT(ct).priority == 2
         ASC.hktasks.prior.EXP(end+1) = ct; caso = 3;
      else
         ASC.hktasks.nonprior.EXP(end+1) = ct; caso = 4;
      end
   end
% Remove from the list
elseif add_or_remove == -1
   if strcmp(CT(ct).id,'IMP') == 1
      if CT(ct).priority == 2
         pos = find(ASC.hktasks.prior.IMP == ct);
         if isempty(pos)
            keyboard
         else
            ASC.hktasks.prior.IMP(pos) = []; caso = 5;
         end
      elseif CT(ct).priority == 1
         pos = find(ASC.hktasks.nonprior.IMP == ct);%keyboard
         if isempty(pos)
            keyboard
         else
            ASC.hktasks.nonprior.IMP(pos) = []; caso = 6;
         end
      end
   elseif strcmp(CT(ct).id,'EXP') == 1

      if CT(ct).priority == 2
         pos = find(ASC.hktasks.prior.EXP == ct); caso = 7;
         ASC.hktasks.prior.EXP(pos) = [];
      elseif CT(ct).priority == 1
         pos = find(ASC.hktasks.nonprior.EXP == ct); caso = 8;
         ASC.hktasks.nonprior.EXP(pos) = [];
      end
   end
end
```

```
function [sorted_list] = HK_list_sort(list,id)

global BL CT

sorted_list = []; idbt= [];

for i = 1:length(list)
   idbt(i,1) = i;
   idbt(i,2) = CT(list(i)).position.bay(end);
   gs =  CT(list(i)).position.gs(end); % Calcualte cts on top
   idbt(i,3) = BL.GS(gs).ocup - CT(list(i)).position.tier(end);
end

%keyboard
if isempty(idbt) == 0
    % First sort by bay
    idbt = sortrows(idbt,2);
    if strcmp(id,'IMP')==1
        idbt(:,:) = idbt(end:-1:1,:);
    else
        %keyboard
    end
    % Second sort by tier
    idbt = sortrows(idbt,3); %The order will be reversed, we order
    %idbt(:,:) = idbt(end:-1:1,:);

    sorted_list = list(idbt(:,1));
end
```

```
function INIT(i)
% This function initializes the terminal

global BF BL SEA_DELIVERY TIME

TIME_init()

BL_init()

BF_init() % After BL

ASC_init() % After BL

COST_init()

CT_init()

TRF_init2(i)
%repeat  to overwrite
overw = 0;
if overw == 1
BF.stacks = BL.stacks; BF.tiers = 20;
BF.size = BF.tiers * BF.stacks;
BF.sea.exp.slots = zeros(BF.tiers,BF.stacks);
BF.sea.exp.cts = zeros(BF.tiers,BF.stacks);
BF.sea.imp.slots = zeros(BF.tiers,BF.stacks);
BF.sea.imp.cts = zeros(BF.tiers,BF.stacks);
BF.land.exp.slots = zeros(BF.tiers,BF.stacks);
BF.land.exp.cts = zeros(BF.tiers,BF.stacks);
BF.land.imp.slots = zeros(BF.tiers,BF.stacks);
BF.land.imp.cts = zeros(BF.tiers,BF.stacks);
BF.land.exp.bay = BL.bays + 2;
BF.land.imp.bay = BL.bays + 3;
end
%FILL_init()

COUNT_init()

%GROUP_init()

LIMITS_init()

INTERS_init()

for day = 1:TIME.simulation.days + 2
   SEA_DELIVERY(day).ct = [];
   SEA_DELIVERY(day).sea_reservation = 0;
end
%load('PT.mat');

%global PT
```

```
function [XSo,TSo,XLo,TLo,LCbays,LCtime,SCbays,SCtime] = INTERS_asc_trajectories()

global ASC TIME

xs = ASC_act_pos(ASC.sea);  XS(1) = xs.bay; TS(1) = xs.time;
xl = ASC_act_pos(ASC.land); XL(1) = xl.bay; TL(1) = xl.time;

delayfactor = 1.0; go = 1;

% There are several posibilities depending on the cranes status
if and(strcmp(ASC.sea.status,'wait') == 1, strcmp(ASC.land.status,'wait') ~= 1)
   % SEA is idle, land is not
   Ltask = ASC.land.ciclo.c_task;
   LCbays = ASC.land.ciclo.bay(Ltask:end);
   LCtime = ASC.land.ciclo.time(Ltask:end);
   XL = [XL,LCbays];
   tl = AUX_vect_ac(LCtime) + TIME.t;
   TL = [TL,tl];
   XS = ones(1,length(XL))*xs.bay;
   TS = [TS,TL(2:end)];
elseif and(strcmp(ASC.sea.status,'wait') ~= 1, strcmp(ASC.land.status,'wait') == 1)
   % LAND is idle, SEA is not
   Stask = ASC.sea.ciclo.c_task;
   SCbays = ASC.sea.ciclo.bay(Stask:end);
   SCtime = ASC.sea.ciclo.time(Stask:end);
   XS = [XS,SCbays];
   ts = AUX_vect_ac(SCtime) + TIME.t;
   TS = [TS,ts];
   XL = ones(1,length(XS))*xl.bay;
   TL = [TL,TS(2:end)];
elseif and(strcmp(ASC.sea.status,'wait') ~= 1, strcmp(ASC.land.status,'wait') ~= 1)
   % Both cranes are busy
   Ltask = ASC.land.ciclo.c_task;
   LCbays = ASC.land.ciclo.bay(Ltask:end);
   LCtime = ASC.land.ciclo.time(Ltask:end);
   XL = [XL,LCbays];
   tl = AUX_vect_ac(LCtime) + TIME.t;
   TL = [TL,tl];
   Stask = ASC.sea.ciclo.c_task;
   SCbays = ASC.sea.ciclo.bay(Stask:end);
   SCtime = ASC.sea.ciclo.time(Stask:end);
   XS = [XS,SCbays];
   ts = AUX_vect_ac(SCtime) + TIME.t;
   TS = [TS,ts];
else
   disp('error'); go = 0; keyboard
end

% Filter the vectors for intersection
if go == 1
   [XSo,TSo] = AUX_filter_repeated(XS,TS);
   [XLo,TLo] = AUX_filter_repeated(XL,TL);
end
```

```
function [num,side] = INTERS_crosses(X1,T1,ibay,itime)


X2 = [ibay,ibay];
T2 = [T1(1),T1(end)];
side = '';
[ix,iy,iout,jout] = intersections(X1,T1,X2,T2);
num = length(ix);
if num == 0
   side = '';
else
   % Remove the first point of the intersections
   if iy(1) == T1(1)
      iy = iy(2:end);%keyboard
      ix = ix(2:end);
   end
   num = length(ix);
   for i = 1:num
      if iy(i) > itime
         side{i} = 'D';
      elseif iy(i) < itime
         side{i} = 'I';
      else
         side{i} = 'C';
      end
   end
   if and(num > 2, sum(ix-mean(ix)) == 0)
      num = 1;
      side = side(num);
   end
end
```

```
function [intersection,xi,yi] = INTERS_exist(XS,TS,XL,TL,baymargin)

global BL EXEC

intersection = 0; xi = 0; yi = 0;

XSi = XS + baymargin-0.0001;
XLi = XL; % - S.baymargin; %1.4999;
if or(length(XSi)==1, length(XLi)==1)
    disp('No intersection to analyze. One line is just a point'); return
else
   [xi,yi,iout,jout] = intersections(XSi,TS,XLi,TL);

   if isempty(xi) == 1
      %keyboard
      if or(length(XS)==1, length(XL)==1)
         disp('Hey, wrong intersection'); keyboard
      end
      [xi,yi,iout,jout] = intersections(XS,TS,XL,TL);
      if isempty(xi) == 0
         figure;    subplot(2,1,1);
         plot(TS,XSi,TL,XLi); title('Sea crane + margin trajectories')
         axis([min(TS(1), TL(1)) min(TS(end),TL(end)) 0 BL.bays ])
                  subplot(2,1,2);
         plot(TS,XS,TL,XL); title('Real trajectories')
         axis([min(TS(1), TL(1)) min(TS(end),TL(end)) 0 BL.bays ])

         intersection = 1; keyboard
         xi = (XS(1)+XL(1))/2; yi = TS(1);
      end
   else
      intersection = 1;
   end

   if intersection == 1
      if EXEC.plot == 1
         plot_intersection(XS,TS,XL,TL,xi,yi);
      end
   end

   if length(xi) > 1
      [yi,pos] = min(yi); xi = xi(pos);
   end

end
```

```
function [escenario,matrizcaso,orden] = INTERS_fict_escenario(itype,Sside,Lside,nSi,nLi,Sintslope,Lintslope)
% This function calculates approaching scenarios.

global ASC INTERS

escenario = 0; matrizcaso = 0; orden = 1; timecalc = 0;

% Approach type intersection
[land_bay] = ASC_target_bay(ASC.land);
[sea_bay] = ASC_target_bay(ASC.sea);
if nSi > 2
   nSi = 0;
end
if strcmp(itype,'Fict') == 1
   if nSi < nLi
      matrizcaso = 21;
      if nLi == 2
         % Step(1196,1200,1208)
         escenario = 6; keyboard % better  escenario = 5;
         disp('Warning with escenario 6 sometimes bad calculation');
      else % ASC land stack
         %keyboard
         if strcmp(Lside,'I') == 1
            escenario = 3;
         elseif strcmp(Lside,'D') == 1
            escenario = 4;
         else
            %keyboard
            timecalc= 1;
         end
         %timecalc = 1;
%           if Lslope < 0
%              escenario = 4;%keyboard  % step(827,1931), ok
%           else % ASC land deliverying: ok. Lslope = 0, Sslope > 0
%              %keyboard % Step(X,1228,1234)
%              escenario = 3;
%           end
      end
   elseif nSi > nLi
      matrizcaso = 22;
      %keyboard
      if nSi == 2
         %keyboard % make it depend on the slope?
         if Sintslope > 0
            escenario = 3;
         else
            escenario = 4;
         end
%           elseif nLi == 1
%              escenario = 4
%           end

%          if strcmp(ASC.land.status,'stack')==1
```

```
%              escenario = 7
%           else
%              escenario = 3
%           end
      else
         %keyboard
         if strcmp(Sside,'I') == 1
            escenario = 4; % Step(393) Give priority to sea, ok
         elseif  strcmp(Sside,'D') == 1
            escenario = 3;
         else
            keyboard % Step 3866 y anterior dan fallo
            timecalc = 1; % Step 1030 1551, 4183 Stack-Stack ok
         end
      end
   else % In this case, look which crane is gantrying

      Stask = ASC.sea.ciclo.c_task;  Smove = ASC.sea.ciclo.moves(Stask);
      Ltask = ASC.land.ciclo.c_task; Lmove = ASC.land.ciclo.moves(Ltask);
      if and(strcmp(Smove,'gantry') == 1, strcmp(Lmove,'gantry') == 0)
         escenario = 5; matrizcaso = 231;  % This works ok
      elseif and(strcmp(Smove,'gantry') == 1, strcmp(Lmove,'gantry') == 0)
         escenario = 4; matrizcaso = 232;
      elseif strcmp(Smove,'trans') == 1
         escenario = 4; matrizcaso = 234; %keyboard
      elseif strcmp(Lmove,'trans') == 1
         escenario = 3; matrizcaso = 235; %keyboard
      else
         %keyboard
         matrizcaso = 233;
         timecalc = 1; % Works during ASC.sea delivery
%           if Sslope > 0
%              escenario = 3;
%           elseif Sslope <= 0
%              escenario = 4;
%           else
%              timecalc = 1; %keyboard % Step(501, 693, 1459, 1576) caso chungo
%           end
      end
      %timecalc = 1;
   end
end

% if matrizcaso == 0
%    keyboard
% end

if timecalc == 1;
   % See if one of the cranes has commenced and the other not
   if and(ASC.sea.ciclo.no > 0, ASC.land.ciclo.no == 0)
      escenario = 4;
   elseif and(ASC.sea.ciclo.no == 0, ASC.land.ciclo.no > 0)
      %escenario = 3;
      escenario = 6;
```

```
    else
        if ASC.sea.nextevent <= ASC.land.nextevent
            escenario = 4;
        else
            escenario = 3;
        end
    end
end

if escenario == 3
    orden = 'S';
elseif escenario == 4
    orden = 'L';
elseif escenario == 5
    orden = 'S';
elseif escenario == 6
    orden = 'L';
elseif escenario == 7
    orden = 'S';
end
disp(['\\\\\\\\\ Fict Intersection case ' num2str(matrizcaso) ' \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\'])

INTERS.fict.no = INTERS.fict.no+1;
INTERS.fict.tipo(INTERS.fict.no) = matrizcaso;
INTERS.fict.escenario(INTERS.fict.no) = escenario;
```

```
function INTERS_init()

global INTERS

INTERS.real.no = 0;
INTERS.real.tipo = 0;
INTERS.real.escenario = 0;
INTERS.fict.no = 0;
INTERS.fict.tipo = 0;
INTERS.fict.escenario = 0;
```

```
function [escenario,matrizcaso,orden]= INTERS_type(Sintslope,Lintslope,itype,XL,XS,TL,TS,xRi0,yRi0,xFi0,yFi0)

global ASC

if strcmp(ASC.sea.status , 'wait') == 1
   escenario = 1; matrizcaso = 11; orden = 'L';
elseif strcmp(ASC.land.status , 'wait') == 1
   escenario = 2; matrizcaso = 12; orden = 'S';
else
   if strcmp(itype,'Normal')==1
      % Real intersection
      [nLi,Lside] = INTERS_crosses(XL,TL,xRi0,yRi0);
      [nSi,Sside] = INTERS_crosses(XS,TS,xRi0,yRi0);
      [escenario,matrizcaso,orden] = INTERSECTION_escenario(Sintslope,Lintslope,XL,XS,nSi,nLi);
   elseif strcmp(itype,'Fict')==1
      % Intersection due to excessive approximation
      [nLi,Lside] = INTERS_crosses(XL,TL,xFi0,yFi0);
      [nSi,Sside] = INTERS_crosses(XS,TS,xFi0,yFi0); keyboard
      [escenario,matrizcaso,orden] = INTERS_fict_escenario(itype,Sside,Lside,nSi,nLi,Sintslope,Lintslope);
   else
      keyboard
   end
end

if escenario == 0
   keyboard
% elseif matrizcaso == 8
%     keyboard
end
```

```
function [escenario,matrizcaso,orden]= INTERS_typev2(XL,XS,TL,TS,xi,yi)

global ASC

if strcmp(ASC.sea.status , 'wait') == 1
   escenario = 1; matrizcaso = 11; orden = 'L';
elseif strcmp(ASC.land.status , 'wait') == 1
   escenario = 2; matrizcaso = 12; orden = 'S';
else
   [escenario,matrizcaso,orden] = INTERSECTION_set_escenario(XL,XS,TL,TS,xi,yi);
end
```

```
function [escenario,matrizcaso,orden] = INTERSECTION_escenario(Sslope,Lslope,XL,XS,nSi,nLi)

global ASC S INTERS

% ASC priority
% Escenario 3: land
% Escenario 4: SEA

escenario = 0; matrizcaso = 0; orden = 'S'; timecalc = 0;

% [LGslopes] = ASC_gantryslope(ASC.land);
% [SGslopes] = ASC_gantryslope(ASC.sea);
if and(Sslope < 0, Lslope < 0)  % Matrizcaso 1
   escenario = 6; matrizcaso = 1; % Step 6393 ok
   %keyboard
elseif and(Sslope < 0, Lslope == 0)  % Matrizcaso 2
   escenario = 0; matrizcaso = 2;
   % keyboard
elseif and(Sslope < 0, Lslope > 0)  % Matrizcaso 3
   escenario = 4;   % SEA CRANE PRIORITY
   matrizcaso = 3;
   disp('Esta intersección calcula mal el delay'); %keyboard
elseif and(Sslope == 0, Lslope < 0)  % Matrizcaso 4
   %keyboard
   matrizcaso = 4;
   upbay = max(ASC.sea.ciclo.bay);

   if strcmp(ASC.sea.status,'delivery') == 1
      bdif = XL(1)-upbay;
      if and(bdif>0, bdif < S.baymargin)
         escenario = 3; orden = 'S'; %keyboard
      else
         escenario = 4; orden = 'S';
      end
   else
      timecalc = 1;
   end
elseif and(Sslope == 0, Lslope == 0)  % Matrizcaso 5
   escenario = 0; matrizcaso = 5; % Step 6621 escenario = 3 was used
   keyboard
elseif and(Sslope == 0, Lslope > 0)  % Matrizcaso 6
   escenario = 3; matrizcaso = 6; orden = 'S';
   %keyboard % Step 6611 6642
elseif and(Sslope > 0, Lslope < 0)  % Matrizcaso 7
   matrizcaso = 7;
   % SEA CRANE PRIORITY
   %     if strcmp(ASC.sea.status,'wait') == 0
   %         escenario = 4;
   %     else
   if strcmp(ASC.land.status,'delivery') == 1
      %keyboard %  works. several  times
      timecalc = 1;
      %         lowbay = min(ASC.land.ciclo.bay);
```

```
    %        if lowbay - XS(1) <= S.baymargin
    %            %keyboard
    %            escenario = 4; orden = 'S';
    %        else
    %            escenario = 3; orden = 'L';
    %        end
  else
    %keyboard
    timecalc = 1;
  end
elseif and(Sslope > 0, Lslope == 0)  % Matrizcaso 8
  matrizcaso = 8;
  %     if strcmp(ASC.sea.status,'wait') == 0
  %         escenario = 4;
  %     else
  if strcmp(ASC.land.status,'delivery') == 1
    escenario = 3;
    orden = 'L'; keyboard
  else % COMPLETE
    %keyboard
    if nLi < nSi
      % Step (40 ok)
      escenario = 3;%keyboard
      orden = 'S';
    else
      timecalc = 1;
      % escenario = 4; %  Step()
    end
  end
  %keyboard
elseif and(Sslope > 0, Lslope > 0)  % Matrizcaso 9
  escenario = 5; matrizcaso = 9;
end

% if matrizcaso == 0
%     keyboard
% end

if timecalc == 1;
  % See if one of the cranes has commenced and the other not
  if and(ASC.sea.ciclo.no > 0, ASC.land.ciclo.no == 0)
    escenario = 4;
  elseif and(ASC.sea.ciclo.no == 0, ASC.land.ciclo.no > 0)
    escenario = 3;
  else
    if ASC.sea.nextevent <= ASC.land.nextevent
      escenario = 4;
    else
      escenario = 3;
    end
  end
end

disp(['\\\\\\\\\ Real Intersection case ' num2str(matrizcaso) ' \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\'])
```

```
INTERS.real.no = INTERS.real.no+1;
INTERS.real.tipo(INTERS.real.no) = matrizcaso;
INTERS.real.escenario(INTERS.real.no) = escenario;
```

```
function [escenario,orden] = INTERSECTION_set_escenario(XL,XS,TL,TS,xi,yi)

global ASC INTERS TIME

% ASC priority
% Escenario 3 y 5: LAND
% Escenario 4 y 6: SEA

escenario = 0; matrizcaso = 0; orden = 'S'; timecalc = 0;

seagantry = 0; landgantry = 0; t_int = yi-TIME.t;
gantrypositions = find(strcmp(ASC.land.ciclo.moves,'gantry'));
t_passed = sum(ASC.land.ciclo.originaltime) - sum(ASC.land.ciclo.time);
v_time_m = AUX_vect_ac(ASC.land.ciclo.originaltime) - t_passed;
lp = find(v_time_m > t_int); gp = find(gantrypositions == lp(1));
if isempty(gp) == 0
   landgantry = 1;
end
gantrypositions = find(strcmp(ASC.sea.ciclo.moves,'gantry'));
t_passed = sum(ASC.sea.ciclo.originaltime) - sum(ASC.sea.ciclo.time);
v_time_m = AUX_vect_ac(ASC.sea.ciclo.originaltime) - t_passed;
sp = find(v_time_m > t_int); gp = find(gantrypositions == sp(1));
if isempty(gp) == 0
   seagantry = 1;
end

if and(seagantry == 0, landgantry == 0)
   disp('Warning case 0 intersection');%keyboard
elseif and(seagantry == 1, landgantry == 0)
   escenario = 3; orden = 'S'; %keyboard
elseif and(seagantry == 0, landgantry == 1)
   escenario = 4; orden = 'L'; %keyboard
elseif and(seagantry == 1, landgantry == 1)

   [Lslope]= ASC_way(xi,yi,XL,TL);
   [Sslope]= ASC_way(xi,yi,XS,TS);
   [nLi,Lside] = INTERS_crosses(XL,TL,xi,yi);
   [nSi,Sside] = INTERS_crosses(XS,TS,xi,yi);

   if and(Sslope < 0, Lslope < 0)  % Matrizcaso 1
      %keyboard
      matrizcaso = 1; escenario = 4;
   elseif and(Sslope < 0, Lslope == 0)  % Matrizcaso 2
      keyboard
      %escenario = 0; matrizcaso = 2;
   elseif and(Sslope < 0, Lslope > 0)  % Matrizcaso 3
      keyboard % step 73399 essa1000 trf 60 seed(1) wrong what if change to 3
      matrizcaso = 3; escenario = 4;  % SEA CRANE PRIORITY
      disp('Esta intersección calcula mal el delay'); %keyboard
   elseif and(Sslope == 0, Lslope < 0)  % Matrizcaso 4
      %keyboard
      matrizcaso = 4;  escenario = 4;
   elseif and(Sslope == 0, Lslope == 0)  % Matrizcaso 5
```

```
        keyboard
        matrizcaso = 3;  escenario = 0; % Step 6621 escenario = 3 was used
    elseif and(Sslope == 0, Lslope > 0)  % Matrizcaso 6
        matrizcaso = 4; keyboard
        % step 6395 better the timecalc = 1;
    elseif and(Sslope > 0, Lslope < 0)  % Matrizcaso 7
        matrizcaso = 7; %keyboard
        timecalc = 1;
    elseif and(Sslope > 0, Lslope == 0)  % Matrizcaso 8
        matrizcaso = 8; keyboard

        if strcmp(ASC.land.status,'delivery') == 1
            escenario = 3;
            orden = 'L'; keyboard
        elseif strcmp(ASC.land.status,'housekeeping') == 1
            escenario = 3;
            orden = 'L'; keyboard
        else % COMPLETE
            if nLi < nSi
                escenario = 3;%keyboard
                orden = 'S';
            else
                timecalc = 1;
            end
        end
    elseif and(Sslope > 0, Lslope > 0)  % Matrizcaso 9
%         if ASC.land.ciclo.originalct == 2740
%             keyboard
%         end
        matrizcaso = 9; escenario = 3;
    else
        keyboard
    end
end

if timecalc == 1;
    % See if one of the cranes has commenced and the other not
    if and(ASC.sea.ciclo.no == 0, ASC.land.ciclo.no > 0)
        escenario = 3;% Escenario 3: SEA priority
    elseif and(ASC.sea.ciclo.no > 0, ASC.land.ciclo.no == 0)
        escenario = 4;% Escenario 4: LAND priority
    else
        if ASC.sea.nextevent <= ASC.land.nextevent
            escenario = 4;
        else
            escenario = 3;
        end
    end
end

disp(['\\\\\\\\\\\\\\\ Crane intersection: Escenario ' num2str(escenario) ' Matriz caso: ' num2str(matrizcaso) '\\\\\\\\\\\\\\\\\\\'])

INTERS.real.no = INTERS.real.no+1;
INTERS.real.tipo(INTERS.real.no) = matrizcaso;
```

INTERS.real.escenario(INTERS.real.no) = escenario;

```
function [xRi0,yRi0,xFi0,yFi0,Ri,Fi,intersectiontype,Lintslope,Sintslope] =
INTERSECTION_yesno_init(xRi,yRi,xFi,yFi,siono,XSo,TSo,XLo,TLo,XSi,XLi)

% This function filters the points given as intersection points

global ASC TIME

Ri = 0; Fi = 0; xRi0 = []; yRi0 = []; xFi0 = []; yFi0 = [];

% Type of intersection
if or(xRi > 0, length(xRi) > 0)
   Ri = 1;
   [yRi0,yp] = min(yRi); xRi0 = xRi(yp);
   if yRi0 == TIME.t
      disp('Real Intersection at cycle start')
   end
end

if or(xFi>0, length(xFi) > 0)
   Fi = 1;
   [yFi0,yp] = min(yFi); xFi0 = xFi(yp);
   if yFi0 == TIME.t
      disp('Fict Intersection at cycle start')
   end
end

if and(Ri>0, Fi>0) % There is real and fict
   %keyboard
   [iSR] = ASC_get_gantry_task(ASC.sea,yRi0);
   [iSF] = ASC_get_gantry_task(ASC.sea,yFi0);
   [iLR] = ASC_get_gantry_task(ASC.land,yRi0);
   [iLF] = ASC_get_gantry_task(ASC.land,yFi0);
   if and(iSR == iSF, iLR == iLF) % Real and fict inters are at the same gantry
      %intersectiontype = 'Normal'; % normal
      if yRi0 < yFi0
         intersectiontype = 'Normal'; % normal
      elseif yFi0 < yRi0
         intersectiontype = 'Fict'; % ficticiuos
      end
   else
      keyboard
   end
elseif and(Ri==0, Fi>0)
   intersectiontype = 'Fict'; % ficticiuos
elseif and(Ri>0, Fi == 0)
   %keyboard
   intersectiontype = 'Normal'; % Real intersection case
   %xFi0 = xRi0; yFi0=yRi0;
else
   intersectiontype = 'N'; % weird case
end

% Plot intersections
```

```
if siono == 1
   figure(99);
   subplot(2,1,1); plot(TSo,XSi,'.-g',TLo,XLi,'.-b');
   subplot(2,1,2); plot(TSo,XSo,'.-g',TLo,XLo,'.-b');
end

if strcmp(intersectiontype, 'N') == 0
   if strcmp(intersectiontype, 'Normal') == 1
      xi = xRi0; yi = yRi0;
   elseif strcmp(intersectiontype, 'Fict') == 1
      xi = xFi0; yi = yFi0;
   end
   [Lintslope]= ASC_way(xi,yi,XLi,TLo); %(XLo(2)-XLo(1))/(TLo(2)-TLo(1));
   [Sintslope]= ASC_way(xi,yi,XSi,TSo); %(XSo(2)-XSo(1))/(TSo(2)-TSo(1));
   plot_intersection(XSo,TSo,XLo,TLo,Ri,xRi0,yRi0,Fi,xFi0,yFi0);
else
   [Lintslope]= 0;%keyboard
   [Sintslope]= 0;
end
```

```
function [xi,yi,intersectiontype,Lintslope,Sintslope] = INTERSECTION_yesno_initv2(xi,yi,siono,XS,TS,XL,TL)

% This function filters the points given as intersection points

global ASC TIME

% Type of intersection

[iS] = ASC_get_gantry_task(ASC.sea,yi);
[iL] = ASC_get_gantry_task(ASC.land,yi);

seagantry = 0; landgantry = 0;
if iS == ASC.sea.ciclo.c_task
   seagantry = 1;
end
if iL == ASC.land.ciclo.c_task
   landgantry = 1;
end
keyboard

if strcmp(intersectiontype, 'N') == 0
   if strcmp(intersectiontype, 'Normal') == 1
      xi = xRi0; yi = yRi0;
   elseif strcmp(intersectiontype, 'Fict') == 1
      xi = xFi0; yi = yFi0;
   end
   [Lintslope]= ASC_way(xi,yi,XLi,TLo); %(XLo(2)-XLo(1))/(TLo(2)-TLo(1));
   [Sintslope]= ASC_way(xi,yi,XSi,TSo); %(XSo(2)-XSo(1))/(TSo(2)-TSo(1));

else
   [Lintslope]= 0;%keyboard
   [Sintslope]= 0;
end
```

```
function [xRi0,yRi0,xFi0,yFi0,Ri,Fi,intersectiontype] =
INTERSECTION_filter_pointsv5(xRi,yRi,xFi,yFi,siono,XSo,TSo,XLo,TLo,XSi,XLi)

% This function filters the points given as intersection points

global TIME

Ri = 0; Fi = 0; xRi0 = []; yRi0 = []; xFi0 = []; yFi0 = [];

% Type of intersection
if or(xRi > 0, length(xRi) > 0)
   Ri = 1;
   [yRi0,yp] = min(yRi); xRi0 = xRi(yp);
   if yRi0 == TIME.t
      disp('Real Intersection at cycle start')
   end
end

if or(xFi>0, length(xFi) > 0)
   Fi = 1;
   [yFi0,yp] = min(yFi); xFi0 = xFi(yp);
   if yFi0 == TIME.t
      disp('Fict Intersection at cycle start')
   end
end

if and(Ri>0, Fi>0)
   intersectiontype = 'Normal'; % normal
elseif and(Ri==0, Fi>0)
   intersectiontype = 'Fict'; % ficticiuos
elseif and(Ri>0, Fi == 0)
   intersectiontype = 'Weird'; % weird case
   xFi0 = xRi0; yFi0=yRi0;
else
   intersectiontype = 'W'; % weird case
end

% Plot intersections
if siono == 1
   figure(99);
   subplot(2,1,1); plot(TSo,XSi,'.-g',TLo,XLi,'.-b');
   subplot(2,1,2); plot(TSo,XSo,'.-g',TLo,XLo,'.-b');
end
```

```
function [x0,y0,iout,jout] = intersections(x1,y1,x2,y2,robust)
%INTERSECTIONS Intersections of curves.
%   Computes the (x,y) locations where two curves intersect.  The curves
%   can be broken with NaNs or have vertical segments.
%
% Example:
%   [X0,Y0] = intersections(X1,Y1,X2,Y2,ROBUST);
%
% where X1 and Y1 are equal-length vectors of at least two points and
% represent curve 1.  Similarly, X2 and Y2 represent curve 2.
% X0 and Y0 are column vectors containing the points at which the two
% curves intersect.
%
% ROBUST (optional) set to 1 or true means to use a slight variation of the
% algorithm that might return duplicates of some intersection points, and
% then remove those duplicates.  The default is true, but since the
% algorithm is slightly slower you can set it to false if you know that
% your curves don't intersect at any segment boundaries.  Also, the robust
% version properly handles parallel and overlapping segments.
%
% The algorithm can return two additional vectors that indicate which
% segment pairs contain intersections and where they are:
%
%   [X0,Y0,I,J] = intersections(X1,Y1,X2,Y2,ROBUST);
%
% For each element of the vector I, I(k) = (segment number of (X1,Y1)) +
% (how far along this segment the intersection is).  For example, if I(k) =
% 45.25 then the intersection lies a quarter of the way between the line
% segment connecting (X1(45),Y1(45)) and (X1(46),Y1(46)).  Similarly for
% the vector J and the segments in (X2,Y2).
%
% You can also get intersections of a curve with itself.  Simply pass in
% only one curve, i.e.,
%
%   [X0,Y0] = intersections(X1,Y1,ROBUST);
%
% where, as before, ROBUST is optional.

% Version: 1.12, 27 January 2010
% Author:  Douglas M. Schwarz
% Email:   dmschwarz=ieee*org, dmschwarz=urgrad*rochester*edu
% Real_email = regexprep(Email,{'=','*'},{'@','.'})


% Theory of operation:
%
% Given two line segments, L1 and L2,
%
%   L1 endpoints:  (x1(1),y1(1)) and (x1(2),y1(2))
%   L2 endpoints:  (x2(1),y2(1)) and (x2(2),y2(2))
%
% we can write four equations with four unknowns and then solve them.  The
% four unknowns are t1, t2, x0 and y0, where (x0,y0) is the intersection of
```

```
% L1 and L2, t1 is the distance from the starting point of L1 to the
% intersection relative to the length of L1 and t2 is the distance from the
% starting point of L2 to the intersection relative to the length of L2.
%
% So, the four equations are
%
%    (x1(2) - x1(1))*t1 = x0 - x1(1)
%    (x2(2) - x2(1))*t2 = x0 - x2(1)
%    (y1(2) - y1(1))*t1 = y0 - y1(1)
%    (y2(2) - y2(1))*t2 = y0 - y2(1)
%
% Rearranging and writing in matrix form,
%
%  [x1(2)-x1(1)       0       -1   0;      [t1;      [-x1(1);
%       0       x2(2)-x2(1)  -1   0;   *   t2;   =   -x2(1);
%   y1(2)-y1(1)       0        0  -1;       x0;       -y1(1);
%       0       y2(2)-y2(1)   0  -1]       y0]       -y2(1)]
%
% Let's call that A*T = B.  We can solve for T with T = A\B.
%
% Once we have our solution we just have to look at t1 and t2 to determine
% whether L1 and L2 intersect.  If 0 <= t1 < 1 and 0 <= t2 < 1 then the two
% line segments cross and we can include (x0,y0) in the output.
%
% In principle, we have to perform this computation on every pair of line
% segments in the input data.  This can be quite a large number of pairs so
% we will reduce it by doing a simple preliminary check to eliminate line
% segment pairs that could not possibly cross.  The check is to look at the
% smallest enclosing rectangles (with sides parallel to the axes) for each
% line segment pair and see if they overlap.  If they do then we have to
% compute t1 and t2 (via the A\B computation) to see if the line segments
% cross, but if they don't then the line segments cannot cross.  In a
% typical application, this technique will eliminate most of the potential
% line segment pairs.


% Input checks.
error(nargchk(2,5,nargin))

% Adjustments when fewer than five arguments are supplied.
switch nargin
    case 2
        robust = true;
        x2 = x1;
        y2 = y1;
        self_intersect = true;
    case 3
        robust = x2;
        x2 = x1;
        y2 = y1;
        self_intersect = true;
    case 4
        robust = true;
        self_intersect = false;
```

```
    case 5
        self_intersect = false;
end

% x1 and y1 must be vectors with same number of points (at least 2).
if sum(size(x1) > 1) ~= 1 || sum(size(y1) > 1) ~= 1 || ...
        length(x1) ~= length(y1)
    error('X1 and Y1 must be equal-length vectors of at least 2 points.')
end
% x2 and y2 must be vectors with same number of points (at least 2).
if sum(size(x2) > 1) ~= 1 || sum(size(y2) > 1) ~= 1 || ...
        length(x2) ~= length(y2)
    error('X2 and Y2 must be equal-length vectors of at least 2 points.')
end


% Force all inputs to be column vectors.
x1 = x1(:);
y1 = y1(:);
x2 = x2(:);
y2 = y2(:);

% Compute number of line segments in each curve and some differences we'll
% need later.
n1 = length(x1) - 1;
n2 = length(x2) - 1;
xy1 = [x1 y1];
xy2 = [x2 y2];
dxy1 = diff(xy1);
dxy2 = diff(xy2);

% Determine the combinations of i and j where the rectangle enclosing the
% i'th line segment of curve 1 overlaps with the rectangle enclosing the
% j'th line segment of curve 2.
[i,j] = find(repmat(min(x1(1:end-1),x1(2:end)),1,n2) <= ...
    repmat(max(x2(1:end-1),x2(2:end)).',n1,1) & ...
    repmat(max(x1(1:end-1),x1(2:end)),1,n2) >= ...
    repmat(min(x2(1:end-1),x2(2:end)).',n1,1) & ...
    repmat(min(y1(1:end-1),y1(2:end)),1,n2) <= ...
    repmat(max(y2(1:end-1),y2(2:end)).',n1,1) & ...
    repmat(max(y1(1:end-1),y1(2:end)),1,n2) >= ...
    repmat(min(y2(1:end-1),y2(2:end)).',n1,1));

% Force i and j to be column vectors, even when their length is zero, i.e.,
% we want them to be 0-by-1 instead of 0-by-0.
i = reshape(i,[],1);
j = reshape(j,[],1);

% Find segments pairs which have at least one vertex = NaN and remove them.
% This line is a fast way of finding such segment pairs.  We take
% advantage of the fact that NaNs propagate through calculations, in
% particular subtraction (in the calculation of dxy1 and dxy2, which we
% need anyway) and addition.
% At the same time we can remove redundant combinations of i and j in the
```

```
% case of finding intersections of a line with itself.
if self_intersect
    remove = isnan(sum(dxy1(i,:) + dxy2(j,:),2)) | j <= i + 1;
else
    remove = isnan(sum(dxy1(i,:) + dxy2(j,:),2));
end
i(remove) = [];
j(remove) = [];

% Initialize matrices.  We'll put the T's and B's in matrices and use them
% one column at a time.  AA is a 3-D extension of A where we'll use one
% plane at a time.
n = length(i);
T = zeros(4,n);
AA = zeros(4,4,n);
AA([1 2],3,:) = -1;
AA([3 4],4,:) = -1;
AA([1 3],1,:) = dxy1(i,:).';
AA([2 4],2,:) = dxy2(j,:).';
B = -[x1(i) x2(j) y1(i) y2(j)].';

% Loop through possibilities.  Trap singularity warning and then use
% lastwarn to see if that plane of AA is near singular.  Process any such
% segment pairs to determine if they are colinear (overlap) or merely
% parallel.  That test consists of checking to see if one of the endpoints
% of the curve 2 segment lies on the curve 1 segment.  This is done by
% checking the cross product
%
%   (x1(2),y1(2)) - (x1(1),y1(1)) x (x2(2),y2(2)) - (x1(1),y1(1)).
%
% If this is close to zero then the segments overlap.

% If the robust option is false then we assume no two segment pairs are
% parallel and just go ahead and do the computation.  If A is ever singular
% a warning will appear.  This is faster and obviously you should use it
% only when you know you will never have overlapping or parallel segment
% pairs.

if robust
    overlap = false(n,1);
    warning_state = warning('off','MATLAB:singularMatrix');
    % Use try-catch to guarantee original warning state is restored.
    try
        lastwarn('')
        for k = 1:n
            T(:,k) = AA(:,:,k)\B(:,k);
            [unused,last_warn] = lastwarn;
            lastwarn('')
            if strcmp(last_warn,'MATLAB:singularMatrix')
                % Force in_range(k) to be false.
                T(1,k) = NaN;
                % Determine if these segments overlap or are just parallel.
                overlap(k) = rcond([dxy1(i(k),:);xy2(j(k),:) - xy1(i(k),:)]) < eps;
            end
```

```
            end
            warning(warning_state)
        catch err
            warning(warning_state)
            rethrow(err)
        end
        % Find where t1 and t2 are between 0 and 1 and return the corresponding
        % x0 and y0 values.
        in_range = (T(1,:) >= 0 & T(2,:) >= 0 & T(1,:) <= 1 & T(2,:) <= 1).';
        % For overlapping segment pairs the algorithm will return an
        % intersection point that is at the center of the overlapping region.
        if any(overlap)
            ia = i(overlap);
            ja = j(overlap);
            % set x0 and y0 to middle of overlapping region.
            T(3,overlap) = (max(min(x1(ia),x1(ia+1)),min(x2(ja),x2(ja+1))) + ...
                min(max(x1(ia),x1(ia+1)),max(x2(ja),x2(ja+1)))).'/2;
            T(4,overlap) = (max(min(y1(ia),y1(ia+1)),min(y2(ja),y2(ja+1))) + ...
                min(max(y1(ia),y1(ia+1)),max(y2(ja),y2(ja+1)))).'/2;
            selected = in_range | overlap;
        else
            selected = in_range;
    end
    T;
    selected;
        xy0 = T(3:4,selected).';

        % Remove duplicate intersection points.
        [xy0,index] = unique(xy0,'rows');
        x0 = xy0(:,1);
        y0 = xy0(:,2);

        % Compute how far along each line segment the intersections are.
        if nargout > 2
            sel_index = find(selected);
            sel = sel_index(index);
            iout = i(sel) + T(1,sel).';
            jout = j(sel) + T(2,sel).';
        end
else % non-robust option
        for k = 1:n
            [L,U] = lu(AA(:,:,k));
            T(:,k) = U\(L\B(:,k));
        end

        % Find where t1 and t2 are between 0 and 1 and return the corresponding
        % x0 and y0 values.
        in_range = (T(1,:) >= 0 & T(2,:) >= 0 & T(1,:) < 1 & T(2,:) < 1).';
        x0 = T(3,in_range).';
        y0 = T(4,in_range).';

        % Compute how far along each line segment the intersections are.
        if nargout > 2
            iout = i(in_range) + T(1,in_range).';
```

```
            jout = j(in_range) + T(2,in_range).';
        end
end

% Plot the results (useful for debugging).
% plot(x1,y1,x2,y2,x0,y0,'ok');
```

```
function LIMITS_init()

global LIMITS

LIMITS.land.bay = -10;
LIMITS.land.ct = 0;

LIMITS.sea.bay = 50;
LIMITS.sea.ct = 0;
```

```
% This file executes ASC main a number of times
clear

%matlabpool open 4
trf_levels = [40 60]; % 40 or 60

batch = 1;
endcase = 7;
for trfi = 1: length(trf_levels)
   if trf_levels(trfi)== 40
      casos = [1 2 3 4 5 6 7];
   elseif trf_levels(trfi)== 60
      casos = [2 3 4 5 6 7];
   end

   for c = 1:length(casos)
      opt = casos(c);
   %for opt = 3:3
      %clearvars -except i casos batch
      close all
      clc
      if opt == 1
         stacks = 5;
         bays = 72;
      elseif opt == 2
         stacks = 6;
         bays = 60;
      elseif opt == 3
         stacks = 8;
         bays = 45;
      elseif opt == 4
         stacks = 9;
         bays = 40;
      elseif opt == 5
         stacks = 10;
         bays = 36;
      elseif opt == 6
         stacks = 12;
         bays = 30;
      elseif opt == 7
         stacks = 15;
         bays = 24;
      end

      ASC_main(stacks,bays,opt,batch,trf_levels(trfi),-1); %hotstart
      ASC_main(stacks,bays,opt,batch,trf_levels(trfi),1); %hotstart
   end
end
```

```
% This file executes ASC main a number of times
clear

%matlabpool open 4
trflevels = [40 60 ]; % 40 or 60

%batch = 1;


%ASC_main(trflevel(trfi),energy_crit,stack_alg,0); %hotstart
% ASC_main4exec(trflevels(1),1,'psrandom',-1,1);
% ASC_main4exec(trflevels(2),1,'psrandom',-1,1);

for trfi = 1: length(trflevels)
%     ASC_main4exec(trflevels(trfi),1,'crandom',1,0); %hotstart
%     ASC_main4exec(trflevels(trfi),1,'psrandom',1,0); %hotstart
%    ASC_main4exec(trflevels(trfi),1,'tercat',1,0); %hotstart
     ASC_main4exec(trflevels(trfi),1,'essa',1,0); %hotstart
    ASC_main4exec(trflevels(trfi),0.5,'essa',1,0); %hotstart
    ASC_main4exec(trflevels(trfi),0,'essa',1,0); %hotstart
end
```

```
function makebays()

global BAYS BL
s = 0;
for bay = 1:BL.bays
   for stack = 1:BL.stacks
      s = s+1;
      BAYS(bay).GS(stack)=s;
   end
end
```

```
function old_asc_results

an_asc = 1;
if an_asc ==1

    ld = 0; ls = 0; lt = 0; % Land delivery and stack
    L = length(ASC.land.tasks.executed);
    for l = 1: L
        cl_time =  ASC.land.tasks.time(l);
        if and(cl_time > t_ini, cl_time < t_fin)
            %ascnr = ascnr + ASC.land.
            if ASC.land.tasks.ct(l) > 0
                % delivery
                move = ASC.land.tasks.action(l);
                if strcmp(move,'delivery') == 1
                    ld = ld +1;
                    ldtime(ld) = ASC.land.tasks.time(l);
                    ld_duration(ld) = ASC.land.tasks.duration(l);
                    av_ld(ld) = mean(ld_duration);
                % stack
                elseif strcmp(move,'stack') == 1
                    ls = ls +1;
                    lstime(ls) = ASC.land.tasks.time(l);
                    ls_duration(ls) = ASC.land.tasks.duration(l);
                    av_ls(ls) = mean(ls_duration);
                elseif strcmp(move,'trans') == 1
                    lt = lt +1;
                    keyboard
                end
            else
                lt = lt+1;% keyboard
                lt_duration(lt) = ASC.land.tasks.duration(l);
            end
        end
    end

    ss = 0; sd = 0; st = 0;
    S = length(ASC.sea.tasks.executed);
    for s = 1: S
        cs_time = ASC.sea.tasks.time(s);
        if and(cs_time > t_ini, cs_time < t_fin)
            if ASC.sea.tasks.ct(s) > 0
                move = ASC.sea.tasks.action(s);
                if strcmp(move,'delivery') == 1
                    sd = sd +1; sdtime(sd) = ASC.sea.tasks.time(s);
                    sd_duration(sd) = ASC.sea.tasks.duration(s);
                    av_sd(sd) = mean(sd_duration);
                % stack
                elseif strcmp(move,'stack') == 1
                    ss = ss +1; sstime(ss) = ASC.sea.tasks.time(s);
                    ss_duration(ss) = ASC.sea.tasks.duration(s);
                    av_ss(ss) = mean(ss_duration);
                elseif strcmp(move,'trans') == 1
```

```
            st = st + 1;
            keyboard
          end
        else
          st = st+1;
          st_duration(st) = ASC.sea.tasks.duration(s);
        end
      end
    end
end

SSdur = mean(ss_duration); SDdur = mean(sd_duration); STdur = mean(st_duration);
LSdur = mean(ls_duration); LDdur = mean(ld_duration); LTdur = mean(lt_duration);
disp(' ')
disp(['Land crane stack cycles: ' num2str(ls) ' Av duration: ' num2str(LSdur)])
disp(['Land crane delivery cycles: ' num2str(ld) ' Av duration: ' num2str(LDdur)])
disp(['Land crane translations: ' num2str(lt) ' Av duration: ' num2str(LTdur)])
disp(['Sea  crane stack cycles: ' num2str(ss) ' Av duration: ' num2str(SSdur)])
disp(['Sea  crane delivery cycles: ' num2str(sd) ' Av duration: ' num2str(SDdur)])
disp(['Sea  crane translations: ' num2str(st) ' Av duration: ' num2str(STdur)])
disp(' ')

figure;
subplot(2,1,1);
plot(sstime/3600/24, av_ss,'r');hold on;title('Average cycle durations vs. time')
plot(sdtime/3600/24, av_sd,'b'); axis([0 TIME.simulation.days 0 400])
subplot(2,1,2);
plot(lstime/3600/24, av_ls,'r');hold on
plot(ldtime/3600/24, av_ld,'b'); axis([0 TIME.simulation.days 0 400])
```

```
function [ndf,cdf,top_xf] = pdfcdf(no_wc)
% This function calculates both the PDF and CDF of a
calc = 1;
if calc == 1
   % 1 Container weight type to be analyzed
   cont_category=40;
   if cont_category==40
      owl=load('tw40.dat');
      minw=4400;
      maxw=32500;
   elseif cont_category==20
      owl=load('tw20.dat');
      minw=2200;
      maxw=30480;
   end

   % 2 length of the list
   ol_w=length(owl);

   % 3 Remove the empty containers, if there are any
   l_w=0;
   for cont=1:ol_w
      if owl(cont)>minw;
         l_w=l_w+1;
         wl(l_w)=owl(cont);
      end
   end

   % 4 Probability density function NDF
   %no_wc=9;
   [xf,top_xf,cdf]=cdf_x2(no_wc,wl,minw,maxw);
   ndelx=(maxw-minw)/no_wc;
   low_xf=top_xf-ndelx;
   ndf(1)=cdf(1);
   for j=2:no_wc
      ndf(j)= cdf(j)-cdf(j-1);
   end
end

save('pdf.mat','ndf')
save('cdf.mat','cdf')
```

```
function [pile] = PILE_copy(side,flow)

global BF

switch char(side)
    case 'sea'
        switch char(flow)
            case 'IMP'
                pile = BF.sea.imp;
            case 'EXP'
                pile = BF.sea.exp;
        end
    case 'land'
        switch char(flow)
            case 'IMP'
                pile = BF.land.imp;
            case 'EXP'
                pile = BF.land.exp;
            case 'DUAL'
                pile = BF.land.exp;
        end
end
```

```
function [pos] = PILE_LOCATION(pile)

global BF
% This function takes a pile and returns the location for a CT

pos.bay = pile.bay;
cols = sum(pile.slots);
% Find the stacks that have such positions
candidate_stacks = find(cols == min(cols));
% Randomly choose a candidate among such
index = random('unid',length(candidate_stacks));
pos.stack = candidate_stacks(index);

[val, pos.tier] = min(pile.slots(:,pos.stack));
if pos.tier >2
   disp('Warning pile occupation too big') %keyboard
end

if pos.tier == BF.tiers+1
   keyboard
end
pos.gs = 0;
```

```
function plot_ASC()

global ASC BL S
```

```
function plot_ASC_hist()

global ASC BL TIME

% mins = TIME.simulation.days * 24*60;
% T1 = (1:mins)*60;
figure_close_ifexists(301); figure_close_ifexists(302);
divisiones = 1000;

uPL = ASC.land.position.bay(2:end-1);
uTL = ASC.land.position.time(2:end-1);
uPS = ASC.sea.position.bay(2:end-1);
uTS = ASC.sea.position.time(2:end-1);

bays= -1:BL.bays+2;

if sum(uTL)>0
   figure(301);
   [TL,PL] = AUX_filtrar(uTL,uPL);
   plot(TL,PL,'.-b');
   delt = (uTL(end)-uTL(1))/divisiones;
   T1 = uTL(1):delt: uTL(end);
   P1L = interp1(TL,PL,T1,'linear');
   figure(302);
   HL = hist(P1L,bays);
   plot(bays,HL,'b')
end

if sum(uTS)>0
   figure(301);hold on;
   [TS,PS] = AUX_filtrar(uTS,uPS);
   plot(TS,PS,'.-g');
   delt = (uTS(end)-uTS(1))/divisiones;
   T2 = uTS(1):delt: uTS(end);
   P1S = interp1(TS,PS,T2,'linear');
   figure(302); hold on
   HS = hist(P1S,bays);
   plot(bays,HS,'g')
end
% YI = interp1(X,Y,XI) interpolates to find YI, the values of the
%    underlying function Y at the points in the array XI. X must be a
%    vector of length N.
```

```
function plot_ASC_tasks(asc)

global PLOT

if sum(asc.tasks.ct) > 0
   f = PLOT.fig + 1;
   PLOT.fig = f;
   figure_close_ifexists(105); figure(105);
   plot(asc.tasks.time/3600/24,asc.tasks.ct); hold on %,)
   plot(asc.tasks.time/3600/24,asc.tasks.ct,'mo','MarkerSize',4); hold on
   if asc.tasks.current>0
      act_ct = asc.tasks.ct(asc.tasks.current);
      plot(asc.tasks.time(asc.tasks.current)/3600/24,act_ct,'r.')
   end
end
%plot(asc.tasks.time,asc.tasks.executed,'g.')
%axis([0 150 0 100000])
```

```
function plot_ASC_trajectories(NP)

global ASC TIME

f= 107; figure_close_ifexists(f); figure(f)

n = length(ASC.land.position.time); NP = min(NP,n);
tl = ASC.land.position.time(n-NP+1:n);
xl = ASC.land.position.bay(n-NP+1:n);
plot(tl/3600/24,xl,'.-b');hold on

n = length(ASC.sea.position.time);NP = min(NP,n);
ts = ASC.sea.position.time(n-NP+1:n);
xs = ASC.sea.position.bay(n-NP+1:n);
plot(ts/3600/24,xs,'.-g')

LP= ASC_act_pos(ASC.land);
SP= ASC_act_pos(ASC.sea);

plot(LP.time/3600/24,LP.bay,'b*')
plot(SP.time/3600/24,SP.bay,'g*'); hold on

% PLOT TIME LINES
y(1) = -2; y(2) = 42;
t(1) = TIME.t/3600/24; t(2)=t(1);
tdelt(1) = (TIME.t+TIME.delt)/3600/24; tdelt(2)= tdelt(1);
plot(t,y,'.-r'); hold on
plot(tdelt,y,'.-r'); hold on

if ASC.land.ciclo.no>0
   LCT(1)=ASC.land.ciclo.basetime(ASC.land.ciclo.no) ; LCT(2)=LCT(1);
   plot(LCT/3600/24,y,'.-b')
end
n = length(ASC.land.position.time); NP = min(NP,n);
xl = ASC.land.position.bay(n-NP+1:n);
tl = ASC.land.position.time(n-NP+1:n)/3600/24;
ctl = ASC.land.position.ct(n-NP+1:n);
plot(tl,xl,'.-b');hold on
for i=1:NP
   text(tl(i),xl(i),num2str(ctl(i)));
end

if ASC.sea.ciclo.no>0
   SCT(1)=ASC.sea.ciclo.basetime(ASC.sea.ciclo.no) ; SCT(2)=SCT(1);
   plot(SCT/3600/24,y,'.-g')

end
n = length(ASC.sea.position.time);NP = min(NP,n);
xs = ASC.sea.position.bay(n-NP+1:n);
ts = ASC.sea.position.time(n-NP+1:n)/3600/24;
cts = ASC.sea.position.ct(n-NP+1:n);
plot(ts,xs,'.-g'); hold on
for i=1:NP
```

```
    text(ts(i),xs(i),num2str(cts(i)));
end


grid on
```

```
function plot_BF(fig,buffer,flow)

global BF S

figure(fig)
x = buffer.bay*S.l;
for stack = 1: BF.stacks
    y = stack*S.w;
    for tier = 1:BF.tiers
        z = tier*(S.h+0.5);
        if buffer.slots(tier,stack)==1;
            if strcmp(flow,'EXP') == 1
                col = 'r<';
            elseif strcmp(flow,'IMP') == 1
                col = 'b>';
            end
        else
            col='g.';
        end
        plot3(x,y,z,col);
    end
end
```

```
function plot_BL()
% This funciton plots the Block

global ASC BF BL S

f = 102; figure_close_ifexists(f); figure(f)


% BLOCK
plot_Block(f)

% CONTAINERS
for p = 1: BL.no_gs
   x = BL.GS(p).bay * S.l;
   y = BL.GS(p).stack * S.w;
   if strcmp(BL.GS(p).id,'EXP') == 1
      col = 'b'; %col = 'b<';
   elseif strcmp(BL.GS(p).id,'IMP') == 1
      col = 'g'; %col = 'g>';
   else
      col = '.r';
   end
   for h = 1:BL.GS(p).ocup
      %plot3(x,y,h*S.h,col); hold on
      plot_ct(x,y,h,col);
   end
end

axis equal

% ASCs
z = ones(1,2)*(BL.tiers+1)*S.h;
zs = zeros(1,2);
y(1) = 0; y(2) = BL.stacks*S.w;

ascpos = ASC_act_pos(ASC.land);
x = ones(1,2)*ascpos.bay*S.l;
plot3(x,y,z,'b', 'LineWidth',5)
plot3(x,y,zs,'k', 'LineWidth',1)
ascpos = ASC_act_pos(ASC.sea);
x = ones(1,2)*ascpos.bay*S.l;
plot3(x,y,z,'g','LineWidth',5)
plot3(x,y,zs,'k', 'LineWidth',1)

% Plot bay index
y = -4; %(BL.stacks+1)*S.h;
z = 0;
for b = 1:BL.bays
   bay = num2str(b);
   x = b*S.l;
   text(x,y,z,bay)
end
```

```
% x = ones(1,BL.bays)
% y = ones(1,BL.bays)*
% z = zeros(1,BL.bays);

axis([ -2*S.l 43*S.l 0 9*S.w 0 BF.tiers*S.h])
view(-20,30);
```

```
function plot_Block(f)

global BF BL S

figure_close_ifexists(f); figure(f)


% GROUND SLOTS
for p = 1: BL.no_gs
    x = BL.GS(p).bay * S.l;
    y = BL.GS(p).stack * S.w;
    plot3(x,y,0,'k.'); hold on
end

% BUFFER
% plot_BF(f,BF.sea.imp,'IMP');
% plot_BF(f,BF.sea.exp,'EXP');
% plot_BF(f,BF.land.imp,'IMP');
% plot_BF(f,BF.land.exp,'EXP');

axis equal
```

```
function plot_caja()

global BL S

nx=40;
ny=6;
nz=5;

delx = nx*S.l;
dely = ny*S.w;
delz = nz*S.h;

bx = 20*S.l;
by = 3*S.w;
bz = 5*S.h;

quiver3(0,S.w,0,0,0,bz,'b','LineWidth',4); hold on
quiver3(0,S.w,bz,0,by-S.w,0,'b','LineWidth',4);
quiver3(0,by,bz,bx,0,0,'b','LineWidth',4);

text(by/3,0,delz/2,'Hoist 3%','FontSize',22);
text(by,dely/2,2*delz/3,'Trolley 1%','FontSize',22);
text(bx/3,0,3*bz,'Gantry 90%','FontSize',22);

axis equal; axis([0 delx 0 dely 0 delz])
grf= gca;
x = (1:nx)*S.l;
y = (1:ny)*S.w;
z = (1:nz)*S.h;
set(grf,'XTick', x);
set(grf,'YTick', y);
set(grf,'ZTick', z);

for i = 1:nx
   xl{i}=num2str(i);
end
for i = 1:ny
   yl{i}=num2str(i);
end
for i = 1:nz
   zl{i}=num2str(i);
end
set(grf,'XTickLabel', xl);
set(grf,'YTickLabel', yl);
set(grf,'ZTickLabel', zl);
title('Energy expenditure in a anverage cycle')
```

```
function plot_COEFS(CGS,COEFS)

global BL

f = 103; figure_close_ifexists(f); figure(f);

for i = 1:length(CGS)
    gs = CGS(i);
    bay = BL.GS(gs).bay;
    stack = BL.GS(gs).stack;
    x(bay) = bay;
    y(stack) = stack;
    z(bay,stack) = COEFS(i);
    plot3(bay,stack,COEFS(i)); hold on
end

%mesh(x,y,z);
```

```
function plot_ct(x,y,z,col)

global S

%  A = [0 0 0];
%  B = [1 0 0];
%  C = [0 1 0];
%  D = [0 0 1];
%  E = [0 1 1];
%  F = [1 0 1];
%  G = [1 1 0];
%  H = [1 1 1];

 %my_vertices = [A;B;F;H;G;C;A;D;E;H;F;D;E;C;G;B];
 %plot3(P(:,1),P(:,2),P(:,3))
h = z*S.h; l = (S.l-.2); w = (S.w -.5);

%my_vertices = [0 0 0; 0 1 0; 1 1 0; 1 0 0; 0 0 1; 0 1 1; 1 1 1; 1 0 1];
my_vertices = [0 0 0; 0 w 0; l w 0; l 0 0; 0 0 h; 0 w h; l w h; l 0 h];
my_vertices(:,1) = my_vertices(:,1)+x-S.l/2;
my_vertices(:,2) = my_vertices(:,2)+y-S.w/2;
%keyboard

my_faces = [1 2 3 4; 2 6 7 3; 4 3 7 8; 1 5 8 4; 1 2 6 5; 5 6 7 8];
colprop = 'FaceColor'; %'interpolated';%
patch('Vertices', my_vertices, 'Faces', my_faces, colprop , col);
```

```
function plot_CT_travel(ct)


global CT S

f = 105;
plot_Block(105)
for pos=1:length(CT(ct).position.bay)
    x(pos) = CT(ct).position.bay(pos)*S.l;
    y(pos) = CT(ct).position.stack(pos)*S.w;
    z(pos) = CT(ct).position.tier(pos)*S.h;
    plot3(x(pos),y(pos),z(pos),'r>');hold on
end

plot3(x,y,z)
```

```
%function plot_energy_bars()

AI40 = [6.36 6.20 6.59 5.47 5.39 5.47];
AE40 = [9.98 9.41 9.49 7.04 7.59 7.73];
AI60 = [5.42 5.48 5.22 5.44 5.45 5.55];
AE60 = [8.80 9.00 9.27 7.02 7.51 7.71];

subplot(2,1,1)
x = [1 3 5 7 9 11];
xl = x+0.5;
bar(x,AI40, 0.5,'g'); hold on
axis([0 13 0 10]);
title('40% Block Occupancy');
set(gca,'XTick', xl)
set(gca,'XTickLabel',{'RSA', 'PRSA', 'LSA', 'ESSA 0-100', 'ESSA 50-50', 'ESSA 100-0'})

x2 = x+1;
bar(x2,AE40, 0.5,'b')
ylabel('E_{c} (kWh)')

subplot(2,1,2)
bar(x,AI60, 0.5,'g'); hold on
axis([0 13 0 10])
title('60% Block Occupancy');
set(gca,'XTick', xl)
set(gca,'XTickLabel',{'RSA', 'PRSA', 'LSA', 'ESSA 0-100', 'ESSA 50-50', 'ESSA 100-0'})

bar(x2,AE60, 0.5,'b')
ylabel('E_{c} (kWh)')
```

```
function plot_intersection(XS,TS,XL,TL,xi,yi)

global ASC BL TIME

f= 200; figure_close_ifexists(f);figure(f);

NP = 10;

% 1. PLOT TRAJECTORIES
% LAND CRANE
n = length(ASC.land.position.time); NP = min(NP,n);
xl = ASC.land.position.bay(n-NP+1:n);
tl = ASC.land.position.time(n-NP+1:n)/3600/24;
ctl = ASC.land.position.ct(n-NP+1:n);
plot(tl,xl,'.-b');hold on
for i=1:NP
   text(tl(i),xl(i),num2str(ctl(i)));
end

% SEA CRANE
n = length(ASC.sea.position.time);NP = min(NP,n);
xs = ASC.sea.position.bay(n-NP+1:n);
ts = ASC.sea.position.time(n-NP+1:n)/3600/24;
cts = ASC.sea.position.ct(n-NP+1:n);
plot(ts,xs,'.-g'); hold on
for i=1:NP
   text(ts(i),xs(i),num2str(cts(i)));
end

% 2. PLOT PROJECTION
plot(TS/3600/24,XS,'.g');
plot(TL/3600/24,XL,'.b');
plot(TS/3600/24,XS,'-.g');
plot(TL/3600/24,XL,'-.b');

LP= ASC_act_pos(ASC.land);
SP= ASC_act_pos(ASC.sea);
plot(LP.time/3600/24,LP.bay,'b*')
plot(SP.time/3600/24,SP.bay,'g*')

% PLOT TIME IN VERTICAL LINE
y(1) = -3; y(2) = BL.bays;
t(1)=TIME.t/3600/24; t(2)=t(1);
tdelt(1) = (TIME.t+TIME.delt)/3600/24; tdelt(2)= tdelt(1);
plot(t,y,'r');
plot(tdelt,y,'r');
grid on

% PLOT THE INTERSECTION POINTS

   plot(yi/3600/24,xi,'xk')
```

```
function plot_inventory()

global BF COUNT TIME

f = 103; figure_close_ifexists(f); figure(f)

yi = COUNT.INVENTORY.imp.no_cts;
xi = COUNT.INVENTORY.imp.time/3600/24;

ye = COUNT.INVENTORY.exp.no_cts;
xe = COUNT.INVENTORY.exp.time/3600/24;

% plot bf arrivals
a = 1:length(BF.sea.imp.arrivals);
b = 1:length(BF.land.exp.arrivals);
c = 1:length(BF.land.imp.arrivals);
d = max(xe);

subplot(2,1,1) % IMP
plot(BF.sea.imp.arrivals/3600/24,a,'k'); hold on
plot(BF.land.imp.arrivals/3600/24,c,'m');
plot(xi,yi,'xg');
title('IMP traffic')
axis([0 TIME.simulation.days 0 1000])
grid on

subplot(2,1,2) % EXP
plot(BF.land.exp.arrivals/3600/24,b,'c'); hold on
plot(xe,ye,'xb');
title('EXP traffic')

d = max(xe);
axis([0 TIME.simulation.days 0 1000]) %BL.tiers*BL.stacks*BL.bays
grid on
```

```
function plot_trf()

global BF

figure_close_ifexists(101); figure(101)

% SEA ARRIVALS
n1 = length(BF.sea.imp.arrivals);
plot(BF.sea.imp.arrivals/24/3600,'g.');
hold on
n2 = length(BF.sea.imp.pickuptimes);
plot(BF.sea.imp.pickuptimes/24/3600,'r.');
n3 = length(BF.sea.exp.trigger);
plot(BF.sea.exp.trigger/24/3600,'k.');

% LAND ARRIVALS
n4 = length(BF.land.exp.arrivals);
plot(BF.land.exp.arrivals/24/3600,'r.');
n5 = length(BF.land.dual.arrivals);
plot(BF.land.dual.arrivals/24/3600,'m.');


grid
%keyboard
```

```
A = sort(CSlots);
B = sort(ASC.sea.tasks.ct);

a=ones(1,length(A));
b=ones(1,length(B));

figure
plot(b,B,'*r'); hold on
plot(a,A,'.')

% C = sort(v_ct);
% c = ones(1,length(C));
% plot(c,C,'og')
```

```
function [t_gs,asc,found] = PSRANDOM_search_drop(ct,asc)
% This function makes a block search for candidate slots to drop a CT

global ASC BAYS BL CT TRF

found = true; t_gs = 0;
acs = 0; bcs = 0; ccs = 0; dcs = 0;
AGS = 0; BGS = 0; CGS = 0; DCS = 0;

[ini_gs, end_gs, del_gs] = ASC_bay_direction(asc);
[imp_bays,exp_bays] = BL_bay_types();

% Satisfy the criteria to be a candidate slot
for gs = ini_gs:del_gs:end_gs
   bay = BL.GS(gs).bay;

   % a) GS and Bay occupation
   [bocup,bres] = BAY_occupation(bay);
   if bocup + bres > BL.tiers*BL.stacks - BL.tiers;
     continue
   end

   if BL.GS(gs).ocup == BL.tiers
      continue
   end

   % b) This is to prevent a worse position during housekeeping
   if asc.housekeeping == 1
      [conflict] = BAY_backwards(bay,ct);
      if conflict == 1
         continue
      end
   end

   % c) Special criteria: if the other asc is delivering, don't put
   if strcmp(asc.id,'land') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.sea);
   elseif strcmp(asc.id,'sea') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.land);
   end
   if conflict == 1
      continue
   end

   % d) Traffic flow.
   bayflow = BL.GS(gs).id;

   if strcmp(CT(ct).id,'IMP') == 1   % Import flow
      % a) Disregard exp bays
      if strcmp(bayflow,'EXP') == 1
         continue
      elseif bocup >= TRF.PARAM.baymaxocup
         continue
```

```
      end
      if and(BL.GS(gs).ocup>0, BL.GS(gs).ocup < 3)
         acs=acs+1;  AGS(acs) = gs;
      elseif BL.GS(gs).ocup == 0
         bcs = bcs+1; BGS(bcs) = gs;
      else
         ccs = ccs+1; CGS(ccs) = gs;
      end

   elseif strcmp(CT(ct).id,'EXP') == 1 % Export flow. Piles can be EXP
      % a) Disregard imp bays
      if strcmp(bayflow,'IMP') == 1
         continue
      end
      % b) Group: disregard other groups
      if BL.GS(gs).group > 0 % For bays with a group assigned
         if BL.GS(gs).group == CT(ct).group
            acs = acs+1;  AGS(acs) = gs;
         else
            ccs = ccs+1; CGS(ccs) = gs;
         end
      elseif BL.GS(gs).group == 0
         if exp_bays > 22
            if bocup > 0
               bcs = bcs + 1; BGS(bcs) = gs;
            end
            %dcs = dcs + 1; DGS(dcs) = gs; % No solution
         else
            bcs = bcs + 1; BGS(bcs) = gs;
         end
      end
      % c) Pile occupation
   end
end


% Analysis of candidates
if acs>0
   GS = AGS; CT(ct).strategy{end+1} = 'A';
elseif bcs>0
   GS = BGS; CT(ct).strategy{end+1} = 'B';
elseif ccs >0
   GS = CGS; CT(ct).strategy{end+1} = 'C';
else
   plot_BL
   disp('PSrandom search error: no candidate slots'); %keyboard
   found = 0;
end

% Determine the coeficients of the candidate slots
if found
   %keyboard
   n = length(GS); CT(ct).strategypossibilities(end+1) = n;
   % Select ground slot randomly
```

```
    t_gs = GS(random('unid',n));
end
```

```
function [t_gs,asc,found] = PSRANDOM_search_drop(ct,asc)
% This function makes a block search for candidate slots to drop a CT

global ASC BAYS BL CT TRF

found = true; t_gs = 0;
acs = 0; bcs = 0; ccs = 0;
AGS = 0; BGS = 0; CGS = 0;

[ini_gs, end_gs, del_gs] = ASC_bay_direction(asc);
[imp_bays,exp_bays] = BL_bay_types();
% if and(imp_bays/exp_bays <16/24, imp_bays>9)
%     plot_BL
%     keyboard
% end

% Satisfy the criteria to be a candidate slot
for gs = ini_gs:del_gs:end_gs
   bay = BL.GS(gs).bay;

   % a) GS and Bay occupation
   [bocup,bres] = BAY_occupation(bay);
   if bocup + bres > BL.tiers*BL.stacks - BL.tiers;
     continue
   end

   if BL.GS(gs).ocup == BL.tiers
      continue
   end

   % b) This is to prevent a worse position during housekeeping
   if asc.housekeeping == 1
      [conflict] = BAY_backwards(bay,ct);
      if conflict == 1
         continue
      end
   end

   % c) Special criteria: if the other asc is delivering, don't put
   if strcmp(asc.id,'land') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.sea);
   elseif strcmp(asc.id,'sea') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.land);
   end
   if conflict == 1
      continue
   end

   % d) Traffic flow.
   bayflow = BL.GS(gs).id;

   if strcmp(CT(ct).id,'IMP') == 1   % Import flow
      % a) Disregard exp bays
```

```
      if strcmp(bayflow,'EXP') == 1
         continue
      elseif bocup >= TRF.PARAM.baymaxocup
         continue
      end
      if and(BL.GS(gs).ocup>0, BL.GS(gs).ocup < 3)
         acs=acs+1;  AGS(acs) = gs;
      elseif BL.GS(gs).ocup == 0
         bcs = bcs+1; BGS(bcs) = gs;
      else
         ccs = ccs+1; CGS(ccs) = gs;
      end

   elseif strcmp(CT(ct).id,'EXP') == 1 % Export flow. Piles can be EXP
      % a) Disregard imp bays
      if strcmp(bayflow,'IMP') == 1
         continue
      end
      % b) Group: disregard other groups
      if BL.GS(gs).group > 0 % For bays with a group assigned
         if BL.GS(gs).group == CT(ct).group
            acs = acs+1;  AGS(acs) = gs;
         else
            ccs = ccs+1; CGS(ccs) = gs;
         end
      elseif BL.GS(gs).group == 0
         if exp_bays > 22 % No new bays allowed, check other bays
            if bocup > 0
               bcs = bcs + 1; BGS(bcs) = gs; %ccs = ccs + 1; CGS(ccs) = gs;
            end
         else
            bcs = bcs + 1; BGS(bcs) = gs;
         end
      end
      % c) Pile occupation
   end
end


% Analysis of candidates
if acs>0
   GS = AGS; CT(ct).strategy{end+1} = 'A';
elseif bcs>0
   GS = BGS; CT(ct).strategy{end+1} = 'B';
elseif ccs >0
   GS = CGS; CT(ct).strategy{end+1} = 'C';
else
   plot_BL
   disp('PSrandom search error: no candidate slots'); %keyboard
   found = 0;
end

% Determine the coeficients of the candidate slots
if found
```

```
    %keyboard
    n = length(GS); CT(ct).strategypossibilities(end+1) = n;
    % Select ground slot randomly
    t_gs = GS(random('unid',n));
% else
%     if and(strcmp(CT(ct).id,'EXP')==1, asc.housekeeping == 1)
%         keyboard
%     end
end
```

```
function [stat] = R_stat(c)

global CT
stat = 0;
if strcmp(CT(c).strategy,'A') == 1
   stat = 1;
elseif strcmp(CT(c).strategy,'B') == 1
   stat = 2;
elseif strcmp(CT(c).strategy,'C') == 1
   stat = 3;
elseif strcmp(CT(c).strategy,'D') == 1
   stat = 4;
else
   strat = 0;
end
```

```
function [no_reshuffles,r_heights] = Reshuffles(gs,tier)
% This function calculates the number of reshuffles and the height that the
% containers must be elevated. The dimension is therefore [CT*heights]

global BL COST

% Initialize outputs
no_reshuffles = 0; r_heights = 0;

% Number of containers in that pile
no_ct = BL.GS(gs).ocup;

% Tier numbers go up to down. Need to get the target CT from down to up.
% rs_ct_tier = BL.tiers - tier+1;

% THe number of CT to be reshuffled will be:
no_reshuffles = no_ct - tier;
% no_reshuffles = no_ct-rs_ct_tier;

% THe height to which CT will be elevated:
h = BL.tiers+1;

% The magnitude of resuffles is
for r=1:no_reshuffles
    ct_height = tier + r; %rs_ct_tier+1;
    ct_elevation = h - ct_height;
    r_heights = r_heights + ct_elevation;
end

COST.reshuffle.IMP = COST.reshuffle.IMP + no_reshuffles;
COST.r_heights.IMP = COST.r_heights.IMP + r_heights;
```

```
if and(Sintslope == 0, Lintslope == 0)
   %keyboard
   if and(LGslopes(1)>0 ,SGslopes(1)>0)
      escenario = 3; matrizcaso = 9; % LAND CRANE PRIORITY
   elseif and(LGslopes(1)<0 ,SGslopes(1)<0)
      escenario = 4; matrizcaso = 1; % SEA CRANE PRIORITY
      keyboard
   else
      escenario = 4; matrizcaso = 5; % SEA CRANE PRIORITY
   end
   %keyboard
elseif and(Sintslope > 0, Lintslope == 0)
   matrizcaso = 8;
   %keyboard
   if and(strcmp(ASC.land.status,'stack')== 1, strcmp(ASC.sea.status,'stack')== 1)
      if ASC.sea.nextevent <= ASC.land.nextevent
         escenario = 4;
      else
         escenario = 3;
      end
   elseif strcmp(ASC.sea.status,'delivery') == 1
      Lbasetime = ASC_basetime(ASC.land);
      Sbasetime = ASC_basetime(ASC.sea);

      if Sbasetime <= Lbasetime
         escenario = 4; % SEA CRANE PRIORITY
      else
         escenario = 3; % LAND CRANE PRIORITY
      end
   else
      [LGslopes] = ASC_gantryslope(ASC.land);
      [SGslopes] = ASC_gantryslope(ASC.sea);
      adddelay = 1.1;
      if and(LGslopes(1)>0 ,SGslopes(1)>0)
         escenario = 3; matrizcaso = 9; % LAND CRANE PRIORITY
      elseif and(LGslopes(1)<0 ,SGslopes(1)<0)
         escenario = 4; matrizcaso = 1; % SEA CRANE PRIORITY
         keyboard
      else
         disp('Hay que descomponer este caso en otros'); keyboard
         if strcmp(ASC.land.status,'delivery') == 1
            escenario = 3; matrizcaso = 5; % LAND CRANE PRIORITY
         else
            escenario = 4; matrizcaso = 5; % SEA CRANE PRIORITY
            keyboard
         end
      end
   end
```

```
        if and(Sintslope == 0, Lintslope == 0)
    keyboard
    lowbay = min(ASC.land.ciclo.bay);
    [t_task_ini] = AUX_find_vector_pos_e(LCbays,lowbay);
    t_ini = TS(t_task_ini+1);
end
```

```
function RESULTS()

global ASC BL COUNT CT TIME TRF

close all
clc
clear IMP EXP
%TRF.PARAM.maxh=5
if strcmp(TRF.PARAM.stackmode, 'essa')==1
    texto = ['Results for ' TRF.PARAM.stackmode num2str(TRF.PARAM.weight.E) num2str(TRF.PARAM.weight.t) '
strategy with traffic level: ' num2str(TRF.PARAM.level) ' maxh ' num2str(TRF.PARAM.maxh)]; %]; %
else
   texto =  ['Results for ' TRF.PARAM.stackmode ' strategy with traffic level: ' num2str(TRF.PARAM.level) ' maxh '
num2str(TRF.PARAM.maxh)]; % ]; %
end
disp('----------------------------------------------------------------------')
disp(texto)
disp('----------------------------------------------------------------------')

t_ini = TIME.cut;   %0; %
t_fin = TIME.t; % 10*3600*24;%

e = 0; i = 0; ie=0;

MOVE.no = 0; MOVE.e = 0; MOVE.t = 0;
OPER.P.G.eunl = MOVE; OPER.P.G.eloa = MOVE; OPER.P.T = MOVE; OPER.P.H = MOVE;
OPER.U.G = MOVE;  OPER.U.T = MOVE; OPER.U.H = MOVE; OPER.U.W = MOVE; OPER.U.D = MOVE;
EXP.S = OPER; EXP.D = OPER; IMP.S = OPER; IMP.D = OPER;

IMP.cont = zeros(1,TRF.PARAM.no.ct_weights); IMP.Ec = zeros(1,TRF.PARAM.no.ct_weights);
EXP.cont = zeros(1,TRF.PARAM.no.ct_weights); EXP.Ec = zeros(1,TRF.PARAM.no.ct_weights);


nc = length(CT);
icont = zeros(1,TRF.PARAM.no.ct_weights);
econt = zeros(1,TRF.PARAM.no.ct_weights);

for c = 1:nc
    c_time = CT(c).events.time(2); % When the ct arrives to TP
    if  and(c_time > t_ini, strcmp(CT(c).events.type(end),'BF Leave')== 1)
        ie=ie+1;
        if and(strcmp(CT(c).events.type(end), 'BF Leave') == 1, length(CT(c).events.type)>6)

            clase = CT(c).w_class;

            if strcmp(CT(c).id,'IMP') == 1
                i = i+1;
                IMP.ids(i) = c;
                IMP.E(i) = CT(c).energy;
                IMP.T(i) = sum(CT(c).ciclotimes);
                IMP.clase(i) = clase;
                IMP.W(i) = CT(c).weight;
                IMP.cont(clase) = IMP.cont(clase)+1;
```

```
        IMP.Ec(clase) = IMP.Ec(clase) + CT(c).energy;
        IMP.R.n(i) = CT(c).R.n;
        IMP.R.induced(i) = CT(c).R.induced;
        IMP.timedif(i) = CT(c).timedif;
        IMP.no_hk(i) = length(CT(c).strategy);
         if CT(c).nr >0
%               if CT(c).timedif == 0
                 keyboard
%               end
         end

        %IMP.res.can(i) = CT(c).reservations.no_candidates;
        %IMP.res.stack(i) = CT(c).reservations.stack;
        %IMP.res.delsea(i) = CT(c).reservations.delivery.sea;
        %IMP.res.delland(i) = CT(c).reservations.delivery.land;
        il = find(strcmp(CT(c).events.type,'BF Leave'));
        ir = find(strcmp(CT(c).events.type,'Requested'));
        IMP.exit(i) = CT(c).events.time(il(end))-CT(c).events.time(ir);
        IMP.avE(i) = mean(IMP.E);
        IMP.group(i) = CT(c).group;
        %gantry,trans,trolley,trolleyr,hoistr,up

        CT_check_UnMoves(c,CT(c).ciclomoves)
        % Energy breakdown
        % Productive
        if isempty(CT(c).ciclomoves)
           keyboard
        end
        if length(CT(c).ciclomoves)==1
           keyboard
        end
        [IMP.S,IMP.D] = Energybreakdown(IMP.S,IMP.D,CT(c).ciclomoves,CT(c).cicloe,CT(c).ciclotimes);

        IMP.stat(i) = R_stat(c);
        %IMP.statpos(i)= CT(c).strategypossibilities;
    elseif strcmp(CT(c).id,'EXP') == 1
        e = e+1;
        EXP.ids(e) = c;
        EXP.E(e) = CT(c).energy;
        EXP.T(e) = sum(CT(c).ciclotimes);
        EXP.W(e) = CT(c).weight;
        EXP.clase(e) = clase;
        EXP.cont(clase) = EXP.cont(clase)+1;
        EXP.Ec(clase) = EXP.Ec(clase) + CT(c).energy;
        EXP.avE(e) = mean(EXP.E);
        EXP.R.n(e) = CT(c).R.n;
        EXP.R.induced(e) = CT(c).R.induced;
        EXP.group(e) = CT(c).group;
        EXP.no_hk(e) = length(CT(c).strategy);
        %gantry,trans,trolley,trolleyr,hoistr,up
        [EXP.S,EXP.D] = Energybreakdown(EXP.S,EXP.D,CT(c).ciclomoves,CT(c).cicloe,CT(c).ciclotimes);
        EXP.stat(e) = R_stat(c);
        %EXP.statpos(e)= CT(c).strategypossibilities;
    else
```

```
            keyboard
          end
        end
    end
end

n = 0; E.R(1) = 0; E.H = zeros(1,BL.tiers);  E.SH = 0; E.SHmean = 0;
T.E(1) = 0;x = 0; nh = zeros(1,BL.tiers); E.NH=0;
E.Rmean = 0;
%COUNT.RE=COUNT.R;
for j = 1:COUNT.RE.events
    if COUNT.RE.revent(j) > t_ini;
        n = n+1;
        x(n) = n;
        T.E(n) = COUNT.RE.revent(j);
        r = COUNT.RE.nr(j);  % number of reshuffles
        hr = COUNT.RE.sheight(j); % Height of the reshuffle
        % Reshuffles and mean
        E.R(n) = r;
        E.Rmean(n) = mean(E.R);
        % Height of the reshuffles and mean
        E.H(r) = E.H(r) + 1;
        E.SH(n) = hr;
        E.SHmean(n) = mean(E.SH);
        E.NH(n) = E.R(n)/E.SH(n);
    end
end

% PLOT THE MEAN NUMBER OF RESHUFFLES AND THE HEIGHT
figure
subplot(3,1,1); plot(COUNT.S.time/3600/24,COUNT.S.I.mean); hold on;
plot(COUNT.S.time/3600/24,COUNT.S.I.std,'r'); axis([0 28 0 BL.tiers]); title('IMP stack height: mean and SD')
subplot(3,1,2); plot(COUNT.S.time/3600/24,COUNT.S.E.mean); hold on;
plot(COUNT.S.time/3600/24,COUNT.S.E.std,'r'); axis([0 28 0 BL.tiers]); title('EXP stack height: mean and SD')
subplot(3,1,3); plot(T.E/3600/24,E.Rmean); hold on;
plot(T.E/3600/24,E.SHmean,'r'); axis([0 28 0 BL.tiers]); title('Reshuffles mean: number and stack height')

Nre = sum(EXP.R.n); Nri = sum(IMP.R.n);
disp(['Number of imp reshuffles: ' num2str(Nri)])
disp(['Number of exp reshuffles: ' num2str(Nre)])
disp(['Number of HK jumps per imp ct: ' num2str(mean(IMP.no_hk)-1)])
disp(['Number of HK jumps per exp ct: ' num2str(mean(EXP.no_hk)-1)])

Nrsi = find(IMP.R.n>1); Nrsi = length(Nrsi);
Nrse = find(EXP.R.n>1); Nrse = length(Nrse);
disp(['Number of secondary imp reshuffles: ' num2str(100*Nrsi/Nri) '%'])
disp(['Number of secondary exp reshuffles: ' num2str(100*Nrse/Nre) '%'])

disp(['Secondary movements per operation (mean number/std): ' num2str(E.Rmean(end)) '/' num2str(std(E.R))])
disp(['Height of pile per delivery operation with reshuffles (mean height/std): ' num2str(E.SHmean(end)) '/'
num2str(std(E.SH)) ])
disp(' ')

% ENERGY
```

```
% Average energy
figure(21); plot(EXP.avE,'r'); hold on; plot(IMP.avE,'g'); title('Average energy per ct')
% CT Energy vs. CT weight
figure(22); plot(EXP.W,EXP.E,'r.',IMP.W,IMP.E,'g.'); title('Energy vs ct weight')

% CT Energy histogram
he = hist(EXP.E,20);
hi = hist(IMP.E,20);

figure(23);
plot(he,'r'); hold on
plot(hi,'g'); title('CT Energy Histogram')

disp(['Numero cts: ' num2str(nc)]);
disp(['Longitud de la simulacion: ' num2str(TIME.t/3600/24)])

% Energy
EXP.mE = mean(EXP.E); IMP.mE = mean(IMP.E);

figure;
Gi=hist(IMP.W,24); subplot(2,1,1); plot(Gi,'.-'); hold on; title('IMP Energy distribution per group');
Ge=hist(EXP.W,24); subplot(2,1,2); plot(Ge,'.-'); hold on; title('EXP Energy distribution per group');
disp('------------------------------------------------------------------')
disp(['Imp cts: ' num2str(i) ' mean energy: ' num2str(IMP.mE) ])
disp(['Exp cts: ' num2str(e) ' mean energy: ' num2str(EXP.mE) ])

disp('- IMPORT ----------------------------------------------------------')
decompose(IMP,i);

disp('- EXPORT ----------------------------------------------------------')
decompose(EXP,e);

% Mean energy per weight class
% ---------------------------------------------------------------------
figure;
title('Mean energy per weight class'); hold on
subplot(2,1,1); plot(EXP.Ec./EXP.cont,'.-');
subplot(2,1,2); plot(EXP.Ec./EXP.cont,'.-');

display(['Export probability of reshuffling: ' num2str(E.H/i)])

% Cumulative Density Plot
% ---------------------------------------------------------------------
figure;
plot(IMP.timedif,IMP.R.induced,'.'); title('IMP Induced Reshuffles')

% Cumulative Density Plot
% ---------------------------------------------------------------------
nr1 = 0 ; nr2 = 0; nr3 = 0; nr4= 0; xcdf = 1:10;

for i =1:length(IMP.timedif)
   nr = IMP.R.induced(i);
   if nr == 1
      nr1 = nr1 +1;
```

```
        R1(nr1) = IMP.timedif(i);
        %ycdf1= cdf(R1,xcdf)
    elseif nr == 2
        nr2 = nr2 +1;
        R2(nr2) = IMP.timedif(i);
    elseif nr == 3
        nr3 = nr3 +1;
        R3(nr3) = IMP.timedif(i);
    elseif nr == 4
        nr4 = nr4 +1;
        R4(nr4) = IMP.timedif(i);
    end
end

figure
cdfplot(R1);  hold on
cdfplot(R2); cdfplot(R3); cdfplot(R4);
y1 = hist(R1,xcdf); y2 = hist(R2,xcdf); y3 = hist(R3,xcdf); y4 = hist(R4,xcdf);
title('Cumulative density plot')

% Vessel stay plot
% ------------------------------------------------------------------------
nv = 0;
for vs = 1: length(TRF.VS)
    if and(TRF.VS(vs).exit > 0,TRF.VS(vs).arrival/3600/24 >= 10)
        nv = nv+1;
        downl(nv) = (TRF.VS(vs).download-TRF.VS(vs).arrival)/3600;
        upl(nv) = (TRF.VS(vs).exit-TRF.VS(vs).download)/3600;
        uamount(nv) = TRF.VS(vs).CT.EXP;
        damount(nv) = TRF.VS(vs).CT.IMP;
        dprod(nv) = TRF.VS(vs).CT.IMP/downl(nv);
        uprod(nv) = TRF.VS(vs).CT.EXP/upl(nv);
    end
end

mdstay = mean(downl); mdprod = mean(dprod); mustay = mean(upl); muprod = mean(uprod);
figure; plot(downl,damount,'.g',upl,uamount,'.r'); xlabel('Vessel Stay'); ylabel('Number of CTs')
disp(['Average time for vessels (D/U): ' num2str(mdstay) '/' num2str(mustay) ' and average prod ' num2str(mdprod) '/'
num2str(muprod)])
disp(['Average time for IMP ct delivery : ' num2str(mean(IMP.exit)/60)])
% Crane historgrams
ASC_traj_histograms()
```

```
function RESULTS()

global ASC BL COUNT CT TIME TRF

close all
clc
clear IMP EXP
%TRF.PARAM.maxh=5
if strcmp(TRF.PARAM.stackmode, 'essa')==1
    texto = ['Results for ' TRF.PARAM.stackmode num2str(TRF.PARAM.weight.E) num2str(TRF.PARAM.weight.t) '
strategy with traffic level: ' num2str(TRF.PARAM.level) ' maxh ' num2str(TRF.PARAM.maxh)]; %]; %
else
   texto =  ['Results for ' TRF.PARAM.stackmode ' strategy with traffic level: ' num2str(TRF.PARAM.level) ' maxh '
num2str(TRF.PARAM.maxh) 'BL ' num2str(BL.bays)]; % ]; %
end

disp('----------------------------------------------------------------------')
disp(texto)
disp('----------------------------------------------------------------------')

t_ini = TIME.cut;   %0; %
t_fin = TIME.t; % 10*3600*24;%

e = 0; i = 0; ie=0;

[IMP,EXP] = Results_ini();

nc = length(CT);
icont = zeros(1,TRF.PARAM.no.ct_weights);
econt = zeros(1,TRF.PARAM.no.ct_weights);

for c = 1:nc
    c_time = CT(c).events.time(2); % When the ct arrives to TP
    if  and(c_time > t_ini, strcmp(CT(c).events.type(end),'BF Leave')== 1)
        ie=ie+1;
        if and(strcmp(CT(c).events.type(end), 'BF Leave') == 1, length(CT(c).events.type)>6)

            clase = CT(c).w_class;

            if strcmp(CT(c).id,'IMP') == 1
                i = i+1;
                IMP.ids(i) = c;
                ipos = CT(c).cicloe > 0;
                IMP.E(i) = sum(CT(c).cicloe(ipos)); %CT(c).energy;
                IMP.Eend(i) = CT(c).events.time(1);
                IMP.T(i) = sum(CT(c).ciclotimes);
                IMP.clase(i) = clase;
                IMP.W(i) = CT(c).weight;
                IMP.cont(clase) = IMP.cont(clase)+1;
                IMP.Ec(clase) = IMP.Ec(clase) + IMP.E(i); %+ CT(c).energy;
                IMP.R.n(i) = length(find(strcmp(CT(c).ciclomoves,'ulower'))); %CT(c).R.n;
                IMP.R.induced(i) = CT(c).R.induced;
                IMP.timedif(i) = CT(c).timedif;
```

```
            IMP.no_hk(i) = length(CT(c).strategy);
             if CT(c).nr >0
%                if CT(c).timedif == 0
                  keyboard
%                end
             end

            %IMP.res.can(i) = CT(c).reservations.no_candidates;
            %IMP.res.stack(i) = CT(c).reservations.stack;
            %IMP.res.delsea(i) = CT(c).reservations.delivery.sea;
            %IMP.res.delland(i) = CT(c).reservations.delivery.land;
            il = find(strcmp(CT(c).events.type,'BF Leave'));
            ir = find(strcmp(CT(c).events.type,'Requested'));
            IMP.exit(i) = CT(c).events.time(il(end))-CT(c).events.time(ir);
            IMP.avE(i) = mean(IMP.E);
            IMP.group(i) = CT(c).group;
            %gantry,trans,trolley,trolleyr,hoistr,up

            CT_check_UnMoves(c,CT(c).ciclomoves)
            % Energy breakdown
            % Productive
            if isempty(CT(c).ciclomoves)
               keyboard
            end
            if length(CT(c).ciclomoves)==1
               keyboard
            end
            [IMP.S,IMP.D,en] = Energybreakdown(IMP.S,IMP.D,CT(c).ciclomoves,CT(c).cicloe,CT(c).ciclotimes);
            if abs(en-IMP.E(i)) > 0, keyboard, end
            IMP.stat(i) = R_stat(c);
            IMP.leaps(i,:) = CT_leaps(c);

        elseif strcmp(CT(c).id,'EXP') == 1
            e = e+1;
            EXP.ids(e) = c;
            epos = CT(c).cicloe > 0;
            EXP.E(e) = sum(CT(c).cicloe(epos)); %CT(c).energy;
            EXP.Eend(e) = CT(c).events.time(1);
            EXP.T(e) = sum(CT(c).ciclotimes);
            EXP.W(e) = CT(c).weight;
            EXP.clase(e) = clase;
            EXP.cont(clase) = EXP.cont(clase)+1;
            EXP.Ec(clase) = EXP.Ec(clase) + EXP.E(e) ; % +CT(c).energy
            EXP.avE(e) = mean(EXP.E);
            EXP.R.n(e) = length(find(strcmp(CT(c).ciclomoves,'ulower'))); %CT(c).R.n;
            EXP.R.induced(e) = CT(c).R.induced;
            EXP.group(e) = CT(c).group;
            EXP.no_hk(e) = length(CT(c).strategy);
            %gantry,trans,trolley,trolleyr,hoistr,up
            [EXP.S,EXP.D,en] = Energybreakdown(EXP.S,EXP.D,CT(c).ciclomoves,CT(c).cicloe,CT(c).ciclotimes);
            if abs(en-EXP.E(e)) > 0, keyboard, end
            EXP.stat(e) = R_stat(c);
            EXP.leaps(e,:) = CT_leaps(c);
        else
```

```
            keyboard
          end
        end
    end
end

n = 0; E.R(1) = 0; E.H = zeros(1,BL.tiers);  E.SH = 0; E.SHmean = 0;
T.E(1) = 0;x = 0; nh = zeros(1,BL.tiers); E.NH=0;
E.Rmean = 0;

%COUNT.RE=COUNT.R;
for j = 1:COUNT.RE.events
    if COUNT.RE.revent(j) > t_ini;
        n = n+1;
        x(n) = n;
        T.E(n) = COUNT.RE.revent(j);
        r = COUNT.RE.nr(j);  % number of reshuffles
        hr = COUNT.RE.sheight(j); % Height of the reshuffle
        % Reshuffles and mean
        E.R(n) = r;
        E.Rmean(n) = mean(E.R);
        % Height of the reshuffles and mean
        E.H(r) = E.H(r) + 1;
        E.SH(n) = hr;
        E.SHmean(n) = mean(E.SH);
        E.NH(n) = E.R(n)/E.SH(n);
    end
end

% PLOT THE MEAN NUMBER OF RESHUFFLES AND THE HEIGHT
figure
subplot(3,1,1); plot(COUNT.S.time/3600/24,COUNT.S.I.mean); hold on;
plot(COUNT.S.time/3600/24,COUNT.S.I.std,'r'); axis([0 28 0 BL.tiers]); title('IMP stack height: mean and SD')
subplot(3,1,2); plot(COUNT.S.time/3600/24,COUNT.S.E.mean); hold on;
plot(COUNT.S.time/3600/24,COUNT.S.E.std,'r'); axis([0 28 0 BL.tiers]); title('EXP stack height: mean and SD')
subplot(3,1,3); plot(T.E/3600/24,E.Rmean); hold on;
plot(T.E/3600/24,E.SHmean,'r'); axis([0 28 0 BL.tiers]); title('Reshuffles mean: number and stack height')

Nri = sum(IMP.R.n);    Nre = sum(EXP.R.n);
disp(['Number of imp reshuffles: ' num2str(Nri)])
disp(['Number of exp reshuffles: ' num2str(Nre)])
disp(['Number of HK jumps per imp ct: ' num2str(mean(IMP.no_hk)-1)]) % COUNT.RhkI.events

disp(['Number of HK jumps per exp ct: ' num2str(mean(EXP.no_hk)-1)]) % COUNT.RhkE.events

Nrsi = find(IMP.R.n>1); Nrsi = length(Nrsi);
Nrse = find(EXP.R.n>1); Nrse = length(Nrse);
disp(['Number of secondary imp reshuffles: ' num2str(100*Nrsi/Nri) '%'])
disp(['Number of secondary exp reshuffles: ' num2str(100*Nrse/Nre) '%'])

disp(['Secondary movements per operation (mean number/std): ' num2str(E.Rmean(end)) '/' num2str(std(E.R))])
disp(['Height of pile per delivery operation with reshuffles (mean height/std): ' num2str(E.SHmean(end)) '/'
num2str(std(E.SH)) ])
disp(' ')
```

```
% Compute the leaps
imp_leaps=zeros(BL.bays,BL.bays); exp_leaps=zeros(BL.bays,BL.bays);
for b = 1:BL.bays
   impf = find(IMP.leaps(:,b)>0);
   for j = 1:length(impf)
      leapL = IMP.leaps(impf(j),b);
      imp_leaps(b,leapL) = imp_leaps(b,leapL) + 1;
   end
   expf = find(EXP.leaps(:,b)>0);
   for j = 1:length(expf)
      leapL = EXP.leaps(expf(j),b);
      exp_leaps(b,leapL) = exp_leaps(b,leapL) + 1;
   end
end
imp_leaps = imp_leaps/sum(sum(imp_leaps));
exp_leaps = exp_leaps/sum(sum(exp_leaps));
figure;bar3(imp_leaps);
figure;bar3(exp_leaps);
%keyboard

% ---------------------------------------------------------------------
% ENERGY
% ---------------------------------------------------------------------

% Average energy
figure(21); plot(EXP.Eend/3600/24,EXP.avE,'b'); hold on; plot(IMP.Eend/24/3600,IMP.avE,'g'); title('Average energy
per ct');
% axis([0 TIME.simulation.T/3600/24 0 max(max(IMP.E),max(EXP.E))*1.2])
xlabel('Days');ylabel('Mean Energy (kWh)')
figure(22); plot(EXP.W,EXP.E,'b.',IMP.W,IMP.E,'g.'); title('Energy vs ct weight')

% CT Energy histogram
he = hist(EXP.E,20);
hi = hist(IMP.E,20);

figure(23);
plot(he,'b'); hold on
plot(hi,'g'); title('CT Energy Histogram')

disp(['Numero cts: ' num2str(nc)]);
disp(['Longitud de la simulacion: ' num2str(TIME.t/3600/24)])

% Energy distribution per group
% ---------------------------------------------------------------------
figure;
Gi=hist(IMP.W,24); subplot(2,1,1); plot(Gi,'.-'); hold on; title('IMP Energy distribution per group');
Ge=hist(EXP.W,24); subplot(2,1,2); plot(Ge,'.-'); hold on; title('EXP Energy distribution per group');
disp('-----------------------------------------------------------')

% Mean energy per weight class
% ---------------------------------------------------------------------
figure;
plot(EXP.Ec./EXP.cont,'b.-'); hold on; plot(IMP.Ec./IMP.cont,'g.-');
```

```
title('Mean energy per CT weight class');
xlabel('CT class');ylabel('Mean Energy (kWh)')

disp('- IMPORT ------------------------------------------------------------')
decompose(IMP,i);

disp('- EXPORT ------------------------------------------------------------')
decompose(EXP,e);

display(['Export probability of reshuffling: ' num2str(E.H/i)])

% Cumulative Density Plot
% ------------------------------------------------------------------------
figure;
plot(IMP.timedif,IMP.R.induced,'.'); title('IMP Induced Reshuffles')

% Probability Density Plot
% ------------------------------------------------------------------------
nr0 = 0 ;nr1 = 0 ; nr2 = 0; nr3 = 0; nr4= 0;

for i =1:length(IMP.timedif)
   nr = IMP.R.induced(i);
   if nr == 0
      nr0 = nr0 +1;
      R0(nr0) = IMP.timedif(i);
   elseif nr == 1
      nr1 = nr1 +1;
      R1(nr1) = IMP.timedif(i);
   elseif nr == 2
      nr2 = nr2 +1;
      R2(nr2) = IMP.timedif(i);
   elseif nr == 3
      nr3 = nr3 +1;
      R3(nr3) = IMP.timedif(i);
   elseif nr == 4
      nr4 = nr4 +1;
      R4(nr4) = IMP.timedif(i);
   end
end
nr0
figure
cdfplot(R0); hold on; cdfplot(R1); cdfplot(R2); cdfplot(R3); cdfplot(R4);
title('Probability Density Plot');
xlabel('Time (days)'); ylabel('Probability')

xcdf = 1:14;
y0 = hist(R0,xcdf);  y1 = hist(R1,xcdf); y2 = hist(R2,xcdf); y3 = hist(R3,xcdf); y4 = hist(R4,xcdf);
figure
plot(y0/sum(y0)); hold on; plot(y1/sum(y1)); plot(y2/sum(y2)); plot(y3/sum(y3)); plot(y4/sum(y4));

% Vessel stay plot
% ------------------------------------------------------------------------
nv = 0;
for vs = 1: length(TRF.VS)
```

```
   if and(TRF.VS(vs).exit > 0,TRF.VS(vs).arrival/3600/24 >= 10)
      nv = nv+1;
      downl(nv) = (TRF.VS(vs).download-TRF.VS(vs).arrival)/3600;
      upl(nv) = (TRF.VS(vs).exit-TRF.VS(vs).download)/3600;
      uamount(nv) = TRF.VS(vs).CT.EXP;
      damount(nv) = TRF.VS(vs).CT.IMP;
      dprod(nv) = TRF.VS(vs).CT.IMP/downl(nv);
      uprod(nv) = TRF.VS(vs).CT.EXP/upl(nv);
   end
end

mdstay = mean(downl); mdprod = mean(dprod); mustay = mean(upl); muprod = mean(uprod);
figure; plot(downl,damount,'.g',upl,uamount,'.r'); xlabel('Vessel Stay'); ylabel('Number of CTs')
disp(['Average time for vessels (D/U): ' num2str(mdstay) '/' num2str(mustay) ' and average prod ' num2str(mdprod) '/'
num2str(muprod)])
disp(['Average time for IMP ct delivery : ' num2str(mean(IMP.exit)/60)])
% Crane historgrams
ASC_traj_histograms()
```

```
function [IMP,EXP] = Results_ini()

global TRF

MOVE.no = 0; MOVE.e = 0; MOVE.t = 0;

OPER.P.G.eunl = MOVE;
OPER.P.G.eloa = MOVE;
OPER.P.T = MOVE;
OPER.P.H = MOVE;
OPER.P.L = MOVE;
OPER.U.G = MOVE;
OPER.U.T = MOVE;
OPER.U.H = MOVE;
OPER.U.L = MOVE;
OPER.U.W = MOVE;
OPER.U.D = MOVE;
EXP.S = OPER; EXP.D = OPER;
IMP.S = OPER; IMP.D = OPER;

IMP.cont = zeros(1,TRF.PARAM.no.ct_weights); IMP.Ec = zeros(1,TRF.PARAM.no.ct_weights);
EXP.cont = zeros(1,TRF.PARAM.no.ct_weights); EXP.Ec = zeros(1,TRF.PARAM.no.ct_weights);
```

```
function RESULTS()

global ASC BL COUNT CT TIME TRF

close all
clc
clear IMP EXP
%TRF.PARAM.maxh=5
if strcmp(TRF.PARAM.stackmode, 'essa')==1
    texto = ['Results for ' TRF.PARAM.stackmode num2str(TRF.PARAM.weight.E) num2str(TRF.PARAM.weight.t) '
strategy with traffic level: ' num2str(TRF.PARAM.level) ' maxh ' num2str(TRF.PARAM.maxh)]; %]; %
else
   texto =  ['Results for ' TRF.PARAM.stackmode ' strategy with traffic level: ' num2str(TRF.PARAM.level) ' maxh '
num2str(TRF.PARAM.maxh) 'BL ' num2str(BL.bays)]; % ]; %
end

disp('----------------------------------------------------------------------')
disp(texto)
disp('----------------------------------------------------------------------')

t_ini = TIME.cut;   %0; %
t_fin = TIME.t; % 10*3600*24;%

e = 0; i = 0; ie=0;

[IMP,EXP] = Results_ini();

nc = length(CT);
icont = zeros(1,TRF.PARAM.no.ct_weights);
econt = zeros(1,TRF.PARAM.no.ct_weights);

for c = 1:nc
    c_time = CT(c).events.time(2); % When the ct arrives to TP
    if  and(c_time > t_ini, strcmp(CT(c).events.type(end),'BF Leave')== 1)
        ie=ie+1;
        if and(strcmp(CT(c).events.type(end), 'BF Leave') == 1, length(CT(c).events.type)>6)

            clase = CT(c).w_class;

            if strcmp(CT(c).id,'IMP') == 1
                i = i+1;
                IMP.ids(i) = c;
                %IMP.E(i) = CT(c).energy;
                IMP.Eend(i) = CT(c).events.time(1);
                IMP.T(i) = sum(CT(c).ciclotimes);
                IMP.clase(i) = clase;
                IMP.W(i) = CT(c).weight;
                IMP.cont(clase) = IMP.cont(clase)+1;
                IMP.Ec(clase) = IMP.Ec(clase) + CT(c).energy;
                IMP.R.n(i) = length(find(strcmp(CT(c).ciclomoves,'ulower'))); %CT(c).R.n;
                IMP.R.induced(i) = CT(c).R.induced;
                IMP.timedif(i) = CT(c).timedif;
                IMP.no_hk(i) = length(CT(c).strategy);
```

```
        if CT(c).nr >0
%           if CT(c).timedif == 0
             keyboard
%           end
        end

        %IMP.res.can(i) = CT(c).reservations.no_candidates;
        %IMP.res.stack(i) = CT(c).reservations.stack;
        %IMP.res.delsea(i) = CT(c).reservations.delivery.sea;
        %IMP.res.delland(i) = CT(c).reservations.delivery.land;
        il = find(strcmp(CT(c).events.type,'BF Leave'));
        ir = find(strcmp(CT(c).events.type,'Requested'));
        IMP.exit(i) = CT(c).events.time(il(end))-CT(c).events.time(ir);

        IMP.group(i) = CT(c).group;
        %gantry,trans,trolley,trolleyr,hoistr,up

        CT_check_UnMoves(c,CT(c).ciclomoves)
        % Energy breakdown
        % Productive
        if isempty(CT(c).ciclomoves)
           keyboard
        end
        if length(CT(c).ciclomoves)==1
           keyboard
        end
        [IMP.S,IMP.D,E] = Energybreakdown10(IMP.S,IMP.D,CT(c).ciclomoves,CT(c).cicloe,CT(c).ciclotimes);
        IMP.E(i) = E;    IMP.avE(i) = mean(IMP.E);
        IMP.stat(i) = R_stat(c);
        IMP.leaps(i,:) = CT_leaps(c);

    elseif strcmp(CT(c).id,'EXP') == 1
        e = e+1;
        EXP.ids(e) = c;
        %EXP.E(e) = CT(c).energy;
        EXP.Eend(e) = CT(c).events.time(1);
        EXP.T(e) = sum(CT(c).ciclotimes);
        EXP.W(e) = CT(c).weight;
        EXP.clase(e) = clase;
        EXP.cont(clase) = EXP.cont(clase)+1;
        EXP.Ec(clase) = EXP.Ec(clase) + CT(c).energy;

        EXP.R.n(e) = length(find(strcmp(CT(c).ciclomoves,'ulower'))); %CT(c).R.n;
        EXP.R.induced(e) = CT(c).R.induced;
        EXP.group(e) = CT(c).group;
        EXP.no_hk(e) = length(CT(c).strategy);
        %gantry,trans,trolley,trolleyr,hoistr,up
        [EXP.S,EXP.D,E] = Energybreakdown10(EXP.S,EXP.D,CT(c).ciclomoves,CT(c).cicloe,CT(c).ciclotimes);
        EXP.E(e) = E;    EXP.avE(e) = mean(EXP.E);
        EXP.stat(e) = R_stat(c);
        EXP.leaps(e,:) = CT_leaps(c);
    else
        keyboard
    end
```

```
      end
   end
end

n = 0; E.R(1) = 0; E.H = zeros(1,BL.tiers);  E.SH = 0; E.SHmean = 0;
T.E(1) = 0;x = 0; nh = zeros(1,BL.tiers); E.NH=0;
E.Rmean = 0;

%COUNT.RE=COUNT.R;
for j = 1:COUNT.RE.events
   if COUNT.RE.revent(j) > t_ini;
      n = n+1;
      x(n) = n;
      T.E(n) = COUNT.RE.revent(j);
      r = COUNT.RE.nr(j);  % number of reshuffles
      hr = COUNT.RE.sheight(j); % Height of the reshuffle
      % Reshuffles and mean
      E.R(n) = r;
      E.Rmean(n) = mean(E.R);
      % Height of the reshuffles and mean
      E.H(r) = E.H(r) + 1;
      E.SH(n) = hr;
      E.SHmean(n) = mean(E.SH);
      E.NH(n) = E.R(n)/E.SH(n);
   end
end

% PLOT THE MEAN NUMBER OF RESHUFFLES AND THE HEIGHT
figure
subplot(3,1,1); plot(COUNT.S.time/3600/24,COUNT.S.I.mean); hold on;
plot(COUNT.S.time/3600/24,COUNT.S.I.std,'r'); axis([0 28 0 BL.tiers]); title('IMP stack height: mean and SD')
subplot(3,1,2); plot(COUNT.S.time/3600/24,COUNT.S.E.mean); hold on;
plot(COUNT.S.time/3600/24,COUNT.S.E.std,'r'); axis([0 28 0 BL.tiers]); title('EXP stack height: mean and SD')
subplot(3,1,3); plot(T.E/3600/24,E.Rmean); hold on;
plot(T.E/3600/24,E.SHmean,'r'); axis([0 28 0 BL.tiers]); title('Reshuffles mean: number and stack height')

Nre = sum(EXP.R.n);
Nri = sum(IMP.R.n);
disp(['Number of imp reshuffles: ' num2str(Nri)])
disp(['Number of exp reshuffles: ' num2str(Nre)])
disp(['Number of HK jumps per imp ct: ' num2str(mean(IMP.no_hk)-1)])
disp(['Number of HK jumps per exp ct: ' num2str(mean(EXP.no_hk)-1)])

Nrsi = find(IMP.R.n>1); Nrsi = length(Nrsi);
Nrse = find(EXP.R.n>1); Nrse = length(Nrse);
disp(['Number of secondary imp reshuffles: ' num2str(100*Nrsi/Nri) '%'])
disp(['Number of secondary exp reshuffles: ' num2str(100*Nrse/Nre) '%'])

disp(['Secondary movements per operation (mean number/std): ' num2str(E.Rmean(end)) '/' num2str(std(E.R))])
disp(['Height of pile per delivery operation with reshuffles (mean height/std): ' num2str(E.SHmean(end)) '/'
num2str(std(E.SH)) ])
disp(' ')

% Compute the leaps
```

```
imp_leaps=zeros(BL.bays,BL.bays); exp_leaps=zeros(BL.bays,BL.bays);
for b = 1:BL.bays
   impf = find(IMP.leaps(:,b)>0);
   for j = 1:length(impf)
      leapL = IMP.leaps(impf(j),b);
      imp_leaps(b,leapL) = imp_leaps(b,leapL) + 1;
   end
   expf = find(EXP.leaps(:,b)>0);
   for j = 1:length(expf)
      leapL = EXP.leaps(expf(j),b);
      exp_leaps(b,leapL) = exp_leaps(b,leapL) + 1;
   end
end
imp_leaps = imp_leaps/sum(sum(imp_leaps));
exp_leaps = exp_leaps/sum(sum(exp_leaps));
figure;bar3(imp_leaps);
figure;bar3(exp_leaps);
%keyboard

% ENERGY
% Average energy
figure(21); plot(EXP.Eend/3600/24,EXP.avE,'b'); hold on; plot(IMP.Eend/24/3600,IMP.avE,'g'); title('Average energy
per ct');
% axis([0 TIME.simulation.T/3600/24 0 max(max(IMP.E),max(EXP.E))*1.2])
xlabel('Days');ylabel('Mean Energy (kWh)')
figure(22); plot(EXP.W,EXP.E,'b.',IMP.W,IMP.E,'g.'); title('Energy vs ct weight')

% CT Energy histogram
he = hist(EXP.E,20);
hi = hist(IMP.E,20);

figure(23);
plot(he,'b'); hold on
plot(hi,'g'); title('CT Energy Histogram')

disp(['Numero cts: ' num2str(nc)]);
disp(['Longitud de la simulacion: ' num2str(TIME.t/3600/24)])

% ---------------------------------------------------------------------
% Energy
% ---------------------------------------------------------------------
EXP.mE = mean(EXP.E); IMP.mE = mean(IMP.E);

figure;
Gi=hist(IMP.W,24); subplot(2,1,1); plot(Gi,'.-'); hold on; title('IMP Energy distribution per group');
Ge=hist(EXP.W,24); subplot(2,1,2); plot(Ge,'.-'); hold on; title('EXP Energy distribution per group');
disp('-----------------------------------------------------------------')
disp(['Imp cts: ' num2str(i) ' mean energy (kWh): ' num2str(IMP.mE) ]); xlabel('CT class'); ylabel('Mean Energy
(kWh)');
disp(['Exp cts: ' num2str(e) ' mean energy (kWh): ' num2str(EXP.mE) ]); xlabel('CT class'); ylabel('Mean Energy
(kWh)');

disp('- IMPORT ----------------------------------------------------')
decompose(IMP,i);
```

```
disp('- EXPORT ------------------------------------------------------')
decompose(EXP,e);

% Mean energy per weight class
% ------------------------------------------------------------------------
figure;
plot(EXP.Ec./EXP.cont,'b.-'); hold on; plot(IMP.Ec./IMP.cont,'g.-');
title('Mean energy per CT weight class');
xlabel('CT class');ylabel('Mean Energy (kWh)')

display(['Export probability of reshuffling: ' num2str(E.H/i)])

% Cumulative Density Plot
% ------------------------------------------------------------------------
figure;
plot(IMP.timedif,IMP.R.induced,'.'); title('IMP Induced Reshuffles')

% Cumulative Density Plot
% ------------------------------------------------------------------------
nr1 = 0 ; nr2 = 0; nr3 = 0; nr4= 0; xcdf = 1:10;

for i =1:length(IMP.timedif)
   nr = IMP.R.induced(i);
   if nr == 1
      nr1 = nr1 +1;
      R1(nr1) = IMP.timedif(i);
      %ycdf1= cdf(R1,xcdf)
   elseif nr == 2
      nr2 = nr2 +1;
      R2(nr2) = IMP.timedif(i);
   elseif nr == 3
      nr3 = nr3 +1;
      R3(nr3) = IMP.timedif(i);
   elseif nr == 4
      nr4 = nr4 +1;
      R4(nr4) = IMP.timedif(i);
   end
end

figure
cdfplot(R1);  hold on
cdfplot(R2); cdfplot(R3); cdfplot(R4);
y1 = hist(R1,xcdf); y2 = hist(R2,xcdf); y3 = hist(R3,xcdf); y4 = hist(R4,xcdf);
title('Cumulative density plot')

% Vessel stay plot
% ------------------------------------------------------------------------
nv = 0;
for vs = 1: length(TRF.VS)
   if and(TRF.VS(vs).exit > 0,TRF.VS(vs).arrival/3600/24 >= 10)
      nv = nv+1;
      downl(nv) = (TRF.VS(vs).download-TRF.VS(vs).arrival)/3600;
      upl(nv) = (TRF.VS(vs).exit-TRF.VS(vs).download)/3600;
```

```
        uamount(nv) = TRF.VS(vs).CT.EXP;
        damount(nv) = TRF.VS(vs).CT.IMP;
        dprod(nv) = TRF.VS(vs).CT.IMP/downl(nv);
        uprod(nv) = TRF.VS(vs).CT.EXP/upl(nv);
    end
end

mdstay = mean(downl); mdprod = mean(dprod); mustay = mean(upl); muprod = mean(uprod);
figure; plot(downl,damount,'.g',upl,uamount,'.r'); xlabel('Vessel Stay'); ylabel('Number of CTs')
disp(['Average time for vessels (D/U): ' num2str(mdstay) '/' num2str(mustay) ' and average prod ' num2str(mdprod) '/'
num2str(muprod)])
disp(['Average time for IMP ct delivery : ' num2str(mean(IMP.exit)/60)])
% Crane historgrams
ASC_traj_histograms()
```

```
function Sea_delivery()

global BF BL COUNT TIME TRF

% Find 80% of the EXP blocks
cs = 0; nts = 0;

day = floor(TIME.t/3600/24)+1;

vs_line= TRF.VS(day).line; %keyboard
for gs = 1:BL.no_gs
   if strcmp(BL.GS(gs).id,'EXP')==1
      nts = nts +1;
      group = BL.GS(gs).group;
      if group > 0
         ctline = TRF.PARAM.CT.groups(group,3);
      else
         ctline = 1000;
      end
      if vs_line ~= ctline
         continue
      elseif BL.GS(gs).sreservations >0
         continue
      elseif BL.GS(gs).Sdelreservations >0
         continue
      else
         for ict = BL.GS(gs).ocup:-1:1
            cs = cs +1;
            CSlots(cs) = BL.GS(gs).cts(ict);
         end
      end
   end
end

if cs < 0.8*nts
  disp('Too many sea delivery reservations to achieve 80%')
  pause(1)
end

if cs > 0
  %plot_BL

  [no_cts] = ASC_add_list_tasks(CSlots);
  %
else
   disp('Sea Delivery error: not enough slots available')
   plot_BL
   keyboard
end

COUNT.CT.sea.exp = COUNT.CT.sea.exp + 1 ; % This is different for TIME_next
COUNT.ASC.sea.tasks = COUNT.ASC.sea.tasks + no_cts;
COUNT.VS = COUNT.VS + 1;
```

```
%disp('-----------------------------------------------------------------')
disp(['SEA delivery commences for VS: ' num2str(COUNT.VS) ])
disp('-----------------------------------------------------------------')
```

```
function Sea_delivery_tasks()

global ASC BL COUNT CT SEA_DELIVERY TIME TRF

%keyboard

COUNT.VS = COUNT.VS + 1;
vs = COUNT.VS;
no_exp_cts = length(SEA_DELIVERY(vs).ct);

disp('-----------------------------------------------------------------')
disp(['ASC(SEA) Adding ' num2str(no_exp_cts) ' delivery tasks for VS: ' num2str(COUNT.VS) ])
disp('-----------------------------------------------------------------')

TRF.VS(vs).download = TIME.t;
TRF.VS(vs).active = 1;

for i = 1:no_exp_cts
   ct = SEA_DELIVERY(vs).ct(i);
   [ASC.sea] = ASC_addtask(ASC.sea,ct,'delivery',-1);
end

COUNT.CT.sea.exp = COUNT.CT.sea.exp + 1 ; % This is different for TIME_next
COUNT.ASC.sea.tasks = COUNT.ASC.sea.tasks + no_exp_cts;
```

```
function Sea_delivery2()
% This function makes the list of containers for export delivery
% It has reshuffles

global BF BL COUNT CT TIME TRF

% Find 80% of the EXP blocks

%keyboard
cs = 0; nts = 0;

vs = COUNT.VS + 1; %floor(TIME.t/3600/24)+1;

vsline= TRF.VS(vs).line; %keyboard

for gs = 1:BL.no_gs
   if strcmp(BL.GS(gs).id,'EXP')==1
      nts = nts + 1;
%        group = BL.GS(gs).group;
%        if group > 0
%           ctline = TRF.PARAM.CT.groups(group,3);
%        else
%           ctline = 1000;
%        end
%        if vsline ~= ctline
%           continue
%        elseif BL.GS(gs).sreservations >0
%           continue
%        elseif BL.GS(gs).Sdelreservations >0
%           continue
%        else
%           for ict = BL.GS(gs).ocup:-1:1
%              cs = cs +1;
%              CSlots(cs) = BL.GS(gs).cts(ict);
%           end
%        end

      if BL.GS(gs).ocup > 0
         for ict = BL.GS(gs).ocup:-1:1
            ct = BL.GS(gs).cts(ict);
            group = CT(ct).group;
            ctline = TRF.PARAM.CT.groups(group,3);
            if ctline == vsline
               cs = cs +1;
               CSlots(cs) = BL.GS(gs).cts(ict);
            end
         end
      end
   end
end

if cs < 0.8*nts
  disp('Too many sea delivery reservations to achieve 80%')
```

```
   pause(1)
end

if cs > 0
   %plot_BL
   [no_cts] = ASC_+add_list_tasks(CSlots);
   TRF.VS(vs).CT.EXP = no_cts;
   TRF.VS(vs).active = 1;
else
    disp('Sea Delivery error: not enough slots available')
    plot_BL
    keyboard
end

COUNT.CT.sea.exp = COUNT.CT.sea.exp + 1 ; % This is different for TIME_next
COUNT.ASC.sea.tasks = COUNT.ASC.sea.tasks + no_cts;
COUNT.VS = COUNT.VS + 1;

%disp('-----------------------------------------------------------------')
disp(['SEA delivery commences for VS: ' num2str(COUNT.VS) ])
disp('-----------------------------------------------------------------')
```

```
function Sea_delivery3()
% This function makes the list of containers for export delivery
% It has reshuffles

global ASC BL COUNT CT SEA_DELIVERY TIME TRF

% Find 80% of the EXP blocks


cs = 0; nts = 0;

COUNT.VS = COUNT.VS + 1; vs = COUNT.VS; %floor(TIME.t/3600/24)+1;

disp('-----------------------------------------------------------------')
disp(['SEA delivery commences for VS: ' num2str(COUNT.VS) ])
disp('-----------------------------------------------------------------')

TRF.VS(vs).download = TIME.t;

vsline= TRF.VS(vs).line; %keyboard

for gs = 1:BL.no_gs
   if strcmp(BL.GS(gs).id,'EXP')==1
      nts = nts + 1;
%        group = BL.GS(gs).group;
%        if group > 0
%           ctline = TRF.PARAM.CT.groups(group,3);
%        else
%           ctline = 1000;
%        end
%        if vsline ~= ctline
%           continue
%        elseif BL.GS(gs).sreservations >0
%           continue
%        elseif BL.GS(gs).Sdelreservations >0
%           continue
%        else
%           for ict = BL.GS(gs).ocup:-1:1
%              cs = cs +1;
%              CSlots(cs) = BL.GS(gs).cts(ict);
%           end
%        end

      if BL.GS(gs).ocup > 0
         for ict = BL.GS(gs).ocup:-1:1
            ct = BL.GS(gs).cts(ict);
            group = CT(ct).group;
            ctline = TRF.PARAM.CT.groups(group,3);
            if ctline == vsline
               cs = cs +1;
               CSlots(cs) = BL.GS(gs).cts(ict);
            end
         end
```

```
        end
    end
end

if cs > 0
    factor = TRF.PARAM.perc_imp; %0.8; %TRF.PARAM.rand(COUNT.VS+1);
    no_exp_cts = fix(factor*cs);
    cont = 0; %keyboard
    for i = 1:no_exp_cts
        cont = cont + 1; ct = CSlots(i);
        [ASC.sea] = ASC_addtask(ASC.sea,ct,'delivery',-1);
        CT(ct).position.vs = vs;
        CT(ct).pointed = CT(ct).pointed +1;
        if CT(ct).pointed > 1
            disp(['CT(' num2str(ct) ') should have been delivered is a previous vessel'])
        end
        SEA_DELIVERY(vs).ct(cont) = ct;
        t_gs = CT(ct).position.gs(end);
        BL.GS(t_gs).ctspointed = BL.GS(t_gs).ctspointed + 1; % Delivery reservation
        %BL.GS(t_gs).Sdelreservations = BL.GS(t_gs).Sdelreservations + 1; % Delivery reservation
    end
    TRF.VS(vs).CT.EXP = no_exp_cts;
    TRF.VS(vs).active = 1;
else
    disp('Sea Delivery error: not enough slots available')
    plot_BL
    keyboard
end

COUNT.CT.sea.exp = COUNT.CT.sea.exp + 1 ; % This is different for TIME_next
COUNT.ASC.sea.tasks = COUNT.ASC.sea.tasks + no_exp_cts;
```

```
function Sea_reservation()
% This function makes the list of containers for export delivery
% It has reshuffles

global ASC BL COUNT CT SEA_DELIVERY TIME TRF

% Find 80% of the EXP blocks
%keyboard

cs = 0; nts = 0;

%COUNT.VS = COUNT.VS + 1;
vs = COUNT.VS+1; %floor(TIME.t/3600/24)+1;
SEA_DELIVERY(vs).sea_reservation = 1;

disp('-----------------------------------------------------------------')
disp(['SEA Reservation for VS: ' num2str(vs) ])
disp('-----------------------------------------------------------------')

vsline= TRF.VS(vs).line;

for gs = 1:BL.no_gs
   if strcmp(BL.GS(gs).id,'EXP')==1
      nts = nts + 1;
      if BL.GS(gs).ocup > 0
         for ict = BL.GS(gs).ocup:-1:1
            ct = BL.GS(gs).cts(ict);
            group = CT(ct).group;
            ctline = TRF.PARAM.CT.groups(group,3);
            if ctline == vsline
               cs = cs +1;
               Cslots(cs,1) = ct;
               Cslots(cs,2) = ict;
               CT(ct).vs = vs;
            end
         end
      end
   end
end

if cs > 0
   factor = TRF.PARAM.perc_imp; %TRF.PARAM.rand(COUNT.VS+1);
   no_exp_cts = fix(factor*cs);
   cont = 0;
   %keyboard
   indlist = randperm(cs);
   indlist = indlist(1:no_exp_cts);
   indlist = sort(indlist);
   Cslots = Cslots(indlist,:);
   %Cslots = flip(sortrows(Cslots,2));
   for i = 1:no_exp_cts
      cont = cont + 1;
      ct = Cslots(i,1);
```

```
        CT(ct).position.vs = vs;
        HK_ct_priority(ct);
        CT(ct).pointed = CT(ct).pointed + 1;
        if CT(ct).pointed > 1
            disp(['CT(' num2str(ct) ') should have been delivered is a previous vessel'])
        end
        SEA_DELIVERY(vs).ct(i) = ct;
        t_gs = CT(ct).position.gs(end);
        BL.GS(t_gs).ctspointed = BL.GS(t_gs).ctspointed + 1; % Delivery reservation
        %BL.GS(t_gs).Sdelreservations = BL.GS(t_gs).Sdelreservations + 1; % Delivery reservation
    end
    TRF.VS(vs).CT.EXP = no_exp_cts;
    %TRF.VS(vs).active = 1;
else
    disp('Sea Delivery error: not enough slots available')
    plot_BL
    keyboard
end
```

```
function [no_cts] = SEARCH_Housekeeping()
% This function makes the list of containers for export delivery
% There are two types of HK:
% Premove: the CT has not been really requested:
% - IMP CTs: the delivery time far from the actual time
% - EXP CTs: the associated VS is there or it is the next one
% Prepositioning:
% - IMP CTs: the delivery time is close to the actual time
% - EXP CTs: the complimentary condition

global ASC BAYS BL COUNT CT TIME TRF

AVL = []; IVL = [];
% First determine the HK priority:
% - During day (7 and 21) we have import priority to help
% - During night, export

[day,hour] = TIME_split();

% if strcmp(asc.id,'sea') == 1
%     flow = 'EXP';
% Determine the number of vessels present at the facility
avs = 0; ivs = 0;
for vs = 1:TIME.simulation.days + 1
   if TRF.VS(vs).active == 1
      avs = avs +1;
      AVL(avs) = TRF.VS(vs).line;
   end
   % Vessels which will come in the next 24 hrs
   t_left = TRF.VS(vs).arrival - TIME.t;
   if and(t_left >0, t_left < 24*3600)
      %keyboard
      ivs = ivs +1;
      IVL(ivs) = TRF.VS(vs).line;
   end
end

if ivs > 0 % Priority to IMP operations
   pflow= 'EXP';
else
   if and(hour >7, hour <21) % Day: Priority to IMP operations
      pflow = 'IMP';
   else
      pflow= 'EXP';
   end
end

seabay = BL.hklimit.sea+1;   seags = BAYS(seabay).GS(1);
landbay = BL.hklimit.land-1; landgs = BAYS(landbay).GS(end);

if strcmp(pflow,'EXP') == 1
   p_ct_list = ASC.hktasks.EXP;
   s_ct_list = ASC.hktasks.IMP;
```

```
    npflow = 'IMP';
    end_gs = seags; ini_gs = landgs;  del_gs = -1;
elseif strcmp(pflow,'IMP') == 1
    p_ct_list = ASC.hktasks.IMP;
    s_ct_list = ASC.hktasks.EXP;
    npflow = 'EXP';
    ini_gs = seags;  end_gs = landgs;  del_gs = 1;
end


% Analyze first the ct list that coincides with the flow
if isempty(p_ct_list) == 0
    [fct_list,priority] = HK_list_analyze(pflow,p_ct_list,IVL,AVL);
end

% If there is no priority or priority = 2,
if or(priority == 0, priority == 2)
    keyboard
    if isempty(s_ct_list) == 0
        [fct_list2,priority2] = HK_list_analyze(pflow,s_ct_list,IVL,AVL);
    end
    % And if the other flow has prioroity,
    if priority2 == 1
        fct_list = fct_list2;
        priority = priority2;
    end
end

if priority == 1
    no_cts = ASC_add_list_housekeeping(fct_list,pflow,priority);
elseif priority == 2
    no_cts = ASC_add_list_housekeeping(fct_list,npflow,priority);
else
    disp(['Housekeeping search warning: not candiates available'])
    %plot_BL
    %keyboard
end
```

```
function [exp_list,imp_list] = SEARCH_Housekeeping2()
% This function makes the list of containers for export delivery
% There are two types of HK:
% Premove: the CT has not been really requested:
% - IMP CTs: the delivery time far from the actual time
% - EXP CTs: the associated VS is there or it is the next one
% Prepositioning:
% - IMP CTs: the delivery time is close to the actual time
% - EXP CTs: the complimentary condition

global ASC BAYS BL COUNT CT TIME TRF


% 1. Start with IMP cts
keyboard
e = 0;  ct_list = 0;

for i = 1:length(ASC.hktasks.nonprior.IMP)
   ct = ASC.hktasks.nonprior.IMP(i);
   if CT(ct).priority == 1
      keyboard
   end
   priority = HK_ct_priority(ct);
   if priority == 2
      % Save the list of erased cts for later
      e = e+1; ct_list(e) = ct;
      % add the ct to the prior list
      ASC.hktasks.prior.IMP(end+1) = ct;
   end
end
%keyboard
% Erase cts if any
for i = 1:e
   pos = find(ASC.hktasks.nonprior.IMP == ct_list(i));
   ASC.hktasks.nonprior.IMP(pos) = [];
end

% 2. Now with EXP cts
e = 0;  ct_list = 0;

for i = 1:length(ASC.hktasks.nonprior.EXP)
   ct = ASC.hktasks.nonprior.EXP(i);
   if CT(ct).priority == 1
      keyboard
   end
   priority = HK_ct_priority(ct);
   if priority == 2
      % Save the list of erased cts for later
      e = e+1; ct_list(e) = ct;
      % add the ct to the prior list
      ASC.hktasks.prior.EXP(end+1) = ct;
   end
end
```

```
% Erase EXP cts if any
for i = 1:e
    pos = find(ASC.hktasks.nonprior.EXP == ct_list(i));
    ASC.hktasks.nonprior.EXP(pos) = [];
end

exp_list = length(ASC.hktasks.prior.EXP);
imp_list = length(ASC.hktasks.prior.IMP);
```

```
function [exp_list,imp_list] = SEARCH_Housekeeping3()
% This function makes the list of containers for hk
% There are two types of HK:
% Premove: the CT has not been really requested:
% - IMP CTs: the delivery time far from the actual time
% - EXP CTs: the associated VS is there or it is the next one
% Prepositioning:
% - IMP CTs: the delivery time is close to the actual time
% - EXP CTs: the complimentary condition

global ASC CT EXEC TIME

% 1. Start with IMP cts
%keyboard
ct_list = ASC.hktasks.nonprior.IMP;

for i = 1:length(ct_list)
   ct = ct_list(i);
   if CT(ct).priority == 2
      keyboard
   end
   HK_ct_priority(ct);
end


% 2. Now with EXP cts
ct_list = ASC.hktasks.nonprior.EXP;

for i = 1:length(ct_list)
   ct = ct_list(i);
   if CT(ct).priority == 2
      keyboard
   end
   HK_ct_priority(ct);
end

exp_list = length(ASC.hktasks.prior.EXP);
imp_list = length(ASC.hktasks.prior.IMP);

EXEC.lasthksearch = TIME.t;

% Sort the lists according to height and bay position
ASC.hktasks.prior.IMP = HK_list_sort(ASC.hktasks.prior.IMP,'IMP');
ASC.hktasks.prior.EXP = HK_list_sort(ASC.hktasks.prior.EXP,'EXP');
ASC.hktasks.nonprior.IMP = HK_list_sort(ASC.hktasks.nonprior.IMP,'IMP');
ASC.hktasks.nonprior.EXP = HK_list_sort(ASC.hktasks.nonprior.EXP,'EXP');
```

```
% [xSmoothed, ySmoothed] = SMOOTHLINE(x,y,smoothAmount)
% Creates the data for a smooth line going through the points given.
% Unlike spline this does not create artifacts, it locks
% the line to the data points (i.e. it dosen't go above
% or below the Y data points).
%
% Usage:
% x = (1:10);
% y = rand(1,10);
% [xx,yy] = smoothLine(x,y);
% plot(x,y,'or-');
% hold on;
% plot(xx,yy);
%
% The user has the option of changing how smooth the line gets.
% [xx,yy] = smoothLine(x,y,10);
%
% Author: Andrew Hastings (based on work by Patty Pun)
% Version: 6.1 R12.1
function varargout = smoothLine(varargin)

smoothAmt = 1000;

if nargin >= 2
   x = varargin{1};
   y = varargin{2};
end
if nargin >= 3
   smoothAmt = varargin{3};
end

% The type of interpolation can be:
% 'cubic' - creates smooth data points for a smooth line (best results).
% 'pchip' - same as cubic.
% 'nearest' - creates a square wave effect.
% Do NOT use interpolation:
% 'linear' - results are as though this function were not used.
% 'spline' - resulting data/line will not be an accurate representation of the data. Might as well use the SPLINE function.

xInterp = x(1):(x(2)-x(1))/smoothAmt:x(end); % Set up the mesh.
yInterp = interp1(x,y,xInterp,'cubic');

varargout{1} = xInterp;
varargout{2} = yInterp;
```

```
function [RT,PT,RE,PE] = Stack_coefs(C,R)
% This function calculates the energy and time used to calculate the
% candidate slot
global BL

%keyboard
RT = 0; RE = 0; PE = 0; PT = 0;

for t = 1:length(C.bay);
   include = 0; t_include = 1;

   if strcmp(C.moves(t),'gantry') == 1
      include = 0; t_include = 0;
   elseif strcmp(C.moves(t),'trolley') == 1
      include = 1;
   elseif strcmp(C.moves(t),'pickbf') == 1
      include = 0;
   elseif strcmp(C.moves(t),'trans') == 1
      keyboard
   elseif strcmp(C.moves(t),'ugantry') == 1
      include = 1; keyboard
   elseif strcmp(C.moves(t),'pickbl') == 1
      include = 1;
   elseif strcmp(C.moves(t),'raise') == 1
      include = 1;
   elseif strcmp(C.moves(t),'lower') == 1
      include = 1;
   elseif strcmp(C.moves(t),'ulower') == 1
      include = 1;
   elseif strcmp(C.moves(t),'upickbl') == 1
      include = 1;
   elseif strcmp(C.moves(t),'uraise') == 1
      include = 1;
   elseif strcmp(C.moves(t),'utrolley') == 1
      include = 1;
   elseif strcmp(C.moves(t),'dropbl') == 1
      %keyboard
      include = 1;
      t_tier = C.tier(t);
      t_ct = C.ctmove(t);
   elseif strcmp(C.moves(t),'udropbl') == 1
      %keyboard
      include = 1;
   else
      disp(C.moves(t)); keyboard
   end

   if include == 1
      RE = RE + C.E(t);
   end
   if t_include == 1
      RT = RT + C.time(t);
   end
```

```
end

if sum(R.time) > 0
    %keyboard
    % Calculate the future pickup
    [ftime,fE,fmove] = ASC_energy(BL.tiers+1,t_tier,t_ct,'pickbl','full');
    PT = sum(R.time)+ftime;
    PE = sum(R.E)+fE;
end
```

```
function stopatct(ct, nct)

if ct == nct
    keyboard
end
```

```
function [t_gs,asc,found] = TERCAT_search_drop(ct,asc,demand)

global ASC BAYS BF BL CT TRF

%keyboard
acs = 0;  bcs = 0;  ccs = 0;  dcs= 0;
ACGS = 0; BCGS = 0; CCGS = 0; DCGS = 0;
found = true; t_gs = 0;

% Determine the bays within a solution can be found with priority

if asc.housekeeping == 1
   modo = 'hkstack';
else
   modo = asc.tasks.action(asc.tasks.current);
   if strcmp(modo,'delivery')==1
      keyboard
   end
end

[minbay,maxbay] = BAY_limit(CT(ct).id,modo,demand);

[ini_gs, end_gs, del_gs] = ASC_bay_direction(asc);

[imp_bays,exp_bays] = BL_bay_types();

% Satisfy the criteria to be a candidate slot
for gs = ini_gs:del_gs:end_gs
   bay = BL.GS(gs).bay;
   % a) GS and Bay occupation
   [bocup,bres] = BAY_occupation(bay);
   if bocup + bres > BL.tiers*BL.stacks - BL.tiers;
     continue
   end

   if BL.GS(gs).ocup == BL.tiers
      continue
   end

   % b) This is to prevent a worse position during housekeeping
   if asc.housekeeping == 1
      [conflict] = BAY_backwards(bay,ct);
      if conflict == 1
         continue
      end
   end

   % c) Special criteria: if the other asc is delivering, don't put
   if strcmp(asc.id,'land') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.sea);
   elseif strcmp(asc.id,'sea') == 1
      [conflict] = BAY_prevent_conflicts(gs,asc,ASC.land);
   end
```

```
if conflict == 1 % and(, bay < BL.bays)
   continue
end

% d) Analysis of the flow
bayflow = BL.GS(gs).id;
% FOR EXPORT FLOW
% -----------------------------------------------------------------
if strcmp(CT(ct).id,'EXP') == 1
   if strcmp(bayflow,'IMP')==1 % Disregard IMP
      continue
   else
      if and(bay > BL.hklimit.sea, bay < BL.hklimit.land)
         if BL.GS(gs).group > 0 % the pile has containers
            if CT(ct).group == BL.GS(gs).group
               acs = acs + 1; ACGS(acs) = gs;
            else
               ccs = ccs + 1; CCGS(ccs) = gs;
            end
         elseif BL.GS(gs).group == 0
            if imp_bays/exp_bays < 0.6
               dcs = dcs + 1; DCGS(dcs) = gs;
            else
               bcs = bcs + 1; BCGS(bcs) = gs;
            end
         end
      else
         dcs = dcs + 1; DCGS(dcs) = gs;
      end
   end

% FOR IMPORT FLOW
% -----------------------------------------------------------------
elseif strcmp(CT(ct).id,'IMP') == 1
   if strcmp(bayflow,'EXP') == 1 % Disregard export bays
      continue
   end
   if BL.GS(gs).ocup >= BL.tiers-1
      continue
   end
   if bocup >= TRF.PARAM.baymaxocup
      continue
   end
   % TERCAT criteria
   if and(bay > BL.hklimit.sea, bay < BL.hklimit.land)
      if and(BL.GS(gs).ocup > 0, BL.GS(gs).ocup < 3)
         if bocup < 27 % Then can be A
            acs = acs+1;    ACGS(acs) = gs;
         else
            ccs = ccs+1;    CCGS(ccs) = gs;
         end
      else
         if bocup < 27 % Then can be A
            bcs = bcs+1;    BCGS(bcs) = gs;
```

```
            else
               ccs = ccs+1;    CCGS(ccs) = gs;
            end
         end
      else
         dcs = dcs + 1; DCGS(dcs) = gs;
      end
   end
end

% Determine the coeficients of the candidate slots

if acs > 0
   GS = ACGS;     CT(ct).strategy{end+1} = 'A';
elseif bcs >0
   GS = BCGS;     CT(ct).strategy{end+1} = 'B';
elseif ccs > 0
   GS = CCGS;     CT(ct).strategy{end+1} = 'C';
elseif dcs > 0
   GS = DCGS;     CT(ct).strategy{end+1} = 'D';
else
   found = false;
end

if found == true
   % keyboard
   CT(ct).strategypossibilities(end+1) = length(GS);
   % Select ground slot
   t_gs = GS(1);
else
   % If we ct belongs to the list of tasks (does not belong to HKtasks)
   if asc.housekeeping == 1
      disp(['Tercat search warning: no candidate slots to HK ' CT(ct).id ' CT (' num2str(ct) ')']); %keyboard
   else
      disp(['Tercat search warning: no candidate slots to Stack ' CT(ct).id ' CT (' num2str(ct) ')']); %keyboard
      %keyboard
   end
end
```

```
global CT
c = 0;
for i = 1:ct
    CT(ct).id
   if strcmp(CT(ct).id , 'EXP') == 1
       c = c+1;
       G(c) = CT(ct).group;
   end
end
keyboard
figure; plot(G)
```

```
clear leave arrival dwell sleave

cts_imp_vs = 91;
av_dwell = 3.4722;

K = 20;
L = cts_imp_vs/gamma(1+1/K);
block_cts_rate = 3600/10;
tot_cts = fix(wblrnd(L,K));

c = 0;
for day = 1:10
   basetime = (day-1)*3600*24 + ceil(3600* ((day-1)*24 + random('unif',8,20)));
   for i = 1:cts_imp_vs
      c = c+1;
      delt = poissrnd(block_cts_rate);
      arrival(c) = basetime + delt;
      dwell(c)=  exprnd(av_dwell*3600*24);
      leave(c) = arrival(c) + dwell(c);
      basetime = basetime + delt;
   end

end

no_cts = 1:c;

sleave = sort(leave);

f = 6; figure_close_ifexists(f); figure(f)
plot(arrival/3600/24,no_cts,'.-k'), hold on
plot(sleave/3600/24,no_cts,'.-c')

grid on

delt = 60; % seg

A = interp(arrival,10);
L = interp(sleave,10);
```

```
par = 20;

for i = 1: 100000
    a(i) = ceil(random('unif',0,par));
end

b = hist(a,par);

figure; plot(a)
figure;plot(b)
```

```
for i =1:10000
    r(i) = exprnd(3.47);
end

r = sort(r);
h = hist(r);
figure(1);plot(r,'.')
figure(2); plot(h)
```

```
clear inventario

ct =150
inventario(1) =ct;
for dia=2:10
    inventario(dia) = inventario(dia-1)*0.2+ct;
end

figure(4); plot(inventario)
```

```
function TIME_init()

global TIME PLOT

TIME.t = 0;
TIME.delt = 0;
TIME.day = floor(TIME.t/24) + 1;

TIME.start = clock;
TIME.simulation.fill = 0;
%TIME.fill = 3.5;
TIME.simulation.weeks = 4;                    % In weeks
TIME.simulation.days = TIME.simulation.weeks * 7;      % In days
TIME.simulation.T = TIME.simulation.days * 24 * 60 * 60; % In secs
TIME.progreso = 0;
TIME.it = 0;
TIME.e = 0;
TIME.parar = 0;

PLOT.fig = 0;
```

```
function TIME_next()

global ASC BF COUNT EXEC SEA_DELIVERY TIME

% if TIME.it == 9636
%     keyboard
% end

TIME_progreso()

TIME.it = TIME.it + 1;

% Check the ASCs workload
[ASC.sea] = ASC_check_wl(ASC.sea);
[ASC.land] = ASC_check_wl(ASC.land);

TIME.t = TIME.t + TIME.delt;
TIME.day = floor(TIME.t/24/3600) + 1;

TIME.delt = 0;

% Events
TIME.e =0;

% LAND
%  Arrival of ET to buffer
lec = COUNT.CT.land.exp + 1;
tdel(1) = BF.land.exp.arrivals(lec) - TIME.t;
action{1} = ['EXP Truck arrival'];

lic = COUNT.CT.land.imp + 1;
tdel(2) = BF.land.imp.arrivals(lic) - TIME.t;
action{2} = ['IMP Truck arrival'];

ldc = COUNT.CT.land.dual + 1;
tdel(3) = BF.land.dual.arrivals(ldc) - TIME.t;
action{3} = ['Dual Truck arrival'];

% SEA
% Arrival of vessel
sic = COUNT.CT.sea.imp + 1;
tdel(4) = BF.sea.imp.arrivals(sic) - TIME.t;
action{4} = ['Vessel arrival'];

sec = COUNT.CT.sea.exp + 1;
tdel(5) = BF.sea.exp.trigger(sec) - TIME.t;
action{5} = 'Start vessel loading';

%  SEA ASC action
tdel(6) = ASC.sea.nextevent;
action{6} = ['ASC sea ' ASC.land.status];

%  LAND ASC action
```

```
tdel(7) = ASC.land.nextevent;
action{7} = ['ASC land ' ASC.land.status];

[TIME.delt,TIME.e] = min(tdel);

if TIME.delt < 0
   disp(['Time delt error' num2str(TIME.delt)])
   keyboard
end

% Now check if there are events that take place at the exact same time
s_ev = find(min(tdel(1:5)) == tdel(1:5));
if length(s_ev) > 1
   disp('Simultaneous events'); %keyboard
   it = 0; inc_t = 0.01;
   for ev = s_ev(end-1):-1:s_ev(1)
      it = it + 1;
      if ev == 2
         BF.land.imp.arrivals(lic) = BF.land.imp.arrivals(lic) + it; % add one second
      elseif ev == 3
         BF.land.dual.arrivals(ldc) = BF.land.dual.arrivals(ldc) + it;
      elseif ev == 4
         BF.sea.imp.arrivals(sic) = BF.sea.imp.arrivals(sic) + it;
      elseif ev == 5
         BF.sea.exp.trigger(sec) = BF.sea.exp.trigger(sec) + it;
%       elseif ev == 6
%          ASC.sea.nextevent = ASC.sea.nextevent + inc_t;
%          c_time = find(ASC.sea.ciclo.time>0); c_time = c_time(1);
%          ASC.sea.ciclo.time(c_time) = ASC.sea.ciclo.time(c_time) +inc_t;
%          ASC.sea.ciclo.originaltime(c_time) = ASC.sea.ciclo.originaltime(c_time) +inc_t;
%       elseif ev == 7
%          ASC.land.nextevent = ASC.land.nextevent + inc_t;
%          c_time = find(ASC.land.ciclo.time>0); c_time = c_time(1);
%          ASC.land.ciclo.time(c_time) = ASC.land.ciclo.time(c_time) +inc_t;
%          ASC.land.ciclo.originaltime(c_time) = ASC.land.ciclo.originaltime(c_time) +inc_t;
      end
   end
end
%
if TIME.t < EXEC.cutofftime
   if TIME.e == 4
      if SEA_DELIVERY(COUNT.VS+1).sea_reservation == 0
         Sea_reservation();
      end
   end
else

   if tdel(4) < 24*3600
      %keyboard
      if SEA_DELIVERY(COUNT.VS+1).sea_reservation == 0
         Sea_reservation();%keyboard
      end
   end
end
```

```
disp('----------------------------------------------------------------------')
disp(['>>>> Time (' num2str(TIME.t/3600/24) ') Step (' num2str(TIME.it) '): ' num2str(TIME.delt) ' secs. Next event: '
action{TIME.e}])
```

```
function TIME_progreso()

global ASC BAYS BF BL COUNT COST CT INTERS LIMITS PLOT PT S SEA_DELIVERY TIME TRF

if 100 * TIME.t/TIME.simulation.T > TIME.progreso
   disp(['Progreso = ' num2str(100 * TIME.t/TIME.simulation.T)])
   disp('------------------------------------------------------------')
   TIME.progreso = TIME.progreso + 5;
%    if TIME.progreso == 95
%        keyboard
%    end
   plot_BL
   plot_inventory
   if TRF.PARAM.averias == 1
      TIME.parar = 0;
      disp('Saving state for hot start')
      disp('------------------------------------------------------------')
      if exist('previoustate.mat')==2
         movefile('previoustate.mat','antepreviousstate.mat');
      end

save('previoustate.mat','ASC','BAYS','BF','BL','COST','COUNT','CT','INTERS','LIMITS','PLOT','PT','S','SEA_DELIVE
RY','TIME','TRF');
   end
end
```

```
function [day,hour] = TIME_split()

global TIME

day = floor(TIME.t/3600/24);
hour = floor(TIME.t/3600/24 -day)*24;
```

```
function [x,t,full_speed,ms] = travel_time(X,s,a,d)

% 1.1 Time
t.ac = s/a;
t.dc = s/d;
t.fs = 0;

% 1.2 Distance needed for acceleration
x.ac = 0.5*a*t.ac^2;
x.dc = s*t.dc-0.5*d*t.dc^2;

% 2 Full speed
% 2.1 Distance traveled at full speed
x.fs = X - x.ac - x.dc;

% 2.2 Time at full speed
if x.fs > 0 % Full speed achieved
    full_speed = 1;
    % time at full speed
    t.fs = x.fs/s;
    ms = s;
else % Full speed not achieved
    full_speed = 0; x.fs = 0;
    t.dc = ( 2*X*a / (d*(a+d)) )^0.5;
    t.ac = ( 2*X*d / (a*(a+d)) )^0.5;
    x.ac = a*t.ac^2/2;
    x.dc = X-x.ac;
    ms = t.ac*a;
end

t.total = t.fs + t.ac + t.dc;
```

```
function [time] = TRF_generate_time_list(no_days,ct_dist,daily_cts,start_day)

ct = 0;
for day = start_day:no_days+3;
    no_ct_day = poissrnd(daily_cts);
    for hour = 1:24
        ctsperhour = fix(ct_dist(hour)*no_ct_day);
        for i = 1:ctsperhour
            base_time = ((day-1)*24 + hour-1)*3600; % In secs
            ct = ct + 1;
            time(ct) = base_time + ceil(random('unif',0,3600));
        end
    end
end

time = sort(time);
```

```
function TRF_init2(siono)

global BF EXEC TIME TRF

finame = ['C' num2str(EXEC.caso), 'TRF' num2str(TRF.PARAM.level) '.mat'];

if siono == 1

    % TRAFFIC PARAMETERS
    TRF.PARAM.no.pod = 1;
    TRF.PARAM.no.lines = 4;
    TRF.PARAM.no.ct_weights = 6;

    % EXPORT
    % ---------------------------------------------------------------------
    if TRF.PARAM.level == 40;
       TRF.PARAM.no.cts_exp_vs = 125;
    elseif TRF.PARAM.level == 60;
       TRF.PARAM.no.cts_exp_vs = 149; %154;
    elseif TRF.PARAM.level == 70;
       TRF.PARAM.no.cts_exp_vs = 176;
    end
    TRF.PARAM.perc_imp = 0.85;

    % IMPORT
    % ---------------------------------------------------------------------
    if TRF.PARAM.level == 40;
       TRF.PARAM.no.cts_imp = 91;  %91 gives 40%, 136 gives 60
    elseif TRF.PARAM.level == 60;
       TRF.PARAM.no.cts_imp = 136;
    elseif TRF.PARAM.level == 70;
       TRF.PARAM.no.cts_imp = 150;
    end
    TRF.PARAM.av_dwell = 3.4722;

    % DUAL
    TRF.PARAM.no.cts_dual = 4;
    TRF.PARAM.no.exp_limit = 180;
    TRF.PARAM.CT.no_groups = TRF.PARAM.no.ct_weights * TRF.PARAM.no.pod * TRF.PARAM.no.lines;

    for i = 1:100
       TRF.PARAM.rand(i)=random('unif',0.7,0.9);
    end
    % WEIGHT CONTAINER DISTRIBUTION
    % Weight cathegories
    no_wc = 10;
    [TRF.CT.pdf,TRF.CT.cdf,TRF.CT.top_xf]= pdfcdf(10);

    % CONTAINER TRAFFIC
    %----------------------------------------------------------------------

    qc_rythm = 25; % Movements/hour
    no_qc = 4;
```

```
ct_rythm = qc_rythm * no_qc;
T_blocks = 3;
block_cts_perday = ct_rythm/T_blocks;
block_cts_rate = 3600/10; % 30 cts per hour means one every 2 minutes

%exp = [0, 0, 0,0,0,0,57,76,60,79,89,64,160,149,143,206,258,276,156,60,0,0,0,0];
exp = [0,0,0,0,20,30,57,76,77,79,89,94,143,149,143,196,198,216,156,110,0,0,0,0];
imp = [0,0,0,0,0,0,226,302,238,238,266,144,240,149,95,91,115,123,52,20,0,0,0,0];
dual = [0,0,0,0,0,0,2,10,8,6,5,5,5,5,5,14,20,10,2,1,0,0,0,0];

CT_day = exp + imp + dual;
EXP = sum(exp); IMP = sum(imp); DUAL = sum(dual);
tot_cts = EXP + IMP + DUAL;
pexp = EXP/tot_cts;  pimp = IMP/tot_cts; pdual = DUAL/tot_cts;
%dist_exp = (exp + dual)/(EXP + DUAL);
dist_exp  = exp/EXP;
dist_imp  = imp/IMP;
dist_dual = dual/DUAL;

% LAND BUFFER
% ------------------------------------------------------------------------
% EXPORT CTS
exp_cts = TRF.PARAM.no.cts_exp_vs;
[exp_time] = TRF_generate_time_list(TIME.simulation.days,dist_exp,exp_cts,1);
no_ct=length(exp_time);
BF.land.exp.arrivals(1:no_ct) = exp_time;

% IMPORT CTS
imp_cts = TRF.PARAM.no.cts_imp; %fix(IMP/EXP*TRF.PARAM.no.cts_exp_vs*0.5);
% [imp_time] = TRF_generate_time_list(TIME.simulation.days,dist_imp,imp_cts,5);
% no_ct = length(imp_time);
% BF.land.imp.arrivals(1:no_ct) = imp_time;

% DUAL CTS
dual_cts = TRF.PARAM.no.cts_dual; %fix(DUAL/EXP*TRF.PARAM.no.cts_exp_vs*0.5);
% [dual_time] = TRF_generate_time_list(TIME.simulation.days,dist_dual,dual_cts,1);
% no_ct=length(dual_time);
% BF.land.dual.arrivals(1:no_ct) = dual_time;
ric = imp_cts/(imp_cts+dual_cts);
rdc = dual_cts/(imp_cts+dual_cts);

% SEA BUFFER
% ------------------------------------------------------------------------
% LIST OF VESSELS
% Parameters of the weibull distribution
TRF.PARAM.K = 20;
TRF.PARAM.L = (imp_cts+dual_cts+5)/gamma(1+1/TRF.PARAM.K); %L =
TRF.PARAM.no.cts_exp_vs/gamma(1+1/K);

ct = 0; ict = 0; ect = 0; dct = 0;
lict = 0; ldct = 0;

for day = 1:TIME.simulation.days + 2
   TRF.VS(day).arrival = ceil(3600* ((day-1)*24 + random('unif',8,19)));
```

```
   TRF.VS(day).exit = 0;
   TRF.VS(day).active = 0;

   %keyboard
   tot_cts = fix(wblrnd(TRF.PARAM.L,TRF.PARAM.K));
   imp_cts(day) = fix(ric*tot_cts);
   dual_cts(day) = tot_cts-imp_cts(day);
   TRF.VS(day).CT.IMP = tot_cts;
   TRF.VS(day).CT.EXP = 0;
   TRF.VS(day).CT.loaded = 0;
   TRF.VS(day).line = rem(day,TRF.PARAM.no.lines)+1;
   base_time = TRF.VS(day).arrival + 30*60;

   % 1) Generate IMP SEA ARRIVALS
   %keyboard
   for i = 1:tot_cts
      ct = ct+1;
      delt = poissrnd(block_cts_rate)/1.1;
      arrival(ct) = base_time + delt;
      BF.sea.imp.arrivals(ct) = arrival(ct);
      %keyboard
      if ct > 1
         if BF.sea.imp.arrivals(ct)< BF.sea.imp.arrivals(ct-1)
            keyboard
         end
      end
      s = random('unif',0,1);
      if s <= ric
         ict = ict +1;
         idwell(ict) = exprnd(TRF.PARAM.av_dwell*3600*24)+ 3600*4;
         if idwell(ict) == 0
            keyboard
         end
         impleave(ict) = arrival(ct) + idwell(ict);
      else
         dct = dct +1;
         ddwell(dct)=  exprnd(TRF.PARAM.av_dwell*3600*24) + 3600*4;
         dualeave(dct) = arrival(ct) + ddwell(dct);
      end
      base_time = base_time + delt;
   end

   % 1) Generate EXP SEA TRIGGER
   ect = ect+1;
   delt = ceil(poissrnd(block_cts_rate));
   BF.sea.exp.trigger(ect) = base_time + delt;
   base_time = base_time + delt;
end

BF.sea.imp.dwell = idwell;

% Generate land imp arrivals
impleave = sort(impleave);     lict = length(impleave);
BF.land.imp.arrivals(1:lict) = impleave;
```

```
    % Generate land dual arrivals
    dualeave = sort(dualeave);     ldct = length(dualeave);
    BF.land.dual.arrivals(1:ldct) = dualeave;

    figure; plot(BF.land.imp.arrivals/3600/24); hold on;
    plot(BF.sea.imp.arrivals/3600/24);
    plot(BF.land.exp.arrivals/3600/24);
    plot(BF.land.dual.arrivals/3600/24,'m');
    view(90,-90)
    %keyboard
    grid on
    %plot_trf()

    % Now assign weights and all that to the CTs
    GROUP_init()
    % BF sea import
    %keyboard
    for i = 1:length(BF.sea.imp.arrivals)
       [BF.sea.imp.weight(i),BF.sea.imp.w_class(i)] = CT_weight(random('unif',TRF.CT.cdf(1),1));
       pod = fix(random('unif',1,TRF.PARAM.no.pod+1));
       line = fix(random('unif',1,TRF.PARAM.no.lines+1));
       BF.sea.imp.group(i) = CT_group_search(pod, line, BF.sea.imp.w_class(i));
    end
    %keyboard
    % BF land exp
    for i = 1:length(BF.land.exp.arrivals)
       [BF.land.exp.weight(i),BF.land.exp.w_class(i)] = CT_weight(random('unif',TRF.CT.cdf(1),1));
       pod = fix(random('unif',1,TRF.PARAM.no.pod+1));
       line = fix(random('unif',1,TRF.PARAM.no.lines+1));
       BF.land.exp.group(i) = CT_group_search(pod, line, BF.land.exp.w_class(i));
    end

    % BF land dual
    for i = 1:length(BF.land.exp.arrivals)
       [BF.land.dual.weight(i),BF.land.dual.w_class(i)] = CT_weight(random('unif',TRF.CT.cdf(1),1));
       pod = fix(random('unif',1,TRF.PARAM.no.pod+1));
       line = fix(random('unif',1,TRF.PARAM.no.lines+1));
       BF.land.dual.group(i) = CT_group_search(pod, line, BF.land.dual.w_class(i));
    end

    disp('TRF initialized')

    save ( finame ,'BF','TRF')
    disp(['There are ' num2str(ict) '/' num2str(ict) 'IMP/EXP cts'])
    pause(1)
else
    load(finame); %keyboard
    disp(['IMP/EXP CTs generated (' num2str(length(BF.sea.imp.dwell)) '/' num2str(length(BF.land.exp.arrivals)) ')'])
    pause(1)
end
```

```
function [newvect] = vect_insert(oldvect, elem, pos)

newvect = -1000;
%keyboard
if length(oldvect) == 0
   newvect = elem;
else
   if pos > length(oldvect)+1
      disp('Error in double vector insert')
   else
      izq = oldvect(1:pos-1);
      dch = oldvect(pos:end);
      newvect = [izq elem dch];
   end
end

if newvect == -1000
   keyboard
end
```

```
function [newvect] = vect_insert_char(oldvect, elem, pos)


if isempty(oldvect) == 1
   newvect{1} = elem;
else
   %keyboard
   if pos > length(oldvect)+1
      disp('Error ')
   else
      %keyboard
      izq = oldvect(1:pos-1);
      dch = oldvect(pos:end);
      newvect = [izq elem dch];
   end
end
```

```
function vectarrow(p0,p1)
%Arrowline 3-D vector plot.
%   vectarrow(p0,p1) plots a line vector with arrow pointing from point p0
%   to point p1. The function can plot both 2D and 3D vector with arrow
%   depending on the dimension of the input
%
%   Example:
%       3D vector
%       p0 = [1 2 3];   % Coordinate of the first point p0
%       p1 = [4 5 6];   % Coordinate of the second point p1
%       vectarrow(p0,p1)
%
%       2D vector
%       p0 = [1 2];     % Coordinate of the first point p0
%       p1 = [4 5];     % Coordinate of the second point p1
%       vectarrow(p0,p1)
%
%   See also Vectline

%   Rentian Xiong 4-18-05
%   $Revision: 1.0

  if max(size(p0))==3
     if max(size(p1))==3
        x0 = p0(1);
        y0 = p0(2);
        z0 = p0(3);
        x1 = p1(1);
        y1 = p1(2);
        z1 = p1(3);
        plot3([x0;x1],[y0;y1],[z0;z1]);   % Draw a line between p0 and p1

        p = p1-p0;
        alpha = 0.1;  % Size of arrow head relative to the length of the vector
        beta = 0.1;  % Width of the base of the arrow head relative to the length

        hu = [x1-alpha*(p(1)+beta*(p(2)+eps)); x1; x1-alpha*(p(1)-beta*(p(2)+eps))];
        hv = [y1-alpha*(p(2)-beta*(p(1)+eps)); y1; y1-alpha*(p(2)+beta*(p(1)+eps))];
        hw = [z1-alpha*p(3);z1;z1-alpha*p(3)];

        hold on
        plot3(hu(:),hv(:),hw(:))  % Plot arrow head
        grid on
        xlabel('x')
        ylabel('y')
        zlabel('z')
        hold off
     else
        error('p0 and p1 must have the same dimension')
     end
  elseif max(size(p0))==2
     if max(size(p1))==2
        x0 = p0(1);
```

```
        y0 = p0(2);
        x1 = p1(1);
        y1 = p1(2);
        plot([x0;x1],[y0;y1]);   % Draw a line between p0 and p1

        p = p1-p0;
        alpha = 0.1;  % Size of arrow head relative to the length of the vector
        beta = 0.1;  % Width of the base of the arrow head relative to the length

        hu = [x1-alpha*(p(1)+beta*(p(2)+eps)); x1; x1-alpha*(p(1)-beta*(p(2)+eps))];
        hv = [y1-alpha*(p(2)-beta*(p(1)+eps)); y1; y1-alpha*(p(2)+beta*(p(1)+eps))];

        hold on
        plot(hu(:),hv(:))  % Plot arrow head
        grid on
        xlabel('x')
        ylabel('y')
        hold off
    else
        error('p0 and p1 must have the same dimension')
    end
else
    error('this function only accepts 2D or 3D vector')
end
```

```
function VS_check_complete(ct,timeleft)

% This function checks whether we ended up placing cts in the vessel

global CT TIME TRF

if strcmp(CT(ct).id,'EXP') == 1
   if CT(ct).position.vs > 0
      vs = CT(ct).position.vs;
      TRF.VS(vs).CT.loaded = TRF.VS(vs).CT.loaded +1;
      if TRF.VS(vs).CT.loaded == TRF.VS(vs).CT.EXP
         %keyboard
         TRF.VS(vs).exit = TIME.t+timeleft;
         TRF.VS(vs).active = 0;
      end
   else
      keyboard
   end
end

% Now check the cts that are not prioritary anymore
%keyboard
```