UNIVERSITAT POLITÈCNICA DE CATALUNYA
DEPARTAMENT DE LLENGUATGES I SISTEMES INFORMÀTICS
ARTIFICIAL INTELLIGENCE PhD PROGRAM

DOCTORAL THESIS

A Multiagent Approach to Qualitative Navigation in Robotics

Submitted by
Dídac Busquets i Font
to obtain the degree of
Doctor in Computer Science

Supervisors:
Dr. Ramon López de Màntaras i Badia (IIIA-CSIC)
Dr. Carles Sierra i Garcia (IIIA-CSIC)

Tutor:
Dr. Mario Martín i Muñoz (LSI-UPC)

Departament de Llenguatges i Sistemes Informàtics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Institut d'Investigació en Intel·ligència Artificial

Consejo Superior de Investigaciones Científicas

Contents

List of Figures

List of Tables

Acknowledgments

Since I started my PhD adventure, four years ago, I have received help, advice and support from many people, to all of whom I am really thankful.

First of all, I am really grateful to my supervisors, Ramon López de Màntaras and Carles Sierra, without whom I would have never finished this thesis. Actually, I would have not even started it!

I am also thankful to the Institut d'Investigació en Intel·ligència Artificial, my second home (or was it the first?) during these last four years, to all its staff members and specially the rest of PhD students with whom we have had great times.

I also thank Thomas Dietterich for his help and comments, and also for being such an excellent host in my visits to Corvallis.

Jose Sánchez and Madhur Ambastha deserve special thanks for their help on the development of the vision system and the genetic algorithm approach, respectively.

I am also really grateful to my family, who has supported and helped me (and also homed and fed) during these years.

And last, but not least, thanks to all of my friends, to whom I have tried to explain what has kept me so busy during this time. Although they have not helped on the technical issues of the thesis, they have always been there to remind me that a PhD is not incompatible with going out and having fun (some of them reminded it quite often!). To name some of them (in friendship antiquity order): Roger, Fèlix, Cris, Meri(bad), Glòria, Sandra, Albert, Lluc (thanks for pointing out the Czech origin of the word "robot"), Meri(jem), Dic... Thank you all.

This research has been partially supported by the Spanish MCYT project ARGOS (DPI2000-1352-C02-02), CICYT project number TAP97-1209, CIRIT project CeRTAP and the US-Spain Joint Commission for Scientific and Technological Cooperation. During the four years of my research I have enjoyed a CIRIT doctoral scholarship (2000FI-00191) from Generalitat de Catalunya.

Abstract

Navigation in unknown unstructured environments is still a difficult open problem in the field of robotics. In this PhD thesis we present a novel approach for robot navigation based on the combination of landmark-based navigation, fuzzy distances and angles representation and multiagent coordination based on a bidding mechanism. The objective has been to have a robust navigation system with orientation sense for unstructured environments using visual information.

To achieve such objective we have focused our efforts on two main threads: navigation and mapping methods, and control architectures for autonomous robots.

Regarding the navigation and mapping task, we have extended the work presented by Prescott, so that it can be used with fuzzy information about the locations of landmarks in the environment. Together with this extension, we have also developed methods to compute diverting targets, needed by the robot when it gets blocked.

Regarding the control architecture, we have proposed a general architecture that uses a bidding mechanism to coordinate a group of systems that control the robot. This mechanism can be used at different levels of the control architecture. In our case, we have used it to coordinate the three systems of the robot (Navigation, Pilot and Vision systems) and also to coordinate the agents that compose the Navigation system itself. Using this bidding mechanism the action actually being executed by the robot is the most valued one at each point in time, so, given that the agents bid rationally, the dynamics of the biddings would lead the robot to execute the necessary actions in order to reach a given target. The advantage of using such mechanism is that there is no need to create a hierarchy, such in the subsumption architecture, but it is dynamically changing depending on the specific situation of the robot and the characteristics of the environment.

We have obtained successful results, both on simulation and on real experimentation, showing that the mapping system is capable of building a map of an unknown environment and use this information to move the robot from a starting point to a given target. The experimentation also showed that the bidding mechanism we designed for controlling the robot produces the overall behavior of executing the proper action at each moment in order to reach the target.

Resum

La navegació en entorns desconeguts no estructurats és encara un problema obert en el camp de la robòtica. En aquesta tesi presentem una aproximació per a la navegació de robots basada en la combinació de navegació basada en landmarks, representació fuzzy d'angles i distàncies i una coordinació multiagent basada en un mecanisme de dites. L'objectiu de la tesi ha sigut desenvolupar un sistema de navegació robust amb sentit de l'orientació per a entorns no estructurats usant informació visual.

Per tal d'assolir aquest objectiu, hem centrat els nostres esforços en dues línies d'investigació: mètodes de navegació i construcció de mapes, i arquitectures de control per a robots autònoms.

Pel que fa als mètodes de navegació i construcció de mapes, hem extès el treball presentat per Prescott per tal que es pugui utilitzar amb informació fuzzy sobre la localitazció dels landmarks. A part d'aquesta extensió, també hem desenvolupat mètodes per a calcular objectius alternatius, necessaris quan el robot troba el camí bloquejat.

Pel que fa a l'arquitectura de control, hem proposat una arquitectura general que utilitza un mecanisme de dites per a coordinar un grup de sistemes que controlen el robot. Aquest mecanisme pot ser usat en diferents nivells de l'arquitectura. En el nostre cas l'hem usat per a coordinar els tres sistemes del robot (Navegació, Pilot i Visió), i també per a coordinar els agents que composen el sistema de Navegació. Usant aquest mecanisme de dites, l'acció que executa el robot és la més ben valorada en cada instant. D'aquesta manera, i si els agents fan les dites d'una manera racional, la dinàmica de les dites porta el robot a executar les accions necessàries per tal d'arribar a l'objectiu indicat. L'avantatge d'utilitzar aquest mecanisme és que no cal imposar una jerarquia entre els sistemes, com passa en l'arquitectura de subsumpció, si no que aquesta jerarquia canvia dinàmicament, depenent de la situació en què es troba el robot i les característiques de l'entorn.

Hem obtingut resultats satisfactoris, tant en simulació com en experimentació amb un robot real, que confirmen que el sistema de navegació és capaç de construir un mapa d'un entorn desconegut i utlitzar-lo per a moure el robot d'una posició inicial a un objectiu donat. L'experimentació també ha mostrat que el sistema de coordinació basat en dites que hem dissenyat produeix el comportament global d'executar les accions necessàries en cada instant per tal d'arribar a l'objectiu.

Chapter 1

Introduction

robot

From translation of 1920 play "R.U.R." ("Rossum's Universal Robots"), by Karel Čapek (1890-1938), from Czech *robota* "forced labor, drudgery", from *robotiti* "to work, drudge", from Old Church Slavonic *rabota* "servitude", from *rabu* "slave", from a Slavic stem related to German *arbeit* "work".

1.1 Overview and Motivation

Since the late 1960s, with the development of SRI's Shakey robot [54], artificial intelligence (AI) and mobile robotics have been closely related. A mobile robot must be autonomous, deal with uncertainty, plan and decide what to do, react to unpredicted situations, that is, it has to overcome really hard problems if we want it to act in an intelligent and autonomous way. Thus, mobile robots pose one of the biggest challenges for AI.

Although impressive successes have been obtained since Shakey, it cannot still be said that the objective of having truly autonomous robots has been achieved. One of the fields in which there is still much to do is on mobile robotics for unknown environments.

Robotic systems for navigating through unknown environments are a focus of research in many application areas including, among others, spacecraft (rovers for Mars and the Moon) and search and rescue robotics. These systems have to perform very different tasks, from looking for rocks, picking them up and analyzing them, to assessing damages or looking for survivors after a natural disaster or accident has happened. However, they all share two key characteristics: first, they have to achieve their goals ***autonomously***, and second, they have to ***navigate*** in unknown environments.

The first key point in these applications, ***autonomy***, arises from the impossibility of always having a human operator controlling the robotic system. Although the ideal situation would be to have an expert operator controlling the robot, the necessary conditions cannot always be met. These conditions are usually related to the communication between the operator and the robot. In many situations it is very difficult to guarantee that the communication link will be robust, in terms of speed and availability. A clear

example is found on planetary exploration missions. A major problem in such missions is the distance between the robot and the control station (usually located on the Earth); the time of sending an order to the robot and having it executed can be in the order of minutes. In the case a fast reaction were needed (changing the trajectory of the robot, selecting a new scientific target that might be more relevant to the mission, etc.), this time would not be acceptable at all. Another disadvantage of depending on external agents (be it a human or any other device — e.g. a GPS device for localisation) is that the robot can get blocked if any of these agents providing basic information for accomplishing the task crashes. This would leave the robot with no means to continue with its mission. Therefore, all the decisions should be taken on-board, without having to exchange commands or information with any external agent, or at least, make this exchange minimal, such as sending only information about task initialisation (e.g. target selection, task description, etc.).

The other important point for such applications is ***navigation***. The robot must be able to start in an unknown location and navigate to a desired target. Navigation in unknown unstructured environments is still a difficult open problem in the field of robotics. The first difficulty of such an environment is that there is no a priori knowledge about it, and therefore a map can only be built while exploring. Secondly, unstructured environments are characterized, precisely, by the lack of structure among the different objects of the world. This is usually the case for outdoor environments. On the other hand, in structured environments, such as offices, buildings, etc. many suppositions about their structure can be done. For instance, walls and corridors are straight, they are usually orthogonal, most of the doors have the same size, etc. These characteristics are very helpful when building a map of the environment, as its structure can be inferred without the need of sensing the whole environment. Contrarily, in unstructured environments such suppositions do not hold, so the robot can only rely on the information it is able to gather from its sensors. This makes the task of map building and navigating even more difficult.

This research work is part of a larger robotics project. Another partner (IRI[1]) in the project is building a six legged robot with on board cameras for outdoor landmark recognition. The goal of the project is to have a completely autonomous robot capable of navigating in outdoor unknown environments. A human operator will select a target using the visual information received from the robot's camera, and the robot will have to reach it without any intervention of the operator. The robot could also have an image or description of the target, so the human intervention would not even be needed for selecting the target. A first milestone for achieving the final goal of the project is to develop a navigation system for indoor unknown unstructured environments for a wheeled robot. Moreover, the environments of this first stage are composed of easily recognizable landmarks, since the vision system for outdoors is not yet available.

We propose a robot architecture to accomplish this first milestone. The approach used and the results obtained are the subject of this thesis. The robot architecture is composed of three systems: the *Pilot* system, the *Vision* system and the *Navigation* system. Each system competes for the two available resources: motion control (direction of movement) and camera control (direction of gaze). The three systems have the

[1]Institut de Robòtica i Informàtica Industrial, http://www.iri.csic.es

following responsibilities. The Pilot is responsible for all motions of the robot. It selects these motions in order to carry out commands from the Navigation system and, independently, to avoid obstacles. The Vision system is responsible for identifying and tracking landmarks (including the target landmark). Finally, the Navigation system is responsible for choosing higher-level decisions in order to move the robot to a specified target. This requires requesting the Vision system to identify and track landmarks (in order to build a map of the environment) and requesting the Pilot to move the robot in various directions in order to reach the goal position or some intermediate target.

From the brief description of the robot architecture given above, it can be observed that the three systems must ***cooperate*** and ***compete***. They must cooperate because they need one another in order to achieve the overall task of reaching the target position. But at the same time they are competing for motion and camera control.

The Navigation system is implemented as a multiagent system, where each agent is competent in a specific task. Depending on its responsibilities and the information received from other agents, each agent proposes which action the Navigation system should take. Again, we find that the agents must cooperate, since an isolated agent is not capable of moving the robot to the target, but they also compete, because different agents want to perform different actions.

The problem of cooperation and competition between different agents is very common in robotics, and *Behavior-based Robotics* [3] addresses exactly this issue. This approach to robotic systems deals with coordinating, or arbitrating, different behaviors sending requests for actions, usually incompatible with each other, to a robot. The role of the coordination is to select a single action command that will be sent to the robot. When this action is a selection of one of the behaviors' requests, we talk about *competitive* coordination, whereas if the action is a mix of several behaviors' requests, we talk about *cooperative* coordination. In our architecture, each agent plays the role of a behavior, and there is an additional agent playing the role of coordinator.

For both the overall robot system and the Navigation system, we propose the use of a new competitive coordination system based on a *bidding mechanism*. In the overall robot system, the Navigation and the Pilot systems generate bids for the services offered by the Pilot and Vision systems. These services are to move the robot toward a given direction, and to move the camera and identify the landmarks found on its view-field, respectively. The service actually executed by each system depends on the winning bid at each point in time. Similarly, in the Navigation system, each agent bids for the action it wants the robot to perform. These bids are sent to a special agent that gathers all bids and determines the winning action. The selected action is then sent as the Navigation system's bid for the services of the Vision and Pilot systems.

The architecture just described above is actually an instantiation of a general coordination architecture we have developed. In this architecture there are two types of systems: *executive systems* and *deliberative systems*. Executive systems have access to the sensors and actuators of the robot. These systems offer services for using the actuators to the rest of the systems (either executive or deliberative) and also provide information gathered from the sensors. On the other hand, deliberative systems take higher-level decisions and require the services offered by the executive systems in order to carry out the task assigned to the robot. Although we differentiate between these

two types of systems, the architecture is not hierarchical, and coordination is made at a single level involving all the systems. This coordination is based on a simple mechanism: *bidding*. Deliberative systems always bid for the services offered by executive systems, since this is the only way to have their decisions executed. Executive systems that only offer services do not bid. However, those executive systems that require services from any executive system (including themselves) must also bid for them. The systems bid according to the internal expected utility associated to the provisioning of a service. A coordinator receives these biddings and decides which service each of the executive systems has to engage on. In the instantiation for our navigation problem, there is a deliberative system, the Navigation system, one executive system that bids, the Pilot system, and one executive system that only offers services, the Vision system.

To navigate in an unknown environment, the robot must build a map. Existing approaches assume that an appropriately detailed and accurate metric map can be obtained through sensing the environment. However, most of these approaches rely on odometry sensors, which can be very imprecise, due to the wheels or legs slipping, and lead to many errors that grow as the robot moves.

Our approach considers using only visual information. The advantage of using visual information is that it is independent of any past action the robot may have performed, which is not the case for odometry. The robot must be equipped with a robust vision system capable of recognising landmarks, and use them for mapping and navigation tasks. The specific scenario that we are studying assumes that there is a target landmark that the robot is able to recognize visually. The target is visible from the robot's initial location (so that the human operator can select it), but it may subsequently be occluded by intervening objects. The challenge for the robot is to acquire enough information about the environment (locations of other landmarks and obstacles) so that it can move along a path from the starting location to the target.

But even vision-based navigation approaches assume unrealistically accurate distance and direction information between the robot and the landmarks. We propose a fuzzy set based approach that assumes only very rough vision estimation of the distances and, therefore, does not rely on any localisation device.

The main goal of our research is to design a robust vision-based navigation system for unknown unstructured environments. In particular, we want to provide the robot with orientation sense, similar to that found in humans or animals. The rationale of the orientation sense is that the robot does not need to know the exact route from its starting point to the target's location, but it uses landmarks as references to find its way to the target. To make a parallel with humans, when giving directions for going somewhere in our city, no one gives exact distances, turning angles, etc., but gives approximate distances, and more important, reference points (distinctive places such as buildings, squares, etc.) that help us getting to the destination. In our approach, this orientation sense is realized by the use of *landmark-based navigation*, *topological mapping* and *qualitative computation* of landmark locations. We give a brief definition of each of these three concepts:

- *Landmark-based navigation*: A landmark is a visually salient object of the environment the robot is able to identify. Other navigation approaches that do not use vision systems define a landmark as a set of features the robot can detect with its

sensors (usually sonar or laser readings). As the robot explores the environment, it stores the detected landmarks on a map. When the robot needs to locate itself on the map, it can do it by matching the detected landmarks with the landmarks already stored on the map. This approach avoids requiring odometry as the main information source for navigating and building maps.

- *Topological mapping*: this approach to map building has a close relationship with landmark-based navigation. A topological map is usually a graph, where nodes represent landmarks and arcs represent paths or motion instructions for going from one landmark to another. The advantage of this approach is that there is no need for building accurate geometric maps. Storing the topological relationships among the landmarks in the environment is enough.

- *Qualitative computation*: we use the term "qualitative" in the sense that we do not need to work with exact distance or angle information; we can deal with some imprecision about it. More specifically, we deal with it by means of fuzzy numbers and fuzzy arithmetic. Thus, when we talk about qualitatively computing the location of a landmark, it means that we are taking into account the possible imprecision about its location.

Our map representation, however, is slightly different from the one given above. The map is a labeled graph whose nodes, instead of representing isolated landmarks, represent triangular shaped regions delimited by groups of three non-collinear landmarks, and whose arcs represent the adjacency between regions, that is, if two regions share two landmarks, the corresponding nodes are connected by an arc. The arcs of the graph are labeled with costs that reflect the easiness of the path between the two corresponding regions. A blocked path would have an infinite cost, whereas a flat, hard paved path would have a cost close to zero. Since the map is not given, but built while the robot moves, these costs can only be assigned after the robot has moved (or tried to move) along the path connecting the two regions. Although the adjacency of nodes in our graph also represents adjacency of topological places, the arcs contain only cost information, not instructions on how to get from one place to another. But, actually, this information is not missing, it is implicit in the adjacency of regions. Given that two nodes are adjacent only if their regions share two landmarks, it is clear that to go from one region to another the robot has to cross the edge formed by the two common landmarks, unless there is a long obstacle blocking this path.

Although this topological map would be sufficient for carrying out navigational tasks, we also provide the robot with the capability of storing the spatial relationships among landmarks. To realize this capability, we have extended Prescott's beta-coefficients system [55]. Prescott's model stores the relationships among the landmarks in the environment. The location of a landmark is encoded based on the relative locations (headings and distances) of three other landmarks. This relationship is unique and invariant to viewpoint. Once this relationship has been stored, the location of each landmark can be computed from the locations of the three landmarks encoding it, no matter where the robot is located, as long as the robot can compute the heading and distance to each of the three landmarks. As the robot explores the environment, it stores the relationships among the landmarks it sees. This creates a network of relationships among

the landmarks in the environment. If this network is sufficiently-richly connected, then it provides a computational map of the environment. Given the headings and distances to a subset of currently-visible landmarks, the network allows us to compute the locations of all of the remaining landmarks, even if they are currently not visible from the robot. Prescott assumed that the robot could have the exact distances and headings to the landmarks, but as we mentioned previously, we need to deal with the imprecision of the real world. To deal with it, we have extended the model using fuzzy numbers and fuzzy arithmetic. With this extension, if the target is ever lost during the navigation, the robot will compute its location with respect to a set of previously seen landmarks whose spatial relation with the target is qualitatively computed, both in terms of fuzzy distance and direction.

We have implemented the overall architecture and the Navigation system and first tested it over a simulator. After obtaining promising results on simulation, we have implemented the algorithm on a wheeled robot and tested it on real environments.

Although the tuning of our system was carried out through the experimentation with the real robot, we also employed simulation to apply machine learning techniques in order to improve the performance of the system. In particular, we have applied Reinforcement Learning and Genetic Algorithms. We have used Reinforcement Learning to have the system learn to use the camera only when appropriate. The camera is a very expensive resource, and it has also a very high demand (the Pilot and several agents compete for its control). Since manual tuning of the parameters controlling the agents' behaviors is very difficult, and the problem we are trying to solve is a quantitative trade-off, Reinforcement Learning is found to be the most appropriate technique to use, as its main goal is to maximize expected reward. We have obtained good results that show the feasibility of applying Reinforcement Learning to improve our system. Nonetheless, we still need further experimentation and tuning of the learning algorithm, in order to integrate the learned policy into the multiagent system. In parallel, we have used a Genetic Algorithm to tune the different parameters of the agents. The tuning of these parameters cannot be done manually, neither can it be done using Reinforcement Learning. Therefore, we have chosen to use a genetic algorithm approach.

1.2 Contributions

The objective of the research carried out during the completion of this PhD thesis has been to accomplish the first milestone of the above mentioned project, that is, developing a navigation system for indoor unknown unstructured environments for a wheeled robot. More precisely, we have focused on the Navigation system and on the overall robot architecture. However, we have also had to implement simple versions of the Pilot and Vision systems in order to realize and test the Navigation system.

As it may have already been noticed, this work has two main research threads: the ***control architecture*** and the ***mapping and navigation method***.

Regarding the ***control architecture***, we have proposed a general coordination architecture that uses a bidding mechanism for coordinating a group of systems (and agents) that control a robot. This mechanism can be used at different levels of the control architecture. In our case, we have used it to coordinate two of the systems of the robot

(Navigation and Pilot systems) and also to coordinate the agents that compose the Navigation system itself. Moreover, the multiagent view of the Navigation system could also be applied to other systems, having a multiagent Pilot or a multiagent Vision system. Using this bidding mechanism, the action actually being executed by the robot is the most urgent one at each point in time, and thus, if the agents bid rationally, the dynamics of the biddings would lead the robot to execute the necessary actions in order to reach a given target. An advantage of using such mechanism is that there is no need to create a hierarchy, such as in the subsumption architecture, but it is dynamically changing depending on the specific situation of the robot and the characteristics of the environment. A second advantage is that its modular view conforms an extensible architecture. To extend this architecture with a new capability we would just have to plug in a new system (or agent).

Regarding the ***mapping and navigation method***, we have extended the work presented by Prescott [55], so that it can be used with fuzzy information about the locations of landmarks in the environment. This is of great importance when working with real robots, as it is impossible to avoid dealing with the imprecision of real world environments. Together with this extension, we have also developed methods that permit computing diverting targets, needed by the robot when there is no clear path to the goal.

1.3 Publications

The publications related with this research that have been published as journal articles or in conference proceedings are the following:

- C. Sierra, R. López de Màntaras and D. Busquets. Multiagent bidding mechanisms for robot qualitative navigation. *Lecture Notes in Computer Science (Proceedings ATAL'00), vol. 1986*, pages 198–212, Springer, Verlag, 2001.

- D. Busquets, R. López de Màntaras and C. Sierra. A robust MAS coordination mechanism for action selection. *Proceedings of 2001 AAAI Spring Symposium, Stanford, CA. Robust Autonomy Serie*, pages 38–40, 2001.

- D. Busquets, R. López de Màntaras, C. Sierra and T.G. Dietterich. Reinforcement Learning for Landmark-based Robot Navigation. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 841–842. ACM press, 2002.

- T.G. Dietterich, D. Busquets, R. López de Màntaras and C. Sierra. Action Refinement in Reinforcement Learning by Probability Smoothing. *Proceedings of the 19th International Conference on Machine Learning (ICML'02)*, pages 107–114, 2002.

- D. Busquets, T.G. Dietterich, R. López de Màntaras and C. Sierra. A multi-agent architecture integrating learning and fuzzy techniques for landmark-based robot navigation. *Topics in Artificial Intelligence. Lecture Notes in Artificial Intelligence (Proceedings of CCIA'02), vol. 2504*, pages 269–281, Springer, Verlag, 2002.

- D. Busquets, C. Sierra and R. López de Màntaras. A multi-agent approach to fuzzy landmark-based navigation. *Journal of Multiple-Valued Logic and Soft Computing*, Old City Publishing (In press).

- D. Busquets, C. Sierra and R. López de Màntaras. A Multi-agent approach to qualitative landmark-based navigation. *Autonomous Robots*, Kluwer Academic Publishers (In press).

1.4 Structure of the Thesis

This PhD thesis report is organized as follows:

Chapter 1. Introduction

This chapter gives an overview of this PhD thesis, its motivations, objectives and its main contributions. It also gives a list of the publications originated from the research carried out during the completion of the thesis.

Chapter 2. Related work and state-of-the-art

This chapter is devoted to relevant related work and state-of-the-art on the field of autonomous robots for unknown unstructured environments. The relevant work has been divided in two parts, one for each main thread of research of the thesis: control architectures, and mapping and navigation methods. The relevant work concerning control architectures gives an overview of the different approaches on autonomous robots control, focusing on the behavior-based approach. Regarding the mapping methods, we review and compare the two main approaches for map building, the metric one and the topological one. A comparison between two different localisation approaches (landmark-based localisation and model matching) is also given.

Chapter 3. Mapping and Navigation

In this chapter we firstly describe Prescott's model for storing spatial relationships among the landmarks of the environment. After that, we describe how we have extended this model for dealing with imprecise information about the location of the landmarks. We also present the algorithm for building a topological map of the environment and how it is used to compute diverting targets, needed by the robot when it is blocked.

Chapter 4. The Robot Architecture

In this chapter a general coordination architecture based on a bidding mechanism is presented. We also present the particular instantiation of the general architecture that we have used to solve the navigation problem. A detailed description of the multiagent Navigation system is also given in this chapter.

Chapter 5. Simulation Results

In this chapter the results of the simulated experiments are presented. These experiments include the testing of our architecture and the application of Machine Learning techniques in order to improve the performance of the system. In particular, we present the application of Reinforcement Learning, which we have used to make the system learn how to appropriately use the camera, and an application of Genetic Algorithms, used to tune some of the parameters of the agents of the Navigation system.

Chapter 6. Real Experiments

This chapter is devoted to present the results of the experiments on real environments with a real robot. Firstly, the wheeled robot platform and a simple vision system used for the real environments experiments are described. Then, we describe the different scenarios in which the experiments have been carried out. Finally, the results of the experimentation in such scenarios are given and discussed.

Chapter 7. Conclusions and Future Work

In this chapter, we summarize the main contributions of the thesis, and point out some open problems and future research perspectives that we plan to tackle in the near future.

Chapter 2

Related Work and State-of-the-art

In this chapter we review relevant related work and the state-of-the-art on the field of autonomous robotics. We have divided it in two sections, one for each main thread of our research: Control Architectures and Mapping and Navigation.

2.1 Control Architectures

A mobile robot working in unknown environments has to be able to perceive the world, reason about it, and act consequently in order to achieve its goals. The way in which this process is done is defined by the robot's control architecture. Many approaches for control architectures have been developed, and there also exist many definitions of what a control architecture is:

"Robotic architecture is the discipline devoted to the design of highly specific and individual robots from a collection of common software building blocks." – Adaptation of Stone's [62] definition of computer architecture.

" [an architecture refers to] the abstract design of a class of agents: the set of structural components in which perception, reasoning, and action occur; the specific functionality and interface of each component, and the interconnection topology between components." – Hayes-Roth [30].

"An architecture provides a principled way of organizing a control system. However, in addition to providing structure, it imposes constraints on the way the control problem can be solved." – Mataric [48].

"An architecture is a description of how a system is constructed from basic components and how these components fit together to form the whole." – James Albus, at the 1995 AAAI Spring Symposium.

The main difference between the architectures proposed in the past years relies on whether they are more deliberative or more reactive. Figure 2.1 depicts the spectrum of control architectures.

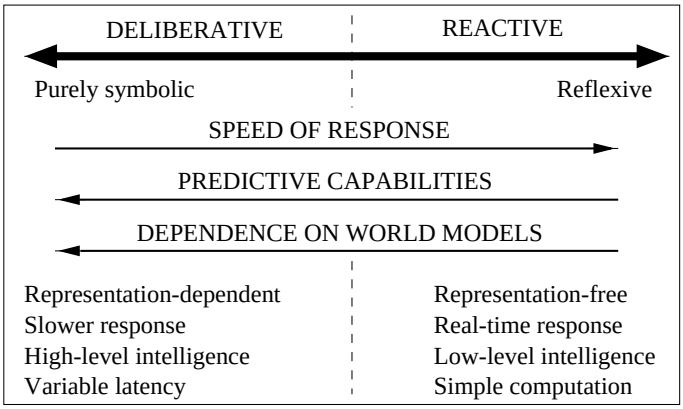


Figure 2.1: Control architectures' spectrum

In this section we give an overview (characteristics, advantages and disadvantages) of the three main approaches: purely deliberative or *hierarchical architectures*, purely reactive or *behavior-based architectures*, and *hybrid architectures*, which combine both previous methods.

2.1.1 Hierarchical Architectures

Hierarchical architectures, also named deliberative control architectures, were used for many years since the first robots began to be built. Examples of such architectures and robots are SRI's Shakey [54], Stanford's CART [50], NASA's Nasrem system [42] and Isik's ISAM [32], among others. These architectures are based on a top-down philosophy, following a *sense-plan-act* model (see Figure 2.2). The control problem is decomposed into a set of modules, sequentially organized: first the perception module gets the sensory information, which is passed to the modeling module that updates an internal model of the environment; after that, planning is done using this internal model, and finally the execution module implements the solution with the appropriate commands for the actuators.

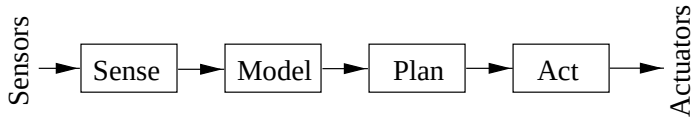


Figure 2.2: Sense-plan-act model

This model works very well when the environment in which the robot is working can be tailored to the task to be performed (e.g. industrial robots in factories, with magnetic beacons, marked paths, etc.). However, when the task is to be performed in an unknown, unpredictable, noisy environment, they fail to succeed, as the planning is

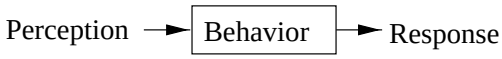


Figure 2.3: Single behavior diagram

usually out-of-date by the time it is being executed.

Another drawback of such architectures is their lack of robustness. Since the information is processed sequentially, a failure in any of the components causes a complete breakdown of the system.

2.1.2 Behavior-based Robotics

Behavior-based robotics [3] appeared in the mid 1980s in response to the traditional hierarchical approach. Brooks [8] proposed to tightly couple perception to action, and thereby, provide a reactive behavior that could deal with any unpredicted situation the robot may encounter. Moreover, Brooks advocated for avoiding keeping any model of the environment in which the robot operates, arguing that "the world is its own best model". Behavior-based robotics is a bottom-up methodology, inspired by biological studies, where a collection of behaviors acts in parallel to achieve independent goals. Each of these behaviors is a simple module that receives inputs from the robot's sensors, and outputs actuator commands (see Figure 2.3). The overall architecture consists of several behaviors reading the sensory information and sending actuator commands to a coordinator that combines them in order to send a single command to each actuator (see Figure 2.4).

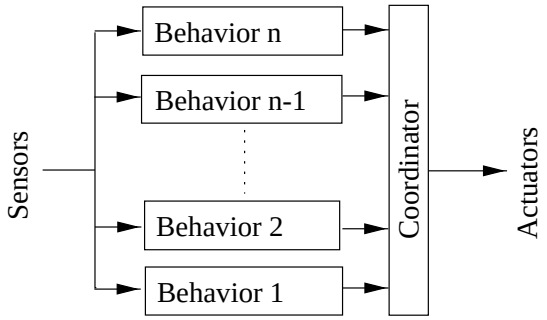


Figure 2.4: Behavior-based architecture

The most representative of such architectures are Brooks' *subsumption architecture* [8], Maes' *action selection* [43] and Arkin's *motor schemas* [4]. Since then, many other architectures have been proposed.

Behavior-based architectures are classified depending on how the coordination between behaviors is done:

- *Competitive*: in these architectures the coordinator selects an action coming from one of the behaviors and sends it to the actuators, that is, it is a winner-take-

all mechanism. Subsumption architecture and action selection are examples of competitive coordination.

- *Cooperative*: in these architectures the coordinator combines the actions coming from several behaviors to produce a new one that is sent to the actuators. Motor schemas is an example of cooperative coordination.

In this section we give a brief overview of the three most known behavior-based architectures and point out some others relevant to our work.

Subsumption architecture

The Subsumption architecture, designed by Rodney Brooks [8], was the first of the Behavior-based architectures. In this architecture each behavior is represented as a separate layer, having direct access to sensory information. Each layer has an individual goal, and they all work concurrently and asynchronously. A layer is constructed of a set of Augmented Finite State Machines (AFSM), connected by wires through which signals can be passed from one AFSM to another. These layers are organized hierarchically, and higher levels are allowed to subsume, hence the name, lower ones. This subsumption can take form of inhibition or suppression. Inhibition eliminates the signal coming out from an AFSM of the lower level, leaving it inactive. Suppression substitutes the signal of the AFSM by the signal given by the higher level. Higher level AFSMs can also send reset signals to lower ones. These mechanisms provide a competitive, priority-based coordination.

The hierarchical organization permits an incremental design of the system, as higher layers are added on top of an already working control system, with no need of modifying the lower levels. An example of such behavior layering is depicted in Figure 2.5.

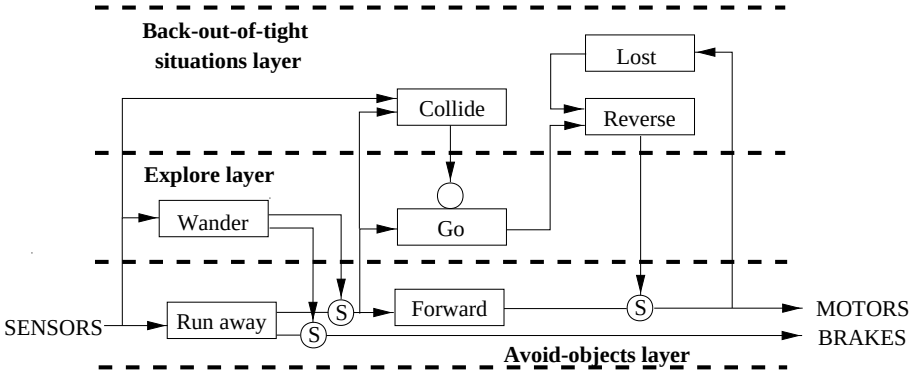


Figure 2.5: Example of a control system using the subsumption architecture. Each box is an AFSM, and signals are passed through the arrows connecting the AFSMs. An encircled S is a suppression point, and an empty circle is a reset point

The main strengths of this architecture are its incremental design methodology, which makes it easy and intuitive to build a system, its hardware retargetability (each

of the layers can be implemented directly on logic circuitry), and the support for parallelism, since each layer can run independently and asynchronously.

However, this theoretic independence is not absolute, since higher layers can suppress, inhibit and also read the signals of lower layers. Moreover, these connections between layers are hard-wired, so they cannot be changed during execution, thus, not allowing on-the-fly adaptability of the system to changes in the environment. One final aspect against this architecture is that it forces the designer to prioritize behaviors, therefore, the case of behaviors with equal priority cannot be represented with the subsumption architecture.

Action selection

Action selection is an architectural approach developed by Pattie Maes [43] that uses a dynamic mechanism for behavior (or action) selection. This dynamic mechanism solves the problem of the predefined priorities used in the subsumption architecture. Each behavior has an associated activation level, which can be affected by the current situation of the robot (gathered from the sensors), its goals, and the influence of other behaviors. Each behavior also has some preconditions that have to be met in order to be active. From all the active behaviors, the one with the highest activation level is chosen for actual execution.

This coordination mechanism resembles very much our bidding approach. In our architecture, each system (or agent within the Navigation system) bids according to the urgency for having the action executed, which is equivalent to the activation level. However, our bidding agents have no preconditions to be met in order to become active, and they are always ready to bid. Another important difference is that behaviors in action selection can influence the activation level of other behaviors, whereas in our approach the agents are totally independent, since an agent cannot influence the bids of another agent.

Motor schemas

The Motor schemas approach was proposed by Ronald Arkin [4], and it is a more biologically based approach to control architectures than the previous two. As in the previous approaches, each behavior receives sensory information as inputs and generates an action as output. This output is always a vector that defines how should the robot move, and can have as many dimensions as needed (e.g. two dimensions for ground-based navigation, three for flying or underwater navigation, etc.). Each behavior uses the potential field approach (developed by Khatib [34] and Krogh [37]) to produce its output vector. However, instead of computing the entire potential field, only the response at the current location of the robot is computed, allowing a simple and fast computation. Contrarily to the previous two approaches, motor schemas uses a cooperative coordination mechanism. The way the different behaviors are coordinated is through vector summation. Each behavior contributes to the global reaction depending on a gain factor (G_i). Each output vector (R_i) is multiplied by its behavior gain factor and summed up with the rest to produce a single output vector that will be sent to the robot's actuators (see Figure 2.6). These gain factors are very useful for adaptability purposes, as they

can be dynamically changed during execution, thus, as the action selection architecture, also overcoming the restricting subsumption architecture's priority scheme.

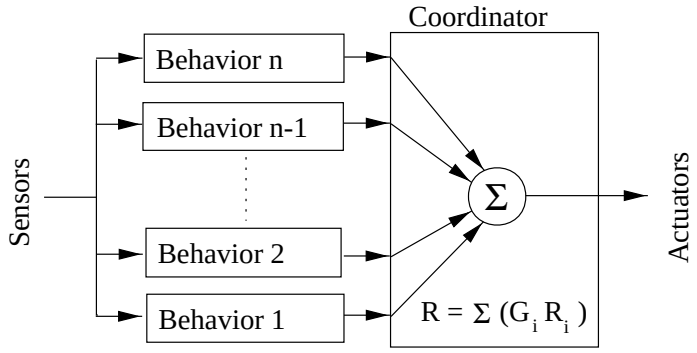


Figure 2.6: Motor-schemas architecture

However, cooperative mechanisms have some problems. A first problem is that they can reach local minima in the potential field. Imagine the situation in which the robot has an obstacle in front of it, and the task to be performed is to reach a target located right behind the obstacle. In this situation, the behavior for avoiding obstacles would compute a repulsive vector coming from the obstacle, while the go-to-target behavior would compute a vector going to the target, which would also point to the obstacle. Thus, in a particular location, the sum of both vectors would be null, and the robot would not move anymore from that location. This problem is easily solved by adding a noise schema, that always produces a small random vector in order to avoid these blocking situations from happening. Another problem of cooperative mechanisms is that the action actually executed is one that no behavior has generated. Again, imagine a robot with an obstacle ahead, and imagine that two different behaviors generate outputs for avoiding that obstacle, one trying to avoid it through the right and the other one trying to avoid it through the left. The sum of the vector would be a vector going straight ahead to the obstacle, which obviously would not be the best thing to do.

Other behavior-based systems

Rosenblatt [56], in CMU's Distributed Architecture for Mobile Navigation project (DAMN), proposed an architecture that is similar to our approach. In this architecture, a set of modules (behaviors) cooperate to control a robot's path by voting for various possible actions (steering angle and speed), and an arbiter decides which is the action to be performed. The action with more votes is the one actually executed. However, the set of actions is pre-defined, while in our system each agent can bid for any action it wants to perform. Moreover, in the experiments carried out with this system (DAMN), the navigation system used a grid-based map and did not use at all landmark based navigation.

Saffioti et al [58, 57] developed the Saphira architecture, which uses fuzzy logic to implement the behaviors. Each behavior consists of several fuzzy rules that have

fuzzy variables as antecedents (extracted from sensory and world model information), and generate as output a control set (i.e. fuzzy control variable). This control set is computed from the values of the fuzzy variables, and it represents the desirability of executing the control action, being similar to the activation level of the action selection architecture. Each behavior also has a fixed priority factor which is used for coordinating all the behaviors. This coordination is very similar to the cooperative mechanism used in Motor schemas. However, instead of combining vectors, it combines control sets and then defuzzifies the resulting set in order to get a single control value.

Humphrys [31] presents several action selection mechanisms that use a similar coordination mechanism to ours. Each agent suggests the action it wants the robot to perform with a given strength or weight (equivalent to our bid), and the action with the highest weight is the one executed. These weights are computed (and learned through Reinforcement Learning) using the one-step reward of executing an action, which each agent is able to predict for the actions it suggests. This is an important difference with our problem, since we cannot assign a one-step reward to an action; the only reward the robot may receive is when the robot reaches the target, and it is very difficult to split this reward into smaller rewards for each action taken during the navigation to the target.

2.1.3 Hybrid Architectures

Although it has been widely demonstrated that behavior-based architectures effectively produce a robust performance in dynamic and complex environments, they are not always the best choice for some tasks. Sometimes the task to be performed needs the robot to make some deliberation and keep a model of the environment. But behavior-based architectures do avoid this deliberation and modeling. However, as we have mentioned at the beginning of this section, purely deliberative architectures are also not the best choice for tasks in complex environments. Thus, a compromise between these two completely opposite views must be reached. This is what *hybrid architectures* achieve.

In these hybrid architectures there is a part of deliberation, in order to model the world, reason about it and create plans, and a reactive part, responsible of executing the plans and quickly reacting to any unpredicted situation that may arise. Usually these architectures are structured in three layers (see Figure 2.7): (1) the deliberative layer, responsible of doing high-level planning for achieving the goals, (2) the control execution layer, which decompose the plan given by the deliberative layer into smaller subtasks (these subtasks imply activating/deactivating behaviors, or changing priority factors), and (3) the reactive layer, which is in charge of executing the subtasks set by the control execution layer and can be implemented with any behavior-based architecture. Examples of such hybrid architectures, among others, are Arkin's AuRA [2] and Gat's Atlantis system [29] for JPL's rovers.

Another hybrid architecture, although not following the three-layer structure, is that of Liscano et al [25]. In their architecture, they use an activity-based blackboard consisting of two hierarchical layers for strategic and reactive reasoning. A blackboard database keeps track of the state of the world and a set of activities to perform the navigation. Arbitration between competing activities is accomplished by a set of rules that decide which activity takes control of the robot and resolves conflicts.

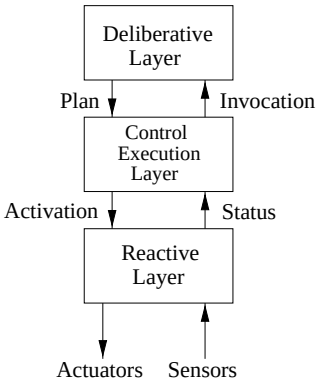


Figure 2.7: Three layers hybrid architecture

Although our approach is presented as a behavior-based system, it is not a purely reactive system, since there is some modeling (one of the agents of the Navigation system is in charge of building a map of the environment and computing routes) and deliberation (the agents reason about the world and communicate with each other). So if we had to classify it on the spectrum of control architectures, we would place it in the hybrid group, having the reactive and deliberative parts mixed in one single layer.

2.1.4 Bidding Mechanisms

Regarding the use of bidding mechanisms, we have found very few systems making use of it. At CMU, the FIRE project [19] uses a market-oriented approach to model the co-operation of a team of robots. In this approach, instead of using the bidding mechanism to coordinate the agents of a single robot, bidding is used to coordinate a team of robots that have to accomplish several tasks. The rationale is that with this mechanism, each task is performed by the best suited robot for the task, thus achieving a better global performance.

Sun and Sessions [63] have also proposed an approach for developing a multi-agent reinforcement learning system that uses a bidding mechanism to learn complex tasks. The bidding is used to decide which agent gets the control of the learning process. The agents bid according to the expected reward that would receive if they were given the control. Thus, although they are competing for the control, they also cooperate, since they seek to maximize the overall system reward.

2.2 Mapping and Navigation

The mapping problem is regarded as one of the most important problems in the field of autonomous robotics, and it dates back to SRI's famous Shakey robot [54]. A robot operating autonomously needs to answer the three basic questions about mapping and navigation, as posited by Levitt and Lawton [39]:

- Where am I?
- How do I get to other places from here?
- Where are other places relative to me?

This would be easy if an a priori map were available, but we are dealing with the scenario of unknown environments. That is, the robot has no knowledge at all about what the environment looks like, where the landmarks, the obstacles, etc are. To be able to answer these questions and, thereby, be able to perform its task, the robot must acquire a model of the environment in which it has to navigate through. Recent research on modeling unknown environments is based on two main approaches: *occupancy grid-based* (or metric), and *topological maps*.

Another distinctive and very important feature of mapping approaches is *localization*. The localization problem can be split in two very different particular problems: *local* localization and *global* localization. Local localization, also known as position tracking, aims at compensating odometric errors occurring during robot navigation. On the other hand, global localization is concerned with the problem of finding out where a robot is relative to a map of the whole environment. In this thesis we tackle the problem of global localization. There are two main approaches for solving it: *model matching* and *landmark based* localization.

In the rest of this section we will go through all these approaches, starting with the global localization approaches, and then the grid-based and topological mapping ones.

2.2.1 Localization

As just mentioned, global localization is the problem of finding out where a robot is relative to a map (i.e. align the robot's local coordinate system with the global coordinate system of the map). This problem is as important as being able to build a good map of the environment. No matter how good the map is, it will be of no use if we are not able to localize the robot on it. Conversely, even if we know how to localize the robot with high precision, that will be useless if there is no good map available on where to localize it. Moreover, the accuracy of a metric map depends highly on the alignment of the robot with its map. If we are not able to localize the robot, the resulting maps are too erroneous to be of practical use. As seen, these two problems are closely related, and most of the mapping approaches try to address both problems at the same time, in what is known as *simultaneous localization and mapping* (SLAM).

Model matching localization

These algorithms extract geometric features from the sensor readings and try to match them with a map of the environment, in order to correct possible odometric errors. This approach is closely related to grid-based mapping (described below), since these geometric features are the information pieces that grid-based mapping approaches store on the map.

The position of the robot is incrementally computed using odometry and information from sensors, by matching this information with the map already built. The sensor

information used for matching can be single sonar scans, which are matched with the obstacles on the map, such in Moravec and Elfe's approach [23, 52]. Other approaches, such as Chatila and Laumond's [18] extract geometric features (line segments and polyhedral objects) from the sensor readings and match them to a geometric map of the environment.

One problem with this approach is that it requires accurate odometry to disambiguate among positions that look similar. Probabilistic approaches (Smith et al [61], Fox et al [27], Castellanos and Tardós [16]) try to solve this ambiguity problem, and they are the most frequently used in the field of robot mapping. The basic idea of these algorithms is to employ probabilistic models of the robot and the environment to cope with the uncertainty of robot motion and sensor reading. In order to localize the robot, they use consecutive sensor readings to estimate a distribution over the space of all locations in the environment. The more readings the robot gets, the more precisely its location can be computed.

In our case we do not have to deal with this ambiguity, since we have developed a Vision system robust enough to correctly identify the landmarks. Thus, there is no uncertainty about the presence of a landmark. However, there is some imprecision about its location, which we deal with using fuzzy techniques.

The model matching approach, however, is computationally very expensive, since the process of matching the current sensor readings with the map requires many computations.

Landmark-based localization

In these approaches landmarks are used as references to compute the location of the robot. Landmarks can range from a set of sensor readings to artificial landmarks such as beacons or bar-codes or natural landmarks detected by vision systems. Because of its computational simplicity and also its close relationship with human navigational abilities, this approach is the most widely used, and it has been used with both grid-based and topological approaches.

This approach also suffers from the problem of ambiguity among landmarks that look similar. Again, the probabilistic approach can help solving this problem. Thrun [65] and Dissanayake et al [21], among others, use this approach together with grid-based maps, and Simmons and Koenig [60] and Kaelbling et al [33] combine it with the topological approach.

2.2.2 Map Representation

In order to navigate through the environment, the robot must create a model of it. There are two approaches to model the environment, the metric or grid-based approach, and the topological approach. Depending on the type of environment one or the other approach is most appropriate. Table 2.1 summarizes the advantages and disadvantages of these two approaches.

	Grid-based approaches	Topological approaches
ADVANTAGES	• easy to build, represent and maintain • non-ambiguous recognition of places and view-point independent • facilitates computation of shortest paths	• permits efficient planning, low space complexity • does not require accurate determination of robot's pose • convenient representation for symbolic planner/problem solver
DISADVANTAGES	• inefficient and space-consuming planning • requires accurate determination of the robot's position • poor interface for most symbolic problem solvers	• difficult to construct and maintain in large-scale environments if sensor information is ambiguous • recognition of places often difficult, sensitive to viewpoint • may yield suboptimal paths

Table 2.1: Advantages and disadvantages of grid-based and topological mapping approaches

Grid-based mapping

This approach was originally proposed by Elfes [23] and Moravec [51]. Cells in an occupancy grid contain information about the presence or not of an obstacle. Each of these cells is updated using sensor readings, and its value represents the degree of belief in the presence of an obstacle. The vast number of grid-based algorithms differ on the way in which sensor readings are translated into occupancy levels. Among other techniques, probability theory [51, 66, 67] and fuzzy set theory [41, 40] have been used. This mapping approach can be used in conjunction with the two localization approaches, as has been just described above.

In this approach, navigation is performed using path planning algorithms, which compute precise routes through the environment in order to reach a goal avoiding the obstacles.

Although this approach is widely used and achieves very good results, it is mainly focused for indoor structured environments. The size of such environments permits the robot to maintain a grid with a high enough resolution (i.e. small cells). In large outdoor environments, however, this technique cannot be applied, as the computational cost of the grid would be too high.

Moreover, in most of the algorithms following this approach, the robot has a training period in which it navigates through the environment with the only purpose of building

a map. After this training period, the robot is able to perform its task and localize itself using the already built map. In our scenario, however, there is no such training period, as the robot does not have the opportunity to inspect the environment before attempting to reach the target, but has to reach it while exploring the environment for the first time.

Topological mapping

In comparison to grid-based representations, topological representations (such as those proposed by Chatila [17], Kuipers and Byun [38], Matarić [47] and Kortenkamp [36], among others) are computationally cheaper. They use graphs to represent the environment. Each node corresponds to an environment feature or landmark and arcs represent paths or motion instructions between them. Some approaches (Kuipers [38], Kortencamp and Weymouth [35]) also define the nodes as "places", where a "place" is defined as a location where a set of features or landmarks fulfill a given property (e.g. sonar readings matching, landmark visibility, etc.).

With this graph, the problem of navigation is reduced to the problem of finding a route from one node to another – the target one. This can be easily computed with many graph search algorithms (Dijkstra's shortest path, A*, dynamic programming). However, this simplicity of computing routes has the disadvantage that the routes are not always the optimal ones, since there is not an accurate geometric description of the environment, and path planning algorithms for metric worlds cannot be applied. Moreover, in topological graphs there is no explicit representation of the obstacles, as in a metric map. Therefore, when moving from one node to another, there is no way of planning an optimal path, since there may be some obstacles on the way.

The advantage of topological approaches is that they do not rely on odometry in order to build the map nor localize the robot on it. The only point in which odometry is sometimes used is to label the arcs between nodes. As already mentioned, the arcs contain information about how to get from one node to another. This information can be, depending on the approach, metric information (heading and distance to the next node). If this were the case, the odometry error would influence the precision of this information. However, since neighboring nodes are close to each other, this error is bounded and does not accumulate as the robot navigates through the environment.

The drawback of not using metric information is that topological approaches have difficulties in determining if two places that look similar are the same place, since they compute the position of the robot relative to the known landmarks. This problem can be tackled if a robust enough landmark recognition system is in place. Landmark recognition is a very active field of research in vision and very promising results are being obtained [46]. In this work we assume that the vision system can recognize landmarks. However, in the absence of a robust recognition system, a probabilistic approach, similar to the one described for metric maps, could be applied.

Topological approaches can also be combined with grid-based approaches. Thrun [66] combines both representations in his work on learning maps for navigation in indoor structured environments. The grid-based map is partitioned in coherent regions to generate a topological map on top of the grid. By combining both methods, his approach gains the advantages of both methods, resulting in an accurate, consistent and efficient mapping approach. This is indeed a good idea for indoor environments but for

large-scale outdoor environments may not be worth the computational effort of maintaining a grid representation under a topological one.

In our work we use the approach where nodes represent regions defined by groups of three landmarks and that are connected by arcs if the regions are adjacent, that is, if they have two landmarks in common. The arcs, instead of containing motion information, represent the cost of going from one region to another. This graph is incrementally built while the robot is moving within the environment. This incremental map building approach is based on previous work by Prescott [55] that proposed a network model that used barycentric coordinates, also called beta-coefficients by Zipser [68], to compute the spatial relations between landmarks for robot navigation. By matching a perceived landmark with the network, the robot can find its way to a target provided it is represented in the network. Prescott's approach is quantitative whereas our approach uses a fuzzy extension of the beta-coefficient coding system in order to work with fuzzy qualitative information about distances and directions. Another difference with Prescott's approach is that his topological graph contains only adjacency information, thus, not maintaining any information about costs, as in ours. This cost information is very important when planning routes from one region to another, since it is the only way to know whether a path is blocked or free. One final point to mention is that in Prescott's experiments, carried out only on simulation, the robot was allowed a training period, while this period is not present in our approach.

Levitt and Lawton [39] also proposed a qualitative approach to the navigation problem. In this approach, landmark pairs divide the environment into two regions, one for each side of the line connecting the two landmarks. The combination of all such linear divisions generates a topological division of the environment, on which navigation can be performed. Navigation consists of crossing a series of landmark pairs in order to reach the region containing the target landmark. Our navigation method uses the same idea for computing and navigating to diverting targets. The difference between this approach and ours is that we use three landmarks for creating the region subdivision, instead of only two. This gives as result a better and more compact division of the environment. Moreover, this third landmark permits the robot to compute a relationship among the landmarks that is unique and invariant to viewpoint.

Another qualitative method for robot navigation was proposed by Escrig and Toledo [24], using constraint logic. However, they assume that the robot has some a priori knowledge of the spatial relationship of the landmarks, whereas our system builds these relationships while exploring the environment.

One of the drawbacks of most of the mapping approaches is that they are thought for static environments. That is, landmarks are not supposed to change their location while the robot is exploring the environment. Thus, research on vision systems capable of extracting robust (distinguishable, invariant to viewpoint and illumination, static) landmarks is very important. However, some mapping approaches are already able to cope with dynamic environments. In [1] landmarks have an existence state (using the principles of neural networks). This mechanism permits the removal of landmarks for which their existence is not certain enough. We have used a similar idea to devise a *Visual Memory* (see chapter 4), a short term memory of detected landmarks.

Chapter 3

Mapping and Navigation

As already mentioned, the task the robot has to perform is to navigate through an unknown unstructured environment and reach a target landmark specified by a human operator. This task is not easy to solve, since it has to be carried out in a complex environment, and the target can be occluded by other objects. Purely reactive robotic systems would have problems trying to accomplish this task, since they do not build any model of the environment. If the target were lost, it would be difficult to recover its visibility and continue the navigation towards it. For this reason, we thought that the robot should build a map of the environment in order to navigate through it. The information stored in the map must permit the robot to compute its location, the location of the target, and how to get to this target. Although the objective of this PhD thesis is to develop a navigation system for indoor environments, we have used a map representation that also works outdoors, since this is the next milestone of the project in which we are involved. Thus, instead of using a grid-based approach, the most widely used approach for indoor environments, we have used a topological one, most appropriate also for outdoors.

Our approach is based on the model proposed by Prescott in [55]. The principles underlying this model are inspired by studies of animal and human navigation and wayfinding behavior. This model, called *beta-coefficient system*, does not only deal with how to represent the environment as a map, but also adds a mechanism for computing the location of landmarks when they are not visible, based on the relative positions of the landmarks. This mechanism is what we have used to provide the robot with orientation sense, since it captures the relationship among different places of the environment. The robot makes use of this orientation sense to compute the location of the target when it is occluded by other objects or obstacles.

In this chapter we firstly describe how Prescott's model works when the robot is able to have exact information about its environment, and then we explain how we have extended it to work with imprecise information. We also describe the method used for dividing the environment into appropriate topological regions, and finally how the topological map is used to navigate through the environment.

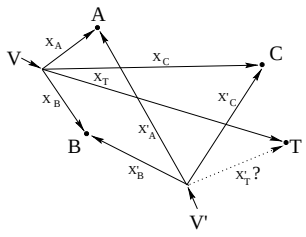


Figure 3.1: Possible landmark configuration and points of view. Landmarks A, B, C and T are visible from viewpoint V. Only landmarks A , B and C are visible from viewpoint V'

3.1 Beta-coefficient System

The idea behind Prescott's model is to encode the location of a landmark (which we refer to as target – not to confuse with the target or goal of the Navigation system) with respect to the location of three other landmarks. Having seen three landmarks and a target from a viewpoint (e.g., landmarks A, B and C and target T from viewpoint V, in Figure 3.1), the system is able to compute the target's position when seeing again the three landmarks, but not the target, from another viewpoint (e.g., V'). A vector, called the β-vector of landmarks A, B, C and T, is computed as

$$\beta = X^{-1}X_T \tag{3.1}$$

where $X = [X_A X_B X_C]$ and $X_i = (x_i, y_i, 1)^T$, are the homogeneous Cartesian coordinates of object i, $i \in \{A, B, C, T\}$, from viewpoint V. This relation is unique and invariant for any viewpoint if landmarks are distinct and non collinear. The target's location from viewpoint V' is computed as

$$X'_T = X'\beta, \tag{3.2}$$

where $X' = [X'_A X'_B X'_C]$.

It should be noted that, although Prescott's system works with Cartesian coordinates, once all the computations have been done, the resulting target's location is converted to polar coordinates, since, as will be seen in next chapters, this is the coordinate system that uses the Navigation system.

This method can be implemented with a two-layered network. Each layer contains a collection of units, which can be connected to units of the other layer. The lowest layer units are *object-units*, and they represent the landmarks the robot has seen. Each time the robot recognizes a new landmark, a new object-unit is created. The units of the highest layer are *beta-units* and there is one for each β-vector computed.

When the robot has four landmarks in its viewframe, it selects one of them to be the target, a new beta-unit is created, and the β-vector for the landmarks is calculated. This beta-unit will be connected to the three object-units associated with the landmarks (as incoming connections) and to the object-unit associated with the target landmark (as an outgoing connection). Thus, a beta-unit will always have four connections, while

an object-unit will have as many connections as the number of beta-units it participates in. An example of the network can be seen in Figure 3.2b. In this figure there are six object-units and three beta-units. The notation ABC/D is understood as the beta-unit that computes the location of landmark D when the locations of landmarks A, B and C are known.

This network has a propagation system that permits the robot to compute the location of non-visible landmarks. It works as follows: when the robot sees a group of landmarks, it activates (sets the value) of the associated object-units with the egocentric locations of these landmarks. When an object-unit is activated, it propagates its location to the beta-units connected to it. On the other hand, when a beta-unit receives the location of its three incoming object-units, it gets active and computes the location of the target it encodes using its β-vector, and propagates the result to the object-unit representing the target. Thus, an activation of a beta-unit will activate an object-unit that can activate another beta-unit, and so on. For example, in the network of Figure 3.2b, if landmarks A, B and C are visible, their object-units will be activated and this will activate the beta-unit ABC/D, computing the location of D, which will activate BCD/E, activating E, and causing BDE/F also to be activated. In this case, knowing the location of only three landmarks (A, B and C), the network has computed the location of three more landmarks that were not visible (D, E and F). This propagation system makes the network compute all possible landmarks' locations. Obviously, if a beta-unit needs the location of a landmark that is neither in the current view nor activated through other beta-units, it will not get active.

This propagation system adds robustness to the computation of non-visible landmarks, since a landmark can be the target of several beta-units at the same time. Because of imprecision in the perception on landmark locations, the estimates of the location of a target using different beta-units are not always equal. When this happens, the different location estimates must be combined. Prescott uses the size of the β-vector as the criterion to select one among them. A beta-unit with a smaller β-vector is more precise than those with larger β-vectors (see [55] for a detailed discussion on how to compute the estimate error from the size of the β-vector). The propagation system does not only propagate location estimates, but also the size of the largest β-vector that has been used to compute each estimate. When a new location estimate arrives to an object-unit, its location is substituted with the new one if the size of the largest β-vector used is smaller than that used for the last location estimate received.

The network created by object and beta units can be converted into a graph where the nodes represent triangular shaped regions delimited by a group of three landmarks, and the arcs represent paths. These arcs can have an associated cost, representing how difficult it is to move from one region to another. Although the arcs are created immediately when adding a new node to the graph, the costs can only be assigned after the robot has moved (or tried to move) along the path connecting the two regions. In the case the path is blocked by an obstacle, the arc is assigned an infinite cost, representing that it is not possible to go from one region to the other. This graph is a topological map, and we call its nodes *topological units*. An example of how the topology is encoded in a graph is shown in Figure 3.2c.

This topological map is used when planning routes to the target. Sometimes, when

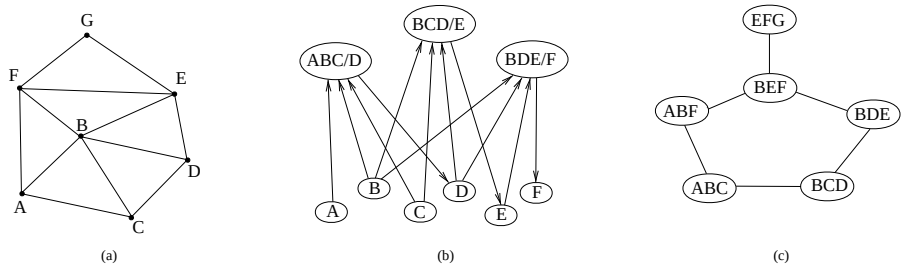


Figure 3.2: (a) Set of landmarks (b) associated network (partial view) and (c) associated topological map

the position of the target is known, the easiest thing to do is to move in a straight line towards it, but sometimes it is not (the route can be blocked, the cost too high...). With the topological map, a route to the target can be computed. In Section 3.4 a detailed explanation on how to compute routes to the target is given.

3.2 Extending Prescott's System: Moving to Fuzzy

The beta-coefficient system, as described by Prescott, assumes that the robot can compute the position of the landmarks with small errors, in order to create the beta-units and use the network. But this is never the case: the Vision system provides the robot with inexact information about the location of landmarks. To work with this imprecise information we use fuzzy numbers.

3.2.1 Fuzzy Numbers and Fuzzy Operations

A fuzzy number can be thought of as a weighted interval of real numbers, where each point of the interval has a degree of membership, ranging from 0 to 1 [7]. The higher this degree, the higher the confidence that the point belongs to the fuzzy number. The function $F_A(x)$, called *membership function*, gives us the degree of membership for x in the fuzzy number A.

Before defining the arithmetic with fuzzy numbers, we have to introduce the concept of α-cut. The α-cut ($\alpha \in [0,1]$) of a fuzzy number A, is the interval $\{A\}_\alpha = [a_1, a_2]$ such that $F_A(x) >= \alpha, \forall x \in [a_1, a_2]$.

Let A and B be fuzzy numbers, and $\{A\}_\alpha$ and $\{B\}_\alpha$ α-cuts. The fuzzy arithmetic operations are defined as follows,

$A + B = C, s.t.\ \{C\}_\alpha = \{A\}_\alpha \oplus \{B\}_\alpha\ \forall\alpha$
$A - B = C, s.t.\ \{C\}_\alpha = \{A\}_\alpha \ominus \{B\}_\alpha\ \forall\alpha$
$A \times B = C, s.t.\ \{C\}_\alpha = \{A\}_\alpha \otimes \{B\}_\alpha\ \forall\alpha$
$A \div B = C, s.t.\ \{C\}_\alpha = \{A\}_\alpha \oslash \{B\}_\alpha\ \forall\alpha$

where the operations $\oplus, \ominus, \otimes$ and $\oslash$ are performed on intervals and are defined as

$[a_1, a_2] \oplus [b_1, b_2] = [a_1 + b_1, a_2 + b_2]$
$[a_1, a_2] \ominus [b_1, b_2] = [a_1 - b_2, a_2 - b_1]$

$[a_1, a_2] \otimes [b_1, b_2] = [min(a_1b_1, a_1b_2, a_2b_1, a_2b_2), max(a_1b_1, a_1b_2, a_2b_1, a_2b_2)]$
$[a_1, a_2] \oslash [b_1, b_2] = [a_1, a_2] \otimes [\frac{1}{b_2}, \frac{1}{b_1}],\ 0 \notin [b_1, b_2]$

3.2.2 Fuzzy Beta-coefficient System

To use the beta-coefficient system with fuzzy numbers, we simply perform the calculations described in the previous section using the fuzzy operators defined above. However, because of the nature of fuzzy operators, some landmark configurations may not be feasible (the matrix inversion used for computing the β-vector – Equation 3.1– may produce a division by 0), so not all configurations can be stored in the network.

When using the network to compute the position of a landmark, we obtain a fuzzy polar coordinate (r, ϕ), where r and ϕ are fuzzy numbers, giving us qualitative information about its location. An advantage of working with fuzzy coordinates is that it gives us information about how precise the location estimate is, since it represents the location not as a crisp coordinate, but as a spatial region where the landmark is supposed to be.

Another difference with Prescott's model is the criterion used to select among different estimated locations for the same landmark. In our extended system, instead of looking at the size of the β-vectors, we use the imprecision of the estimated location itself. The imprecision of a landmark location, $I(l)$, is computed by combining the imprecision in the heading and in the distance as follows. $I_h(l)$ is the imprecision in heading, and it is defined by taking the interval corresponding to the 70% α-cut of the fuzzy number representing the heading to the landmark (see Figure 3.3). This imprecision is normalized dividing it by its maximum value of 2π. Similarly, $I_d(l)$ is the imprecision in distance, and it is defined as the 70% α-cut of the fuzzy number representing the distance. It is normalized by applying the hyperbolic tangent function, which maps it into the $[0, 1]$ interval. Finally, the two imprecisions are combined according to:

$$I(l) = \lambda \cdot \tanh(\beta \cdot I_d(l)) + (1 - \lambda) \cdot \frac{I_h(l)}{2\pi} \tag{3.3}$$

where λ weighs the relative importance of the two imprecisions, and β controls how quickly the transformed I_d approaches 1. In our experiments, we set $\beta = 1$ and $\lambda = 0.2$. When an object-unit receives a new location estimate, it computes the imprecision of this estimate, compares it with the imprecision of the current location estimate, and keeps the least imprecise location.

3.3 Building the Map

In Section 3.1 we mentioned that when the robot has four landmarks in its viewframe, it creates a new beta-unit for them. However, with four landmarks, there are four candidates to be the target of the beta-unit. Moreover, if the robot has more than four landmarks in the viewframe, there are many possible beta-units to be created. More precisely, if there are n visible landmarks, there are $\binom{n}{4} \cdot 4$ candidates for being new beta-units. However, it is not feasible to store them all, firstly because of the huge num-

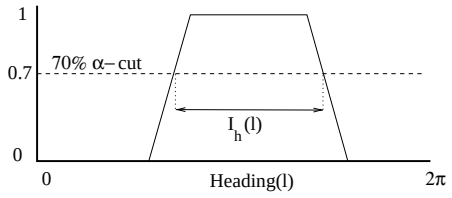


Figure 3.3: Computation of the imprecision of the heading toward landmark l as a fuzzy number

ber of combinations, and secondly, and more important, because some configurations are better than others. Thus, some selection criterion must be used.

Before describing the criterion we have used, we explain how the obstacles are represented in the map. We differentiate two types of obstacles: *point* obstacles and *linear* obstacles. *Point* obstacles are those the robot can easily avoid by slightly modifying its trajectory, since they do not completely block the path. In our indoor environment such obstacles are boxes and bricks. In outdoor environments they could be small rocks, trees, etc. These obstacles do not affect the global navigation, as the Pilot can tackle them alone, so the Navigation system does not take them into account and they are not stored in the map. On the other hand, *linear* obstacles are long obstacles that completely block the path of the robot. They can also be avoided by the Pilot, but the trajectory has to be drastically modified. In our indoor environment we use several bricks to form these obstacles. In an outdoor environment these obstacles could be fences, walls, groups of rocks, etc. Since these obstacles do highly affect the navigation task, they have to be represented in the map, so that they are taken into account when computing routes to the target. The information about these obstacles is stored on the arcs of the topological map. An arc is labelled with an infinite cost to indicate that there is an obstacle between the two regions connected by the arc. Notice that with this representation we can only represent those obstacles placed along the line connecting two landmarks. Although in our experiments we have designed the environments so that they satisfy this condition, the system would also work if it were not satisfied. However, in this latter case, the Navigation system could not take all the obstacles into account, and thus, its performance would be worse. The arcs' labels are updated whenever the Pilot system informs about the presence of an obstacle between two landmarks.

Going back to the selection criterion, given a set of landmarks, for which their location is known, we seek to obtain a set of triangular regions with the following constraints:

- *Low collinearity*: the collinearity of a region is computed as

$$Col(R) = 1 - \frac{\alpha\beta\gamma}{(\frac{\pi}{3})^3} \tag{3.4}$$

  where α, β and γ are the three angles of the triangular region. The best quality is associated to equilater triangles, where $\alpha = \beta = \gamma = \frac{\pi}{3}$, and hence their collinearity is 0. When one of the angles is 0, landmarks would be maximally

collinear and $Col(R) = 1$. The higher the collinearity, the higher the error on the computation of the β-vector and landmark locations (see [55] for a detailed explanation). Therefore, the regions with lower collinearity are preferred. For example, in Figure 3.4 the two regions on the right are preferred over the two on the left, since the region ABD is too collinear.

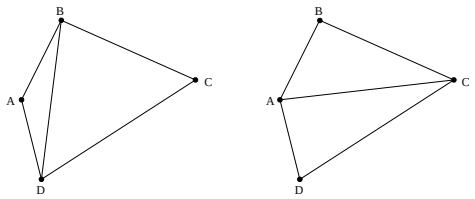


Figure 3.4: *Left*: bad set of regions; region ABD is too collinear. *Right*: good set of regions

- *Connectivity*: the set of regions must be converted into a graph with a single component, so that there is a path between any two nodes of the graph. In Figure 3.5, the set of regions on the left is not acceptable, since there are two disconnected components, whereas in the set on the right all the regions are connected.

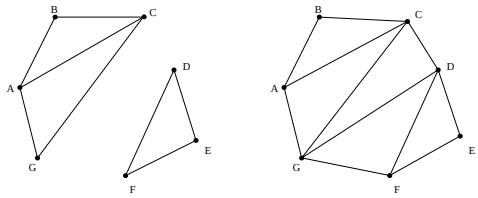


Figure 3.5: *Left*: bad set of regions; there are two disconnected components. *Right*: good set of regions

- *Convex hull covering*: the regions must cover the convex hull of the set of landmarks, so that the environment is represented completely, with no unrepresented regions. In Figure 3.6, the set on the left is not acceptable, since the region DFG is not represented.

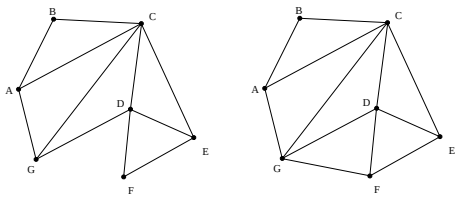


Figure 3.6: *Left*: bad set of regions; region DFG is missing. *Right*: good set of regions

- *Non overlapping*: the regions should not overlap with each other. If this were the case, the robot could be in more than one region at the same time, which could cause some problems when computing routes to the target. For instance, if the robot were in the overlapping area of the two regions, it would make no sense to order the robot to move from one region to the other, since it would already be inside both regions, and the order would not have any effect. Moreover, if one of the overlapping edges is an obstacle, the path from one side of the adjacent region to the other side would be blocked, which is obviously a bad representation of the environment, since the robot must be able to move around the whole space of a region. In Figure 3.7, the set of regions on the left is a bad set, since part of the obstacle between landmarks B and D lies inside the region ADC. In this case, the associated graph would have two nodes, ABD and ACD, which would be connected, so the robot would think that it can move from region ABD to region ADC, but it would find the path blocked because of the obstacle.

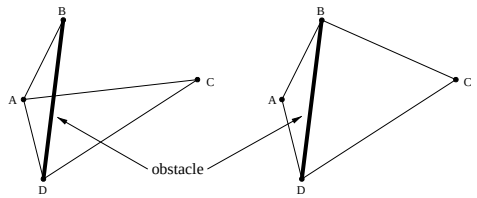


Figure 3.7: *Left*: bad set of regions; the obstacle between landmarks B and D is inside the region ACD. *Right*: good set of regions

- *Keep obstacles*: if an edge of a region is marked as an obstacle, this edge must be kept in the map, even if it causes the robot to keep high collinear regions. The obstacle edges are the only ones that cannot be removed from the map. If we did so, the information about the location of obstacles would be lost and would not be taken into account when computing routes to the target.

To compute the optimal set of regions for a given set of landmarks, we have developed an incremental algorithm that treats landmarks one by one to update the map. However, the algorithm only starts working when the locations of at least four landmarks are known, since this is the number of landmarks needed to create a beta-unit. With these four landmarks, the mapping algorithm computes the best set of regions according to the constraints given above. Then, the rest of visible landmarks, if any, are added one by one to the already built map. When adding a new landmark to the map, two situations can happen: (1) the landmark is inside an already existing region, or (2) the landmark is outside any region. In the first case, the region containing the new landmark is replaced by three new regions (see Figure 3.8). In the second case, all the possible new regions are created (see Figure 3.9). No matter the situation of the landmark, once the new regions have been created, the algorithm checks if the resulting map is still optimal. This optimization consists of analyzing each pair of adjacent regions and checking if their configuration is optimal according to the constraints. If it

finds that some regions could be changed so that a better configuration is obtained, it does so. An example of this step by step updating is shown in Figure 3.10.

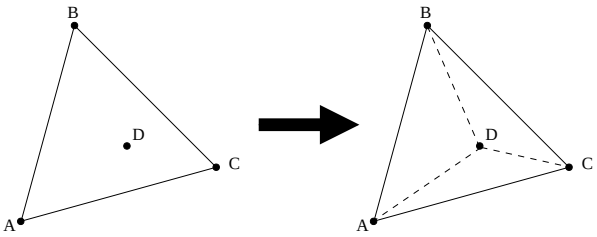


Figure 3.8: Adding a new landmark (D) located inside an existing region (ABC) resulting in the substitution of the original region for three new regions (ABD,ACD,BCD)

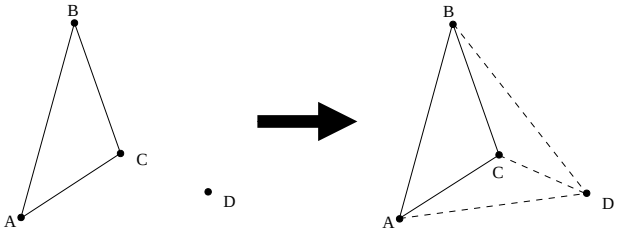


Figure 3.9: Adding a new landmark (D) located outside any existing region resulting in the addition of two new regions (ACD,BCD)

Once the set of regions is computed, new beta and topological units can be created. For each new region a beta-unit is created for each region adjacent to it, taking the three landmarks of the first region as the encoding landmarks, and the landmark of the second region that is not in the first one as the target. In other words, for each pair of adjacent regions, two "twin" beta-units are created. An example can clarify this explanation: with the regions ABC and ACD shown on the right in Figure 3.4, the beta-units ABC/D and ACD/B would be created. One topological unit is also created for each new region, and the graph is updated according to the adjacency of regions. Initially, the arcs are labelled with a default cost of 1, and they are changed to ∞ whenever an obstacle is detected. The topological units corresponding to regions that are not used any more are removed from the graph. However, beta-units are never removed, since they add robustness to the system, as in Section 3.1.

This triangulation algorithm needs the location of the landmarks to be known (either recognized by the Vision system or computed by the beta-coefficient system). However, not all landmark locations can always be known. The algorithm only takes into account those landmarks whose locations are known. This ensures that the five constraints explained above are satisfied only for the located landmarks. When one of the unlocated landmarks is seen or computed, some constraints might become unsatisfied. Whenever any constraint is broken, the map is rebuilt in order to satisfy again all the constraints. This constraint break can also be caused by the fuzziness of the locations. Because of

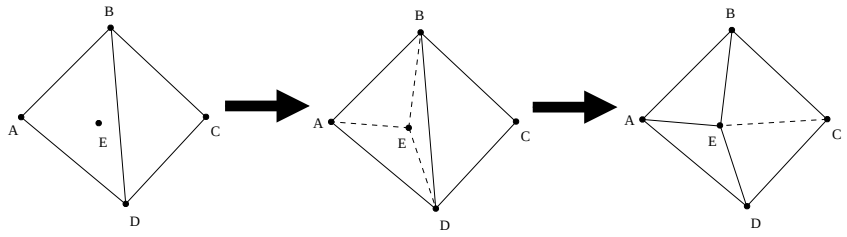


Figure 3.10: Adding a new landmark (E) into a map with two regions (ABD and BCD): first, region ABD is substituted for three new regions (ABE,ADE,BDE); after that, optimization for regions BCD and BDE is performed and they are substituted for the new regions BCE and CDE

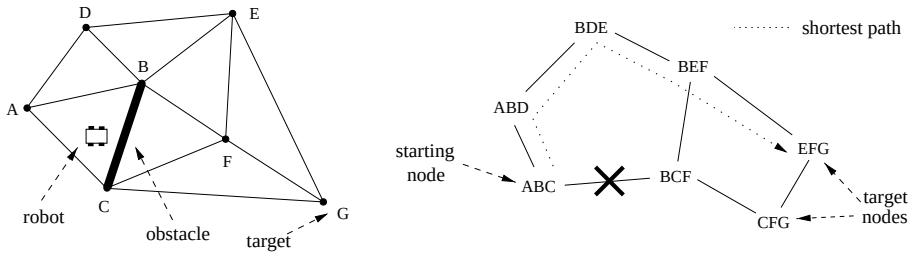
the imprecision of the locations, the map can suddenly be breaking some of the constraints. To avoid having an inconsistent map, every once in a while the satisfaction of the constraints is checked, and, if needed, the map is rebuilt.

3.4 Navigating Through the Environment

The beta-coefficient system described above provides the means for computing the location of a target even if it is not visible. This is very useful if the robot is navigating in an environment with a high density of landmarks and obstacles that occlude the target. In this case, the robot is able to go towards the target by seeing other landmarks. However, in some cases the obstacles might be blocking the direct path to the target. In this case, knowing the location of the target is not enough and an alternative route to reach it must be computed using the topological map.

Although a route consists of a sequence of regions the robot should navigate through in order to reach the target, only the first region is taken into account. The reason for doing so is that since the environment is never fully known, the robot cannot commit to a given route because it might encounter new landmarks and obstacles that would change the shape of the map, and possibly, the route to the target. Therefore, hereafter, instead of talking about routes, we will talk about diverting targets. A diverting target can be: (1) an *edge* between two landmarks, which the robot has to cross in order to go from one region to another, or (2) a *single landmark* to which the robot has to approach.

When the system is asked for a diverting target in order to reach another target, it first finds out in which region the robot is currently located, using the information about the landmarks whose location is known. This region will be the starting node on the topological map. The shortest path from this node to any of the nodes containing the target landmark (a landmark can be component of several topological regions) is computed. The edge connecting the current region with the next one on the shortest path will be the diverting target. The edge is given as a pair of landmarks, one that has to be kept on the left hand side of the robot and another to be kept on the right hand side, so the robot knows which way the edge has to be crossed. An example is shown in Figure 3.11. In this case, the robot is in region ABC, the target is G, and the shortest



Real environment Topological map

Figure 3.11: Diverting target computation

path to the target would be {ABC,ABD,BDE,BEF,EFG}. Thus, the diverting target would be the edge AB.

However, it could happen that there is no such shortest path. The cases in which such path does not exist are the following:

- The robot is not in any topological region.
- The cost of the shortest path is infinite. This means that the path is blocked by an obstacle, so it is not a valid path.
- The target is not found in any topological region.

To solve the first two cases, the map has to be enlarged with virtual regions through which the robot can navigate. The idea is to let the robot move in an unknown area outside the map. The virtual regions are built by placing some virtual landmarks around the existing map, and creating the appropriate regions using the same algorithm as described in the previous section. An example of these virtual regions is depicted in Figure 3.12. To force the robot to use regions of the original map, the arcs connecting virtual regions are labelled with a high cost (though not infinite), so that they are used only if it is absolutely necessary. With this enlarged map, the shortest path is computed again. However, it can be that the edge to be crossed contains one virtual landmark. In this case, the edge cannot be given as the diverting target, since the virtual landmarks do not exist on the real environment and cannot be tracked. In this situation, the direction to the middle point of the edge is computed and given as the diverting target. We assume that there is always some free space around the explored area, so that the regions created with the virtual landmarks can be traversed.

In the case the target is not in any topological region, there is no way to compute which should be the next region to visit, since there is no destination node. When this happens, the diverting target is set to any of the visible landmarks, hoping that on the way to this diverting target, the map is updated and the target for which a diverting target has been computed is incorporated into it.

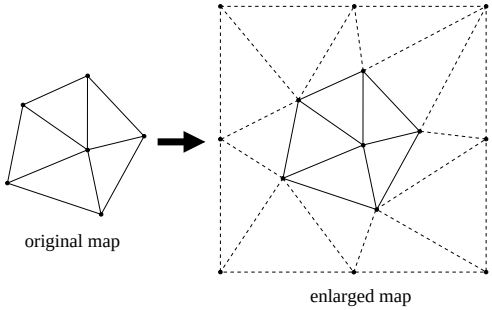


Figure 3.12: Enlarging the map with virtual regions (dotted lines)

3.5 Future Work

Although the extension of Prescott's method, together with the algorithms to compute diverting targets, is enough for permitting a robot build a map and navigate through an unknown environment, we would like to explore other mapping methods, so that the combination of the different methods adds robustness to the Navigation system. With the current mapping method, the robot needs to see at least three landmarks in order to be able to use the information stored in the map. We would like to develop some other mapping methods to cope with the situations in which the robot has very little information (i.e. less than three landmarks). These methods would be even more qualitative than our fuzzy extension of Prescott's method. We could, for example, look at the field of Spatial Cognition, which works with spatial relationships such as "landmark X is at the left hand side of the line connecting landmark Y and landmark Z".

Chapter 4

The Robot Architecture

Navigation, as the general task of leading a robot to a target destination, is naturally intermingled with other low-level tasks such as obstacle avoidance, and high-level tasks such as landmark identification. We can see each of the tasks, from an engineering point of view, as a system, that is, systems require and offer services one another. These systems need to *cooperate*, since they need one another in order to achieve the overall task of reaching the target. However, they also *compete* for controlling the available actuators of the robot. To exemplify this cooperation and competition, imagine a robot controlled by three systems, the Pilot system, the Vision system and the Navigation system. Actually, these three systems compose the architecture we have used to control our robot, which will be described in detail in the rest of this chapter. Regarding the cooperation, the Navigation system needs the Vision system to recognize the known landmarks in a particular area of the environment or to find new ones, and it also needs the Pilot system to move the robot towards the target location. Regarding the competition, the Navigation system may need the robot to move towards the target, while the Pilot system may need to change the robot's trajectory to safely avoid an obstacle. Moreover, the Pilot may need the camera to check whether there is any obstacle ahead and, at the same time, the Navigation system may need to look behind to localize the robot by recognizing known landmarks. Thus, some coordination mechanism is needed in order to handle this interaction among the different systems. The mechanism has to let the systems use the available resources in such a way that the combination of these interactions results in the robot reaching its destination.

We propose a general architecture for managing this cooperation and competition. We differentiate two types of systems: *executive systems* and *deliberative systems*. *Executive systems* have access to the sensors and actuators of the robot. These systems offer services for using the actuators to the rest of the systems and also provide information gathered from the sensors. On the other hand, *deliberative systems* take higher-level decisions and require the services offered by the executive systems in order to carry out the task assigned to the robot. Despite this distinction, the architecture is not hierarchical, and the coordination is made at a single level involving all the systems. The services offered by the executive systems are not only available to the deliberative systems; they are also available to the executive systems themselves. Actually, an

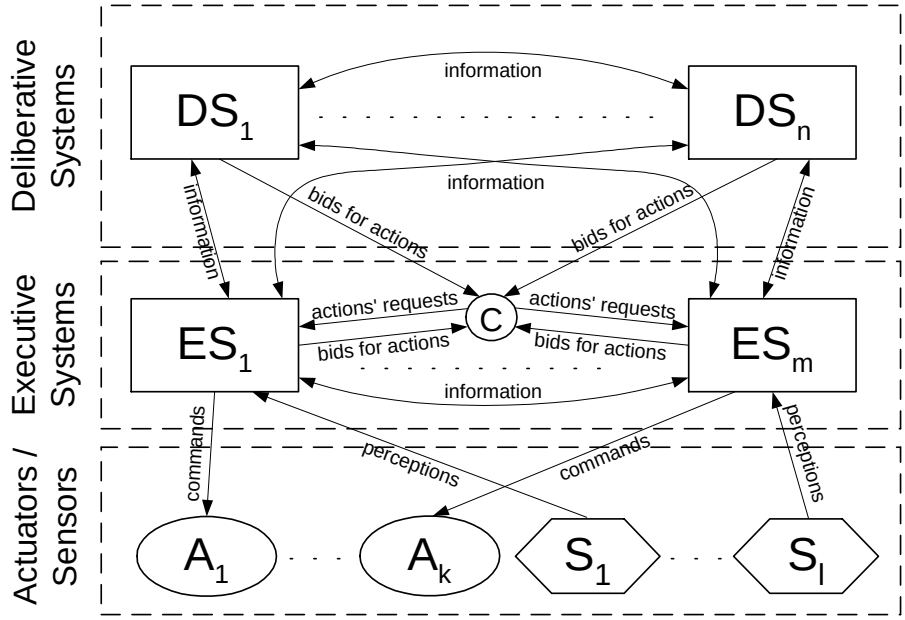


Figure 4.1: General bidding coordination architecture

executive system must compete with the rest of the systems even for the services it is offering. The systems (no matter their type) can exchange information between them (be it sensory information or any other information they could have – e.g. map of the environment). The architecture is depicted in Figure 4.1.

The coordination is based on a simple mechanism: *bidding*. Deliberative systems always bid for the services offered by executive systems, since this is the only way to have their decisions executed. Executive systems that only offer services do not bid. However, those executive systems that require services from any executive system (including themselves) must also bid for them. The systems bid according to the internal expected utility associated to the provisioning of the services. A coordinator receives these bids and decides which service each of the executive systems has to engage in.

Although we use the term "bidding", there is no economic connotation as in an auction. That is, systems do not have any amount of money to spend on the bids, nor there is any reward or good given to the winning system. We use it as a way to represent the urgency of a system for having a service engaged. The bids are in the range $[0, 1]$, with high bids meaning that the system really thinks that the service is the most appropriate to be engaged at that moment, and with low bids meaning that it has no urgency in having the service engaged.

This bidding mechanism is a competitive coordination mechanism, since the action executed by each system is the consequence of a request of one of the systems, not a combination of several requests for actions made by different systems, as it would be in a cooperative mechanism.

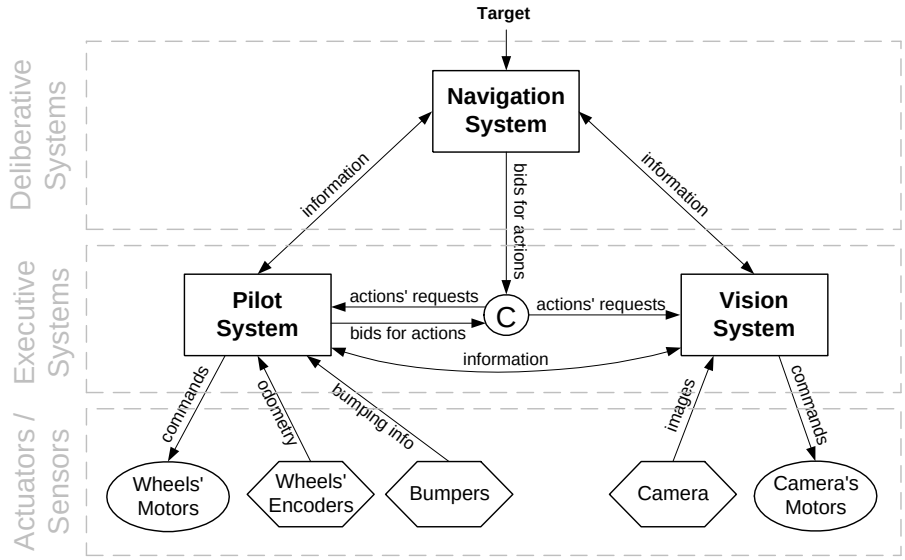


Figure 4.2: Specific robot architecture

This modular view forms an extensible architecture. To extend this architecture with a new capability we would just have to plug in one or more new systems, eventually adding new sensors or actuators, and eventually changing the bidding functions of the existing systems. Not only that, it also permits us to recursively have a modular view of each one of the systems, as will be soon seen in the design of our Navigation system. Moreover, this architecture is not thought only for navigation purposes since its generality can be used for any task that could be assigned to a robotic system.

For our specific robot navigation problem, we have instantiated the general architecture described above (see Figure 4.2). It has two executive systems, the *Pilot* and *Vision* systems, and one deliberative system, the *Navigation* system. Each system has the following responsibilities. The Pilot is responsible for all motions of the robot, avoiding obstacles if necessary. The Vision system is responsible for identifying and tracking landmarks (including the target landmark). Finally, the Navigation system is responsible for taking higher-level decisions in order to move the robot to a specified target. The robot has two actuators: the *wheels' motors*, used by the Pilot system, and the *camera motor*, used by the Vision system. The available sensors are the wheel encoders and bumpers, which provide *odometric* and *bumping* information to the Pilot, and the *images* obtained by the camera, used by the Vision system to identify landmarks. The Pilot system offers the service of moving the robot in a given direction, and the Vision system offers the service of moving the camera and identifying the landmarks found within a given area. The bidding systems are the Pilot and the Navigation system, while the Vision system does not bid for any service.

In the next sections we describe each of the three systems of the robot architecture, focusing on the Navigation system, the main subject of this thesis.

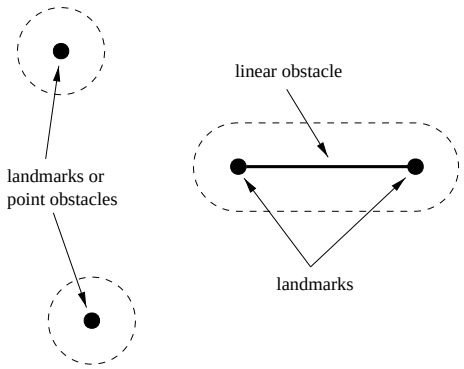


Figure 4.3: Growing obstacles. Points and solid lines are the obstacles; dotted lines show grown obstacles

4.1 Pilot System

The Pilot is able to safely command the motors that control the robot to move in a given direction. It bids for motion control to avoid obstacles, and also for the control of the camera to look forward in order to detect possible obstacles. Although this system is not the focus of this thesis, we have had to develop a simple Pilot in order to test our Navigation system.

For obstacle avoidance, it uses the information coming from the Vision system and the information stored in the Visual Memory (described in the next section), applying an obstacle growing technique. The obstacles are grown a given size to define forbidden areas occupied by the obstacles. The obstacles are represented as points (for landmarks and simple obstacles) and lines (for linear obstacles between landmarks), which, after growing them, become circles and rounded rectangles, respectively. In our case, the growing size is the diameter of the robot. An example of how the obstacles are grown is shown in Figure 4.3. The Pilot uses a simple obstacle avoidance algorithm. It checks whether the robot is about to enter any of the forbidden areas associated to the obstacles. If the robot is in such a situation, the Pilot bids to modify the trajectory in order to avoid the obstacle. The modified trajectory is tangential to the grown obstacle to be avoided. Since obstacle avoidance is of maximal importance, the bid should be higher than the other systems. However, it should not be set to the highest possible value, 1, so that there is the possibility of adding a new system that overrides the Pilot (e.g. a teleoperation system). If the robot is in a safe area, the Pilot does not bid at all.

Regarding the bids for camera control, it is based on a function that increases the bid depending on the distance traveled since the last time the robot looked forward:

$$bid(look(ahead)) = \left(\frac{dist\_since\_last\_look}{max\_dist\_not\_looking} \right)^{exp} \tag{4.1}$$

where $max\_dist\_not\_looking$ is the maximum distance allowed to travel without looking ahead, and exp defines the increasing shape of the bidding function.

The Pilot also informs the Navigation system and the Visual Memory about any obstacle it detects. Whenever it detects a single obstacle (i.e. it bumps into it), it stores the obstacle's location in the Visual Memory, and checks whether it can be part of a larger linear obstacle. Such linear obstacles are detected when a series of single obstacles have been detected along the line connecting two landmarks and the distance between these obstacles is below a given threshold. If this is the case, the Pilot informs the Navigation system about the presence of a blocking obstacle between two landmarks.

4.2 Vision System

The Vision system is able to identify new landmarks in the vision field of the camera and is also able to recognize previously identified landmarks. This system does not bid for any of the available services. Again, although this system is not on the focus of the thesis, we have had to develop a simple Vision system for carrying out the experiments. A detailed description of the vision system developed to recognize indoor landmarks is given in Chapter 6.

The Vision system is simple but robust enough to correctly identify the landmarks. Thus, there is no uncertainty about the presence of a given landmark. However, there is some imprecision about its location, since the Vision system only gives approximate distance and angular information. To deal with this imprecision we use the fuzzy techniques described in Section 3.2.

The goal of this thesis is to develop a vision-based navigation system that does not use any specialized localization device (e.g. GPS) nor odometric information. However, we found that it was very restricting for the Navigation system to use only the visual information available after processing each viewframe. Firstly, because it is very difficult to have more than three landmarks on the view field, since it is very narrow, and the beta-coefficient system needs to have at least four visible landmarks in order to create a new β-unit. But even if four landmarks were in the view field, they would probably be highly collinear, which is not a good configuration for creating β-units. Secondly, it was a very unrealistic behavior to completely forget the landmarks that were not in the view field, even though they had been recently seen. We thought that adding the ability of remembering what has been previously seen would improve the behavior of the robot. Moreover, as it has already been mentioned, we want the robot to imitate the navigational behavior of humans and other animals, and we certainly have the ability of remembering what has been recently seen. A short term memory, called *Visual Memory*, implements this ability, and it is part of the Vision system.

4.2.1 Visual Memory

The Visual Memory stores landmarks and detected obstacles, with their location constantly updated using odometric information. To deal with the imprecision in odometry we use, again, a fuzzy approach. The odometric information coming from the robot is indeed fuzzy information about its motion, used to recompute the location of the objects stored in the Visual Memory. The imprecision of this motion is higher when the robot turns, and lower if it moves straight.

As the robot moves, the imprecision on these locations grows unless the landmarks are recognized again by the Vision system (which obviously reduces their location's imprecision). When the imprecision about the location of a landmark reaches a given upper threshold, the landmark is removed from the Visual Memory. The idea behind this being that the Visual Memory only remembers those landmarks whose location is precise enough.

The information stored in the Visual Memory is treated by the Navigation system in the same way as the information coming from the Vision system. The only difference is that the Visual Memory will be more imprecise than the Vision system. The Pilot system also uses this information to avoid colliding with remembered obstacles and landmarks.

4.3 Navigation System

This thesis has been mainly motivated by this system. We have used the modular view inspiring the overall robot architecture in the design of the Navigation system. The overall activity of leading the robot to the target destination is decomposed into a set of simple tasks. Working with simple tasks instead of using a single large module carrying out the whole navigation process is the basis of *Behavior-based robotics*. The idea is to divide the overall behavior of the robot into simpler behaviors, each one with its own goal, acting in parallel. These simpler tasks are much easier to build and debug than a larger module, since we only have to focus on separately solving smaller problems. Moreover, it permits us to incrementally increase the complexity of the robotic system, that is, adding new capabilities, by simply adding new behaviors, without having to modify already existing code. A detailed description of Behavior-based architectures was given in Chapter 2.

The Navigation system is defined to be a multiagent system where each agent is competent in one of these tasks (see Figure 4.4). These agents must cooperate, since an isolated agent is not capable of moving the robot to the target, but they also compete, because different agents may want to perform conflicting actions. Again, we use the bidding mechanism to coordinate the agents. Each agent bids for services provided by other robot systems (Pilot and Vision systems), and an additional agent, the communication agent, gathers the different bids and determines which one to select at any given time. This agent is also responsible of all the communication between the Navigation system and the other systems of the robot. The coordination between the agents is also made through a common representation of the map. Agents consult the map and the Pilot and Vision systems provide information about the environment —position of landmarks, obstacles — which is used to update it.

The local decisions of the agents take the form of bids for services and are combined into a group decision: which set of compatible services to require, and hence, gives us a handle on the difficult combinatorial problem of deciding *what to do next*. In the next section we describe in detail the society of agents that models the navigation process.

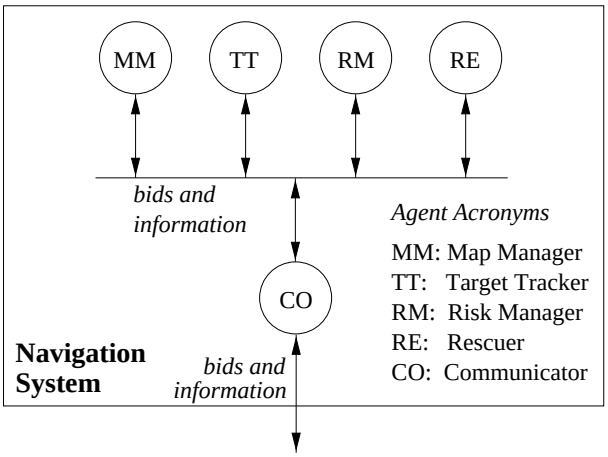


Figure 4.4: Multiagent view of the navigation system

4.4 The Group of Bidding Agents

In the model reported in this thesis we present a group of agents that take care of different tasks that, when coordinated through the bidding mechanism, provide the overall desired behavior of leading the robot to a target landmark. The tasks are:

- to *keep the information on the map* consistent and up-to-date,
- to *keep the target located* with minimum imprecision and *move towards it*,
- to *keep the risk* of losing the target low,
- to *recover* from blocked situations.

Four agents have been designed to fulfill each one of these goals (*Map Manager, Target Tracker, Risk Manager* and *Rescuer*, respectively), plus a *communicator* agent that is the responsible for communicating the Navigation system with the other robot systems (Pilot and Vision).

The actions that agents can bid for are:

- $\mathrm{Move}(direction)$, instructs the Pilot system to move the robot in a particular *direction*,
- Stop, instructs the Pilot system to stop the robot,
- $\mathrm{Look}(angle)$, instructs the Vision system to identify all the possible landmarks that can be found in the area at $angle$ radians from the current body orientation.

Finally, agents may ask one another with respect to the different knowledge they have. For instance, any agent in the society may request from the *Map Manager* to compute the location of the target or of a diverting target. Agents may also broadcast

messages to the rest of the agents in the society. For example, the *Rescuer* informs about the target to be reached, and the *Target Tracker* informs about the imprecision on the target's location.

In the next sections we describe each of the agents, and their code schemas can be found in Section 4.4.6.

4.4.1 Map Manager

This agent is responsible for maintaining the information of the explored environment in the topological map. The activity of this agent consists of processing the information associated with the incoming viewframes – expanding the graph, creating β-vectors, and asynchronously changing arcs' cost labels when informed by other robot systems. This agent uses the fuzzy beta-coefficient system described in Chapter 3 to build the map and answer questions about landmark positions.

The *Map Manager* is also responsible for computing the quality of the set of landmarks in the current viewframe, when required by the *Risk Manager*. This quality is a function of the collinearity of the landmarks. Having a set S of landmarks, their quality is computed as: $q_s = \max\{1 - Col(S') | S' \subseteq S, |S'| = 3\}$ where $Col(S')$ is computed using the equation 3.4.

This agent also computes diverting targets when asked for by the *Rescuer*. To do so, it uses the topological map, where all path costs are recorded, to compute which should be the next region to visit in order to reach the target. A description of the computation of diverting targets was already given in Chapter 3.

4.4.2 Target Tracker

The goal of this agent is to keep the target located at all times and move towards it. Ideally, the target should be always within the view field of the camera. If it is not, the imprecision associated to its location is computed by this agent using the information of the map. Actions of other systems are required to keep the imprecision as low as possible.

We model the imprecision as a function on the size of the angle arc, ϵ_θ, from the robot's current position, where the target is thought to be located. When the robot is sure of the position of the target (because it is in the current view field of the camera) we have a crisp direction and, hence, $\epsilon_\theta = 0$ and the imprecision is 0. If the target's location is obtained from the Visual Memory or computed by the *Map Manager*, ϵ_θ is computed as the size of the interval corresponding to the 70% α-cut of the fuzzy number representing the heading to the landmark. When the robot is completely lost, any direction can be correct, $\epsilon_\theta = 2\pi$, and the imprecision level is 1. Thus, the imprecision level is computed as:

$$I_a = \left(\frac{\epsilon_\theta}{2\pi}\right)^\beta \tag{4.2}$$

where β gives a particular increasing shape to the imprecision function. If β is much smaller than 1, the imprecision increases quickly as the imprecision in angle grows. For β values well over 1, imprecision will grow very slowly until the error angle gets very big.

The actions required by this agent are to move towards the target and to look towards the place where the target is assumed to be. The bids for moving towards the target start at a value κ_1 ($\leq$ 1) and decrease polynomially to 0, depending on a parameter α. The rationale for this is that when the imprecision about the target location is low, this agent is confident about the target's position and therefore bids high to move towards it. As the imprecision increases, this confidence decreases and so does the bid. Bids for looking at the target increase from 0 to a maximum of κ_2 (≤ 1) and then decrease again to 0. The rationale being that when the imprecision is low there is no urgency in looking to the target, since its location is known with high precision. This urgency starts to increase as the imprecision increases. When the imprecision reaches a level in which the agent has no confidence on the target location, it starts decreasing the bid so as to give the opportunity to other agents to win the bid. The equations involved are :

$$bid(move(\theta)) = \kappa_1(1 - I_a^{1/\alpha}) \tag{4.3}$$

$$bid(look(\theta)) = \kappa_2 \sin(\pi I_a) \tag{4.4}$$

where α controls how rapidly the moving bids decrease, and θ is the crisp angle where the target is thought to be. The bidding functions are shown in Figure 4.5.

This agent is constantly asking the *Map Manager* for the location of the target. When it receives an answer (obtaining θ and ϵ_θ), it computes the imprecision and informs the rest of the agents about it. If the *Target Tracker* is not informed about the target's location within a given time limit, it sets the imprecision level to 1.

The behavior described above is applied when the goal is to reach a single landmark. However, as mentioned in Section 3.4, the goal can also be to cross the edge connecting two landmarks (if the *Rescuer* has set it as the diverting target). In this latter case, this agent is constantly asking for the location of the two landmarks (thus, obtaining θ and ϵ_θ for each landmark) and computing their associated imprecision. The highest imprecision is used as I_a for computing the bidding values for moving and looking actions. It is also used to decide where the camera should look; it looks in the direction of the landmark with highest imprecision. Regarding the motion action, the agent bids to move in the direction of the angle between the two landmarks.

The *Target Tracker* is also the responsible for deciding whether the robot is *at target*. If the target is a single landmark, it considers that the robot has reached the target if the upper bound of the α-cut of level ϕ of the fuzzy number modeling the distance to the target is less than δ times the body size of the robot. The parameters ϕ and δ can be tuned to modify the accuracy of the agent. In the case of the target being an edge (between landmarks L_l and L_r), it checks whether the robot is on the desired side of line connecting the two landmarks. If the robot is on the left of the directed line through L_l and L_r, it is on the correct side, that is, the edge has been crossed. If it is on the right of the line, it means that the robot has not still crossed the edge.

4.4.3 Risk Manager

The goal of this agent is to keep the risk of losing the target as low as possible. While the *Target Tracker*'s goal is to locate the target by maintaining it in the camera's view field, this agent tries to keep a reasonable amount of known landmarks, as non collinear

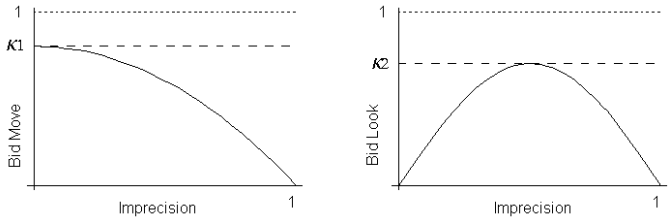


Figure 4.5: *Target Tracker*'s bidding functions

as possible, in the surroundings of the robot. The rationale is to have as many visible landmarks as possible so that the *Map Manager* is able to compute the location of the target using the beta-coefficient system when it is not visible nor in the Visual Memory. The fewer surrounding landmarks whose locations are known, the more risky is the current situation and the higher the probability of losing the target and getting lost. Also, the more collinear the landmarks, the higher the error in the location of the target, and thus, the higher the imprecision on its location.

We model the risk as a function that combines: 1) the number of landmarks ahead (elements in set A), 2) the number of landmarks around (elements in set B), and 3) their "collinearity quality" (q_A and q_B). As we have described, these qualities are computed by the *Map Manager*. A minimum risk of 0 is assessed when there are at least six visible landmarks in the direction of the movement and minimally collinear. Although the locations of only three landmarks are needed in order to use the beta-coefficient system, we want to have additional landmarks around the robot whose locations are known, so that there are more chances to compute the target's location. A maximum risk of 1 is assessed when there are no landmarks ahead nor around:

$$R = 1 - \min\left(1, q_A\left(\frac{|A|}{6}\right)^{\gamma_A} + q_B\left(\frac{|B|}{6}\right)^{\gamma_B}\right) \tag{4.5}$$

The values γ_A and γ_B determine the relative importance of the situation of landmarks (ahead or around).

Given that the robot cannot decrease the collinearity of the visible landmarks, the only way to decrease the risk level is by increasing the number of landmarks ahead and around. Having more landmarks, besides increasing $|A|$ or $|B|$, also helps by possibly increasing the qualities q_A and q_B.

We encourage having landmarks ahead by bidding

$$bid\left(look\left(random\left(\left[-\frac{\pi}{4}, +\frac{\pi}{4}\right]\right)\right)\right) = \gamma_r \cdot R \tag{4.6}$$

for the action of looking at a random direction in front of the robot and trying to identify the landmarks in that area, if $|A| < 6$, and

$$bid\left(look\left(random\left(\left[+\frac{\pi}{4}, +\frac{7\pi}{4}\right]\right)\right)\right) = \gamma_r \cdot R^2 \tag{4.7}$$

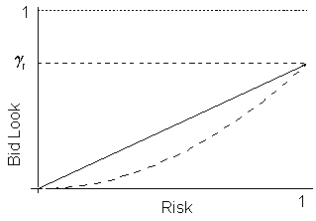


Figure 4.6: *Risk Manager*'s look bidding functions (look ahead -solid line- and look behind -dashed line-)

(which is obviously smaller than $\gamma_r \cdot R$) for the action of looking at a random direction around the robot and trying to identify landmarks, if $|B| < 6$, where γ_r is a parameter to control the maximum value of the bidding function. The bidding functions are shown in Figure 4.6.

The behavior of this agent also helps the *Map Manager* build the map when the robot is in an unexplored area. Since it bids for looking for landmarks when there are not many visible, its bids will be high, and thus new landmarks (if there are landmarks, obviously) will be identified and the map will be updated.

4.4.4 Rescuer

The goal of the *Rescuer* agent is to rescue the robot from problematic situations. These situations may happen due to two reasons. First, the Pilot can lead the robot to a position with a long obstacle ahead that cannot be easily avoided. Second, the imprecision of the location of the target may be too high (over a threshold $\overline{I}_a$).

If the robot gets blocked, this agent asks the *Map Manager* to compute a diverting target, and informs the rest of the agents about the new target. If the diverting target computed by the *Map Manager* is just a direction (this means that the robot should cross an edge containing a virtual landmark, as explained in Section 3.4), the *Rescuer* bids for turning the robot in the given direction. In order to have the robot moving in this direction for a short period of time, it sets the target to be a landmark that does not exist. However, the rest of the agents do not know that it does not exist, therefore, they behave as if it was an existing landmark. Thus, the *Map Manager* will not be able to compute its location when asked by the *Target Tracker*. This latter agent, after asking several times for the location of the target and not receiving any answer, will set the imprecision level to 1, which will cause the *Rescuer* to get active again. The rationale of this "trick" is that during the time the robot has been moving, it will have probably (and hopefully) recognized more landmarks so that the *Map Manager* can compute a better diverting target. Finally, if the *Map Manager* fails to compute a diverting target, the *Rescuer* bids for making the robot turn around (a random angle in $\pi \pm \frac{\pi}{6}$), hoping again that with the new direction it detects landmarks that help computing the location of the target or a diverting target. In case the current diverting target cannot be reached, this agent will ask for a new diverting target for the initial target.

On the other hand, if the imprecision of the target's location is too high, the agent

bids for stopping the motion and starting a visual scan around the robot, trying to detect as many landmarks as possible. The scan will stop when the imprecision of the location of the target has decreased to an acceptable level, either because it has been recognized by the Vision system or because its location has been computed by the *Map Manager* using other landmarks' locations. Since in this situation no obstacle has been detected, the *Rescuer* assumes that the path to the target is not blocked, so there will not be any target change. However, if at the end of the scanning the imprecision level is still too high, it will ask for a diverting target.

This agent also performs a visual scan at the very beginning, when the initial target is given, in order to detect some landmarks and start building the map before the robot begins moving to the target. Only after the scan is completed, this agent will inform the other agents what is the target to be reached.

The bidding values for the actions required by this agent are constant (parameter ω) and should be higher than those of the other agents ($\omega > max(\kappa_1, \kappa_2, \gamma_r)$), since it is absolutely necessary to execute the actions in order to continue the navigation to the target.

4.4.5 Communicator

The multiagent system implementing the navigation algorithm communicates with the remaining robot systems through the *Communicator* agent. This agent receives the information about the visible landmarks and obstacles detected, which is passed to the appropriate agents (*Map Manager* and *Rescuer*). This agent also receives bids for actions from the other agents and is responsible for determining which one to select and send as the Navigation system's bid. The actions required may be conflicting or not. For instance, an agent requiring the camera to look behind and another requiring it to identify a new landmark on the right, bid for conflicting actions, that is, actions that cannot be fulfilled at the same time. On the contrary, an agent requiring the robot to move forward, and an agent requiring the camera to look behind might be perfectly non-conflicting. It can be easily seen that the conflicts occur when the actions require the use of the same resource (robot motion or camera control). Thus, the request for actions will be separately treated depending on the resource required: Move and Stop actions on one side, and Look actions on the other. The *Communicator* agent receives the bids for the two different types of actions, and selects the moving action with the highest bid and the looking action with the highest bid. The resulting two action-bid pairs are sent to the Pilot and Vision system, respectively. This agent waits some time before processing the received bids, so that all the agents have time to send their bids. If, during this time window, an agent sends more than one bid for the same type of action, it replaces the previously sent bid. When the time window expires, the *Communicator* processes all the received bids and determines the winners.

As already mentioned, the bidding mechanism implements a competitive coordination mechanism. This mechanism has problems with selfish agents. The problem arises when there is one (or more) agents that always bids very high so that it wins all the bids, thus, not letting the other agents having their actions executed. In this case, there is no coordination at all between the agents, and it is very difficult, if not impossible, to achieve the goal of reaching the target destination. For instance, if we set the *Target*

Tracker to bid always higher than the Pilot system, the robot would not be able to avoid any obstacle, and would get stuck if any was encountered. To avoid such problem, the agents and systems should bid rationally, that is, bidding high only when the action is found to be the most appropriate for the current situation, and bidding low when it is not clear that the action will help, giving the opportunity to other agents to win the bid. Thus, special attention must be payed when designing the agents and their bidding functions.

To solve this problem we could use a more economic view of the bidding mechanism, assigning a limited credit to each agent, and allowing them to bid only if they had enough credit. With this new system there should also have to be a way to reward the agents. If not, they would run out of credit after some time and no agent would be able to bid. However, we face the credit assignment problem, that is, deciding when to give a reward and which agent or set of agents deserve to receive it. This problem is very common in multiagent learning systems, especially in Reinforcement Learning, and there is not a general solution for it. Each system uses an ad hoc solution for the task being learned. Other possible solutions would be to have a mechanism to evaluate the bidding of each agent, assigning them succeeding or failing bids, or some measure of trust, in order to take or not take into account their opinions. However, we would have again the credit assignment problem. Thus, in the multiagent system reported in this thesis we have designed the agents so that they bid rationally, leaving the exploration of these evaluation mechanisms as a line of future research.

4.4.6 Agents code schemas

In this section we present the code schemas for the agents *Map Manager*, *Target Tracker*, *Risk Manager* and *Rescuer*, and also for the Pilot system. The schemas have some parameters, such as the target that has to be reached, its initial heading, and some other particular parameters for each agent (bidding function parameters, thresholds...). These particular parameters define the behavior of the agents, and thereby, the overall behavior of the robot. Varying the values of the parameters, we may obtain better or worst navigation performances, and we may also adjust the conservativeness or riskiness of the robot. Thus, appropriately tuning these parameters is very important. In the next chapter we explore the use of learning techniques in order to do such parameter tuning.

When describing the algorithm schemas, the speech acts will appear as expressions in a KQML-style language [26]. Agents refer to themselves by the special symbol "self". When referring to all the agents of the society, they use the symbol "all".

Agents have a hybrid architecture. We will use the following construct to model the reactive component of agents:

On *condition* **do** *action*

Whenever the *condition* holds (typically an illocution arriving to the agent), the *action* is executed immediately. The illocutions used by the agents are the following: $ask(asking\_agent, asked\_agent, question)$ and $inform(informing\_agent, informed\_agent, information)$.

System Pilot(ν,max_dist_not_looking,exp) =

Begin deliberative
Repeat
inform(self,Vision System,odometric_information)
$\langle$avoid,$\theta\rangle$:= *avoid_obstacles_of_Visual_Memory()*
If avoid **then** inform(self,Coord,$\{(\mathrm{Move}(\theta),\nu)\}$)
inform$\left(\text{self,Coord,}\left\{\left(\mathrm{Look}(0),\left(\frac{dist_since_last_look}{max_dist_not_looking}\right)^{exp}\right)\right\}\right)$
Until
End deliberative

Begin reactive
On bumpers_active **do**
backup_safe_distance()
$\langle$obstacle_detected,$L_1, L_2\rangle$:= *update_Visual_Memory()*
If obstacle_detected **then** inform(self,NavigationSystem,obstacle(L_1, L_2))

On inform(VisionSystem,self,current_view(CV) **do**
$\langle$avoid,$\theta\rangle$:= *avoid_obstacles(CV)*
If avoid **then** inform(self,Coord,$\{(\mathrm{Move}(\theta),\nu)\}$)
End reactive

End system

System NavigationSystem($\alpha, \beta, \kappa_1, \kappa_2, \phi, \delta, \gamma_A, \gamma_B, \gamma_r, \overline{I}_a, \omega$) =

Agent MM() =

Begin reactive
On inform(CO,self,current_view(CV)) **do**
update_map(CV)

On inform(CO,self,obstacle(L_1, L_2)) **do**
update_obstacle(L_1, L_2)

On ask(X,self,position-landmark?(L)) **do**
$\langle \theta, \epsilon_\theta, d, \epsilon_d \rangle$:= *compute_landmark_position(L)*
inform(self,X,position-landmark($L, \theta, \epsilon_\theta, d, \epsilon_d$))

On ask(X,self,position-landmarks?(L_1, L_2)) **do**
$\langle \theta_1, \epsilon_{\theta_1}, d_1, \epsilon_{d_1} \rangle$:= *compute_landmark_position(L_1)*
$\langle \theta_2, \epsilon_{\theta_2}, d_2, \epsilon_{d_1} \rangle$:= *compute_landmark_position(L_2)*
inform(self,X,position-landmarks($L_1, \theta_1, \epsilon_{\theta_1}, d_1, \epsilon_{d_1}, L_2, \theta_2, \epsilon_{\theta_2}, d_2, \epsilon_{d_2}$))

On ask(X,self,landmarks-quality?) **do**
$\langle |A|, |B|, q_A, q_B \rangle$:= *compute_landmarks_quality()*
inform(self,X,landmarks-quality($|A|, |B|, q_A, q_B$))

On ask(X,self,diverting-target?(L)) **do**
$\langle T, L_l, L_r, \theta, type \rangle$:= *compute_diverting_target(L)*
If type=landmark **then**
inform(self,X,diverting-target(T))
else if type=edge **then**
inform(self,X,diverting-edge(L_l, L_r))
else if type=direction **then**
inform(self,X,diverting-direction(θ))
else if type=failed **then**
inform(self,X,diverting-target-failed)
End reactive
End agent

Agent TT($\alpha, \beta, \kappa_1, \kappa_2, \phi, \delta$) =
Begin deliberative
target_set := false
initial_target_reached := false
Repeat
If target_set **then**
If target_type = landmark **then**
ask(self,MM,position-landmark?(target))
else
ask(self,MM,position-landmarks?(EL_l,EL_r))
endif
endif
Until initial_target_reached
End deliberative

Begin reactive
On inform(RE,self,initial-target(T)) **do**
target_set := true
target_type := landmark
initial_target := T
target := initial_target

On inform(RE,self,target(T)) **do**
target_type := landmark
target := T

On inform(RE,self,target(L_l,L_r)) **do**
target_type := edge
$\langle EL_l, EL_r \rangle := \langle L_l, L_r \rangle$

On inform(MM,self,position-landmark(target,θ, ϵ_θ,dist,ϵ_{dist})) **do**
$I_a := (\frac{\epsilon_\theta}{2\pi})^\beta$
inform(self,all,imprecision(I_a))
inform(self,CO, $\{(\text{Move}(\theta), \kappa_1(1 - (I_a^{1/\alpha}))), (\text{Look}(\theta), \kappa_2 \sin(\pi I_a))\}$)
[min,max] := $\{dist\}_\phi$
at_target := max $\leq \delta$*bodyshape
If at_target **then** inform(self,all,at-target(target))

On inform(MM,self,position-landmarks($EL_l, \theta_l, \epsilon_{\theta_l}, d_l, \epsilon_{d_l}$, $EL_r, \theta_r, \epsilon_{\theta_r}, d_r, \epsilon_{d_r}$)) **do**
$I_a^l := (\frac{\epsilon_\theta^l}{2\pi})^\beta$
$I_a^r := (\frac{\epsilon_\theta}{2\pi})^\beta$
$I_a := max(I_a^l, I_a^r)$
angle_move := $(\theta_l + \theta_r)/2$
If $I_a^l > I_a^r$ **then** angle_look := θ_l
else angle_look := θ_r
inform(self,all,imprecision(I_a))
inform(self,CO, $\{(\text{Move}(angle\_move), \kappa_1(1 - (I_a^{1/\alpha})))$
$(\text{Look}(angle\_look), \kappa_2 \sin(\pi I_a))\}$)
edge_crossed := check_edge_crossed(θ_l, θ_r)
If edge_crossed **then** inform(self,all,edge-crossed(EL_l,EL_r))

On inform(self,self,at-target(initial_target)) **do**
initial_target_reached := true
End reactive
End agent

Agent RM($\gamma_A, \gamma_B, \gamma_r$) =

Begin deliberative
target_set := false
initial_target_reached := false
Repeat
If target_set **then**
ask(self,MM,landmarks-quality?)
endif
Until initial_target_reached
End deliberative

Begin reactive
On inform(RE,self,initial-target(T)) **do**
target_set := true
initial_target := T

On inform(MM,self,landmarks-quality($|A|$,$|B|$,q_A,q_B)) **do**
$R := 1 - \min\left(1, q_A\left(\frac{|A|}{6}\right)^{\gamma_A} + q_B\left(\frac{|B|}{6}\right)^{\gamma_B}\right)$
If $|A| < 6$ **then**
inform(self,CO,$\{(\text{Look}(random\_angle\left(\left[-\frac{\pi}{4}, +\frac{\pi}{4}\right]\right), \gamma_r R)\})$
else if $|B| < 6$ **then**
inform(self,CO,$\{(\text{Look}(random\_angle\left(\left[+\frac{\pi}{4}, +\frac{7\pi}{4}\right]\right), \gamma_r R^2)\})$
endif

On inform(TT,self,at-target(initial_target)) **do**
initial_target_reached := true

End reactive

End agent

Agent RE($\overline{I}_a, \omega$) =
Begin reactive
On inform(CO,self,new-target(T)) **do**
initial_scan()
inform(self,all,initial-target(T))

On inform(CO,self,Blocked) **do**
ask(self,MM,diverting-target?(initial_target))

On (inform(TT, self, imprecision(I_a)) **and** ($I_a > \overline{I}_a$)) **do**
$angle := compute\_scan\_angle()$
If $scan\_finished(angle)$ **then**
ask(self,MM,diverting-target?(initial_target))
else
inform(self,CO,$\{(\mathrm{Stop}, \omega), (\mathrm{Look}(angle), \omega)\}$)

On inform(TT,self,at-target(T)) **or** inform(TT,self,edge-crossed(L_l, L_r)) **do**
target := initial_target
inform(self, all, target(target))

On inform(MM,self,diverting-target(T)) **do**
inform(self, all, target(T))
target := T

On inform(MM,self,diverting-edge(L_l, L_r)) **do**
inform(self, all, target(L_l, L_r))

On inform(MM,self,diverting-direction(θ)) **do**
inform(self, all, target(fake_target))
inform(self,CO,$\{(\mathrm{Move}(\theta), \omega)\}$)

End reactive

End agent

End system

4.5 Future Work

We should explore the feasibility of using an economic view of the bidding mechanism, as mentioned in Section 4.4.5, and analyze how to solve the difficult problem of credit assignment.

The design of each one of the agents of the Navigation system should be revised according to the results obtained through the experimentation. This revision could range from simple tuning of some of the agents' behavior to the inclusion of new agents. Some of this changes will be discussed in Chapter 6, devoted to the experimentation with a real robot.

Chapter 5

Simulation Results

In this chapter we describe the experiments we have carried out through simulation. We have used simulation for three different tasks: firstly, to check that the multiagent Navigation system we have designed works properly; secondly, we have applied Reinforcement Learning techniques in order to learn a policy on the use of the camera; and finally, we have used a Genetic Algorithm approach to tune the parameters of the agents in the Navigation system.

For these different tasks, we have used two simulators. We started using the Webots[1] simulator. On this simulator we implemented the Navigation system and we also used it for the Reinforcement Learning task. However, we found some problems with the Webots simulator, mainly related to batch execution, which made the experimentation very slow. Although we were able to get results when used for Reinforcement Learning, we decided to develop our own simulator, to do extensive simulation with no problems. We used this new simulator to run again the multiagent Navigation system, and for the Genetic Algorithm approach to tune the parameters.

5.1 The Simulated System

It has to be pointed out that the overall system (that is, the Navigation, Pilot and Vision systems) used in the simulations is not exactly the same as the one described in the previous chapter (also described in [13]). Since the beginning of this research, four years ago, the Navigation, Pilot and Vision systems have been evolving (agents of the Navigation system have been added, modified and removed, and the capabilities of the Pilot and Vision systems have also changed) until we have reached what, by now, is the definitive version, which has just been described. This evolution has been guided by the experimentation, both on simulation and with the real robot. The simulation experiments described in this chapter show the performance of a previous version of our system [59, 12].

One of the main differences between the simulated system and the definitive one is that in the simulated one the Vision system did not provide information about the

[1] From Cyberbotics, http://www.cyberbotics.com

distance to the visible landmarks; it provided the Navigation system only with angular information. Moreover, the simulated Vision system had no range limitation, that is, it could identify any landmark, no matter how far it was, as long as it was in the view field of the camera. Obviously, this does not hold on the real Vision system.

Due to this lack of distance information, the *Map Manager* agent had to compute the distance to the landmarks using the change in angle of each landmark on successive viewframes. Since the change in angle can vary very little for the landmark the robot is going towards (i.e. the target), it was very difficult to accurately compute the distance to the target. In the simulated system, there was an additional agent, the *Distance Estimator*, that helped on computing the distance to the target. The role of this agent was to move the robot orthogonally with respect to the line connecting the robot and the target landmark while pointing the camera in the direction of the target, so that the change in angle was maximal, permitting the *Map Manager* to compute the distance accurately. The *Distance Estimator* agent computed the imprecision associated to the distance to the target. This imprecision is computed as $I_d = 1 - 1/e^{\kappa\epsilon_t}$, where κ is a parameter to control the shape of the function, and ϵ_t is the error in distance, and, similarly to what the *Target Tracker* does, it is computed as the size of the interval corresponding to the 70% α-cut of the fuzzy number representing the distance to the target. The *Distance Estimator* agent bids were a function on this imprecision. If the imprecision was high, it bid high to move the robot orthogonally, so the distance to the target could be computed with a lower error.. On the other hand, if the imprecision was low, so were the bids. This agent played a very important role at the beginning of the navigation, since the distance to the target was unknown, and therefore, the imprecision maximal. Thus, the *Distance Estimator* would bid very high in order to let the *Map Manager* get a first estimate of the distance. This agent was also responsible for deciding if the robot had reached the target, since it had the distance information. On the definitive system, this is responsibility of the *Target Tracker*.

Another important difference is that the simulated system did not use Visual Memory. That is, the Navigation system was only informed about the landmarks currently visible within the view field of the camera. This restriction made it difficult to create "good" beta-units, since all the visible landmarks were within a narrow view field, and thus, very collinear.

The *Rescuer* agent also had some differences: apart from getting active when the robot was blocked and when the imprecision in the target's location was too high, it also got active when the risk (computed and broadcasted by the *Risk Manager*) was over a threshold. Furthermore, its behavior was to always visually scan the surroundings of the robot and, after that, ask for a diverting target, not taking into account the reason of its activation.

There were also differences on the Pilot system. Another partner on the project we are involved in was responsible of building the Pilot system. Therefore, initially, we did not focus on this system, and did not worry about how it was designed. As long as it was able to avoid the obstacles encountered in its way, its design did not affect at all our coordination mechanism nor the design of the agents. For this reason, we started using a built-in pilot system of the Webots simulator that used simulated sonar sensors in order to avoid obstacles. In the real robot, however, such sonar sensors are not available, and,

as explained in the previous chapter, the Pilot system we finally implemented is only able to detect obstacles by bumping into them.

A final difference is that the mapping and navigation method used was not as explained in Chapter 3. Firstly, the criterion used to select topological regions was based only on the collinearity of the region and its size, thus, permitting overlapping regions, and not assuring a complete representation of the environment. And secondly, the computed diverting targets were always single landmarks; the computation of edges as diverting targets was introduced after experimenting with the real robot.

Despite all these differences, the basic elements of our approach have not been drastically modified during the evolution of the system: the bidding coordination mechanism has not been changed at all, and the mapping method has experienced only slight modifications.

5.2 Multiagent Navigation System Simulation

The goal of simulation was to check whether our approach, that is, the architecture, the bidding coordination mechanism and the mapping method, could lead to a robust navigation system.

We implemented the agents of the Navigation system and tested the algorithm on the Webots simulator and in our own developed one. Each agent was executed as an independent thread, and they used shared memory for message passing. We also simulated the Pilot and Vision systems on both simulators. We set the parameters of each of the agents by hand. We first set their values intuitively, and slightly modified them after some simulation trials.

As a first step, we checked whether the bidding mechanism was able to adequately coordinate the agents of the Navigation system and the Pilot, so that the task of reaching the target was accomplished. The Pilot system used was not able to inform about the presence of long obstacles between landmarks, although it would avoid them. For this reason, we were not still checking the mapping and navigation capabilities of the system.

Figure 5.1 shows a navigation run in the Webots simulator. It shows the path followed by the robot from a starting point to a target landmark. The environment was composed by a set of landmarks (shown as circles), a river (the thick blue traversing line) with a couple of bridges, and some fences and other obstacles. These obstacles did not occlude the target landmark, so it was visible from any location of the environment. The task to be performed was to reach the target (at the left-hand side of the world) avoiding any obstacle encountered on the way.

At the very beginning, the distance to the target is unknown, so the *Distance Estimator* agent (DE) bids very high to move the robot orthogonally to the line connecting it to the target and looking to the target, so that the *Map Manager* can estimate the distance to the target. The *Target Tracker* agent (TT) bids for moving and looking towards the target, but the bids of DE are higher and the robot moves orthogonally. As the robot moves, the *Map Manager* computes the distance to the target, and the imprecision computed by the DE decreases, causing its bids also to decay. At a given point, the bids of TT are higher than those of DE, and the robot starts going towards the tar-

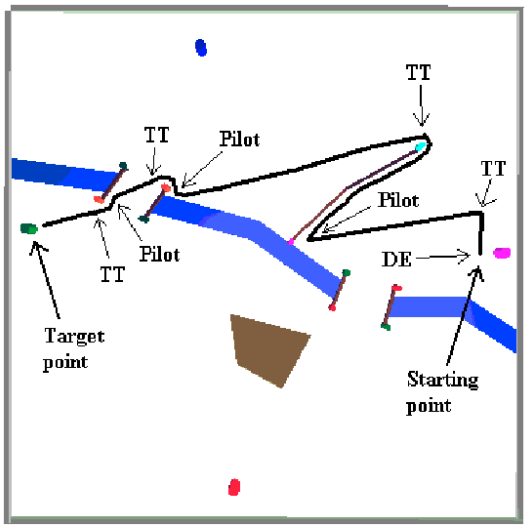


Figure 5.1: Robot's path from starting point to the target

get. Since there are no obstacles around, the Pilot does not bid at all. However, after some advance, the robot encounters an obstacle, and the Pilot bids very high to avoid it, surpassing the bids of TT and DE. When the obstacle has been totally avoided, the Pilot stops bidding, the bids of TT win again, and the robot moves towards the target. This situation is repeated a couple of times until the robot finally reaches the target.

Although the environment used in this first step was simple, mainly because of the constant visibility of the target, simulations showed that the bidding coordination mechanism worked properly, since it was able to coordinate the different agents and the Pilot.

The next step was to test the mapping and navigation capabilities of the Navigation system. In this step we used our own developed simulator, with a better Pilot system, capable of informing about the linear obstacles between landmarks, and with more realistic environments including occluding obstacles, so that the target was not visible all the time.

In Figure 5.2 we see how the Navigation system computes diverting targets for reaching the initial target when this is lost. In this environment, filled polygons are occluding obstacles, and empty ones are non-occluding ones, thus, permitting the visibility of the target from the starting point. At point A, it sees the target and starts going towards it. However, at point B, it detects an obstacle, so the Pilot forces the robot to turn. When it reaches point C, it cannot see the target anymore, as it is behind an occluding obstacle. At this point, a diverting target is computed (in this case, landmark 30 is selected). The robot starts going to this diverting target. Once reached (point D), a new diverting target is computed (landmark 38 is selected), and the robot goes toward it. At point E, after reaching the current diverting target, a new one is computed (landmark 12), which is reached at point F. From this point, it sees the initial target again,

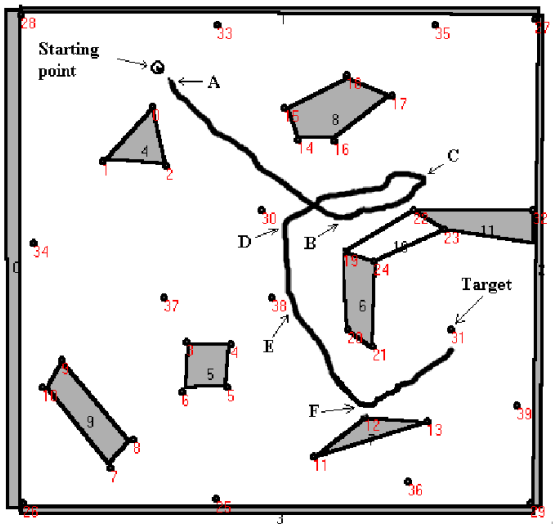


Figure 5.2: Computing diverting targets

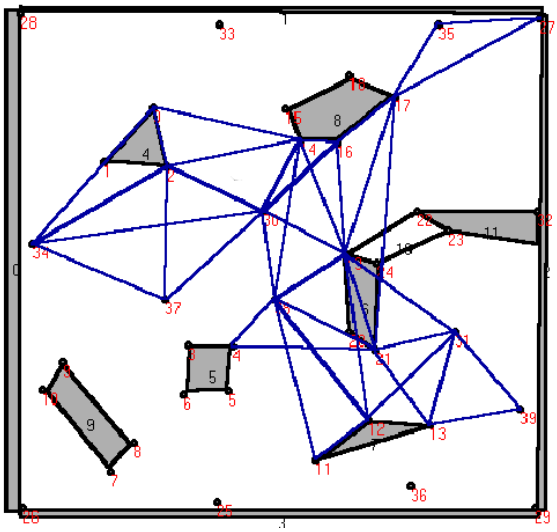


Figure 5.3: Associated map

goes straight towards it, and finally reaches the target.

Someone may ask why the Navigation system computed so many diverting targets, instead of trying to go towards the initial target more frequently. The reason was that the risk was too high very often. This was because of the narrow view field of the camera and the fact that the system was not using Visual Memory, thus, having too few landmarks in sight very often. Although the performance was good enough – the robot reached the target – this behavior of constantly computing diverting targets was not what we really wanted. Moreover, in the situation of the robot being in an area with very few landmarks, possibly seeing only the target, the risk would be very high, but it would not be a wise decision to stop going towards the target and, instead, compute a diverting target. That is why the *Rescuer* agent was modified so that it did not take into account the risk, as presented in the previous chapter.

In Figure 5.3 the map generated while reaching the target is shown. Although internally the *Map Manager* agent stores the map as a graph, here, for clarity, we show the triangular regions corresponding to the nodes of this graph. As can be seen, the map has many overlapping regions, unconnected regions and regions with obstacles inside. Obviously, it is not a very good representation of the environment. In order to obtain a better map of the environment, we modified the mapping algorithm so that it included the constraints presented in Chapter 3. As will be seen in the experimentation with the real robot (Chapter 6), the modified mapping algorithm obtains much better maps.

Although in the simulation we simplified the task in comparison to navigating through a real environment (the Vision system worked perfectly, without any limitation on its view range, the Pilot used sonars for obstacle avoidance), the results obtained, showing that the coordination and mapping worked well, were very promising and encouraged us to keep working on the refinement of the system in order to test it on the real robot. However, even though the main experimentation was to be done with the real robot, we still employed simulation to apply Machine Learning techniques in order to automatically tune the parameters and obtain better performance. In the following sections we describe how we have applied these techniques.

5.3 Reinforcement Learning

As mentioned, each of the agents within the Navigation system has a bidding function that is controlled by a set of internal parameters. These parameters need to be tuned in order to achieve the best performance of the Navigation system and of the overall system. Although, as shown in the previous section, we achieved good results with hand-tuned parameters, we wanted to explore if there were other parameter configurations that led to better performance of the system. Adjusting these parameters manually can be very difficult, particularly because of the tradeoffs confronting the top-level agents. An alternative to manual tuning is to employ Machine Learning techniques, specifically Reinforcement Learning methods [64]. In this section, we describe some experiments to test the feasibility of applying Reinforcement Learning within this multiagent system.

Reinforcement Learning is one of the most commonly used learning techniques in Robotics. In Behavior-based architectures learning can be applied at two levels: at the coordination level, where the goal is to apply learning to the coordination system

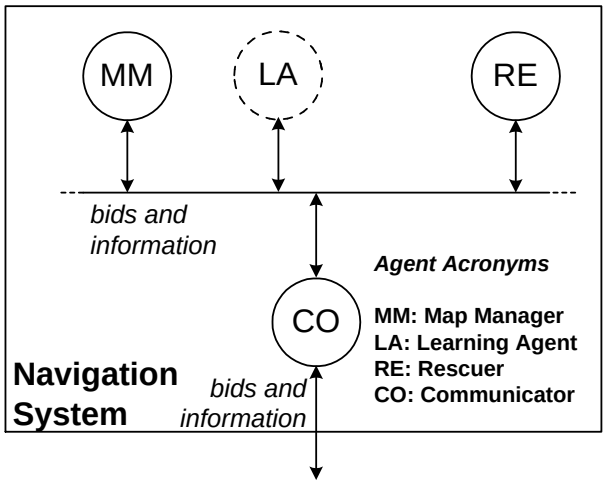


Figure 5.4: Modified navigation system, with the new agent

[44, 28], or at the behavior level, where the goal is to apply learning to the individual behaviors of the system [45, 14]. In our case, we have taken the latter approach [10, 11].

Ideally, we would like to apply Reinforcement Learning to tune all of the parameters of all of the agents in the system. However, this is a very difficult problem, and it is not clear that Reinforcement Learning is the best solution at all levels of the system. Instead, we have chosen to focus on a particular learning problem within the Navigation system. Reinforcement Learning is most needed and most appropriate in cases where there is a complex, quantitative tradeoff between behaviors. In such cases, manual tuning is difficult, and the quantitative criterion of maximizing expected reward, which is the goal of Reinforcement Learning, permits us to represent the tradeoff nicely.

Within the Navigation system, such a tradeoff exists between the *Target Tracker* agent, the *Risk Manager*, and the *Distance Estimator* — recall that we use the initial version of the system, as described in Section 5.1. The *Target Tracker* wants to know the exact heading and distance to the target at all times. This can be achieved by pointing the camera at the target and moving towards it. The *Risk Manager* wants to ensure that the robot is surrounded by a rich network of landmarks so that the robot does not get lost. This can be achieved by pointing the camera in various directions around the robot to identify and track landmarks. Finally, the *Distance Estimator* seeks to know accurate distances to the target landmark. This can be achieved by pointing the camera in the direction of the target while moving the robot orthogonally to the direction of the target. In addition to this conflict, the Navigation system must not monopolize the camera, because the Pilot needs to use it for obstacle avoidance.

Instead of trying to learn the appropriate values for each of the parameters of these agents, we propose to replace the *Target Tracker*, the *Risk Manager*, and the *Distance Estimator* by a new *Learning Agent* that learns its behavior through Reinforcement Learning. We formulate the reward function for this agent so that it is rewarded for

reaching the current target location while minimizing the use of the camera. The two remaining agents have very different roles. The *Map Manager* maintains the beta-coefficient map, but does not bid on actions. The only remaining bidding agent is the *Rescuer*, which is responsible for the higher-level choice of diverting targets whenever the robot becomes blocked. This activity is better-implemented by path planning algorithms than by Reinforcement Learning, so we have not included the *Rescuer*'s responsibilities within the *Learning Agent*. The modified architecture for the Navigation system is shown in Figure 5.4.

5.3.1 The Task to be Learned

The task confronting the *Learning Agent* is to choose actions (for both motion and vision) in order to reach the current target location while minimizing the use of the camera. The *Map Manager* informs the *Learning Agent* about the target location. If the robot becomes blocked, the *Rescuer* will ask the *Map Manager* for a new target (a diverting target), and then the *Learning Agent* will take control and choose actions to reach that new target. Once the diverting target is reached, the *Rescuer* may be able to set the current target to be the original goal, and then the *Learning Agent* will attempt to move to that target (and hence, solve the original task).

5.3.2 The Reinforcement Learning Algorithm

There are two general types of Reinforcement Learning algorithms: Model-based and Model-free. Model-based algorithms learn a transition model $P(s'|s,a)$ for the environment, where s is the state of the environment at time t, a is an action to be executed, and s' is the resulting state of the environment at time $t+1$. Model-based algorithms also learn a reward model $R(s,a,s')$, which gives the expected one-step reward of performing action a in state s and making a transition to state s'. Once these models have been learned, dynamic programming algorithms [6] can be applied to compute the optimal value function V^* and the optimal policy π^* for choosing actions.

In contrast, model-free methods (such as Q learning and SARSA(λ)) directly learn a value function V^* by repeatedly interacting with the environment without first learning transition or reward models. They rely on the environment to "model itself". For robot learning, however, model-free methods are impractical, because they require many more interactions with the environment to obtain good results. They make sense in simulated worlds where the cost of performing an action can be much less than the cost of storing the transition and reward models, particularly if the environment is evolving over time. But the cost of performing an experimental action with a real robot is very high.

Hence, for our experiments, we have chosen the model-based algorithm known as Prioritized Sweeping [49]. Prioritized Sweeping works as follows. At each time step, the learner observes the state s of the environment, chooses an action a, performs the action, receives a one-step reward r, and observes the resulting state s'. The learner then updates its estimate of $P(s'|s,a)$ and of $R(s,a,s')$ using the observed result state s' and the observed reward r. Finally, the learner performs the k most important Bellman

backups to update its estimate of the value function V. A Bellman backup in state s is computed as follows:

$$V(s) := \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + V(s')]$$

This is essentially a one-step lookahead that considers all possible actions a and all possible resulting states s', computes the expected backed-up value of each a, and assigns the maximum such value to be the new estimate of V at state s.

Prioritized Sweeping maintains a maximizing priority queue of states in which it believes a Bellman backup should be performed. First, it performs a Bellman backup for the most recent state s. In each Bellman backup, it computes the change in the value $V(s)$ resulting from the backup:

$$\Delta(s) = \left| V(s) - \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + V(s')] \right|$$

After performing the Bellman backup, Prioritized Sweeping considers all states s^- that are known predecessors of s, and computes the potential impact C of the change in $V(s)$ on the change in the value of s^- according to

$$C(s^-) = \sum_a P(s|s^-,a)\Delta(s)$$

It then places the state s^- on the priority queue with priority $C(s^-)$. Finally, Prioritized Sweeping performs $k-1$ iterations in which it pops off the state with the maximum potential impact, performs a Bellman backup in that state, and then computes the potential impact of that backup on all predecessor states. In our experiments, $k = 5$. (In our implementation, we actually use the state-action, or Q, representation of the value function rather than the state value function V. We have described the method using V in order to simplify the presentation.)

Prioritized Sweeping is essentially an incremental form of value iteration, in which the most important updates are performed first. Because every interaction with the environment is applied to update the model, Prioritized Sweeping makes maximum use of all of its experience with the environment. Prioritized Sweeping is an "off-policy" learning algorithm. During the learning process, any exploration policy can be employed to choose actions to execute. If the exploration policy guarantees to choose every action in every state several times, then Prioritized Sweeping will converge to the optimal action-selection policy. We employ ϵ-greedy exploration. In this form of exploration, when the robot reaches state s, it executes a random action with probability ϵ. With probability $1-\epsilon$, it executes the action that is believed to be optimal (according to the current value function V). Ties are broken randomly.

We represent both the transition model $P(s'|s,a)$ and the reward model $R(s,a,s')$ by three-dimensional matrices with one cell for each combination of s, s', and a. This technique will only work if the state and action spaces are small. There are two reasons for this. First, the tables must fit into memory. Second, the time required for learning is proportional to the number of cells in these tables, because the *Learning Agent* must

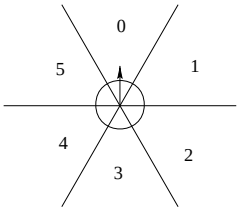


Figure 5.5: Division of environment in sectors. The arrow shows the direction in which the robot is facing (direction of motion, not direction of gaze)

experience multiple visits to each state s so that it can perform each action a several times and gather enough data to estimate $P(s'|s,a)$ and $R(s,a,s')$. Hence, the most challenging aspect of applying Reinforcement Learning is the proper design of the state representation.

State Representation

We want the *Learning Agent* to learn a general policy that works for any environment, independently of the locations of the landmarks and targets. Hence, our state representation must not directly employ the locations of the landmarks. Moreover, the robot cannot directly observe the complete state of the environment, which would include the location of the robot, all obstacles, and all landmarks! Instead, the task of the robot is to learn, under conditions of incomplete knowledge, about the locations of obstacles, landmarks, and targets.

State spaces that encode incomplete knowledge are known as "belief state spaces" [15]. The purpose of a belief state representation is to capture the current *state of knowledge* of the agent, rather than the current state of the external world. In our case, the *Learning Agent* is trying to move from a starting belief state in which it knows nothing to a goal belief state in which it is confident that it is located at the target location. Along the way, it seeks to avoid getting lost (which is a belief state in which it does not know its location relative to the target position).

To explain our state representation, we begin by defining a set of belief state variables. Then we explain how these are discretized to provide a small set of features each taking on a small set of values, so that $P(s'|s,a)$ and $R(s,a,s')$ can be represented with small tables.

At any given point in time, the headings to all objects (landmarks and the target position) are divided into six sectors. The field of view of the robot is 60 degrees, so at any point in time, the robot can observe one sector, see Figure 5.5. For each sector, we represent information about the number of landmarks believed to be in that sector and the precision of our beliefs about their headings and distances. This information is gathered from an initial version of the Visual Memory that constantly updates the location of the seen landmarks, and to which the *Learning Agent* has access.

Given these sectors, the following state variables can be defined:

- Distance to target, and its imprecision, $D(t), I_d(t)$

- Heading to target, and its imprecision, $H(t), I_h(t)$
- The landmarks in each sector, $L(s) = \{l_1, ..., l_{n_s}\}$
- Number of landmarks in each sector, $N(s) = \min(4, |L(s)|)$
- Average imprecision of landmarks in each sector, $\overline{I}(s) = \frac{1}{N(s)} \sum_{l \in Best(4,L(s))} I(l)$

We now explain each of these. The distance $D(l)$ to a landmark (or $D(t)$ to the target) is a fuzzy number in the range $[0, \infty]$. The heading to a landmark $H(l)$ (or $H(t)$ to the target) is a fuzzy number with range $[0, 2\pi]$. For each of these, its imprecision ($I_d(l)$ for distance, $I_h(l)$ for heading) is defined by taking the size of the interval corresponding to the 70% α-cut of the fuzzy number.

The imprecision of a landmark is computed using the equation 3.3 already given in Section 3.2.2:

$$I(l) = \lambda \cdot \tanh(\beta \cdot I_d(l)) + (1 - \lambda) \cdot \frac{I_h(l)}{2\pi}$$

For an explanation of the equation see the mentioned section.

We summarize the agent's knowledge of the landmarks in each sector by averaging the imprecision of the four most-precisely-known landmarks. The function $Best : N \times 2^L \rightarrow 2^L$ selects a subset, $B = Best(n, L)$, of a group of landmarks, $L = \{l_1, ..., l_m\}$, such that $|B| \leq n \wedge \forall_{l \in B} \forall_{l' \in L-B} I(l) \leq I(l')$. Having 4 landmarks in one sector is already very good, since only 3 landmarks are needed to use the beta-coefficient system network. Furthermore, we do not want these measures to be affected by bad landmarks when we have some that are good enough. That is why we use $Best(4, L(s))$ when computing $\overline{I}(s)$.

Features

After computing these state variables, we combine and discretize them to define a small number of features each of which takes on a small number of values. These features define the state space, and they are used to access the tables $P(s'|s, a)$, $R(s, a, s')$ and $V(s)$ in the learning phase, and also to access $\pi(s)$ for policy exploitation.

We employ the following features:

- Target Distance, $D(t)$, discretized to 5 intervals.
- Target Location Imprecision: measure of imprecision on the location of the target, $I(t)$, discretized to 7 intervals.
- Landmark Count: average number of landmarks over the six sectors, $\overline{C} = \frac{1}{6} \sum_{s=0}^{5} N(s)$, discretized to 4 intervals.
- Landmark Imprecision: average imprecision of landmarks' locations in each sector, $\overline{I} = \frac{1}{6} \sum_{s=0}^{5} \overline{I}(s)$, discretized to 7 intervals.

This gives a total of 980 belief states.

Actions

Just as Reinforcement Learning requires careful design of the state space to ensure that it is compact, it also requires careful design of the action set to ensure that it is small but also sufficient for the robot to achieve its goals.

Physically, the robot is able to simultaneously perform two types of actions: *moving* actions and *looking* actions. Moving actions make the robot move in a given direction. Looking actions employ the camera to identify or track landmarks in the environment in specified sectors. The Vision system can either search for new landmarks or re-acquire already-detected landmarks, but it is not able to do both things at the same time, because different image processing routines are required for each. In either case, however, the Vision system returns the heading and distance to the landmarks it detects.

An additional constraint on the design of actions is that the Vision system is most effective when the robot is moving in certain directions relative to the landmarks being observed.

Given these constraints, we have designed the following set of actions for the *Learning Agent*:

- Move Blind (MB): move toward the target (i.e., in the direction in which the target is *believed* to be). Do not use the Vision system.
- Move and Look for Landmarks (MLL): move toward the target. Point the camera in the sector that contains the fewest number of known landmarks, and look for new landmarks in this sector.
- Move Orthogonally to Target (MOT): move orthogonally to the direction of the target. Point the camera at the target and attempt to improve the precision of the heading and distance to the target.
- Move and Verify Landmarks (MVL): move toward the target. Point the camera to the sector with the maximum imprecision, $\overline{I}$, and attempt to re-acquire known landmarks and measure their heading and distance more accurately.
- Move and Verify Target (MVT): move toward the target. Point the camera at the target and attempt to re-acquire it and measure its heading and distance more accurately.

These actions should affect the state variables as follows. All actions except MOT make the distance to the target decrease. MB makes all imprecisions grow. MLL should increase the number of detected landmarks. MOT should reduce the imprecision about the target's location, while MVL should reduce the overall imprecision. MVT also reduces the imprecision of the target's location, but not as much as MOT. All actions require that the heading to the target is known (at least approximately). The heading is chosen as the center of the fuzzy interval for $H(t)$. If the heading is completely unknown, the center of this interval is π. This causes the robot to "pace" back and forth, turning 180 degrees (π radians) each time an action is executed.

We have assigned an immediate reward to each action to reflect the load on the Vision system and the motion system. The rewards are negative, because they are costs.

MB is the cheapest action, since it does not use the camera. It has a reward of -1. MVL and MVT produce a reward of -5, since they make moderate demands on the Vision system. MOT gives a reward of -6, because it requires more motion in addition to the same image processing as MVL and MVT. Finally, MLL is the most expensive, with a reward of -10, because it must do extensive image processing to search for new landmarks and verify that they are robust to changes in viewpoint.

The system receives a reward of 0 when it reaches the target location. The Reinforcement Learning objective is to maximize the total reward. In this case, this is equivalent to minimizing the total cost of the actions taken to reach the target.

5.3.3 Experimentation

We have employed the Webots simulator to perform our experiments. The environment contains a set of landmarks, one of which is designated as the target. There is also a wall that surrounds the region in which the robot is navigating. The landmarks are the only objects in the environment. There are no obstacles, as obstacle avoidance is handled by the Pilot system. However, the robot can be blocked by the landmarks or by the wall. In each trial, the robot starts at a random location in this environment, and it has to reach the target. The trial terminates under three conditions: (a) if the robot reaches the target (and is confident that it has reached the target), (b) if the robot takes 500 steps without reaching the target, or (c) if the robot is blocked. When the trial is finished, the next one begins with another random initial location for the robot.

In order to see if the performance of the system improves after learning, we compared it with a hand-coded policy. The hand-coded policy used the same discretized features as the learning algorithm (Target Distance, Landmark Count, Landmark Imprecision and Target Location Imprecision). The following table shows the policy for choosing an action depending on the values of these features :

Target Distance	Landmark Count	Landmark Imprecision	Target Loc. Imprecision	Action
$high$	low	$*$	$*$	MLL
$high$	$\neg low$	$high$	$*$	MVL
$high$	$\neg low$	$\neg high$	$high$	MOT
$high$	$\neg low$	$\neg high$	$\neg high$	MB
$\neg high$	$*$	$high$	$high$	MVL
$\neg high$	$*$	$\neg high$	$high$	MVT
$very\ low$	$*$	$*$	$\neg high$	MVT
low	$*$	$*$	$\neg high$	MB

where $high$, low and $very\ low$ are defined as follows:

Variable	*very low*	*low*	*high*
Target Distance	< 1	≤ 2	> 2
Target Location Imprecision	–	< 5	≥ 5
Landmark Count	–	< 2	≥ 2
Landmark Imprecision	–	< 5	≥ 5

The reader should note that this hand-coded policy is not the same as the policy produced by the hand-coded bidding functions described in Chapter 4. We have chosen this policy because it allows us to debug and test the *Learning Agent* separately from the rest of the multi-agent system.

The *Learning Agent* was trained for 2000 simulated trials. At regular intervals, the learned value function was tested by placing the robot in 100 randomly-chosen starting locations, running one trial from each location, and measuring the total reward, the total number of actions, and whether the robot succeeded in reaching the target position. The same set of 100 starting locations was employed in each testing period. The hand-coded policy was also evaluated on these 100 starting locations.

First, let us consider the fraction of successful trials. Figure 5.6 shows that even after only 100 trials, the *Learning Agent* is already out-performing the hand-coded policy. After 2000 trials, the *Learning Agent* succeeds in reaching the target in 84 of the trials, compared to only 24 for the hand-coded policy. From these results we also see that our hand-coded policy was pretty bad. Although we could have tried to rewrite the policy to improve its performance, the results show that Reinforcement Learning can greatly help on solving complex tradeoffs, very difficult to handle manually.

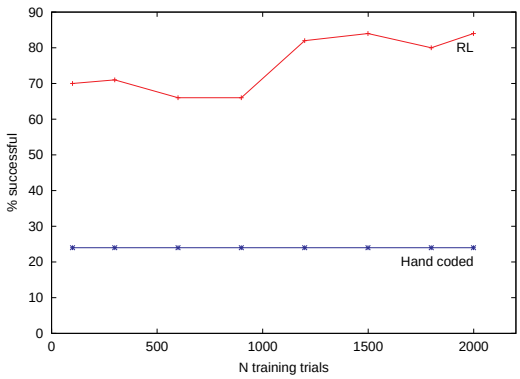


Figure 5.6: Number of successful test trials as a function of the amount of training

A second way of analyzing the performance of the *Learning Agent* is to compute the average reward per trial, the number of actions per trial, and the number of actions of each type. Table 5.1 displays this information after 2000 training trials. Each value is averaged over five test runs. The only difference between test runs is the random number seed for the Webots simulator. We see that while the hand-coded policy receives an average of -858 units of reward, the learned policy only receives -336 units, which is a huge improvement. In addition, the *Learning Agent* on the average only requires

Table 5.1: Comparison of the *Learning Agent* (LA) and the hand-coded policy (HC) after 2000 training trials.

	Reward per trial	Actions per trial	MB	MOT	MVT	MVL	MLL
HC	-858	153.33	4.94	18.59	0.52	121.96	7.32
LA	-336	49.95	11.41	6.52	5.61	4.97	21.43

50 steps to terminate a trial (reach the goal, become blocked, or execute 500 steps) compared to 153 steps for the hand-coded policy. Actually, the *Learning Agent* never terminates because of reaching the 500-step limit.

Table 5.1 contains other interesting information. In particular, we see that the *Learning Agent* has learned to perform fewer MOT and MVL actions and more MB, MVT, and MLL actions. Note particularly that the *Learning Agent* is executing an average of 11.4 MB (Move Blind) actions per trial, compared to only 4.9 for the hand-coded policy. One of the goals of applying Reinforcement Learning was to find a policy that freed the camera for use by the low-level obstacle avoidance routines, and this is exactly what has happened: the hand-coded policy uses the camera 96% of the time, while the *Learning Agent* uses it only 77% of the time. On the other hand, we were surprised to see that the *Learning Agent* chooses to execute the most expensive action, MLL, so often (21.4 times per trial, compared to only 7.3 times per trial for the hand-coded policy). Certainly, it has found that a mix of MLL and MB gives better reward than the combination of MVL and MOT that is produced by the hand-coded policy. The *Learning Agent* spends much more time looking for new landmarks and much less time verifying the direction and distance to known landmarks.

5.3.4 Future Work

Although the obtained results show that the *Learning Agent* has learned to select actions to resolve the complex camera tradeoff, we need to integrate it into the overall multi-agent system (as depicted in Figure 5.4), to see if the performance of the whole system is also improved. Even though the *Learning Agent* knows which actions it has to bid for (following the learn policy), it is not clear how its bidding function should be (e.g. constant, depending on the values of $V(s)$).

Some more further work will be focused on the design of the state and feature representation and the set of available actions. Asada et al. [5] proposed a solution for coping with the "state-action deviation problem", in which actions operate at a finer grain than the features can represent, having the effect that most actions appear to leave the state unchanged, and learning becomes impossible. We plan to evaluate the suitability of this approach in our experiments. Regarding the action set design, we found that the set of available actions was maybe too small and some more actions may be needed. We are working on an "action refinement" method [20] that exploits prior knowledge information about the similarity of actions to speed up the learning process. In this approach, the set of available actions is larger, but in order to not slow down the learning, the actions are grouped into subsets of similar actions. Early in the learning process, the Reinforcement Learning algorithm treats each subset of similar actions as a single "abstract" action, estimating $P(s'|s, a)$ not only from the execution of action a, but also

from the execution of its similar actions. This action abstraction is later on stopped, and then each action is treated on its own, thus, refining the values of $P(s'|s,a)$ learned with abstraction.

5.4 Evolving the Multiagent Navigation System

As we have already mentioned previously, our Navigation system is decomposed into a set of different agents that are responsible for different tasks. Each of these agents has certain parameters that affect its bidding behavior. Trying to manually find the best values for the parameters of the bidding functions is an extremely difficult task. In this section we describe the application of an evolutionary approach to do this optimization.

5.4.1 Navigation Tasks

For a given environment we consider two different navigation tasks. Each one of them with a different level of complexity. The best parameter set may change depending on the complexity of the task. We conjecture that the parameters found depend mainly on the complexity of the navigation task and not so much on the structure of the overall environment. This complexity is dependent, though not equal, to the cartographic complexity of the world in which the agent moves, and is based on the following factors:

1. Number of visible landmarks at any time
2. Density of obstacles in the region of navigation
3. Visibility of the target at any time

Using this notion of navigational complexity, the total space of all navigation tasks can be split into two representative classes: going towards the target free of obstacles, and reaching targets located behind obstacles. In our experiments we use clusters C_1 (encircled targets in Figure 5.7) and C_2 (encircled targets in Figure 5.8) as representatives of the two task complexity classes. The best parameter set is determined for both these classes. The aim of the experiments is to endow the Navigation system of the robot with the capability to switch between these two parameter sets according to the actual task complexity it is facing.

5.4.2 The Agents

Although a detailed description of the agents was already given in Chapter 4, as well as the description of the differences between the simulated system and the final system, (given at the beginning of this chapter), we review the parameters of each of the agents:

- **Target Tracker ($\alpha, \beta, \kappa_1, \kappa_2$)**
  - α: controls how rapidly the bids for moving towards the target decrease, $bid(move(\theta)) = \kappa_1(1 - I_a^{1/\alpha})$; high values of α make bids increase fast, while low values make bids increase slowly

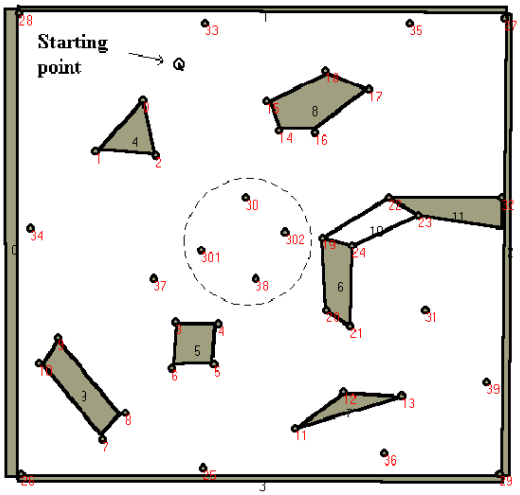


Figure 5.7: Cluster C1

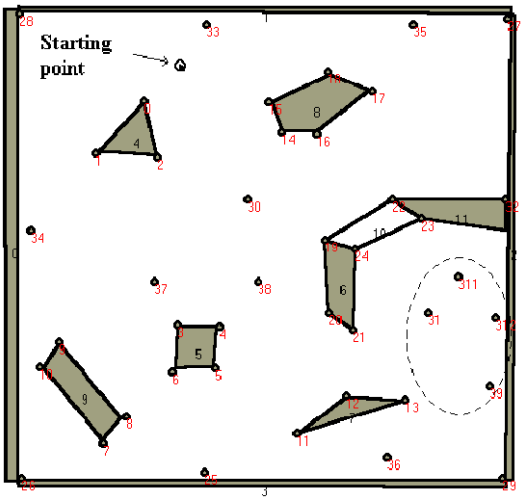


Figure 5.8: Cluster C2

  - β: controls the shape of the imprecision function, $I_a = \left(\frac{\epsilon_\theta}{2\pi}\right)^\beta$; high values make it increase slowly, while low values make it increase fast
  - $\kappa 1$: maximum value for moving actions bids
  - $\kappa 2$: maximum value for looking actions bids

- **Distance Estimator (κ, ϕ, δ)**
  - κ: controls the shape of the distance imprecision function, $I_d = 1 - 1/e^{\kappa\epsilon_t}$; high values of κ make the imprecision grow fast, while low values make it increase slowly
  - ϕ, δ: controls the *at target* computation; it considers that the robot has reached the target if the upper bound of the α-cut of level ϕ of the fuzzy number modeling the distance to the target is less than δ times the body size of the robot

- **Risk Manager ($\gamma_A, \gamma_B, \gamma_r$)**
  - γ_A, γ_B: control the relative importance of the position of landmarks, ahead and around, respectively, used in the risk computation,

$$R = 1 - \min\left(1, q_A\left(\frac{|A|}{6}\right)^{\gamma_A} + q_B\left(\frac{|B|}{6}\right)^{\gamma_B}\right)$$

  - γ_r: maximum value for looking actions bids

- **Rescuer ($\overline{I}_a, \overline{R}$)**
  - $\overline{I}_a$: imprecision threshold, above which this agent gets active
  - $\overline{R}$: risk threshold, above which this agent gets active

5.4.3 The GA algorithm

Representation

We seek to optimize the Navigation system with respect to its 10 parameters: *Target Tracker* (α, β, κ_1, κ_2), *Distance Estimator* (κ), *Risk Manager* (γ_A, γ_B, γ_r), and *Rescuer* ($\overline{I}_a$, $\overline{R}$). The *Distance Estimator*'s parameters ϕ and δ are fixed to 0.7 and 2 respectively since they do not affect the efficiency of the system. We use a real valued chromosome, each chromosome being a vector of 10 dimensions (see Figure 5.9). The initial population is generated randomly.

Evaluation

Each individual in the population specifies a particular parameter set for the system, and is evaluated by running a simulation with the specified parameters in a given environment. Consider that the agent navigates from an initial position p_0 to the target cluster C containing the n target positions (t_1, t_2, ..., t_n) and that it takes d_i steps to reach the target t_i from p_0 with a success value s_i. A threshold is defined for the number of steps that are taken to reach the target, above which the agent is said to have failed in its attempt to navigate to the target (i.e. its success value is 0, otherwise it is 1).

This formalization gives the clues to define the fitness function that permits the selection of the best parameter sets. It is clear that the average cost of reaching a target from the initial position p_0 is defined as the summation of the steps required to reach each target divided by the number of targets. That is,

$$\overline{c} = \frac{\sum_{i=1}^{n} d_i}{n}$$

Similarly, we can naturally define the average success value as:

$$\overline{s} = \frac{\sum_{i=1}^{n} s_i}{n}$$

The best behavior for a navigation system is the one that has a high success rate with a low average cost and with a low standard deviation for this average cost, σ_c. Thus, we define the fitness function as follows:

$$f = \frac{\overline{s}}{\overline{c} + \sigma_c}$$

Evolution

We follow an elitist approach. That is, from a population of individuals, the fittest individual is passed to the next generation. The remaining individuals form the pool from which the new generation offspring are created. We randomly select two individuals from the mating pool whose fitness is over a randomly determined value. Then we apply crossover and mutation on them to generate new individuals:

```
begin
    counter := 0;
    repeat
        r := generate a random number;
        i := find the first individual whose fitness ≥ r;
        r' := generate a random number;
        i' := find the first individual whose fitness ≥ r';
        apply crossover operator on i and i';
        apply mutation operator on i and i';
        counter := counter+1;
    until counter = population_size / 2
end
```

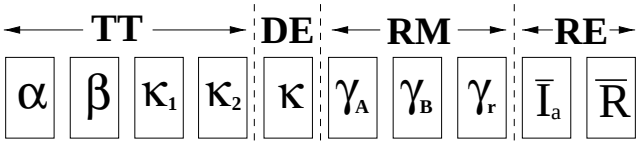


Figure 5.9: Chromosome with the set of parameters

Crossover

A simple two point crossover is used with the two parents exchanging their genetic material between two randomly generated breakpoints in the gene string. A point to note is that the chromosomes are broken only at agent boundaries (see Figure 5.9). The idea is that one of the parents may have good genes for a particular agent while the other parent may have good genes for another agent. This way the crossover could result in an offspring having a higher fitness value than both its parents.

Mutation

The mutation operator for the genetic algorithm has been adopted from the Breeder Genetic Algorithm [53]. Given any set of parameters as a chromosome, we can view it as a point *x* within a 10 dimensional space. Using our mutation operator, we seek to search for optimality within a "small" hypercube centered at *x*. How small this hypercube is, depends on the ranges in each parametric dimension within which we allow the chromosome to mutate. The parametric dimensions are not homogeneous, hence mutation ranges differ for each dimension, being directly proportional to the variance allowed in that parameter. Another feature of this mutation operator is that while it searches within the hypercube centered at *x*, it tests more often in the very close neighborhood of *x*, the idea being that, while we want to conduct a global search for optimum using our recombination, mutation is used for a more restricted local search. Having understood the broad features which the mutation operator should demonstrate, we formally define the mutation as follows:

Given a chromosome *x*, each parameter x_i is mutated with probability 0.1. The number of parameters being 10 implies that at least one parameter will be probably mutated. Further, given the mutation range for the parameter x_i as $range_i$, the parameter x_i is mutated to the value ${x_i}^*$ given by

$$ {x_i}^* = x_i \pm range_i \cdot \rho $$

As previously discussed, ρ should be such that it lies between 0 and 1 (to generate the hypercube centered at x) and also it should probabilistically take on small values so as to test more often in the close neighborhood of x. This is realized by computing ρ from the distribution

$$ \rho = \sum_j \alpha_j 2^{-j} $$

where each α_j is probabilistically either 0 or 1.

	α	β	κ_1	κ_2	κ	γ_A	γ_B	γ_R	$\overline{I}_a$	$\overline{R}$
C1	1.731	2.03	0.314	0.493	0.355	0.240	0.521	0.054	0.386	0.215
C2	1.231	2.12	1.0	0.564	0.178	1.377	4.39	0.707	0.871	0.906

Table 5.2: Optimal parameter values for each of the clusters for one execution of the GA over 100 generations

Diversity

The convergence of the genetic algorithm is estimated through its population diversity. Initially, the population has a high diversity since all the individuals are randomly selected. As the algorithm converges, the individuals in the population converge towards the best solution, thus decreasing the diversity. In our case, the individuals are points in a heterogeneous dimension space, with α, β, γ_A and $\gamma_B \in \Re^+$ while the other parameters ranging between 0 and 1. Hence we use the Mahalanobis distance measure to determine the diversity of a population [22].

The Mahalanobis distance takes into account the heterogeneity in dimensions and correspondingly scales each dimension while estimating the distance between two points. Given a set of data points $\{z_i\}$ with each data point z_i being an n-tuple $\langle z_{ij} | 1 \leq j \leq n \rangle$, the Mahalanobis distance d_m between two points z_k and z_l is given as

$$d_m(z_k, z_l) = (z_k - z_l)^T \Sigma^{-1} (z_k - z_l)$$

Here Σ is the $n \times n$ variance-covariance matrix for the given data points. To compare the diversity of populations across generations, the covariance matrix is computed taking into account all the chromosomes over all generations. The diversity of a population is then calculated as the average Mahalanobis distance of each chromosome from the mean chromosome.

5.4.4 Results

The genetic algorithm was run on the two task complexity classes represented by the target clusters C_1 and C_2 in our simulator. The population size was of 20 individuals, and we ran the genetic algorithm for 100 generations. The initial position was the same for both tasks, with the crossover and the mutation rates being 0.8 and 0.1 respectively. In the algorithm, four of the parameters — α, β, γ_A and γ_B lie on the positive real axis and hence we have to choose an upper limit on the real line. This upper limit is important since a low upper limit value implies that we implicitly restrict our real valued parameters to that limit, while a high upper limit value may increase the number of generations for which the genetic algorithm may have to be run since the initial random generation will be very disperse. α and β are exponents of numbers less than 1 and hence their large values will not be useful. Keeping these factors in consideration, the upper limit value has been fixed to 5 in our simulations.

The genetic algorithm converges to an optimal solution for each cluster as can be seen in Figures 5.10-5.15. By optimal solution we refer to the best solution the algorithm has found, which may not necessarily be the optimal solution to the navigation

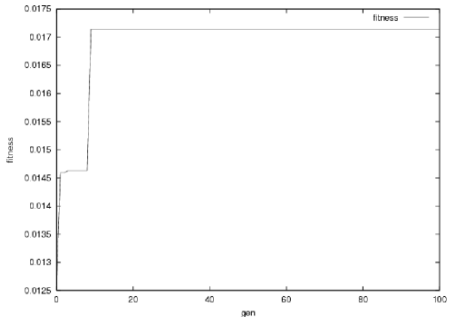


Figure 5.10: Fitness of the fittest individual along generations (cluster C_1)

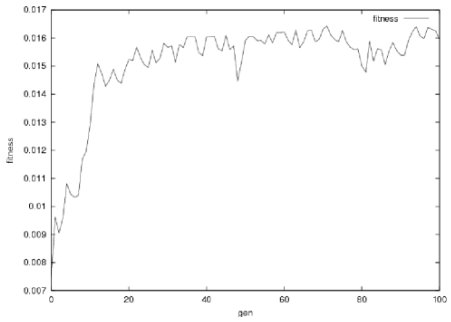


Figure 5.11: Average fitness of the population along generations (cluster C_1)

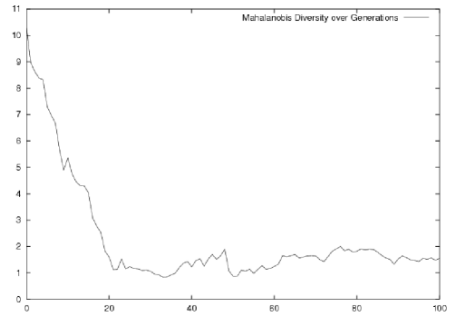


Figure 5.12: Mahalanobis diversity (cluster C_1)

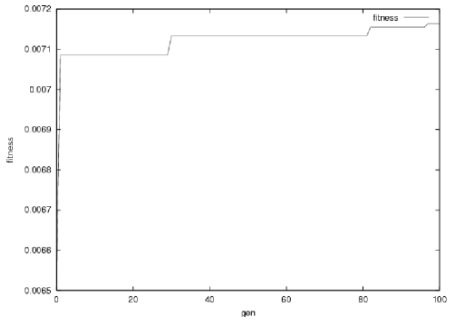


Figure 5.13: Fitness of the fittest individual along generations (cluster C_2)

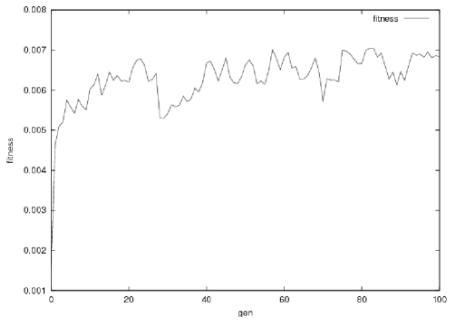


Figure 5.14: Average fitness of the population along generations (cluster C_2)

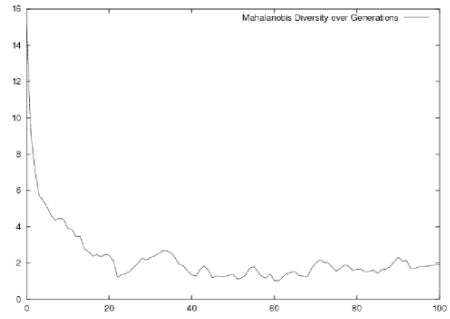


Figure 5.15: Mahalanobis diversity (cluster C_2)

	Going to C_1			Going to C_2		
	$\overline{s}$	$\overline{c}$	f	$\overline{s}$	$\overline{c}$	f
C_1 set	1	50.5	0.017	0.5	127.5	0.003
C_2 set	0.5	42.5	0.011	1	122	0.007
HT set	0.5	69	0.005	0	–	0

Table 5.3: Results obtained by the different parameter sets

task. The optimal values for some of the parameters differ significantly for the two clusters as shown in Table 5.2. The parameters associated to the bidding function of the *Risk Manager* agent differ the most between the two clusters. This is so because the *Risk Manager* is very sensitive to the complexity of the task. The more obstacles, the higher the risk of losing sight of landmarks.

In order to check the results obtained for each of the clusters, we have tested the two parameter sets found by the genetic algorithm on the two different navigation tasks (going to cluster C_1 and going to cluster C_2). We have also tested our original parameter set, which we set by hand, on the same two navigation tasks. The results obtained by each set on each of the tasks are shown in Table 5.3. For each task, the mean average success value ($\overline{s}$), average cost ($\overline{c}$) and the fitness value (f) is computed. As expected, the parameter set found for cluster C_1 performs perfectly when going to cluster C_1 and it only reaches the targets of cluster C_2 50% of the time. On the other hand, the parameter set found for cluster C_2 reaches the targets of cluster C_2 all the times, while it only reaches the targets of cluster C_1 50% of the time. Finally, the hand-tuned parameter set reaches 50% of the time for targets in cluster C_1, and never reaches the targets of cluster C_2. Therefore, the evolutionary approach has improved the global navigation behavior.

In Figures 5.16 and 5.17 we can see some paths followed by the robot using each of the parameter set on each of the tasks. Successful paths are only shown for those parameter set with a success value of 1. Otherwise, an example of a failing path (marked with a cross at its end) is shown.

5.4.5 Future Work

We will analyze the generality, in terms of different environments and starting points, of the parameters obtained by the genetic algorithm. Further work should also focus on designing an agent capable of identifying the complexity of the task being performed, so that the parameters can be switched from one set to another. We will explore the use of Case Base Reasoning techniques on this “situation identifier” agent.

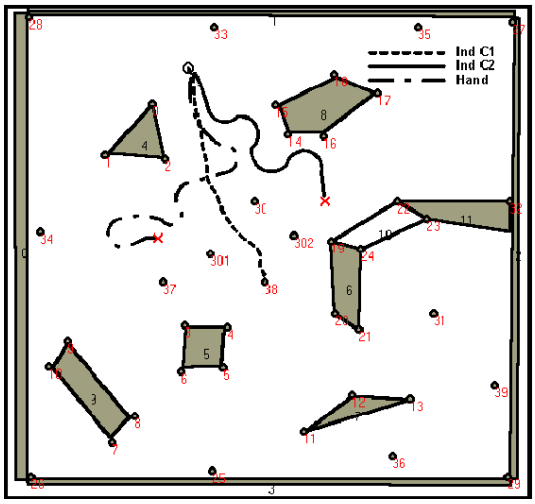


Figure 5.16: Going to cluster C_1

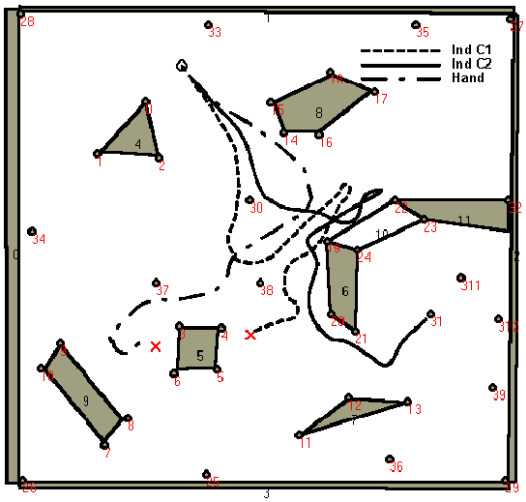


Figure 5.17: Going to cluster C_2

Chapter 6

Real Experiments

In this chapter we describe the experiments carried out with the real robot on real environments. We firstly describe the real robotic platform we have used for this experimentation, and the vision system we have developed in order to recognize the bar-coded landmarks used in the experimentation environment. A brief description of a graphical control interface is also given. Finally, we describe in detail the different scenarios on which the experiments have been carried out and the results we have obtained.

6.1 The Robot

The robot used in the experimentation is an ActivMedia [1] Pioneer 2 AT. It is a 4-wheel drive all-terrain robot, equipped with a pan and tilt unit with two B&W cameras. It is also equipped with front and rear bumpers for collision detection. The dimensions of the robot are 50×50×26 (in cm, length×width×height). The field of view of the cameras is of 45 degrees, and the pan/tilt unit can pan from +150 (left) to -150 (right) degrees and tilt from -90 (down) to +90 (up) degrees. The robot is called *MarkFinder*, since its navigational skills are based on finding landmarks in the environment. Some pictures of the robot are shown in Figure 6.1.

Although the final objective of the project we are involved in is to have a completely autonomous robot, we are currently working with off-board control and vision processing, as it is easier for programming and debugging our algorithms. We use a wireless Ethernet to communicate with the robot (to send commands to the wheels' and pan/tilt unit's motors, and to receive information about odometry and bumper activation), and the images are sent through a video transmitter (see Figure 6.2). To make the robot fully autonomous, we would only need to put the control and vision processing algorithms into its on-board computer, although it should still need to send some information back to an off-board computer for manually selecting the target.

The experimentation has been carried out in an indoor unstructured (not office-like) environment, with easily recognizable and controlled landmarks and obstacles. The environment is an area of about $50m^2$, containing ten landmarks plus the target and a

[1] ActivMedia Robotics, http://www.activmedia.com

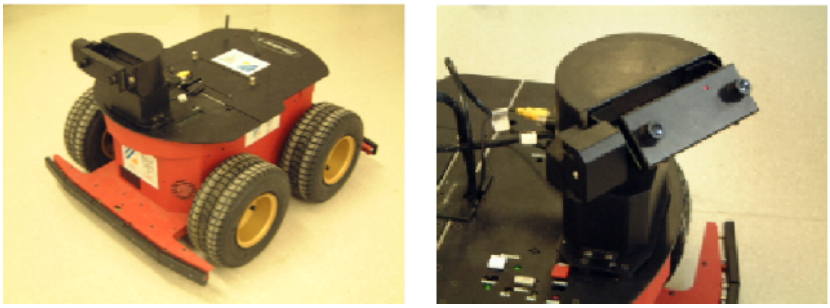


Figure 6.1: *Left*: *MarkFinder* robot. *Right*: Detail of the pan and tilt unit with two B&W cameras

few non visible obstacles. A difficulty in real environments is the vision system, as it is highly sensitive to changes in the illumination, which makes it very hard to detect objects. Therefore, we have developed a simple and robust vision system that recognizes barcoded landmarks. Moreover, the simplicity of the landmarks permits us to easily configure scenarios with different complexity levels by changing their location, as well as the location of the obstacles. The vision system and the landmarks are described in the following section.

6.2 Vision

Since we do not focus our research on the Vision system of the robot, we did not intend to develop a Vision system capable of recognizing complex objects, but just a very simple type of landmark. The simplest type we thought of was barcodes.

Landmark labels have a common part of five vertical black bars, to indicate that it is a landmark, and at the right side of the bars, a vertical binary codification with black and white squares. The binary code is composed of five squares (black meaning 1, white meaning 0), so we have 32 different codes. However, codes 0 and 31 are not used, as they give many problems when trying to identify them, so we have a total of 30 different codes, which is enough for our environment. We have used boxes with the same landmark label on their four sides so the Vision system is able to detect the landmarks from any perspective. The labels are printed on DIN A4 papers, and the dimensions of the boxes are $30 \times 30 \times 40$ (length$\times$width$\times$height), having the labels at the top of each side. Examples of such landmarks are shown in Figure 6.3.

The algorithm for recognizing these landmarks is based on the fact that the pattern of a series of alternated black and white bars of equal width is very unusual. First of all, the image is binarized, since it is in gray scale, and the algorithm needs to have pure black and white images. A *close* operation is also applied. This operation is useful for removing noise from the image. Once the binarization and the close operations are done, the algorithm starts scanning the image line by line, looking for the pattern of black and white bars. When it finds such a pattern, it scans vertically the binary code to identify which landmark has been detected. Depending on the lighting, a landmark can

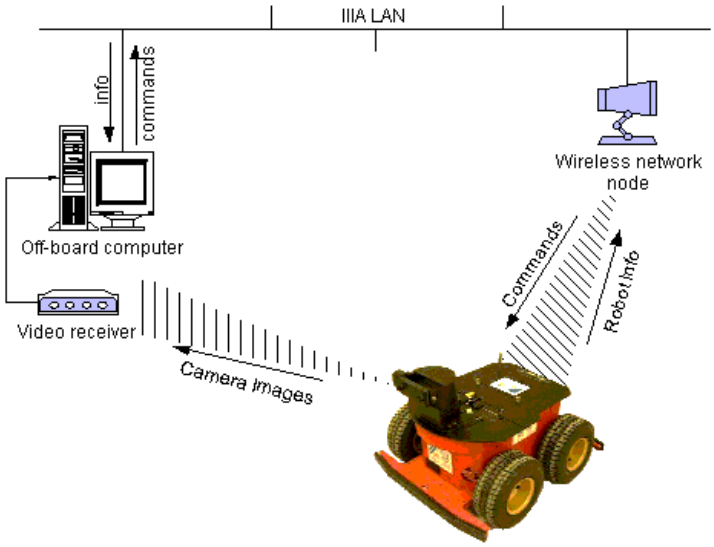


Figure 6.2: Communication with the robot

be detected using a binarization threshold, but not detected for other thresholds. Thus, this scanning process is done several times with different thresholds. Once the whole image has been processed with all the thresholds values, the information of all detected landmarks is sent to the Navigation system. A flowchart of the process is shown in Figure 6.4.

Although the robot is equipped with two cameras, we are now processing only the images of one of them, as we have not yet finished the implementation of the stereo vision algorithm. This algorithm would use the images from both cameras to compute the distance to the detected landmarks. However, we simulate that we already have this stereo vision algorithm. To do so, we have designed the landmarks so that all of them have the same size. This way, knowing the height of the bars (in pixels of the image) of a landmark, the distance from the robot to that landmark can be computed. The heading is taken as the angle to the central point of the label. However, even with the robot stopped, and due to illumination conditions, the image processing algorithm does not always detect the landmarks in the same place (it can vary some pixels). Thus, the computed distances and angles have some imprecision.

Since the quality of the cameras is not very good, the Vision system has some problems with recognizing landmarks that are far from the robot. To have a robust recognition system, we have set that it only informs about the landmarks that are within a distance of 3 meters around the robot. However, even if a landmark is in this "visible area", the Vision system sometimes misidentifies it. To solve this problem, we require that a landmark has to be recognized in several subsequent frames with the same code before informing about its detection.

But even this last requirement is not always enough to give correct landmark identification. To add more robustness to the Vision system, the detected landmarks are

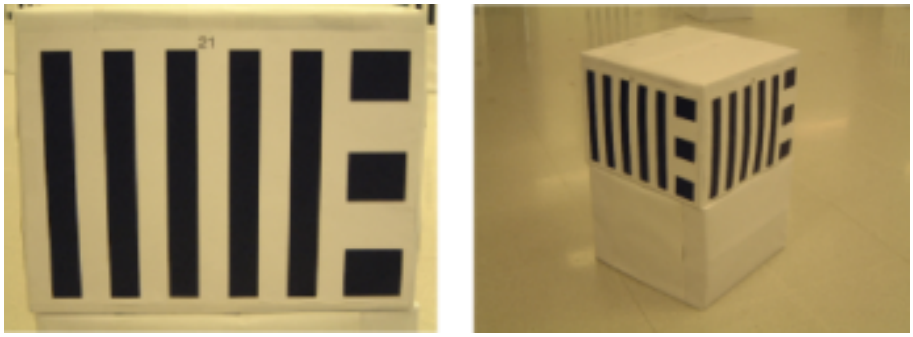


Figure 6.3: *Left*: Landmark label 21 (code = 10101 = 21). *Right*: One of the boxes with multiple landmark labels

checked against the Visual Memory (see Chapter 4 for a detailed description of the Visual Memory). For each landmark in the list of detected landmarks, two checks are done. First, we check that the detected landmark is not in a location close to another landmark stored in the Visual Memory (i.e. the distance between the two locations – one given by the Vision system and the other one stored in the Visual Memory – is below a threshold). If this is the case, and the code of the landmark differs from the one given by the Vision system, we replace the code of the detected landmark by the one stored in the Visual Memory on that location. If the code is the same, then the location given by the Vision system is assumed to be correct, and it replaces the location stored in the Visual Memory. Secondly, we check that the detected landmark is not stored in the Visual Memory at a very different location than that given by the Vision system. If this is the case, and the location stored in the Visual Memory lies in the view field of the camera, this location is given as the location of the detected landmark. If the location does not lie in the view field, the landmark is ignored. Finally, if the detected landmark is neither stored in the Visual Memory nor located close to another landmark, it means that it is a new landmark, and it is added to the Visual Memory. Table 6.1 summarizes the actions taken in each situation. We indicate the information about the landmark (code and location) that is finally sent to the Navigation system, and how the information of the Visual Memory is modified. The subscript VS stands for the information given by the Vision system, while the subscript VM refers to the information stored in the Visual Memory.

Although this check adds robustness to the Vision system, it may have undesired effects in some situations, since it gives more importance to the information stored in the Visual Memory than to that coming from the Vision system. For instance, if the location of a correctly detected landmark differs too much from its location stored in the Visual Memory, not because of an error of the Vision system, but due to the imprecision of the stored location, it will not be updated, although it should be. Another problematic situation would arise if the robot were moved to another location, without it noticing it (what is known as the “kidnapping problem”). From the new location, the Vision system would detect some landmarks, but their locations would not match at all with

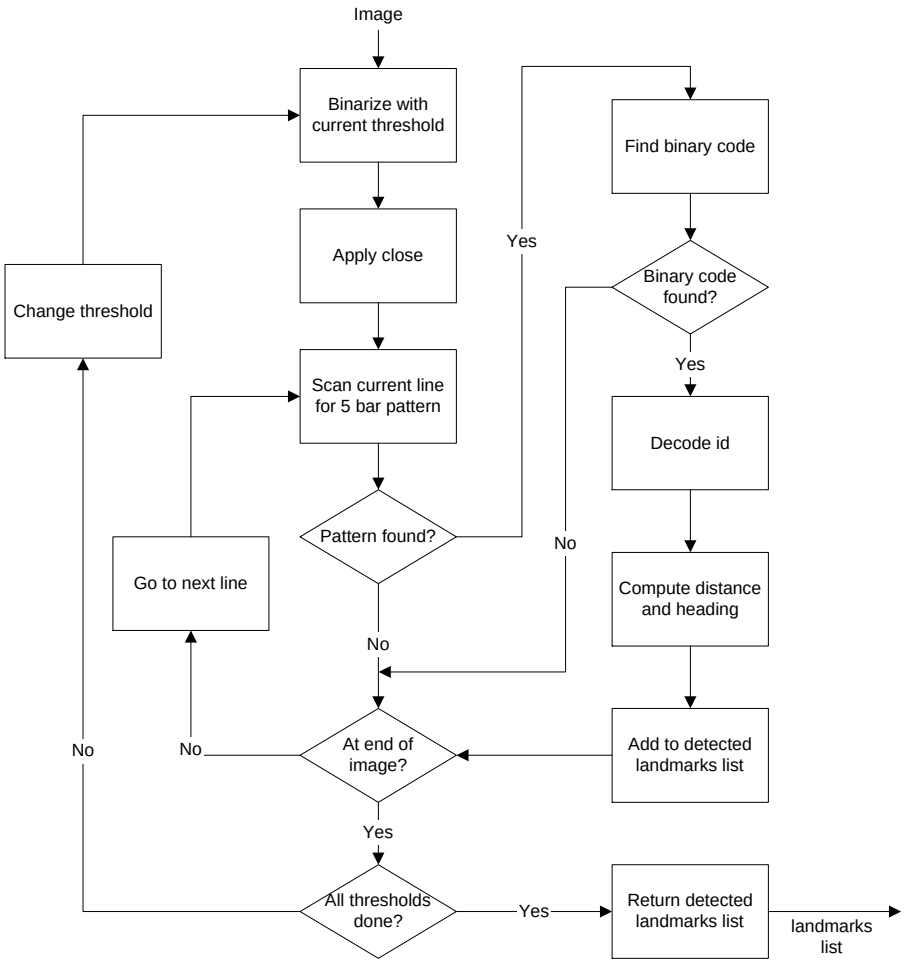


Figure 6.4: Landmark recognition process

Table 6.1: Check against Visual Memory

	Close location	Different location
Same ID	**–Right recognition–** return (id_{VS}, loc_{VS}) Update location in VM	**–Wrong recognition–** if loc_{VM} in viewfield then return (id_{VS}, loc_{VM}) else ignore landmark
Different ID	**–Wrong recognition–** return (id_{VM}, loc_{VS}) Update location in VM	**–Right identification–** return (id_{VS}, loc_{VS}) Add to VM

the locations stored in the Visual Memory, and, therefore, they would not be updated either. The first problem can be solved by changing the imprecision threshold above which the landmarks are removed from the Visual Memory, so that it only keeps those landmarks whose location is very precisely known. However, there is no way to solve the "kidnapping problem". The only way to handle it would be to have a better Vision system, so that it would not need to check the locations with the Visual Memory. Since we still do not have such a Vision system, and in our experiments the robot is never "kidnapped", we rely on the Visual Memory.

With all these provisions, landmarks are always correctly identified, therefore there is no uncertainty about the presence of landmarks, although there is imprecision about their exact location.

The fact of the Vision system being only capable of recognizing landmarks not further than 3 meters from the robot, together with the assumption of the initial visibility of the target, restricts the possible environments on which we can experiment. In order to be able to test the Navigation system on more interesting (larger) environments, we have a special landmark label that is considered as the target and can be seen from 7-8 meters. This landmark label is of the same type as the rest, but has a larger size (DIN A1), and when computing the distance from the robot to it, this is taken into account. In Figure 6.5 this larger target landmark is shown (there are four "standard" landmarks, plus the larger target, placed higher than the others).

Figure 6.5: Larger target landmark label

6.3 Graphical Interface

In order to carry out the experimentation, we have developed a graphical interface so that a human operator can give orders to the robot. The interface, shown in Figure 6.6, permits the operator to manually control the robot motion (translational and rotational speeds) and the pan and tilt unit movements. The interface has a three-dimensional representation of the environment, showing the robot and the detected landmarks and obstacles (including those stored in the Visual Memory). It also shows the images gathered from the cameras and a list of detected landmarks.

The operator can select the type of landmarks to be recognized. In our case, we were only able to use the bar-coded landmarks described in the previous section. Once the landmarks' type has been selected, the Vision system starts processing the images coming from the cameras, and the detected landmarks are displayed in the interface. The operator can then select one of the detected landmarks and set it as the target landmark to be reached. Once the target is selected, the operator can instruct the robot to go to the target. From this point on, the robot will autonomously navigate towards the target until either it reaches the target or it is instructed to stop navigating.

The interface also gives information about the Navigation system, such as the current target or how many object, beta and topological units the *Map Manager* has stored, and a graphical representation of the topological map. When the target is reached, the relevant information about the trial is given: trial duration, total length of the path, distribution of winning bids among the agents and number of diverting targets computed. This information can also be stored for later statistical analysis.

Although the interface has been used only with our robot, we have developed it so that it can be used with any robotic system, so there is no need to have a specific control interface for each different robot we may have in the lab. The idea is to let the operator configure a specific system by choosing a robot platform (be it wheeled, legged, or any other kind of autonomous robot), the type of landmarks to be used (which may imply having more than one Vision system running in parallel), and the Pilot and Navigation systems that will control the robot . Once the robotic system has been configured, it can be controlled as described above.

6.4 Goals of the Experimentation

The first goal of the real experimentation is to check whether the good results obtained through simulation are also obtained with the real robot. Ideally this would be the case, so the only modifications needed would be to make the existing Navigation system use the real robot instead of a simulated one. However, moving from simulation to the real world is not that easy, as many problems arise when working with physical robots which were not present on the simulated world (unless the simulator used has very high realism). These problems are mainly related to the motion and vision systems of the real robot.

Regarding the motion system, we have to take into account that the robot needs some time in order to execute motion commands. On simulation, we could run the system as fast as we liked, since the commands were executed immediately, however,

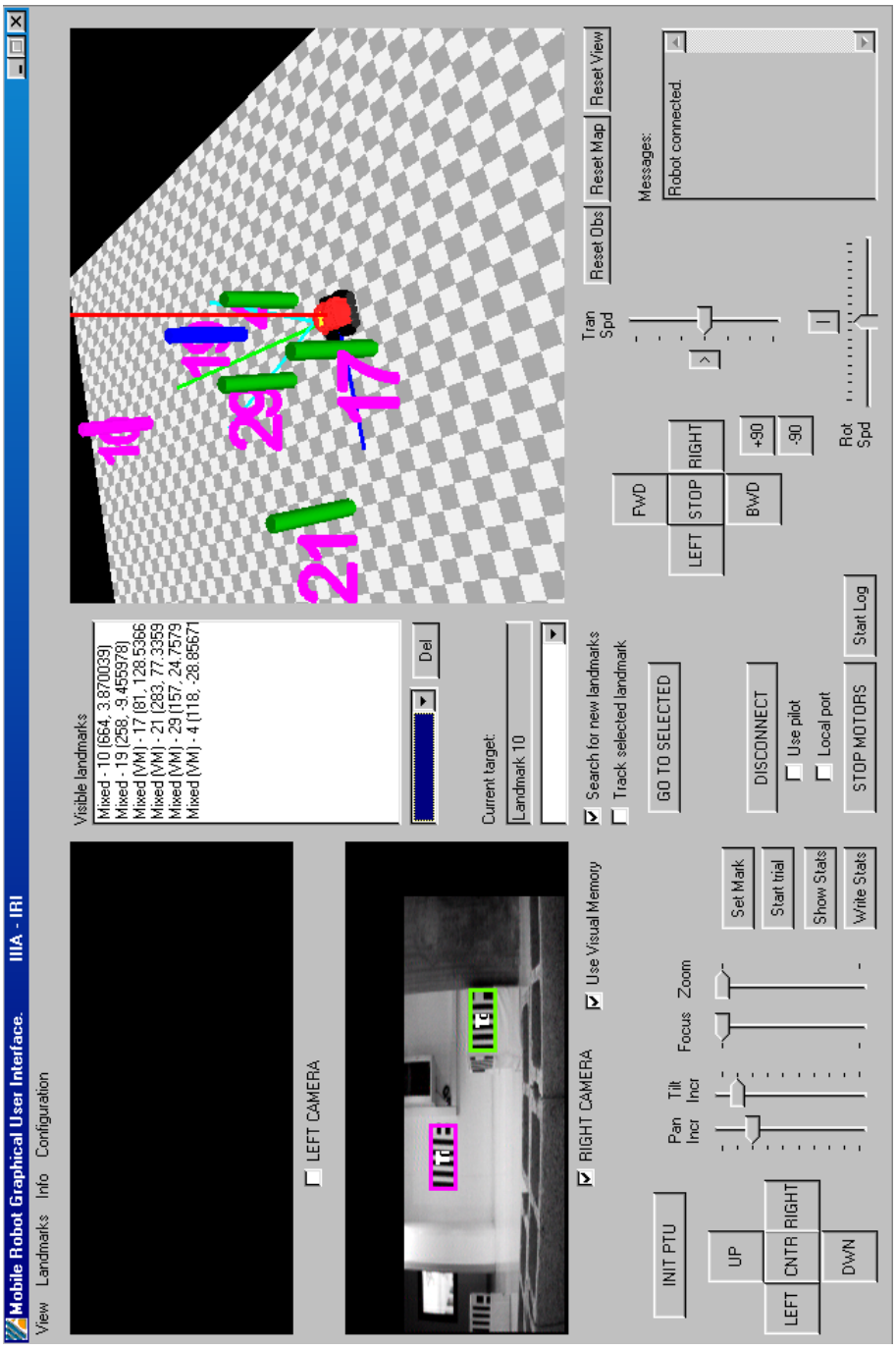


Figure 6.6: Graphical control interface

we cannot do so with the real robot. The frequency of sending these motion commands to the robot should be set according to the response time of the robot, so a command is only sent when the robot is really prepared to execute it.

Another problem of using a real robot is the vision system. Although the vision system and the landmarks we have designed are very simple, the system is not able to identify the landmarks all the time, due to changes in illumination, interference on video transmission, blurring caused by motion of the camera, etc. Therefore, as already mentioned, the vision system needs to process some frames before it is able to inform about the detected landmarks. Thus, the actions for moving the camera and identifying landmarks must also be sent with the proper frequency so that the vision system has time to process enough frames.

To overcome these problems, we have tuned the agents so that the robot is able to execute all the commands generated by the system.

Through the real experiments we also check whether the Navigation system we have designed is able to perform well in different types of environments, and if the design of each individual agent is the most appropriate for obtaining good overall performance of the Navigation system. To check this, we have experimented with different scenarios, starting with simpler ones and increasing their complexity step by step. The two main variables that describe the complexity of a scenario are:

- *Density of landmarks*: the fewer landmarks in the scenario, the more risky it is, since the map contains very little information about the relative location of the target and other landmarks. On the other hand, if the density of landmarks is high, there will very probably be always some landmarks visible, and the Navigation system will be able to compute the location of the target from the visible landmarks.

- *Density of obstacles*: if the density of obstacles is low, the path from the starting point to the target may not be blocked, or only blocked by easily avoidable obstacles, so the robot may not need to compute diverting targets to reach the original one. Contrarily, in a scenario with many obstacles, the robot is forced to change direction very often, which may cause it to lose sight of the target, and therefore, to increase the imprecision about its location. Moreover, if the obstacles block the way to the target, the Navigation system may need to compute a diverting target to reach the original one.

6.5 The Real Scenarios

The different classes of scenarios on which the experimentation has been carried out are the following:

1. *Single landmark*: in this class of scenario there is only one landmark, which is the target, and no obstacles. This class of scenarios is used to check that the robot is able to reach a target when there are no references to it and there exists a clear path to the target.

2. *Single landmark and obstacles*: these scenarios are composed of a single landmark which is the target, and several small obstacles that do not occlude the target, but force the robot to avoid them in order to get to the target.

3. *Several landmarks*: in these scenarios there are several landmarks, one of them being the target, but no obstacles (apart from the landmarks themselves, which are obviously seen as obstacles). In these scenarios the Navigation system is able to build a map of the environment, and we will check how good it is.

4. *Several landmarks and obstacles*: in these scenarios we add obstacles between the landmarks of the previous scenarios so that they block the robot and it is forced to compute diverting targets to reach the original one. In these scenarios the Navigation system is also able to build a map of the environment, including the detected blocking obstacles.

Some pictures of the different scenarios can be seen in Figure 6.7.

The first two classes of scenarios are very simple, and the experiments on such scenarios just check the very basic behavior of reaching a target through a quite clear path. In these scenarios the target is visible all the time, as the only obstacles are small ones, therefore not occluding the view field of the camera. The real tests are in classes 3 and 4, as the target may be occluded by other landmarks, and the path to the target might be blocked by landmarks and obstacles. Thus, in these scenarios, the robot must make use of its navigational skills.

We impose the restriction of the objects on the environment (that is, landmarks and obstacles) be static, so their location cannot change during a trial. If that were allowed, the computed relation among landmarks would be inconsistent, and thus the β-vector computation would not be valid at all.

6.6 Experimentation Results

We describe the experimentation carried out in each one of the four scenarios described above. We have used the parameters obtained through the Genetic Algorithm approach described in Chapter 5 (discarding those that are not used in the final version of the Navigation system). For each scenario, there is a brief discussion of the results. In each of these scenarios we have defined different starting points (two starting points in scenarios 1 and 2, and three in scenarios 3 and 4). We have run 40 trials for each starting point and stored the following statistics:

- Success/failure rate
- Number of diverting targets
- Distribution of winning bids among the agents

The relevant statistics of the experiments are shown in Table 6.2.

Figure 6.7: *Top left:* one of the obstacles used in the environments. *Top right:* scenario 1. *Middle left:* scenario 2. *Middle right:* scenario 3. *Bottom left and right:* scenario 4.

Table 6.2: Results of experimentation (TT: *Target Tracker*; RM: *Risk Manager*; RE: *Rescuer*; PS: *Pilot system*)

Scenario class	Success rate	#d.t.	Winning moving bids			Winning looking bids			
			TT	RE	PS	TT	RM	RE	PS
1	100%	0	100%	0%	0%	0%	54%	0%	46%
2	100%	0	79%	0%	21%	0%	66%	0%	34%
3	85%	0	78%	0%	22%	0%	56%	0%	44%
4	84%	0: 24% 1: 58% 2: 18%	67%	2%	31%	3%	41%	0%	56%

Scenario 1. Single landmark

Description: Scenario with just one landmark and no obstacles.

Task: Reach the landmark.

Results: In this scenario the robot behavior was, as expected, to go directly to the target in a straight line. The *Target Tracker* won 100% of the moving actions it bid for, since its bids were high because the imprecision about the location of the target was very low. The *Rescuer* did not bid because it never reached its activation levels: the imprecision was never high enough, and there were no blocking situations. Similarly, as there were no obstacles, the *Pilot* did not have to bid for changing the robot's trajectory. Regarding the looking actions, the *Risk Manager* and the *Pilot* won a similar number of bids. Since there was only one landmark, the risk was very high, and the *Risk Manager* always bid to look ahead. The target was precisely located all the time, so the looking bids of the *Target Tracker* were very low, and never won.

Scenario 2. Single landmark and small obstacles

Description: Scenario with just one landmark and some small obstacles between the robot and the landmark. The small obstacles are not visible and can only be detected by bumping into them.

Task: Reach the landmark, avoiding the obstacles detected by the bumpers.

Results: The robot did always reach the target. The winning bids for looking actions were distributed, again, among the *Risk Manager* and the *Pilot*. The *Target Tracker* did not win any of the bids because the imprecision of the target's location was not high enough. As in the previous scenario, the *Rescuer* did not have to intervene at any point. Regarding the moving actions, only the *Pilot* and *Target Tracker* won bids: the *Pilot* when an obstacle was detected and avoided, and the *Target Tracker* when the path to the target was free.

Scenario 3. Several landmarks

Description: Scenario with many landmarks and with no obstacles apart from the landmarks themselves. In order to have an interesting scenario, we placed the target landmark label higher, so that it was visible from the starting point, even if there were other landmarks in the view line from the robot to the target. If we had not done so,

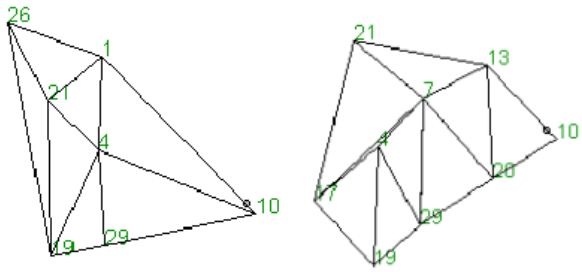


Figure 6.8: Maps of 2 different scenarios of scenario class 3

the path from the starting point to the target would always have been clear, since the target has to be initially visible, which actually corresponds to the first scenario. This change required the robot to move the camera up and down to be able to have the target landmark in its view field (in the previous scenarios, it was only doing a pan movement, with no tilt at all). Thus, we had to change the looking actions in order to incorporate the tilt angle. The agents bidding for looking actions added the tilt angle in the following way: the *Target Tracker* selects a random tilt angle, ranging from 0 degrees (so that the target landmark can be in the view field when it is 7-8 meters away) to 35 degrees (so that the target can be in the view field when it is less than 1 meter away); the *Risk Manager* does a similar thing, but it only selects a random tilt angle on one third of the actions it bids for, while it sets a null tilt angle on the other two thirds, since most of the landmarks (actually, all but the target) are at the same height of the cameras (i.e. in the null tilt angle plane); finally, the *Rescuer*, when bidding because the imprecision is too high, does two visual scans around the robot, one with a null tilt angle, and another one with a random positive tilt angle.

Task: Reach the target landmark, eventually avoiding others along the way and build a map of the environment.

Results: The behavior of the robot in this scenario was similar to the one exhibited in the previous one. However, it reached the target in 85% of the trials; in 15% of the trials it failed because the error on the location of the target made it suppose it was at the target location when it was really not there yet. This was caused by the target being occluded by other landmarks, and the constant change in trajectory needed to avoid these landmarks. These two factors caused the location of the target stored in the Visual Memory to increase its imprecision. However, the imprecision was not high enough for the *Rescuer* to become active. A difference with the previous scenario is that the *Risk Manager* bid for looking both ahead and around, since there were many landmarks, and at some point, it had enough landmarks ahead, but not around, so it bid to look around. Some examples of maps built in scenarios of this class during the trials are shown in Figure 6.8. In these maps, numbers represent landmarks the robot has seen, and the triangular regions correspond to *topological units* of the *Map Manager*'s topological map (see Chapter 3 for details on how this map is built).

Scenario 4. Several landmarks and obstacles

Description: In this scenario there are also a few non visible long obstacles between some landmarks that completely block the shortest path from the starting point to the target landmark.

Task: Reach the target landmark avoiding obstacles and building a map of the environment, and using it to compute diverting targets.

Results: the robot did successfully encode the obstacles on the topological map and used it to compute diverting targets. In 58% of the trials only one diverting target was computed in order to avoid a long obstacle blocking the path; the rest of the obstacles were avoided by the Pilot system, with no need to compute more diverting targets. In 18% of the trials, however, it was necessary to compute another diverting target, since the Pilot found the path blocked again by a long obstacle. On the other hand, in 24% of the trials, the Pilot was able to avoid the long obstacles, but did not realize that they were such long obstacles. This situation happened when the crash points with the long obstacle were not close enough to each other or to the landmarks, so they were considered as independent obstacles. Thus, when the Pilot tried to avoid these "point obstacles", it was actually avoiding the long obstacle, without realizing it. In such situations, the robot reached the target without having to compute any diverting target. Bids for moving actions were distributed very similarly as in the two previous scenarios. The only difference is that the *Rescuer* also won some bids (actually, it only wins one bid for stopping the robot each time it asks for a diverting target). Regarding bids for looking actions, now the *Target Tracker* also won a few bids to look towards the target to decrease its location's imprecision. Be it for these actions or because the scenario was not complex enough, the imprecision was never high enough so that the *Rescuer* had to bid for looking actions. Again, some of the trials failed because of the error on the target's location. In Section 6.7 we describe in detail one trial in this scenario.

6.7 A Trial Example

In this section we describe in detail one of the trials run in a scenario of class 4. The environment and the path followed by the robot are shown in Figure 6.9. The target landmark in this trial is landmark number 10. In Figures 6.10 and 6.11 the incremental building of the map is depicted. They show both a 2D representation and the topological map actually stored by the *Map Manager*. In the topological maps, although not shown, the arcs have a fixed cost of 1, unless otherwise specified. Figures 6.12 and 6.13 show the evolution of the bids of each agent and the Pilot for moving and looking actions, respectively. In these graphics, the filled areas indicate the agent that made the highest bid at that point in time. The corresponding points in Figure 6.9 are also shown. Next, we comment on the relevant points of the path:

- **A**: Starting point of the trial. Initially, landmarks 10, 29 and 19 are visible. With these three landmarks, no map is created, since at least four landmarks are needed in order to start building the map. Landmark 10 is selected as the target by the user and the *Rescuer* is informed about it. Then, the *Rescuer* bids for doing an initial sweep, as described in Section 4.4.4. During this sweep, landmarks 4, 21

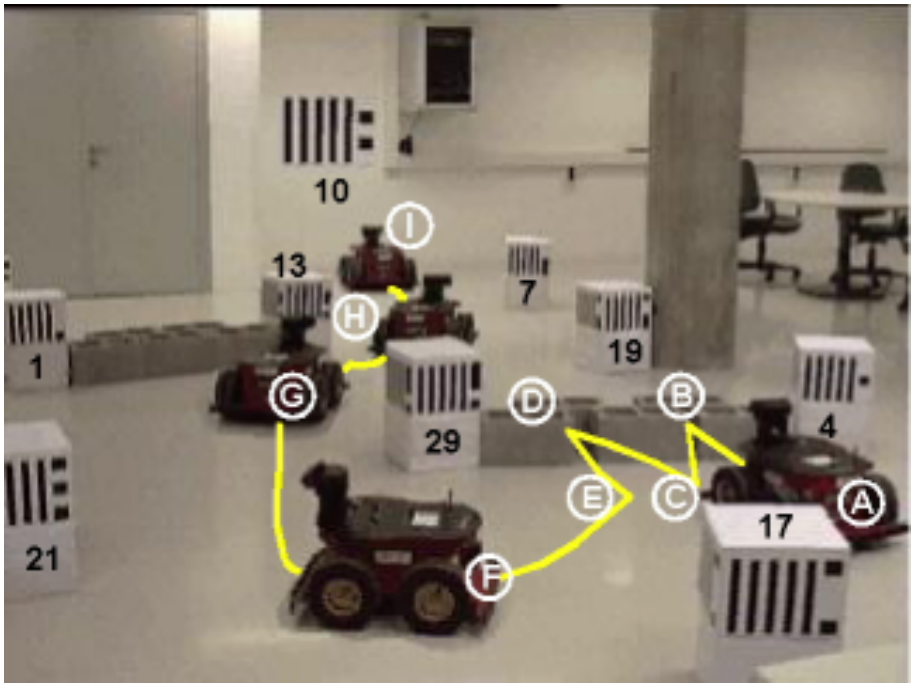


Figure 6.9: Path followed during the trial. See explanation of relevant points on the text

and 17 are also identified. With these new landmarks, the *Map Manager* is able to start building the map. The step by step update of the map is shown in Figure 6.10. The corresponding updates after seeing each of these three landmarks are maps (1) to (3). When the sweep is finished, the *Rescuer* informs the *Target Tracker* about the target being landmark 10, which immediately starts bidding for going towards it, and the robot starts moving. Actually, the point A in the graphics of the bids corresponds to this moment, when the *Target Tracker* starts bidding. As can be seen in the graphic of moving action bids, the Pilot won most of the bids. This was so because landmark 4 was close to the robot, and the Pilot wanted to avoid it. The trajectory, however, was minimally modified. Before reaching point B, landmark 13 is identified, and the map is updated accordingly, resulting in map (4) in Figure 6.10.

- **B**: The robot bumps into the obstacle between landmarks 29 and 4 and immediately backs up. However, it is not yet considered as being a long blocking obstacle, since there is still enough space between the crash point and landmark 29, through which the robot could pass. This back up is a built-in action of the Pilot, and it does not bid for executing it. That is why in the graphic the *Target Tracker* wins the bids. However, while the back up action is being executed, these bids are not taken into account.

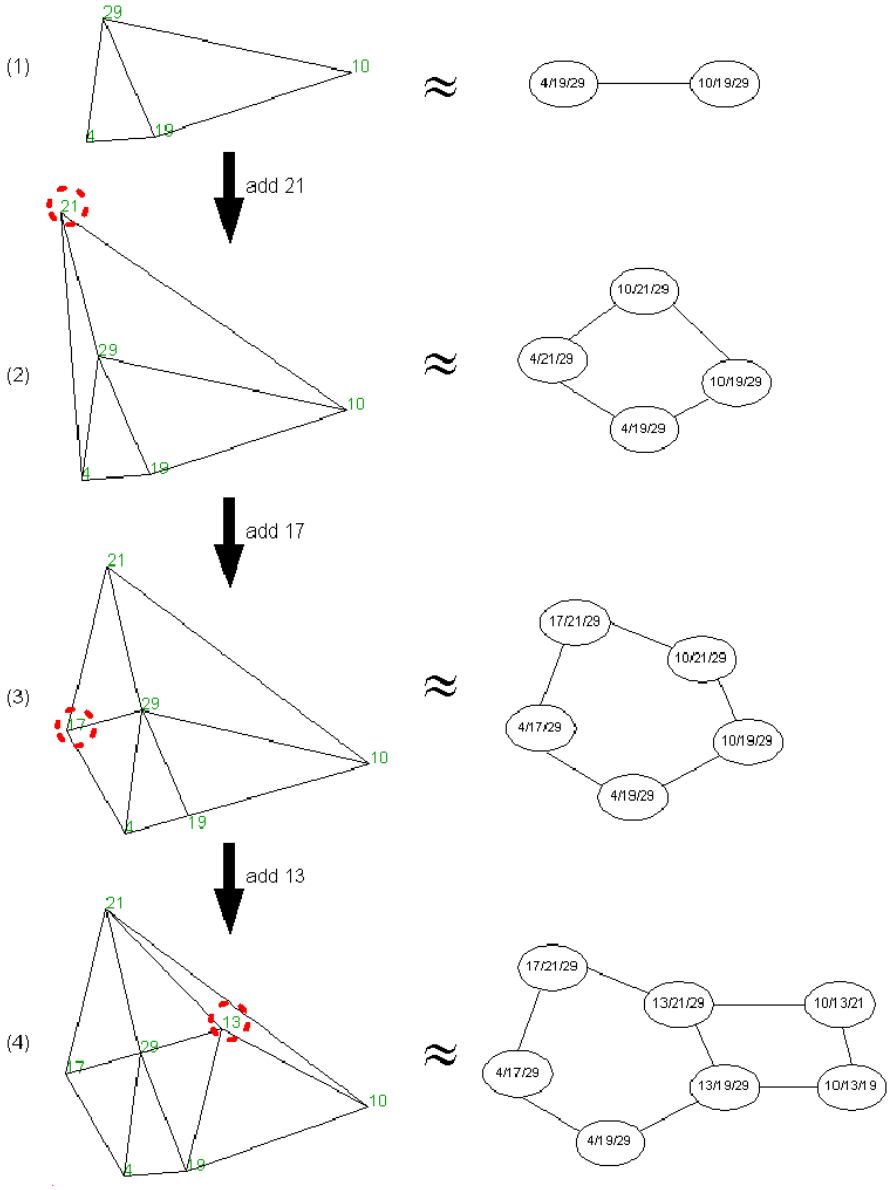


Figure 6.10: Map created during the trial

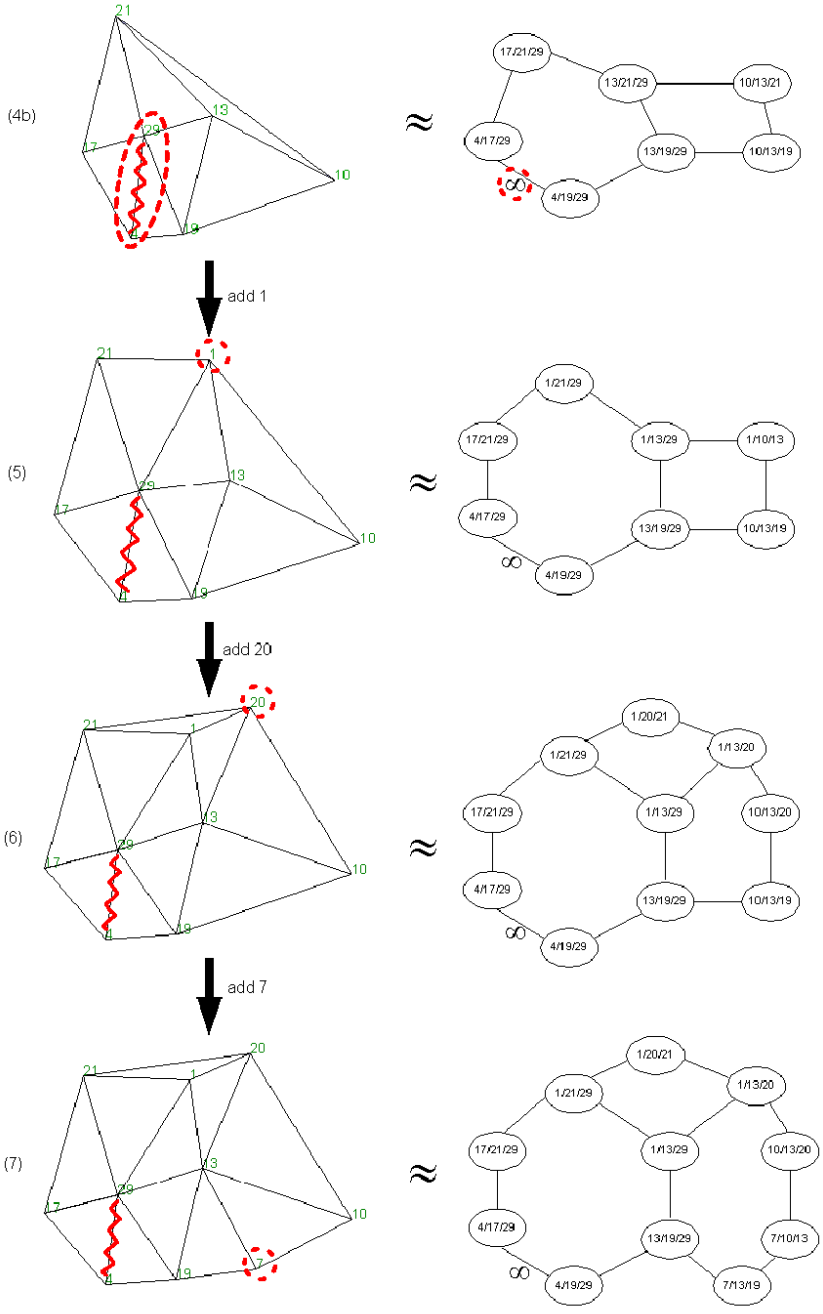


Figure 6.11: Map created during the trial (cont.)

Figure 6.12: Moving bids. *Target Tracker* in red, and Pilot in green

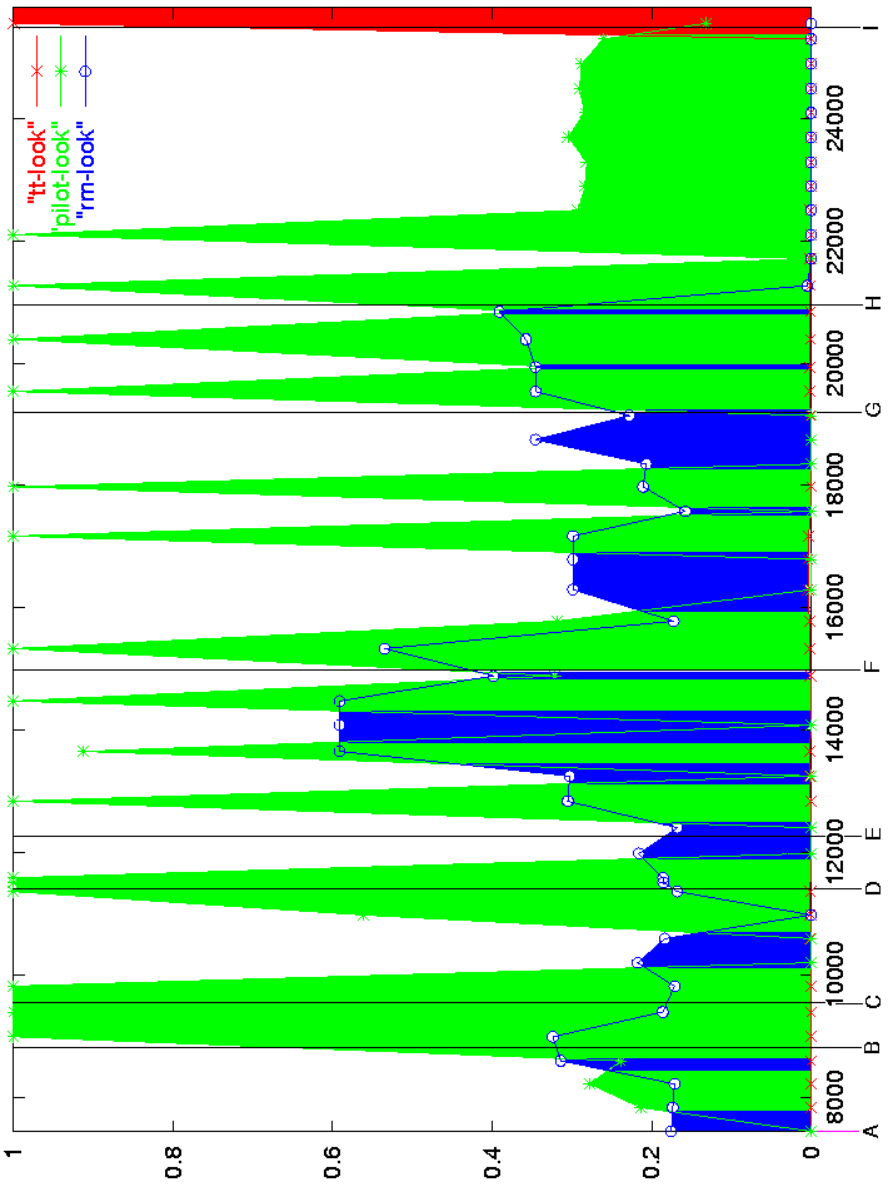


Figure 6.13: Looking bids. *Target Tracker* in red, Pilot in green and *Risk Manager* in blue

- **C**: After backing up, the *Target Tracker* bids again for moving towards the target, but these bids are surpassed by the Pilot's bids to avoid the just detected obstacle (as can be seen in the moving bids graphic), and the trajectory is slightly modified.

- **D**: The robot bumps again into the obstacle and backs up. After this second crash, the obstacle is considered to be blocking the path. The Pilot informs the Navigation system about the blocking situation. This information is internally sent to the *Map Manager*, which updates the map (the corresponding arc is assigned an infinite cost, see map (4b) in Figure 6.11), and to the *Rescuer*, which asks the *Map Manager* for a diverting target. Again, although in the graphics the *Target Tracker* is winning the bidding, the back up action is really being executed.

- **E**: The *Map Manager* computes the diverting target as being: "to cross the edge between landmarks 17 and 29" and informs the *Rescuer*, which will inform the *Target Tracker* about the new target. This agent starts bidding to move the robot so that it crosses the given edge.

- **F**: At this point, the *Target Tracker* considers that the edge 17/29 has been crossed and informs about it. This causes the *Rescuer* to set the target to be the original one (landmark 10). The *Target Tracker*'s bids are again to move towards this landmark. Before reaching point G, landmarks 1 and 20 are detected and the map is updated (maps (5) and (6)). Landmark 20 is not visible in Figure 6.9; it is behind landmark 1.

- **G**: The proximity of landmark 13 makes the Pilot bid high to avoid it, surpassing the *Target Tracker*'s bids, and the robot's trajectory is modified. While avoiding this landmark, landmark 7 is detected, and the map is updated, resulting in the final map (7).

- **H**: At this point the Pilot considers that landmark 13 has been avoided and stops bidding. The *Target Tracker* wins again, and it makes the robot go towards the target.

- **I**: The target is finally reached.

Analyzing the graphic of looking action bids, we can see that the winning bid is periodically changing between the Pilot and the *Risk Manager*. The bids of the *Target Tracker* are very low, since the target is precisely located during the whole trial. Around point H, the bids of the *Risk Manager* also decay. This is so because at that point, there are more than six landmarks behind the robot, which makes the risk 0. The winning bids of the *Target Tracker*, at point I, are due to the fact that this agent bids very high to look towards the target when this has been reached. The execution of this action has no intention of decreasing the imprecision of the target's location, but it is just a way to show that it "knows" that the target has been reached.

Table 6.3: Sources of computation of the target's location

Vision System	12.7%
Visual Memory	76.1%
Map Manager	11.2%

6.8 Discussion and Future Work

The results obtained confirmed that, as already seen through simulation, the bidding coordination mechanism and the mapping and navigation methods work appropriately. The bidding mechanism achieves the desired effect of combining the simple behaviors of the agents into an overall behavior that executes the most appropriate action at each moment, and leads the robot to the target destination. As for the mapping and navigation method, we have seen that it is able to build a map of the environment and is used for two different purposes: on one hand, to compute diverting targets when the robot finds the path to the target blocked, and on the other hand, to compute the location of the target when this is not visible. Regarding this latter use of the map, Table 6.3 shows the statistics of how the target's location is computed. The sources of this computation can be the following: (1) the real Vision system, that is, the target is recognized and its location computed from the images, (2) the Visual Memory (described in Chapter 4), and, (3) the *Map Manager*, that is, the location of the target is computed using the beta-coefficient system and the locations of other landmarks. As can be seen from the statistics, most of the time (76.1%) the location is computed using the Visual Memory, however, sometimes (11.2%) the Navigation system must make use of its "orientation sense" in order to figure out where the target is. Figure 6.14 shows the evolution of the imprecision on the target's location and the different sources (the colored band at the bottom of the graphic). Although, usually, the robot realizes that it has reached the target by obtaining its location from the Visual Memory, it sometimes realizes it using the orientation sense. However, since the computation of the target location using the orientation sense is more imprecise than the Visual Memory (because it accumulates the imprecision of several landmarks' locations), the robot sometimes informs about having reached the target when it has not really done it, thus failing in its mission.

The scenarios used in the real experiments were not very complex. Therefore, some more experimentation on more complex scenarios should be performed. These new scenarios should include more blocking obstacles, possibly having some cul-de-sacs, so that the robot would need to undo the path already done.

Although the good results obtained indicate that the agents are well designed, we could still improve them and, hopefully, improve the performance of the overall robotic system. Actually, during the experimentation with the real robot, we already did some refinement. However, this refinement can be a never-ending task, and for this reason we decided to stop it and do the real experiments with the version of the agents described in Chapter 4. The possible further refinement of some of the agents could go in the following directions:

- *Target Tracker*: this agent could do a more intelligent tilt angle selection, such

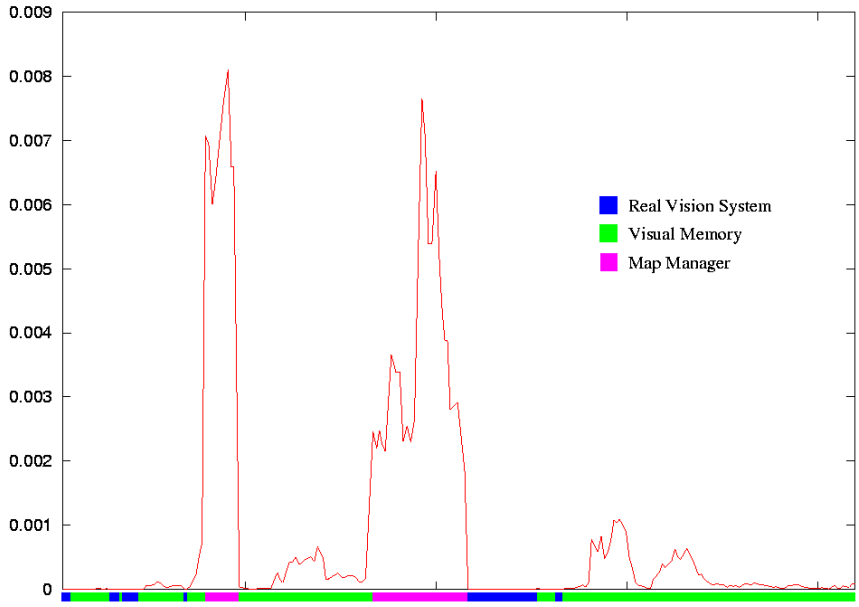


Figure 6.14: Evolution of the target's location imprecision and sources of computation

as being a function of the distance to the target, thus, increasing the chances of having it in the view field of the camera.

- *Risk Manager*: this agent could also bid, not only for looking ahead or around, but also to other areas with fewer landmarks, or even selecting a random direction to look to. Right now, if there are very few landmarks ahead, this agent sticks to bidding for looking ahead, and never bids for looking around, thus, ignoring a large part of the environment. An alternative to modifying the *Risk Manager* would be to add a new agent with this behavior.

Some improvements could also be done on the Pilot and Vision systems. Regarding the Pilot, we could use a better obstacle avoidance algorithm. With the current algorithm, only the closest obstacle is considered for computing the avoidance path. We could improve the robot's performance if the Pilot took into account all the obstacles and landmarks stored in the Visual Memory, thus, producing better avoidance paths. We are also planning to equip the robot with a laser scanner. This laser would be continuously scanning a 180 degree area in front of the robot to accurately detect obstacles that are several meters away. With this new sensor, the Pilot could avoid the obstacles before bumping into them, thus, generating better paths. Regarding the Vision system, we plan several improvements. The first one is to finish the stereo algorithm, so we can use the two available cameras for computing the distance to the landmarks. Another very important improvement is to make the Vision system more robust, so that it does not need to check the recognized landmarks against the Visual Memory. Actually, we

should use the robust Vision system to adjust the imprecisions of the Visual Memory. We also plan to convert the Vision system into a Multiagent Vision system. In this system, several agents would process the camera images with different algorithms, and the agents should agree on what could be a good landmark candidate (salient enough, robust, static, etc.). A final improvement of the Vision system would be to let it bid for services by other systems (either the Pilot system or itself). With the bidding capability, it could request the Pilot to approach a landmark to better recognize it, or even "request itself" to slightly move the camera so that a partially visible landmark enters completely the view field.

Chapter 7

Conclusions and Future Work

7.1 Revisiting the Objectives

The need for autonomous robots has been rapidly increasing in the last years. There are many areas in which these robots are used, ranging from "service robots", such as museum guides or transportation robots in factories, to robots used for tasks to be performed in inaccessible environments, such as planetary exploration, hazardous material handling and rescue missions.

Usually, service robots operate in indoor structured environments. The problem of navigating through indoor environments has been the focus of robotics research during many years, and many successful results have been achieved. Usually, the map of the environment is given a priori (either a detailed metric map or a topological one, showing the spatial relationship among different places of the environment), or, when it is not given, there is an initial phase for learning the map. Once it is learned, the robot repeatedly performs the task in this environment. Examples of such robots are those performing delivery tasks in office environments or guiding tours in museums [67, 9].

On the other hand, inaccessible environments are usually unknown and unstructured (as is the case in most outdoor environments), which pose a more difficult problem. The lack of structure of such environments makes the map building very difficult. Moreover, the large scale of these environments also adds to the difficulty of mapping and navigation tasks. These characteristics make it impossible to apply the approaches used in indoor structured environments. Although there has been also a lot of research on navigation in unstructured environments, it is still an open problem.

This PhD thesis has focused on this latter problem, that is, on ***navigating in unknown unstructured environments***. The research was part of a robotics project whose goal is to have a completely autonomous robot capable of navigating in outdoor unknown environments. A human operator selects a target using the visual information received from the robot's camera, and the robot has to reach it without any further intervention of the operator. Navigating to a target is a fundamental task of any mobile robot, whatever its mission is (be it grasping objects, analyzing them, looking for something, etc.) The task to be performed once the target has been reached is outside the scope of

the project and this thesis.

A first milestone of the project was to develop a navigation system for indoor unknown unstructured environments. The reason for starting with indoor environments was that the development of robust vision systems for outdoor environments is still an open and very difficult problem in the field of computer vision. Therefore, since the vision system was not the focus of our research, we decided to start experimenting indoors, for which vision systems are much easier to develop. Moreover, we designed the landmarks so that we could easily change their location, thus, permitting us to configure scenarios of different complexity.

This thesis has reported the research carried out in order to accomplish this first milestone. For achieving it, we have combined *landmark-based navigation*, *fuzzy* distance and angle representations and *multiagent coordination* based on a *bidding mechanism*. The objective of our research was to have a ***robust navigation system with orientation sense for unknown unstructured environments using visual information***.

7.2 Contributions

The research has been focused on two main threads: the ***control architecture*** and the ***mapping and navigation method***. The contributions of the thesis on these two areas are presented next.

Regarding the ***control architecture***, we have proposed a general coordination architecture based on a *bidding* mechanism. In this architecture there are two types of systems: *executive systems* and *deliberative systems*. Executive systems have access to the sensors and actuators of the robot. These systems offer services for using the actuators to the rest of the systems (either executive or deliberative) and also provide information gathered from the sensors. On the other hand, deliberative systems take higher-level decisions and require the services offered by the executive systems in order to carry out the task assigned to the robot. Although we differentiate between these two types of systems, the architecture is not hierarchical, and coordination is made at a single level involving all the systems. This coordination is based on a simple mechanism: *bidding*. Deliberative systems always bid for the services offered by executive systems, since this is the only way to have their decisions executed. Executive systems that only offer services do not bid. However, those executive systems that require services from any executive system (including themselves) must also bid for them. The systems bid according to the internal expected utility associated to the provisioning of the services. A coordinator receives these bids and decides which service each of the executive systems has to perform.

The bidding mechanism assures that the action actually being executed by the robot is the most valued one at each point in time, and thus, if the systems bid rationally, the dynamics of the bids lead the robot to execute the necessary actions in order to reach a given target. An advantage of using such mechanism is that there is no need to create a hierarchy, such as in the subsumption architecture, but it is dynamically changing depending on the specific situation of the robot and the characteristics of the environment. A second advantage is that its modular view conforms an extensible architecture. To extend this architecture with a new capability we would just have to

plug in a new system. Moreover, the coordination mechanism can be applied at different levels of the architecture, be it at the overall architecture level, or within each one of the systems.

For our specific navigation problem, we have instantiated this architecture with three systems: the Pilot, Vision and Navigation systems. The first two being executive systems, and the latter one being deliberative. The Navigation system has been designed as a multiagent system using the same bidding coordination mechanism used in the overall architecture. The high-level task of navigating to a given target has been decomposed into a set of simpler tasks, and we have designed one agent competent in each of these tasks. These agents compete, since they may request the execution of conflicting actions. As in the overall architecture, each agent bids for the services offered by the executive systems, and there is a coordinator agent that decides which is the most urgent request. This request is then sent as the request of the Navigation system, which will have to compete with the requests of the Pilot system.

Regarding the ***mapping and navigation method***, we have addressed two problems: the problem of providing the robot with orientation sense and the problem of building a map of the environment and using it for navigational purposes. Concerning the orientation sense, we have built upon previous work presented by Prescott [55], which describes a model for storing spatial relationships among landmarks in the environment. We have extended Prescott's model so that it can be used with fuzzy information about the locations of landmarks. This is of great importance when working with real robots, as it is impossible to avoid dealing with the imprecision of real world environments. As far as we know, this is the first application of Prescott's model on a real robotic system. As part of this extension, we have also developed methods for building a topological map of the environment, which is used for computing diverting targets, needed by the robot when it finds that the path to the target is blocked.

Although the robotic system proposed in this thesis has been presented as a whole system, including both the control architecture and the mapping method, they are two solutions for two completely independent problems. Thus, we could substitute Prescott's mapping method by any other mapping method (be it another topological approach, a metric approach, etc.). Obviously, the particularities of each system depend on the mapping method (e.g. it would make no sense having a Vision system if the map uses sonar readings), but the overall architecture and its coordination mechanism would not be affected at all by the choice of this mapping method. Similarly, our mapping method could be used in a robotic system controlled by any other architecture (be it hybrid, centralized, etc.).

We have obtained successful results, both on simulation and on real experimentation, showing that the mapping method is capable of building a map of an unknown environment and using this information to move the robot from a starting point to a given target. The experimentation also showed that the bidding mechanism we designed for controlling the robot produces the overall behavior of executing the proper action at each moment in order to reach the target. Thus, we consider that we have satisfactorily achieved the objective of developing a navigation system with orientation sense for unknown unstructured environments.

In parallel with the experimentation with the real robot, we have also used simula-

tion to apply Machine Learning techniques. More concretely, we have used Reinforcement Learning for having the system learn how to use the camera more appropriately, that is, to use it only when needed. We have also used a Genetic Algorithm approach, in order to tune some of the parameters that define the behavior of the agents in the Navigation system. Successful results have been obtained with both techniques, though there is still much work to do. Actually, they could easily be the subject of several PhD theses, especially the work on Reinforcement Learning.

7.3 Future Work

Although, as we have just said, we consider that the goal of the thesis has been accomplished, there are plenty of improvements that could be done in order to achieve better results. In the following sections we present, for each of the aspects of the research carried out in this thesis, some of the open issues that deserve further research (some of which we are already working on). Note that it is basically a compilation of the Future Work sections of each of the previous chapters.

7.3.1 Mapping and Navigation

The extension of Prescott's method, together with the algorithms to compute diverting targets, has been shown to successfully encode the environment into a map that permits navigating from a starting point to the target. However, we would like to explore other mapping methods, so that the combination of the different methods adds robustness to the Navigation system. With the current mapping method, the robot needs to see at least three landmarks in order to be able to use the information stored in the map. We would like to develop some other mapping methods to cope with the situations in which the robot has very little information (i.e. less than three landmarks). These methods would be even more qualitative than our fuzzy extension of Prescott's method. We could, for example, look at the field of Spatial Cognition, which works with spatial relationships such as "landmark X is at the left hand side of the line connecting landmark Y and landmark Z".

7.3.2 Robot Architecture and Multiagent Navigation System

One of the first things to explore in our coordination architecture is the use of a more economic view of the bidding mechanism. With this approach, each system (or agent) would be assigned a limited credit, and they would only be allowed to bid if they had enough credit. There should also be a way to reward the systems (agents). If not, they would run out of credit after some time and no one would be able to bid. The difficulty of the reward mechanism is how to decide when to give a reward and who deserves to receive it. This problem, known as the credit assignment problem, is very common in multiagent learning systems, especially in Reinforcement Learning, and there is not a general solution for it; each system uses an ad hoc solution for the task being learned.

An alternative to the economic view would be to have a mechanism to evaluate the bidding of each system (agent), assigning them succeeding or failing bids, or some

measure of trust, in order to take or not take into account their opinions. However, we would face again the credit assignment problem.

Regarding the specific set of agents we have designed for solving the navigation problem, we could introduce some improvements on some of them, and even add new agents to the Navigation system. Some of these improvements could go in the following lines:

- *Target Tracker*: this agent could do some more intelligent tilt angle selection, being a function of the distance to the target, thus, increasing the chances of having it in the view field of the camera.

- *Risk Manager*: this agent could also bid not only for looking ahead or around, but also to specific areas with fewer landmarks, or even selecting a random direction to look to. Right now, if there are very few landmarks ahead, this agent sticks bidding for looking ahead, and never bids for looking around, thus, ignoring a large part of the environment. An alternative to modifying the *Risk Manager* would be to add a new agent with this behavior.

Some improvements could also be done on the Pilot and Vision systems. Regarding the Pilot, we could use a better obstacle avoidance algorithm. With the current algorithm, only the closest obstacle is considered for computing the avoidance path. We could improve the robot's performance if the Pilot took into account all the obstacles and landmarks stored in the Visual Memory, thus, producing better avoidance paths We are also planning to equip the robot with a laser scanner. This laser would be continuously scanning a 180 degree area in front of the robot to accurately detect obstacles that are several meters away. With this new sensor, the Pilot could avoid the obstacles before bumping into them, thus, generating better paths. Regarding the Vision system, we plan several improvements. The first one is to finish the stereo algorithm, so we can use the two available cameras. Another very important improvement is to make the Vision system more robust, so that it does not need to check the recognized landmarks against the Visual Memory. Actually, we should use the robust Vision system to adjust the imprecisions of the Visual Memory. We also plan to convert the Vision system into a Multiagent Vision system. In this system, several agents would process the camera images with different algorithms, and the agents should agree on what could be a good landmark (salient enough, robust, static, etc.). A final improvement of the Vision system would be to let it bid for services by other systems (either the Pilot system or itself). With the bidding capability, it could request the Pilot to approach a landmark to better recognize it, or even "request itself" to slightly move the camera so that a partially seen landmark enters completely the view field.

7.3.3 Reinforcement Learning

Although the results obtained through Reinforcement Learning showed that the system learned to select actions in order to solve the complex camera tradeoff, we still need to integrate it into the overall multi-agent system, to see if the performance of the whole system is also improved. Even though the *Learning Agent* knows which actions it has

to bid for (following the learned policy), it is not clear what its bidding function should be; it could be a constant bidding value, or a bidding depending on the values of $V(s)$.

Some more further work will be focused on the design of the state and feature representation and the set of available actions. Asada et al. [5] proposed a solution for coping with the "state-action deviation problem", in which actions operate at a finer grain than the features can represent, having the effect that most actions appear to leave the state unchanged, and learning becomes impossible. We plan to evaluate the suitability of this approach in our experiments. Regarding the action set design, we found that the set of available actions was maybe too small and some more actions may be needed. We are working on an "action refinement" method [20] that exploits prior knowledge information about the similarity of actions to speed up the learning process. In this approach, the set of available actions is larger, but in order to not slow down the learning process, the actions are grouped into subsets of similar actions. Early in the learning process, the Reinforcement Learning algorithm treats each subset of similar actions as a single "abstract" action, estimating $P(s'|s, a)$ not only from the execution of action a, but also from the execution of its similar actions. This action abstraction is later on stopped, and then each action is treated on its own, thus, refining the values of $P(s'|s, a)$ learned with abstraction.

7.3.4 Genetic Algorithm

We should analyze the generality, in terms of different environments and starting points, of the parameters obtained by the genetic algorithm. Further work should also focus on designing an agent capable of identifying the complexity of the task being performed, so that the parameters can be switched from one set to another. We will explore the use of Case Base Reasoning techniques on this "situation identifier" agent.

7.3.5 Real experimentation

The results obtained through real experimentation confirmed that, as already seen through simulation, the bidding coordination mechanism and the mapping and navigation methods work appropriately. Nonetheless, the scenarios used in the real experiments were not very complex, and some more experimentation on more complex scenarios should be performed. These new scenarios should include some more obstacles, eventually having some cul-de-sacs, so that the robot would need to undo the path already done.

However, the big next step on our research is to move the experimentation to outdoor environments. The main difficulty of doing so is the availability of a vision system for outdoors, which we do not have at this moment. However, we think that the successful results obtained on indoor unstructured environments could be quite easily obtained outdoors, since neither the navigation method nor the control architecture are dramatically affected by the differences of indoor/outdoor environments.

7.3.6 Case Based Reasoning

Besides the use of CBR described in the Genetic Algorithm approach, we also plan to add a CBR agent that would bid for actions. This agent would use the information of past experiences in different trials (stored in form of $\{$situation,action,result$\}$ tuples) to recognize similar situations, and would then bid for executing the actions (or similar actions) that best suited those situations. The difficulty of this approach is to find the proper way to characterize the situations and how to compare two situations in order to find out how similar they are. In this approach we also face the credit assignment problem, since we cannot evaluate a situation-action experience until the robot either successfully reaches the target or fails in its mission.

Bibliography

[1] J. Andrade-Cetto and A. Sanfeliu. Topological map learning for a mobile robot in indoor environments. In J. S. Sànchez and F. Pla, editors, *Proceedings of the 9th Spanish Symposium on Pattern Recognition and Image Analysis*, pages 221–226, 2001.

[2] R. C. Arkin. Path planning for a vision-based autonomous robot. In *Proceedings of the SPIE Conference on Mobile Robots*, pages 240–249, 1986.

[3] R. C. Arkin. *Behaviour-based Robotics*. MIT Press, 1998.

[4] R.C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics research*, 8(4):92–112, 1989.

[5] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.

[6] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[7] G. Bojadziev and M. Bojadziev. *Fuzzy sets, fuzzy logic, applications*, volume 5 of *Advances in Fuzzy Systems*. World Scientific, 1995.

[8] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[9] W. Burgard, A. B. Cremers, D. Fox, G. Lakemeyer, D. Hähnel, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, 1998.

[10] D. Busquets, R. López de Màntaras, C. Sierra, and T.G. Dietterich. Reinforcement learning for landmark-based robot navigation. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 841–842. ACM press, 2002.

[11] D. Busquets, T.G. Dietterich, R. López de Màntaras, and C. Sierra. A mulit-agent architecture integrating learning and fuzzy techniques for landmark-based

robot navigation. *Lecture Notes in Computer Science (Proceedings of CCIA'02)*, 2504:269–281, 2002.

[12] D. Busquets, C. Sierra, and R. López de Màntaras. A multi-agent approach to fuzzy landmark-based navigation. *Journal of Multiple-Valued Logic and Soft Computing*, Old City Publishing (In press).

[13] D. Busquets, C. Sierra, and R. López de Màntaras. A multi-agent approach to qualitative landmark-based navigation. *Autonomous Robots*, Kluwer Academic Publishers (In press).

[14] M. Carreras, J. Batlle, and P. Ridao. Hybrid coordination of reinforcement learning-based behaviours for auv control. In *Proceedings of IEEE/RSJ IROS*, 2001.

[15] A. R. Cassandra, L. Pack Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 1023–1028, Cambridge, MA, 1994. AAAI Press/MIT Press.

[16] J.A. Castellanos and J.D.Tardós. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Kluwer Academic Publishers, 2000.

[17] R. Chatila. Path planning and environment learning in a mobile robot system. In *Proceedings of the 1982 European Conference on Artificial Intelligence (ECAI-82)*, 1982.

[18] R. Chatila and J.P. Laumond. Position referencing and consistent world modeling for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1985.

[19] M.B. Dias and A. Stentz. A market approach to multirobot coordination. Technical report, Robotics Institute, Carnegie Mellon University, 2001.

[20] T.G. Dietterich, D. Busquets, R.López de Màntaras, and C. Sierra. Action refinement in reinforcement learning by probability. In *Proceedings of the 19th International Conference on Machine Learning (ICML'02)*, pages 107–114, 2002.

[21] G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (slam) problem. *IEEE Transaction of Robotics and Automation*, 17(3):229–241, 2001.

[22] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., 2001.

[23] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265, 1987.

[24] M. Teresa Escrig and F. Toledo. Autonomous robot navigation using human spatial concepts. *International Journal of Intelligent Systems*, 15:165–196, 2000.

[25] R. Liscano et al. Using a blackboard to integrate multiple activities and achieve strategic reasoning for mobile-robot navigation. *IEEE Expert*, 10(2):24–36, 1995.

[26] T. Finin, R. Fritzson, and D. McKay. An overview of KQML: A knowledge query and manipulation language. Technical report, Department of Computer Science, University of Maryland, Baltimore county, 1992.

[27] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[28] D. Gachet, M. A. Salichs, L. Moreno, and J. R. Pimentel. Learning emergent tasks for an autonomous mobile robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 290–297, 1994.

[29] E. Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, 1991.

[30] B. Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1-2):329–365, January 1995.

[31] M. Humphrys. Action selection methods using reinforcement learning. In Pattie Maes et al, editor, *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 135–144, Cambridge, MA, 1996. MIT Press.

[32] C. Isik and A.M. Meystel. Pilot level of a hierarchical controller for an unmanned mobile robot. *IEEE J. Robotics and Automation*, 4(3):241–255, 1988.

[33] L.P. Kaelbling, A.R. Cassandra, and J.A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

[34] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 500–505, St. Louis, MO, 1985.

[35] D. Kortencamp and T. Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 979–984, 1994.

[36] D. M. Kortenkamp. *Cognitive maps for mobile robots: A representation for mapping and navigation*. PhD thesis, University of Michigan, Computer Science and Engineering Department, Michigan, 1993.

[37] B. Krogh. A generalized potential field approach to obstacle avoidance control. Technical Report MS84-484, Society of Manufacturing Engineers, Dearborn, Michigan, 1984.

[38] B.J. Kuipers and Y.-T. Byun. A robust qualitative method for spatial learning in unknown environments. In *Proceedings of AAAI-88*, Menlo Park, CA, 1988. AAAI Press/MIT Press.

[39] T.S. Levitt and D.T. Lawton. Qualitative navigation for mobile robots. *Artificial Intelligence Journal*, 44:305–360, 1990.

[40] M. López. *Approaches to Map Generation by means of Collaborative Autonomous Robots*. PhD thesis, Universitat Autònoma de Barcelona, Bellaterra, Barcelona, 1999. Also as *Monografies de l'IIIA* (Vol. 11).

[41] M. López, F. Esteva, R. López de Mantaras, C. Sierra, and J. Amat. Map generation by cooperative low-cost robots in structured unknown environments. *Autonomous Robots*, 5:53–61, 1998.

[42] R. Lumia. Using nasrem for real-time sensory interactive robot control. *Robotica*, 12(2):127–135, 1994.

[43] P. Maes. The dynamics of action selection. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 991–997, 1989.

[44] P. Maes and R. Brooks. Learning to co-ordinate behaviours. In Thomas Dietterich and William Swartout, editors, *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, volume 2, pages 796–802, July29 August–3 1990.

[45] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(2-3):311–365, June 1992.

[46] E. Martinez and C. Torras. Qualitative vision for the guidance of legged robots in unstructured environments. *Pattern Recognition*, 34(8):1585–1599, 2001.

[47] M.J. Matarić. Navigating with a rat brain: a neurobiologically-inspired model for robot spatial representation. In J.-A. Meyer and S.W. Wilson, editors, *From Animals to Animats*, Cambridge, MA, 1991. MIT Press.

[48] M.J. Matarić. Behavior-based control: Main properties and implications. In *Proceedings of Workshop on Intelligent Control Systems, International Conference on Robotics and Automation*, Nice, France, 1992.

[49] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103, 1993.

[50] H.P. Moravec. Toward automatic visual obstacle avoidance. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, page 584, Cambridge, MA, 1977.

[51] H.P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[52] H.P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 116–121, 1985.

[53] H. Muhlenbein and D. Schlierkamp-Voosen. The science of breeding and its application to the breeder genetic algorithm (bga). *Evolutionary Computation*, 1(1):335–360, 1993.

[54] N.J. Nilsson. A mobile automaton: An application of ai techniques. In *Proceedings of the 1969 International Joint Conference on Artificial Intelligence (IJCAI'69)*, 1969.

[55] T.J. Prescott. Spatial representation for navigation in animats. *Adaptive Behavior*, 4(2):85–125, 1996.

[56] J. Rosenblatt. Damn: A distributed architecture for mobile navigation. In *Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*. AAAI Press, March 1995.

[57] A. Saffiotti, K. Konolige, and E.H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1/2):481–526, 1995.

[58] A. Saffiotti, E.H. Ruspini, and K. Konolige. Integrating reactivity and goal-directedness in a fuzzy controller. In *Proceedings of the 2nd Fuzzy-IEEE Conference*, 1993.

[59] C. Sierra, R. López de Màntaras, and D. Busquets. Multiagent bidding mechanisms for robot qualitative navigation. *Lecture Notes in Computer Science. Intelligent Agents VII (Proceedings of ATAL'00)*, 1986:198–212, 2001.

[60] R. Simmons and S. Koenig. Probabilistic robot navigation in paritally observable environments. In *Proceedings of IJCAI-95*, pages 1080–1087, 1995.

[61] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. *Autonomous Robot Vehicles*, pages 167–193, 1990.

[62] H. Stone. *Introduction to Computer Architecture*. SRA, 2nd edition, 1980.

[63] R. Sun and C. Sessions. Bidding in reinforcement learning: A paradigm for multi-agent systems. In O. Etzioni, J.P. Müller and J.M. Bradshaw, editor, *Proceedings 3rd Annual Conference on Autonomous Agents*, pages 344–345, Seattle, 1999.

[64] R. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.

[65] S. Thrun. Bayesian landmark learning for mobile robot localization. *Machine Learning*, 33(1):41–76, 1998.

[66] S. Thrun. Learning Metric-Topological Maps for Indoor Mobile Robot Navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[67] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Henning, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in rhino. In D. Kortenkamp, R.P. Bonasso, and R Murphy, editors, *AI and Mobile Robots: Case Studies of Successful Robot Systems*, pages 21–52. MIT Press, 1998.

[68] D. Zipser. Biologically plausible models of place recognition and goal location. In J.L. McClelland and D.E. Rumelhart, editors, *Parallel Distributed Processing: Explorations in the Micro-Structure of Cognition*, Vol. 2, pages 432–470, Cambridge, MA, 1986. Bradford Books.