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.

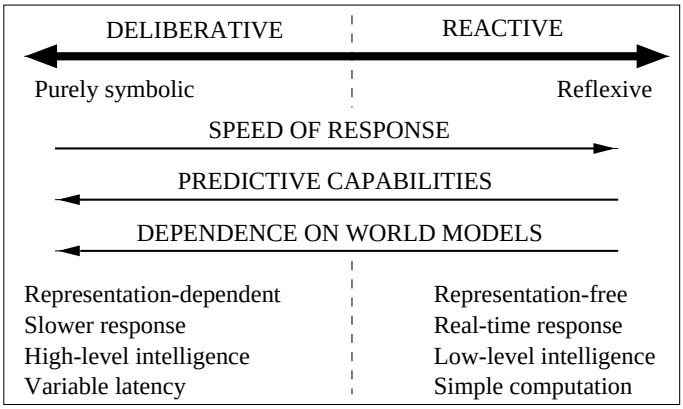

Figure 2.1: Control architectures' spectrum

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.

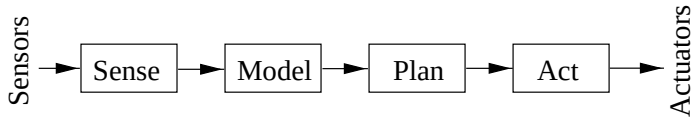

Figure 2.2: Sense-plan-act model

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

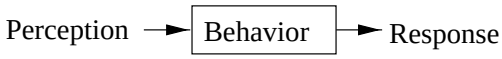

Figure 2.3: Single behavior diagram

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).

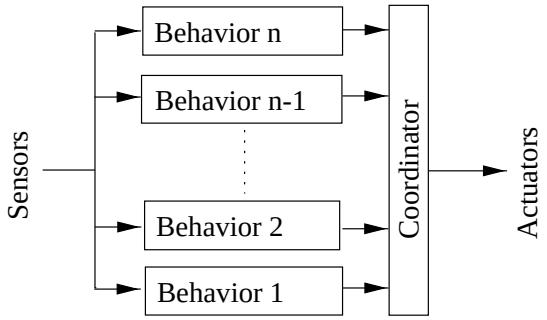

Figure 2.4: Behavior-based architecture

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.

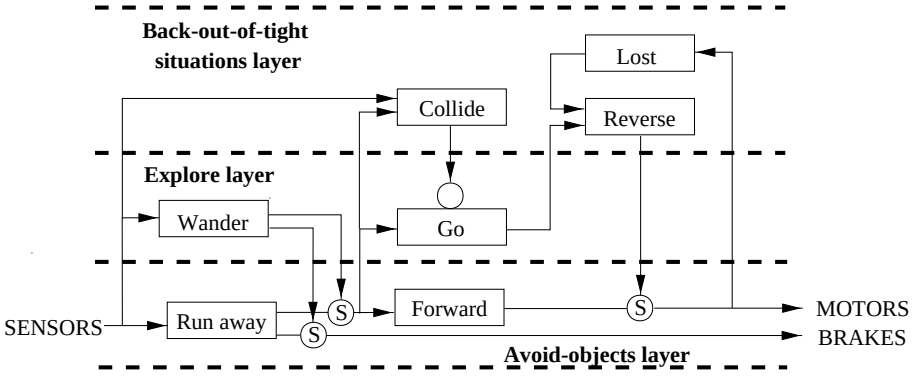

Figure 2.5: Example of a control system using the subsumption architecture. Each box is an AFSM, and signals are passed through the arrows connecting the AFSMs. An encircled S is a suppression point, and an empty circle is a reset point

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.

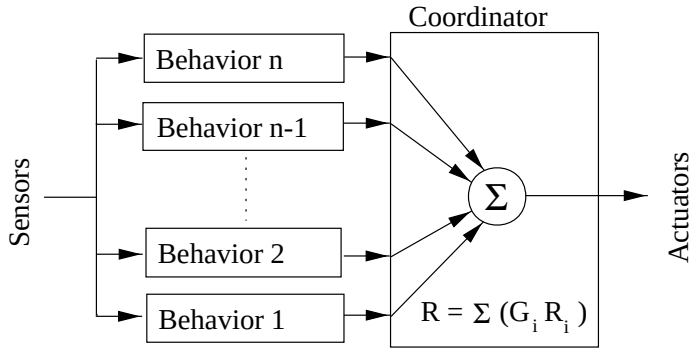

Figure 2.6: Motor-schemas architecture

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.

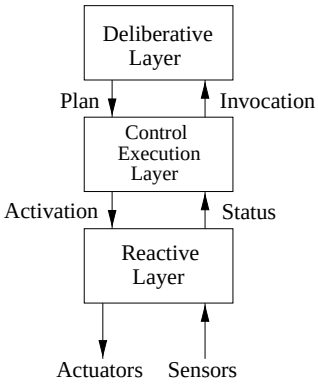

Figure 2.7: Three layers hybrid architecture

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.