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

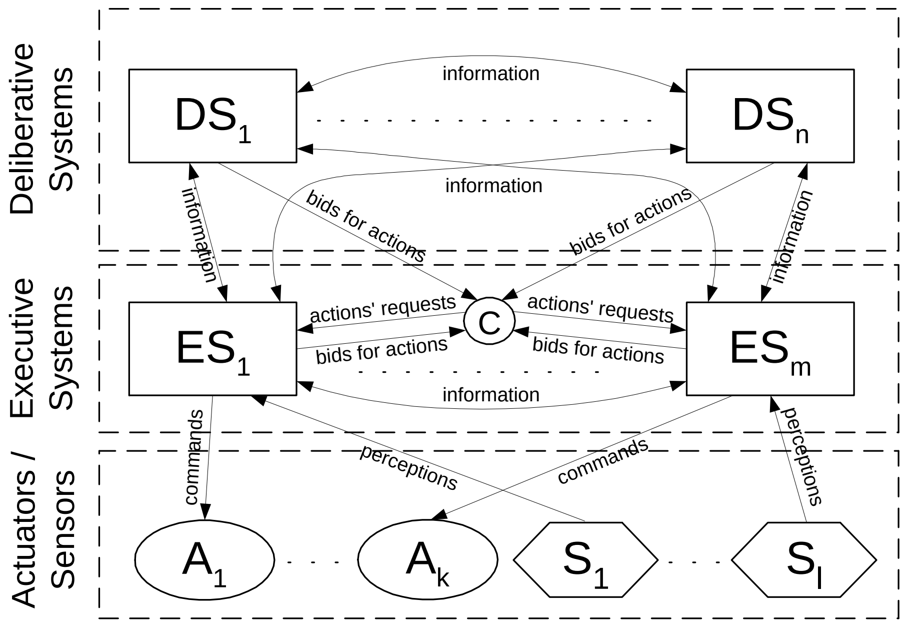

Figure 4.1: General bidding coordination architecture

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.

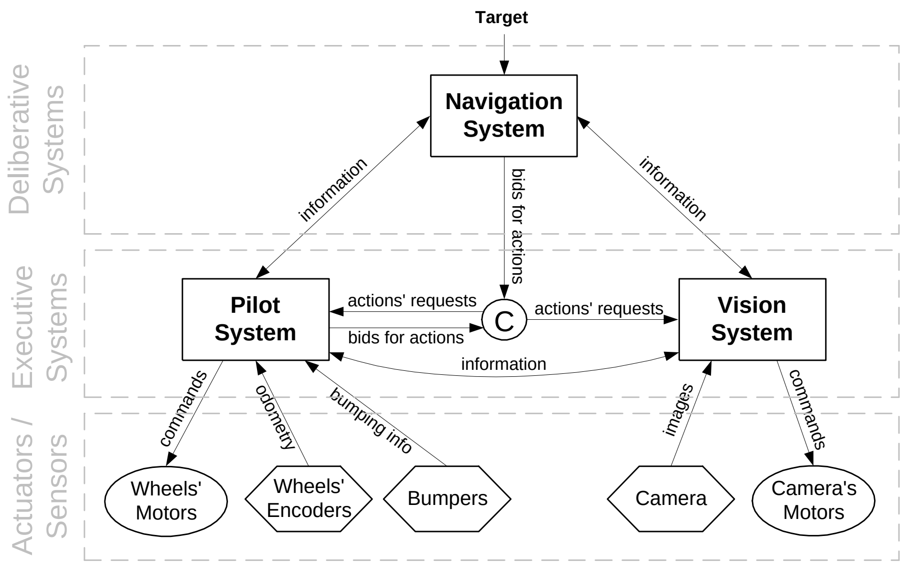

Figure 4.2: Specific robot architecture

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.

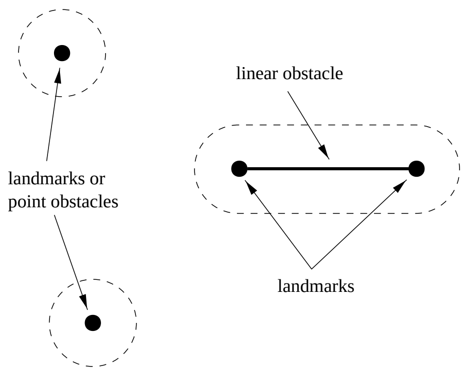

Figure 4.3: Growing obstacles. Points and solid lines are the obstacles; dotted lines show grown obstacles

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.

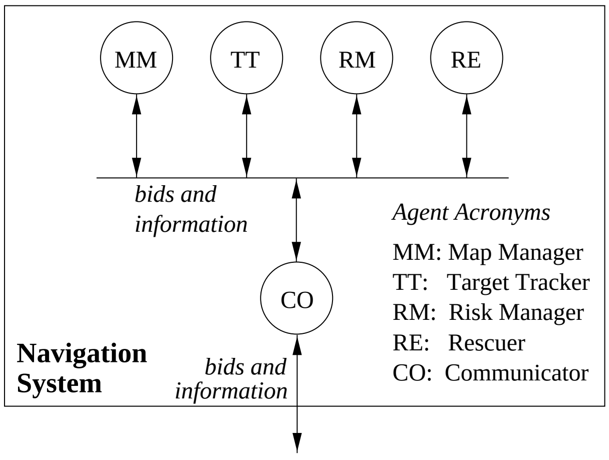

Figure 4.4: Multiagent view of the navigation system

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 (≤ 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

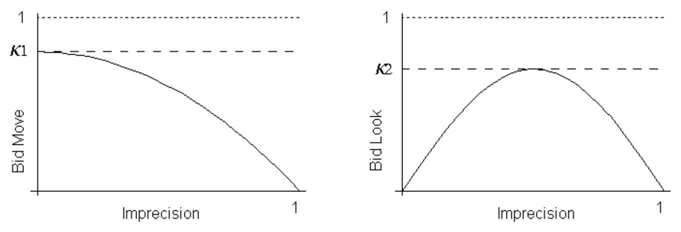

Figure 4.5: *Target Tracker*'s bidding functions

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

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

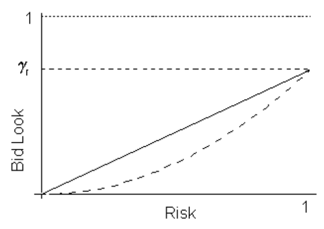

Figure 4.6: *Risk Manager*'s look bidding functions (look ahead -solid line- and look behind -dashed line-)

(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($\nu$,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($\theta$))
            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,$\theta, \epsilon_\theta$,dist,$\epsilon_{dist}$)) 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
            $I_a^l := (\frac{\epsilon_\theta^l}{2\pi})^\beta$
            $I_a^r := (\frac{\epsilon_\theta^r}{2\pi})^\beta$
            $I_a := max(I_a^l, I_a^r)$
            angle_move := $(\theta_l + \theta_r)/2$
            If $I_a^l > I_a^r$ then angle_look := $\theta_l$
                else angle_look := $\theta_r$
                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($\theta_l, \theta_r$)
                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,$\{$(Look($random\_angle\left(\left[-\frac{\pi}{4}, +\frac{\pi}{4}\right]\right), \gamma_r R)\}$)
            **else if** $|B| < 6$ **then**
                inform(self,CO,$\{$(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($\theta$)) 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.