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

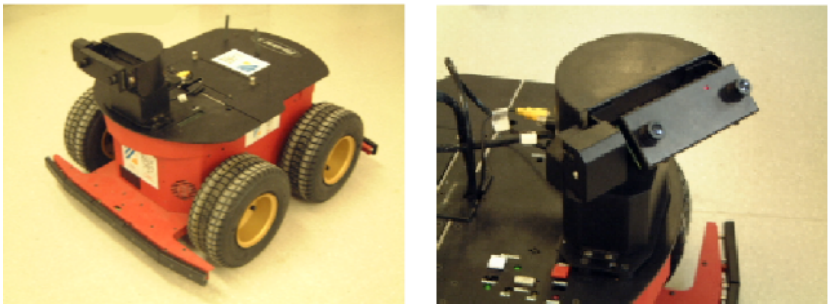

Figure 6.1: *Left*: *MarkFinder* robot. *Right*: Detail of the pan and tilt unit with two B&W cameras

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

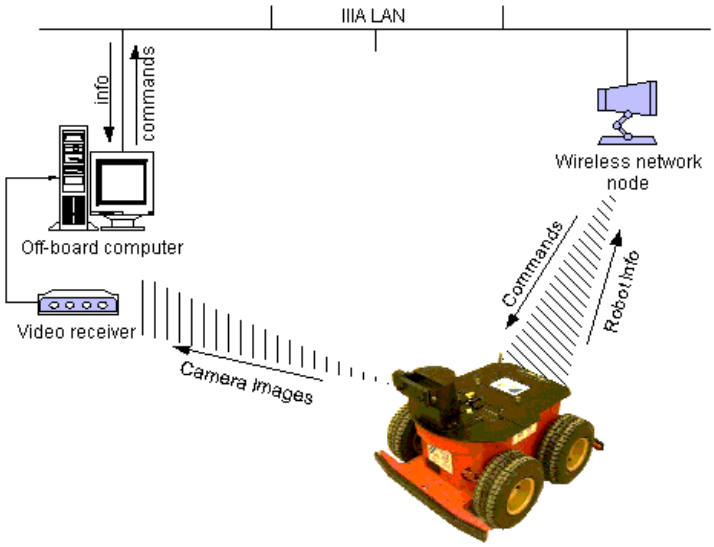

Figure 6.2: Communication with the robot

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

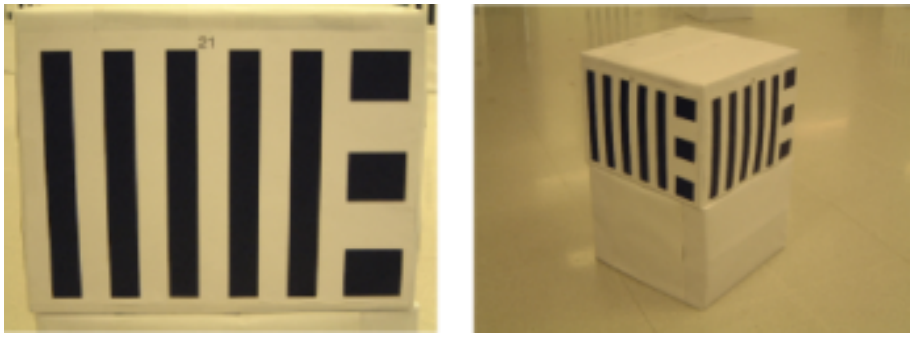

Figure 6.3: *Left*: Landmark label 21 (code = 10101 = 21). *Right*: One of the boxes with multiple landmark labels

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

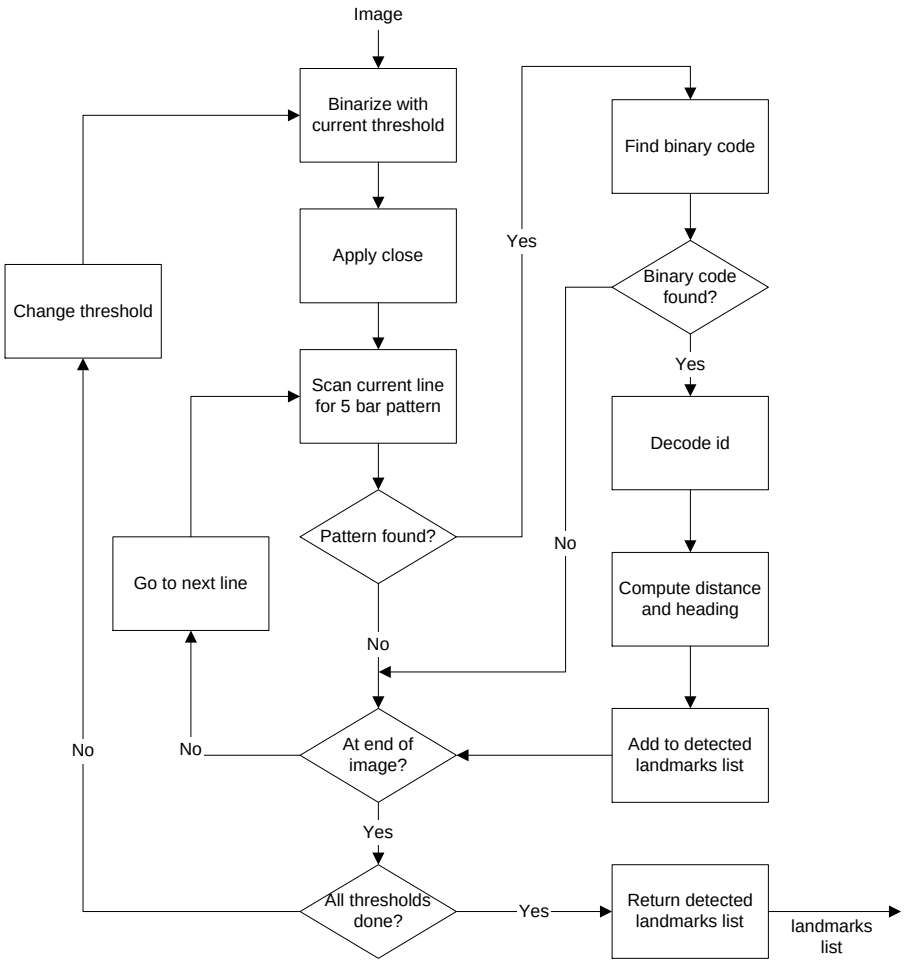

Figure 6.4: Landmark recognition process

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,

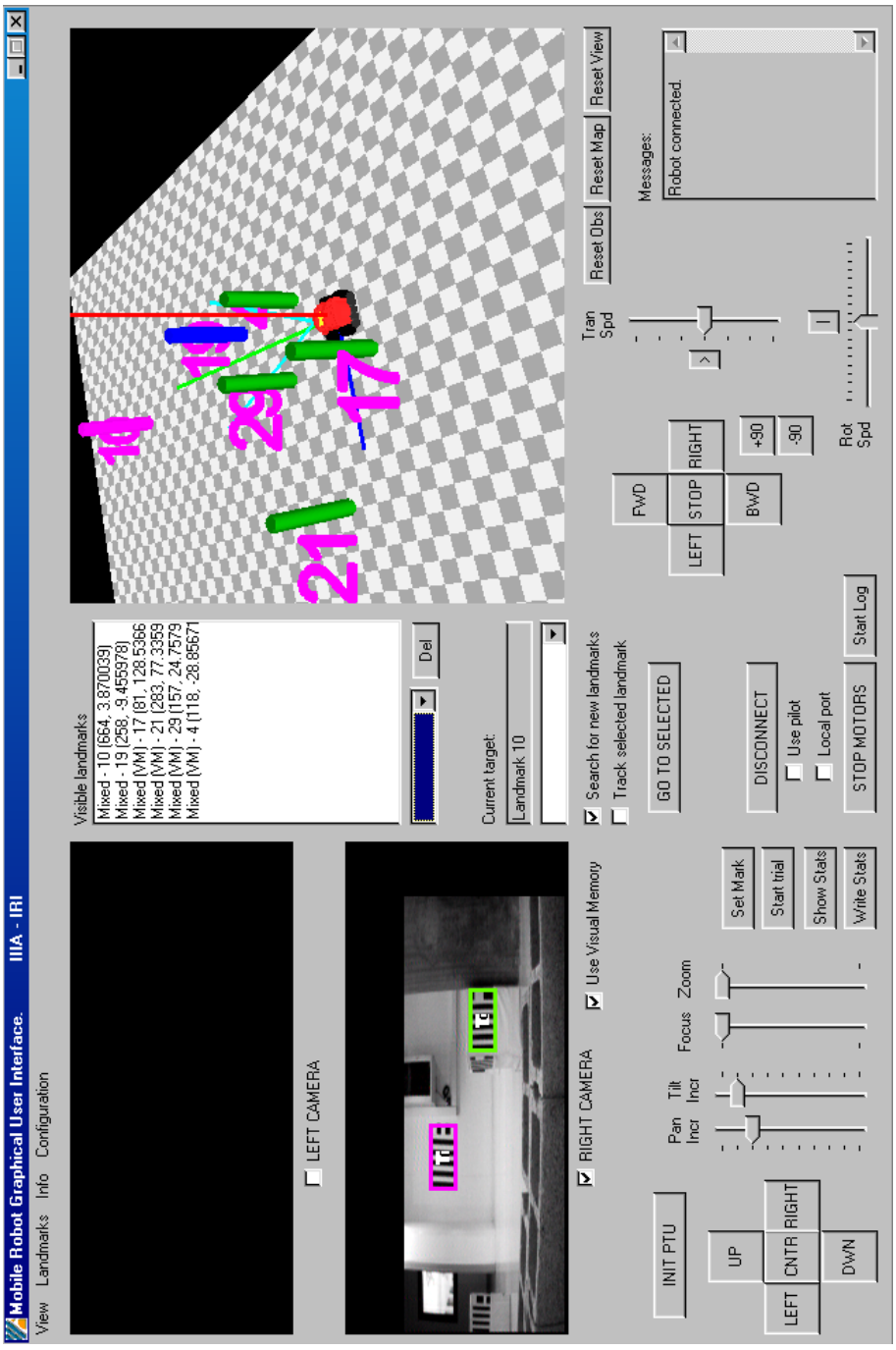

Figure 6.6: Graphical control interface

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,

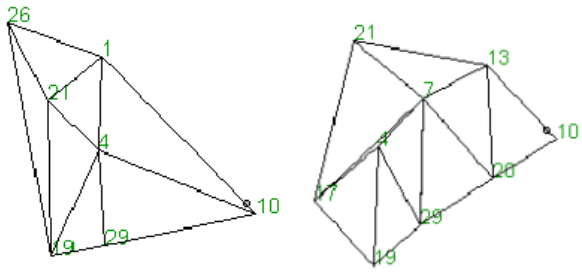

Figure 6.8: Maps of 2 different scenarios of scenario class 3

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

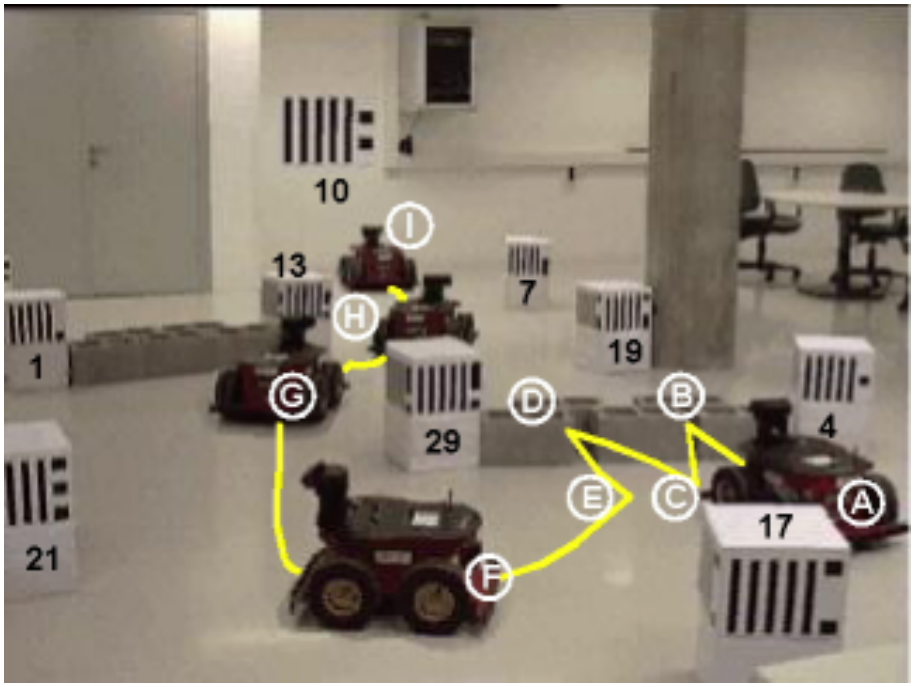

Figure 6.9: Path followed during the trial. See explanation of relevant points on the text

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.

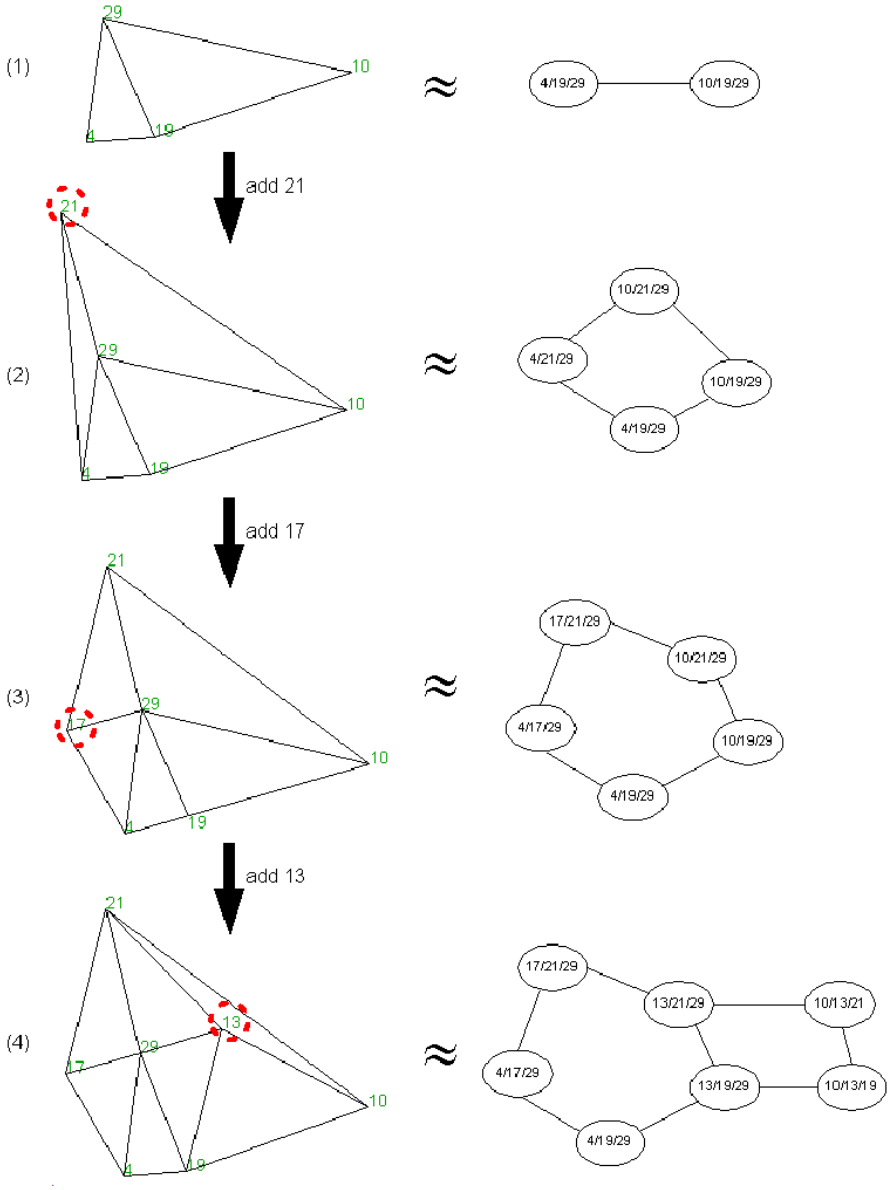

Figure 6.10: Map created during the trial

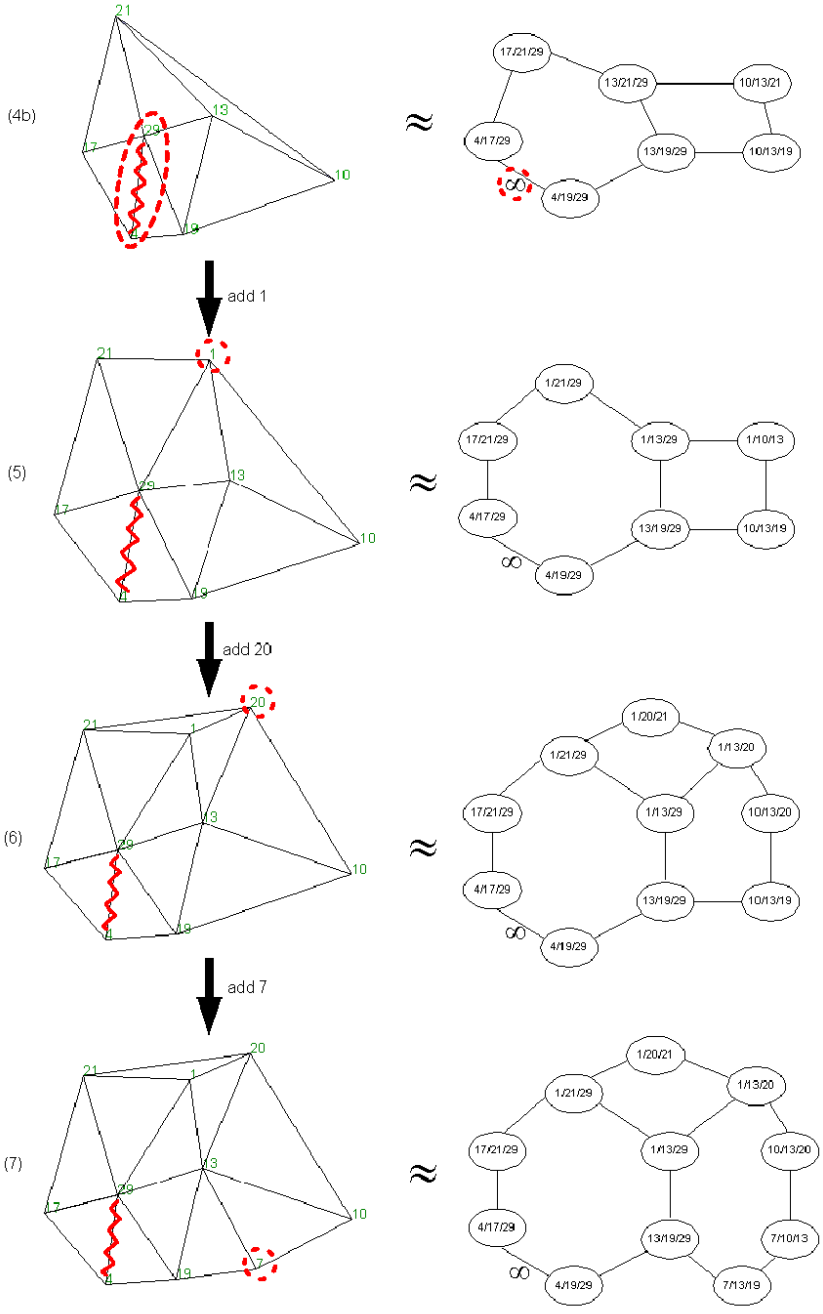

Figure 6.11: Map created during the trial (cont.)

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

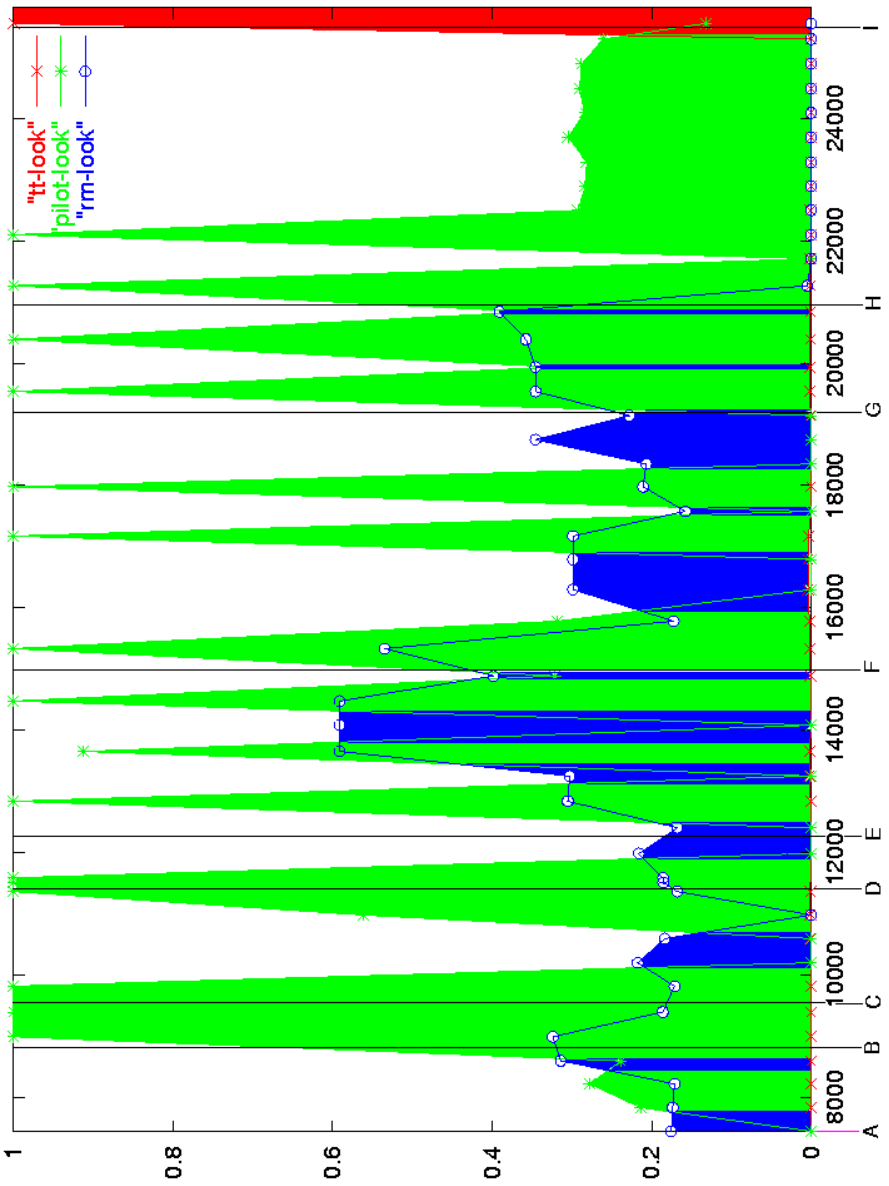

Figure 6.13: Looking bids. *Target Tracker* in red, Pilot in green and *Risk Manager* in blue

- **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

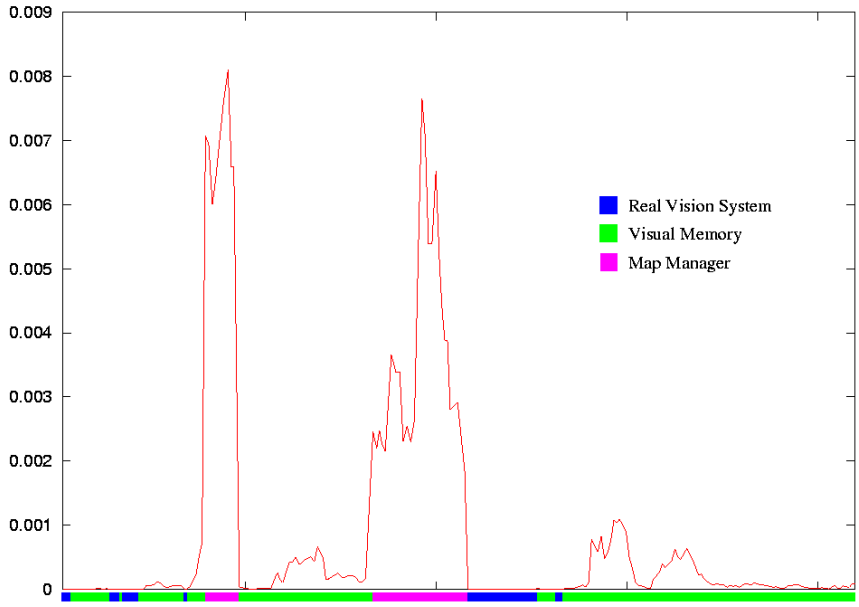

Figure 6.14: Evolution of the target's location imprecision and sources of computation

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.