Vision-Based Navigation Using Natural Landmarks

Andrew Howard
Department Of Computer Science
University of Melbourne
Parkville 3052
Australia
andrbh@cs.mu.oz.au

Les Kitchen
Department Of Computer Science
University of Melbourne
Parkville 3052
Australia
ljk@cs.mu.oz.au

Abstract

MYNORCA is a vision-based navigation system for mobile robots, designed principally for operation in indoor environments. The system uses vision for detecting obstacles and locating natural landmarks. In addition, it is able to solve navigation problems in which the robot's initial location is completely unknown. In this paper, we present an overview of MYNORCA, describe its implementation and present some experimental results.

1 Introduction

MYNORCA is a vision-based navigation system for mobile robots, designed principally for operation in indoor environments, such as office buildings. MYNORCA divides the overall navigation problem into two parts: local and global navigation. Local navigation is defined as the immediate problem of detecting and avoiding obstacles, whilst global navigation is defined as the problem of reaching distant goals. This division of responsibilities has a double benefit: global navigation can be treated as a fairly abstract planning problem, since all the messy details of actually detecting and avoiding obstacles can be delegated to the local navigation subsystems; and local navigation can be treated as a reactive problem, in which no long-term planning required.

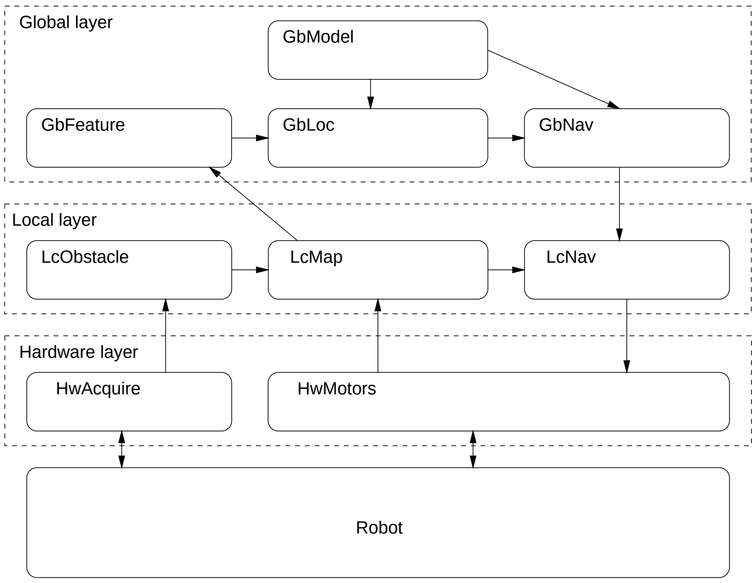
MYNORCA makes use of a *global model* containing two different kinds of information. Firstly, it contains *landmarks*, such as walls and doorways, which can be matched to observed landmarks to help determine to robot's location. These are *natural* landmarks; no modification of the environment is required. Secondly, the model contains *connections*, which indicate how different parts of the environment are connected to each other. This information is used to plan paths. Presently, the global model must pre-defined.

MYNORCA also adds a twist to the standard navigation problem. Normally, this problem is stated in the following terms: 'given a robot that is at some known initial location, plan and execute a series of actions that will take the robot to the goal'. The important assumption here is that the robot's initial location is *known*. Unfortunately, in practice, robots tend to get lost, usually as a result of sensor noise. Under these circumstances, the robot may have to execute a complex series of actions in order to 're-localise' itself. Although re-localisation algorithms do exist [Dudek *et al.*, 1995], we have chosen instead to restate the navigation problem. Our version is as follows: 'given a robot that is at some *unknown* initial location, plan and execute a series of actions that will take the robot to the goal'. That is, we attempt to develop an single algorithm that can be used for both re-localisation and navigation; one that does not make any distinction between these two processes [Howard and Kitchen, 1996]. Such an algorithm incorporated into MYNORCA, and is described in this paper.

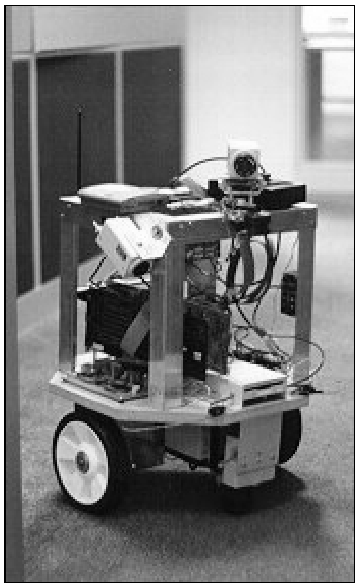
This paper is intended as an overview only. In the following sections, we describe each element of the system and how they work together as a whole. We also discuss the implementation of MYNORCA on Robot J Edgar and present some experimental results showing the system in action.

2 System Overview

MYNORCA employs a highly modular agent-oriented control system, inspired in part by the *subsumption architecture* approach [Brooks, 1986]. Each agent is responsible for a specific task and communicates with other agents via a message passing mechanism. Agents are grouped into layers, with lower layers carrying out routine hardware-oriented tasks, such as image acquisition, and higher layers carrying out more abstract tasks, such as navigation. There are two key advantages to this design. Firstly, agents can be developed and tested independently, greatly simplifying the development process. Secondly, the system can be distributed, with some agents running on the mobile robot and others running



(a)



(b)

Figure 1: (a) System overview. Each agent is represented by a box, with arrows indicating the flow of information between agents. (b) Robot J Edgar

on a remote host or hosts. This minimises the computational requirements of the robot's on-board processor.

Figure 1 shows the various agents that make up MYNORCA. The agents are divided into *hardware*, *local* and *global* layers as follows.

- The hardware layer is responsible for low-level tasks, such as motor control, odometry and image acquisition. It presents an abstracted interface to higher layers so that, in principle, MYNORCA can be implemented on any platform that supports the required hardware functionality.
- The local layer is responsible for tasks involving the local environment (the area within a few meters of the robot). Agents in this layer carry out tasks such as obstacle detection, avoidance and map building.
- The global layer is responsible for tasks involving the global environment. Agents in this layer carry out tasks such as determining the robot's location and navigating to distant goals.

Since the hardware layer has only very simple tasks to perform, it will not be discussed further in this paper. The local and global layers are described in more detail in the following sections.

3 The Local Layer

The local layer has three agents: LcObstacle, LcMap and LcNav. Collectively, these agents are capable of carrying out local navigation tasks. The basic process is as follows. The LcObstacle agent analyses images to determine the location of obstacles. This information is fed into the LcMap agent, where it is used to form a set of *local maps*. These maps are used by the LcNav agent to determine collision free paths. This layer can very reliably navigate the robot down corridors, through doorways and around rooms.

3.1 Obstacle Detection

The LcObstacle agent processes images to detect obstacles. In order to achieve reasonable speed (at least 10 frames per second on a Pentium-class computer), certain assumptions must be made. Specifically, we have assumed that the robot is operating in an indoor environment in which all carpeted areas can be regarded as free-space, and all non-carpeted areas can be regarded as obstacles. This is a common assumption with vision-guided robots [Horswill, 1993] and works reasonably well in most indoor environments. The methods employed by the LcObstacle agent are described in detail in [Howard and Kitchen, 1997].

3.2 Mapping

The LcMap agent combines the obstacle data produced by LcObstacle to form an *occupancy map* of the robot's local environment. This is a grid-based map in which each cell has a value indicating the probability that it is occupied, i.e. that there is some obstacle at that location [Howard and Kitchen, 1997] [Elfes, 1990] [Moravec, 1988]. Figure 2 shows an occupancy map generated dur-

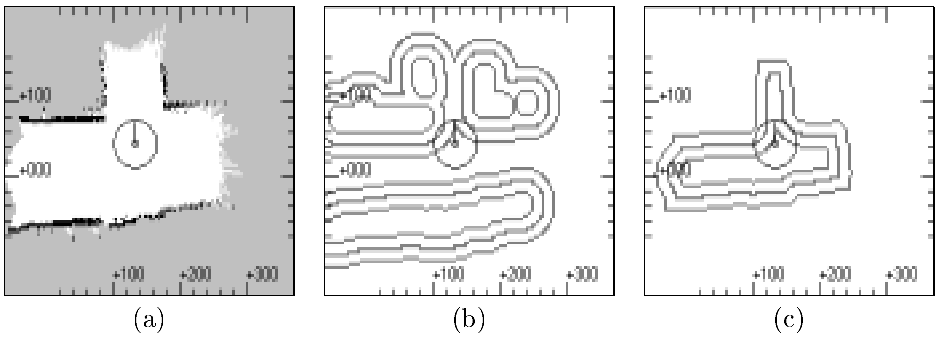


Figure 2: Local maps. (a) Occupancy map. Black cells are probably occupied, white cells are probably unoccupied, grey cells are unknown. (b) Upper proximity map. The lines indicate contours of equal proximity. (c) Lower proximity map. The lines indicate contours of equal proximity.

ing an experimental run. In this map, cells which are probably occupied are black, cells which are probably unoccupied are white, and cells whose occupancy state is unknown are grey.

Since the raw occupancy map is not particularly useful for navigation tasks, the LcMap agent produces another kind of map: a *proximity map*. This is a grid-based map that lists, for each cell, the distance to the nearest obstacle (a proximity map can be thought of as a generalised *configuration space* map [Lozano-Perez and Mason, 1984]). Forming the proximity map requires knowledge of the location of each and every obstacle in the robot's local environment; knowledge which is, in principle, encoded in the occupancy map. Unfortunately, in practice, the occupancy map will always contain regions whose occupancy state is unknown. This may be the result of occlusion (by a wall, for example), or it may be that the robot has simply failed to point the camera at these regions. It is not possible to form a single, unique, proximity map from such incomplete data.

It is possible, however, to generate an upper and lower *bound* on the proximity. The LcMap agent therefore generates two proximity maps. The first lists the distance to the nearest occupancy map cell that is *definitely* occupied. The second lists the distance to the nearest cell that *may be* occupied. In effect, the first map assumes that unknown cells are unoccupied and generates a proximity map on that basis, whilse the second map assumes that unknown cells are occupied. The true proximity value must lie somewhere between these two extremes. Figure 2 shows the two proximity maps generated during an experimental run.

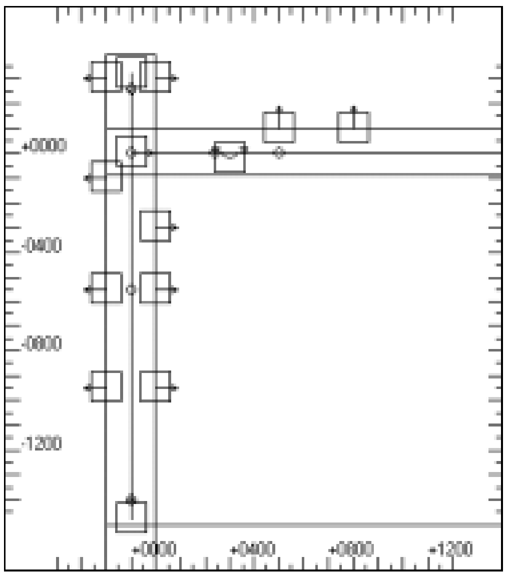
3.3 Navigation

The LcNav agent is responsible for reaching local goals. It looks for a sequence of in-place turns and straight-line motions that lead from the robot's current location to the goal location. Collision free paths are determined by analysing the proximity maps generated by the LcMap agent: the robot cannot enter into any cell whose proximity value is less that the robot's maximum radius. Of course, since there are two proximity maps, representing upper and lower bounds on the true proximity value, we must chose which map to use. The lower bound will produce very conservative behaviour (the robot will not go anywhere unless it is absolutely sure that there are no obstacles in its path), while the upper bound will produce very optimistic behaviour (the robot will zoom off into areas where there may of may not be obstacles). Agents in the global layer can switch the LcNav agent between these two modes of behaviour, depending upon the circumstances in which the robot finds itself.

In addition to its goal-seeking behaviour, LcNav has an information-seeking behaviour. In this mode, the agent attempts to acquire information about the robot's local environment. It continually analyses the occupancy map, looking for unknown regions that are not occluded. When such a region is detected, the agent temporarily suspends its goal-seeking behaviour, stops the robot and points the camera at the detected region. This behaviour ensures that the occupancy map always contains as much information as is possible. Agents in the global layer, which look for landmarks in the local map, rely on this behaviour to detect certain landmarks. If this behaviour is disabled, landmarks such as doorways may be missed.

4 The Global Layer

The global layer contains four agents: GbModel, GbFeat, GbLoc, GbNav. Collectively, and in association with the agents in the local layer, these agents carry out global navigation tasks. The process is as follows. The GbLandmark agent extracts significant *landmarks*, such as doorways or intersections, from the local map. These landmarks are passed to the GbLoc agent, which compares them with landmarks in a *global model* to de-



```
LOCATION Loc1 AT (0 0)
LOCATION Loc2 AT (0 260)
LOCATION Loc3 AT (340 0)
LOCATION Loc4 AT (0 -1400)
LOCATION Loc5 AT (600 0)
LOCATION Loc6 AT (0 -550)
LOCATION Loc7 AT (+1900 0)
LOCATION Loc8 AT (+1900 -1600)
CONNECT Loc1 Loc2
CONNECT Loc1 Loc3
CONNECT Loc1 Loc6
CONNECT Loc6 Loc4
CONNECT Loc3 Loc5
CONNECT Loc5 Loc7
CONNECT Loc7 Loc8
FEATURE Wall AT (0 0 0) SIZE (10000 10000 15)
FEATURE Wall AT (0 0 90) SIZE (10000 10000 15)
FEATURE Wall AT (0 0 180) SIZE (10000 10000 15)
FEATURE Wall AT (0 0 -90) SIZE (10000 10000 15)
FEATURE Int AT (0 0 0) SIZE (60 60 30)
FEATURE Tee AT (0 0 0) SIZE (60 60 30)
FEATURE Tee AT (0 320 -90) SIZE (60 60 30)
FEATURE Tee AT (0 -1480 +90) SIZE (60 60 30)
FEATURE Tee AT (400 -20 +150) SIZE (60 60 30)
FEATURE Tee AT (400 -20 +30) SIZE (60 60 30)
                    ⋮
```

Figure 3: Global model of a section of corridor. Small boxes indicate the location of landmarks (principally doorways). Circles indicate the end-points of connections. A textual description of this model is shown on the right.

termine the location of the robot. The GbNav agent uses the location information maintained by the GbLoc agent, together with information in the global model, to plan and execute paths to the goal. The global model is maintained by the GbModel agent.

Both the GbLoc and GbNav agents are complicated by the fact that the robot's initial location is unknown: it is merely assumed that it is somewhere within the model. Consequently, at any given time, the robot may be at any one of a number of possible locations that are compatible with the landmarks that have been observed. The GbLoc agent must therefore maintain multiple possible robot locations, and the GbNav agent must be able to make plans which are compatible with the robot being at any of these locations.

4.1 The Global Model

The global model stores two types of objects: landmarks and connections. Landmarks are used to localise the robot. These are always *local* landmarks, that is, landmarks that can be detected by the robot from a single location. Thus, the model may store a *wall-segment* landmark, representing a small section of a wall, but will never store a landmark representing the wall in its entirety. Other kinds of landmarks that the model may store are doorways and intersections.

For each landmark, the model lists the landmark *pose* (position and orientation) in a global coordinate system. This system is arbitrary, but is useful for capturing geometrical relationships between landmarks, such as relative orientation. In addition, the model stores an *uncertainty value* for both the landmark position and orientation. This uncertainty value has two important uses. Firstly, it allows the model to capture the fact that certain landmarks have well determined positions, but poorly determined orientations, or vice-versa. A wall-segment landmark, for example, always has a well determined orientation, but may have a poorly determined position (since a single wall-segment may be used to represent an entire wall). Secondly, it allows the precision of the model to be varied: in some areas, we may wish to know the exact pose of every doorway; in others, it may suffice to indicate that there are doorways 'in the general area'.

The model also stores connections, which are used for navigation. A connection is a declaration that two points are connected by some action. For each connection, the model stores the start and end points and an action that connects the points. These actions are *local* actions, that is, actions that can be sent to the LcNav agent for execution.

Figure 3 shows a sample model, in both graphical and textual format.

4.2 Landmark Extraction

The GbFeat agent extracts landmarks from the local map generated by LcMap. The agent has a set of predefined templates, corresponding to landmarks such as wall-segments and doorways, and uses these templates to search for landmarks in the local map. Empirically, we have found that a small set of relatively abstract templates can capture useful information about the environment. For example, there is no single template corresponding to a 'corridor intersection'; rather, each particular intersection can be represented by a collection of simpler landmarks.

The template matching procedure is straight-forward. Each template consists of a set of *sample points* which list a location (relative to the template origin) and a *match criterion*. The match criterion is used to evaluate the match error, and is usually framed in terms of a condition on the proximity value. For example, a sample point that is meant to correspond to part of a wall may specify that the proximity must be less than 10cm. On the other hand, a sample point that is meant to correspond to an opening might specify that the proximity value must be greater than 40cm. In the first case, the match error is lowest for small proximity values; in the second it is lowest for large proximity values. The match error for the template as a whole is given by the sum of the individual sample point error terms.

To locate a landmark, the GbLandmark agent places a template at a randomly chosen initial location and orientation, then uses gradient descent to minimise the match error. A landmark is detected whenever the match error drops below some threshold. Once found, the landmark can be 'tracked' with minimal computational cost. Figure 4 shows a series of local map snapshots taken from an experimental run, with the detected landmarks indicated by open circles.

The only complicating factor in this procedure is that the proximity value is always uncertain: only the bounds on the proximity value (as expressed by the two proximity maps) are known. We do not want the GbLandmark agent to report landmarks that *might be* present; we only want those landmarks which *are* present. Consequently, the GbLandmark agent has been designed to evaluate worst-case matches. If, for example, a sample point specifies that the proximity value must be less than 10 cm, the upper proximity bound will be used to evaluate the match error. On the other hand, if a sample point specifies that the proximity value must be greater than 40 cm, the lower proximity bound will be used.

4.3 Localisation

The GbLoc agent attempts to keep track of the robot's global pose. Recall, however, that we have assumed that the robot's initial pose is unknown. Therefore, rather than maintaining a single pose estimate, the GbLoc agent maintains a *set* of pose estimates. This set is updated whenever a landmark is detected, or the robot moves.

The detection of a landmark will generally eliminate some of the pose estimates. The detected landmark can be compared with similar landmarks in the global model to determine which of the current pose estimates are incompatible with the observed landmark. For example, if the robot observes a nearby doorway, all those estimates that do not place the robot near a doorway can be eliminated. This works for orientation also. If, for example, the robot detects a wall segment oriented in a particular way, all those pose estimates that do not indicate a nearby wall segment at the appropriate orientation can be eliminated.

The pose estimates will also change as the robot moves. Whereas the detection of a landmark will reduce the number of pose estimates, the uncertainties associated with robot motion (i.e. odometric errors) will increase the number of estimates.

Note that the GbLoc agent does not use the *absence* of landmarks to eliminate pose estimates. This is for the sake of robustness: it is quite likely that the GbLandmark agent will fail to detect some landmarks (because of noise in local map, for example), but quite *unlikely* that it will detect landmarks that are not really present. That is, the probability of false negatives is far higher than the probability of a false positives. A natural generalisation to the localisation technique described here would be to quantify these probabilities and use them to maintain a *probability distribution* over a global pose space. We have avoided this particular generalisation because it is difficult to find efficient implementations.

By using using only the presence of landmarks, and not their absence, the GbLoc agent can also cope with *changing environments* (at least in a limited sense). If landmarks are carefully selected, changes in the environment will manifest themselves not as new landmarks, but as the failure to detect an old one. The canonical example is a doorway : when open, a doorway landmark will be detected; when closed, nothing will be detected. In the former case, the GbLoc agent will be able to eliminate some pose estimates; in the latter, the set of pose estimates will remain unchanged.

Occasionally, a landmark will be detected for which there are no compatible pose estimates. This may be the result of an false positive error on the part of the GbLandmark agent, or it may indicate that the global model is incomplete in some way. In either case, the robot is effectively 'lost'. Presently, we handle this situation by resetting the set of pose estimates to include all possible poses; i.e. we assume that the robot could be anywhere. This works reasonably well (the robot will always reach the goal sooner or later), but better responses are possible [Howard and Kitchen, 1996]. This area remains the subject of ongoing research.

Finally, note that this localisation scheme can make use many different kinds of landmarks, not just those extracted from the local map by GbLandmark. The colour, brightness or even the smell of a place could be stored in the global map and used for localisation purposes.

4.4 Navigation

The general navigation problem that must be solved by the GbNav agent is quite complex. The robot's initial pose is complete unknown, so it cannot simply plan and execute a series of actions that lead from the initial robot location to the goal location. Instead, it must plan and execute a series of actions at whose conclusion all of the remaining possible robot poses correspond to the goal location. In other words, it is not sufficient for the robot to reach the goal; the GbNav agent must *know* that it has reached the goal.

In principle, the optimal series of actions can only be found by considering all possible poses the robot might have, all possible actions it might take, and all possible outcomes of these actions. If effect, the optimal series of actions can only be found by searching the space of all possible *sets of pose estimates*, using the information stored in the global model to determine the connectivity of points in this space. Clearly, this space is vast and such a search would require vast computational resources. Therefore, we have devised a sub-optimal navigation strategy that will enable the robot to reach the goal location in a reasonable amount of time, on most occasions, in most environments.

The strategy is as follows. The GbNav agent chooses, at random, one candidate pose from among those that are possible. It then plans and executes a series of actions that will take the robot to the goal location, if the robot is truly has this pose. These actions are in a form that can be passed on to the LcNav agent for execution. As the robot moves, the candidate pose is shifted so that it remains consistent with the robot's motion. There are two possible outcomes of this process: either a point will be reached where the candidate pose corresponds to the goal location, or else some landmark or landmarks will be observed that causes the GbLoc agent to conclude that the candidate pose is not a possible pose. In either case, the GbNav agent can pick another candidate pose and repeat the process. This procedure only terminates when all possible poses correspond to the goal location.

There are certain environments in which this strategy will fail. The GbNav agent can become locked into an infinite loop, in which the robot shuttles back and forth between two locations, unable to determine which is the true goal. Detecting and coping with this circumstance is the subject of ongoing research. There are also pathological environments in which this strategy will work, but will take a great deal of time. Such environments are usually characterised by large repeating units. An multi-storey car park, in which every floor except for the first and last has an identical layout, is a good example of such an environment.

Note that the approach we have taken with this navigation strategy does not emphasise localising the robot before attempting to reach the goal. Rather, localisation is seen as the natural outcome of such an attempt. In practice, most environments are constructed in such a way that all but a few possible robot poses will be quickly eliminated; most of the time is spent deciding which of the few remaining poses is the correct one, and physically moving the robot to the goal.

5 Implementation and Experiments

MYNORCA has been implemented and tested on a medium-sized mobile robot which goes by the name of J Edgar. This robot has a pair of independent drive wheels, a pan head and a single monochrome camera. It has an on-board computer with a frame-grabber and is able to communicate with a base-station via a UHF data-link. Both the on-board computer and the base-station are Pentium-class PC's.

All agents have been implemented in C++. A number of other languages, including Java, were considered, but in the end C++ was chosen because of its maturity, relative portability, and speed. Hardware layer agents execute on the robot, as does the obstacle detection agent, LcObstacle. All other agents execute on the base-station. This particular distribution of agents was chosen because it minimises data-link bandwidth requirements. The LcObstacle agent is able to encode the information it extracts from each image into less than 128 bytes, whereas transmitting the raw image back to the base-station would require many kilobytes. With a maximum transfer rate of 115 kb/s, the data-link is able to transmit data faster than the LcObstacle agent can generate it. Currently, the system can handle up to 10 frames per second, and is limited by the speed of the frame-grabber, not the speed of the data-link.

There are a number of significant advantages to having most of the agents executing on the base-station. Firstly, it greatly simplifies debugging – rather than having to employ program traces and the like, standard interactive debugging tools can be used. Secondly, it is possible to construct sophisticated user-interfaces that allow real-time monitoring of agent state. We have found that the ability to distribute and debug agents in this way has greatly reduced development time and improved the overall reliability of the system. It also reduces the computational requirements of the robot's on-board processor, where CPU power is at a premium.

Figure 4 shows a series of snapshots taken during an experimental run. In this experiment, the robot was place in a corridor and given the task of reaching a relatively close goal (about 10m from it's starting position). The first series images shows the local map at various times, with the robot's motion and the detected landmarks indicated. The second series of images shows the global pose estimate. At $t = 0$ seconds, the robot's pose

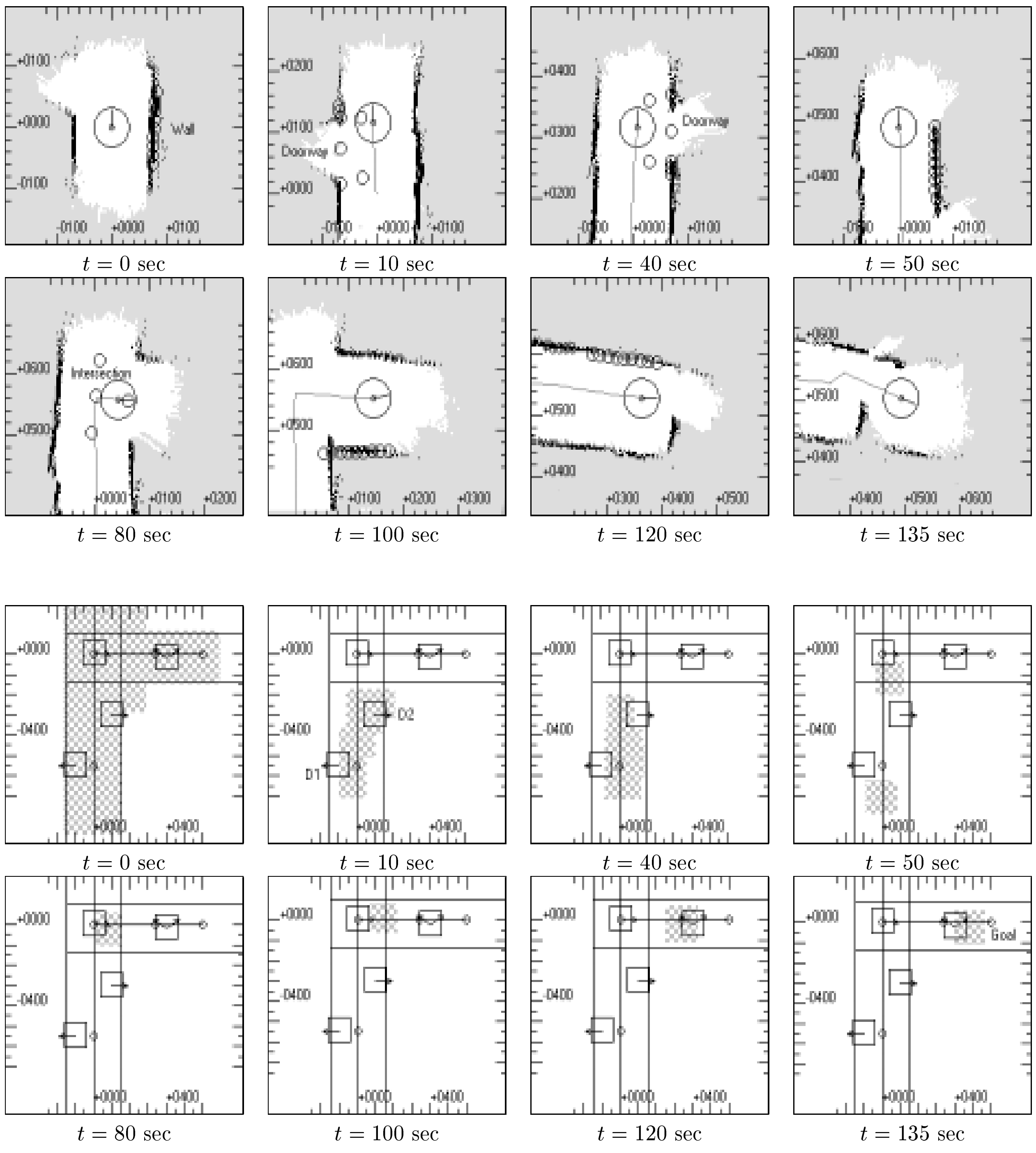


Figure 4: Experimental run. The upper set of images is a series of snapshots of the local occupancy map. The robot's current location and path is also indicated. The lower set of images is a series of snapshots showing the evolution of the global pose estimate. The cross-hatched region indicates possible robot locations in the global model.

is entirely unknown (the robot does not even know which direction it is facing). However, the detection of a doorway at time $t = 10$ seconds eliminates all but two possibilities: the robot is either next doorway D1, heading north, or else it is next to doorway D2, heading south. The robot assumes that the former is true and heads north, towards the goal. Another doorway is detected at time $t = 40$ seconds, but unfotunately this does not add any new information. It is the detection of an intersection at time $t = 80$ seconds that allows all but one possibility to be rejected. From this point, the robot proceeds directly to the goal. The maximum speed of the robot over this run was 20 cm/s, with an overall average speed of 7 cm/s. The low average speed is mainly due to the fact that the robot must stop to inspect both doorways and the intersection.

In other experiments, we have found that system copes well with presence of humans. If people keep moving, the system will ignore them (the mapping system is too slow to pick up a person walking at normal speed). If people insist on standing still, the robot will simply move around them. Occasionally, the presence of a human will cause the robot to detect a landmark that is not really there and the robot may become lost. The system generally recovers, but may spend a great deal of time doing so. We plan to quantify this recovery behaviour in the near future.

6 Conclusion

Although we have not yet conducted systematic tests, the experiments that have been conducted to date indicate that MYNORCA is a very reliable navigation system. We have identified two key factors required to produce this level of reliability. Firstly, the local map must be accurate. Failure to detect an obstacle can lead to a collision, or the detection of a false landmark. Secondly, the global model must be both accurate and complete. If this is not the case, the robot may become perpetually lost.

There are many extensions to this system that are the subject of ongoing research. We will mention just two here. Firstly, the system is being extended to cope with *changing environments*. In its current form, the system can successfully localise the robot in a changing environment (to some extent), but cannot generate effective navigation strategies. Secondly, we would like the system to be able to *learn* global models. Currently, models must be hand-coded. We are working on a procedure whereby the robot can be *trained* by an operator, but in the long run would like the system to be able to acquire models in an autonomous fashion.

References

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

[Dudek *et al.*, 1995] Gregory Dudek, Kathleen Romanik, and Sue Whitesides. Localizing a robot with minimum travel. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[Elfes, 1990] Alberto Elfes. Occupancy grids: A stochastic spatial representation for active robot perception. In *Proceedings of the Sixth Conference on Uncertainty in AI*. Morgan Kaufmann Publishers, Inc, July 1990.

[Horswill, 1993] Ian Horswill. Polly: a vision-based artificial agent. In *Proceedings AAAI-93*, 1993.

[Howard and Kitchen, 1996] Andrew Howard and Les Kitchen. Navigation without localisation: a reactive network approach. In *Proceedings of the Fourth International Conference on Control, Automation, Robotics, and Vision*, pages 873–877, 1996.

[Howard and Kitchen, 1997] Andrew Howard and Les Kitchen. Fast visual mapping for mobile robot navigation. In *Proceedings of the IEEE International Conference on Intelligent Processing Systems*, page to appear, 1997.

[Lozano-Perez and Mason, 1984] T Lozano-Perez and M Mason. Automatic synthesis of finre-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[Moravec, 1988] Hans Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, Summer 1988.