# Navigation Using Natural Landmarks

Andrew Howard, Les Kitchen

*Department Of Computer Science, University of Melbourne, Parkville 3052, Australia.*

**Abstract**

This paper describes the navigation system used by J Edgar, a small vision-guided robot that roams the corridors at the University of Melbourne. Given a model of the environment, J Edgar is able to find its way to any given location, even when its initial location is *completely unknown*. To do this, the robot makes use of a probabilistic localisation technique, in which the robot maintains a probability distribution over the space of all possible robot locations. Given this distribution, the robot then applies a particular navigation strategy, whose aim is to ensure that the robot not only reaches the goal, but *knows* that it has reached the goal.

*Key words:* Mobile robots. Localisation. Navigation.

## 1 Introduction

Navigation is a key problem for any mobile robot. In its most conventional form, the navigation problem can be stated as follows: given that a robot is at a known location in a known environment, how does it go about reaching a goal location? In this paper, we address a somewhat different problem: given that a robot is at some *unknown* location in a known environment, how does it go about reaching a goal? Our interest in this problem derives from the fact that, sooner or later, all robots will become lost. Therefore, if robots are to operate unattended for long periods of time, they must have some technique that will enable them to determine that they are lost and to take appropriate steps towards recovery.

Traditional approaches to navigation generally divide the problem into two processes: localisation and planning. These processes can be defined as follows.

- Localisation is the process whereby the robot matches observed landmarks against a model to determine its location. For a robot operating in an indoor environment, typical landmarks might be walls and doorways.

- Planning is the process whereby the robot attempts to find a series of actions that will take it from its current location to the goal location.

In order to solve the more general navigation problem posed above, we have generalised both of these processes.

- Localisation becomes the process whereby the robot matches observed landmarks against a model to construct a *probability distribution* over the space of all possible locations. This generalisation allows the localisation process to handle both uncertainty and ambiguity in the robot's location.
- Planning becomes the process whereby the robot tries to find a series of actions that will not only take the robot to the goal, but will enable the robot to *know* that it has reached the goal. We will show that while it is possible in principle to find an optimal plan that meets these criteria, in practice such plans are very difficult to construct. As an alternative, we therefore present a number of sub-optimal *navigation strategies*, which will enable the robot to reach the goal in a reasonable amount of time, on most occasions, and in most environments.

The navigation system described in this paper has been implemented on J Edgar, a small vision-guided robot that roams the corridors at the University of Melbourne. This robot is capable of finding its way to any given point in the environment, using only naturally occurring landmarks, even when its initial location (and orientation) is completely unknown. In the following sections, we discuss the basic concepts and the mathematical formalism that underpin J Edgar's navigation system. In Section 5, we give an overview of J Edgar itself and describe how the system is implemented in practice. Finally, in Section 6, we present a range of experimental results.

### *1.1 Assumptions*

There are a number of key assumptions made in this paper:

- The robot is able to detect landmarks and measure their properties (in the case of J Edgar, using vision).
- The robot is equipped with odometry or some similar sensor that allows it to make estimates of its motion.
- The robot is *not* equipped with an absolute position sensor, such as differential GPS.
- The robot is *not* equipped with an absolute orientation sensor, such as a compass.
- The environment has not been modified in any way, such as by the addition of artificial beacons.

Collectively, these assumptions make the navigation task quite difficult, since the robot must rely entirely on naturally occurring landmarks, such as walls and doorways, to localise itself. An implicit assumption here is that these landmarks are *ambiguous*; that is, that the robot cannot localise itself simply by observing a single landmark. Rather, the robot must undertake some relatively complex series of actions, and may have to travel a considerable distance, in order to unambiguously determine its location.

We make one further assumption:

- The environment is (mostly) static.

This assumption is made for the sake of simplifying the following discussion rather than out of strict necessity. In reality, J Edgar is quite capable of dealing with some degree of variability in the environment. For example, it is not affected by moving obstacles, such as people, unless they actively set out to block the robot's path. Similarly, J Edgar is not affected by variations in the distribution of landmarks, so long as these variations are reflected in the model (a good example is a doorway, which will be perceived as one type of landmark when open, and another when closed). Nevertheless, the discussion in this paper will focus primarily on static environments.

## 2 The Model

### *2.1 Basic Concepts*

In constructing a model of the environment, we need to consider the processes in which the model will be used : localisation and planning. For localisation, we want a model that describes the environment in terms of the location of significant landmarks, such as walls and doorways. For planning, we want a model that describes the connectivity of the environment; that is, how the robot can move from one part of the environment to another. In addition, the model must capture the inherent uncertainty associated with both sensing and action.

As a concrete example, consider the environment shown in Figure 6. This environment can be described in terms of many different kinds of landmarks, but we will consider just two: walls and doorways. Since there is always some uncertainty associated with the sensing of landmarks (due to noise in the sensors, for example), it is not sufficient to simply specify the location of each wall and doorway in the environment. Instead, we must specify the *distribution* of these landmarks. Figure 6 (middle row), for example, shows the distribution of 'doorway' landmarks. In this image, the dark regions correspond to areas where the robot is most likely to observe a doorway, whilst light regions correspond to areas where the robot is least likely to observe a doorway.

In a similar fashion, Figure 6 (top row) shows the distribution of 'wall' landmarks. In this case, however, the distribution captures another property of the environment — the fact that there is variation associated with wall-type landmarks. Consider the doorways shown in Figure 6(middle row). Each of these doorways may be either open or closed, and since the robot cannot distinguish between a closed doorway and a wall, there is a finite probability that the robot will observe a wall landmark at each of these doorways. This probability is, of course, proportional to the probability that the doorway will be closed and is correspondingly less than the probability of observing a wall landmark where there really is a wall. Figure 6 (top row) indicates this fact through the appropriate shading of these regions.

The *connectivity* of the environment is captured in Figure 6 (bottom row). In this image, the environment is divided up into cells, and it is assumed that a robot in one of these cells can reach an adjacent cell by moving in the appropriate direction. There will, of course, be some uncertainty associated with such an action: a robot heading for one cell may well end up at another. This is particularly true for J Edgar, which relies on a potential field based local navigation system for carrying out basic navigation tasks [13]. If an unexpected obstacle presents itself, the robot will move around it. Consequently, the robot may arrive at a location that is quite different from the one that was anticipated.

### *2.2 Formalism*

The environment model used by J Edgar can be defined formally as follows.

- Let $x$ denote the *pose* (position and orientation) of a landmark in some arbitrary global coordinate system; let $r$ denote the pose of the robot in the same global coordinate system.
- Let $\mathrm{p}(f, x)$ denote the probability of observing a landmark of type $f$ with pose $x$.
- Let $\mathrm{p}(r \mid r', a)$ denote the probability that the robot will reach pose $r$, given that it starts at pose $r'$ and performs action $a$.
- The model is then given by:

$$\text{Model} = \{\mathrm{p}(f, x), \mathrm{p}(r \mid r', a)\} \tag{1}$$

Note that both landmarks and the robot are described in terms of their *pose*, i.e. in terms of their position *and* orientation. Thus, for example, a wall landmark may be described as having either an east-west or north-south orientation. Similarly, the shape of the robot may be such that it can only move through a doorway if it first assumes a particular orientation. In practice, however, the orientation of landmarks provides little additional information (at least for the landmarks used by J Edgar), and since J Edgar is a cylindrical robot, the connectivity of the environment is not affected by the robot's orientation. Nevertheless, we retain this notation for the sake of generality.

### *2.3 Coordinate Systems*

J Edgar makes use of two coordinate systems. The first is the *global coordinate system*, or GCS, in which the model is defined. The second is the odometry-based *local coordinate system*, or LCS, in which measurements are made. Recall that J Edgar is not equipped with any kind of absolute position or orientation sensor (such as a GPS or a compass); therefore, measurements must be made with respect to some coordinate system whose origin is arbitrary. For this, J Edgar uses the odometry-based LCS. Note that, by definition, the robot *always* knows its pose in the LCS.

Clearly, the GCS and LCS must be related by some coordinate transform $\Gamma$ such that:

$$r = \Gamma \hat{r} \tag{2}$$

where $r$ is the robot's pose in the GCS and $\hat{r}$ is the robot's pose in the LCS. The same transform, when applied to the pose $\hat{x}$ of an observed landmark, must give the landmark's pose $x$ in the GCS:

$$x = \Gamma \hat{x}. \tag{3}$$

In a similar fashion, actions in the model are defined with respect to the GCS, but the actions that the robot executes are defined with respect to the LCS. Therefore, given some action $a$ defined with respect to the GCS, the corresponding action in the LCS is determined using the inverse transform $\Gamma^{-1}$:

$$\hat{a} = \Gamma^{-1} a. \tag{4}$$

An explicit form for $\Gamma$ is given in [14].

The difficulty that the robot faces, of course, is that the transform $\Gamma$ will not generally be known a priori. It is therefore the task of the localisation process to determine the correct transform.

# 3 Localisation

## *3.1 Formalism*

In its simplest form, localisation can be defined as the process whereby the robot matches observed landmarks against a model to determine the pose of the robot. As noted in the previous section, however, the pose of the robot in the GCS can be determined only if one first knows the transform $\Gamma$ that describes the relationship between the LCS and the GCS. In addition, we must allow for both uncertainty and ambiguity in the localisation process. We therefore define localisation as the process whereby the robot matches observed landmarks against the model to construct a *probability distribution* $\mathrm{p}(\Gamma)$ over the space of all possible transforms.

The distribution $\mathrm{p}(\Gamma)$ must be updated every time the robot detects a new landmark. Given that the robot has observed some landmark of type $f$ with pose $\hat{x}$ in the LCS, the probability of a transform $\Gamma$ can be computed using Bayes' Law [1]:

$$\mathrm{p}(\Gamma \mid f, \hat{x}) = \frac{\mathrm{p}(f, \hat{x} \mid \Gamma)}{\mathrm{p}(f, \hat{x})} p_0(\Gamma) \tag{5}$$

$$= \frac{\mathrm{p}(f, \Gamma\hat{x})}{\mathrm{p}(f, \hat{x})} p_0(\Gamma) \tag{6}$$

where $\mathrm{p}(f, \Gamma\hat{x})$ is taken directly from the model and $\mathrm{p}(f, \hat{x})$ is a normalisation factor. The term $p_0(\Gamma)$ is known as the *prior* distribution and reflects the robot's prior knowledge. If, for example, the robot's initial pose is completely unknown, all transformations are equally likely and hence $p_0(\Gamma)$ will take some constant value. [1]

Multiple observations can be combined using a generalised form of Bayes' Law:

$$\mathrm{p}(\Gamma \mid f_1, \hat{x}_1 \cdots f_n, \hat{x}_n) = \prod_{i=0}^{n} \frac{\mathrm{p}(f_i, \Gamma\hat{x}_i)}{\mathrm{p}(f_i, \hat{x}_i)} p_0(\Gamma), \tag{7}$$

where the assumption is made that observations are statistically independent [1]. Note that while this assumption greatly simplifies computations, it is not

[1] Strictly speaking, this may not always be true. Depending on the form of $\Gamma$, a uniform distribution over the space of all possible transforms may not lead to a uniform distribution over the space of all possible poses. Some care must therefore be exercised.

entirely unproblematic. If a sensor were to generate multiple identical measurements of the same landmark, this would tend to skew the distribution. In effect, too much weight would be given to these measurements and not enough to others. Therefore, care must be taken to ensure that the information generated by the robot's sensors conforms to the statistical independence assumption.

So far in this discussion, we have not considered the issue of odometric drift. The LCS is based on odometric measurements which are notoriously prone to cumulative drift. As a consequence, the relationship between the LCS and GCS will change slowly over time. Generally, this change will depend on the change in robot's pose. Given that the robot's pose has changed by some amount $\Delta\hat{r}$ (as measured in the LCS), we can write down an expression for the change in the distribution $\mathrm{p}(\Gamma)$:

$$\mathrm{p}(\Gamma \mid \Delta\hat{r}) = \sum_{\epsilon} \mathrm{p}(\epsilon \mid \Delta\hat{r})\mathrm{p}(\Gamma - \epsilon) \tag{8}$$

where $\mathrm{p}(\epsilon \mid \Delta\hat{r})$ is a *drift term* that describes the probability that the transform will change by some amount $\epsilon$, given that the robot's pose in the LCS has changed by some $\Delta\hat{r}$. We assume that this term arises from stochastic effects and is not dependent on the robot's pose.

Sometimes, we will be interested in knowing the probability distribution $\mathrm{p}(r)$, which gives the probability that the robot has pose $r$ in the GCS. This distribution can be generated from $\mathrm{p}(\Gamma)$ using the relationship:

$$\mathrm{p}(r) = \sum_{\Gamma \,:\, r=\Gamma\hat{r}} \mathrm{p}(\Gamma) \tag{9}$$

where $\hat{r}$ is the pose of the robot in the LCS and the summation is over all transforms that imply that the robot is at $r$.

### *3.2 Implementation*

In order to implement this approach to localisation, we must first discretize the problem in some way. The approach we have taken employs a genetic algorithm (of sorts) [8,9] that is extremely efficient in terms of both memory and computation. In this approach, the robot maintains a *population* of candidate transforms, with each member of the population representing one possible relationship between the LCS and the GCS. The size of the population is both finite and fixed, and the initial distribution of candidate transforms is chosen to match the robot's knowledge about its initial pose. Thus, for example, if the robot's initial pose is completely unknown, the population consists of candidate transforms selected at random from the space of all possible transforms. The size of the population required to achieve robust localisation will depend on the size of the environment, but is typically of the order of a few thousand transforms.

The population is subject to three processes — evaluation, mutation and reproduction — which are defined as follows.

**Evaluation.** Each candidate transform $\Gamma$ has a probability $\mathrm{p}(\Gamma)$ associated with it. When landmarks are detected, this probability is updated using an incremental form of Equation 6:

$$\mathrm{p}(\Gamma) \longleftarrow \frac{\mathrm{p}(f, \Gamma\hat{x})}{\mathrm{p}(f, \hat{x})}\mathrm{p}(\Gamma) \tag{10}$$

**Mutation.** The population undergoes a regular mutation process, which is used to model odometric drift. In this process, all transforms undergo a small random mutation. That is, each transform $\Gamma$ is subject to a mutation $\epsilon$ such that:

$$\Gamma \longleftarrow \Gamma + \epsilon. \tag{11}$$

An explicit form for $\epsilon$ is given in [14], together with a discussion that outlines how other parameters, such as population size, may be chosen.

**Reproduction.** The population undergoes a regular reproduction process. In this process, transforms with a high probability are 'cloned' to produce two new transforms. The new transforms are identical to the parent in every respect save for their probability: the probability of the parent is divided equally between the two new transforms. A parent transform $\Gamma_P$ is cloned to produce child transforms $\Gamma_1$ and $\Gamma_2$ as follows:

$$\begin{aligned} \Gamma_1 &\longleftarrow \Gamma_P \\ \Gamma_2 &\longleftarrow \Gamma_P \\ \mathrm{p}(\Gamma_1) &\longleftarrow \tfrac{1}{2}\mathrm{p}(\Gamma_P) \\ \mathrm{p}(\Gamma_2) &\longleftarrow \tfrac{1}{2}\mathrm{p}(\Gamma_P) \end{aligned} \tag{12}$$

Since the population size is fixed, one of the child transforms takes the place of the parent, while the other takes the place of some other transform whose probability is low. The net effect of the reproduction process is that, over time, bad transforms get tend to get 'weeded out' and replaced with the progeny of more likely transforms. The population will also cluster around the more probable regions of transform space.

Note that there are many, many variations to this implementation that could be explored. However, this particular implementation is simple, computationally inexpensive, and works well in practice.

As an illustration, consider the first series of images shown in Figure 3. These images show a robot moving through a simulated environment totally lacking in landmarks. The transform population is represented by a cloud of dots, with each dot representing one estimate of the robot's pose. The population has been initialised so that all members are identical. Over time, however, cumulative mutations cause the population to spread out, in a manner which quite accurately models the odometric drift associated with twin drive-wheel robots, such as J Edgar.

As another illustration, consider the second series of images shown in Figure 3. These images show the robot at various times as it moves through a simulated environment containing a variety of landmarks. It is readily apparent that the detection of landmarks quickly reduces an initial, evenly distributed population to a few distinct clusters. Eventually, most of these clusters are eliminated until the only remaining cluster is the one that corresponds to the correct transform.

## 4 Planning and Strategies

### *4.1 Basic Concepts*

In simplest form, the planning problem can be stated as follows: given that the robot's pose in the GCS is known, find a series of actions that will take the robot to the goal. The problem becomes more complex when the robot's pose is uncertain or ambiguous, as is usually the case. We therefore re-state the planning problem as follows: given that the *probability* that the robot is at any particular pose is known, find a series of actions that will both take the robot to the goal and allow the robot to know that it has reached the goal.

It is possible, in principle, to find an optimal solution for this problem. The robot's *state* can be described in terms of the distribution $\mathrm{p}(r)$, which describes the probability that the robot has pose $r$ in the GCS. The *space* of such states can then be searched to find a series of actions that transforms the robot's initial state (in which the pose of the robot may be completely unknown), to the goal state (in which the distribution is peaked around the location corresponding to the goal). In reality, of course, it is extremely difficult to perform such a search. Since each point in the search space corresponds to a possible distribution $\mathrm{p}(r)$, the search space is extremely large. Hence, this approach seems impractical for realistic environments.

As an alternative, we have developed a number of sub-optimal *navigation strategies*. The intent of these strategies is that the robot should be able to reach the goal in a reasonable amount of time, on most occasions, and in most environments (examples of pathological environments are given in Section 4.6). These strategies are all fundamentally alike in that they are composed of a *plan*, a *selection rule* and a *termination rule*.

A *plan* is a function that specifies, for each pose in the GCS, the action the robot should perform when it has that pose. We write this as:

$$a = \mathrm{Plan}(r). \tag{13}$$

We can make use of this plan, even when the robot's pose is uncertain, by augmenting it with a *selection rule*. In its simplest form, the selection rule picks out one possible transform $\Gamma_s$ from the space of all possible transforms, and directs the robot to carry out the corresponding action in the plan. That is,

$$\hat{a} = \Gamma_s^{-1}\mathrm{Plan}(\Gamma_s \hat{r}). \tag{14}$$

where $\hat{r}$ is the robot's pose in the LCS. The inverse transform is applied to convert the planned action (which is specified with respect the GCS) into an action that is specified with resepect to the LCS. The robot applies this procedure until the *termination rule* is satisfied, at which point it will conclude that it has reached the goal, and stop.

We will consider three different strategies in this paper: the *most probable strategy*, the *proportional voting strategy* and the *persistent strategy*. The most probable and proportional voting strategies have been chosen because of their simplicity, and because they are analogous to similar techniques described in previous work on probabilistic navigation [20,17]. The persistent strategy is novel, and is designed to address some of the observed limitations of the first two strategies. In the following sections, we describe each of these strategies in detail, while in Section 6, we compare the performance of these strategies experimentally.

The plan used by all three strategies is constructed by considering the *cost* of reaching the goal from any given pose. Usually, cost is a measure of time required to reach goal or the distance that must be travelled. We make the following definitions.

- Let the *plan function* $\text{Plan}(r)$ describe the planned action for pose $r$.
- Let the *cost function* $\text{Cost}(r)$ describe the cost of reaching the goal from pose $r$.
- Let $\text{Cost}(a)$ denote the cost of action $a$. We assume that this term is either fixed or can be calculated from the model.

If we assume that the robot always performs the action with the lowest cost, we can immediately write down an expression for the cost function: [2]

$$\text{Cost}(r) = \min_a \sum_{r'} \text{p}(r' \mid r, a)\text{Cost}(r') + \text{Cost}(a), \tag{15}$$

from which it follows that the plan function must be given by:

$$\text{Plan}(r) = \arg\min_a \sum_{r'} \text{p}(r' \mid r, a)\text{Cost}(r') + \text{Cost}(a). \tag{16}$$

The term $\text{p}(r' \mid r, a)$ denotes the probability that robot will reach pose $r'$, given that it has pose $r$ and performs action $a$; this term is taken directly from the model.

Equation 15 (which can be though of as a set of simultaneous linear equations) is solved numerically using a simple value iteration scheme. The procedure is as follows.

(1) Let $\text{Cost}_i(r)$ denote the value of the cost function on the $i^{th}$ iteration.
(2) Set $\text{Cost}_0(r) = \infty$ for all $r$.
(3) Compute the value of $\text{Cost}_{i+1}(r)$ from $\text{Cost}_i(r)$ using a variant of Equation 15.

$$\text{Cost}_{i+1}(r) = \min_a \sum_{r'} \text{p}(r' \mid r, a)\text{Cost}_i(r') + \text{Cost}(a) \tag{17}$$

(4) Repeat step 3 until the total difference between cost functions on successive iterations falls below some arbitrary cutoff value.

[2] The reader may recognise this as a result from Markov Decision Process Theory [7]. It has been pointed out by previous authors [17] that mobile robot navigation can be treated as a partially observable Markov decision process (POMDP).

The value of $\text{Cost}_i(r)$ will converge to the true value in the limit $i \to \infty$.

### *4.3 Most Probable Strategy*

The most probable strategy is extremely simple: it directs the robot to perform the action corresponding to the most probable transform. The robot continues to perform this action until the most probable transform indicates that the robot is at the goal. The strategy is defined formally as follows:

- Selection rule: Select some $\Gamma_s$ such that:

  $$\Gamma_s = \arg\max_{\Gamma} \mathrm{p}(\Gamma). \tag{18}$$

  and perform the action given by

  $$\hat{a} = \Gamma_s^{-1}\mathrm{Plan}(\Gamma_s \hat{r}). \tag{19}$$

- Termination rule: Terminate when $\Gamma_s \hat{r} = \text{goal}$.

### *4.4 Proportional Voting Strategy*

The proportional voting strategy is somewhat more democratic than the most probable strategy. Each transform is allowed to 'vote' for an action in proportion to the probability that that transform is correct. This procedure continues until the action with the most votes is 'stop'. The proportional voting strategy is defined formally as follows:

- Selection rule: Let the number of votes for an action $\hat{a}$ be defined as follows:

  $$\mathrm{Votes}(\hat{a}) = \sum_{\Gamma\,:\,\hat{a}=\Gamma^{-1}\mathrm{Plan}(\Gamma\hat{r})} \mathrm{p}(\Gamma). \tag{20}$$

  The robot performs the action with highest number of votes, that is, the action given by

  $$\hat{a} = \arg\max_{\hat{a}'} \mathrm{Votes}(\hat{a}'). \tag{21}$$

- Termination rule: terminate when $\hat{a} = \text{stop}$.

The rationale behind this selection rule is that it allows all transformations to have a say in what the robot should do, but gives more weight to those transforms with a high probability.

The persistent strategy is designed to address two problems which are observed experimentally with the first two strategies. Firstly, both strategies are prone to oscillation. This occurs when the robot rapidly switches back and forth between two actions, as subtle changes in the shape of the distribution $p(\Gamma)$ result in different actions being selected. While the robot will generally extract itself from such oscillations eventually, it can waste a great deal of time in doing so.

The second problem with the first two strategies is that there is a high probability that the robot reach at a place that looks like, but is not, the goal. Consider, for example, the most probable strategy. This will terminate as soon as the most probable transform indicates that the robot is that the goal. However, the probability that this transform is true, and hence the probability that the robot is *actually* at the goal, could be anywhere between zero and one.

The persistent strategy addresses the problem of oscillation by adding the notion of *persistence* to the selection rule. Once a transform is selected, the robot persists with this transform until some condition is met. The second problem, that of reaching false goals, is addressed in a manner which is at first sight somewhat paradoxical. In selecting and persisting with a transform, the robot is effectively *testing* that transform. There can only be two outcomes of such a test: either the robot will reach the goal (according to the selected transform), or the transform will prove to be false. The robot can maximise its chances of falsifying the selected transform, and of consequently reducing the ambiguity in the distribution $p(\Gamma)$, by selecting whichever transform indicates that the robot is *most distant* from the goal. Hence the persistent strategy should perhaps be called the persistent pessimist strategy. As we will show in Section 6, the persistent strategy does indeed lead to a much higher probability that the robot will reach the goal.

The persistent strategy is defined formally as follows:

- Selection rule: Select some transform $\Gamma_s$ such that

$$\Gamma_s = \arg\max_{\Gamma} \text{Dist}(\Gamma\hat{r}) \quad \text{and} \quad p(\Gamma_s) > c. \tag{22}$$

  where $\text{Dist}(r)$ is the distance the robot must travel to reach the goal and $c$ is some cutoff probability. Perform the action given by

$$\hat{a} = \Gamma_s^{-1}\text{Plan}(\Gamma_s\hat{r}). \tag{23}$$

  The robot persists with the selected transform for as long as it satisfies the condition:

$$\Gamma\hat{r} \neq \text{Goal} \quad \text{and} \quad \mathrm{p}(\Gamma_s) > c \tag{24}$$

- Termination rule: stop when there is no transform that simultaneously satisfies Equations 22 and 24.

### 4.6 Pathological Environments

Consider the simple environment shown in Figure 4(a); this is an example of a pathological environment. Clearly, since the environment is rotationally symmetric, there is no method by which a robot whose pose is initially unknown can unambiguously determine its location. Hence, there is no strategy, of any kind, by which the robot can both reach the goal and *know* that it has reached the goal. The three navigation strategies respond to this environment in different ways.

- The most probable strategy will quickly terminate, but the probability that the robot is at the goal will be no more than 50%.
- The proportional voting strategy will also terminate; the probability that the robot is at the goal will be no more than 50%.
- The persistent strategy will never terminate, since there will always be some transform that needs to be tested. The robot will shuttle back-and-forth between the two locations that could be the goal.

This is, of course, an extreme example — real environments are not usually constructed in such a fashion (with the possible exception of major international airports).

A similar effect can occur, however, if the environment contains rotationally symmetric *regions* and if the goal is contained in one of these regions. Consider, for example, the section of corridor depicted in Figure 4(b). For most configurations of the transform population, this environment does not pose a problem. However, for the particular configuration shown in Figure 4(b), this environment is pathological — the robot will either arrive at the wrong goal (with probability 50%) or will shuttle back and forth between the two doorways. This example, which can occur quite frequently in real environments, highlights the need for navigation strategies that are more sophisticated than those described here. Such strategies are the subject of ongoing research.

## 5 Implementation — Robot J Edgar

Robot J Edgar is shown in Figure 1. It is a simple robot, equipped only with a pair of drive wheels, a single board computer, a frame grabber, and a monochrome camera. The robot also has a spread-spectrum UHF data-link, which allows it to communicate with a base-station. This data-link is particularly useful, since it allows us to shift much of the robot's control software off the robot and onto the base station. This greatly reduces software development cost and time.

J Edgar employs a highly modular agent-oriented control system, inspired in part by the *subsumption architecture* approach [2]. 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. Figure 1 shows the various agents that make up J Edgar. These 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 also provides an abstract interface to higher layers, so that, in principle, they know nothing of the underlying hardware. Alternatively, the hardware layer can play the role of simulator. For J Edgar, we have developed a simulator that is quite detailed and very transparent: there is no way in which higher layers can distinguish between the simulated robot and the real robot.

**The local layer** has two basic responsibilities: processing visual data to build local occupancy maps (see, for example, Figure 2) and performing local navigation tasks (i.e. obstacle avoidance). A more detailed description of the techniques used in this layer can be found in [6,11,13].

**The global layer** is responsible for tasks involving the global environment. It is made up of three agents, which operate as follows.

- The feature-extraction agent (GbFeature) detects significant landmarks by analysing the local occupancy map. This agent has a set of pre-defined templates, corresponding to features such as walls and doorways, which it tries to match to structures in the local map (a more detailed description of this process can be found in [12]). When a landmark is found, information about its type and pose (in the LCS) is passed on to the localisation agent agent.
- The localisation agent (GbLoc) maintains a population of candidate transforms. The population is updated on a regular basis using information about the robot's motion (from the hardware layer) and information about de-

tected landmarks.
- The navigation agent (GbNav) selects an action using one of strategies described in Section 4. This action is passed back down to local layer for execution.

Note that J Edgar effectively divides the overall navigation task into two subtasks: 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 layer; and local navigation can be treated as a reactive problem, in which no long-term planning is required. Empirically, we have found that this division of responsibilities leads to very robust behaviour. In the course of collecting the experimental data described Section 6, the robot had to deal with the presence of a large number of moving obstacles (people) in its environment. Whilst this sometimes caused confusion for the global layer — in one instance, the robot decided that three people standing together must be a wall — there was no occasion on which the robot actually collided with an obstacle, moving or otherwise.

## 6 Experiments

In this section, we evaluate the overall performance of the navigation system in a range of environments, both simulated and real. In addition, we compare the individual performance of the three navigation strategies described in Section 4. Note that there are many variables whose effects are not explored in the results presented here. These include:

- The use of different parameters for the localisation system (such as population size and odometric drift rate).
- The use of different kinds of landmarks.
- The use of partial or inaccurate models.

These are all interesting topics, but they are beyond the scope of this paper.

### *6.1 Protocol*

The overall performance of the system is evaluated by conducting repeated trials. In each trial, the robot is placed at some initial pose (which may or may not be known to the robot) and is allowed to proceed to the goal (or perhaps to a place that it thinks is the goal). We define two statistical performance measures:

- The total failure rate, which is defined as the fraction of trials in which the robot fails to reach the goal.
- The mean time per trial, which is defined as the average time taken to complete each trial.

Clearly, the navigation system should aim to minimise both of these measures.

In addition to measuring the performance of the system as a whole, we are also interested in comparing the individual performances of the three different navigation strategies. To do this, we need to distinguish between effects that can be attributed to the localisation system and effects that must be attributed to the navigation strategy alone. That is, in those cases in which the robot fails to reach the goal, we would like to know whether this failure was due to a failure of the localisation system, or a failure of the navigation strategy. The tool for making this distinction is called a *confusion matrix* and takes the following form.

| | $\bar{g}$ | $\neg\bar{g}$ |
|---|---|---|
| $g$ | $\mathrm{p}(g \wedge \bar{g})$ | $\mathrm{p}(g \wedge \neg\bar{g})$ |
| $\neg g$ | $\mathrm{p}(\neg g \wedge \bar{g})$ | $\mathrm{p}(\neg g \wedge \neg\bar{g})$ |

In this matrix, the proposition $g$ is true if the robot reaches the goal. Similarly, the proposition $\bar{g}$ is true if the robot *believes* that it has reached the goal, i.e. if the probability p(goal) computed using Equation 9 exceeds 0.8 (finding a good definition for what the robot *believes* can be tricky, but this definition seems as good as any). The first row of the confusion matrix gives two values:

- $\mathrm{p}(g \wedge \bar{g})$ is the fraction of trials in which the robot reached the goal and believed that it had reached the goal.
- $\mathrm{p}(g \wedge \neg\bar{g})$ is the fraction of trials in which the robot reached the goal but did not believe that it had reached the goal.

The second row of the confusion matrix gives the complementary values:

- $\mathrm{p}(\neg g \wedge \bar{g})$ is the fraction of trials in which the robot did not reach the goal but believed that it had reached the goal.
- $\mathrm{p}(\neg g \wedge \neg\bar{g})$ is the fraction of trials in which the robot did not reach the goal and did not believe that it had reached the goal.

Clearly, if the robot's localisation system is perfect, the off-diagonal elements of the confusion matrix — which indicate a disagreement between where the robot actually is and where the robot thinks it is — should be zero. Similarly, if

Table 1
Results for the simulated environment.

| | Most Probable | Prop. Voting | Persistent |
|---|---|---|---|
| Total trials | 234 | 238 | 244 |
| Failures | 58 | 33 | 13 |
| Mean time per trial (sec) | 63 [±23] | 69 [±21] | 100 [±49] |
| Total failure rate (%) | 25 [20, 32] | 14 [10, 20] | 5 [3, 10] |
| Localisation failure rate (%) | 3 | 3 | 3 |
| Strategy failure rate (%) | 22 | 11 | 2 |

the navigation strategy is perfect, the on-diagonal element $\mathrm{p}(\neg g \wedge \neg \bar{g})$ — which can be interpreted as the fraction of trials in which the robot failed to reach the goal, even though the localisation system worked flawlessly — should be zero. We therefore define the following additional pair of performance measures:

- The localisation failure rate, given by $\mathrm{p}(\neg g \wedge \bar{g})$.
- The strategy failure rate, given by $\mathrm{p}(\neg g \wedge \neg \bar{g})$.

Note that we count as failures only those trials in which the robot does not arrive at the goal. Trials in which the robot arrives at the goal, but does not believe that it has arrived at the goal, are not counted as failures. Such outcomes are quite rare, of course.

### *6.2 A Simulated Environment*

matrix

As an initial experiment, we conducted a large number of trials using the simulated environment shown in Figure 5. In each trial, the robot's initial pose was chosen at random, and this pose was *unknown* to the robot. The results for this experiment are given in Table 1, which lists both the mean time per trial and the overall failure rate. For both measures, we have calculated a 95% confidence interval (shown in square brackets). Note that the relatively large interval on the mean time per trial mostly reflects the variation in the robot's initial pose — trials will take less time when the robot starts near the goal.

Inspecting this table, it is apparent that a significant fraction of the failures can be attributed to the localisation system. There are a number of possible causes for such a failure:

- The model may be inaccurate.
- The landmark detection agent may produce a spurious match.

- The population mutation rate may not accurately model odometric drift.
- The population size may not be large enough to capture the correct initial transform.

Given that this is a simulation, the first three causes can presumably be discounted, which leads to the conclusion that localisation failures are most probably due to an inadequate population size.

The remainder of the failures in Table 1 must be attributed to failures of the navigation strategy. Comparing the three strategies, it is immediately apparent that the persistent strategy has a much lower failure rate than the other two. This difference can be further highlighted by breaking down the results to reveal the effect of initial location on the strategy failure rate. Consider Figure 5, which shows the failure rate for a number of initial locations superimposed on a map of the environment. From this figure, it is quite clear that while the average failure rates for the most probable and proportional voting strategies are 22% and 11% respectively, the failure rates for some initial locations are much higher. These failures are a direct result of the termination rules for both these strategies, which makes them prone to stopping at places that look like, but are not, the goal. As noted in Section 4, the most probable strategy will terminate as soon as the most probable location is the goal location, regardless of how probable this is. Similarly, the proportional voting strategy will terminate as soon as a weighted majority of transforms indicates that the robot is at the goal, i.e. when there is only a 50% chance that the robot is actually at the goal.

Inspecting Figure 5, it is apparent that the failure rate for the persistent strategy is relatively constant for all initial locations. This is to be expected, since this strategy does not suffer from the problem of reaching false goals. The question might then be asked — under what circumstances does the persistent strategy fail? There are two possible causes for failures of the persistent strategy:

- The robot may become 'stuck' behind some obstacle. That is, the robot may attempt to perform an action that would require it to move through an extended obstacle, such as a wall. Since the local navigation layer will refuse to carry out such actions, the robot will stop, and the navigation strategy will be deemed to have failed.
- The environment may contain regions that have either rotational or translational symmetries. In this case, it is possible for the population of transforms to assume a pathological configuration which traps the robot within the symmetric region of the environment. This scenario was dealt with in Section 4.6.

All of the failures seen in this experiment were a result of the robot becoming

Table 2
Results for the real environment.

| | Most Probable | Prop. Voting | Persistent |
|---|---|---|---|
| Total trials | 12 | 12 | 12 |
| Failures | 4 | 4 | 1 |
| Mean time per trial (sec) | 86 [±27] | 123 [±60] | 117 [±52] |
| Total failure rate (%) | 33 [10, 72] | 33 [10, 72] | 8 [0, 49] |
| Localisation failure rate (%) | 8 | 17 | 8 |
| Strategy failure rate (%) | 25 | 17 | 0 |

stuck behind some obstacle.

Looking now at the mean time per trial, it can be seen that the persistent strategy takes somewhat longer on average than the other strategies. While it is tempting to attribute this difference to the pessimistic nature of the persistent strategy — it will always select transforms that indicate that the robot is far from the goal — a closer analysis of the data reveals that this is not the case. Consider the strategy failure rates shown in Figure 5. For the first two strategies, the failure rate is highest for those locations that are most distant from the goal. That is, for these locations, there is a high probability that the robot will find a nearby false goal. As a result, the mean time for the first two strategies is lower than would be expected if the robot was reaching the goal more often. Thus, the higher mean time exhibited by the persistent strategy can be attributed to the fact that with this strategy, the robot is reaching the goal more often.

### *6.3 A Real Environment*

Consider Figure 6, which shows a fragment of the environment that J Edgar normally inhabits. We have chosen to model this environment in terms of just two kinds of landmarks: walls and doorways.

Figure 6 (top row) shows the distribution of walls in this environment. Dark regions corresponding to locations where the robot is likely to find a wall and light regions to locations where the robot is unlikely to find a wall. It also shows the the template used by the feature extraction agent to identify walls in the local occupancy map. Note that since J Edgar cannot tell the difference between a closed doorway and a wall, there is finite probability that the robot will observe a wall where there is in fact a doorway. Figure 6(top row) indicates this fact through the shading of the appropriate regions. In a similar fashion, Figure 6 (middle row) shows the distribution of doorway landmarks, together with the doorway template.

The *connectivity* of the environment is captured in Figure 6(bottom row). In this image, the environment is divided up into cells, and it is assumed that a robot in one of these cells can reach an adjacent cell by moving in the appropriate direction.

Consider now Figure 7, which shows the results for a typical trial using the persistent strategy. In this trial, the robot's initial pose was chosen at random and this pose was not known to the robot. The left column shows the local occupancy map at a number of different points in time. The right column shows the corresponding transform population, with each dot representing one estimate of the robot's pose. Looking at this sequence of images, it is clear that the detection of a number of wall landmarks quickly reduces the initial, uniformly distributed population to a population containing three distinct clusters. Further reduction, however, takes much longer. The robot has to travel about 10m before it can reduce the three clusters to just one cluster. At this point, the robot moves directly to the goal.

The results for 36 such trials are summarised in Table 2. In each trial, the robot's initial pose was chosen at random and this pose was *unknown* to the robot. The goal in all cases was the same. Some care must be taken when interpreting these results, in view of the relatively small number of trials (performing trials with a real robot, as opposed to a simulator, is quite time-consuming). Nevertheless, the performance of the persistent strategy appears to be quite good — the robot experienced only 1 failure in 12 trials, and that failure was due to the failure of the localisation system. Contrast this with the performance of the most probable and proportional voting strategies, which each failed on 4 occasions. These results are in line with those obtained using the simulator.

Interestingly, the observed localisation failures where ultimately found to be a consequence of inaccuracies in the model. The model was constructed manually from the architect's original floor plans. Unfortunately, it was later discovered that the builder had taken a somewhat liberal interpretation of these plans, with some walls being misplaced by over a meter! This certainly highlights the need for robots that can construct their own models, a topic we hope to address in the near future.

## 7 Related Work

An algorithm for explicitly re-localising a robot is described by Dudek et al. [5]. This is a theoretical analysis that describes how a robot at some unknown location in a polygonal environment can determine its location with the least possible effort. That is, it describes an algorithm for obtaining the shortest path the robot can take to unambiguously determine its location. While this work is attractive because of the theoretical guarantees it places on the robot's performance, it is unclear how the technique can be applied real robots, where uncertainties associated with both sensing and action must be considered.

Takeda et al. describe an approach to planning that takes account of the sensory uncertainty associated with different parts of the environment [21]. That is, they describe an algorithm for finding paths that minimise localisation errors. In some ways, this work complements that of Dudek et al.: if one can first localise the robot, this algorithm should prevent it from getting lost again. This is not the approach we have taken, however; we have tried to find a single algorithm that unifies both of these processes.

There are many, many examples of mobile robots that can carry out navigation tasks in indoor environments. See, for example [4,10,15,16,18,19]. However, a key difference between these robots and J Edgar is in the assumptions that are made about the robot's initial pose. Either the pose is assumed to be known, or else it is assumed that it can be quickly determined by identifying a small number of landmarks. In effect, while these robots may allow for uncertainty in the robot's pose, they do not allow for *ambiguity*.

A notable exception to the above is the work of Thrun et al. [22,3] in their museum tour-guide robots (and to a lesser extent, of Koenig [17,20]). These robots employ a Bayesian localisation technique, in which the robot maintains a probability distribution over a grid representing all possible robot locations. The robots are equipped with a combination of sensors — including vision, sonar and laser range finders — whose measurements are matched against a model that is either pre-programmed or can be acquired through a relatively painless training procedure. There are two of key differences between these museum-guide robots and J Edgar. Firstly, J Edgar relies on the notion of *landmarks* for localisation. That is, rather than attempting to model the entire environment, J Edgar only models that part of the environment that is relevant to the localisation task. Secondly, while the Bayesian localisation technique used by both systems is conceptually similar, we believe that the algorithm implemented on J Edgar is more efficient than that described in [3].

## 8 Conclusion

The experiments that we have conducted indicate that J Edgar's navigation system is very robust. This is highlighted by the fact that localisation system failed in only 3 out of 36 trials in the real environment, despite the fact that model was later found to be quite inaccurate. Similarly, the persistent strategy appears to be a very effective solution to the problem of navigation when the robot's initial pose is unknown.

There are many extensions to this system that are the subject of ongoing research. We will mention just three here. Firstly, we would like to extend the system to cope with *changing environments*. In its current form, the system can successfully localise the robot in a changing environment, but cannot always 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 we would like the system to be able to acquire models in an autonomous fashion. Finally, we are looking at some aspects of *cooperative* navigation, in which a team of robots can carry out navigation tasks that are difficult or impossible for a single robot.

## References


[1] James O. Berger. *Statistical Decision Theory and Bayesian Analysis, Second Edition.* Springer-Verlag, 1985.

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

[3] W. Burgard, A. B. Cremers, D. Fox, D. Haehnel, G. Lakemeyer, D. Shultz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. Technical Report CMU-CS-98-139, Carnegie Mellon University, 1998.

[4] Ingemar J. Cox. Blanch — an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, April 1991.

[5] 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.

[6] 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.

[7] Richard M. Feldman. *Applied probability and stochastic processes.* Boston : PWS Pub. Company, 1996.

[8] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[9] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* MIT Press, 1992. Original edition published 1975.

[10] Ian Horswill. Polly: a vision-based artificial agent. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 824–829, 1993.

[11] Andrew Howard and Les Kitchen. Fast visual mapping for mobile robot navigation. In *Proceedings of the IEEE International Conference on Intelligent Processing Systems*, pages 1251–1255, 1997.

[12] Andrew Howard and Les Kitchen. Vision-based navigation using natural landmarks. In *International Conference on Field and Service Robotics*, pages 348–355, 1997.

[13] Andrew Howard and Les Kitchen. Vision-based obstacle avoidance. Technical Report CS-TR97/35, University of Melbourne, 1997.

[14] Andrew Howard and Les Kitchen. Solving the drop-off problem: how to get to your goal when you don't know where you are. Technical Report CS-TR98/13, University of Melbourne, 1998.

[15] R A Jarvis and A Lipton. Go-2:- an autonomous mobile robot for a science museum. In *Proceedings of the Fourth International Conference on Control, Automation, Robotics, and Vision*, pages 260–266, 1996.

[16] C M Johnson and J O Gray. Hierarchical path-planning for a mobile robot. *Engineering Applications of Artificial Intelligence*, 7(2):137–149, 1994.

[17] S. Koenig and R.G. Simmons. A robot navigation architecture based on partially observable markov decision process models. In D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, chapter 4, pages 91–122. MIT Press, 1997.

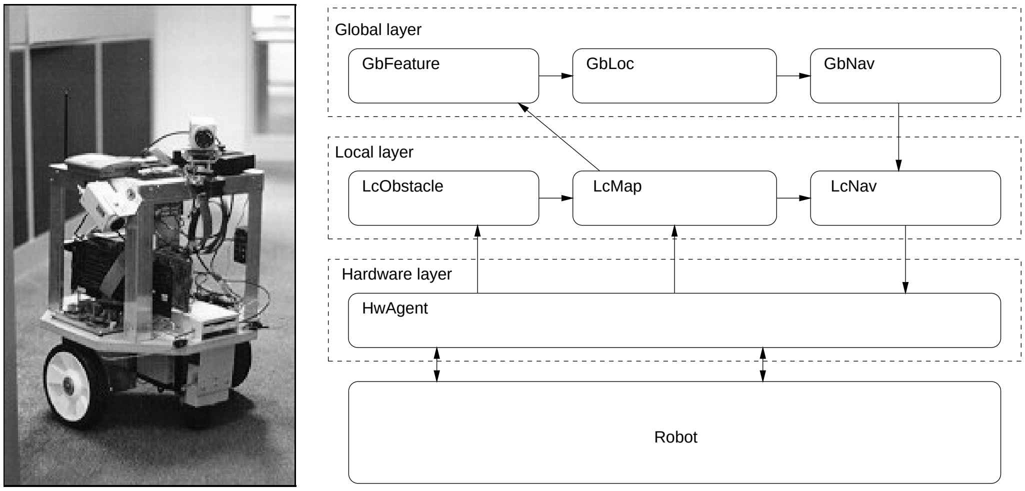
[18] David Kortenktamp, R. Peter Bonasso, and Robin Murphy, editors. *Artificial Intelligence and Mobile Robots : Case Studies of Successful Robot Systems.* MIT Press, 1998.

[19] Yoshio Matsumoto, Masayuki Inaba, and Hirochika Inoue. Memory-based navigation using omni-view sequence. In *Proceedings of the International Conference on Field and Service Robotics*, pages 184–191, 1997.

[20] Reid Simmons and Sven Koenig. Probabilistic navigation in partially observable environments. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1995.

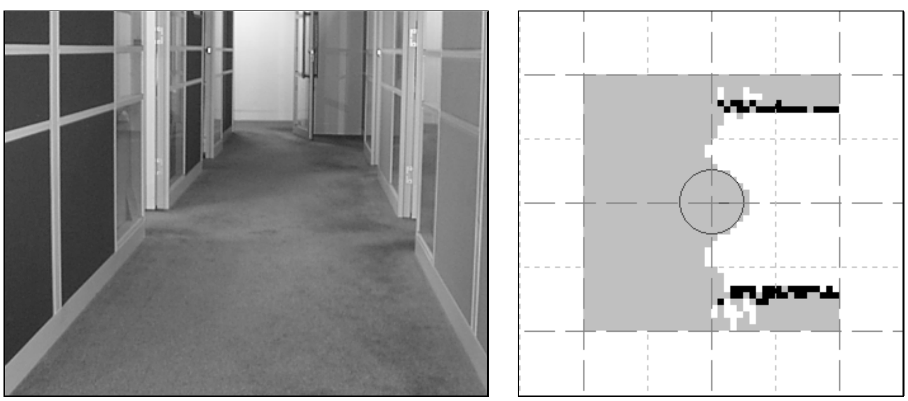
[21] Haruo Takeda, Claudio Facchinetti, and Jean-Claude Latombe. Planning the motions of a mobile robot in a sensory uncertainty field. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(10), 1994.

[22] S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localisation for mobile robots. *Machine Learning*, 31:29 – 55, 1998.




(a) (b)

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



(a) (b)

Fig. 2. (a) Typical image taken by J Edgar's camera. The robot is looking down a corridor. (b) Occupancy map produced by the local mapping system. Black pixels indicate occupied regions, white pixels indicate unoccupied regions and grey pixels indicate unknown regions.

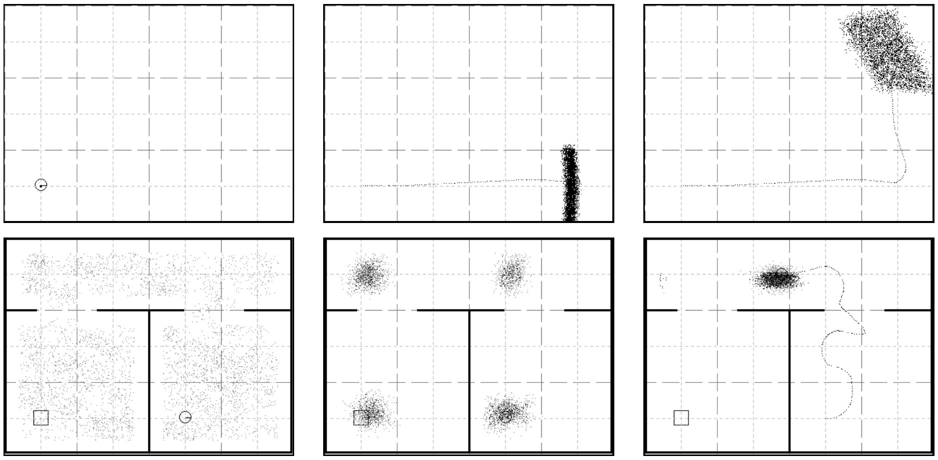
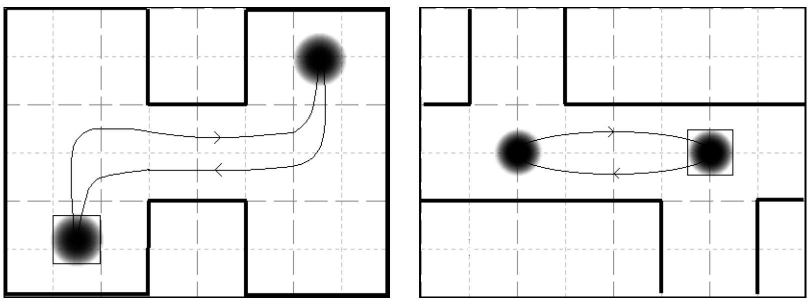
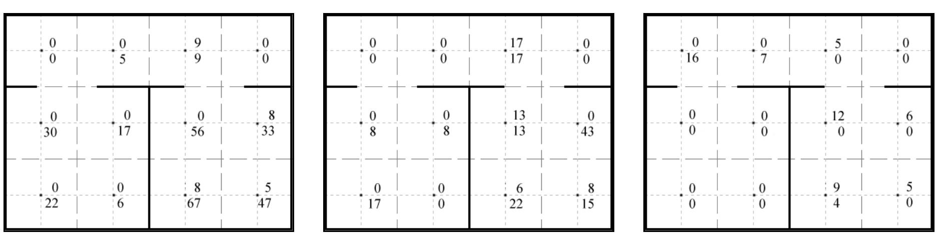


Fig. 3. Top row: Evolution of the transform population as the robot moves through a featureless simulated environment. Each dot represents one estimate of the robot's pose; the dotted line represents the robot's true path. Bottom row: Evolution of the transform population in an environment containing wall and corner landmarks.



(a) (b)

Fig. 4. (a) Pathological environment. Under the persistent strategy, the robot will shuttle back-and-forth on the indicated path. (b) Pathological situation. While the environment as a whole is not pathological, the particular population distribution shown in the figure is.




(a) (b) (c)

Fig. 5. (a) Results for the TwoRoom simulation with unknown pose using the 'most probable' navigation strategy. The map shows the localisation and strategy failure rates (expressed as percentages) for a range of initial locations. (b) Proportional voting strategy. (c) Persistent strategy.

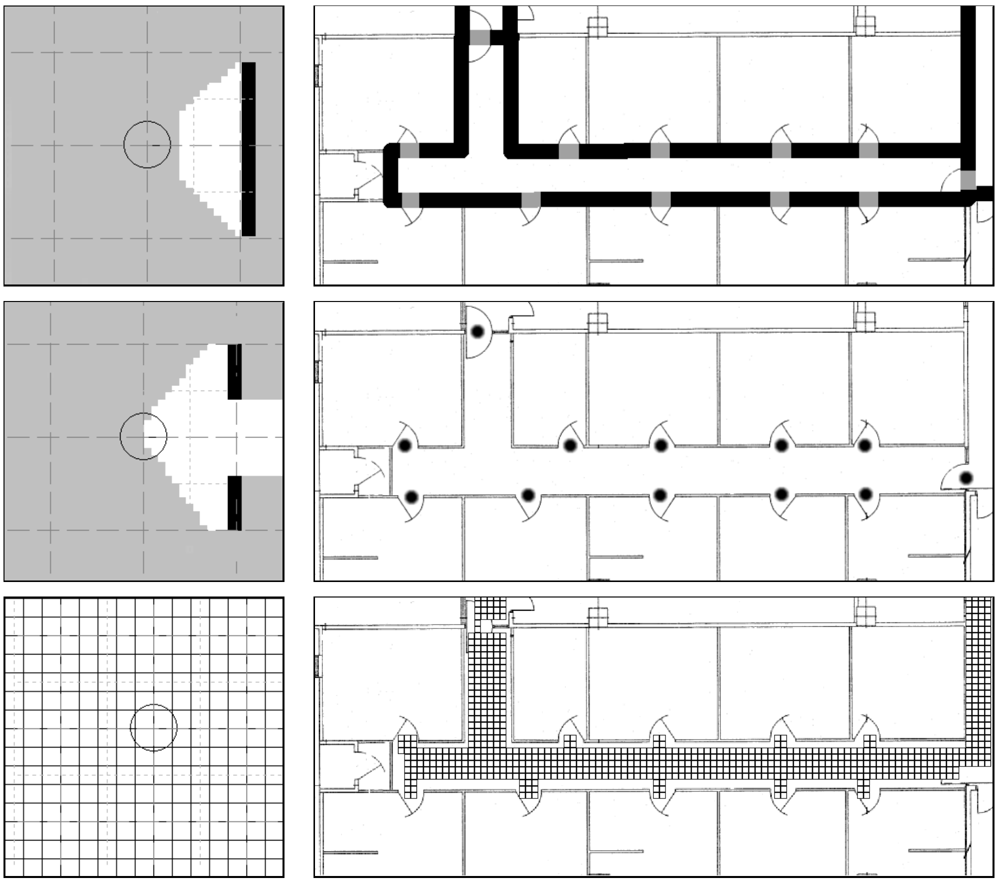


Fig. 6. Model for a fragment of the real environment. (Top row) Distribution of wall landmarks. The probability of observing a wall type landmark is indicated by the shading; hence the doorways (which may be either open or closed) are a lighter shade than the fixed walls. (Middle row) Distribution of doorway landmarks. (Bottom row) Location model — allowed locations are indicated by the grid cells.

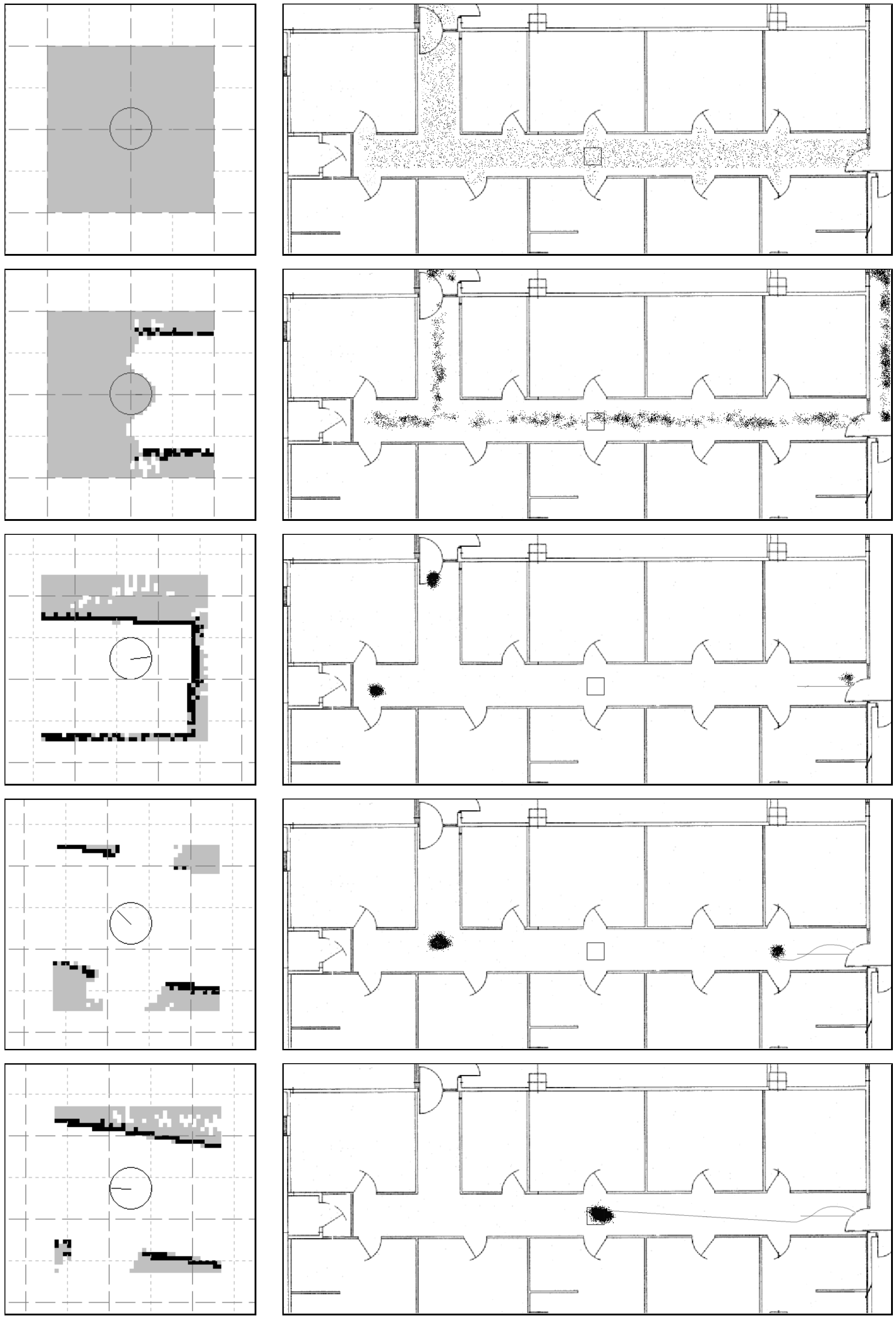


Fig. 7. Typical trial in the real environment using the persistent strategy. The figures on the left show the robot's local occupancy map. The figures on the right show the evolution of the probability distribution $p(r)$. The dotted line shows the actual path taken by the robot; the goal is indicated by the square.