Navigation without Localisation: A Reactive Network Approach

Andrew Howard, Les Kitchen

Computer Vision and Machine Intelligence Laboratory
Department of Computer Science
University of Melbourne, Victoria 3053
Australia
andrbh@cs.mu.OZ.AU, ljk@cs.mu.OZ.AU

Abstract

In this paper, we address the following problem: given a robot which is at some *unknown* location in a known environment, how does the robot go about reaching its goal? In resolving this problem, we have abandoned the notion of localisation and instead developed the concept of a *reactive network*. A reactive network is simply a specialized kind of finite state machine whose states correspond to actions and whose transitions correspond to observations. We show that it is possible to generate reactive networks that can be used to reach a particular goal from *any* initial location, and hence to solve the problem of reaching a goal from an unknown location. In this paper, we develop the reactive network concept, describe how they can be constructed and use simulation to assess their performance.

1 Introduction

One of the principal problems facing any mobile robot is that, sooner or later, the robot will become lost. If we wish to construct robots that can operate unattended for long periods of time, these robots must have some technique that enables them to determine that they are lost and to take steps towards recovery. Conventional approaches, which are based on localisation and path-planning, do not address this problem well. With a few notable exceptions [1], most attempts at making systems more robust have concentrated on increasing the reliability of the localisation process, either by physically altering the environment [2]; by using sophisticated sensor fusion and map registration techniques [3][4][5]; or by using planning techniques that consider localisation issues [6]. In effect, the emphasis has been on making sure the robot does not become lost at all. The problem these approaches are addressing can be stated as follows: given a robot which is at a known location in a known environment, construct a plan that will enable the robot to reach the goal. In this paper, we address a somewhat different problem: given a robot which is at some *unknown* location in a known environment, construct a plan that will enable the robot to reach the goal. That is, we assume that the robot is lost to begin with, and address the problem of how it can reach the goal under these circumstances. Our solution involves two important steps. Firstly, we abandon the notion of localisation: we show that a plan framed entirely in terms of actions and observations can be used for navigation. Secondly, we show that it is possible to construct a single plan that will enable the robot to reach the goal from *any* initial location, and hence will solve the problem of reaching the goal from an *unknown* location. Such plans can be encoded by a specialized kind of finite state machine we call a *reactive network*.

In the following sections, we develop the idea of a reactive network and show how it can be used; we describe a fast, simple algorithm for generating networks; and we use simulation to assess the performance of reactive networks, particularly in the presence of noise. We will also see that reactive networks can implicitly encode some very complex behaviour, including behaviour that is directed towards knowledge aquisition.

2 Navigation using Reactive Networks

Consider the simple office corridor environment depicted in Figure 1. We assume that we have a robot equipped with a set of sensors and a compass such that it is able to make basic classifications of its local environment. For example, the robot can distinguish a north-south corridor from an east-west one. We further

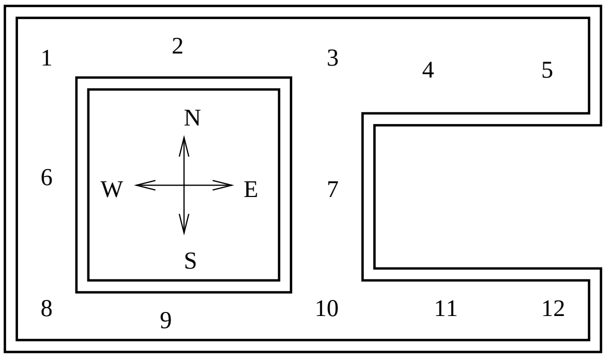

Figure 1: Simple office corridor environment

assume that the robot can move in any of the cardinal compass directions: north, south, east or west. Given these constraints, this environment can be modelled as set of locations (each of which has a classification) connected by actions. The model can be expressed in tabular form, as follows:

Loc	Class	(↑)	(↓)	(→)	(←)
L_1	╔		L_6	L_2	
L_2	═			L_3	L_1
L_3	╦		L_7	L_4	L_2
L_4	═			L_5	L_3
L_5	⊐				L_4
L_6	‖	L_1	L_8		
L_7	‖	L_3	L_{10}		
L_8	╚	L_6		L_9	
L_9	═			L_{10}	L_8
L_{10}	╩	L_7		L_9	L_{11}
L_{11}	═			L_{12}	L_{10}
L_{12}	⊐				L_{11}

Each row in this table describes a location; each of the columns describes the effect of taking an action. Note that not all locations can be uniquely determined by their classification; there are, for example, four different locations which are classified as east-west corridors.

The problem we wish to solve is as follows: given that the robot's initial location is unknown, construct a plan that will enable the robot to reach the goal. Traditionally, such plans are framed in terms of locations and actions. Consider the following plan for reaching location L_{12} in Figure 1:

Loc	Action	Loc	Action
L_1	(→)	L_7	(↓)
L_2	(→)	L_8	(→)
L_3	(↓)	L_9	(→)
L_4	(←)	L_{10}	(→)
L_5	(←)	L_{11}	(→)
L_6	(↓)	L_{12}	(→)

If, for example, the robot starts at location L_6, it will execute the path $\{L_6, L_8, L_9, L_{10}, L_{11}, L_{12}\}$. Similarly, one can verify that the robot will reach the goal from any other initial location. This plan encodes paths from all possible starting locations to the goal. The problem with this plan, of course, is that the robot's location must be known at all times. Sometimes this is not a problem: the corner L_1, for example, has a unique classification; a robot initially placed at L_1 can easily determine its location. However, for other initial locations (such as the corridors in this environment) unambiguous localisation is not possible. Under these circumstances, plans such as the one above will fail.

As an alternative, we propose an approach in which the robot's location does not need to be determined at *any* time. To see how this can be achieved, imagine placing a robot at L_6 and giving it the following plan:

- Step 1: move south until you reach a corner, go to step 2.
- Step 2: move east, past an intersection, until you reach dead-end, go to step 3.
- Step 3: stop, you are at the goal.

Despite the fact that this plan contains no information about location at all, it is nevertheless easy to verify that the robot will execute the path $\{L_6, L_8, L_9, L_{10}, L_{11}, L_{12}\}$ to reach the goal. In fact, we can generalise this plan so that it works for *any* initial location. Such a plan can be written compactly as follows:

$S, A(S)$	╔	═	╦	⊐	‖	╚	╩
S_1, ◯	S_2	S_2	S_3	S_5	S_3	S_4	S_4
S_2, (→)	S_2	S_2	S_3	S_5			S_4
S_3, (↓)			S_3		S_3	S_4	S_4
S_4, (→)		S_4		S_6		S_4	S_4
S_5, (←)		S_5	S_3	S_5			S_4
S_6, ◯				S_6			

Each of the rows in this table corresponds to a step or a *state*, each of which has an associated action. The columns specify the next state, given some particular observation. Clearly, this a special kind of finite state machine; we use the term *reactive network* to describe plans of this sort. The plan always starts at state S_1 and terminates at state S_6; if the robot reaches S_6, it should be at the goal.

To show how the above plan works in practice, we will consider two examples. First, imagine once again that the robot is placed at the north-south corridor L_6. We can write down a 'trace' of the robot's activity as

S_1	S_3	S_4	S_6
L_6	L_6, L_8	$L_8, L_9, L_{10}, L_{11}, L_{12}$	L_{12}

In this trace, the plan state is associated with the locations the robot is traversing. It should be read as follows: the robot starts at at location L_6 in state S_1; it observes that it is in an east-west corridor and changes state to S_3; the associated action for S_3 is 'move south', which the robot does until it comes to the corner at L_8, at this point it changes state to S_4; and so on. Inspecting the trace, we can see that the robot moves from starting state state S_1 to the goal state S_6, whilst simultaneously moving from location L_6 to the goal location L_{12}. In this case, we say that the plan *encodes* the path $\{L_6, L_8, L_9, L_{10}, L_{11}, L_{12}\}$. This is, in fact, the shortest path to the goal.

Consider next the case when the robot starts at the goal location. The trace is

S_1	S_5	S_4	S_6
L_{12}	L_{12}, L_{11}, L_{10}	L_{10}, L_{11}, L_{12}	L_{12}

That is, the robot first heads away from the goal, then returns to it. The ambiguity in location forces the robot to take steps to 'confirm' that it is at the goal; in effect, it checks to make sure it is not at the similar location L_5. The encoded path, $\{L_{12}, L_{11}, L_{10}, L_{11}, L_{12}\}$, is far from being the shortest path to the goal (which is of zero length); it is, however, the shortest path that will enable the robot to *know* that it is at the goal. Thus the plan encodes both steps for reaching the goal and steps for aquiring knowledge. This results in some quite startling robot behaviour, particularly in large environments with a high degree of self-similarity. The robot appears to be exploring its environment, gathering knowledge, when in fact it is simply working its way through a finite state machine. This complex behaviour has been encoded into the plan, where it becomes implicit rather than explicit.

In the previous discussion, we have been concerned exclusively with navigation in the absence of noise. In the real world, of course, noise introduces errors into both sensing and motion. Fortunately, with reactive networks such errors will almost always manifest themselves sooner or later as a *plan failure*. A plan failure occurs whenever the classification for the current location does not match any of the entries for the current plan state. For example, if the robot reaches state S_5 of the plan, but detects that it is in a corner, then the robot must have made an incorrect classification at some point. Having detected the failure, the robot can employ a failure resolution strategy. The simplest failure resolution strategy is to reset the plan and start the whole process again: after all, the plan is supposed to work for *any* initial location. More complex failure resolution strategies can also be implemented, one of which is described in Section 4. Unfortunately, it is impossible to guarantee that all errors will be detected. There is always a chance that the plan will be followed to completion, but that the robot will merely be at a location that *looks like* the goal. In Section 4 we show that such failures are rare, even when error rates are high.

3 Creating Reactive Networks

There are many algorithms one can employ to create reactive networks. It is possible, for example, to construct the space of all possible reactive networks, then search that space to find the network which is optimal for some particular goal. Optimality in this case might be defined strictly as some function of path lengths encoded by the plan, or it might also include measures of plan size and comlexity. This approach is, of course, computationally expensive. In this paper, we abandon considerations of optimality and instead describe a fast, simple algorithm for generating at least *one* solution.

The algorithm we employ makes use of the plan failure mechanism (noted in the previous section) to build plans incrementally. Imagine that we have some plan P which is known work for some particular starting location L_a. Using the environment model, this plan can be tested against another possible starting location L_b. There are four possible outcomes of such a test:

1. The robot will reach the goal.
2. The plan fails – the robot will make an observation that is not in the plan.
3. The robot will enter an infinite loop and travel in circles.
4. The robot will reach a place that looks like the goal, but is not.

If the plan fails, we have an opportunity to extend the plan so that is *does* work for the starting location L_b. Basically, the plan will fail with the robot at some location L_c, so we can take a plan that is known to work for L_c and append it to the plan P at the point of failure. The resultant plan will work for both L_a and L_b. Proceeding in this manner through all initial locations, it is possible to build up a the desired general plan. Of course, if the plan already works for a particular initial location, there is no need to extend it.

On the other hand, there are two failure modes that cannot be resolved by plan extension. If the robot reaches a false goal, or enters an infinite loop, the plan cannot work for this initial condition. In this case we have no choice but to discard the plan and repeat the process with a new one. Fortunately, the chances of the event occuring can be greatly reduced by applying some appropriate heuristics to the testing algorithm, and by careful construction of the plans. Infinite loops can be avoided entirely if plans contain no 'backward transitions', that is, if the reactive networks are unrolled to form reactive 'trees'. Likewise, the robot is unlikely to reach false goals if we first try to generate a plan which works for initial locations far from the goal. In our experiments with a variety of environments of varying size and complexity, the generation algorithm that employs this heuristic has always worked on the first attempt. At present, we are unable to prove that this algorithm will always generate a solution; we simply note that we have not been able to find an example for which it fails.

A detailed description of the algorithm is beyond the scope of this paper, but it can be summarized as follows:

1. Generate a specialized plan for each initial location.
2. Combine the specialized plans into a single general plan.
3. Compress the plan to form a smaller (but equivalent) plan.

In the first step, a simple backward-chaining algorithm is used to generate specific plans for each initial location. These plans are then combined using the failure-extension mechanism described above, starting with the plans for locations farthest from the goal. The last step is not strictly necessary, but the plans produced in Step 2 are very large and highly redundant, so it is useful to reduce them to smaller equivalent plans. A modified version of the finite state machine compression algorithm described in [7] is employed. The complexity of this algorithm is no greater than $O(n^2)$, where n is the number of locations, and the number of states in the generated reactive networks is of order n (i.e. they are quite compact).

4 Simulation results

We have used simulation to examine the properties of reactive networks in a wide range of indoor environments. Figure 3 depicts a simple grid-based environment containing 512 locations (32 wide by 16 high) which are either occupied or unoccupied. The simulated robot can determine the occupancy of the adjacent cells and can move in any of the cardinal compass directions: north, south, east or west. The goal location is indicated by the small square at the top-left of the figure. Application of the algorithm described in the previous section yields a reactive network with about 200 states, which is about the number of possible robot locations. The algorithm is implemented in Java and takes a few seconds to run on a Sun Solaris server.

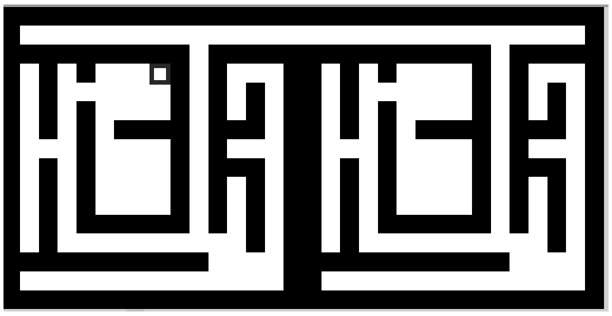

Figure 3: Simulated environment

The environment in Figure 3 has a high degree of self similarity, so a robot placed initially at random will find it very difficult to determine whether it is in the left or right half of this figure. If, for example, the robot is initially placed at the goal, it will execute a very complex series of actions which take it well away from the goal, simply to check whether or not it was originally at the goal. Having performed this check, the robot will return directly to the goal. Although this is hardly the shortest path the the goal, it is in fact the shortest path that can be executed that will allow the robot to *know* that is has reached the goal. In general, when the robot starts from locations close to the goal, the encoded path will be much longer than the shortest path. In contrast, when the robot starts far from the goal, the encoded path will be close to the shortest path. This reflects both the nature of the plan generation algorithm (which considers locations far from the goal first) and the requirements of knowledge aquisition.

The performance of the plan in the presence of noise was evaluated by running multiple simulated trials in which noise was present in the form of an *error rate*. For example, an error rate of 10 percent implies that noise in the sensors causes 10 percent of classifications to be incorrect. In each simulated trial, the robot was placed at some random initial location and the number of locations traversed to reach the goal was recorded. We also tested for false goals, i.e. the situation in which the robot completes the plan, but is not at the goal. If the naive failure resolution strategy described in Section 2 is employed, the time taken to reach the goal becomes very large, very quickly, even for low error rates. Consequently, we employ a slightly more complicated strategy based on the concept of 'retries'. If a plan failure is detected, a new set of sensor readings is aquired and a new classification generated. If the new classification does not result in a failure, the plan

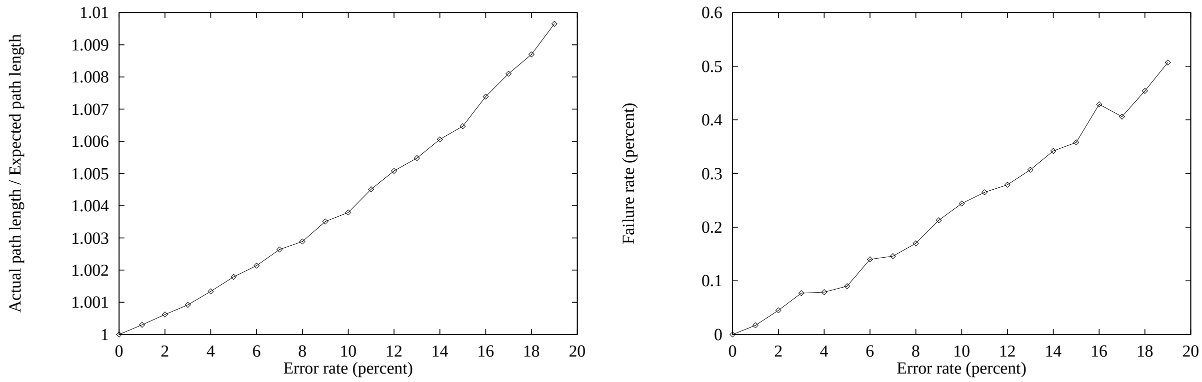

Figure 2: Performance of reactive network in the presence of noise.

proceeds. Otherwise, if the plan fails after a number of retries (5 in this case), it is restarted. There are many other strategies that could be employed, but this one is simple and yields good results.

The results from 2 million trials are shown in Figure 2, where the ratio of the actual path length and the expected path length (i.e. the error free path length) is plotted as a function of error rate, averaged over multiple trials. Thus, an average ratio of 1.0 means that errors are having no effect on robot performance. It is apparent from Figure 2 that average path lengths increase monotonically as a function of error rate, but that the rate of increase is modest. Even when 20 percent of classifications are incorrect, the robot's performance is still very close to its error free performance (within 1 percent). Also plotted in Figure 2 is the rate at which false goals are reached. The rate increases as a function of noise, but is never larger than 1 percent. It is possible that a different failure resolution strategy could reduce this error even further, perhaps at the expense of average performance. It is clear, however, that a reactive network with the failure resolution strategy described above can yield very robust navigation in a noisy environment.

5 Further work

We are currently testing the ideas presented in this paper with a small mobile robot equipped with a rotating sonar sensor and a simple vision system. In the future, we intend to extend the reactive network concept to environments which are both dynamic and probabilistic.

References

[1] 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.

[2] Bernard J Hendrey, Ray A Jarvis, and Ian Bridger, "An automated guided vehicle for industrial environments", in *Proceedings of the 1995 National Conference of the Australian Robot Association*, July 1995.

[3] Raj Talluri and J K Aggarwal, "Position estimation for an autonomous mobile robot in an outdoor environment", *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, 1992.

[4] Ingemar J Cox, "Blanche – an experiment in guidance and navigation of an autonomous robot vehicle", *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, 1991.

[5] Reid Simmons and Sven Koenig, "Probabilistic navigation in partially observable environments", in *IJCAI*, 1995.

[6] 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*, vol. 16, no. 10, 1994.

[7] Alfred V Aho and Jeffrey D Ullman, *Principles of Compiler Design*, pp. 101–102, Addison-Wesley, 1977.