Learning to Select Distinctive Landmarks for Mobile Robot Navigation

Stephen Marsland, Ulrich Nehmzow, and Tom Duckett*
Department of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL
U.K.
{smarsland, ulrich}@cs.man.ac.uk,tom.duckett@aass.oru.se

Abstract

In landmark-based navigation systems for mobile robots, sensory perceptions (e.g. laser or sonar scans) are used to identify the robot's current location or to construct internal representations — maps — of the robot's environment. Being based on an external frame of reference (which is not subject to incorrigible drift errors such as those occurring in odometry-based systems), landmark-based robot navigation systems are now widely used in mobile robot applications.

The problem that has attracted most attention so far in landmark-based navigation research is the question of how to deal with perceptual aliasing, i.e., perceptual ambiguities. In contrast, what constitutes a good landmark, or how to select landmarks for mapping is still an open research topic. The usual method of landmark selection is to map perceptions in *regular* intervals, which has the drawback of being inefficient and possibly missing 'good' landmarks that lie between sampling points.

In this paper we present an automatic and fully autonomous landmark selection algorithm that allows a mobile robot to select conspicuous landmarks from a continuous stream of sensory perceptions, without any pre-installed knowledge or human intervention during the selection process. This algorithm can be used to make mapping mechanisms more efficient and reliable.

Experimental results obtained with two different mobile robots in a range of different environments are presented and analysed.

1 Introduction

1.1 General Considerations — Approach

For mobile robot navigation over realistic distances, navigation systems based on perceptions or landmarks are usually used. This is because, unlike dead-reckoning methods, perception-based systems do not suffer from drift error. However, they are prone to the problem of perceptual aliasing, where a number of perceptions from different parts of an environment look similar.

There is another very relevant aspect of landmark-based navigation that has not yet been solved satisfactorily. This is the question of what constitutes a 'good' landmark, i.e., the question of how landmarks should be selected. So far, this problem has largely been ignored by either logging sensory perceptions in regular intervals, or comparing them with pre-installed models of 'good' landmarks supplied by a human supervisor. The first method is unsatisfactory because of its inefficiency, the latter is has the risk that a human supervisor will either select landmarks that are not easily recognisable by the robot, or overlook landmarks that are inconspicuous to a human, but perfectly recognisable by a robot.

In this paper we present an algorithm that addresses this question of landmark selection, and facilitates the selection of landmarks *automatically*, i.e., without any human supervision. The algorithm allows a mobile robot to select landmarks for navigation from a continuous stream of perceptions *autonomously.*

*Centre for Applied Autonomous Sensor Systems, Dept. of Technology, University of Örebro, S-70182 Örebro, Sweden.

The fundamental assumption we make is that a landmark that is suitable for navigation should be both conspicuous to the robot's sensors and reliably detectable. This paper first describes the landmark selection algorithm, which selects conspicuous landmarks. It then introduces performance measures that evaluate the consistency of landmark selection, and applies them to the experimental results obtained with two different mobile robots in a range of different real-world environments.

1.2 Method of Landmark Selection

We select landmarks by defining conspicuous landmarks to be those that are unexpected, i.e., perceptions that do not conform to an acquired model of the temporal relationship between successive sensory perceptions.

The landmark selection system described in this paper uses a neural network to learn a general model of the relationship between a robot's successive sensory perceptions, acquired over a range of different environments during a training phase. This general model is then used to predict future perceptions of the robot's sensors, based on current sensor readings. Any perception where the prediction does not match the actual reading obtained at the next time step is selected as a landmark on the grounds that it differs markedly from the usual relationship between successive perceptions: it is conspicuous.

Two methods of selecting those places where the prediction fails to predict the next perception accurately are proposed, one based on detecting peaks in the prediction error curve, and the other using a Kalman filter to maintain a model of the error and selecting as landmarks those place which fall beyond the error bounds of the Kalman filter.

1.3 Performance Evaluation

A very important feature of a landmark is that it can be *consistently* detected on every run, so that the robot can navigate using the landmarks. For this reason, we propose a metric by which we can evaluate the consistency of the landmark selection and compare the detection techniques. This is discussed in section 5 of this paper.

1.4 Key Contributions

The experiments discussed in this paper demonstrate that it is possible to devise algorithms that enable a mobile robot to select consistent landmarks for navigation *automatically* and *autonomously*. User guidance or the pre-installation of models is not required. One fundamental feature of the approach presented in this paper is that we use a model of the temporal relationship between sensory perceptions, i.e., a *robot-centred* rather than an *environment-centred* model. This means that the algorithm is able to select landmarks in any environment — even completely novel ones — after training.

The experiments have been conducted on two different mobile robots, in a range of different environments. The results demonstrate that the algorithm is not sensitive to hardware aspects, nor properties of a specific environment.

In order to assess the consistency of landmark selection, we introduce two measures of performance: the sum-squared-difference between prediction errors of different traversals of the same environment, and the landmark consistency measure LMC, which evaluates whether the algorithm selects the same location as landmark during every traversal of the same environment.

2 Related Work

A number of methods of selecting landmarks are described in the literature. Some systems (Yamauchi and Beer, 1996; Duckett and Nehmzow, 1998) take sonar scans at regular intervals, such as every 150 cm of travel in an environment, with every sonar scan being used as a landmark that is put into the map. This avoids the problem of ensuring that landmarks should be detected consistently, but since in many environments — such as corridors — a large number of perceptions are identical, the map is filled up with information that does not aid navigation.

Another problem with mapping at regular intervals is that perceptions that lie between sampling locations are missed. It may well be that these missed perceptions include considerably more distinctive landmarks than those perceived at the user-defined sampling locations.

A number of researchers have considered selecting suitable landmarks for mobile robot navigation. These can be split into off-line approaches which are not suitable for use on the robot as it explores, and systems which can be used on-line.

In the first category, is the technique of asking the user to define the landmarks before the robot explores. Humans typically select objects which they find easy to recognise, such as doors, or line segments extracted from camera images recorded as the robot travels (Kortenkamp and Weymouth, 1994). These approaches suffer from the same problem - the robot's perceptions of the world are very different to those of a human, and so features that the researcher thinks are distinctive may not be noticeable by the robot.

In contrast to this, Thrun (1998) addressed the problem of Bayesian learning to select an optimal set of landmarks for performing self-localisation in one specific environment, where the landmarks consisted of a projection from the robot's raw sensory perceptions (camera images) onto vectors in a lower-dimensional space. This optimisation was performed by minimising directly the quantity of interest, namely the robot's error in self-localisation. Thrun showed that his method produced better performance than localisation using designer-determined landmarks including doors and ceiling lights. A similar but computationally cheaper technique was developed by Vlassis et al. (2000), who showed that their optimisation method produced better results than principal component analysis. Our approach differs in that we do not carry out any analysis of the utility of the landmarks selected, but instead use a self-acquired model of 'typical' sequences of perceptions which is independent of any particular task or environment.

Neither of the above approaches is suitable for *on-line* selection of landmarks, because they rely on an acquired model of a previously explored environment. Zimmer (1996) considered the problem of selecting landmarks in a topological map through a process of 'life-long learning', where the robot's map was continuously adapted on-line during exploration. This approach used global statistical information, based on comparison of accumulated error statistics at each of the nodes, to decide where to add and delete nodes in the map. A related idea can be found in Bourque and Dudek (2000), who addressed the 'vacation snapshot' problem of deciding in which locations to take camera images in order to obtain a set of images which best represents an entire environment. This approach kept running statistics on what is a 'typical' perception, together with backtracking to previously visited locations which were subsequently found to be 'atypical'. Our method differs from these approaches in that we only use *local* sensory information to decide when to add landmarks to the map.

Simhon and Dudek (1998), also use only local information, but by using a function which is carefully designed for the particular environment in which the robot is to be used. They addressed the problem of deciding the best locations in which to create local metric maps, using a distinctiveness function based on the robot's ability to localise itself within such a map. However, this approach relies heavily on the designer's intuition in selecting an appropriate distinctiveness function, and is only suitable for selecting landmarks that can be used for self-localisation.

3 Acquiring a General Model of the Relationship Between Subsequent Sensory Perceptions

The first part of the proposed system for selecting landmarks is the general model of the relationship between subsequent sensory perceptions of the robot. The perceptions are sampled at regular spatial intervals as the robot travels, using dead reckoning to estimate the distance between samples.

One way for the robot to acquire the model is through the use of a neural network trained on data collected by the robot while it explores. Since we want the model to predict the next perception that the robot will encounter, a supervised learning technique is suitable, where each input vector is paired with a target output (in this case the sensory perception perceived at the next time step), and the weights of the network adjusted to match that output for the current input more closely, usually by a gradient descent learning method.

3.1 The Neural Network Architecture

The most simple supervised network is the Perceptron (Rosenblatt, 1962). The inputs to the network are directly connected to the outputs by adjustable weights, and the update rule modifies these weights. More complex networks have intervening layers of 'hidden' units, and the difference between the network output and the target is propagated back through the network to adapt the strength of the connections between the units.

In the current application we do not desire a complex model of the perceptions. Rather, we want a very simple model which contains only the most general features, so that there are environmental features for which the prediction is not good, since poor prediction is what generates the landmarks. For this reason we use a Perceptron rather than any more complicated model. This network calculates the output o at the current time using equation 1.

$$o_i = f(\mathbf{w}_i \cdot \mathbf{i}), \tag{1}$$

where $\mathbf{i}$ is the input vector, $\mathbf{w}_i$ the weight vector of output unit i, and f is the transfer function. In our experiments this was the sigmoid function given in equation 2.

$$f(x) = \frac{1}{1 + e^{-2x}}. \tag{2}$$

The network weights are updated using equation 3:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta(\tau_i - o_i)\mathbf{i}, \tag{3}$$

where τ_i is the target value for output unit i, and η is the learning rate (0.2 in our experiments).

3.1.1 Modifying the Connectivity

As the prediction that is being made is of future sensory perceptions, and the robot is always driving forwards, the future perceptions of any one sensor depend only on the sensors in front of it. The network connectivity was chosen to model this fact, so the perceptions of sensors facing forwards are used as inputs to model rear-mounted sensors, but not vice versa. The reading of a sensor at the current time is also used in the prediction of its own next reading. The architecture is shown in figure 1 and is referred to in this paper as the 'modified Perceptron'. In addition to the network inputs shown in the figure, a bias input which is permanently set to -1 is also used.

3.1.2 Adding Temporal Inputs

The modified Perceptron introduced in the previous sections learns to predict the next reading given the current one. Another possibility of modelling the temporal relationship between sensory perceptions would be to use information from past perceptions. We considered two methods of doing this.

The first is the lagged Perceptron, shown in figure 2. This network uses a lagged input vector, so that the inputs to the Perceptron at the current time consist not just of the current readings, but also the readings of n previous time steps.

The second technique, shown in figure 3, is the recurrent Perceptron. In this network, the inputs to the network are the outputs at previous time steps (as well as the current perceptions), so there is a recurrent mapping between the outputs at the current time and the inputs at the next time.

In both cases, the weight update rule described in equation 3 was used in exactly the same way as before, and the bias input is still present. For both of these networks, we used $n = 1$, so that only one previous time step was considered. This is because preliminary investigations showed that this generalised the best between different environments.

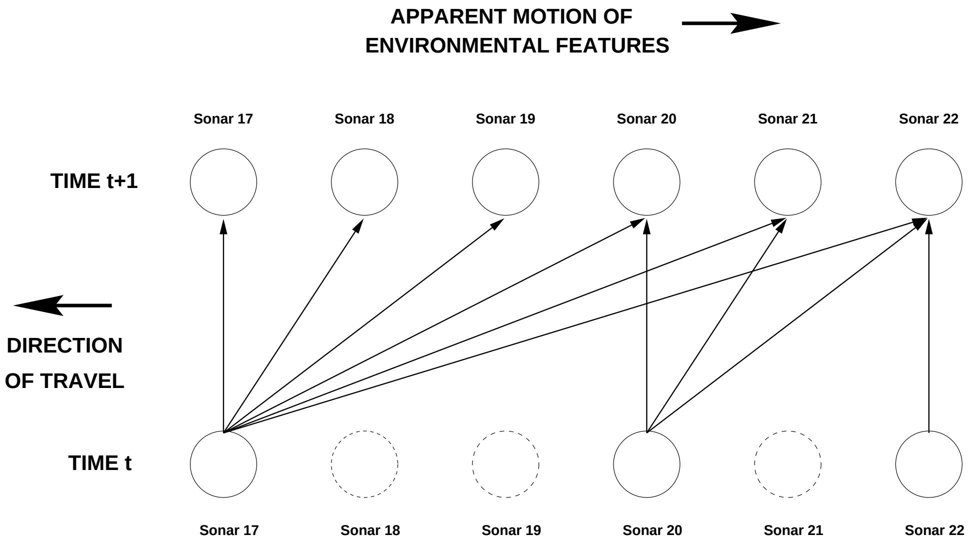

Figure 1: The neural network architecture of the modified Perceptron. All connections are shown. A bias input, permanently set to -1, which was connected to all output units was also used.

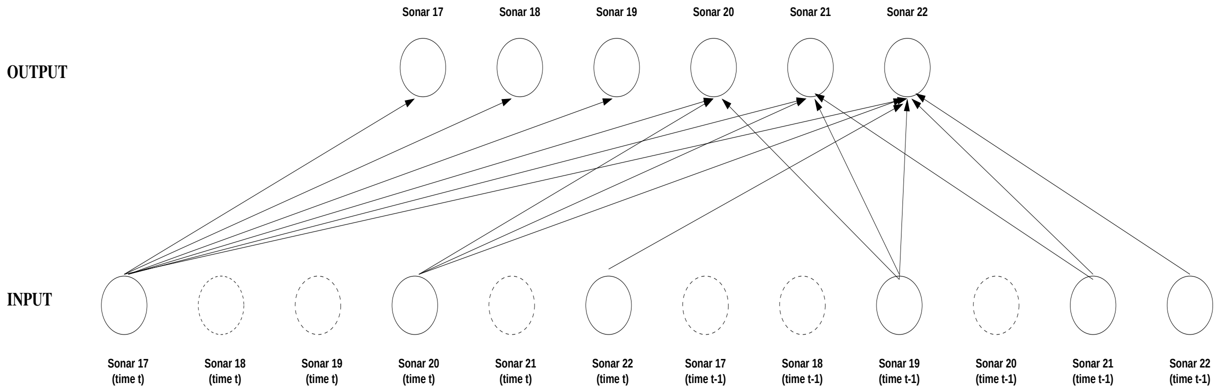

Figure 2: The lagged Perceptron. The inputs to the network comprise the sonar readings at times t and $t-1$, with the reduced connectivity used for both sets of inputs. For clarity, the connections are only shown for a subset of the neurons.

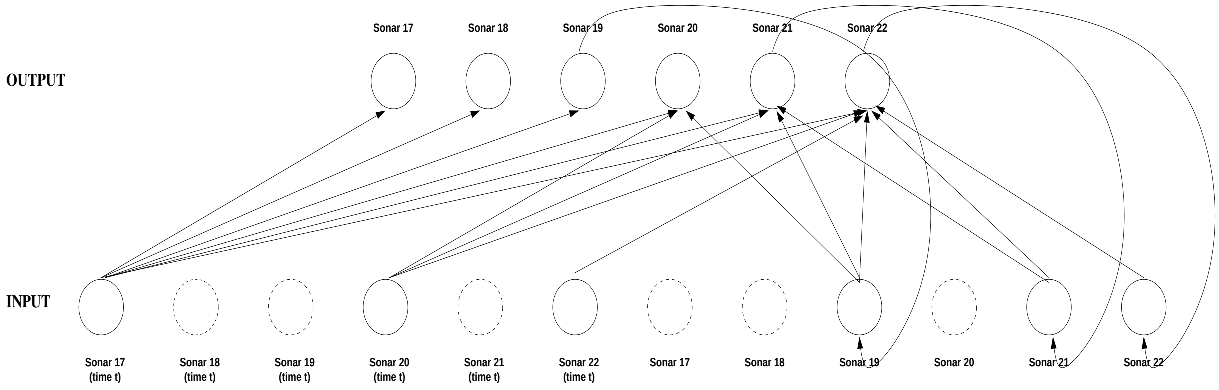

Figure 3: The recurrent Perceptron. The outputs of the network, the predicted perception at the previous time step, are used as inputs as well as the current perceptions of the robot. Again, for clarity, the connections for only some of the neurons are shown.

3.2 Sensor Signal Preprocessing

The sonar readings that are used as inputs to the neural network are intrinsically very noisy. Typical problems include cross-talk, where a sonar pulse emitted by one sensor is received by another and mistaken as its own, and specular reflection, where a sonar pulse rebounds from a surface away from the robot and therefore suggests that objects are much further away than they actually are. This results in a spike in the sonar readings.

One way to deal with this problem would be to ignore any sonar response that differs from the previous one by too much. However, this would be a problem when the robot went past an open doorway, when there would, correctly, be a large discontinuity in the readings. Instead, a median filter can be used. This filter replaces the value at each time t, with the median average of the previous k time steps. Since the door is wide enough to appear on several readings, it will pass the filter.

In our experiments, k was chosen to be 5, so that for each sensor input, the median of the readings for that sensor at times $t, t-1, \ldots, t-4$ replaced the reading at time t. The problem with using the median filter is that it blurs the edge between the readings before and after the discontinuity, delaying the perception of the feature. However, in the present application this is not a problem, as the same effect will occur each time the robot passes that place and so the perception will be the same on each occasion.

3.3 Network Training Procedure

One factor that affects neural network performance is the amount of training which is done. For the type of behaviour which we want the model to exhibit, that is a generalised prediction which is not specific to any individual environment, this is especially important. In this work the technique of early stopping (Bishop, 1995) was used to decide how many learning iterations should be applied. Early stopping monitors the error of the network, evaluated on an independent validation set, stopping the training of the network when the prediction error of the validation set stops decreasing. This suggested that about 20 epochs, or complete runs through the data, was sufficient with the learning rate of $\eta = 0.2$ that was used. The data that was used for this purpose was collected from 10 runs through an environment. Five of the runs were used as training data and the other five as validation data. Once the number of epochs required was found, the data was discarded.

Another important problem is to decide how big the training set should be, that is, how many environments should be used, and how many runs from each environment should be used. The second question was not considered to be a problem. 20 runs were collected in each environment, and these were split into two sets of 10, with one set for training and one set for testing. Since we want the network to learn a general model of the entire environment in which the robot will operate, it makes sense that the network should be trained by using examples from a large number of the different environments which the robot can possibly encounter in operation.

For this reason, in the results which are presented here two different types of training set were considered. In the first, data from three different environments in which data was collected were selected, with different combinations of environments used, while in the second, data from all of the environments was used.

4 Selecting Landmarks with the Trained Network

Once the neural network has been trained, the current perceptions can be fed into the model to produce a prediction of the next perception. This is done by calculating the output value, o, using equation 1. This output is then used in the current work by then calculating the error, F_1, at the current time by computing the squared difference between the predicted output and the actual perceptions obtained at time $t+1$.

In the landmark selection technique presented here, we use the places where the prediction of the general model is bad — that is, the places where the error is high — as landmarks. The 'quality criterion' for suitable landmarks here is that they are selected *consistently*, and our hypothesis is that the places where the model breaks down (resulting in bad predictions) will be the same on every run. The problem is to detect these peaks in the error curve which are above the level of noise in the data. Two ways of doing this are described below.

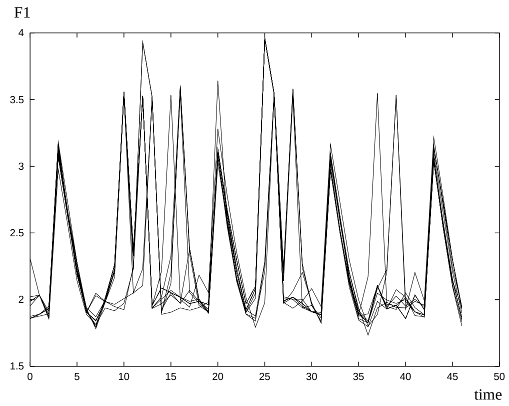

Figure 4: Typical output error curves over several runs in the same environment. The way the error occasionally rises above a baseline can be seen, as can the way the different curves match well at some points, but less so at others.

4.1 Selection Method 1: Peak Detection

The first method of detecting locations where prediction is bad is to detect peaks in the error curve. There are a number of ways in which peaks can be found. The problem is that the error curve is very noisy, as can be seen in figure 4 which shows a typical error curve. Using a Perceptron to train the network means that the accuracy of the predictions is never particularly high, so we are looking for peaks which stand out above the residual noise level.

The peak detection itself can be done by looking for zero crossings of the first derivative. This is approximated by considering points where the sign of $\frac{\Delta t}{\Delta F_1}$ changes. F_1 is defined as

$$F_1(t) = \sum_j [o_j(t-1) - i_j(t)]^2 \tag{4}$$

where $o_j(t-1)$ is the output of the network at the previous time step and $i_t(t)$ is the perception of the robot at the current time step.

A typical error curve is shown in figure 4. It can be seen that, while there are a number of significant peaks in the error curve, in addition there are several very minor peaks within the noise level of the data. These will also be detected by using the peak detection algorithm, which is undesirable. For this reason, a number of filtering steps were carried out before the peak detection. These filters are designed to smooth the curve, reducing the number of peaks, so that only those which are significant stand out. The process that is performed is shown in figure 5 and is described below. More detailed analysis of the effects of the results are given in Duckett and Nehmzow (1999) and Nehmzow et al. (2000).

Integration The current error value, $F_1(t)$, was integrated with the previous 9, effectively providing a mean average of the 10 values.

$$F_2(t) = \sum_{i=0}^{9} F_1(t-i) \tag{5}$$

Median Filter The last nine integrated values were median filtered in order to remove surface noise from the curve. This provides a value $F_3(t)$ for each time t.

Smoothing The curve was then smoothed, removing any remaining small peaks by using a momentum term from previous time steps (equation 6).

$$F_4(t) = \lambda F_3(t) + (1-\lambda)F_4(t-1), \tag{6}$$

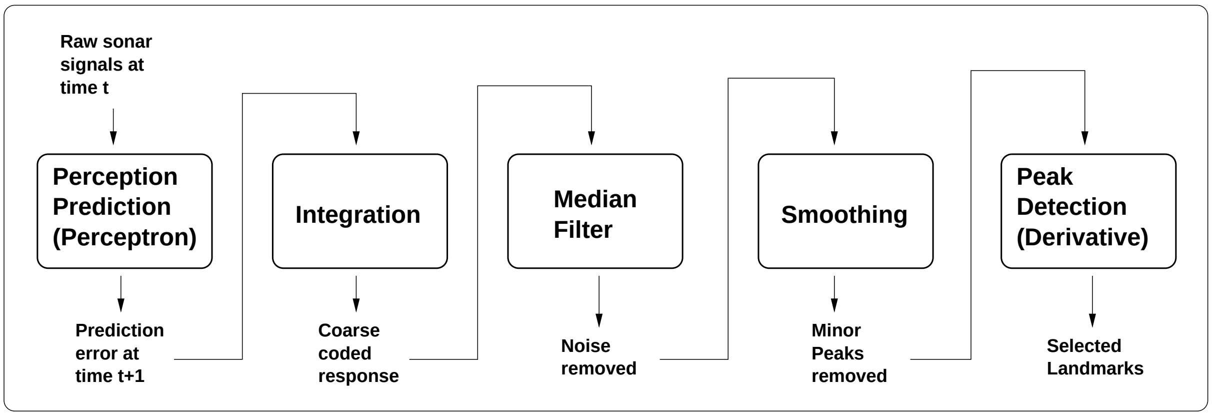

Figure 5: The filtering mechanism applied before peak detection.

where a value of $\lambda = 0.2$ was used. This means that the filter uses information about the previous value to smooth out the curve. The drawback of this technique is that it also pushes the position of the peak back, so that the robot does not perceive the peak in the place where the readings were taken, but some way further along.

4.2 Selection Method 2: Kalman Filtering

The filters described in section 4.1 have the side effect of delaying the appearance of peaks in the data and smearing them across the data. For this reason, an alternative approach was considered, using a Kalman filter (Kalman, 1960) to maintain a model of the error curve and looking for places where the error did not fit the model.

The Kalman filter was designed to produce successively better approximations to the state of a linear process which is controlled by a difference equation. It works by maintaining a continuously updated estimate of the state of the process being measured (here, the predicted sensory perceptions), including an estimate of the error in the measurements. By using the filter to keep a model of the error curve and detecting when the predictions of the model are beyond error bounds, we can detect outliers to the data at the current time. These outliers are suitable candidates for landmarks. For further details, see Marsland and Nehmzow (2000).

The implementation used in this work is a very simple Kalman filter. For a more complete description, see Maybeck (1990). The Kalman filter maintains an estimate of the covariance of the data (in this simple, one-dimensional case, the variance, v) and uses that to give error bounds to the predictions, using the *Kalman gain,* K. At each time step, a number of equations are applied. The first, equation 7, calculates the Kalman gain for the new iteration, based on the estimate of the variance at the previous time step. R is an estimate of the variance in the measurement error, that is, the error which is introduced in the reading of the state. Then, in equation 8, the new estimate of the state of the predicted sensory perception, $y(t)$, is calculated. This includes a (noisy) measurement of the predicted sensory perception taken at the current time, $x(t)$. Finally, equation 9 shows how the variance is updated.

$$K(t) = \frac{v(t-1)}{v(t-1)+R}, \tag{7}$$

$$y(t) = K[x(t) - y(t-1)] + y(t-1), \tag{8}$$

$$v(t) = v(t-1) - Kv(t-1). \tag{9}$$

The value of R, the variance in the measurement noise, and the initial value of v, the variance in the data, were estimated from the training data which was used to train the neural network. The Kalman filter was used to keep a continuously updated estimate of the state of the error curve. The variance v also suggests bounds on the error of the prediction. These bounds were used to define landmarks — a landmark was placed wherever the actual observation differed from the prediction by more than four standard deviations. The value at which a data point becomes an outlier (here: four standard deviations) can be altered to select greater or fewer landmarks. The four standard deviations value was chosen through an investigation of the data that was used for training, with the aim being to select landmarks approximately every 2 m.

5 Assessing the Landmark Prediction Mechanisms

This work investigates three different neural network architectures (the modified Perceptron, the lagged Perceptron and the recurrent Perceptron) for model acquisition, all with and without median filtered inputs, and two methods for selecting landmarks from the error curves of these networks, peak detection and the Kalman filter.

In order to be able to evaluate the performance of all these different methods it is therefore necessary to have some metric so that we can compare them. Two different assessment methods are introduced in this section. Both work by comparing pairs of runs in one environment.

Where we have 10 test runs in an environment, we compare the 1st run with the 2nd, 3rd and so on, the 2nd with the 3rd and 4th, etc., so that we obtain 45 pairs for comparison ($9+8+7+\ldots+1=45$).

The first evaluation method investigates how well two error curves obtained in different runs match, by calculating the sum-squared difference between the two curves. The second evaluates the consistency of the landmark selections, where a landmark is considered to be consistent if it is selected in the same place repeatedly on successive runs.

5.1 Assessment by Sum-of-squares Error

A standard way of comparing two ordered datasets is to measure the square of the distance between the two sets at each point, and to sum these values over the number of elements. This measure increases rapidly when the distance between two points is large, which is a useful feature for our purposes because small deviations between the error curves on two runs are not a problem, but a large difference is likely to lead to a landmark being predicted in one run, but not the other.

Comparing the error curves before the different landmark selection mechanisms are applied allows the performance of the different neural network architectures to be evaluated. We would expect that good performance would mean that the error curves would be similar every time.

5.2 Assessment by Measuring the Consistency of Landmark Selection

The second measure that we propose is a way of evaluating the consistency of a landmark. We say that a landmark is of a high consistency if it is detected in the same place on each run. The following measurement, termed LandMark Consistency (LMC) measures this, but allows for some leeway so that a landmark which has moved by one or two sampling points is also given some credit. The measurement is conducted in the following way:

For each test run in a particular environment, a vector is made. There is one element of the vector for each sampling point in the environment. Each element of the vector contains information about whether or not that place has been selected to be a landmark. If it is, the value $\frac{1}{\sqrt{2}}$ is used, if the place is adjacent to a landmark the value $\frac{1}{2}$ is used, and otherwise a 0 is put into the element. This produces a vector with as many elements as there were samples in the environment, a typical section of which may look like

$$(\ldots 0, 0, \frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{1}{2}, 0, 0, \ldots) \tag{10}$$

The vectors for two different runs are then multiplied together using the scalar product, and the average for each landmark is taken by dividing the result by the number of landmarks detected in the run. If the number of landmarks is different in the two runs, the maximum is taken. This means that if the number of landmarks is not the same in both runs, the value of LMC is lower. The values $\frac{1}{\sqrt{2}}$ and $\frac{1}{2}$ were chosen so that a perfect match between a landmark in two different runs gives a value of 1, and close misses a value of $\frac{1}{\sqrt{2}}$. The information about a landmark being present was spread across three elements of the vector to allow for error in the measurement of the robot's odometry, which could change the position of the landmark. In the event that two consecutive positions were recognised as landmarks, both landmarks would be marked as $\frac{1}{\sqrt{2}}$. In this case, a perfect match would result in a value greater than 1 being produced. This has not happened in any of the testing. The measure can be written in the following way:

$$LMC(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\max(N_a, N_b)}, \tag{11}$$

where N_a is the number of landmarks in vector $\mathbf{a}$ (similarly for N_b), and LMC is the LandMark Consistency performance measure.

Using this measure, each comparison between two runs yields a value between 0 and 1 (assuming that landmarks never appear in consecutive elements), where a 0 means that no landmarks correlated, and 1 means that all the landmarks matched perfectly. The two possible different overlaps (of two elements and one element) produce values of $\frac{1}{\sqrt{2}}$ and $\frac{1}{4}$ respectively.

6 Results

6.1 Data Collection

The mechanism proposed here selects landmarks based on the robot's perceptions. The system is entirely passive, it receives data from the sensors, but does not have any control of the motor actions. For this reason, although the system is designed to run online while the robot is in operation, using data collected by the robot in a number of environments is sufficient. Online operation is demonstrated and discussed in section 6.5.

The data used in the experiments described in this paper were collected on two different robots. The first was the Bremen wheelchair, *Rolland*, which is shown in figure 6.

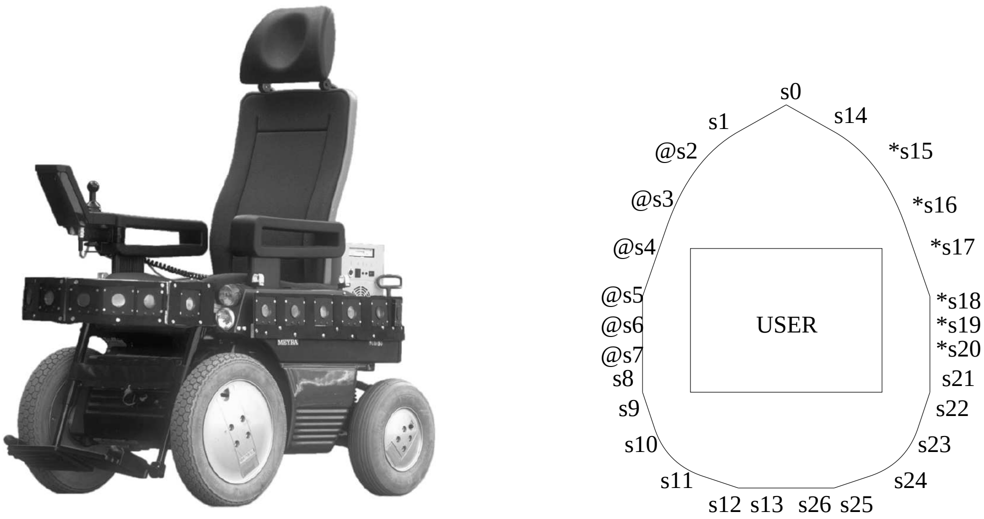

Figure 6: *Left:* The Bremen semi-autonomous wheelchair, *Rolland*. *Right:* The arrangement of the 27 sonars on the wheelchair. @ and $*$ denote the sonars which were used for data collection.

The robot is a motorised wheelchair, made by the German company Meyra, which has been augmented with 27 sonar sensors arranged around the robot. The positions of the sensors are shown on the right

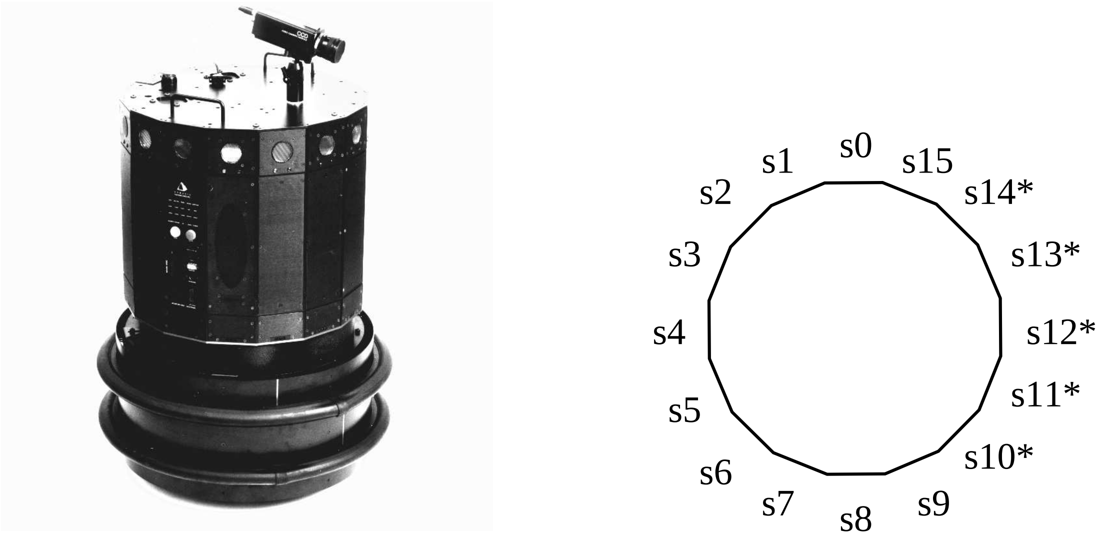

Figure 7: *Left:* The Manchester Nomad 200, *FortyTwo*. *Right:* The arrangement of the 16 sonars on the robot. $*$ denotes the sonars which were used for data collection.

of figure 6. The wheelchair was used to collect data in seven different environments ($A - G$) on the 8th floor of the MZH building at the University of Bremen. The layout of this building, and the location of the various data runs is shown on the left of figure 8. The second robot was the Manchester Nomad 200 robot, *FortyTwo*, shown in figure 7. The robot has 16 sonar sensors arranged in a ring around its circumference. This robot was run in a number of environments on the second floor of the Computer Science department at Manchester University. A map of this area is given of the right on figure 8.

The experimental procedure was the same for both sets of experiments, in Bremen and Manchester. It consisted of the following steps. The robot was placed at a starting location in the environment. From there the robot moved in the environment, pausing every 20 cm and recording the values of the sonar readings at the current place. The path of the robot through the environment was constrained to be a straight line parallel to the wall. The trajectory was controlled through steering with the joystick on the wheelchair, and using a wall-following behaviour produced by a force-field approach on *FortyTwo*. Further details are given in Marsland and Nehmzow (2000) for the wheelchair and Nehmzow et al. (2000) for *FortyTwo*.

The technique of early stopping, described in section 3.3, was used to decide when network training should be terminated. This was done by obtaining data from 10 runs through an environment, and using five of them as training data and the other five runs as validation data. Using this method, it was established that 20 training epochs was sufficient; the data of those 10 runs was then discarded.

For the experiments proper, we obtained data from another 20 data collection runs in each of a number of environments. Of these 20, 10 were used as training data and 10 were used as test data.

We investigated how many of the environments should be used to generate the training inputs, experimenting with using training data from three environments and from seven environments, the maximum number available. However much data is used, the trained network should be able to be used in all environments, including those it was trained in.

6.2 Evaluation Using the Sum-of-Squares Error

Comparing the performance of the trained neural networks before the landmark selection algorithm is used can give useful information about how consistently the networks are performing, and which features are useful for landmark selection. This comparison is done using the sum-of-squares error. The results given in this section were generated using the data collected on the wheelchair at Bremen University.

The different trained networks were all evaluated over the 10 test runs in each environment. The sum-squared difference between each pair of curves was then evaluated, leading to 45 $(9 + 8 + \ldots + 1)$ values for each network in each environment. These numbers were averaged to produce an average error

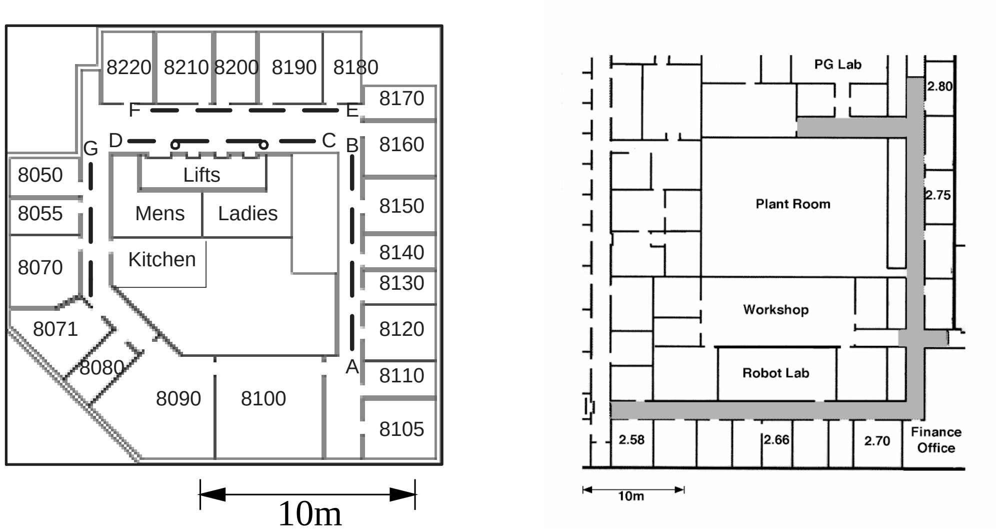

Figure 8: The environments on the 8th floor of the MZH building at Bremen University where data was collected. The letters show where that data run started, and the dashed lines show the intended robot trajectories. *Right:* The 2nd floor of the Computer Science Department at the University of Manchester, where further data was collected.

for each network in each environment. Figure 9 shows the average, best, and worst error values for each of the network architectures tried. In addition, table 1 shows the mean sum-squared error and the standard deviation, averaged across the different test environments.

Algorithm	Training Environments	Mean Error	Standard Deviation
Modified	ABC	16.28	10.61
Perceptron	ABE	25.62	16.48
	BDE	25.68	16.45
	$ABCDEFG$	25.64	16.47
Recurrent	ABC	0.44	0.28
Percepton	ABE	0.64	0.36
	BDE	0.32	0.32
	$ABCDEFG$	0.45	0.28
Lagged	ABC	0.53	0.36
Perceptron	ABE	1.21	0.72
	BDE	1.17	0.75
	$ABCDEFG$	1.13	0.73

Table 1: The mean sum-squared error and the standard deviation, for the neural network error curve, averaged over the testing environments, for the different network training algorithms. The systems were tested using independent data from training sets collected in environments $A - G$.

These results show that the modified Perceptron generally produces the highest errors. This makes sense since this network receives less input that the other networks. The huge errors appear to be caused by places where a landmark is predicted in some runs, but not in others.

By comparing the graph for training in all environments ($ABCDEFG$) at the bottom right of figure 9 with the others, it can be seen that the additional training does not make a significant difference compared to training in only three. A paired Student t-test (Press et al., 1992) comparing the sum-squared error after training in environments BDE and in environments $ABCDEFG$ confirmed this ($p = 0.01$). Using the t-test on the sum-squared error curves for the modified Perceptron and the lagged Perceptron suggests that the chance of them coming from the same distribution is low. The probability of the results of the lagged and recurrent Perceptrons being from the same distribution is even lower.

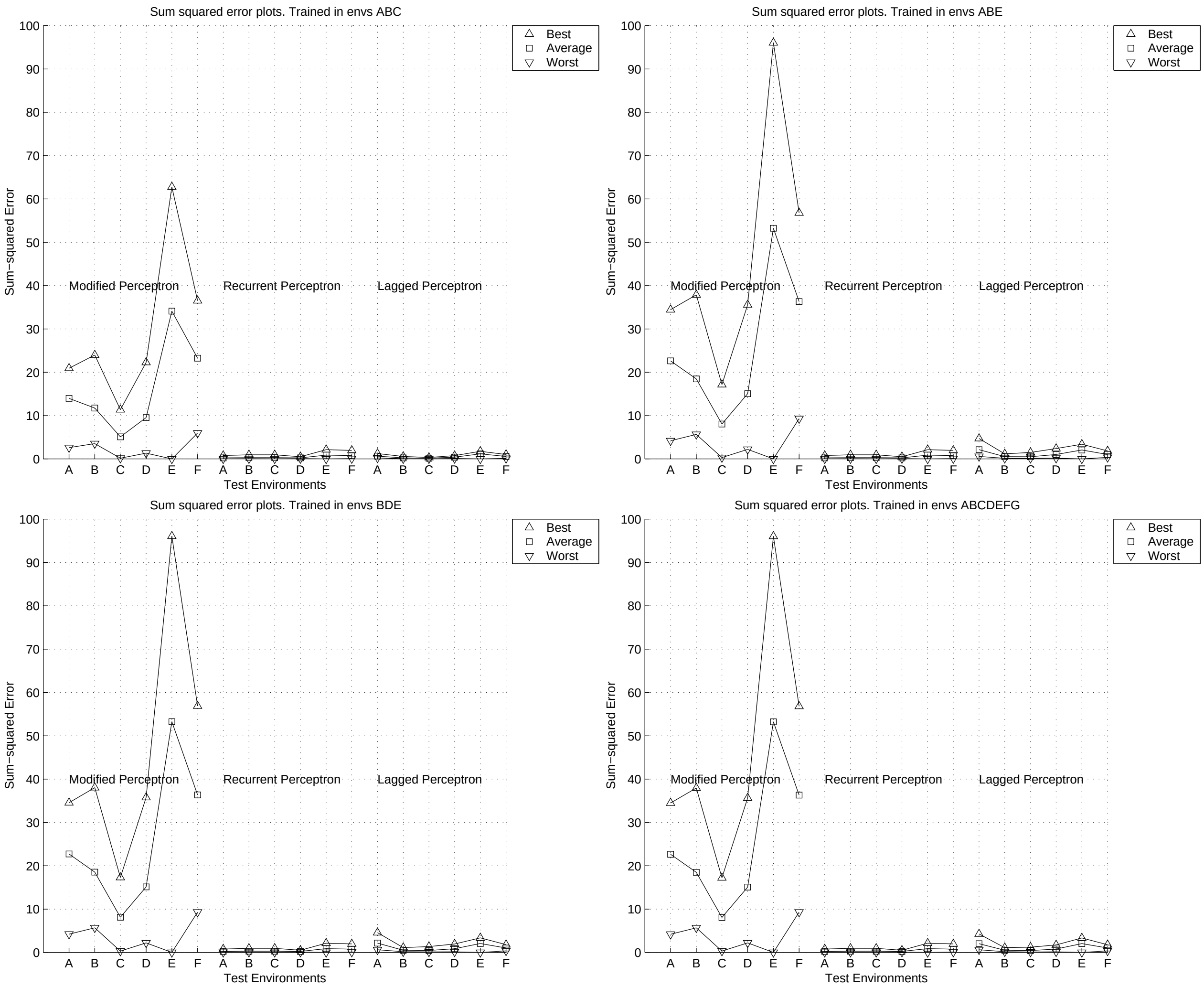

Figure 9: The results of evaluating the sum-of-squares error for the output of each of the different neural networks. The error is shown for each network, tested in each of environments A to F. The four pictures show the results after training in different sequences of environments (ABC, ABE, BDE, $ABCDEF$G, respectively).

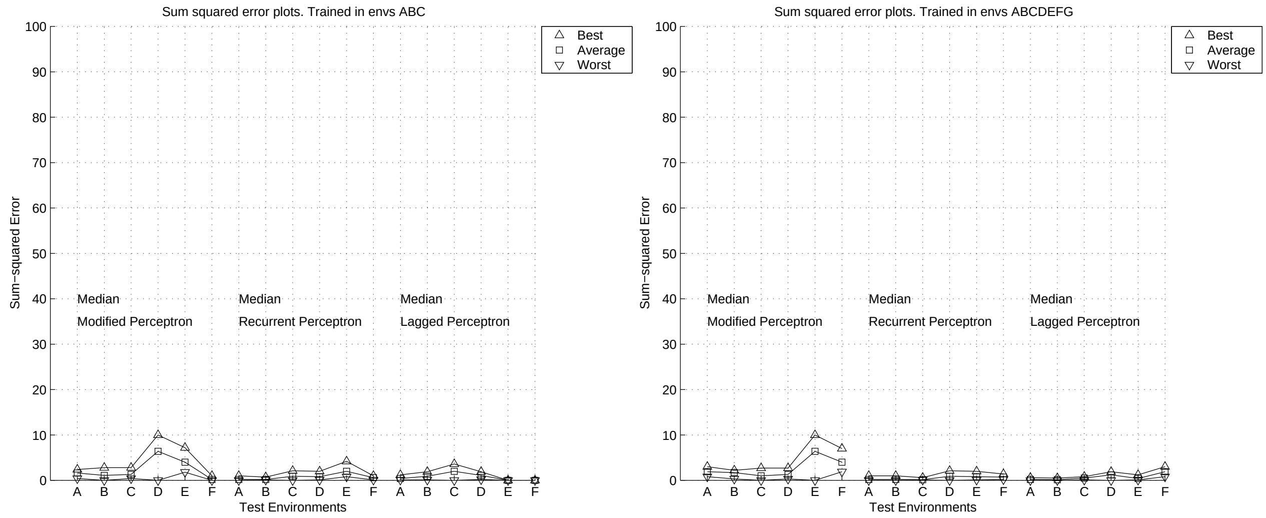

Figure 10: The results of evaluating the sum-of-squares error for the different neural networks with median filter preprocessing. By comparing with figure 9 we can see that it does not make difference for the lagged and recurrent Perceptrons. The error is shown for each network, tested in each of environments A to F. The pictures show the results after training in environments ABC and $ABCDEFG$ respectively.

Algorithm	Training Environments	Mean	Standard Deviation
Modified	ABC	2.69	2.11
Perceptron	$ABCDEFG$	2.42	2.32
Recurrent	ABC	0.51	0.27
Perceptron	$ABCDEFG$	0.78	0.66
Lagged	ABC	0.73	0.67
Perceptron	$ABCDEFG$	0.71	0.77

Table 2: The mean sum-squared error and the standard deviation, for the neural network error curve, averaged over the testing environments, for the different network training algorithms with the inputs median filtered. The systems were tested using independent data from training sets collected in environments $A - G$.

Figure 10 shows the sum-squared error curves generated when the inputs to the networks are first median filtered, and table 2 gives the mean and standard deviation in the measurements. The results are shown for training in three environments (A, B and C) and for training in all seven environments. One can see that this technique makes a significant difference to the modified Perceptron, but does not really affect the lagged and the recurrent Perceptron models. In fact, the results look slightly worse. These results are confirmed by the paired t-test, which suggests that the median filtered versions of the recurrent and lagged perceptions correlate highly with the non-median filtered versions, while the chance of the results from the modified Perceptron and the modified Perceptron using median filtered input coming from the same distribution is negligible ($p = 0.01$).

6.3 Evaluating the Landmark Consistency

Having compared the quality of the neural network error curves, we now compare the consistency of landmark selection, using the LMC performance measure described in section 5.2. This was done in a very similar way to the previous sum-squared error measurements, using an average of the results for each of the testing runs.

The results of this are shown in figures 11 and 12 for the three different neural network architectures (modified Perceptron, lagged Perceptron and recurrent Perceptron) on the peak detection algorithm and the Kalman filter respectively. Again, the average case together with the best and worst cases are shown.

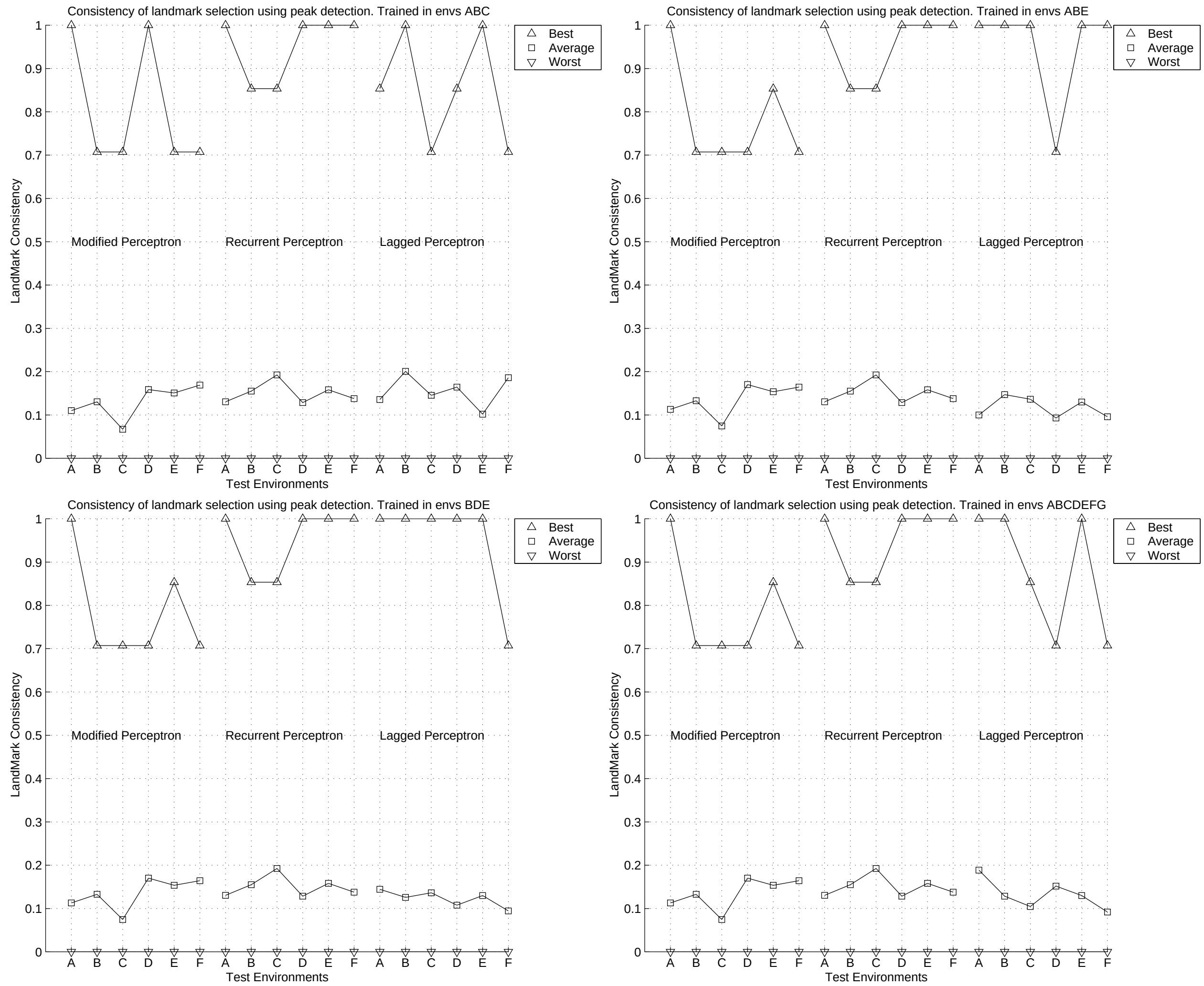

Figure 11: The results of evaluating the consistency of landmarks (LMC) for the peak detection approach. The LMC is shown for each network, tested in each of environments A to F. The four pictures show the results after training in different sequences of environments (ABC, ABE, BDE, $ABCDEFG$, respectively).

Table 3 shows the mean and standard deviation for these results. Again, these are averaged across all of the test environments.

A comparison between the two figures shows very clearly that the performance of the Kalman filter with this measure is significantly better than that of the peak detection. The paired Student t-test was used to compare the peak detection and Kalman filter algorithms for each of the neural network architectures. The results show that the probability of the LMC measures for the two algorithms coming from the same distribution, regardless of neural network used, is negligible.

Looking at figure 11, which shows the LMC measure for the peak detection, we see that while the best case performance is fairly good, in the worst case there are no matches, and the average is also very low. The performance of the algorithm does not appear to vary much between the different network architectures, nor between the different training sets. A paired t-test comparing the different network architectures shows that the performances of the lagged Perceptron and the modified Perceptron are very similar ($p = 0.01$), while the performance of the recurrent Perceptron is significantly different (in both cases, $p = 0.01$).

Figure 12 shows the results of the LMC evaluation for the Kalman filter algorithm. It can be seen that the average landmark consistency is significantly higher than it is for the peak detection, and the

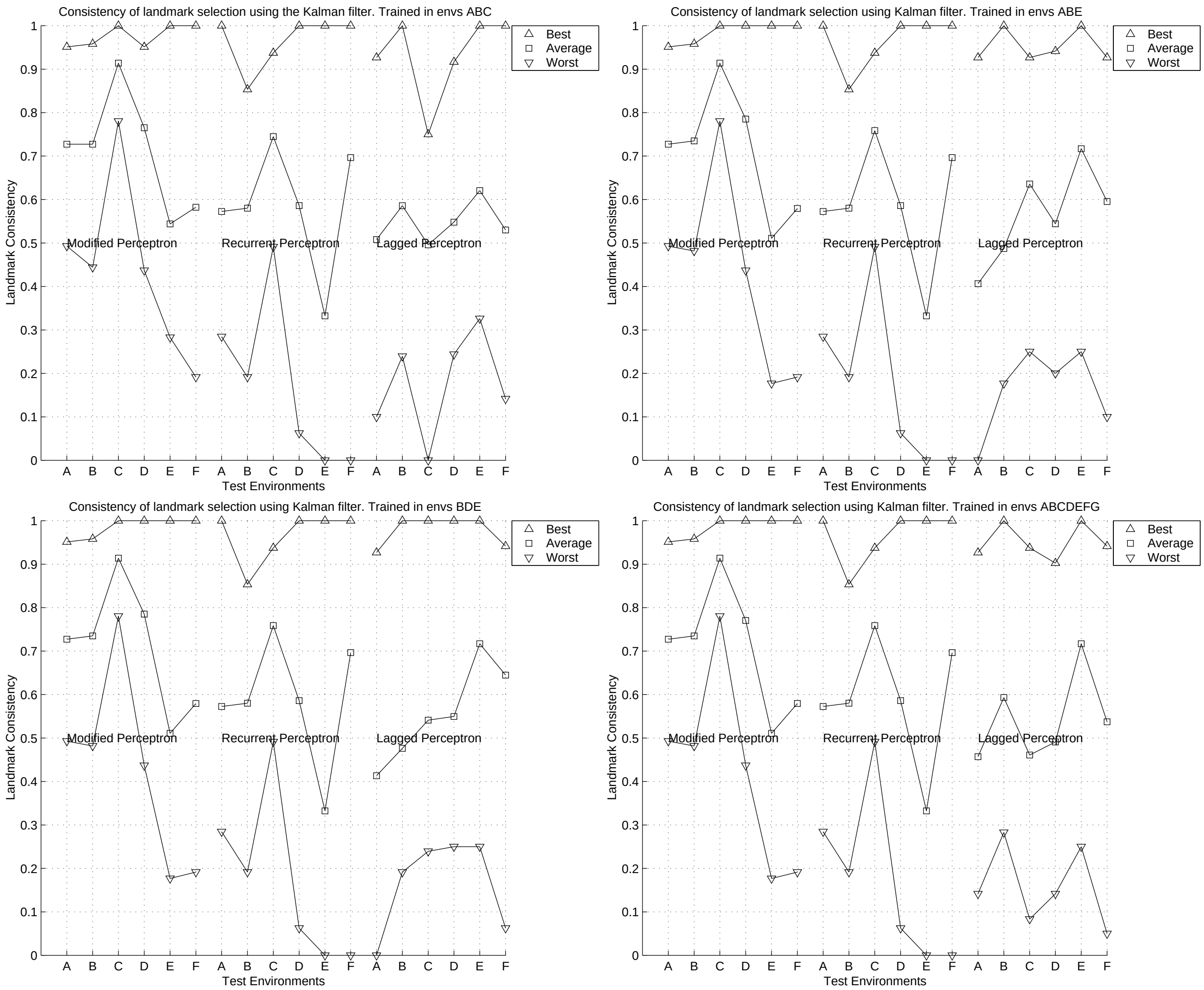

Figure 12: The results of evaluating the consistency of landmarks (LMC) for the Kalman filter approach. The LMC is shown for each network, tested in each of environments A to F. The four pictures show the results after training in different sequences of environments (ABC, ABE, BDE, $ABCDEFG$, respectively).

Algorithm	Training Environments	Peak Detection Mean ± Standard Deviation	Kalman Filter Mean ± Standard Deviation
Modified Perceptron	ABC	0.13 ± 0.04	0.71 ± 0.13
	ABE	0.14 ± 0.04	0.71 ± 0.15
	BDE	0.17 ± 0.03	0.71 ± 0.14
	$ABCDEFG$	0.15 ± 0.04	0.71 ± 0.13
Recurrent Perceptron	ABC	0.15 ± 0.04	0.59 ± 0.14
	ABE	0.14 ± 0.05	0.59 ± 0.15
	BDE	0.15 ± 0.02	0.61 ± 0.15
	$ABCDEFG$	0.15 ± 0.04	0.61 ± 0.14
Lagged Perceptron	ABC	0.15 ± 0.02	0.54 ± 0.05
	ABE	0.14 ± 0.05	0.56 ± 0.11
	BDE	0.15 ± 0.04	0.56 ± 0.11
	$ABCDEFG$	0.14 ± 0.04	0.54 ± 0.09

Table 3: The mean landmark consistency (LMC) and the standard deviation, averaged over the testing environments, for the peak detection and Kalman filter algorithms. The systems were tested using independent data collected in environments $A - G$.

	Average Number of Landmarks ± Standard Deviation	
Test Environment	Peak Detection	Kalman Filter
A	2.1 ± 0.5	4.5 ± 1.1
B	2.5 ± 1.0	5.5 ± 0.5
C	1.8 ± 0.7	3.4 ± 0.9
D	1.0 ± 0.0	5.0 ± 0.0
E	1.4 ± 0.5	4.9 ± 0.5
F	1.9 ± 0.6	3.4 ± 0.6

Table 4: The average number of landmarks chosen in each environment by the two landmark detection algorithms. The neural network which was used was the one which performed best for each of the landmark algorithms, the recurrent Perceptron for the peak detection and the modified Perceptron for the Kalman filter.

worst case has risen above 0 in virtually every case. Interestingly, the Kalman filter technique appears to work best on the Perceptron data which the sum-of-squared error method of the previous section found to be most noisy. This leads to the question of whether or not we actually need the neural network, rather than a very simple model which just predicts that the new perception is the same as the current one. However, while simpler, this system has a number of disadvantages. For instance, whenever a doorway is predicted there will be a large number of potential landmarks detected, as the various sensors detect the doorjamb, followed by the door itself and then the second doorjamb.

6.4 Number of Landmarks

Table 4 shows the average number of landmarks that are produced by each of the algorithms for different test sets. It can be seen that the Kalman filter approach finds significantly more landmarks, and that the standard deviation in the number of landmarks found is similar for both landmark detection algorithms, averaged over the different neural network architectures. The goal of this project is to reduce the number of landmarks which are needed for reliable robot navigation, but obviously there still need to be a usable number of landmarks. As the results given here show that in general the landmarks are not found perfectly, having several landmarks in an environment is a definite advantage, as it increases the probability of the robot finding some of them.

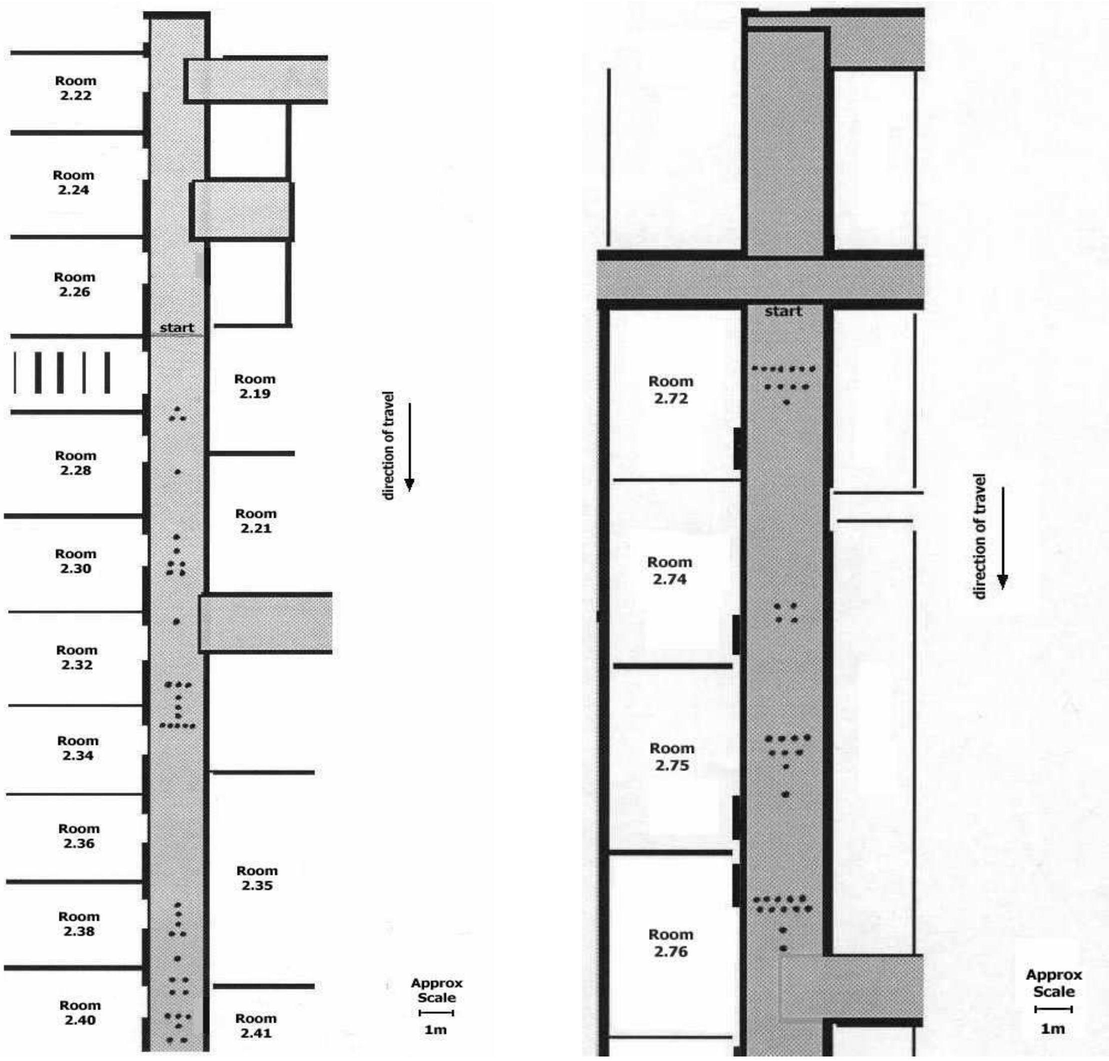

Figure 13: Figures showing the positions of the landmarks (drawn as dots) using the peak detection algorithm on two different environments.

6.5 Online Operation

In addition to the off-line operation that has been discussed in previous sections, we also want to demonstrate that the algorithms can be used online. Once the neural network has been trained the processing of the sonar data is a very simple task and does not require much computational time. It is therefore suitable for online use. We implemented a version of the algorithms which detected landmarks for use as the robot explored.

Using the peak detection algorithm operating on *FortyTwo*, we ran the robot along a number of different corridor environments. Each time the algorithm selected a landmark, the robot notified the user, who noted the position of the landmark on a floor plan. The robot travelled in each environment twelve times. The results of two of these experiments are shown in figure 13. The picture on the left demonstrates one reliable landmark, picked in ten of the twelve runs, outside room 2.32, and one which is spread across several sampling locations (outside room 2.30). The area outside room 2.38 appears as one large landmark.

The results obtained in a second environment, shown on on the right of figure 13 are crisper. There are several stretches of corridor where no landmarks are detected in any of the runs. These mark places where the perceptions of the robot are predicted accurately by the network, so that no landmarks are selected. We can also see three good clusters of landmarks, where a landmark was predicted reliably, and only one area where the prediction was less reliable.

7 Conclusions

7.1 Motivation

For a mobile agent, the ability to navigate is of fundamental importance. Over short distances, this capability of goal-oriented motion may be achieved by path integration through dead reckoning, but very quickly odometry drift errors render dead reckoning navigation systems too inaccurate for useful navigation.

The alternative is to use perceptual landmarks for navigation. Typical landmark-based navigation systems either log perceptions at regular intervals, or match perceptions with pre-supplied templates of 'desirable' landmarks. The former approach is inefficient and runs the risk of missing out clearly visible landmarks that lie between sampling points. The latter approach runs the risk of being based on landmarks that are easily visible to a human, but not to a robot, or (equally undesirable) of missing landmarks that are easily visible to a robot, but which remain unselected because they are not important for human navigation.

We believe that a suitable way to address these problems is to devise an *automatic* landmark selection mechanism that allows the robot to select suitable landmarks automatically and autonomously. This paper has described an algorithm which can be used to achieve this.

7.2 Approach

We argue that a landmark should be

1. clearly detectable and conspicuous
2. reliably and repeatedly detectable.

To identify conspicuous perceptions, we let a robot acquire a model of the usual temporal relationship between subsequent sensory perceptions. In the landmark-selection phase, this model is then used to predict the next sensory perception, based on the current sensory perception. Wherever this prediction fails, i.e., wherever the predicted perception differs markedly from that actually observed, the location is selected as a landmark on the grounds that it 'stands out' from the usual perceptions commonly observed.

To assess the consistency of landmark selection, we use two performance measures — sum-squared-error and landmark consistency, LMC.

7.3 Discussion of Results

We have investigated three different neural networks to acquire the model, and two different methods of selecting landmarks based on the prediction error made by the respective model.

Our experiments show that landmark selection is possible and consistent, using the proposed method, and that this research is promising enough to merit further investigations on an even larger scale.

The three models investigated — modified Perceptron, recurrent Perceptron and lagged Perceptron — do not differ dramatically in their prediction accuracy.

When selecting landmarks by detecting peaks in the error curve, the performance of the modified Perceptron is worst, while the recurrent Perceptron and lagged Perceptron perform practically identically (figure 9). However, with appropriate preprocessing of the raw sensor perceptions, the modified Perceptron also performs as well as the other two models (figure 10).

If the Kalman filter is used for landmark selection, there is no discernible difference in performance between the three models (figure 12).

The Kalman filter approach has another possible benefit: because the definition of an 'outlier' (currently four standard deviations) can be modified, the number of landmarks selected in a particular environment can also be influenced. This property is useful, because it facilitates the application of more stringent selection criteria in 'landmark-rich' environments, and less stringent criteria in 'landmark-poor' environments.

Our results also show that in order to train the model, it is not necessary to expose it to *all* environments that the robot may encounter during operation (figure 9).

This observation confirms our initial expectations. The robot is essentially learning a model of the temporal relationship between its own sensory perceptions, not a model of any particular world. Provided the training data is rich enough to provide most possible temporal constellations of sensory perceptions, the model acquired in one or two training environments is then applicable to *any* environment.

There is a further interesting application of the method described here. Obviously, the robot acquires a model of 'normal' perceptions against which all subsequent sensory perceptions are measured, to detect those perceptions that 'stand out'. This is a form of novelty detection. The algorithm described here can therefore be used to detect abnormalities (with respect to the robot's sensory perceptions) in the environment.

7.4 Future Work

To make predictions, the robot has to move in straight-line segments between subsequent perceptions. It is conceivable that the approach is expanded to arbitrary motions between perceptions.

For the experiments described here, the robots have sampled perceptual data in regular intervals. Clearly, the proposed algorithm is intended to select landmarks from *continuous* observations of the environment. Ultimately, a robot will always only be able to sample the environment in discrete time steps, but it is no problem to increase the sampling rate beyond that used in the experiments discussed in this paper.

Finally, we have not yet investigated how useful the selected landmarks are for actual navigation. The purpose of this research was to investigate the *consistency* of landmark selection. The construction of an entire mobile robot navigation system, incorporating the landmark selection algorithm presented here, is subject to ongoing research at the University of Manchester.

Acknowledgements

This work was supported by an Academic Research Collaboration (ARC) grant from the British Council and Deutscher Akademischer Austauschdienst (grant number ARC 878). The work at Bremen University was carried out at the Bremen Institute of Safe Systems, in the group of Bernd Krieg-Brückner. We thank them for their hospitality and assistance. We also thank Jason Fleischer, Jonathan Shapiro and Roger Hubbold for helpful discussions, and David Gelder for performing the experiments with *FortyTwo*.

References

C.M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, England, 1995. ISBN 0-19-853864-2.

Eric Bourque and Gregory Dudek. On-line construction of iconic maps. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'2000)*, pages 2310–2315, 2000.

Tom Duckett and Ulrich Nehmzow. Mobile robot self-localisation and measurement of performance in middle scale environments. *Robotics and Autonomous Systems*, 24(1–2):57–69, 1998.

Tom Duckett and Ulrich Nehmzow. Learning to predict sonar readings for mobile robot landmark selection, 1999. Internal report, University of Manchester.

R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, March 1960.

David Kortenkamp and Terry Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *Proceedings of the Twelth National Conference on Artificial Intelligence (AAAI'94)*, pages 979–984, Seattle, Washington, 1994.

Stephen Marsland and Ulrich Nehmzow. Selecting landmarks from sonar scans for mobile robot naviagation. Technical Report UMCS-00-8-1, Department of Computer Science, University of Manchester, Technical Report Series, ISSN 1361–1661, 2000. URL `ftp://ftp.cs.man.ac.uk/pub/TR/UMCS-00-8-1.ps.gz`.

Peter S. Maybeck. The Kalman filter: An introduction to concepts. In I. Cox and P. Wilfong, editors, *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, pages 193–204. Springer, Berlin, 1990.

Ulrich Nehmzow, David Gelder, and Tom Duckett. Automatic selection of landmarks for mobile robot navigation. Technical Report UMCS-00-7-1, Department of Computer Science, University of Manchester, Technical Report Series, ISSN 1361–1661, 2000. URL `ftp://ftp.cs.man.ac.uk/pub/TR/UMCS-00-7-1.ps.Z`.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, Cambridge, England, 2nd edition, 1992.

F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan, 1962.

Saul Simhon and Gregory Dudek. Selecting targets for local reference frames. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'98)*, pages 2840–2845, 1998.

Sebastian Thrun. Bayesian landmark learning for mobile robot localisation. *Machine Learning*, 33(1), 1998.

Nikos Vlassis, Yoichi Motomura, and Ben Krose. Supervised linear feature extraction for mobile robot localization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'2000)*, pages 2979–2984, 2000.

Brian Yamauchi and Randall Beer. Spatial learning for navigation in dynamic environments. *IEEE Transactions on Systems, Man and Cybernetics Section B*, 26(3), 1996. URL `http://www.aic.nrl.navy.mil/ yamauchi/papers/compressed/smc.ps.gz`.

Uwe R. Zimmer. Robust world-modelling and navigation in a real world. *Neurocomputing, Special Issue*, 13(2–4), 1996. URL `ftp://ftp.gmd.de/GMD/ai-research/Publications/1996/Zimmer.96.Robust.ps.gz`.