Available online at www.sciencedirect.com
SCIENCE DIRECT®
Artificial Intelligence 170 (2006) 59–113
Artificial Intelligence
www.elsevier.com/locate/artint

Robot introspection through learned hidden Markov models

Maria Fox [a,*], Malik Ghallab [b], Guillaume Infantes [b], Derek Long [a]

[a] *Department of Computer and Information Sciences, University of Strathclyde, 26 Richmond Street, Glasgow, G1 1XH, UK*
[b] *LAAS-CNRS, 7 Avenue du Colonel Roche, 31500 Toulouse, France*

Received 1 December 2004; accepted 6 May 2005

Available online 1 September 2005

Abstract

In this paper we describe a machine learning approach for acquiring a model of a robot behaviour from raw sensor data. We are interested in automating the acquisition of behavioural models to provide a robot with an introspective capability. We assume that the behaviour of a robot in achieving a task can be modelled as a finite stochastic state transition system.

Beginning with data recorded by a robot in the execution of a task, we use unsupervised learning techniques to estimate a hidden Markov model (HMM) that can be used both for predicting and explaining the behaviour of the robot in subsequent executions of the task. We demonstrate that it is feasible to automate the entire process of learning a high quality HMM from the data recorded by the robot during execution of its task.

The learned HMM can be used both for monitoring and controlling the behaviour of the robot. The ultimate purpose of our work is to learn models for the full set of tasks associated with a given problem domain, and to integrate these models with a generative task planner. We want to show that these models can be used successfully in controlling the execution of a plan. However, this paper does not develop the planning and control aspects of our work, focussing instead on the learning methodology and the evaluation of a learned model. The essential property of the models we seek to construct is that the most probable trajectory through a model, given the observations made by the robot, accurately diagnoses, or explains, the behaviour that the robot actually performed when making these observations. In the work reported here we consider a *navigation* task. We explain

* Corresponding author.
E-mail address: maria.fox@cis.strath.ac.uk (M. Fox).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2005.05.007

the learning process, the experimental setup and the structure of the resulting learned behavioural models. We then evaluate the extent to which explanations proposed by the learned models accord with a human observer's interpretation of the behaviour exhibited by the robot in its execution of the task.

Keywords: Stochastic learning; Hidden Markov models; Robot behaviour

1. Introduction

The goal of the work described in this paper is to automate the process of learning how a given robot executes a task in a particular class of dynamic environments. We want to learn an abstract model of the behaviour of the robot when executing its task solely on the basis of the sensed data that the robot records when performing the task. Having learned an execution model of this task we want to use the model to reliably predict and explain the behaviour of the robot carrying out that same task in any other environment belonging to the class. This paper describes how we have approached this goal in the context of an indoor navigation task, and how successful we have been in learning a reliable behavioural model.

1.1. Motivation

The work presented here illustrates that it can be advantageous to approach a complex artifact, such as an autonomous robot, not from the usual viewpoint in robotics of the *designer*, but from the *observer's* point of view. Instead of the typical engineering question of "*how do I design my robot to behave according to some specifications*", here we address the different issue of "*how do I model the observed behaviour of my robot*", ignoring, in this process, the intricacy of its design.

It may sound strange for a roboticist to engage in observing and modelling what a robot is doing, since this should be inferrable from the roboticist's own design. However, a modular design of a complex artifact develops only local models which are combined on the basis of some composition principle of these models; it seldom provides global behaviour models. The design usually relies on some reasonable assumptions about the environment and does not model explicitly a changing, open-ended environment with human interaction. Hence, a precise observation model of a robot behaviour in a varying and open environment can be essential for understanding how the robot operates within that environment.

We are proposing in this paper a machine learning approach for acquiring a particular class of behaviour models of a robot. The main motivation for this work is to build models of robot task execution that are intermediate between the high level representations used in deliberative reasoning, such as planning, and the low level representations used in sensory-motor functions. A high-level action model, such as a collection of planning operators with abstract preconditions and effects, is certainly needed in high level mission planning. However, it is of limited use in monitoring and controlling the execution of plans.

These functions need a more detailed model of how an action breaks down, depending on the environment and the context, into low-level concurrent and sequential sensory-motor primitives, and how these primitives are controlled. On the other hand, the representations used for designing and modelling sensory-motor functions are necessarily too detailed. They are far too complex to be dealt with at a planning level, or even for execution monitoring. The latter requires intermediate level models, either hand-programmed, learned, or refined through specification and learning.

Other authors have considered how intermediate level descriptions of task execution might be used for designing a robot, i.e., how the corresponding models might be encoded and exploited within a plan execution framework. We are not concerned with programming the low level control of the robot but with providing the means by which a robot can *introspect* about the development of its behaviour in the execution of a task. We rely on hidden Markov models (HMMs) [25] as the intermediate level representation of this behaviour. Since these models are built empirically, they take into account the dynamics and uncertainty of the real execution environment. The resulting behavioural models provide a way in which the controller can reason about the robot behaviour in the context of executing a task.

Our focus here is not on learning topological or metric maps for robot navigation. Others have considered this problem in depth [1–4] and shown that navigation with respect to a given environment can be dynamically improved as the robot interacts with its environment. The use of stochastic learning techniques to improve robot navigation in a given environment is therefore quite well-understood. We are concerned with learning abstract models of how a robot performs a compound task, whatever that task might be. Navigation is an example of such a compound task.

1.2. Approach

Our objective is to be able to predict and explain the robot's behaviour as it undertakes a compound task in the uncertain real world. In reality the robot passes through a number of *abstract behavioural states*, some of which can be distinguished and identified by a human observer. For example, when picking up an object in its grippers a robot might be in the state of positioning with respect to the object, approaching it, grasping it, knocking into it, lifting it, and so on.

To illustrate the kind of model we are interested in learning, Fig. 1 shows a high level state transition model of a *pickup* task (this is an artificially simplified example that was not learned from real data). Time is abstracted out of the model and it is assumed that a monitoring process tracks how often the robot revisits the same state.

It can be seen that, according to the model, the probability of knocking into the object is 0.2 when the robot is positioning itself and when it is in the approaching state, having positioned itself ready to grasp the object. The probability of looping on the positioning state is high, suggesting that the robot often fumbles to get into a good grasping position. The trajectories through this model that are actually followed by the robot might revisit the positioning state multiply often and it might be that the state of knocking into the object is entered most frequently when this is the case. Using the HMM to identify the most probable trajectory leading out of the current state provides a monitoring system with a

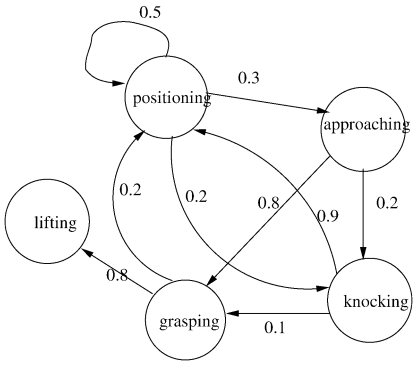

Fig. 1. The compound task of picking up an object.

powerful ability to determine the most likely outcome of the robot's current behaviour. In Section 7 we discuss how the structure of the HMM can be exploited by such a monitoring system.

The behavioural states of the model are *hidden*, because they cannot be sensed directly by the robot. The robot is equipped with noisy sensors from which it can obtain only an estimate of its state. A hidden Markov model (HMM) represents the association between these noisy sensor readings and the possible behavioural states of the system, as well as the probabilities of transitioning between pairs of states. The HMM is therefore ideally suited to our objectives. Our approach is to learn a HMM that relates the sensor readings made by the robot to the hidden real states it traverses when executing its task, in order to equip the robot with the capacity to monitor its progress during subsequent executions of the same task.

Our work makes several innovations. First, we address the problem of learning the structure as well as the parameters of the HMM, using a structural learning approach based on Kohonen network clustering. We begin with no prior knowledge about how many states the HMM will have, or what the relationship between states and observations might be. Second, we learn an HMM that is independent of the physical locations at which activity takes place. The states we are concerned with are abstractions of the behavioural states of the robot. Expectation Maximization (EM) [5] is used to estimate the transition probabilities between them based on multiple sequences of robot observations, each sequence corresponding to the observations made by the robot during an execution of the compound task.

Fig. 2 gives an overview of the whole learning process, and suggests how the resulting model might feed into high level deliberative reasoning processes. In this paper we focus on the processing and clustering of raw sensor data leading to the construction of HMMs. As we discuss in Section 7, these models represent behavioural abstractions that can be used by high level deliberative processes.

We show that it is possible to learn high quality HMMs using a fully automated approach. Although some questions remain to be answered we believe that our work constitutes an interesting step towards the acquisition of a predictive and explanatory model of robot behaviour that is grounded in its actual sensed experience in reality.

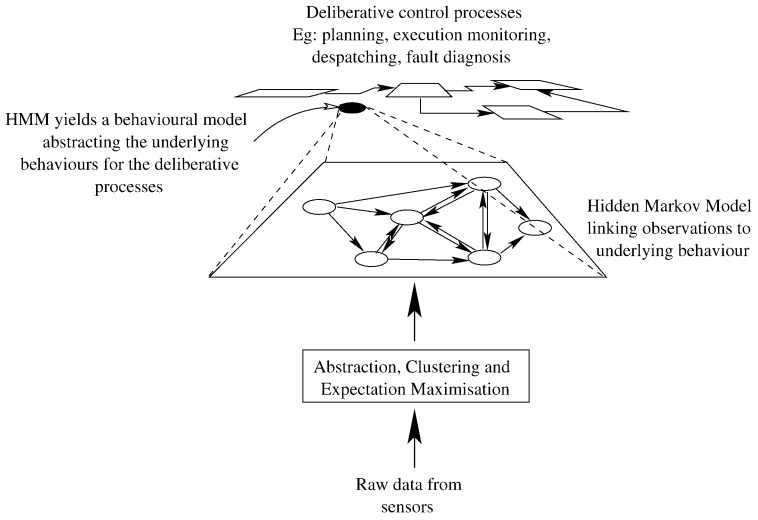

Fig. 2. Learning an HMM from the bottom up.

1.3. Related work

The work described in this paper builds on a varied literature concerned with the automated construction of stochastic behavioural models. This includes work on probabilistic plan recognition [6,7], learning topological and metric maps [1,2], learning stochastic models of human activity [8–13] and learning to recognise facial expressions [10] and gestures [11,13,14]. Previous authors have also considered the automatic classification and interpretation of sensed data [15] and the refinement of behavioural states to introduce previously unaccounted-for distinctions into a world model [2,16]. Our work therefore combines a number of established approaches in the acquisition of stochastic task models.

Koenig and Simmons [1] use EM to learn to improve a robot's ability to navigate successfully within a specific environment. Other approaches [17,18] address the problem of learning how to map and navigate an environment using active exploration strategies. The technique described by Koenig and Simmons uses Partially Observable Markov Decision Process (POMDP) models to represent the robot's understanding of the environment and its position given uncertainty about the topological structure of the environment. The accuracy of the robot's navigation is improved by using EM to reestimate the parameters of the model given navigation traces. The GROW-BW technique allows new states to be added to the model if the model fails to account for the evidence observed during a trace. Increasing the lower bound on the length of a segment of the topological map corresponds to adding states to the POMDP. The learned POMDP is therefore as accurate as possible a representation of a physical space.

Although the work is superficially related to ours, because EM is used to estimate the parameters of a stochastic model, its objectives are very different. Koenig and Simmons are specifically interested in learning to improve the navigation capability of a robot within a given environment, whilst we are interested in learning how a robot accomplishes a task,

whatever the task may be. In the work we describe in this paper we use the navigation task simply as an example of a compound task. The states of our learned model correspond to abstract behavioural states, such as *obstacle avoidance*, not to physically grounded states such as *one metre from a corridor junction*. This is a significant difference because our method is task-independent. The states are acquired automatically by means of the clustering of sensor input and their veracity is established by evaluating the predictive power of the resulting HMM.

Several authors have considered how intermediate level models might be used for describing the execution of a task. For example, RAPS [19] and Structured Reactive Controllers [20] provide the low level programs into which actions at the task-planning level decompose at the executive level of the robot architecture. RMPL [21] and TDL [22] are examples of languages that have been developed for the specification of such programs. These programs might be hand-coded or they might be acquired by learning or by interpretation of learned models. The Procedural Reasoning System (PRS) [23] is a further example of an architecture that supports the relationship between high level plans and execution.

The work done in gesture [11,13,14] and facial expressions [10] recognition is closely related to our concern. If an HMM is used to model the probabilities of a human face transitioning between different expressions, and these expressions are linked to emotional states and actions, it becomes possible to predict the most likely next action of a person based on interpretation of his facial expression. Similarly, if gestures are associated with activities a learned HMM can enable the immediate goals of a person to be predicted on the basis of his recent and current gestures. This work is similar to our own because the states of the learned HMMs are behavioural states of the subject and are not associated with the physical location of the subject.

Liao, Fox and Kautz [8] use learned models to predict human transportation behaviours. They can detect when a person's behaviour deviates from their normal pattern by evaluating the likelihood of an observed behaviour in the context of a learned model. Osentoski, Manfredi and Mahadevan [9] learn models of human behaviours in order to provide robots functioning in human environments with the capacity to predict and explain human activities. In both studies the HMM is used to predict the probability that certain activities are being undertaken at certain physical locations. The structure of the HMM is hierarchical, with the lowest level corresponding to a physical network of locations and higher levels corresponding to the activities that typically take place at these locations. Thus, the work is concerned with relating activities to physical space and its emphasis is therefore different from our own.

In our work the association between the sensor readings of the robot and its behavioural states is learned by means of Kohonen network clustering. A closely related approach in the literature is the work of Oates, Schmill and Cohen [15] in which dynamic time warping is used to cluster multivariate time series sensor data, or *experiences*. Oates et al. present an unsupervised method of clustering experiences into classifications of action outcomes enabling a robot to interpret its state in a way that accords well with human judgements. The objective of their work is to identify cluster prototypes that form the basis of an ontology of activity that can lead to the automated construction of operator models.

The way in which we use the results of the clustering phase is different and a more detailed comparison follows later in the paper.

1.4. Layout of the paper

In Section 2 we formulate the problem we are addressing, specifying the notation we will use in this paper, introducing the main methods and algorithms and defining the key terms that we will use. In Section 3 we discuss data clustering using the Kohonen network clustering approach. We describe how an initial collection of behavioural states is refined by the clustering process. We then explain the process of building an initial sensor model using the *code book* approach, and show how all of the remaining parameters of the HMM are initialised. We then review EM, describing how the initial parameters are iteratively reestimated. Section 4 describes the robot and its array of sensors, the data we collected and the class of environments we studied. We explain how the observable outputs of the robot are processed to extract the features used for clustering, and how the feature vectors we used are constructed. In Section 5 we discuss the implementation of the entire learning process, showing how the EM process was integrated with the clustering phase in our system. The EM process requires evidence to be provided, and we explain how sequences of observations are generated for this purpose.

In Section 6 we describe the evaluation strategy we have devised for determining the quality of the learned HMM in terms of its power to explain the robot's behaviour from its observations. Using the Viterbi algorithm [24] we construct the sequences of states that best explain the observation sequences, and then we compare the Viterbi sequences with what the robot did in reality. This comparison relies on a human observer's interpretation of the robot's real behaviour. We discuss the strengths and weaknesses of this approach.

Finally, in Section 7, we turn to a discussion of how the learned models can be used in the monitoring and control of robot behaviour. Although this is not the focus of the current paper we explain how the HMMs can enable a robot to predict entry into an undesired state and to take averting action in time to avoid a failure. We discuss how HMMs can be used in combination with policies and plans to support a robot in achieving high level mission goals.

2. Formal problem statement

There are three main problem components to define: the *model* of a task as a finite state transition system, the clustering of the observation space into a finite *evidence* space, and the definition of the finite *behaviour state space*. For each component we will present the assumptions that we make concerning the component and its role in the problem, the formal definition of the component and a brief introduction to the algorithm that is used to construct instances of the component. In the following section we present details of the core algorithms introduced here.

In our definitions we use n and m as index variables indicating the lengths of sequences in the context of each definition. These variables should be interpreted as locally defined within the scope of each definition.

2.1. Models and tasks

Assumption. The robot behaviour for task T can be conveniently modelled by a finite stochastic model.

Definition 1. A stochastic state transition model is a 5-tuple, $\lambda = (\Psi, \xi, \pi, \delta, \theta)$, with:

- $\Psi = \{s_1, s_2, \ldots, s_n\}$, a finite set of *states*;
- $\xi = \{e_1, e_2, \ldots, e_m\}$, a finite set of *evidence items*;
- $\pi : \Psi \to [0, 1]$, the prior probability distributions over Ψ;
- $\delta : \Psi^2 \to [0, 1]$, the transition model of λ such that $\delta_{i,j} = \text{Prob}[q_{t+1} = s_j \mid q_t = s_i]$ is the probability of transitioning from state s_i to state s_j at time t (q_t is the actual state at time t);
- $\theta : \Psi \times \xi \to [0, 1]$, the sensor model of λ such that $\theta_{i,k} = \text{Prob}[e_k \mid s_i]$ is the probability of seeing evidence e_k in state s_i.

Under the Markov assumption the state of the robot at time t depends only on its state at time $t - 1$, so that λ produces a hidden Markov model.

Definition 2. A *history* $h = \langle e_1 \ldots e_n \rangle$ is a finite sequence of evidence items.

The algorithm we are using to build the model is the well-known technique of Expectation Maximization (EM) [25], also called the Baum–Welch algorithm [26]. Given a set of histories and the initial parameters of a HMM—an initial sensor model, an initial transition model and a prior state distribution over the states in Ψ—EM iteratively reestimates the HMM parameters. On each iteration EM estimates the probability, or likelihood, of the evidence being seen given the HMM estimated so far. It then updates the model parameters to best account for the evidence. When the estimated likelihoods are no longer increasing EM converges. The probability at convergence is represented as the *maximal log likelihood*: the best local estimate possible given the evidence and the learned model. Log likelihood is used because the probability of a particular observation sequence being seen in a complex model is typically low enough to challenge the arithmetic precision of the machine. It is well known that EM has a tendency to converge on local maxima, but careful selection of the initial HMM parameters can help to mitigate this tendency.

The inputs to the EM algorithm are: a finite set of histories, $H = \{h_1, \ldots, h_n\}$, corresponding to the training data associated with n executions of task T, and an initial model, $\lambda^0 = (\Psi, \xi, \pi, \delta^0, \theta^0)$. The output is a learned stochastic model λ corresponding to a hidden Markov model describing the task T.

2.2. Clustering the observation space into evidence

Assumption. A multi-dimensional non-finite observation space can meaningfully be mapped into a finite set of evidence items.

We refer to the collection of sensor readings that can be made by the robot, at some point in time, as an *observation*. Each reading gives the value of a certain primitive feature which we call a *raw feature*, such as the heading of the robot, the speed at which it is travelling, and so on. The observation space is therefore defined by the particular collection of sensors with which the robot is equipped and its interaction, by means of these sensors, with the environment.

Definition 3. A k-dimensional *observation space* is defined as $\Phi = \gamma_1 \times \gamma_2 \times \cdots \times \gamma_k$, where:

- $\gamma_i \subseteq \Re$;
- A raw feature is defined to be a function $f_i : \mathit{robot} \times \mathit{env} \times \mathit{time} \rightarrow \gamma_i$ mapping the sensory-motor and environmental context of the robot, at a time t, to a value in the range γ_i, thus partially characterising the behaviour of the robot at some instant t in time.

Definition 4. An *observation* is a point in observation space.

Although we describe f_i as a function of the robot and its environment, we have no access to this function or control over how it produces its mapping to raw feature values. Raw feature values are determined by the low level robot control software upon which the learning pursued in this project is based and the interaction between the robot and its environment. We can sample f_i for specific values of its arguments.

Under our assumption mappings exist from the observation space Φ to a set of abstract observations ξ. We call the elements of ξ *evidence items*. We first define a *trajectory*, then explain how the construction of trajectories allows the set ξ and a mapping to be constructed by means of the *clustering* of the observation space.

Definition 5. A *trajectory* $\tau = \langle o_1, o_2, \ldots, o_n \rangle$ is a finite sequence of observations characterising a single execution of the task T.

Our intention is to discretise the non-finite observations, Φ, of the robot into a finite collection of distinct evidence items, ξ, and to determine a mapping $\mathit{cluster} : \Phi \rightarrow \xi$. This requires a process of abstraction and the combination of raw feature values across observations. A single observation is not informative enough to enable us to determine how the robot's behaviour develops over time. If we consider a single observation taken at time t the raw feature values will reveal very little about how the robot's behaviour has evolved up to that time, or will evolve after it. For example, because the robot is reacting to its environment the observation it makes at time t might record a heading several degrees away from the general direction in which the robot travelled over an interval including t. We are less interested in the precise heading at time t than in the general direction in which the robot travelled over a period of time that includes t.

In order to see how the behaviour of the robot changed over time we consider subsequences of trajectories, each containing c consecutive observations, where c is a constant chosen to ensure that the sequences represent sufficient time for interesting behaviour to

occur. We combine the raw feature values associated with observations in these sequences into *features*, then focus on how the values of the features in which we are interested vary over different sequences.

Definition 6. For a given constant c, a *feature* is an abstraction of raw feature values, obtained by combining some subset of the raw features drawn from each of c consecutive observations.

The combinations performed in Definition 6 are typical filtering and smoothing operations used in signal processing. Using features we construct feature vectors from the trajectories in our data set.

Definition 7. A *feature vector* $\vec{f_i}$ is an m-dimensional vector of feature values. The feature values are obtained from a sub-sequence of a fixed number of consecutive observations, starting at observation i, in a trajectory.

The m-dimensional feature vectors are constructed from raw features in an observation space that is k-dimensional where, in general, $m \leqslant k$ depending on the ways in which the raw features are combined in the construction of the features.

The feature vectors are constructed in the following way. For each trajectory we take all possible consecutive sequences of a fixed number of observations using a typical *sliding window* approach as shown in Fig. 3.

We do not allow feature vectors to cross the boundaries between trajectories. This helps the system to learn that the robot never transitions out of the state in which it has reached its goal into any other state.

Before clustering we normalise the feature vectors to ensure that variation in vector magnitude does not distort the clustering results. We also normalise each field of the feature vectors by expressing each value in terms of the number of standard deviations from the

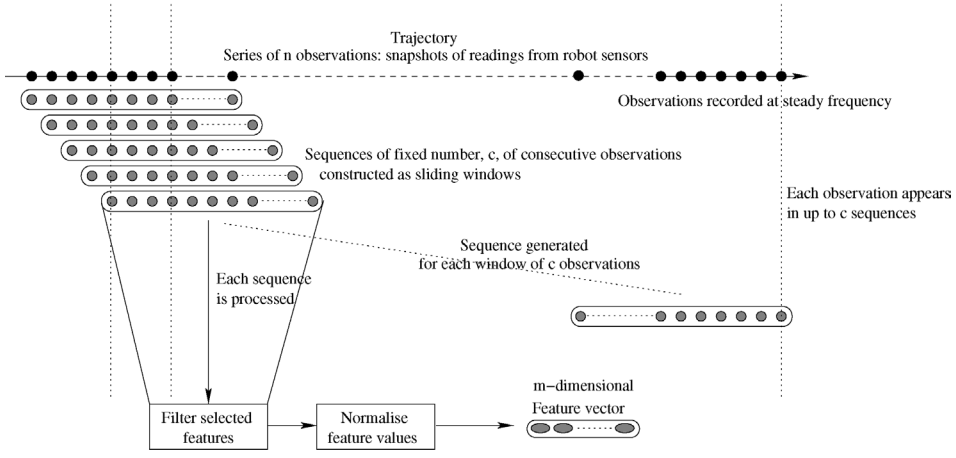

Fig. 3. Sliding window construction of feature vectors.

mean value for that field. This ensures that gross differences in the ranges of values do not get interpreted as magnitude differences by the clusterer.

The algorithm we use for clustering the observation space is Kohonen network clustering. The Kohonen network performs an unsupervised projection of multi-dimensional data onto a smaller dimensional space, resulting in the identification of a cluster landscape in this smaller dimensional space.

We chose to use the Kohonen self-organising network because it gives us the freedom to avoid specifying the number of clusters in advance. We first train the network and then apply a cluster selection function to the landscape to identify the most significant clusters. Thus, although the size of the network places an upper bound on the number of clusters that can be found, there is no need to predetermine how many clusters the data set contains. In vector quantisation approaches [27], such as K-means clustering, the user must supply the number of means, K, which determines the number of clusters that will be found. Similarly, in stochastic clustering using techniques such as EM, the user must supply the number of Gaussians to use in a mixture, which determines the number of clusters that will be learned. In our application it is important that the number of evidence items be determined autonomously from the structure of the observation data, since we do not wish to impose any prior judgements on what observations the robot might be making. Furthermore, the self-organising network has the useful property that clusters that are close together in the network map to concepts that are close in reality. We exploit this property by using scalar product operations to identify relationships between evidence items and behavioural states. We describe this process in Section 3.2.

The input to the clustering process is a finite set of feature vectors constructed from the trajectories. The outputs are the set of evidence items ξ and the mapping $cluster : \Phi \rightarrow \xi$. Using the *cluster* mapping we can construct the set of histories H.

2.3. Defining the state space Ψ

Assumption. It is possible to determine *a priori* a collection of behavioural states associated with a task T.

We distinguish between states that are unambiguously *visible* to the observer, such as s_0, the starting state, s_g, the finishing state and s_f, failure states, and those that must be identified subjectively, such as *hesitating*. These we denote the *subjective states*. The refinement process replaces the subjective states (and, optionally, the visible states), with other hidden states, unknown to the human observer.

A human observer can label observations while the robot is performing T. The labels are associated with the observations as they occur in real time. The labelling indicates the association between an observation and a behavioural state, as perceived by the human observer. The set of labels therefore corresponds to the *a priori* state set. We call the set of labels used by the human observer L.

Given L we can define a *partial labelling* of trajectories by the human operator.

Definition 8. A *partial labelling* maps a trajectory $\tau = \langle o_1, o_2, \ldots, o_n \rangle$ to a labelled trajectory $\tau' = \langle (o_1, l_1), (o_2, l_2), \ldots, (o_n, l_n) \rangle$, where:

- $o_i \in \Phi$;
- $l_i \in L \cup \{nomark\}$, where *nomark* is the label applied to an otherwise unlabelled observation.

We can identify the set of states Ψ by refining the label set, L, using the labelled trajectories, the set of evidence items ξ and the mapping $cluster : \Phi \to \xi$. The algorithm we use, which we call *state splitting*, is described in Section 3.2. It works by finding the maximal cliques in a graph in which the nodes correspond to evidence items in ξ. A separate graph is constructed for each of the state labels in L. The structure of the graph is determined by the *cluster* mapping. An edge is constructed between nodes e_i and e_j if $\cos^{-1}(e_i \cdot e_j) \leqslant \rho$ where ρ is a constant threshold angle between vectors in the feature vector space, ξ. Each maximal clique, corresponding to a subset of evidence items in ξ, is interpreted as a state in Ψ. The elements of Ψ are substates of the label set $L \cup \{nomark\}$.

The inputs to the maximal clique finding algorithm are: the set of labels L, a set of partially labelled trajectories, the set of evidence items ξ, and the mapping $cluster : \Phi \to \xi$. The output is a set Ψ of states, which we take to be the state space of the task T.

3. The core algorithms

We now describe the three main algorithmic components of the system in more detail, showing how they construct the components described in Section 2. We present these algorithms and components in a way that is independent of the specific task, environment and robot platform that we considered. Our objective is to emphasise the generality of the approach we have taken. In the next section we explain how the data we used was collected and prepared for presentation to the system.

Fig. 4 depicts the entire process from data collection to the output of a learned hidden Markov model representing the behavioural transitions of the robot in its execution of the navigation task.

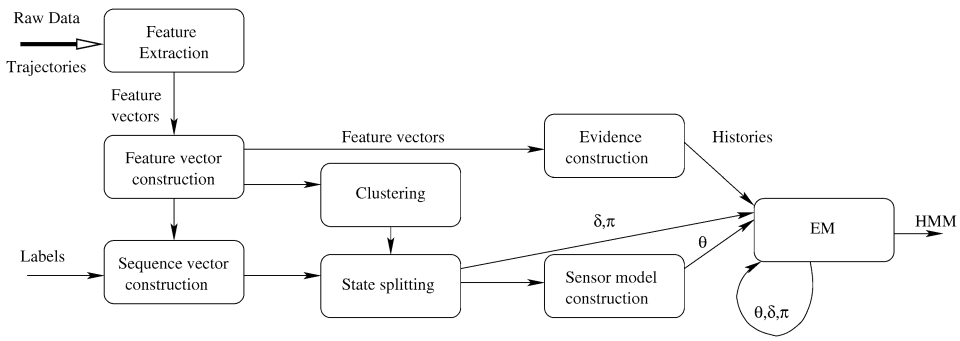

Fig. 4. Learning an HMM from raw sensor data. The bold arrows show the input to and output from the entire learning process.

3.1. Kohonen network clustering

As stated in Section 2.2, the input to the clustering process is a set of feature vectors constructed by smoothing trajectories over intervals of time. The outputs are the set of evidence items, ξ and the mapping $cluster : \Phi \rightarrow \xi$.

3.1.1. The clustering process

We performed clustering using a two-dimensional self-organising map, or Kohonen network [28]. The Kohonen network identifies patterns in feature vector data in a way that is independent of human influence. The number of clusters found depends purely on the form of the data itself and the parameters of the network. The parameters are the dimension of the network (we used a square grid), the learning rate, the neighbourhood size and the random number seed used to initialise the network vectors. This independence is important because we have no way of deciding *a priori* how many observations the raw data contains or what their relationship to one another might be.

Kohonen clustering performs a projection of n-dimensional data onto a smaller, k-dimensional, space, where k is less than n and can be determined by the user. We use $k = 2$, so we are projecting the multi-dimensional structure of our data onto a 2-dimensional space. Within this framework the dimension of the network affects how many clusters are found and how they inter-relate. The dimensions of the network should be at least 500 times smaller than the size of the data set [28] to allow for enough space for clusters to be distinguished, but not so much that they begin to degenerate into noise. Our data set consists of about 15,000 feature vectors so we experimented with dimensions varying between 15 and 45. Increasing beyond a dimension of about 35 seems to increase the amount of noise in the cluster landscape, which has a negative effect on the quality of the learned HMM. Using a dimension below about 20 causes clusters to combine and reduces the level of discrimination, again resulting in a negative effect on learning. Networks of dimension between 25 and 30 seems to give the best results for our data set, as we demonstrate in Section 6 and Appendix B.

The map is initialised with random unit vectors of appropriate dimension. We initialised the network using random vectors that cover the network adequately (we insist that all of the initial vectors must be pairwise separated by at least d degrees, where d is a constant chosen depending on the size of the network). This is to reduce the effects of initial bias in the network. Initial bias is a widely recognized problem in the use of clustering algorithms. All our results are presented as averages over 20 random number seeds, as discussed in Section 6.

The network is trained by presenting each of the feature vectors in turn and aligning the network vectors to the feature vectors to which they are closest. Scalar multiplication is used to determine closeness. Alignment is performed by adding to the network vector a proportion of the sequence vector as determined by the learning rate. A neighbourhood value determines the neighbourhood of network vectors that is influenced by the input feature vector. We implemented a neighbourhood decay rate as a negative exponential function. The effect that this has is to reduce the impact of training vectors over time. Using this function we can iterate over the training data many times without over-learning. We also used a learning rate decay, in the form of MacQueen's averaging law [29]. Thus, in a way

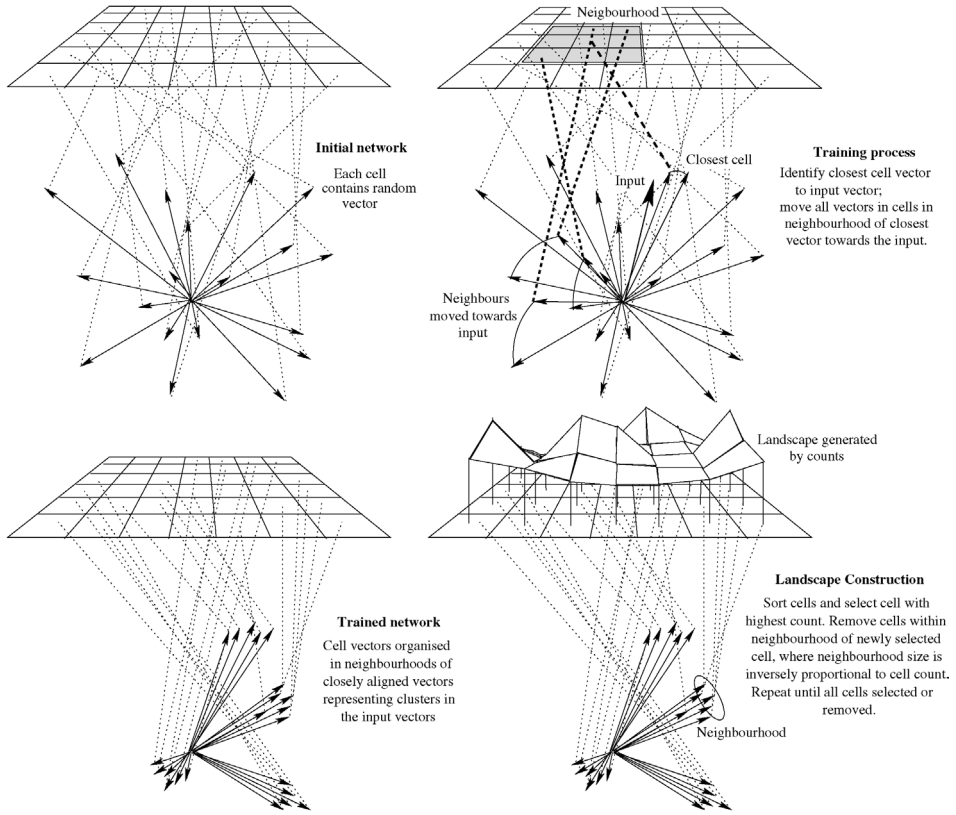

Fig. 5. Training the Kohonen network.

determined by the learning rate and neighbourhood value chosen, the 2-dimensional space partitions into regions.

The training set is presented 100 times to assist convergence of the training process. After the first 50 iterations we shuffle the order in which feature vectors are presented to the network, in order to reduce the extent to which the resulting cluster landscape is sensitive to the order of presentation. We experimented with more frequent shuffling and noticed that it slightly reduces the sensitivity to ordering of presentation but that it results in greater sensitivity to the unequal distribution of the different robot behaviours in the data set. Each trajectory produces fewer examples of the smoothed observations associated with the starting and finishing behaviours of the robot, than examples of observations associated with the intermediate behaviours. Experiments showed that shuffling more frequently led to the network failing to distinguish the starting and finishing observations from observations associated with the intermediate behaviours.

After training the 2-dimensional space can be mapped to a vector space of dim^2 vectors, where *dim* is the dimension of the network. In order to identify the clusters in this vector space we apply an cluster selection strategy to the network which draws the vectors together around the highest peaks. Our first attempt at such a strategy counted, given a cell

$\langle i, j \rangle$, the number of cells that were within a fixed radius (we used 0.085) of $\langle i, j \rangle$. This count was used to measure the influence of $\langle i, j \rangle$ over the whole network. We then used a hill-climbing strategy to associate plateau cells with the first closest peak found.

There are several weaknesses associated with this approach. The first is that, using this strategy, the composition of the peaks ends up very sensitive to the noisiness of the cells in the network. We noticed that, with different random initialisations, we got very different cluster landscapes. A cell might be pulled one way or the other depending on random factors, so that a small change in the initialisation of the network could lead to huge differences in the cluster landscape. Large variations make the later learning results highly dependent on arbitrarily chosen random numbers.

Another weakness is that the cells exerting the most influence in these terms over the network are not necessarily the cells that attracted most of the input during training. Using this method we could end up throwing out the clusters we are really interested in favour of ones that attracted little input and are not good indicators of the behaviour of the robot. Further, by associating the cells in a plateau with the nearest peak we caused the network to distort, sometimes very badly in the cases where there are large plateaus. A better approach seems to be to restrict the amount of draw that one cell can have over another, and thereby spread the clusters more evenly over the landscape.

To address these problems we developed a different cluster selection strategy which uses the number of inputs attracted to each cell as a way of identifying the cluster landscape. The cells that attracted the most inputs we take to be the highest peaks in the landscape. Given that the cluster landscape is intended to represent the structure in the data set we decided that a cell that attracts very few inputs is unlikely to be interesting, so we focus our attention on the high peaks. To achieve this focus we associate a varying neighbourhood size with the peaks in the network, considering the peaks in descending height order. This neighbourhood is different from the learning neighbourhood used during training.

We first order the peaks then, choosing the largest first, remove from the network all of the cells in its neighbourhood. The size of the neighbourhood is determined as $\lceil \frac{H}{C} \rceil$, where H is the number of inputs attracted to the highest overall peak in the network and C is the number of inputs attracted to the current peak. This value is used as a radius around the peak cell. The process is repeated for the next highest peak until no cells remain to be considered.

Let C be the current peak and H be the height of the highest peak in the network. Clearly, if $\frac{H}{C}$ is large for most values of C then we risk losing the interesting structure in the network. This is obviously undesirable, resulting in a small collection of clusters that is unlikely to be discriminating. We require the value of $\frac{H}{C}$ to have a slow, smooth gradient over a sufficiently large collection of discriminating clusters. For the size of our data set *sufficiently large* means tens of clusters. To achieve such a gradient we require the ratio of H to C to be small for tens of Cs. By examining the cluster landscapes constructed from our data set we confirmed that this requirement is satisfied. Fig. 6 gives an example of a typical cluster landscape generated using a network of size 30.

We have found that using this strategy we improve the clustering stability by reducing sensitivity to the noisiness of the randomly generated vectors. Peaks still move around in the network because of the random initialisation, but this is to be expected. The experiments

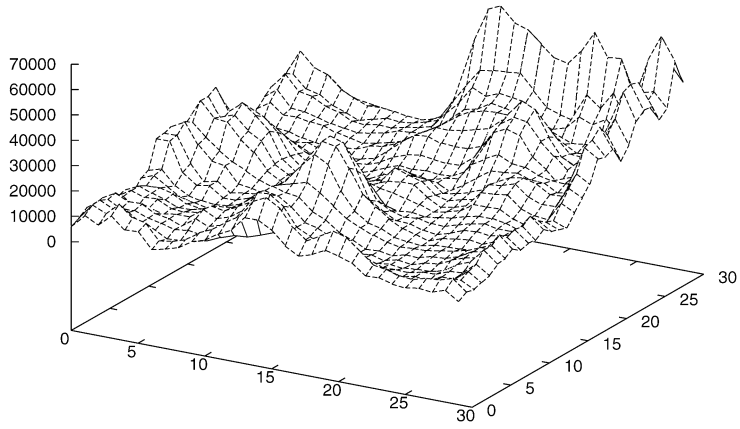

Fig. 6. A clustering landscape obtained using a 30×30 Kohonen network.

presented in Section 6.4 show that we achieve a high degree of stability in the clustering results across different random number initialisations.

Following training the feature vectors are reintroduced to the network for classification. During classification, each input sequence vector is associated with the peak vector to which it is closest according to a scalar multiplication comparison between the feature vector and each peak vector. The cells in the network that correspond to the peaks contain vectors that characterize the evidence items found by the clustering process. These are the elements of the set ξ and correspond to the evidence items that can be observed by the robot as it executes its task. We refer to these vectors as *characteristic vectors*.

Our clustering approach is related to that of Oates et al. [15] who considered the problem of clustering the experiences of a robot into qualitatively different action outcomes. Their *cluster prototypes*, which are closely related to our characteristic vectors, constitute an *ontology* of activity. It is intended that they correspond to the qualitatively different states in which a robot can find itself, following the execution of an action, and that they provide the basis for automating the description of actions at the task-planning level. By contrast, our characteristic vectors are interpreted as high level observations, or evidence items, associated with states at the intermediate level of description rather than at the task-planning level. As we will see, observations contribute to the identification of states, at this intermediate level, which might have no interpretation for the human observer but which may be critical in accurately modelling the behaviour of the robot with respect to its task.

At the end of the classification phase all of the feature vectors in our data set have been classified with one of the characteristic vectors in the network. This puts us in a position to construct an observation code book.

3.1.2. Constructing the sensor model

A code book [27] is a mapping from input values to a finite collection of observation codes. To build our code book it is necessary to associate the characteristic vectors with the labels in $L \cup \{nomark\}$. To facilitate this we annotate each feature vector with the label associated with the last observation in the sliding window from which the feature vector

was constructed. If there are no labelled observations in this sliding window the feature vector is labelled *no mark*.

The association of a feature vector with a label results in a new structure which we call a *sequence vector*. The structure of a sequence vector is defined in Definition 9.

Definition 9. A *sequence vector* $sv = (\vec{f}_i, l)$ is an m-dimensional feature vector associated with a label, l, from the set L, taken from the last labelled observation in the sub-sequence of the partially labelled trajectory from which $\vec{f}_i$ was constructed. If there are no labels in the subsequence then l is the *no mark* label.

In the construction of our code book, the input values are sequence vectors, defined in Definition 9, and the characteristic vectors identified during the clustering phase are used as the codes. The mapping is defined by the classification behaviour of the Kohonen network.

Definition 10. Given a trajectory $t = \langle o_1, \ldots, o_n \rangle$, the *ladder* l_t is the sequence $\langle \vec{f}_1, \ldots, \vec{f}_n \rangle$ of sequence vectors constructed from trajectory t.

The sequence vector construction phase defines a mapping from trajectories to *ladders*, defined in Definition 10, so-called because of the way that the sequence vectors overlap in a sliding window, as shown in Fig. 3.

We can now construct the association between evidence items in ξ and the labels in $L \cup \{nomark\}$ by counting the number of sequence vectors carrying each label, the feature vectors of which were classified with each evidence item. This association can be turned into a probabilistic observation function in the following way. Let $s_0, \ldots, s_n$ be the behavioural states labelled by L and $e_0, \ldots, e_m$ be the evidence items. We interpret the number of associations in a given pair (s_i, e_j) as a proportion, so that the probability of seeing evidence e_j in state s_i can be easily calculated. Let $V_{j_{s_i}}$ be the set of sequence vectors associated with evidence item e_j that were labelled with s_i, and V_{s_i} be the set of sequence vectors labelled s_i. Now the probability of seeing evidence e_j in state s_i is

$$\theta_i(j) = \frac{|V_{j_{s_i}}|}{|V_{s_i}|}.$$

The resulting function can be interpreted as a sensor model specifying the probability of seeing each evidence item given each state. The "sensor" is the compound sensor capable of observing the evidence items found by the clusterer. This means that when subsequently using the model the robot's raw sensor data can be processed by the construction of sequence vectors and their classification by means of $cluster : \Phi \to \xi$.

In Section 6 we present results showing the quality of HMMs learned on the basis of sensor models constructed in this way and not further refined by state splitting. As can be seen from Fig. 19, the quality of the HMMs learned on this basis is often poor. We hypothesised that the states identified by the human observer might not in fact be the states that are most important for distinguishing between the behaviours of the robot, and that better results might be obtained by sub-dividing the human-observed labels. The labels in $L \cup \{nomark\}$ abstract out a great deal of potentially important variation in behaviour, including the transitionary behaviour that the robot exhibits as it passes from one state

to another. To explore this hypothesis we developed the state splitting strategy, briefly described in Section 2.3, which decomposes each of the original labels around the groups of evidence items that are most strongly associated with these states according to the code book sensor model constructed as above.

3.2. Maximal cliques

As stated in Section 2.3, the inputs to the maximal clique finding algorithm are the set of labels L, a set of partially labelled trajectories, the set of evidence items and $cluster : \Phi \rightarrow \xi$. The output is the set of states Ψ.

In the code book multiple evidence items can be associated with the same behavioural state. This occurs because evidence items are not perfect discriminators between states. Sometimes, the characteristic vectors of these evidence items are separated in the vector space by significantly large angles. When these angles exceed 30 or 40 degrees it seems plausible that the association of these clearly different evidence items with the same behavioural state might indicate that a decomposition of that behavioural state into sub-states is possible.

The idea of state-splitting around distant groups of characteristic vectors is illustrated in Figs. 7 and 8. The procedure *refine*θ, in Fig. 7, begins by constructing, for each label $s \in L$, a graph in which the nodes are the characteristic vectors of the evidence items associated with that label in the code book θ. The edges in the graph are the angles in vector space between the evidence items at the two end-points. If two vectors are less than a pre-determined threshold apart—for example, 40 degrees—an edge between their corresponding nodes is added to the graph and the maximal cliques remaining in the graph are found. These steps are illustrated in lines 6 to 15 of the *constructGraph* procedure.

The maximal cliques contain all those evidence items within 40 degrees of one another. Each maximal clique is a subset of the characteristic vectors associated with the original label, suggesting a substate of the behavioural state corresponding to that label. The procedure *refine*θ shows how finding the maximal cliques leads to the construction of a refined sensor model.

The sensor model, θ^0, is constructed from the code book using the identified substates. We want to replace the original behavioural states labelled by $L \cup \{nomark\}$ with their substates and to share out the association between an evidence item and a label amongst all of the substates of that label. Thus: if evidence item e had a k% association with label s, and state s has p sub-states, the quantity k% has to be shared out between the p substates. This is not just a case of dividing the k% into p equal parts—the sharing has to be done in a way that reflects the proximity of each substate to the evidence item e. To do this we need to identify the centre of mass of each sub-state and measure the distance from e to each of these centres of mass. We obtain the average of the characteristic vectors in a sub-state to obtain the centre of mass of that sub-state. We then take the scalar product of the resulting vector and the characteristic vector of evidence item e to obtain the proximity of e to the substate. Finally, each substate is given a proportion of the association between e and the label, depending on its proximity to e. This calculation is shown on line 39 of procedure *refine*θ. Fig. 8 shows how this sharing is achieved.

```
1:  Procedure: constructGraph(θ,s,ξ)
2:  Input: code book θ, label s, evidence items ξ
3:  Output: graph structure G
4:
5:  initialise graph G
6:  for all cluster c in ξ do
7:     if assoc(θ, s, c) > 0 then
8:        add node for c to G
9:     end if
10: end for
11: for all (cluster) node i in G do
12:    for all (cluster) node j in G do
13:       if angle(i,j) < THRESHOLD then
14:          add edge (i,j) to G
15:       end if
16:    end for
17: end for
18: return G
19:
20: Procedure: refineθ(θ,L,ξ)
21: Input: code book θ, labels L, evidence items ξ
22: Output: sensor model θ^0
23:
24: initialise sensor model θ^0
25: for all label s in L do
26:    G = constructGraph(θ,s,ξ)
27:    {First identify the maximal cliques in G}
28:    Cs = maxCliques(G)
29:    initialise 2d array of doubles, ds
30:    for all cliques clq in Cs do
31:       {Find the mean of clusters representing nodes in clq}
32:       avC = computeAverage(clq)
33:       for all characteristic vectors c in ξ do
34: {Record distance between centre of clique and characteristic vector c}
35:          ds[clq][c] = scalarProduct(avC,c)
36:       end for
37:       normalise ds[clq]
38:       for all cluster c in ξ do
39:          θ^0[clq][c] = assoc(θ,s,c)/ds[clq][c]
40:       end for
41:    end for
42: end for
43: return θ^0
```

Fig. 7. Pseudo code showing the state splitting procedure. For both routines, $assoc(\theta, s, c)$ is the association in the code book θ between label s and evidence item c.

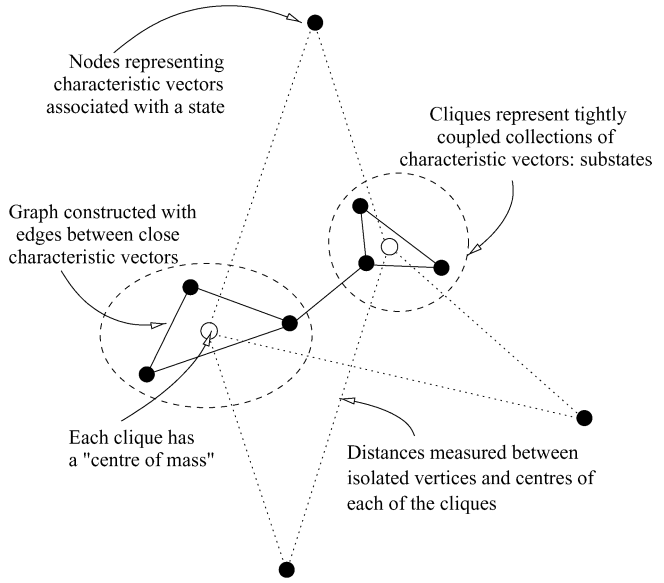

Fig. 8. The state splitting procedure.

As a result of the state splitting process the code book is rewritten in terms of the substates found. The number of evidence items does not change as a result of state splitting, but the number of states increases and is determined by the structure of the vector space following clustering. Interestingly, the states in the refined set have no interpretation for the human other than that they were obtained by decomposition of an original set of human-observed labels. Nevertheless, some of the new states might represent interesting transitionary states that are important for learning a good state transition function and can therefore improve the results obtained from the EM phase.

The state collection that results from the refinement of the initial state labels is the state set Ψ, and the sensor model, constructed using the relationship between Ψ and ξ, is the function θ^0. The relationship defines a function, $ab : \Psi \to L$, which maps states in Ψ to labels in the initial collection (ab indicates an abstraction step). We also define a function $ev : \Psi \to \mathbb{P}\xi$ which, given a state in Ψ produces the set of evidence items that constitute it. The following property holds:

$$\forall s_1, s_2 \in \Psi \cdot ab(s_1) = ab(s_2) \Rightarrow ev(s_1) \neq ev(s_2)$$

which means that, if two states map by ab to the same label, they will not contain the same evidence items. We can also identify a mapping from labels to sets of substates which, given a label produces the set of substates in its decomposition. We call this mapping $refine : L \cup \{nomark\} \to \mathbb{P}\Psi$.

It must be noted that two different substates in Ψ might be composed of exactly the same evidence items. Their association with different labels distinguishes them. However, the fact that two labels contain substates composed of identical evidence items can be taken to indicate a sharing of content between the two labels. Part of the power of the state-splitting

procedure resides in its ability to identify the shared sub-structure of abstract states, as well as the characteristics that distinguish them. We explain in Section 6 how the recognition of shared sub-structure can be exploited in the evaluation of the learned HMM.

The idea of decomposing and augmenting the states of a HMM has been considered by other authors [1,16]. In particular, Koenig and Simmons' GROW-BW algorithm allows new states to be added to a HMM if they are needed to account for observations made by a navigating robot. Chrisman [16] shows how dynamic partitioning of the state space of the model can overcome the problem of *perceptual aliasing* that occurs when a model contains too few states to discriminate between different observations. Stolcke and Omohundro [30] show how states can be dynamically *merged* to generalise a HMM. In these works the HMM starts with a collection of states that is determined *a priori* and is known to be inadequate to account for the observations of the system. State splitting and merging is applied during the learning process to increase the adequacy of the state set as observations are made.

By contrast, we propose a *static* state splitting strategy to be performed prior to the EM learning process. Its purpose is to increase the information content of λ^0 and thereby improve the quality of the learned model. Indeed, the results we present in Fig. 19, Section 6, demonstrate that the quality of models learned after state splitting is significantly higher than is obtained when state splitting is not used.

Once the state set of the HMM is decided it is never changed—only the next-state and observation probability distributions are affected by reestimation. The states to which the splitting algorithm is applied are the labels in $L \cup \{nomark\}$. Splitting allows these states to be refined so that transitionary states emerge and structure is made accessible that was not apparent to the human observer. It is intended that our state splitting algorithm identify a complete (with respect to the available sensors) set of the hidden states that accounts for the behaviour of the robot with respect to its task.

3.3. Expectation maximisation

We require a way to reestimate the parameters of the HMM, and we follow the work of Dempster et al. [25] in using the EM algorithm to perform this reestimation. Our implementation closely follows the presentation of HMM reestimation given in Rabiner's tutorial [5]. In this section of the paper we focus on the issues that arose for us in using EM to perform the reestimation of our initial HMM. These issues are: the initialisation of the HMM parameters and their effects on the results obtained; the need for scaling and the way in which scaling is performed when multiple histories are used in reestimation and, finally, the use of the learned HMM to diagnose the state of the system from a given history. In order to be self-contained, and to clarify our contribution, we summarise the main aspects of the EM technique.

In an EM implementation of reestimation there are two key steps: the E step, which is the calculation of the maximum likelihood of seeing the evidence given the model so far, and the M step, which is the process of updating the model to maximize the probability of seeing the evidence. The E step is performed using the so-called *forward-backward algorithm*, originally described in [31,32], and very clearly presented by Rabiner.

The M step, in which the transition and sensor model components of the HMM are updated, is affected by the *scaling* of the values generated by the forward-backward algorithm. As Rabiner discusses, scaling is necessary in the E step to avoid underflow. Without scaling, underflow occurs because the probability of seeing a long sequence of evidence is very small, so as the history lengths grow the E step calculations tend to zero. It is necessary to demonstrate that the scaled values do not change the interpretation of the update operations. This is straightforward to show when a single history is used for learning, but more subtle when multiple histories are used. In the work we describe in this paper, we used multiple histories because our data set contains multiple separate and independent trajectories. In Appendix A we discuss how we implemented the scaling mechanism following Rabiner's presentation. In this section, we present the core components of the E and M steps, showing how scaling is managed in the case of multiple histories.

3.3.1. Basic framework

We begin by providing here some definitions from Rabiner's tutorial that are necessary for our presentation. The forward and backward variables are defined below. The M step of the EM procedure, which performs the updating of the model, is defined in terms of the forward and backward variables. Definitions 11, 12, 13, 14 and Eqs. (1) and (2) are taken from Rabiner's paper.

Definition 11. Given a history $h = \langle e_1, e_2, \ldots, e_T \rangle$, a collection of states Ψ and a model $\lambda = (\Psi, \xi, \pi, \delta, \theta)$, the *forward variable* $\alpha_t(i)$ is defined to be the probability of being in state s_i at time t, having seen the first t elements of h, given the model λ. This is formalised as:

$$\alpha_t(i) = P(e_1 \ldots e_t, q_t = s_i \mid \lambda).$$

The forward variable is constructed recursively as follows:
Initialisation:

$$\alpha_1(i) = \pi_i . O_i(e_1), \quad 1 \leqslant i \leqslant N.$$

Induction:

$$\alpha_{t+1}(j) = \sum_{i=t}^{N} \alpha_t(i)\delta(i, j)\theta_j(e_{t+1}), \quad 1 \leqslant t \leqslant T-1,\ 1 \leqslant j \leqslant N.$$

Termination:

$$P(h \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

Definition 12. Given a history $h = \langle e_1, e_2, \ldots, e_T \rangle$, a collection of states Ψ and a model $\lambda = (\Psi, \xi, \pi, \delta, \theta)$, the *backward variable* $\beta_t(i)$ is defined to be the probability of seeing the last $T - t$ elements of h, given that the state of the system at time t is s_i and given the model λ. This is formalised as

$$\beta_t(i) = P(e_{t+1} \ldots e_T \mid q_t = s_i, \lambda).$$

The recursive construction of the backward variable is as follows:

Initialisation:

$$\beta_T(i) = 1, \quad 1 \leqslant i \leqslant N.$$

Induction:

$$\beta_t(i) = \sum_{j=1}^{N} \delta(i, j)\theta_j(e_{t+1})\beta_{t+1}(j), \quad t = T-1, T-2, \ldots, 1,\ 1 \leqslant i \leqslant N.$$

With these variables we can now define the transition model and sensor model update components of the M step. The prior probability distribution, π, is not reestimated if an unambiguous initial state can be identified for which the probability is 1. We assume that this is the case, and explain why in Section 3.3.3. We begin with the basic transition model update. In the following, the primed notation $\delta'(i, j)$ and $\theta'_j(k)$ denotes the updated values of $\delta(i, j)$ and $\theta_j(k)$ respectively.

Definition 13. The transition model component δ of λ is updated according to the following equation:

$$\delta'(i, j) = \frac{\sum_{t=1}^{T-1} \alpha_t(i)\delta(i, j)\theta_j(e_{t+1})\beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)}.$$

Definition 13 specifies that the (i, j)th element of δ' is given by the expected frequency of transitions from state i to state j, divided by the expected frequency of state i. The sensor model can be updated according to a similar rule:

Definition 14. The sensor model component θ of λ is updated by

$$\theta'_j(k) = \frac{\sum_{\substack{t=1 \\ s.t. e_t = k}}^{T} \alpha_t(j)\beta_t(j)}{\sum_{t=1}^{T} \alpha_t(j)\beta_t(j)}.$$

Definition 14 states that the probability of observing evidence k while in state j is given by the expected frequency of being in state j and observing evidence k, divided by the expected frequency of being in state j.

We now turn to the scaling issue and its effect on these update equations. The tth forward scaling term can be defined as the likelihood of seeing the first t elements of the history and being in state i. This is expressed as

$$C_t = \prod_{v=1}^{t} c_v,$$

where c_v is the normalisation term:

$$\frac{1}{\sum_{i=1}^{N} \alpha_v(i)}.$$

The $(t+1)$th backward scaling term can be defined as

$$D_{t+1} = \prod_{v=t+1}^{T} c_v.$$

The normalisation term c_v is calculated during the E step. The update equations defining $\delta'(i,j)$ and $\theta'_j(k)$ can be rewritten to incorporate these scaling terms in the M step. Eq. (1) shows how the transition model update is modified.

$$\delta'(i,j) = \frac{\sum_{t=1}^{T-1} C_t \alpha_t(i)\delta(i,j)\theta_j(e_{t+1})D_{t+1}\beta_{t+1}(j)}{\sum_{t=1}^{T-1}\sum_{j=1}^{N} C_t \alpha_t(i)\delta(i,j)\theta_j(e_{t+1})D_{t+1}\beta_{t+1}(i)}. \tag{1}$$

The variables $\alpha_t(i)$ and $\beta_t(i)$ are scaled by multiplying them by C_t and D_t respectively. The scaled forms are written using the notation $\hat{\alpha}$ and $\hat{\beta}$. Thus:

$$C_t\alpha_t(i) = \hat{\alpha}_t(i)$$

and

$$D_t\beta_t(i) = \hat{\beta}_t(i).$$

The sensor model update $\theta'_j(k)$ can be modified in a similar way. Rabiner shows that the terms $C_t D_{t+1}$ can be expressed in a form independent of t, so that they cancel, leaving the update operations as shown in Definitions 13 and 14.

3.3.2. Scaling with multiple sequences

Rabiner discusses the fact that, depending on the kind of HMM being learned, there may be a need to learn using multiple histories in preference to one long sequence of evidence. In this case, it is necessary to modify the reestimation formulas to add together the individual frequencies of occurrence of each sequence. Before this sum, the expected frequency of transitions from i to j in sequence k must be scaled by dividing it by the likelihood of sequence k given the model. The expected frequency of state i in sequence k must also be divided by this likelihood. If P_k is the likelihood of sequence k this can be achieved by multiplying the contributions made by this sequence to both the numerator and denominator by $\frac{1}{P_k}$.

$$\delta'(i,j) = \frac{\sum_{k=1}^{K}\frac{1}{P_k}\sum_{t=1}^{T_k-1}\alpha_t^k(i)\delta(i,j)\theta_j(e_{t+1}^k)\beta_{t+1}^k(j)}{\sum_{k=1}^{K}\frac{1}{P_k}\sum_{t=1}^{T-1}\alpha_t^k(i)\beta_t^k(i)}. \tag{2}$$

From Rabiner we have that

$$C_{T_k} = \frac{1}{P_k},$$

so, by writing Eq. (2) in terms of the scaled forward and backward variables we obtain:

$$\delta'(i,j) = \frac{\sum_{k=1}^{K}\sum_{t=1}^{T_k-1}\hat{\alpha}_t^k(i)\delta(i,j)\theta_j(e_{t+1}^k)\hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K}\sum_{t=1}^{T-1}\hat{\alpha}_t^k(i)\hat{\beta}_t^k(i)}. \tag{3}$$

Eq. (3) corrects Rabiner's equation 111, in [5], in which he erroneously leaves in place the $\frac{1}{P_k}$ terms. These should be removed as they have already been taken into account in the

scaled variables. This observation was also made by Kevin Murphy in his implementation of the HMM code in the BNT package [33].

3.3.3. Initialising the model parameters

In order to help EM to avoid converging on a local maximum that is far from a global maximum, we try to make the initial model $\lambda^0 = (\Psi, \xi, \pi, \delta^0, \theta^0)$ as informative as possible. Ψ is created by state splitting applied to the initial set of state labels, L. We split both the visible and subjective states, with the consequence that the visible states can be subdivided into sets of substates. This makes it difficult to ensure that the useful ordering that exists between the visible and subjective states is maintained.

A simple way to avoid this problem is not to include the visible states in the splitting process. However, we wish to allow interesting sub-states of the starting and finishing behaviours to be identified if they exist in the data. We therefore restore the ordering property by introducing supplementary start and end states that can be ordered before and after (respectively) all the states in Ψ.

During the state-splitting process the visible states, *starting* and *finishing* are replaced by sets of states in Ψ. We specify a supplementary start, s_{start}, that precedes all of the states in Ψ that are associated (through state splitting) with the visible state labelled *starting*, and a supplementary end, s_{end}, that succeeds all of the states in Ψ associated with the visible state labelled *finishing*. These supplementary states are added to Ψ and allow us to define δ^0 as follows:

$$\delta^0(x, s_{start}) = 0, \quad \text{for all states } x,$$
$$\delta^0(s_{end}, s_{end}) = 1,$$
$$\delta^0(s_{end}, x) = 0, \quad \text{for all states } x \neq s_{end}.$$

The initial probabilities of transition between the supplementary states and the other states of the model are arranged so that transitions from the supplementary start state are associated with a very high probability of entering the substates of the original visible *starting* state, and transitions from the substates of the original *finishing* state are associated with a very high probability of entry into the supplementary end state. The probability of transitions between all remaining pairs of states are assumed equal. The details of this construction are discussed in Appendix A.

Introduction of the supplementary states slightly complicates the construction of our initial sensor model, θ^0. We must specify the observation probability associated with each of the supplementary states. These states, which have been artificially introduced, have no particular association with real evidence. However, they must be associated with distributions over the evidence items in such a way that they do not distort the learning process.

Our solution to this problem is to introduce a supplementary start observation and a supplementary end observation, e_{start} and e_{end}, and to associate, with very high probability, the supplementary states with their corresponding supplementary observations. These details are also discussed in Appendix A.

Finally, π must be extended to include the supplementary states, with a probability of 1 associated with the supplementary start state.

Using the supplementary states we construct the initial model:

$$\lambda^0 = \left(\Psi \cup \{s_{start}, s_{end}\}, \xi \cup \{e_{start}, e_{end}\}, \pi, \delta^0, \theta^0\right).$$

3.3.4. Finding the best state sequence

In order to use the learned HMM to diagnose the state of the robot given a history $\langle e_1, e_2, \ldots, e_n\rangle$, we need to be able to find the optimal state sequence associated with the history: that is, the state sequence that best explains $\langle e_1, e_2, \ldots, e_n\rangle$. The Viterbi algorithm [24] is a dynamic programming algorithm that finds the best state sequence $\langle q_1, q_2, \ldots, q_n\rangle$ for the given history.

The Viterbi procedure relies on a quantity

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1, q_2, \ldots, q_t = i, e_1, e_2, \ldots, e_t \mid \lambda)$$

which corresponds to the highest probability, given the model λ, along a single path, $q_1, q_2, \ldots, q_t$, at time t, that accounts for the first t evidence items and ends in state i. Rabiner presents an inductive definition of $\delta_t(i)$ that is identical to the definition of the forward variable, $\alpha_t(i)$, reported here in Definition 11, except in using maximisation over previous states instead of the summation in the inductive definition of $\alpha_t(i)$. The Viterbi procedure must also keep track of the states along the highest probability path, so it maintains an array from which the path can be extracted at the end of the maximisation process.

We use the Viterbi procedure to evaluate the quality of the learned HMM. The details of our evaluation procedure are presented in Section 6.

4. Experimental setup

4.1. Robotics environment

Although our approach is task-independent we chose to experiment with learning a model of a navigation task. This is a fairly complex task for behavioural modelling, whilst at the same time well-understood and therefore easily experimented with. The low level functionalities comprising navigation have been thoroughly explored in mobile robotics, providing a firm foundation to support the learning process. To be performed robustly, navigation involves many different capabilities including localisation, terrain modelling and motion generation adapted to the presence of obstacles. Our approach is built on top of this level. Given the basic navigation capabilities we learn a passive model of the behavioural states that the robot visits when navigating a certain distance in a certain class of environments. We are not trying to improve the way the robot navigates, but to understand how it navigates in order to be able to predict and explain the robot's behaviour in future executions of the navigation task.

In order to build a coherent model of the navigation action, we performed a large number of experiments with a nomadic XR4000 platform. The software system we used was an original architecture developed at LAAS [34]. The sensory-motor functions are separately programmed in functional modules, using a tool named GenoM [35].

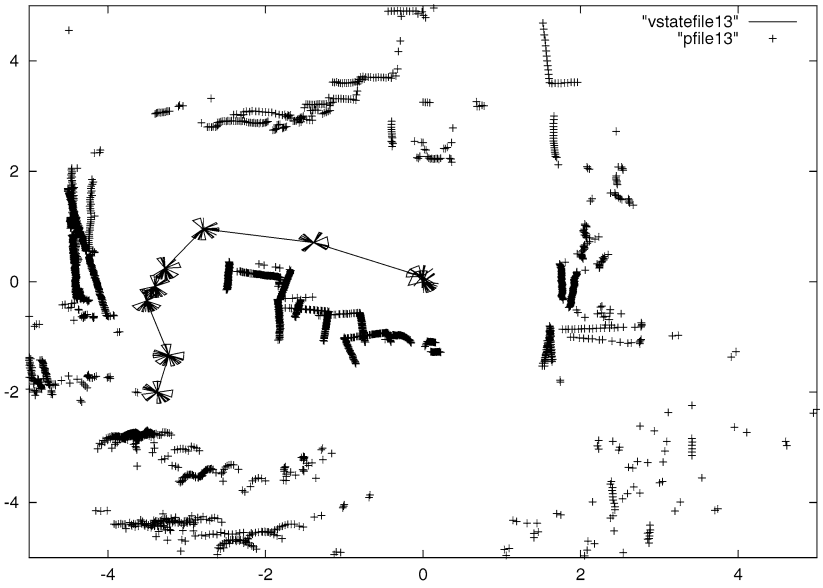

Fig. 9. A typical environment configuration from the robot's point of view.

We chose a particular navigation technology which is well suited to the environments in which our robot can manoeuvre. The technology is based on the use of odometry for localisation, a Sick® laser range scanner for obstacle detection, and the Nearness Diagram technique described in [36–38] for map building, obstacle avoidance and motion generation. This technique for navigation behaves very well in highly cluttered and dynamic indoor environments. It is, of course, not well suited to every kind of environment.

We recorded 58 trajectories, each taking between 30 and 90 seconds to complete, with the robot navigating approximately 10 metres. Our environment was unstructured, consisting of a cluttered open space open to human traffic. We made the environment vary between trajectories, from sparsely to highly cluttered and very dynamic. Fig. 9 shows a typical environment configuration. The space is an open area within a busy laboratory. Obstacles are placed within the space. The picture shows the positions of the obstacles and of the desks and walls bounding the area, according to the laser readings of the robot. The positions of the obstacles are plotted according to readings taken at different points along the trajectory. The localisation technique being used by the robot is based on odometry which explains inaccuracies in the alignments of the obstacle positions as seen from different locations. The approximate trajectory of the robot is shown as it travels from its starting point to its destination in a given run. At each of the points shown the laser scan is represented by a collection of sectors each of which represents a segment that is devoid of obstacles according to the laser scanner.

The state of the system was sampled at a frequency of 5 Hz. Each sampling recorded the values of 16 variables, including the following raw features: the coordinate position of the robot, relative to its starting position within a given coordinate system; the laser readings indicating the positions of obstacles and their proximity to the robot; the speed at which

the robot was travelling in the x and y directions; the angular velocity of the robot and the Euclidean distance travelled since the last measurement.

The choice of variables to record and to use in the construction of feature vectors is, of course, highly dependent on the task, the functional level chosen for modelling and on the sensory capacity of the executive in question. However, the methodology we have followed in the research described in this paper is not restricted to the particular task and robot we have considered. It can be applied to the learning of different tasks, using alternative robot platforms with different sensory capabilities.

4.2. The navigation states

In our experiment we used an *a priori* set of labels consisting of two visible states (the *starting* and *finishing* states) and four subjective states (*hesitation*, *obstacle avoidance*, *progress* and *search*). The progress state is the state in which the robot is moving unencumbered through the environment. Hesitation is the state in which the robot is temporarily trapped in a highly cluttered region and is unsure how to proceed. Searching represents the robot embarking on routes, which turn out to be dead ends, in its effort to find a path. Obstacle avoidance is visually distinguishable from hesitation and searching because the robot is typically making progress and then veers to avoid something in its path. We did not identify any failure states in this experiment although it would be straightforward to include failing trajectories (when the robot collides with an obstacle it prematurely terminates its trajectory) and to identify the corresponding failure states. We make no limiting assumptions that prevent the inclusion of failure states. However, our robot very rarely collided with obstacles, thanks to the efficacy of its control software, so we did not gather data representative of failures in our experiment.

4.3. Sensory-motor data and features

We identified eight features as important for discriminating between the behaviours of the robot in its execution of the navigation task. These are: *distance from origin*, *curvilinear distance* travelled over the sequence, *change in heading* over the sequence, *total rotation*, *clutteredness*, *distance from goal*, *speed* of travel and *acceleration*. These features are obtained by smoothing and integration over 6-second intervals of time. These are standard techniques used in signal processing [39] so we do not describe them here.

The variables *distance from origin* and *distance from goal* are useful because they help to discriminate between the visible states *start* and *end*. The distance from origin is calculated as the Euclidean distance between the position of the robot at the start of the trajectory (its starting coordinate) and its position at the start of the fragment of the trajectory captured by the feature vector. The distance to goal is calculated as the Euclidean distance between the position at the end of this fragment and the goal coordinate.

Curvilinear distance is a segmented approximation of the actual curvilinear distance travelled by the robot over the fragment of the trajectory represented by the feature vector. It is estimated as the sum of the Euclidean distances travelled between successive observations in the fragment. *Change in heading* is a measure of the magnitude of the angular change over the fragment. A large change in heading indicates that the robot is turning fre-

quently, perhaps through large angles. This would tend to indicate that the robot is avoiding an obstacle or searching for a viable path. *Total rotation* measures the extent to which the change in heading is cancelled out by turning back and forth rather than by turning predominantly in one direction. Rapid oscillating is associated with hesitating and searching behaviours, giving rise to small angular turns the sum of which is close to zero. *Clutteredness* is a measure of the density of obstacles in the robot's immediate vicinity over the duration of the sequence. It is a smoothed representation of the clutteredness associated with the individual observations in the fragment represented by the feature vector.

Speed of travel is a smoothed representation of the speed at which the robot is travelling over a fragment corresponding to a feature vector. *Acceleration* is a measure of the change in speed of the robot over the fragment, obtained by taking the difference between the maximum and minimum speeds at which the robot travelled over consecutive observations. If the speed is low but the acceleration is high, this would indicate that the robot is braking often and then speeding up again, as might occur when the robot is negotiating its way around obstacles.

Finding a discriminating set of features in a complex data set is a challenging problem. We experimented with various different combinations before arriving at the above collection of eight features. Our choice of features was influenced by the particular task at hand: a different task would require a different set of discriminating features to characterise it.

5. Learning a hidden Markov model

The preceding sections have described the components necessary to learn a hidden Markov model from the raw signals emitted by a physical system. We now bring these components together into a learning process that receives the signals emitted by the robot's sensors and outputs a learned HMM. In the rest of this paper we discuss the quality of the models of the navigation task learned using our methodology.

However accurate its readings might be, the observations of the robot do not precisely correspond to the reality in which the robot was operating. The robot can observe the world only partially by means of its sensors. Since we are interested in knowing how the robot will behave in reality it is necessary to make a connection between the internal world of the robot and the external world in which it acts and senses. We approached this problem through the use of a simple labelling strategy.

5.1. Labelling the feature vectors

To make a connection between the robot's observations and the states of the HMM we devised a method of labelling the observations with identifiers from the set L, described in Section 2. In our experiments L consisted of the six labels identified in Section 4.2, two of which were visible and four subjective. These six labels correspond to behaviours that the experimentors were able to recognise and distinguish with reasonable certainty. Any visually distinguishable behaviours can be used for labelling. As described in Section 3.1, this *a priori* collection of labels is refined according to the patterns identified automatically in the data set during the clustering phase.

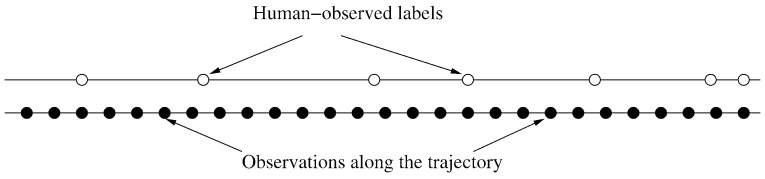

Fig. 10. A labelled trajectory.

During the robot runs, whenever the robot displayed a distinguishable behaviour we interrupted its recording so that it would associate the next observation with the corresponding label identified by the experimentor. Each trajectory is therefore partially labelled. This labelling process, coarse though it is, gives us a way of relating the recorded data to a subjective judgement of the external reality in which it was acquired. Fig. 10 shows a partially labelled robot trajectory.

The experimentor tended to introduce a slight delay into the labelling because of taking time to recognise the behaviour being displayed. Thus, labels tend to occur slightly later in the trajectories than the points at which the associated behaviours really occurred. This can be taken into account in the interpretation of the labelled trajectories, as we explain in Section 6. The subjective nature of the judgements made by the experimentor of course means that these judgements do not precisely correspond to reality. We discuss the consequences of this, for evaluation of the learned model, in Section 6.

It is important to emphasise that the labels play no role at all in the clustering of the data. The clusterer is concerned only with the feature vectors and ignores the labels in both the training and classification processes. The labels are used when clustering is complete, in the construction of the sensor model, as described in Section 3.1.2.

It would of course be surprising if human observers could select a collection of hidden states that turned out by chance to be the most useful ones for learning an accurate next-state transition function. We believe that the state-splitting technique we describe in Section 3.2 helps to mitigate the effects of choosing an *a priori* label set by introducing missing states that are important in determining the behaviour of the robot but are not necessarily susceptible to interpretation by the human observer.

5.2. Constructing the evidence sequence

The EM algorithm learns to improve a given initial model with respect to the evidence that was observed by the signal source (in this case, the robot). Evidence can be presented in a single sequence, or in multiple sequences, depending on the properties of the model. Our experiments were divided into separate trajectories of the robot, each one terminating when the robot reached its goal position. Because of the structure of a trajectory one of the properties of the model is that it is not fully ergodic—the robot always progresses from its *start* state towards its *end* state. We therefore chose to present the evidence as a set of separate histories, each one derived from a different trajectory. This presentation excludes transitions from the end state into the start state, so enables us to construct δ^0 as defined in Section 3.3.3.

Each history, h_t, is derived from the ladder, l_t, obtained by sequence vector construction from a given trajectory t. Our strategy is to build a sparse ladder, sl_t, by taking every kth sequence vector from l_t, where k is a density value determined experimentally. For example, if $k = 1$ then sl_t will be identical to l_t, whilst if $k = 15$ then sl_t will be a thinned version of l_t containing every 15th step in l_t.

Following construction of sl_t each of the feature vectors obtained from the sequence vectors in sl_t are presented to the Kohonen network for classification. Thus, each step $\vec{f}_k$ in sl_t is classified with an evidence item $\xi_{sl_{t_k}}$, so that the sparse ladder $sl_t = \langle \vec{f}_1, \ldots, \vec{f}_n \rangle$ produces a history $h_t = \langle \xi_{sl_{t_1}}, \ldots, \xi_{sl_{t_n}} \rangle$ of evidence associated with the trajectory t.

We would expect the quality of the learning to improve as the frequency k increases. When evidence is sampled at a low frequency much of the robot's behaviour is omitted from the history and the association between the evidence items sampled and the observed behaviour is likely to be missed. With higher frequencies the history is richer and this association is more likely to be found. The results we present in Section 6 show that indeed, up to some point, higher frequencies result in better models being learned.

Fig. 11 shows an example of a learned HMM where the evidence was sampled at 0.6 second intervals. The picture shows that the state-splitting process produces 65 states from the initial set of 7 labels (including the *no mark* label). To simplify interpretation of the graph, states are grouped into rectangles associated with the labels they refine. Dark transitions represent the highest probability transitions between states whilst lighter edges represent lower probability transitions.

5.3. Parameter settings

Before discussing the results we explain how we chose the values of the parameters that govern important aspects of the clustering and learning processes.

When decomposing the initial set of labels, $L \cup \{nomark\}$, using the evidence items in ξ, it is necessary to decide which characteristic vectors should participate in the decomposition of each label. For each label $l \in L \cup \{nomark\}$ we used a fixed threshold of association between the characteristic vectors and that label to determine which vectors to partition into the substates of the label. The *association* of a characteristic vector, c, with a label, l, is given by the number of sequence vectors labelled with l that were classified with c after the network training process. The reason for setting a threshold is that some characteristic vectors turn out to be very marginally associated with some labels. There can be one or two orders of magnitude difference between a low association and the mean for a given label. We judged that low associations can be the consequence of noise effects in the training process.

The threshold was defined as

$$m_l = \mu_l - \sigma_l/4,$$

where μ_l is the mean association between the characteristic vectors and l, and σ_l is the standard deviation of the association. Experiments showed that using the mean association as the threshold led to too many vectors being excluded from the decomposition of the label, and improved results were obtained by lowering the threshold slightly. We achieved

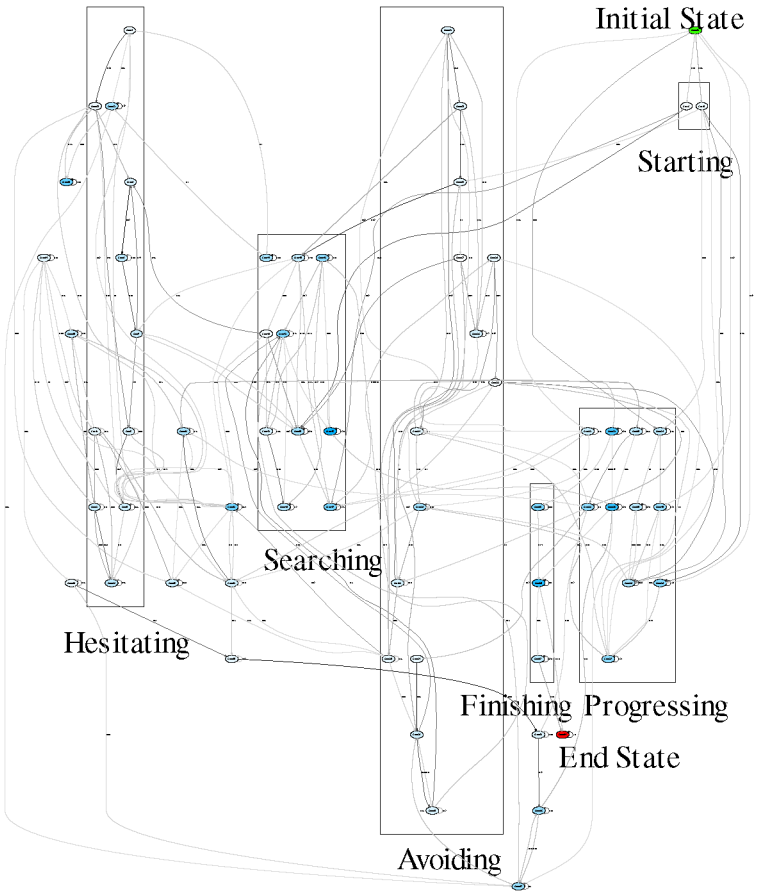

Fig. 11. A learned HMM for the navigation task. The rectangles (which are all tall and narrow) are labelled with the high level labels associated with the collections of states inside them. The Initial and End states are the dummy states used to construct θ^0.

good results by subtracting $\sigma/4$. Our goal is to lower the threshold just enough to increase robustness to noise in the training and classification processes.

Having selected a group of characteristic vectors to partition into the substates of a label we then need to decide on the degree of separation between substates. We used an angular separation of 40 degrees to determine whether two vectors could be considered part of the same substate. A larger angle than this causes the substates to fragment and destroys the structure of the label. Too small an angle results in large substates and inadequate decomposition. The importance of this parameter was discussed in Section 3.2.

To construct the evidence histories we took every third sequence vector from the collections of sequence vectors generated from the 58 trajectories in our data set. We considered lower frequencies, and we present comparative results in the next section. Because observations were sampled at 5 Hz, selecting every third vector corresponds to taking evidence every 0.6 seconds along the trajectory.

6. Results and discussion

We present analyses of two sets of results. We first consider the quality of the learned HMM in terms of its ability to produce traces through the abstract state space that correspond to those that the robot was observed to follow in reality. The quality of the HMM is highly dependent on the clustering phase. As a second stage in our evaluation we therefore examine the stability of the clustering results.

The clustering process is mainly affected by two parameters: the size of the network and its random initialisation. As noted in Section 3.1, the results of the clustering phase are also slightly sensitive to the order of presentation of the training data, but we do not discuss this issue further. We experimented with a range of different network sizes and noted how these affect the quality of the traces produced by the consequent learned HMM. In the discussion below we present the results obtained using a network of size 30. In Appendix B we present an evaluation of HMMs learned using networks of different sizes. We found that the clustering results can be sensitive to the random initialisation, leading to varying sized sets ξ and Ψ. We therefore average our results for a given network size over 20 different random initialisations of the network.

6.1. Evaluation of the HMM

The Viterbi algorithm provides a way of diagnosing the behaviour of the robot from the observations it makes in the execution of its task. Given the histories of evidence items constructed during the clustering process, and the learned state transition and sensor models, we can diagnose the most probable state transitions of the robot by finding the most probable explanation for each evidence item given the states it visited so far.

One way to evaluate the quality of the learned HMM is to compare the sequences of states constructed by the Viterbi algorithm with those that the human observed the robot visiting during its execution of the task. We refer to a sequence of states visited by the Viterbi algorithm as a *Viterbi sequence*, and to the states along such a sequence as the *Viterbi states*. The human observer drew observations from the set of visible and subjective states, L, defined in Section 2.

The only way we have of identifying the states actually visited by the robot in a given trajectory is to use the labelled observations in that trajectory. The labelling process was inaccurate because of the difficulty, for the human observer, of distinguishing between similar states of the robot (for example, between hesitation and searching). Furthermore, the human observer tended to label late because it took time to interpret the robot's behaviour and select the most appropriate label. This means that the label often ended up being associated with data recorded after the robot had already transitioned to a different state. Finally, there were fewer sequence vectors in the data set labelled with visible states than labelled with subjective states, because the robot entered the visible states less frequently. The learning process was therefore slightly biased against recognising the starting and finishing behaviours as distinct from the other behaviours in the model. However, although the labelling process was flawed, the labels do provide us with a way to connect the Viterbi sequences with an observed (though somewhat noisy) reality.

We evaluate a given Viterbi sequence V by comparing it with the labelled trajectory it corresponds to. The states in V are separated by k/r seconds, where k is the frequency with which the sequence vectors are sampled from the trajectories in the construction of the histories and r is the rate (in Hz) at which observations were sampled by the robot. The chosen frequency determines the density of evidence items in the histories. The Viterbi sequence V is obtained from a given history, H, and there is exactly one Viterbi state in V for every evidence item in H. For each history we record the *trail* of Viterbi states corresponding to the evidence items and then super-impose the human-observed labels at the times along these trails at which they occur in the underlying trajectory. We define the association between a history and a trail as follows.

Definition 15. A *trail*, $T = \langle t_1, \ldots, t_h \rangle$ is a sequence of Viterbi states corresponding to a history H of h evidence items. For each evidence item in H there is exactly one Viterbi state in T.

Definition 16. A *labelled trail* is a trail on which human-observed labels have been super-imposed. These labels do not necessarily coincide with states on the trail.

Definition 17. The *trail fragment* preceding label l in a labelled trail is the sequence of Viterbi states, $V = \langle v_k, \ldots, v_m \rangle$, intervening between the last human-observed label before l on the trail, and l. If l is not coincident with v_m then we add to the fragment the Viterbi state, v_{m+1}, immediately following l.

As Definition 17 shows, the trail fragment preceding a label l can actually contain the first Viterbi state following l on the trail. The reason for this is that, if l lies between two Viterbi states, it might correspond to the Viterbi state on either side of it. The human labelling process was not sufficiently reactive for this possibility to be ruled out.

We increment the score for a trail each time there is a match between the human-observed label and the preceding trail fragment. The super-imposition of human-observed labels along the trail, and the association between the label and the preceding trail fragment, can be seen in Fig. 12.

Care has to be taken in defining what is meant by a *match*. Because of the problem of late labelling, discussed in Section 5.1, we look in the preceding trail fragment for a substate that is identical in structure to any one of the substates comprising label l. Definition 19 states this precisely.

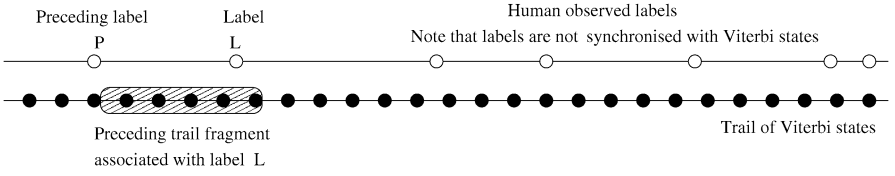

Fig. 12. The structure of a trail.

Definition 18. Two states s_i and s_j in Ψ are ξ-identical if they contain exactly the same elements of ξ.

Definition 19. A human-observed label $l \in L$ is associated with a set of substates $S = \{s_1, \ldots, s_m\}$ by means of the mapping *refine*: $L \to \mathbb{P}\Psi$. We say that l *matches* its preceding trail fragment V if at least one of the substates in V is ξ-identical to one in the set S.

Given that the preceding trail fragment might be long, Definition 19 might seem over-permissive. It suggests that the scoring rate for poor Viterbi sequences could be artificially increased because, in a long preceding trail fragment, the likelihood of seeing a matching sub-state seems high. This might be the case if preceding trail fragments tended to exhibit much fluctuation between states at the level of the human-observed labels. However, Fig. 13 shows that, across all the network sizes we used, the preceding trail fragments are highly stable at this granularity despite the occurrence of many subtle state changes at the granularity of the sub-states.

We compared the fluctuation between Viterbi states within a trail fragment, and between the corresponding interpretations of those states under the application of $ab: \Psi \to L$. In Fig. 13 the rows correspond to different sizes of clustering network. The columns describe the degree of state variation observed in the trail fragments generated by the Viterbi algorithm using a model learned on the bases of these networks.

Variation is measured by counting how many times the value changes within each trail fragment. The final columns show the mean variability ratio, substates to labels, as a percentage, and its standard deviation. It can be observed that, in a network of size 30, there is three times more variation at the substate level than at the label level, and this picture is fairly consistent across the different network sizes. Furthermore, the variability within the label level is very low (consistently less than one state change). We find these results very encouraging, because one would expect, in a rational system, to see more significant state change at fine levels of granularity, with stability increasing as the granularity increases.

6.2. Precision measurements

Analysis of the relationship between substates and labels demonstrates that the same substate can be associated with multiple labels, revealing some confusion in the model's ability to distinguish similar behaviours.

Consider a substate, s, that is shared between k different labels. During the evaluation of a Viterbi sequence the score will be incremented if the human observed any one of

	Label variability		Substate variability		Variability ratios	
	Mean	Std	Mean	Std	Mean (%)	Std (%)
15	0.82	0.29	1.81	0.60	229	60
20	0.80	0.30	1.96	0.69	255	73
25	0.81	0.29	2.13	0.74	274	82
30	0.79	0.28	2.20	0.75	295	96
35	0.78	0.27	2.28	0.77	307	86

Fig. 13. Table showing state fluctuation within trail fragments.

the k labels and the Viterbi sequence visited s during the preceding trail fragment. The usefulness of this evaluation depends on k being as close to 1 as possible.

We observed that in the case where a substate maps to 4 or more labels, it can add points to the evaluation almost regardless of the label applied by the human observer. This makes such a substate almost completely undiscriminating, artificially inflating the correspondence between the Viterbi sequences and the label sequences. This led us to devise a method for measuring the degree of *precision* of the $ab : \Psi \to L$ mapping.

The measure is obtained by calculating, for each substate, the number, n, of labels with which it is associated. This number is used to give the number of pairwise comparisons from which Viterbi sequences containing this substate could benefit. We sum this value over all of the m substates, giving the following quantity:

$$\sum_{i=0}^{m} n_i * (n_i - 1).$$

This quantity is then divided by the total number of substate pairs:

$$\frac{\sum_{i=0}^{m} n_i * (n_i - 1)}{m * (m - 1)}$$

resulting in the proportion of all possible comparisons from which a Viterbi sequence could benefit undeservedly from visiting the substate. The higher this value the lower the precision of the mapping. We call this value the *confusion factor*.

As can be observed from the comparison presented in Appendix B, the confusion factor is highest in small networks. This can be explained because small clustering networks lead to few distinct characteristic vectors, so that state-splitting results in a high degree of sharing of vectors across states.

6.3. Results

Fig. 14 shows the Viterbi sequence evaluations obtained from a HMM learned on the basis of 20 randomly initialised networks of size 30. The results are presented as a distribution over the 58 trajectories and the 20 random numbers (1160 values). The mean score was 76.18%, which is very promising. However, because of the precision issue we must take into account the extent of confusion exhibited by the model. A high degree of confusion would undermine this apparently high score. In order to evaluate how good the score really is we must also take into account the consistency of agreement between the Viterbi sequences generated using different random number seeds. Low consistency (indicating that the quality of the HMM is sensitive to the random initialisation of the clustering network) would also undermine the goodness of the score.

Fig. 15 shows the benefit, as a percentage of overall score, obtained from confusion resulting from a network of size 30. The rows of the table correspond to the human-observed labels, while the columns correspond to states in $L \cup \{nomark\}$ obtained by applying $ab : \Psi \to L$ to the winning states in the preceding trail fragments of the Viterbi sequences (the Viterbi state that wins in a comparison is the one that is responsible for incrementing the score on that comparison). We denote the *no mark* state using NM. This state was never observed by the experimentor but could be identified as the most probable

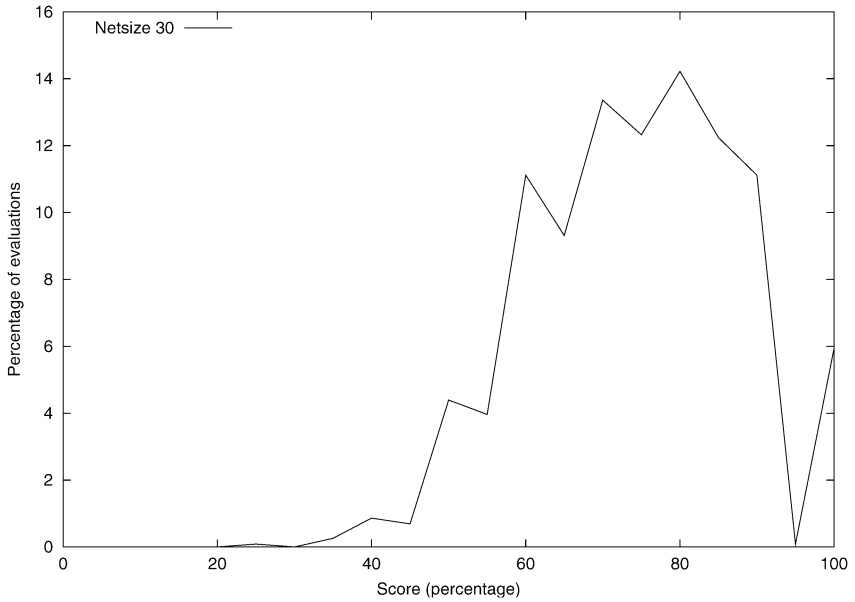

Fig. 14. Comparison of Viterbi sequence evaluations using a network of size 30. Performance shown here does not take into account either confusion or consistency, both of which affect the true quality of the learned HMM.

Human-observed label	Viterbi state						
	1	2	3	4	5	6	7
1		0	0.1%	0	0.3%	0.0%	0.0%
2	0		0.6%	0	0.1%	0.8%	1.1%
3	0	1.7%		0	0.8%	1.8%	1.7%
4	0	0	0		0.0%	0	0
5	0.0%	0.1%	1.1%	8.3%		2.0%	0.8%
6	0	1.4%	1.4%	0	1.3%		0.6%

Fig. 15. Table showing benefit obtained from confusion in a network of size 30. Most benefit is obtained from 5/4 confusion. Other benefits are minimal.

next Viterbi state. It can be observed that by far the greatest benefit was obtained when the human-observed label was 5 and the winning Viterbi state was 4. All other benefits obtained from confusion are minimal.

We calculated the confusion factors for 20 HMMs obtained from size 30 networks using 20 different random number seeds. The upper bound confusion factor, computed over these 20 models using the formula presented in Section 6.2, is 0.023, with a median value of 0.008. By contrast, the upper bound confusion factor for 20 models learned from size 20 networks is 0.052,with a median value of 0.017 (about twice as much confusion as for size 30 networks). Models learned from size 35 networks exhibit a lower level of confusion, with an upper bound of 0.015 and median value 0.006. However, differently randomly initialised size 35 networks lead to lower consistency across the corresponding learned HMMs, as we discuss below.

The high degree of 5/4 confusion can be explained in the following way. The label 4 corresponds to the finishing state and the robot normally entered the finishing state immediately after visiting the state labelled 5. Thus, sequence vectors labelled 5 have many features in common with those labelled 4, causing the clusterer to confuse the corresponding observations. A sequence vector labelled 5 is therefore likely to be classified with an evidence item associated with a substate of state 4. This is occurring when the Viterbi sequence proposes a 4 when the human-observed label was a 5. This confusion very rarely occurs the other way around because δ^0 is a Bakis model [40] (a partially ordered model) which strongly reinforces the recognition of the terminal state.

For a given network size and trajectory, consistency is a measure of the agreement between the Viterbi sequences generated for that trajectory over the 20 different random numbers. Clearly, the most reliable performance is obtained when confusion is low and consistency is high. We obtained the best combination of these factors using a network of size 30. Fig. 16 shows the consistency obtained using 20 size 30 networks. This graph depicts the mean percentage of the Viterbi sequences that agree on each state visited along each of the 58 trajectories. It shows that the highest degree of consistency reached is 92%, whilst the models demonstrated at least 77% consistency on 90% of the trajectories.

The frequency at which evidence items are sampled from the data significantly affects the quality of the learned HMM, according to our evaluation. Fig. 17 shows that performance quickly declines as the frequency decreases, consistent with the hypothesis proposed in Section 5.2.

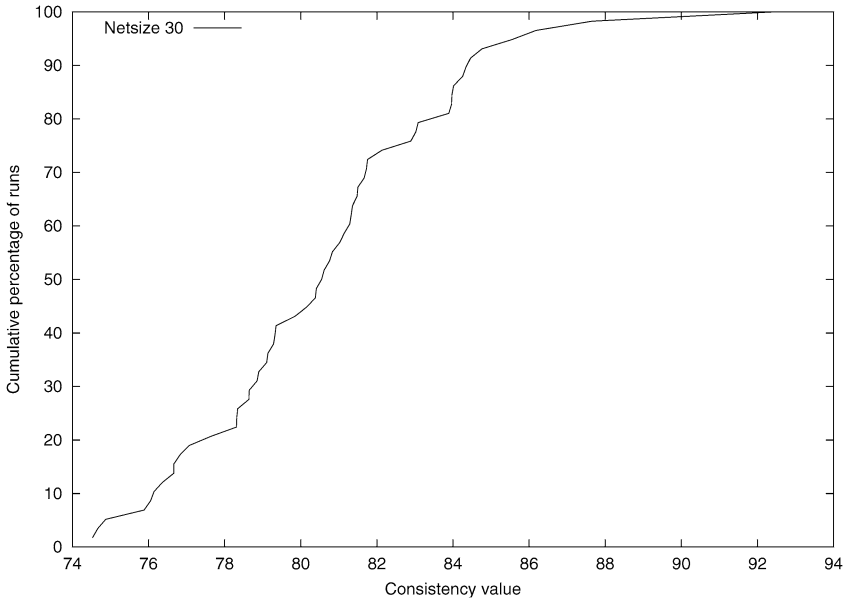

Fig. 16. Consistency of agreement between Viterbi sequences for models based on networks of size 30. The x-axis shows the percentage of agreement obtained. The y-axis shows the cumulative percentage of trajectories bounded by the corresponding degree of consistency.

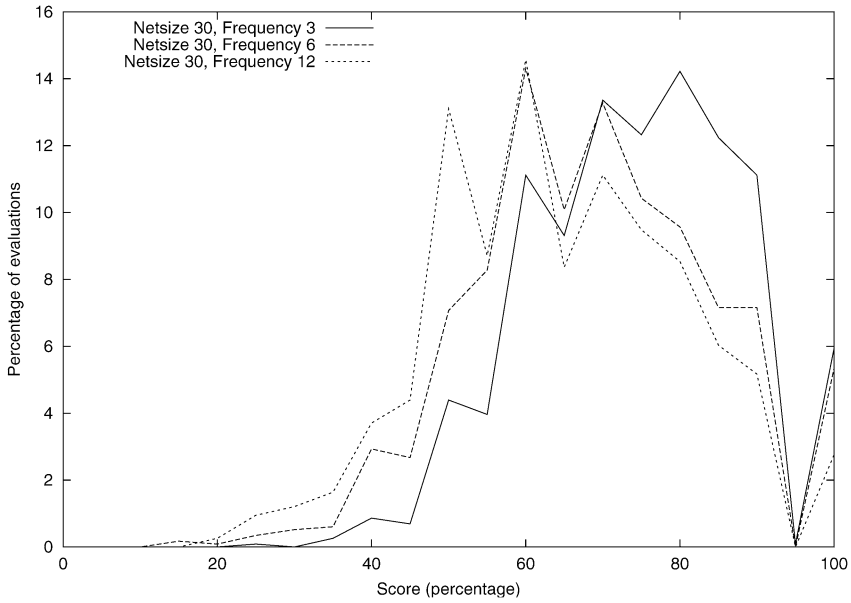

Fig. 17. Comparisons of Viterbi sequence evaluations using different evidence frequencies. The frequency number indicates the separation of successive sequence vectors, in $\frac{1}{5}$ seconds, since the raw data is sampled at 5 Hz. Thus, a large frequency number corresponds to a lower frequency of evidence sampling. Lower frequencies lead to a degradation in performance.

We performed an experiment to determine whether the construction of the initial sensor model using the code book approach gives any advantages over using a random initial sensor model. Fig. 18 shows that a clear advantage is obtained. We also tested the advantage obtained from state-splitting, by comparing the results obtained using state splitting with those obtained from the initial set of user-supplied labels only. Fig. 19 allows us to conclude that state-splitting yields a highly significant advantage.

6.4. Evaluation of the clustering phase

We focus our discussion on the stability of the clustering results we obtained for a given network size. Different random initialisation led to a marked difference in the sizes of the sets ξ and Ψ. We have already seen (in Fig. 16) that the differences observed in the sizes of ξ and of Ψ do not lead to a consequent divergence in the behaviour of the learned models. We now show that the cluster structures are stable despite the variation in the sizes of the ξs constructed using different random initialisations.

In the table in Fig. 20 we show the extent to which the identity of states in L is preserved across different random initialisations. We require the following definitions.

Definition 20. The *weight* of an element $s \in \Psi$ is computed as the sum of the associations between s and each element $e \in \xi$ such that $e \in ev(s)$. The association is determined by the cell (s, e) in the matrix defined by θ^0. We denote the weight of s by ω_s.

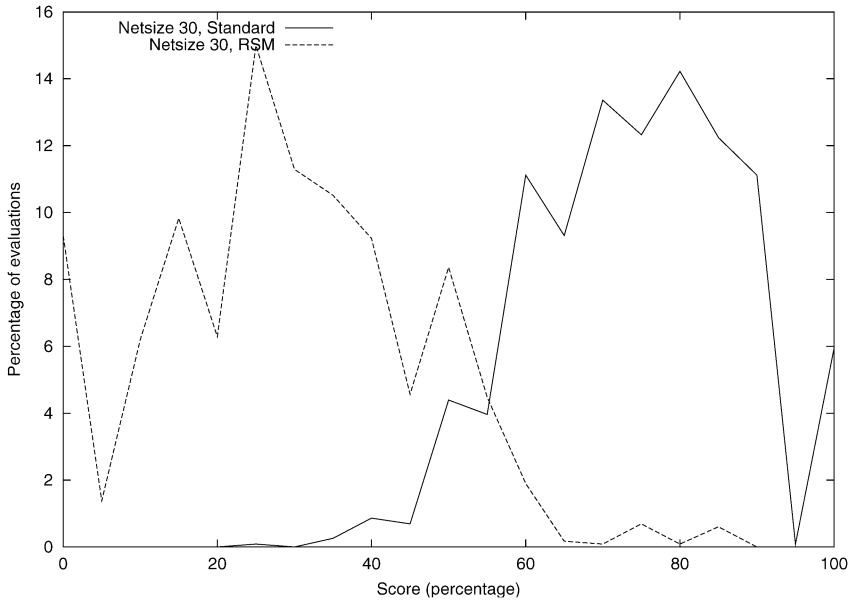

Fig. 18. Comparing random initial sensor model with a code book generated after Kohonen network clustering. Random initial sensor models lead to highly significantly poorer performance than the code book sensor models.

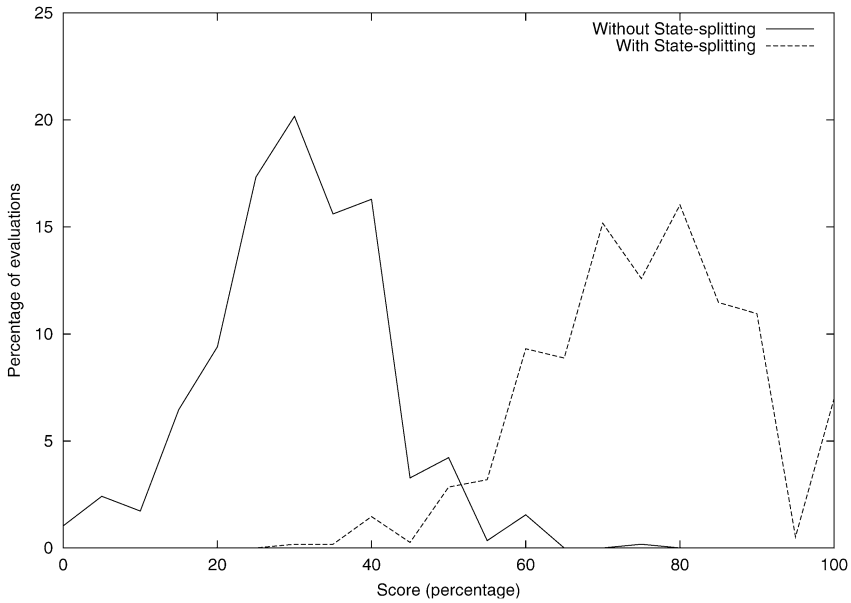

Fig. 19. Comparing the results using state-splitting with results obtained not using splitting.

Net size 30							
Label	1	2	3	4	5	6	NM
1	66.28	0.59	0.53	0	19.44	6.4	6.76
2	0.5	36.18	8.23	0.29	0.28	21.18	33.34
3	0.89	5.49	30.49	0.05	6.04	35.96	21.08
4	0	0.37	0.11	50.8	45.77	0.44	2.51
5	14.65	0.13	2.35	25.57	46.82	2.93	7.54
6	3.81	7.55	13.95	0.24	2.89	59.07	12.48
NM	3.4	12.86	8.66	0.57	7.86	13.12	53.53

Fig. 20. Table showing strength of identity of states in L across random initialisations of the clustering network. Values are percentages. We consider confusion with the *no mark* state, NM, not to be problematic.

Definition 21. The *combined weight* of an element $l \in L$ is computed as the sum of the weights ω_s of each element s in *refine*(l).

We now define a measure of association between two substates drawn from different state sets, Ψ_1 and Ψ_2.

Definition 22. The *association product* of two substates $s_1 \in \Psi_1$ and $s_2 \in \Psi_2$ is computed as the product of the combined weights of the labels, l_1 and l_2, obtained by $ab_{\Psi_1}(s_1)$ and $ab_{\Psi_2}(s_2)$ respectively.

Definition 23. The *centre point* of a substate $s \in \Psi$ is computed as the average of the evidence items in $ev(s)$. We denote the centre point by cp_s.

To measure the degree of preservation of identity of states in L we construct a square matrix in which each cell (i, j) indicates the extent to which is and js coincided across random initialisations of the cluster network. The matrix indicates that states in L preserve their identity well if the values along the diagonal are high (preferably the highest values in each row).

Given two collections of substates, Ψ_A and Ψ_B, computed using different random number seeds A and B, we first compute the centre points of the elements in the two collections. We then sample a centre point, cp_{s_A}, from Ψ_A and measure its closeness to each of the centre points computed in Ψ_B. The closest centre point in Ψ_B is denoted $cp_{s'_B}$. The matrix entry for (s, s') is increased by the association product of s and s'. The results of our analysis are shown in the table in Fig. 20, where it can be observed that the values along the diagonal are indeed the highest, except in the case of label 3 where there is significant confusion with label 6. The table shows the percentages of the associations within each row attaching to each label. We include the *no mark* label, which we denote NM. Some confusion between certain pairs of states, such as 3 and 6, is evident, as we observed in our discussions of confusion in Section 6.1. Nevertheless, the results overall indicate that variation in the number of evidence items and substates constructed by the clusterer, across different random initialisations, does not translate into instability in the recognition of the key behavioural states.

7. Future work

In our future work we intend to use the learned HMMs for monitoring and controlling the execution of the corresponding tasks. We are developing a plan execution architecture in which the learned HMMs provide a low level state estimation capability that supports the monitoring of the execution of a specific planned action (or task). As the robot is executing a task the Viterbi algorithm can be used on-line to track the most likely trajectory followed by the robot and to provide a detailed picture of how the execution path is likely to unfold. The key advantage is that the HMM allows failure to be predicted before it occurs, enabling an appropriate response such as the early termination of activity.

Fig. 21 shows part of the architecture of a plan or policy execution system that uses the learned HMMs for monitoring and controlling the execution of dispatched actions. At its simplest, controlling the execution of a task could be limited to aborting its execution when the probability of failure is predicted to be above a given threshold. The component of the architecture labelled *state estimation* is the component that tracks the traversal of the current HMM(s) and reports the behavioural state of the robot to the execution monitor at regular intervals. On each state report the monitor sends a command to the execution subsystem to either abort the task execution or to continue following the low level program associated with the task (this could be a hand-programmed control strategy or a policy, whatever was the low level control strategy being followed by the robot when the HMM was learned). When the reported state is one of the terminal states of the task, the execution monitor reports the task as having been successfully completed and the HMM corresponding to the next dispatched action is accessed.

The relationship between the monitor and the state estimation component is inspired by the model-based diagnosis work of Williams and his co-authors [21,41]. These techniques underpin the plan execution and monitoring capability of the Remote Agent [42] which remains one of the most prominent and successful applications of plan execution

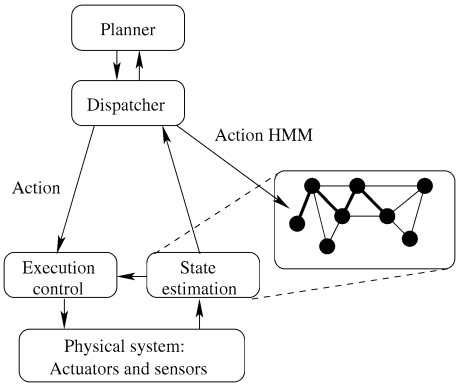

Fig. 21. A plan execution architecture using behavioural state monitoring. When an action is dispatched for execution, the corresponding learned model is dispatched to the state estimation module to allow behaviour tracking. The heavy line in the model suggests the most probable trajectory being followed. There is feedback from the state estimation to the execution monitor and dispatcher to communicate successful termination of a task or its probable failure.

technology. Livingstone, the model-based reasoning component of Remote Agent, is a discrete model-based controller using a single declarative spacecraft model to detect failures during the execution of planned steps and propose strategies for the repair or replacement of failed components. In our work the models are behavioural rather than physical and diagnosis identifies deviant behaviour rather than disfunctional system components. Once recognised, deviant behaviour can be terminated to avoid either task failure or the unnecessary waste of resources committed to a failing endeavour.

We have taken care in this paper to show that our work with the navigation task is not specific to any particular robot platform or environmental setup (although the learned model is dependent on these factors). To explore the generality of our approach we have begun to consider other tasks belonging to other application areas. In particular, we have successfully applied the whole learning process to learning models of both navigation and simulated science-gathering actions on a different robot platform.

8. Conclusions

We have shown that stochastic learning techniques from signal processing can be used to learn a hidden Markov model of a robot's behaviour as it executes a given task. The model provides an introspective capability which can be integrated with high level mission planning and reasoning. The learning of this model can be completely automated from the point of acquisition of sensor readings by the robot. Although we have presented our work in terms of the specific task of indoor navigation, using a specific robot platform, our approach can be generalised to different tasks and platforms. We make no assumptions about the sensory-motor equipment of the robot. The initial clustering phase is completely general, resulting in the organisation of the raw sensor readings into a discretized collection of codes. The codes are taken to be abstractions of the robot's observations, sensed by means of an abstract sensor as described by a code-book based sensor model. A probabilistic state transition model is then learned, together with a refined sensor model, using the Expectation Maximization algorithm.

Although EM has been used before to learn models of behaviour, we have made several innovations. First, we do not define the state set in advance, but leave this to be determined following the clustering phase. Thus, the number of states in the transition model is determined dynamically and the human makes few prejudgements about the nature of the behavioural model. Second, the states in our model correspond to substates of the behaviours, such as *hesitation*, *obstacle avoidance* and *search*, of the robot, rather than configurations of the robot with respect to physical features of its environment. Thus, the learned HMM is a model of how the robot behaves, which applies equally well in any physical environment sharing the same structural features as the ones in which learning took place.

We have so far evaluated the learned HMMs by using the Viterbi algorithm to explain histories of evidence obtained by classification of the observations recorded by the robot during executions of its task. We compared the Viterbi sequences generated with the labels applied by the human observer to the robot observations. These labels provide a connection with reality which, although not perfect, allows us to estimate the extent to which the

learned HMM accounts for the uncertainty in the actual execution environment. Our next step is to evaluate the learned HMMs by using them as the basis of an execution monitoring strategy on-board the robot. Much remains to be done, but we believe we have made an important first step towards learning a reliable connection between the raw sensed data recorded by a robot and a symbolic reasoning level.

Acknowledgements

We would like to thank Felix Ingrand, Derik Schröter, Brian Williams and Nicola Muscettola for helpful discussions and suggestions. We are grateful to Jonathan Gough for helping to explore the generality of our approach by considering its application to learning models of different tasks. We would like to thank the anonymous reviewers for their observations and suggestions. Finally, Maria Fox wishes to thank the CNRS for funding the sabbatical visit to LAAS that made this work possible.

Appendix A. Implementation

A.1. Implementing the scaling and reestimation mechanism

As described in Section 3.3.1, the forward and backward variables $\alpha_t(i)$ and $\beta_t(i)$ are scaled using the scaling coefficient

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)}.$$

Thus, the same term is used for scaling $\beta_t(i)$ as is used for normalising $\alpha_t(i)$, so a simple strategy is to collect the normalisation terms that are computed during the forward computation and use them to scale the backward variables during the backward computation.

To reestimate the δ component of the model we simply calculate the sum, over all times t, of the expected frequency of transitions from any state i into any state j, divided by the sum, over all times t, of the expected frequency of transitions from state i. The accumulation of the first sum is performed by line 15 and the second by line 10 of the procedure *update*δ in Fig. A.3. The division is computed by line 6 of procedure *do*δ*update* in Fig. A.5.

A similar update function can be defined for θ, which calculates the expected frequency of being in a state i while observing evidence e, divided by the expected frequency of being in state i. These functions implement the equations presented in Definitions 13 and 14. When multiple histories are used the situation is complicated by the need to calculate these frequencies independently for the different histories.

In Fig. A.1 we show the reestimation mechanism that is typically presented for single histories of evidence. Fig. A.2 contrasts this with the multiple histories case.

To implement the equation shown in Section 3.3.2 we distinguish between the local and global frequencies of occurrence. For the transition model update function this requires a square matrix and a vector to be defined for each history h_k: F_{ij_k} (the local frequencies of any i, j transitions) and F_{i_k} (the local frequencies of transitions from any state i). A global matrix, F_{ij}, must also be defined, together with a global vector F_i. The dimensions of all these structures are determined by the number of states in δ.

```
1:  Procedure: reEstimate(M,h,P)
2:  Input: model M, single history h of T evidence items, prior state distribution P
3:  Output: updated model M
4:  repeat
5      {forwardBackward returns the array of T state distributions, sv}
6:     {Note that forwardBackward also initialises the α and β terms used in updateδ}
7:     sv = forwardBackward(h,P)
8:     (F_i, F_ij, F̄_i) = updateδ(M,h,sv)
9:     CF_ij = updateθ(M,h,sv)
10:    M = doδupdate(M, F_ij, F_i)
11:    M = doθupdate(M, CF_ij, F̄_i)
12: until convergence
13: return M
```

Fig. A.1. The reestimation function for the single history case. The array F_i is the expected frequency over $T-1$ timepoints of transitioning from each state i. F_{ij} is the expected frequency of transitioning from a state i to a state j. $\overline{F_i}$ is the expected frequency over all T time points of transitioning from each state i. The array CF_{ij} is the expected frequency of being in state i observing evidence j.

For each history h_k the values of F_{ij_k} and F_{i_k} are calculated within the same local *update* procedure as used for the single history case, shown in Fig. A.3. These are then summed into F_{ij} and F_i at the end of each iteration (lines 13 and 16 in Fig. A.2). The division of each F_{ij} by F_i takes place when this summation is complete, as can be seen by examination of the equation in Section 3.3.2. This is implemented by line 6 of Fig. A.5, as in the single history case. Finally, after each iteration of the reestimate procedure we update δ and θ (lines 26 and 27 of Fig. A.2) and then reset the local and global matrices and vectors to zero.

The sensor model θ is updated in a similar way. Given a history h_k, the matrix CF_{ij_k} stores the local frequencies with which evidence items j are seen in states i. These values are summed into the global matrix CF_{ij} on line 22 of Fig. A.2. On line 17 of *do*θ*update* (Fig. A.5) it can be seen that each CF_{ij} is divided by a further array, $\overline{F_i}$, and not by F_i as in *do*δ*update*. The reason is that F_i does not store the expected frequency of exiting the Tth state because no transitions from the Tth state are possible. However, evidence can be observed in the Tth state, so the correct updating of θ relies upon the division of CF_{ij} by the expected frequency of transitions accumulated from all timepoints. We therefore accumulate the values of this array in line 14 of the *reEstimate* procedure shown in Fig. A.2. The need to construct this additional array, storing this one additional value, applies whether single or multiple histories are used.

Fig. A.2 and its auxilliary procedures correctly implement Rabiner's scaling and reestimation mechanism for the case where multiple evidence sequences are presented to the EM procedure.

A.2. Dealing with split starting and finish states

When it is known that a process is characterised by distinct start and end states the best estimation of the underlying HMM can be obtained using a left-right [5], model. In

```
1:  Procedure: reEstimate(M,H,P)
2:  Input: model M, set of histories H, prior state distribution P
3:  Output: updated model M
4:  initialise F_i[], F̄_i[], F_ij[][], CF_ij[][]
5:  repeat
6:     reset F_i
7:     reset F_i j
8:     reset CF_ij
9:     for all h in H do
10:       sv = forwardBackward(h,P)
11:       (F_ih, F_ijh, F̄_ih) = updateδ(M,h,sv)
12:       for i = 0..NUMSTATES-1 do
13:          F_i[i] += F_ih[i]
14:          F̄_i[i] += F̄_ih[i]
15:          for j = 0..NUMSTATES-1 do
16:             F_ij[i][j] += F_ijh[i][j]
17:          end for
18:       end for
19:       CF_ijh = updateθ(M,h,sv)
20:       for i = 0..NUMSTATES-1 do
21:          for j = 0..NUMOBS-1 do
22:             CF_ij[i][j] += CF_ijh[i][j]
23:          end for
24:       end for
25:    end for
26:    M = doδupdate(M, F_ij, F_i)
27:    M = doθupdate(M, CF_ij, F̄_i)
28: until convergence
29: return M
```

Fig. A.2. The modified reestimation function for the multiple history case. The array F_{i_h} is the expected frequency of transitioning from each state i calculated in the local context of history h. F_{ij_h} is the expected frequency of transitioning from i to j in the context of h. CF_{ij_h} is the expected frequency of being in state i seeing evidence j, calculated in the context of h.

such a model there is an ordering on the states that excludes certain state transitions. If the estimation process begins by knowing this ordering it can converge on a better estimation of the state transition function, with a higher log likelihood, than is possible if it begins with an equal probabilities transition model and prior state distribution.

In our experiments the robot always begins a trajectory in its starting position, and it ends the trajectory as soon as it judges itself to be within a given tolerance of the goal coordinate. Having ended the trajectory it never enters other states. It is impossible for the robot to enter the starting state from any other state. Therefore, there is an ordering imposed on the states: the *starting* state is always visited *before* any other state, and the *finishing* state is always visited *after* any other state. The other states are not ordered, so the model is not a strictly left-right model. If we fix the state collection in advance, and identify the *starting* and *finishing* states, we can enforce the ordering that exists by initialising the EM

```
1:  Procedure: updateδ(M,h,sv)
2:  Input: model M, single history h of T evidence items, array of state distributions sv
3:  Output: triple containing:
4:  array of expected number of transitions from each state (F_i)
5:  array of expected number of transitions between state pairs (F_ij) and
6:  array of expected number of times in each state (F̄_i)
7:  initialise F_ij[][], F_i[], F̄_i[]
8:  for i = 0..NUMSTATES-1 do
9:     for t in 0..T-2 do
10:       F_i[i] += sv[t][i]
11:    end for
12:    F̄_i[i] = F_i[i] + sv[T-1][i]
13:    for j = 0..NUMSTATES-1 do
14:       for t in 0..T-2 do
15:          F_ij[i][j] += α_t(i) * β_{t+1}(j) * p(q_{t+1} = j|q_t = i) * p(h[t+1]|q_{t+1} = j)
16:       end for
17:    end for
18: end for
19: return (F_i, F_ij, F̄_i)
```

Fig. A.3. The procedure for calculating the expectation values for the δ update.

```
1:  Procedure: updateθ(M,h,sv)
2:  Input: model M, single history h of T evidence items, array of state distributions sv
3:  Output: array of expected number of times in each state i seeing evidence j (CF_ij)
4:  initialise cprobij[][]
5:  for i in 0..NUMSTATES-1 do
6:     for c in 0..NUMOBS-1 do
7:        for t in 0..T-1 do
8:           if h[t] == c then
9:              CF_ij[i][c] += sv[t][i]
10:          end if
11:       end for
12:    end for
13: end for
14: return CF_ij
```

Fig. A.4. The procedure for calculating the expectation values for the θ update.

process with a state transition model in which the column associated with the *starting* state is set to zero and the row associated with the *finishing* state is also set to zero in all but one position. Because each row in the matrix is a distribution the (final state,final state) position must be set to 1.

Unfortunately, it is not an acceptable approach to fix the state collection in advance: we want the learning process to identify states that are not necessarily apparent to the human observer, and to learn how important they are in explaining the behaviour of the

```
1:  Procedure: doδupdate(M,probij,probi)
2:  Input: model M, array of expected number of transitions between state pairs (probij),
    array of expected number of transitions from each state (F_i)
3:  Output: updated model M
4:  for i = 0..NUMSTATES-1 do
5:     for j = 0..NUMSTATES-1 do
6:        M.tm[i][j] = F_ij[i][j]/F_i[i]
7:     end for
8:     normaliseδrow(i,M.tm)
9:  end for
10: return M
11:
12: Procedure: doθupdate(M,cprobij,allprobi)
13: Input: model M, array of expected number of times in each state i seeing evidence j
    (cprobij), array of expected number of times in each state (F̄_i)
14: Output: updated model M
15: for i = 0..NUMSTATES-1 do
16:    for j = 0..NUMOBS-1 do
17:       M.sm[i][j] = CF_ij[i][j]/F̄_i[i]
18:    end for
19:    normaliseθrow(i,M.sm)
20: end for
21: return M
```

Fig. A.5. The procedures for updating δ and θ. These are shared by both the single and multiple history reestimation procedures.

robot. As described in Section 3.2, we use an automated state-splitting procedure to enable the identification of such states. Thus, although we begin with an initial set of states in which there are defined *starting* and *finishing* states, after state-splitting there might be (and frequently are) several *starting* and *finishing* sub-states.

In our experiments therefore, the selection of the terminal states is complicated by the state splitting procedure. If either (or both) of the *starting* and *finishing* states is associated with distant groups of observations after the clustering process, they will be split into sub-states around these groups. When either the *starting* or the *finishing* state is split, it is not possible to identify any one of their sub-states as definitive terminal states without distorting the transition model and biasing the outcome of the estimation process. One possibility is to initialise the transition network with equal probabilities, instead of with a zero column and row. The problem is that the equal-probability network is uninformative and the quality of the resulting HMM is degraded.

A better solution to the problem is to define supplementary terminal states for the model. We can identify four different cases: the case in which neither state is split; the case in which the *starting* state is split; the case in which the *finishing* state is split and the case in which both states are split. It is possible to treat all of these cases in a uniform way with the introduction of supplementary states. This requires us to modify the sensor model that was constructed following the state-splitting phase. In addition we must create an initial transition model and a prior state distribution that contain the supplementary states.

Modifying the sensor model is straightforward: we simply add two additional rows and columns to the sensor model and associate them with corresponding supplementary observations. The supplementary observations can only be observed in the corresponding supplementary states. The two new rows are then normalised.

Creating the initial transition model is somewhat more complex. We begin by allocating equal probabilities to all state transitions and then we adjust the model to contain the additional two columns and rows. The supplementary starting state has a zero probability of entry from any other state in the model, and a tiny but non-zero probability of exit into any state other than one of the defined starting substates. There is equal probability of transition into any of these. The supplementary finishing state has a zero probability of entering any state other than itself (which it enters with a probability of 1), and the only states that can transition into it are the finishing substates. Each finishing substate can enter one of the other finishing substates or the supplementary finishing state with equal probability. If there are n finishing substates then, for each one, there is a $1/n + 1$ probability of entry into the supplementary finishing state. Of course, in the case where the original finishing state is not split this means that there is a probability of 0.5 of the finishing state entering the supplementary finishing state, and an equal probability that it will re-enter itself. Similarly, where the original starting state is not split the supplementary starting state enters the starting state with probability of almost 1. Fig. A.6 shows how the supplementary starting and finishing states are connected to the rest of the states in the transition model.

Every row in the transition table needs to be normalised to ensure that it is a valid next state distribution. At the end of this process the supplementary starting and finishing states are indistinguishable from the other states in the model.

The prior probability distribution needs to be modified to enforce the fact that the system always starts in the supplementary starting state. From here it can enter the transition model as described above, with the highest probability being associated with a transition into

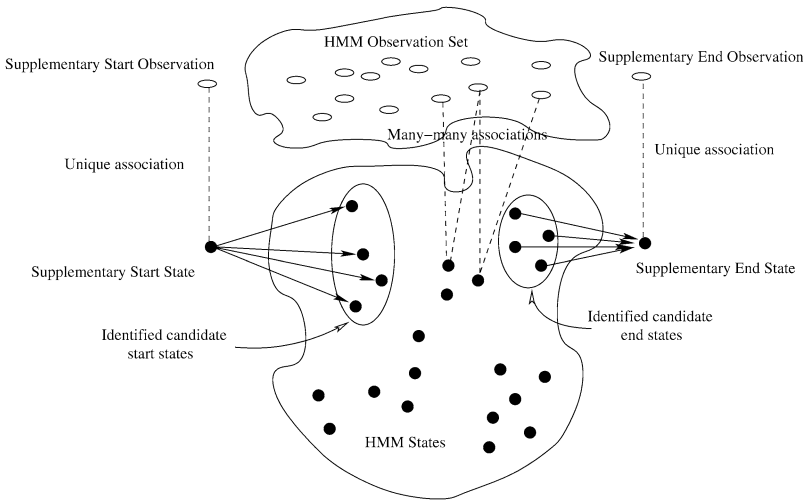

Fig. A.6. The use of supplementary terminal states and supplementary observations.

one of the starting sub-states. We cannot eliminate the possibility that the system starts in some other state however. It does happen that one of the substates associated with the original nomark state is entered before the starting state. This can happen because, in the real data sequences it could happen that the starting state was marked late so that the first few sequence vectors constructed are marked with the nomark identifier.

There is a final modification that must be made. The evidence from which the HMM is estimated must be modified so that the observations corresponding to the supplementary starting and finishing states appear at the start and end, respectively, of each history. Having made this modification it is now possible to treat all cases in a uniform way.

Appendix B. Further experimental comparisons

In Fig. B.3 we examine the effect of varying the network size on the quality of the HMM finally learned. We experimented with five different network dimensions, from 15 to 35, and compared the differences in the results of the exact match evaluation strategy described above. We ran an ANOVA test to discover whether there is any significant difference between the performances of the 5 different sizes. We computed an F value of 4.033, giving a p value of 0.003. This shows that the difference is highly significant. Furthermore, it can be observed in Fig. B.5 that very small network sizes tend to result in much greater sharing of substates between labels, leading to a relatively high confusion factor.

The table in Fig. B.1 shows the means, standard deviations and median values obtained for each of the five network sizes. It can be seen that the standard deviation increases as the network size increases, showing that the ability of the learned HMM to reliably explain the behaviour of the robot decreases as the Kohonen network gets larger.

The same pattern can be seen in Figs. B.5 and B.2, showing confusion and consistency respectively. The confusion factor *de*creases as the network size increases, which shows that the mapping $ab : \Psi \rightarrow L$ becomes increasingly precise as the network size increases. It can be seen that consistency is greatest for the sizes 20, 25 and 30.

Network size 15 shows reduced consistency as well as the highest confusion factor of all of the network sizes. Its deceptively strong performance, as shown in Fig. B.3, is undermined by these factors and we therefore dismiss 15 as being too small to give adequate reliability. At the other extreme, network size 35 shows the weakest performance in Fig. B.3, but has the lowest confusion factor. It might therefore be argued that the loss of performance arises from the fact that it is benefitting less from the imprecision

Size	Mean	Std	Median
15	79.31	12.74	80.00
20	77.43	12.95	78.57
25	77.12	12.90	78.57
30	76.18	13.09	76.47
35	74.63	13.89	75

Fig. B.1. Means, standard deviations and median values for evaluation distributions over 20 random numbers for each of 5 network sizes. We note that the median value is larger than the mean for network sizes 15, 20 and 25.

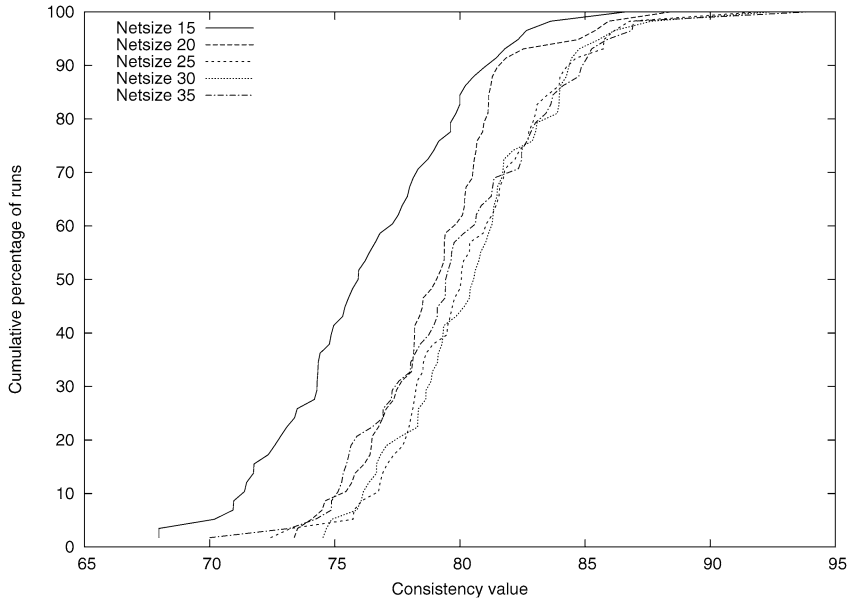

Fig. B.2. Consistency of agreement between Viterbi sequences for a given network size and run, over 20 different random numbers.

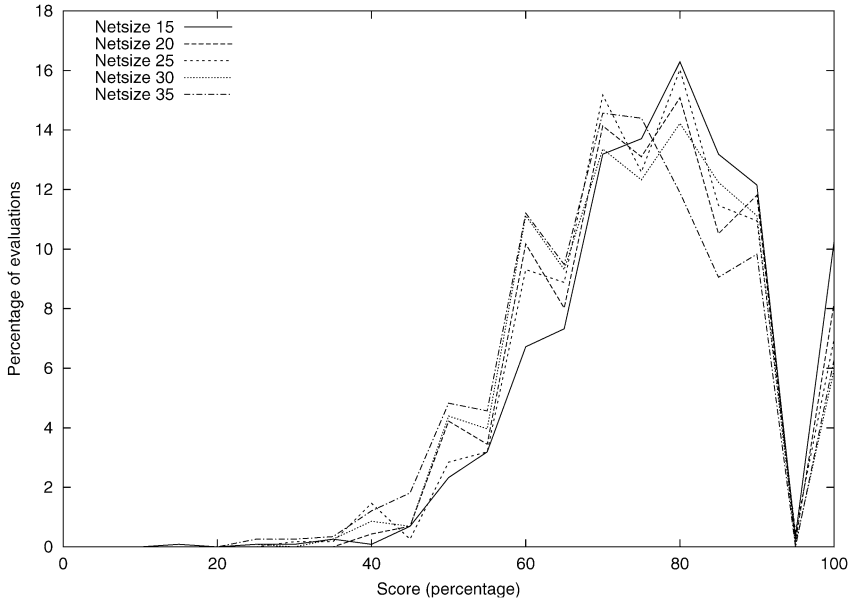

Fig. B.3. Comparison of Viterbi sequence evaluations using 5 different Kohonen network sizes. Best performance is obtained for HMMs based on the smallest networks, and steadily declines as network size increases. However, performance shown here does not take into account either confusion or consistency, both of which affect the true quality of the HMMs.

of $ab : \Psi \rightarrow L$. However, as Fig. B.2 shows, consistency of agreement between Viterbi sequences based on the size 35 network is slightly lower than in the smaller networks.

Our best results are obtained using networks of size 25 and 30. These networks are very similar in terms of the performances shown in Figs. B.3 and B.1. In both cases, consistency is high. The network of size 30 benefits from a slightly lower confusion factor than is obtained for 25, so can be said to produce a slightly better overall picture.

We observed that the standard deviation on the scores is generally higher than might be expected in networks where the consistency is high and the standard deviation on the consistency is low. For a network of size 30, the standard deviation of the consistency is 3.4, with a mean of 80.6. For a network of size 25 these statistics are 3.4 and 80.4 respectively. Given these very small standard deviations, we sought an explanation for the high standard deviations in performance for these network sizes. The only other parameter that can result in variation is the history being considered. The implication of this is that some histories must correspond to Viterbi sequences that score consistently well, whilst others must correspond to sequences that score consistently badly. An examination of the scores for individual Viterbi sequences confirmed that this is indeed the case. To illustrate its significance we removed the five consistently most badly scoring sequences and compared the resulting distribution of results with the full distribution. It can be seen in Fig. B.4 that the removal of these five sequences results in a highly significantly improved performance in networks of size 25 ($t = 10.3$) and 30 ($t = 10.2$).

We performed an experiment to compare the results obtained using an exact match test with those obtained using the weaker subset match test. In the subset match, the score is incremented if any Viterbi state in the trail fragment preceding l is a *subset* of some substate

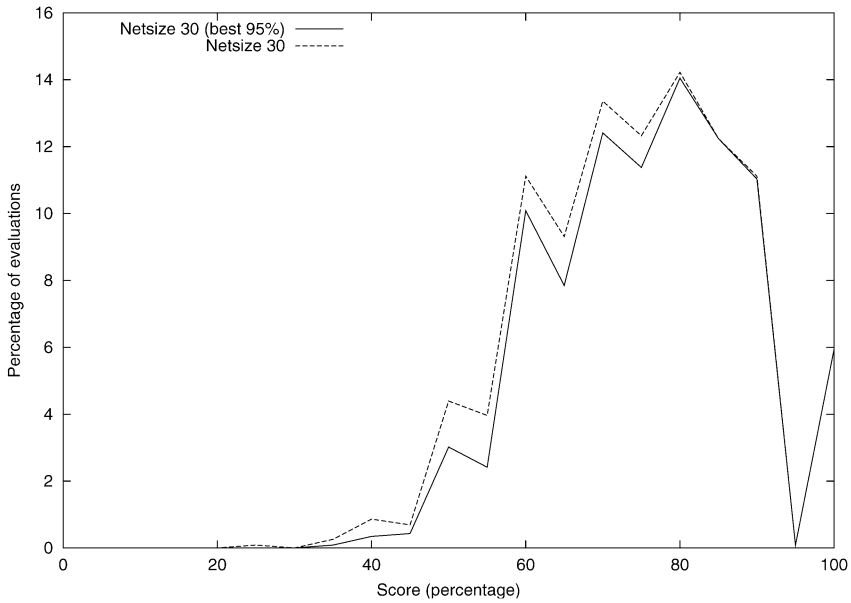

Fig. B.4. Comparison of results obtained from all sequences and results obtained from the best 95%. The removal of the consistently poor scoring sequences results in a significantly better overall performance.

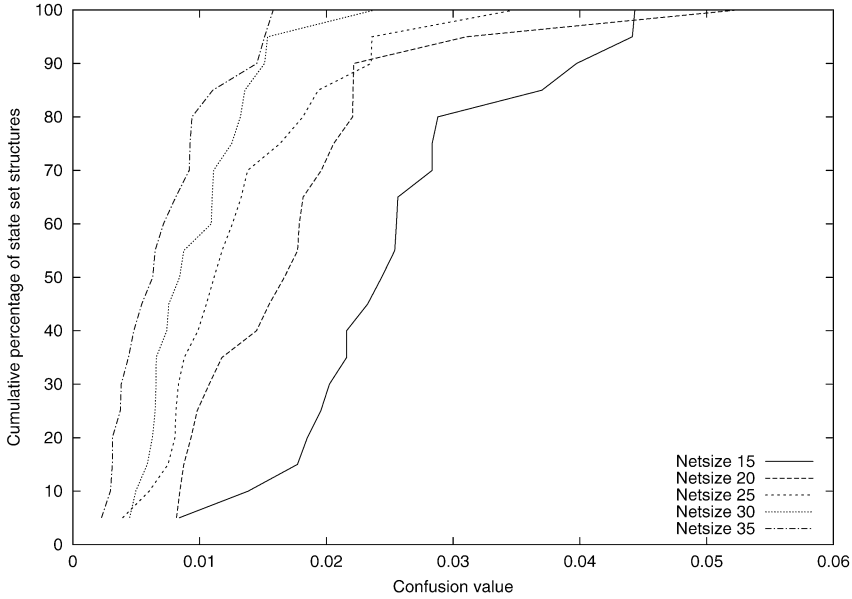

Fig. B.5. Comparison of confusion factors as network sizes grow. For each network size 20 Ψs were generated. The graph shows the cumulative percentage of these 20 models reaching each of the increasing confusion factors shown on the horizontal axis. Small networks show greatest confusion, with confusion factors of 0.052 being reached. Using a network of size 30, a confusion factor of 0.023 is the highest obtained.

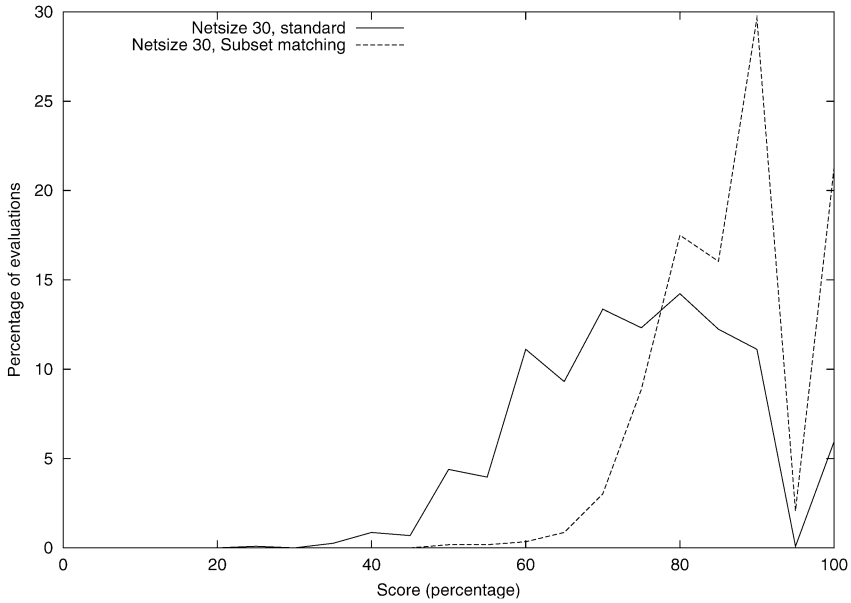

Fig. B.6. Comparing exact matching with subset matching. It can be observed that the weaker subset-based test yields significantly better results than the exact match.

in l. Fig. B.6 shows the comparative results obtained using a network of size 30. This graph shows that great advantages are obtained using the subset test: the mean score increases from 76.18 to 88.97, with a standard deviation of only 8.49 (by comparison with 13.09). If it is accepted that the subset test is adequately rigorous this means that, in general, the Viterbi sequences were consistent with the human-observed behaviour about 89% of the time.

Preliminary investigations suggest that *transitionary* states, which the robot visits between recognised substates, might be identifiable using the subset test. We believe that a state that represents the transition from a previous state, s, will share many features in common with s, and that this commonality might be accessible through the subset test. This needs further investigation and is a focus of further work.

References

[1] S. Koenig, R.G. Simmons, Unsupervised learning of probabilistic models for robot navigation, in: Proceedings of the International Conference on Robotics and Automation, 1996, pp. 2301–2308.
[2] S. Koenig, R.G. Simmons, Passive distance learning for robot navigation, in: Proceedings of International Conference on Machine Learning (ICML), 1996, pp. 266–274.
[3] H. Shatkay, L.P. Kaelbling, Learning geometrically constrained hidden Markov models for robot navigation: Bridging the geometrical-topological gap, J. AI Res. 16 (2002) 167–207.
[4] G. Theocharous, K. Rohanimanesh, S. Mahadevan, Learning hierarchical partially observable Markov decision process models for robot navigation, in: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2001, pp. 511–516.
[5] L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–286.
[6] H. Bui, A general model for online probabilistic plan recognition, in: Proceedings of 18th International Joint Conference on AI (IJCAI-03), Acapulco, Mexico, 2003, pp. 1309–1318.
[7] H. Bui, S. Venkatesh, G. West, Policy recognition in the abstract hidden Markov model, J. AI Res. 17 (2002) 451–499.
[8] L. Liao, D. Fox, H. Kautz, Learning and inferring transportation routines, in: Proceedings of the 19th National Conference on AI (AAAI-04), San Jose, CA, 2004, pp. 348–354.
[9] S. Osentoski, V. Manfredi, S. Mahadevan, Learning hierarchical models of activity, in: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2004.
[10] I. Cohen, N. Sebe, L. Chen, A. Garg, T.S. Huang, Facial expression recognition from video sequences: Temporal and static modelling, Computer Vision and Image Understanding (Special Issue on Face Recognition) 91 (2003) 160–187.
[11] A. Wilson, A. Bobick, Hidden Markov models for modeling and recognizing gesture under variation, Internat. J. Pattern Recognition Artificial Intelligence 15 (1) (2001) 123–160.
[12] A. Bobick, J. Davis, The recognition of human movement using temporal templates, IEEE Trans. Pattern Anal. Machine Intelligence 23 (3) (2001) 257–267.
[13] A. Wilson, A. Bobick, Parametric hidden Markov models for gesture recognition, IEEE Trans. Pattern Anal. Machine Intelligence 21 (9) (1999) 884–900.
[14] Y. Nam, K. Wohn, Recognition of space-time hand gestures using hidden Markov models, in: ACM Symposium on Virtual Reality Software and Technology, 1996, pp. 51–58.
[15] T. Oates, M. Schmill, P. Cohen, A method for clustering the experiences of a mobile robot that accords with human judgements, in: Proceedings of the 17th National Conference on AI (AAAI-00), Austin, TX, 2000, pp. 846–851.
[16] L. Chrisman, Reinforcement learning with perceptual aliasing, in: Proceedings of the 10th National Conference on AI (AAAI-92), San Jose, CA, 1992, pp. 183–188.

[17] K. Basye, T. Dean, J.S. Vitter, Coping with uncertainty in map learning, in: Proceedings of the 11th International Joint Conference on AI (IJCAI-89), Detroit, MI, 1989, pp. 663–668.
[18] T. Dean, D. Angluin, K. Basye, S. Engelson, L. Kaelbling, E. Kokkevis, O. Maron, Inferring finite automata with stochastic output functions and an application to map learning, in: Proceedings of the 10th National Conference on AI (AAAI-92), San Jose, CA, 1992, pp. 208-214.
[19] J. Firby, Modularity issues in reactive planning, in: Proceedings of the 3rd International Conference on AI Planning Systems (AIPS), 1996, pp. 78–85.
[20] M. Beetz, Structured reactive controllers: A computational model of everyday activity, in: Proceedings of the 3rd International Conference on Autonomous Agents, 1999, pp. 228–235.
[21] B.C. Williams, P.P. Nayak, A model-based approach to adaptive self-configuring systems, in: Proceedings of the 13th National Conference on AI (AAAI-96), Portland, OR, 1996, pp. 971–978.
[22] R.G. Simmons, D. Apfelbaum, A task description language for robot control, in: Proceedings of Intelligent Robotics and Systems, 1998, pp. 1931–1937.
[23] F.F. Ingrand, M. Georgeff, A.S. Rao, An architecture for real-time reasoning and system control, IEEE Expert 7 (6) (1992) 34–44.
[24] G.D. Forney, The Viterbi algorithm, Proc. IEEE 61 (1973) 268–278.
[25] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, J. Roy. Statist. Soc. 39 (1) (1977) 1–38.
[26] L.R. Rabiner, An introduction to hidden Markov models, IEEE ASSP Magazine (1986) 4–16.
[27] J. Makhoul, S. Roucos, H. Gish, Vector quantization in speech coding, Proc. IEEE 73 (11) (1985) 1551–1588.
[28] T. Kohonen, Self-Organisation and Associative Memory, Springer-Verlag, Berlin, 1984.
[29] J.B. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.
[30] A. Stolcke, S. Omohundro, Hidden Markov model induction by Bayesian model merging, in: Advances in Neural Information Processing Systems 5, NIPS Conference, 1992, pp. 11–18.
[31] L.E. Baum, J.A. Egon, An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology, Bull. Amer. Meteorolog. Soc. 73 (1967) 360–363.
[32] L.E. Baum, G.R. Sell, Growth functions for transformations on manifolds, Pacific J. Math. 27 (2) (1968) 211–227.
[33] K. Murphy, The Bayes net toolbox for MATLAB, in: Computing Science and Statistics, 2001.
[34] R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. Ingrand, An architecture for autonomy, Internat. J. Robotics Res. 17 (4) (1998) 315–337.
[35] S. Fleury, M. Herrb, R. Chatila, GenoM: A Tool for the specification and the implementation of operating modules in a distributed robot architecture, in: Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Grenoble, France, vol. 2, 1997, pp. 842–848.
[36] J. Minguez, J. Osuna, L. Montano, A "divide and conquer" strategy based on situations to achieve reactive collision avoidance in troublesome scenarios, in: Proceedings of the International Conference on Robotics and Automation (ICRA), New Orleans, USA, 2004.
[37] J. Minguez, L. Montano, T. Simeon, R. Alami, Global nearness diagram navigation (GND), in: Proceedings of the International Conference on Robotics and Automation (ICRA), Korea, 2001, pp. 33–39.
[38] J. Minguez, L. Montano, Nearness diagram navigation (ND): Collision avoidance in troublesome scenarios, IEEE Trans. Robotics Automation 20 (1) (2004) 45–59.
[39] A.V. Oppenheim, R.W. Shafer, J.R. Buck, Discrete Time Signal Processing, second ed., Prentice-Hall, Englewood Cliffs, NJ, 1999.
[40] F. Jelinek, Continuous Speech recognition by statistical methods, Proc. IEEE 64 (1976) 532–536.
[41] P. Kim, B.C. Williams, M. Abramson, Execution of reactive model-based programs through graph-based temporal planning, in: Proceedings of the 17th International Joint Conference on AI (IJCAI-01), Seattle, WA, 2001, pp. 487–493.
[42] N. Muscettola, P.P. Nayak, B. Pell, B.C. Williams, Remote agent: To boldly go where no AI system has gone before, Artificial Intelligence 100 (1998) 5–47.