Using Learned Action Models in Execution Monitoring

Maria Fox, Jonathan Gough and Derek Long
Department of Computer & Information Sciences
University of Strathclyde, Glasgow, UK

Abstract

Planners reason with abstracted models of the behaviours they use to construct plans. When plans are turned into the instructions that drive an executive, the real behaviours interacting with the unpredictable uncertainties of the environment can lead to failure. One of the challenges for intelligent autonomy is to recognise when the actual execution of a behaviour has diverged so far from the expected behaviour that it can be considered to be a failure. In this paper we present further developments of the work described in (Fox *et al.* 2006), where models of behaviours were learned as Hidden Markov Models. Execution of behaviours is monitored by tracking the most likely trajectory through such a learned model, while possible failures in execution are identified as deviations from common patterns of trajectories within the learned models. We present results for our experiments with a model learned for a robot behaviour.

1 Introduction

In (Fox *et al.* 2006) a method is described for learning behavioural models as Hidden Markov Models (HMMs), capturing the underlying patterns in the execution of fundamental operations of some executive system. The purpose of that work was to show that it is possible to construct useful models against which sensor data, collected during execution, can be tracked in order to follow the evolution of the behaviour at a more abstracted level. This is significant, because it forms a bridge from the primitive sensor level, at which raw data is available, to the abstract level at which a behaviour can be understood symbolically, reasoned about and monitored. Relevant work in the literature includes (Liao, Fox, & Kautz 2004).

Our purpose in the current paper is to show that the learned models can be used to monitor execution, so that it is possible to detect when the execution of a behaviour is deviating from the expected trajectory and infer failure in the execution process. To illustrate this idea with a simple example, consider a robot executing an action of navigating between two points. Suppose the robot has available to it sensors detecting the passage of time and its wheel rotations and a laser sensor allowing it to detect distances to objects in line-of-sight. During execution of the navigation task the robot can directly observe the values reported by these sensors, but they do not reveal whether the robot is successfully executing the task itself. In order to detect that, the robot must have a model of the process of execution and compare the trajectory of sensor readings it perceives with the expectations determined by the model. The key point is that the sensors themselves can never sense failure in the execution — the failure can only be determined by comparing the trajectory seen by the sensors with a model of the behaviour.

We begin by summarising the process by which the models are learned. We then describe the activity on which we focus in this paper and the data collection strategy. We go on to explain the techniques by which we monitor execution and, finally, we explore the extent to which these techniques have proved successful.

2 Learning Models of Hidden States

In any physical system there is a gap between its actual operation in the physical world and its state as perceived through its sensors. Sensors provide only a very limited view of reality and, even when the readings of different sensors are combined, the resulting window on the real world is still limited by missing information and noisy sensor readings. The consequence is that, in reality, the system moves through states that are *hidden* from its own, and external, perception. Furthermore, because of the inability of the system to accurately perceive its state, and the uncertainty in the physical world, the transitions between these hidden states are probabilistic. A Hidden Markov Model (Rabiner 1989) is an ideal abstract representation of this situation, comprising both a probabilistic hidden state transition system and a probabilistic relationship between the observations of

the system and its underlying states. The representation is behavioural and is abstracted from the physical organisation of the device.

In (Fox *et al.* 2006) we describe how the behaviour of a mobile robot can be learned from raw sensed data. Low level sensor readings are abstracted into a discrete set of high level observations, by means of feature detection and classification techniques, as a first step in learning a hidden state transition model. We refer to the collection of sensor readings that can be made by the robot, at some point in time, as an *observation*. We discretise the non-finite observations of the robot into a finite collection of distinct *evidence items* and determine a mapping between them. The algorithm we use to achieve this abstraction 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 allows us 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.

To discover the state set of the model we analyse the structure of the observation space, a vector space in which each vector characterises an observation that the robot can make. If two vectors are separated by only a very small angle then they are probably indistinguishable and the corresponding observations can be made from the same state. If two vectors are very far apart then their corresponding observations cannot be made from the same state, suggesting the existence of distinct states of the robot.

We therefore identify the set of states of the model by partitioning the evidence items into non-disjoint classes. The algorithm we use, which we call *state splitting*, is described in detail in (Fox *et al.* 2006). It finds the maximal cliques in a graph in which the nodes correspond to evidence items and an edge exists between two evidence items if the angle between the vectors that represent them is lower than a threshold value. Each maximum clique, corresponding to a subset of evidence items, is interpreted as a state.

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*;*
- $\pi : \Psi \rightarrow [0, 1]$*, the prior probability distributions over* Ψ*;*
- $\delta : \Psi^2 \rightarrow [0, 1]$*, the transition model of* λ *such that* $\delta_{i,j} = Prob[q_{t+1} = s_j | q_t = s_i]$ *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*);*
- $\theta : \Psi \times \xi \rightarrow [0, 1]$*, the sensor model of* λ *such that* $\theta_{i,k} = Prob[e_k | s_i]$ *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 $h = \{e_1 \ldots e_n\}$ *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) (Dempster, Laird, & Rubin 1977), also called the Baum-Welch algorithm (Rabiner 1986). 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 re-estimates 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.

Having learned a HMM it is possible to both diagnose and predict the behaviour of a similar system given the observations made by that system over time. Diagnosis is the identification of the most probable state of the system, obtained by interpreting the sequence of observations as state transitions using the Viterbi algorithm (Forney 1973). HMMs are used in speech processing (Jelinek 1976) to support both the automated recognition of phonemes in spoken text.

3 A Robot Activity

In this paper we concentrate on a model of a single robot activity. The activity we consider is that of capturing a panoramic image by taking a series of individual photographs at fixed angles during a rotation through a full 360 degree turn. The robot is expected to stay located on the same spot,

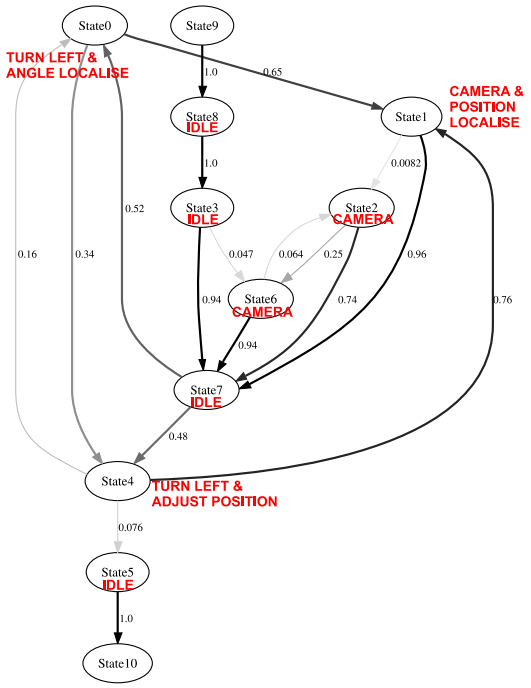

Figure 1: A learned model of execution of the panoramic image capture activity.

simply rotating in place. The robot we used for this experiment is an Amigobot and it is equipped with a ring of sonar sensors and wheel rotation sensors. Data from sonar sensors are notoriously difficult to interpret because of their susceptibility to noise and environmental interference, while the wheel rotation sensors are reasonably reliable during straight traverses but tend to be inaccurate during turns, when slip and granularity are both problematic. The robot is equipped with a map of its immediate environment, which consisted of a small collection of boxes arranged around it. We gathered data from 50 executions and arrived at a model similar to the one shown in Figure 1. The model shown in that figure is learned from both the training data and a further 40 runs containing manufactured errors (described below). It has been labelled with our *post hoc* interpretations of the states, which give an indication of the structure of the activity that the model reflects. However, it should be emphasised that the labels are neither definitive nor do they play any role in the actual use of the model.

In Figure 2 we show plots of three of the features constructed from the raw sensor data and, for comparison, the trace of visited states that was estimated by our online version of the Viterbi algorithm (Gough 2006) during the execution of the corresponding trajectory. The labelled phases of these figures (A, B, ...) are annotations made by

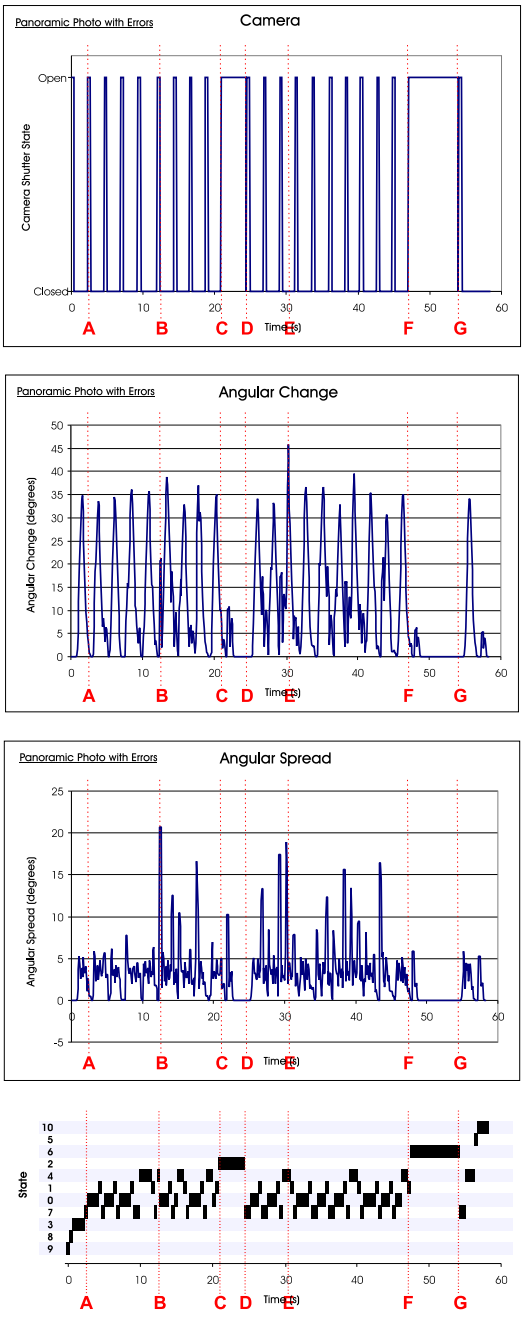

Figure 2: A plot showing the most likely sequence of states visited for an example trace, together with plots of three of the features for the same trace.

hand to highlight relationships between the figures. As can be seen, there is a striking correspondence between the phases of the state trajectories visited in the learned models and the patterns of data recorded from the sensors. What makes this relationship particularly interesting is that the annotation of the sensor data was carried out first, without reference to the state trajectory, and it was performed using human observed phases of behaviour timed independently. Thus, the relationship between these figures corresponds closely to human observed qualitative judgements of phases of behaviour and it reveals a concrete realisation of that behaviour in some of the sensor data and on into the learned model.

4 Monitoring Execution

Our intention in learning a HMM model of the behaviour is to use it in monitoring execution. When a series of observations is generated from sensor readings during an execution of a behaviour, the observations can be fed into the model, using the online Viterbi algorithm to find the most likely trajectory of states to explain the observation sequence. The sequence can then be assigned a likelihood, which is the probability that the model assigns to the particular trajectory it proposes to explain the observation sequence. The sequence of probabilities generated as more observations are considered will be monotonically decreasing, since longer sequences reside in progressively larger spaces of possible trajectories. For long sequences, these values are very small and will typically underflow the accuracy of floating point representations. As a consequence, we work (as is usual) with log-likelihood measures. We have observed that the pattern of developing log-likelihood across a trajectory falls within an envelope for successful traces, while traces generated by failing executions lead to abnormal behaviour. This is illustrated in Figure 3, where we show a divergent trace against a successful trace.

Our data was collected in a series of batches: 50 training executions, 20 verification executions and 40 error executions, split into four groups of 10 executions for different errors. The training data was used to learn the HMMs, and the verification data was kept separate so that the learnt models could be tested. The error data consisted of executions in which a specific type of error was induced. Figures 4 and 5 show the envelope for the log-likelihood values on the training data (which was used to build the HMM in Figure 1) and the log-likelihood values for the subsequent verification data set. There is one execution in the training data that took about twenty seconds longer to complete than the rest, and this protrudes past the end of the rest of the data. The most likely

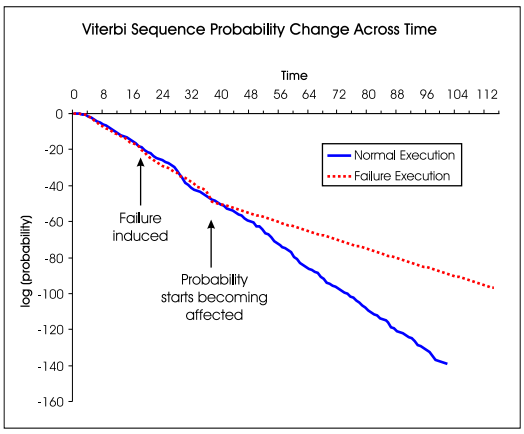

Figure 3: Comparison of the log likelihood values for sequence of observations for a successful and an unsuccessful execution trace.

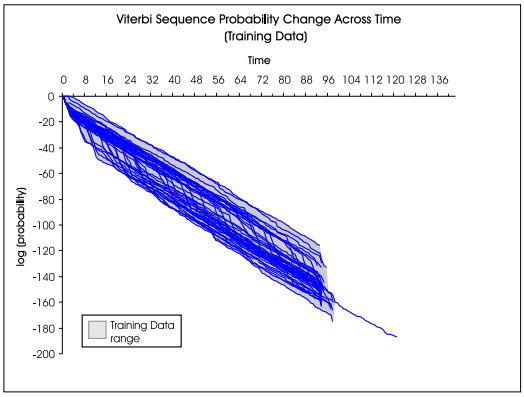

Figure 4: The Viterbi sequence probabilities for the training data

reason for this single execution taking longer is a build up of errors that led to a series of localisation steps that did not correct the error entirely.

It should be noted that the verification data was collected in an entirely reconstructed environment, rather than simply being a subset of the data collected during training. This helps to explain why the traces in the verification data are not evenly placed within the training envelope and also serves to emphasise the robustness of the learned model. The difference in the verification environment was significant (it was constructed in a different laboratory, with quite different physical dimensions and different floor covering), so the fact that the model continues to provide a good characterisation of the underlying behaviour in this task is a significant validation of its performance.

For all of the error executions, data was collected up to the point that the task finished or long enough to allow a reasonable algorithm to detect

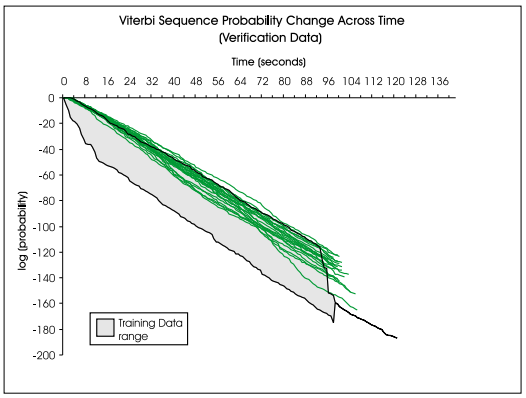

Figure 5: The Viterbi sequence probabilities for the verification data

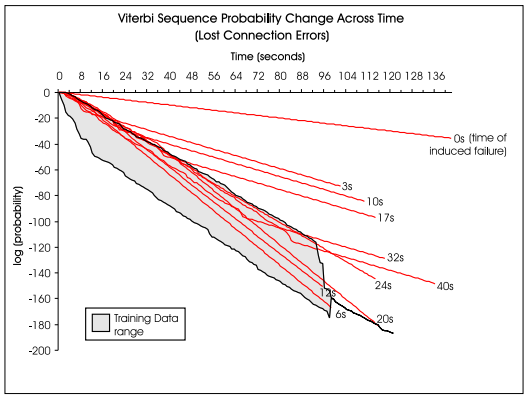

Figure 6: The Viterbi sequence probabilities for the 'Lost Connection' error data

an error. The errors induced were as follows:

Lost Connection The radio connection between the controlling computer and the robot was disconnected, meaning that the robot stopped receiving commands and could not transmit new sensor data.

Blocked The robot was trapped so that it was unable to turn to the next angle to take a photograph. This is to simulate the robot becoming blocked by some environmental factor.

Slowed In this data, the robot was deliberately slowed down by exerting friction on the top of the robot. This caused the robot to turn much more slowly than normal.

Propped Up This set of data was collected to simulate the robot "bottoming-out" by the wheels losing contact on the ground on an uneven surface. The front of the robot was propped-up on a block so that the wheels could not make the robot rotate. The execution continues as normal as the robot wheel sensors indicate that it is still rotating, however there may be an extreme number of localisation steps as it tries to correct the errors of inconsistency in its sonar readings. Eventually the robot's localisation cannot keep up with the errors and it becomes wildly inaccurate.

We now consider the data collected from these failing trajectories.

Lost Connection Errors (Figure 6) The consequence of terminating the connection is that the robot cannot report new sensor data, and the last known values are used instead. These values are repeated until the action is manually terminated. In the Figure, the times at which the failures were induced are indicated at the end of each line. When the failure was induced at 0 seconds, the probability decreases at a very slow rate from the start. This is a behaviour that was not seen in any of the training data. In five of the remaining nine cases the probability decreases roughly the same rate as the training data up to a point, and then breaks off before decreasing at the slower rate. The other four cases (failures induced at 6, 12, 24 and 20 seconds) show no obvious changes in the rate of probability decrease, and all appear with probabilities similar to the training data. It may seem strange at first that in the majority of the failure cases the probabilities are higher than the training data, but this is to be expected. The probability reported by the Viterbi algorithm is the chance that a particular sequence of observations produced a particular output sequence of states. In the case of these errors, the algorithm is simply more confident that it has the correct sequence of states for the given observations. It is important to realise that the higher probability that the Viterbi sequence explains the observation sequence is not to be interpreted to mean that the sequence itself is more likely, but rather that *given the observation sequence* the generated state trajectory is more likely. With thought it is easy to see that a repeated observation is best explained by a repeated visit to a single state. For the model that was used (which can be seen in Figure 1), these state sequences tend to be represented by a repetition of State $\overline{5}$.

This is the state that can occur just before the end of the model, and has a high self-transition probability of 0.68. In one of the error cases, State $\overline{2}$ was repeated in the same manner, which also has a high self-transition probability of 0.75. As would be expected, the HMM is seeking out the best state (or states) to repeat in order to maximise the probability of the sequence.

Blocked Errors (Figure 7) As with the error cases in which the connection was terminated, these executions have probabilities that deviate

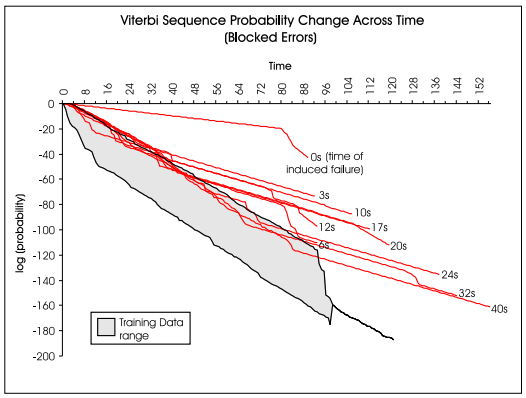

Figure 7: The Viterbi sequence probabilities for the 'Blocked' error data

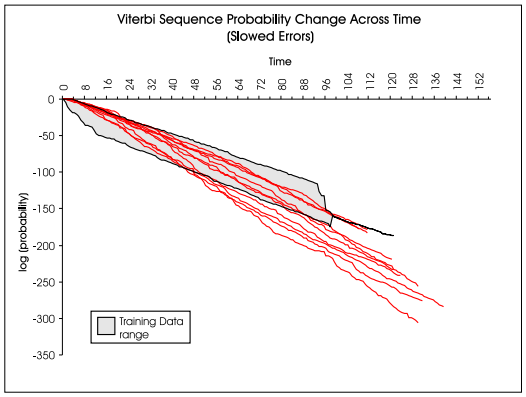

Figure 8: The Viterbi sequence probabilities for the 'Slowed' error data

upwards from the training data. The difference here is that *all* of the sequences exhibit this behaviour, rather than simply the majority. One explanation for this could be that the data reported from the sensors in this case will always indicate that the robot is barely moving, while the lost connection will lead to repetition of whatever sensed data was last detected.

Slowed Errors (Figure 8) Since the robot is being slowed down during turning, the action takes much longer to complete than the normal executions. In these error cases, the probabilities decrease at a greater rate than normal, proceeding to a minimum probability of around 10^{-300}, compared to a minimum of 10^{-120} for the training data. The very low probabilities of sequences are due to the fact that the most appropriate HMM sequences in these cases loop on states that have low self-transition probabilities. Such trajectories are highly unlikely to occur.

Propped Errors (Figure 9) The data collected from these error executions do not show any apparent difference to the training or verification data. The probabilities reported are within acceptable limits and it is impossible to distinguish these executions as possessing any anomalies.

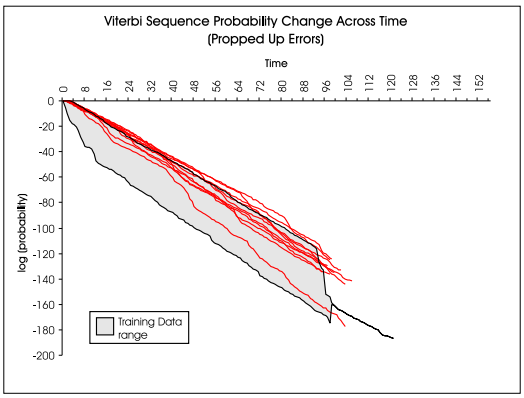

Figure 9: The Viterbi sequence probabilities for the 'Propped Up' error data

5 Automatic Error Detection

The method we propose for automatic detection of anomolous execution traces is based on an an approach we call *Cumulative Log Probability Difference*, or CLPD. The CLPD measures how far the sequence has wandered outside the range defined by the maximum and minimum probabilities seen for the training data. If the sequence wanders outside the boundaries seen for a particular timepoint, the difference between the log(probabilities) is summed. By definition, the training data always remains in this range and will always have a CLPD of 0.

More formally, with the following:

$$\begin{aligned} p_x &= \log\big(\text{prob. seq. at } x\big) \\ m_x &= \max\big(\log(\text{prob. train data at } x)\big) \\ n_x &= \min\big(\log(\text{prob. train data at } x)\big) \end{aligned}$$

CLPD is defined:

$$LPD_x = \begin{cases} 0 & [n_x < p_x < m_x] \\ (p_x - n_x) & [p_x < n_x] \\ (m_x - p_x) & [p_x > m_x] \end{cases}$$

$$CLPD_x = \sum_{i=0}^{x} LPD_i$$

Figure 10 shows the CLPD of the verification data, plotted on a log scale vertically for clarity. Note that all but two of the executions have CLPDs of below 500[1], suggesting that in this task

[1]One execution exceeds a CLPD of 500 just before the end of the task, when there was only one training execution of this length. There was little evidence for the possible spread of values at this timepoint causing CLPD to rise sharply after this point. A larger training sample would probably have removed this problem.

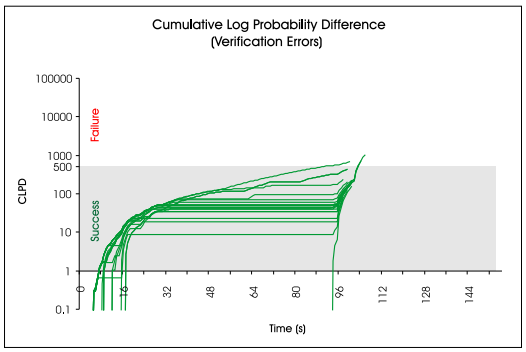

Figure 10: The CLPD values for the verification data

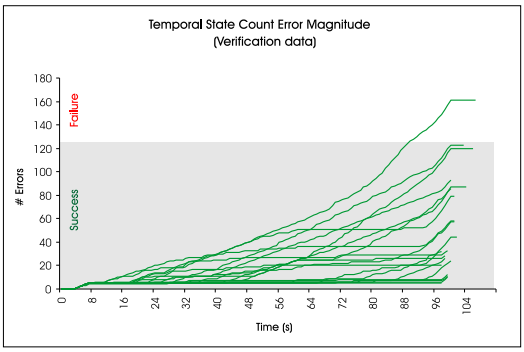

Figure 12: The TSCEM values for the verification data

a CLPD of 500 or lower is a good indicator of successful execution. Once a sequence has a CLPD of over 500 it can be identified as having failed.

Figure 11 shows the times at which the CLPD value exceeds 500 in each of the anomalous traces. Note that most of the errors in the 'Lost Connection', 'Blocked' and 'Slowed' data were detected before the action terminated, but only one of the 'Propped Up' errors was detected. The latter error type is much more difficult to characterise for this robot in terms of a physical behaviour that is different to a normal execution, because of its limited sensory capacity. Indeed, a human presented with the raw data of one of these executions and a normal execution would be unlikely to distinguish between the two. It had been hoped that there would be a difference in the amount of localisation required for this error type, but it seems that the sonar data and subroutines for localisation are not accurate enough to provide meaningful data when such errors occur. Had the robot been equipped with a more accurate localisation device (such as a laser range-finder) as well as a more sophisticated algorithm then these errors might have been detected. We intend to explore this hypothesis with a more sophisticated robot in the future.

6 Temporal Anomaly Detection

We now consider an alternative approach to anomaly detection. Temporal Anomaly Detection attempts to identify Viterbi sequences with an anomalous number of occurrences of states (either too few or too many) in comparison with the numbers of occurrences of states in successful trajectories. To measure the amount of error, a similar technique to the probabilistic anomaly detection above is used. The difference between the observed number of occurrences of each state and the expected number of states at each timepoint is summed. This may be formally defined as follows:

$$\begin{aligned} c(\tau, s, x) &= \#\text{ occs. } s \text{ to time } x \text{ for trace } \tau \\ m(s, x) &= \max_T\big(\#s \text{ to time } x\big) \\ n(s, x) &= \min_T\big(\#s \text{ to time } x\big) \end{aligned}$$

where "$\#s$" is the number of occurrences of state s in the states for the corresponding trajectory and the maximum and mimumum values (for $m(s, x)$ and $n(s, x)$) are defined of the range of traces in T, the set of all training data.

The error magnitude for state s at timepoint x is defined to be a measure of how far the current execution deviates from the training data:

$$e(s, x) = \begin{cases} 0, & [n(s, x) < c(s, x) < m(s, x)] \\ n(s, x) - c(s, x), & [c(s, x) < n(s, x)] \\ c(s, x) - m(s, x), & [c(s, x) > m(s, x)] \end{cases}$$

Temporal State Count Error Magnitude (TSCEM) for a sequence at timepoint x is defined to be the sum of all error magnitudes across all states up to that timepoint:

$$TSCEM_x = \sum_{i=0}^{x} \left(\sum_{s=0}^{t} e(s, i) \right)$$

where t is the number of states in the HMM.

Figure 12 shows the TSCEM values for the verification data. Note that the magnitude of the errors is usually below 100, and only one has a TSCEM value of over 125. Because of this, we consider a TSCEM value over 125 as an indication of failure for this task. Figure 11 also shows the performance of our algorithm based on Temporal Anomaly Detection in our examination of the test data. As with Probabilistic Anomaly Detection, identification of the 'Lost Connection', 'Blocked' and 'Slowed' errors was very successful. However, this technique detected the errors more reliably than the CLPD technique. The technique was less successful with the 'Propped Up' errors.

A possible refinement would be to find the distribution of occurrences of each state at each timepoint, rather than simply the maximum and minimum. The sum of the number of standard deviations by which each state-

Time of induced error (s)	'Lost Connection' error detected (s)		'Blocked' error detected (s)		'Slowed' error detected (s)		'Propped Up' error detected (s)	
	CLPD	TSCEM	CLPD	TSCEM	CLPD	TSCEM	CLPD	TSCEM
0	30.4	23.2	30.4	23.2	75.2	37.6	—	64.0
3	53.6	28.0	64.8	27.2	82.4	50.4	—	—
6	—	36.0	—	36.8	70.4	47.2	—	96.8
10	64.0	40.0	71.2	43.2	111.2	66.4	—	—
12	—	48.0	68.0	48.8	94.4	75.2	—	—
17	78.4	57.6	80.0	59.2	106.4	78.4	—	—
20	—	62.4	80.0	63.2	112.0	84.0	—	—
24	106.4	74.4	99.2	78.4	—	96.0	—	—
32	104.0	51.2	102.4	92.0	—	100.0	—	—
40	105.6	—	106.4	—	—	93.6	96.8	—

Figure 11: Probabilistic and Temporal Anomaly Detection performance across the error data. Time for a successful run was around 100 seconds in the training data and 110 seconds in the verification data.

count varies (Z-score) could be taken instead of the TSCEM value. This is defined as follows:

$$\mu(s,x) = \text{mean}_T\big(\#s \text{ to time } x\big)$$
$$\sigma(s,x) = \text{stddev}_T\big(\#s \text{ to time } x\big)$$

The normalised error magnitude for state s at time x is then:

$$NSCEM_x = \sum_{i=0}^{x}\left(\sum_{s=0}^{t}\frac{c(s,i)-\mu(s,i)}{\sigma(s,i)}\right)$$

Using this value could provide better error detection, but at the cost of more training data required to find accurate distributions for each state at each timepoint. This method remains a subject for further research.

7 Error Detection Evaluation

Temporal Anomaly Detection detects errors earlier, and detects more of them. Temporal Anomaly Detection successfully identified 30/40 errors, while Probabilistic Anomaly Detection identified only 24/40. The combination of techniques, however, is a more reliable test than either test individually, allowing us to correctly identify 33/40 cases, including all of the first three types. As commented earlier, we believe that the 'Propped Up' error type is hard to find given the nature of the sensors available to this robot.

8 Conclusion

When an executive system is required to turn a plan, constructed from abstract action models, into execution, while sensing its environments through imperfect sensors, there is no direct way to detect when the execution of an individual action has failed. We have demonstrated that it is possible to construct models that bridge the gap between the low-level sensory data streams and the higher level abstract action models and to use these models to monitor execution. In doing so, we are able to detect failures in execution, often anticipating the point at which the action might otherwise have completed execution. The approach we have described is complementary to the use of direct sensor interpretation methods and represents a form of model-based reasoning (Williams & Nayak 1996).

We believe that our approach can be used not only in monitoring the execution of actions for robotic systems, but also to monitor the behaviour of systems, both engineered and natural, using models learned from data gathered from monitored traces of their behaviours over time. This work is already progressing in application to condition monitoring of plant items and will be reported in future work.

References

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum Likelihood from Incomplete Data via the EM algorithm. *Journal of the Royal Statistics Society* 39(1):1–38.

Forney, G. D. 1973. The Viterbi Algorithm. *Proceedings of the IEEE* 61:268–278.

Fox, M.; Ghallab, M.; Infantes, G.; and Long, D. 2006. Robot Introspection through Learned Hidden Markov Models. *Artificial Intelligence* 170(2):59–113.

Gough, J. 2006. *Opportunistic Plan Execution Monitoring and Control*. Ph.D. Dissertation, University of Strathclyde, UK.

Jelinek, F. 1976. Continuous Speech Recognition by Statistical Methods. *Proc. of the IEEE* 64:532–536.

Liao, L.; Fox, D.; and Kautz, H. 2004. Learning and Inferring Transportation Routines. In *Proceedings of the 19th National Conference on AI (AAAI)*, 348–354.

Rabiner, L. R. 1986. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine* 4–16.

Rabiner, L. R. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE* 77(2):257–286.

Williams, B. C., and Nayak, P. P. 1996. A Model-based Approach to Adaptive Self-configuring Systems. In *Proceedings of the 13th National Conference on AI (AAAI)*, 971–978.