Opportunistic Plan Execution Monitoring and Control

A thesis presented for the degree of Doctor of Philosophy

Jonathan Gough, 2007
Department of Computer and Information Sciences

University of Strathclyde

Typeset in Charter and AvantGarde with the $\LaTeX\,2_\varepsilon$ Documentation System.

Forsan et haec olim meminisse iuvabit

Contents

Abstract

When executing a plan, the traditional assumption of complete certainty is rarely valid; in real-world situations, actions may fail to complete, or may take much longer than initially anticipated. Because of this, layers of robustness are required that allow flexibility during execution.

There are two aspects of an executive that need addressing to provide the ability to deal with uncertainties: monitoring and control. This thesis examines both of these aspects and shows how these may be used during execution.

For monitoring, the use of Hidden Markov Models is proposed. Tracing the execution of a task through a model allows the executive to detect its current state, even when this is not directly observable from within the system. The trajectory of states through the model can reveal execution irregularities, which can be flagged and identified as failure. This work explores the use of learnt models, evaluating their performance in various contexts.

For control, the use of opportunistic plans is examined. These can be used after failure has occurred, or when an action successfully finishes early. The definition of opportunistic plans is expanded to include extra structures that may be of use during the creation of these. Execution issues surrounding opportunistic plans are addressed, including how to decide when to execute an opportunity. The logical requirements for this are discussed, and several opportunity insertion strategies are developed. This is followed by a technique for deciding if resources should be conserved for later opportunities with the prospect of greater reward.

Good fortune is what happens when opportunity meets with planning

— *Thomas Edison*

CHAPTER 1

Introduction

It is possible to fail in many ways... while to succeed is possible only in one way

—*Aristotle*

THE RESEARCH area of planning has advanced significantly since the first planners were conceived over 35 years ago. Modern planners are now capable of ordering many hundreds of actions to produce highly complex plans with interlinked actions and dependencies. Although working on plans at a theoretical level such as this is important, the issue of real-world execution must also be addressed to ensure the usefulness of plans. Unfortunately, methods of executing plans in real-world situations have not progressed at the same rapid pace that has been seen with planning technologies. There are now many types of plans that can be easily generated, but cannot be easily executed due to the uncertainties present in the real world.

One of the biggest challenges with real-world execution is the problem of uncertainty; it can never be guaranteed that an action will complete on time, or will accomplish all of the effects that it set out to achieve in the first place. Environments are also often dynamically changing, and there may even be unmodelled entities that can interfere with the plan. With this in mind, there are two aspects of execution that must be addressed if these uncertainties are to be confronted:

Monitoring

Tracking execution of a plan to detect any deviation from the norm

Control

Reacting appropriately to issues that arise during execution

1.1 Content Summary

The work presented here builds on the research areas of model-based reasoning as well as plan execution under uncertainty. This thesis contains a proposal for a novel method of monitoring plan execution through learnt models of tasks. A second aspect of this thesis concerns a method of plan execution control using opportunistic plans, and how to best use the resources available to the executive. Specifically, this thesis:

- Identifies suitable tasks from which to learn models for use in evaluating failure prediction
- Builds on the work in [Fox et al., 2006] and uses this to construct Hidden Markov Models (HMMs) for the tasks
- Investigates how these HMMs can be used during execution to track progress
- Proposes and empirically evaluates methods for using HMMs to detect execution failure
- Outlines how opportunistic plans may be used in situations of uncertainty, and how this relates to failure
- Compares and contrasts opportunistic planning with other plan types where uncertainty is present
- Greatly expands the definition of opportunistic plans, allowing for much richer plan structures
- Identifies methods to allow opportunities to be inserted safely into a plan
- Describes a method that allows the best use of resources to maximise utility during execution of an opportunistic plan

1.2 Chapter Summaries

The content of this thesis is divided into chapters as follows:

1.	**Introduction**	*This chapter.*
2.	**Motivation**	Outlines the need for the methods and research contained within this thesis, as well as a discussion on the benefits that the work will give.
3.	**Planning / Execution Strategies**	An introduction to uncertainty in planning, including an assessment of current work on how to deal with uncertainty in execution and control.
4.	**Introspection**	This chapter focuses on the problem of **monitoring**, and addresses the issue of how an executive can determine its own system status, even when this is not directly observable. Suitable tasks are identified for analysis, as well as how the data collected from these can be used to learn HMMs.
5.	**Evaluation 1**	The results from the tasks discussed in Chapter 4 are analysed in depth. This includes analyses of the raw data from individual executions of the robot, as well as the HMMs produced from the data. The raw data is then fed into the HMMs and the resultant sequences analysed. Patterns, structures and other useful features are identified along the way with an in-depth discussion of such structures.
6.	**Failure Detection**	Methods are explored for using the HMMs learnt in Chapter 5 to predict task failure.
7.	**Evaluation 2**	In this chapter, the failure detection methods developed in Chapter 6 are evaluated for their usefulness. This is done by examining the data from executions where errors were deliberately inserted.
8.	**Reacting to Uncertainty and Failure**	Opportunistic plans are discussed as a method of dealing with failure and uncertainty. This is followed by an investigation into the methods required to **control** the execution of opportunistic plans and how best to do so to maximise utility.
9.	**Conclusion**	A final summary of the work presented within this thesis.

CHAPTER 2

Motivation

Doubt is not a pleasant condition, but certainty is an absurd one

— *Voltaire*

THERE ARE very few situations in which there is complete certainty about actions and their outcomes. It could even be claimed that there is no such thing as certainty outside the realms of theoretical calculations. It therefore makes sense to expect uncertainty in the execution of actions when planning tasks to carry out within the real-world. An executive that is to carry out a plan and interact with the world should also be able to recognise the consequences of uncertainty when they occur, and then possess methods for dealing with them in a sensible manner. It should be able to make judgements about its own behaviour and react to these as necessary, adjusting its behaviour according to how the plan is proceeding. One way for an executive to make these judgements is to possess models of the expected behaviours. Tracing the execution of a task through these behaviours allows the executive to determine exactly how the task is progressing, if there are any issues, or if something is wrong within the system.

One research area in which autonomy is increasingly required is that of space exploration systems. These often have to deal with slow, unreliable and limited communications with Earth, and mistakes can be hugely expensive when things go wrong. The Mars Polar Lander [NASA, 1999] was set to study the long-term climate change on Mars, but failed to successfully land on the planet [NASA, 2000]. The probe was designed to use model-based reasoning to switch off the landing thrusters when the probe made contact with the ground. The model that was used to estimate the state of the system was constructed from the English-language requirements [Blackburn et al., 2002]. The cause of the failure is thought to be due to a jolt caused by the landing gear being lowered, making the model believe that the probe had landed. This resulted in the landing thrusters being shut down whilst the probe was still 40 meters from the ground, causing the probe to crash.

The Mars Polar Lander incident illustrates the fact that hand-coded models often do not capture the subtleties that are required for a model of a system. There is clearly a need, in some situations, for the automatic learning of models so that no aspect of the system and its behaviour is overlooked. By using real data to form these models a much more rich and complex model can be constructed, which can describe the system to a much greater extent than hand-coded models. Because the models are generated from the raw data from the executive, they are easily learnt and can be used on any executive that can record data in real-time.

Within this thesis, a method of using automatically learnt models for behaviour control is presented. Such models enable the executive to understand and make judgements about its behaviour, for example decide whether to continue or whether to terminate a task in the face of execution failure. The models presented here represent internal system states, abstracted away from the world, rather than the world itself. A representation at this level allows many aspects of a task to be covered that would otherwise not be directly observable.

Even when models are used for failure detection and behaviour control, the issue of reacting to failure still exists. Various architectures and plan structures have in the past been developed for this task, but usually suffer from problems of scalability (e.g. contingency planning), intractability (e.g. POMDPs) or ability to deal only with logical uncertainty (e.g. probabilistic planning). This thesis proposes the use of opportunistic plans for dealing with failure and uncertainty, once these have been detected. Opportunistic plans are scalable and can deal with uncertainty in resource consumption.

If future robots and executives are to be developed past the current technologies, greater amounts of monitoring and control are required. The effect of this will be to increase robustness and autonomy, and therefore the usefulness of systems. This thesis addresses both of the issues of monitoring and control, and shows how these can be used for effective plan execution.

CHAPTER 3

Uncertainty in Plan Execution

In preparing for battle I have always found that plans are useless, but planning is indispensable

— Dwight D. Eisenhower

EXECUTING A PLAN is not always an easy task. An executive sometimes has to deal with plans generated from an incomplete world model, which therefore cannot be executed as initially planned. Worse still, the planner may have an inconsistent model of the world causing plans to be generated that are invalid with respect to the environment. If an unmodelled entity interacts with the environment, removing an already-achieved precondition for example, then the plan will fail.

A system in which an executive is to carry out generated plans must take into account these problems to ensure that tasks are completed correctly, successfully and on time. Often, it will have to detect and rectify problems with plan steps on a local level, or perhaps report back to the planner when a locally-unfixable problem occurs. On an executive with relatively few sensors this may be a very difficult task and sometimes plan failure cannot be detected at a local level. There have been many attempts to create strategies at the executive level to solve one or more of the problems described above. This section explores and discusses some of these.

3.1 Uncertainty in Planning and Execution

There are essentially three types of uncertainty that have been explored and modelled in planning and execution. These can be categorised as follows:

Resource-consumption uncertainty

This type of uncertainty is to do with the unknown values of resource consumption (or production) inherent within a particular action. The actual values of consumption remain unknown until the action is entirely completed. The resource could be either a physical quantity such as fuel, a non-physical quantity such as memory remaining on a robot, or simply time available. Examples of this type of uncertainty include:

- How long it will take to drive somewhere, dependent on congestion on the roads
- The power generated from a solar panel, affected by how cloudy it is that particular day
- The amount of storage space required for a photograph, stored with a compression format such as JPEG

With resource-consumption uncertainties, such as the above, the consumption amounts of each action can be modelled as distributions rather than as the specific fixed values that are traditionally used in planning. Uncertainty in resources has proved very problematic in both the planning phase [Bresina et al., 2002] as well as execution [Pell et al., 1996].

Logical uncertainty

If it is not certain that an action will complete successfully in a particular state it is said to possess logical uncertainty. An action may have several possible outcomes, for example:

- A robot designed to pick up a cup may accidentally fail to pick up the cup with a probability of 0.05

In situations like this, each effect of the action can have a probability associated with it that it will become true.

Environmental uncertainty

This is where the planner has an incomplete model of the world, and local information about the environment around the executive may not be known until certain actions in the plan have been executed. In situations like this, the plan may contain sensing actions that resolve the uncertainties into known states. Worse still, the planner may have an inconsistent model of the world causing plans to be generated that are invalid with respect to the environment. An example of this type of uncertainty is as follows:

- A robot may need to navigate through an office, but doors may be closed, requiring extra actions to open the door to complete the navigation. The robot may have an action to sense if a door is open or closed.

A subset of this type of uncertainty involves the world being altered by external, unmodelled entities. This could include people being present in the office situation described above, opening and closing doors. These actions are totally outside of the control of the planner and executive.

Uncertainty is typically either dealt with by trying to minimise (or even completely remove) the amount of uncertainty, by trying to convert one type of uncertainty to another, or by planning for every possible outcome. Strategies can usually be categorised into two approaches: those that attempt to deal with uncertainty at the planning level, and those that deal with it during execution.

One area of planning and execution research that has been particularly fruitful with respect to uncertainty is that of robotic space exploration. This has produced a wealth of research into methods of dealing with the problems associated with planning and execution under uncertainty [Muscettola et al., 1998]. This is because failure in these situations can have catastrophic effects, potentially leaving a multi-million dollar craft stranded with no hope of recovery. Communications windows may also be very short and sometimes only limited to a single unidirectional communication per day. Because of these limitations, a certain degree of autonomy allows a craft to accomplish significantly more than it would if it were not present. In 1997, the Mars Pathfinder lander (accompanied by its mobile planetary rover Sojourner) [Matijevic, 1996] illustrated many of the problems that face remote operation in uncertain environments. The Sojourner rover possessed a radiation-hardened CPU that ran at 2 MHz, and consequently lacked the processing power for replanning on plan failure. Because of this shortage of computational ability, plans were represented simply as timestamped actions. This technique however led to a considerable of downtime, which has been estimated to have been between 50–70% of the total time available — all resulting from plan failure [Horstmann and Zilberstein, 2003]. Despite this, the mission was regarded as a huge success and accomplished much more than anticipated. The follow-up twin MER rovers, Spirit and Opportunity, possessed a greater amount of autonomy [Volpe, 2003] and achieved significantly more, although still taking up to three days to visit a single rock [Mausam et al., 2005]. Plans for even more autonomy aboard future Mars missions are currently underway [Volpe, 2005, Pedersen et al., 2005].

3.2 Recognising and Diagnosing Execution Failure

Even when safeguards are in place to minimise the effects of uncertainty, it can often lead to situations in which the plan is no longer valid, causing execution to fail. The capability to recover from failures first requires the ability to detect that an error has occurred, and then possibly diagnose the cause and solution.

3.2.1 Model-Based Reasoning

Reasoning about the state of a system through the use of system models is a popular way of establishing the correct real-world state. This typically works by tracing execution through a model whilst recording observations from sensors. If the sensor values do not match those predicted by the model then the model may be used to correct the executive's beliefs about the world.

One reasoning system that has been successfully deployed in space operations, including the highly successful Deep Space 1, is Livingstone [Williams and Nayak, 1999]. This system uses hand-coded models to track the physical system state. The models are representations of the interconnected system components, and how the system should respond to particular inputs. Sensor data is fed into the models, which calculate the expected outputs of the system given these. If there are discrepancies between expected outputs and the actual observed outputs, then possible explanations for can be generated. There is a search mechanism that explores the components and their relationships between one another, aiming to identify the component that has most likely failed. Livingstone also has the ability to exploit redundancies in the physical executive. For example, if a fuel valve on a space probe fails, then it may re-route fuel through a secondary backup valve. This reconfiguration ability allows a certain amount of robustness in the system.

Techniques for using models have been investigated for many situations, including modelling and tracking human behaviour [Demeester et al., 2003, Liao et al., 2004], robot navigation [Bui et al., 2002, Fox et al., 2006] and then in more general cases

[Kosaka and Kak, 1992]. Intermediate languages (see Section 3.3.3) have also been integrated into model-based executives [Williams and Gupta, 1999].

3.3 Dealing with Uncertainty

Uncertainty may be dealt with in many ways, all dependent on the types of uncertainty that are present in the target domain. These techniques include using different logical representations [Pettersson, 1997, Dix et al., 1990], probabilistic reasoning [Schaffer et al., 2005], or even simply planning for every possible outcome of an action, for example as the technique of Contingency Planning.

3.3.1 Contingency Planning / JIC approach

Contingency planning is an approach which generates multiple plans for many or even all eventualities. These are constructed in a tree-like structure, and executed until a branch is reached. At such a point, the executive chooses the correct branch given its local knowledge of the current environment. A disadvantage of contingency planning is that it suffers from problems of scale: the number of plan steps increases exponentially if multiple outcomes are to be planned for. This branching can produce overly complex plans, and plans have to be calculated for all eventualities, no matter how unlikely they are. Also, in many situations an executive will not have the memory capacity to hold such large plans either.

The "Just In Case" approach (JIC) [Dearden et al., 2003] is an attempt to reduce the problem of scalability by iteratively identifying the most likely possible failure points and adding contingency plans at these locations. This is essentially a cut-down version of contingency planning with reduced scalability issues. JIC planning is more tractable than the traditional approaches to contingency planning as it concentrates efforts on producing a limited number of contingency branches. Unfortunately, although this approach can be scaled to larger plans, the scalability issue still exists, and what is gained in scalability is lost in robustness: there is a cut-off point at which below a certain threshold particular eventualities will not be planned for. If one of these occurs then total replanning will be necessary.

3.3.2 MDP approach

Markov Decision Processes (MDPs) attempt to eliminate the need for a general plan, and instead represent it as a set of policies [Givan and Parr, 2001]. These policies tell the executive which action to execute for every possible system state. Utility values are associated with transitions between states, and the sequence of transitions with the highest utility can be chosen. This allows the most appropriate action to be taken. The maximal return in utility, however, is offset by the fact that the state space increases exponentially with the number of variables. This lack of scalability is a major problem for the MDP approach, and there has been considerable research into combating this and many of the associated issues [Feng, 2004, Lane and Kaelbling, 2002].

3.3.3 Action Packages

An "action package" is a sequence of low-level commands for the executive that represent a high-level action as used by the planner. It is possible to convert many types of logical uncertainty into resource-consumption uncertainty by means of action packages. The benefit of this conversion is that the executive is much less concerned with whether or not actions will complete, and therefore may concentrate on evaluation of other uncertainties. Action packages effectively serve as the 'glue' between *primitive plan actions* (as used by planning software) and *primitive executive actions* (the basic operators of the robot). These two terms are used here to distinguish between the building blocks of plans and the range of possible commands available to an executive. A simple primitive plan action may require several low-level executive commands in sequence that must be executed to complete the plan action. These can be constructed by hand then re-used during execution whenever required in a plan. In the example mentioned above, regarding the robot picking up a cup, there is the primitive plan action of `PICKUP_CUP`, which could translate to the following sequence of primitive executive actions:

```
OPEN_GRIPPER;
MOVE_ARM (CUP);
CLOSE_GRIPPER;
RAISE_ARM;
```

As well as primitive executive actions, action packages may also contain code for program flow and control. With this in mind, the `PICKUP_CUP` action could be potentially be modelled more robustly with the following pseudo-code:

```
int count = 0;
while (GRIPPER_EMPTY && count<3) {
    OPEN_GRIPPER;
```

(continued on next page)

```
        MOVE_ARM (CUP);
        CLOSE_GRIPPER;
        count++;
    };

    if (count==3) {
        FAIL;
    };
    RAISE_ARM;
```

This action package represents a program that would attempt three times to pick up the cup, otherwise fail. Earlier, the basic pickup-cup action was theorised to have a probability of completion of 0.95. The $\frac{1}{20}$ chance of failure has now been reduced to a much more useful $\frac{1}{8000}$ through the use of this simple action package.[1] Robustness could be further improved by adding conditional branches to deal with cases where the robot knocks the cup over, or the cup is slightly out of reach. If, in using this action package, the robot has not successfully picked up the cup after three attempts it is probably the case that something has gone significantly wrong, causing this repeated failure, possibly an environmental uncertainty manifesting itself (e.g. wrong size cup, cup not present etc.). Recovery from such a major failure can only be dealt with by replanning and is outside the scope of recovery by the robot itself. What has happened though is that by enclosing the action in an action package, the logical uncertainty that was initially present in the action has been greatly reduced. The action will now complete with a much higher degree of certainty, at the expense of an uncertain duration; it could succeed on any of the three attempts to pick up the cup. It must be noted that this is a greatly simplified example and real-world action packages would be much more complex, but the principle of converting logical uncertainty into resource-consumption uncertainty remains.

There have been many attempts at execution languages that use action packages, such as Reactive Action Packages (RAP) [Firby, 1989], Reactive Model-based Programming Language (RMPL) [Ingham et al., 2001] and also the Task Description Language (TDL) [Simmons and Apfelbaum, 1998]. A comparison of execution languages can be found in [Gregory, 2001]. Attempts have also been made to integrate the expressibility of these action packages into planning / execution frameworks [Bonasso et al., 1997, Gat, 1992, Volpe et al., 2001].

[1]This is a theoretical value. In reality this would be of a higher probability as the probabilities of failure at each attempt would not be independent

Although action packages provide a reliable method for making specific actions robust, this is not always the most preferable approach. If primitive plan actions are dealt with singularly, then the system can lose sight of the specific task in hand, and concentrate on the individual actions that are to be carried out. For example, consider the situation in which a robot arm were to move a block from a tower to the surface of a table. If the robot were to accidentally lose its grip on the block whilst picking it up, the block may drop onto the table, completing the plan in a somewhat unexpected manner. If an intermediate language were used, it may attempt to finish the pickup action by re-attempting to pick up the block, before eventually returning it to the table on the next action. The second attempt to pick up the block in this case is superfluous. This is a very basic example, but it would have a much greater significance when applied to a large and complex plan with many interactions.

3.4 Conservative Planning

Conservative planning seeks to address the issue of resource-consumption uncertainty. This is achieved by using very conservative resource allocations for each action in the plan. A plan is generated by considering the distributions of the action's resource consumption and creating the plan by using (for example) the 95th percentile of the distribution. This ensures that enough resources are available, in most cases, for the plan to complete. For example, if an action has a duration that follows the Normal distribution $N(20,5)$ seconds, then the action is assumed to take the 95th percentile amount of time to execute, which is 28.27 seconds. This is the value that is fed into the planner to use for plan generation.

Traditional planning and execution frameworks take the schedule generated by the planner as fixed and rigid, and actions are executed when the exact time-point occurs. In domains with resource-consumption uncertainty, planning using such time-points becomes a futile task — it is almost certainly the case that the schedule will change during execution due to the variability in the durations of the actions. Conservative planning therefore seeks to represent plans as orderings rather than timestamped actions. Adopting this methodology means that action preconditions and effects must be carried down to the executive level to ensure that these are not violated at run-time (see Section 8.5). This type of problem can occur in non-linear plans where actions have dependencies on several other actions.

In Figure 3.1, Actions 1, 2 and 3 have an uncertain duration. Action 1 has a precondition of $\neg P$, and an effect of P. Action 3 has a precondition of P. Action 3 must follow Action 2. In Case 1, the actions can execute as planned. If Action 1 were to take longer than planned (as in Case 2), then P is not true when Action 3 is required to start and

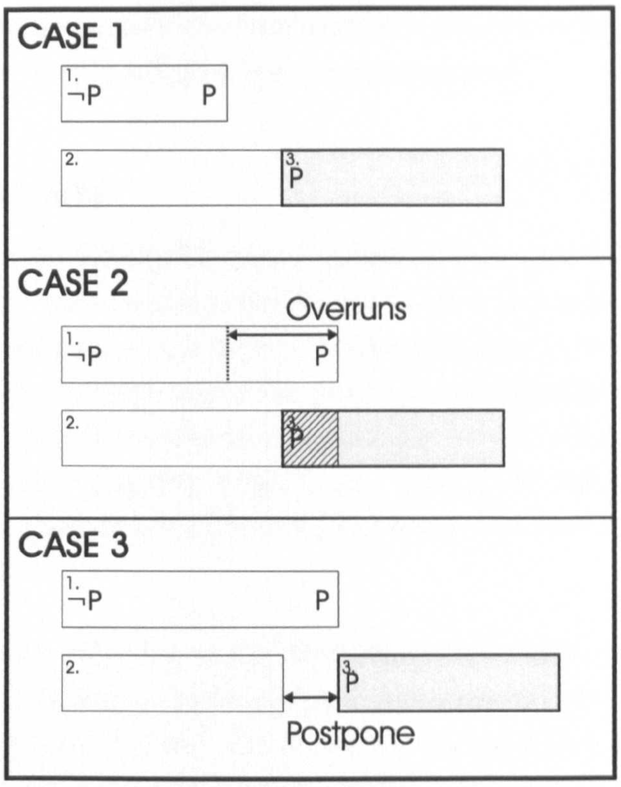


Figure 3.1: *How uncertainty in action durations can lead to plan failure*

the plan fails. To fix this, a partial ordering between Actions 1 and 2 must be imposed, and the execution of Action 3 delayed to ensure that this precondition is not violated, as shown in Case 3.

This method of representing actions as orderings allows the most to be made of any difference in action duration that arises, be it an action finishing early or a delay in action completion. The ordering effectively says *when* the action can possibly happen, and the preconditions and effects specify *if* it can happen. Combining these two allows the execution of conservative plans in the most efficient way possible.

One of the advantages of this approach is that if one action were to take an unusually long amount of time to execute (and over-run the 95th percentile mark), then it is highly likely that the time would be gained back from other actions executing in the plan due to the padding the 95th percentile provides. This would prevent failure of the plan as a whole, even though the individual action failed to complete in its planned duration. It would also probably be true that in the 5% of cases where the action duration exceeds the planned value, there is the potential of a fundamental failure in the plan execution. Such a failure may be irreparable from the executive's view of the world, or the executive may lack the computational power to resolve the failure. If this were to occur, replanning at a higher level may be required.

3.5 Execution Frameworks

Even when an execution strategy like those mentioned in Section 3.3 is used, failure can still sometimes occur. For example in the JIC approach, a step may be reached where a contingency plan wasn't generated for a particular situation. In this case it is necessary for the system to react in some way if the plan is to be successfully completed. To do this, an executive usually lies within an execution framework — a software architecture consisting of intercommunicating components. This allows the executive to receive updated plans from a higher level (be that a computer or a human operator) to repair the plan if required.

Most execution frameworks work on a layered approach, usually consisting of three layers. The idea of a three-layer architecture [Gat, 1998] was the dominant view amongst the AI community in the early 80s. The principle involved splitting the control system of the robot framework into three separate layers, each dealing with different parts of the task. The layers sat on top of each other and could communicate with each other; plans were passed down from planner to executive, and sensed information could be passed back up through the layers to provide a feedback loop. The top layer includes the planner, generating the theoretical instruction steps from a model of the world. The bottom layer usually contains the executive, where the theoretical actions get transformed into output. Often, an intermediate layer is used, which can be used to encode error-correction and detection routines, and can also go some way towards emulating a contingency-plan-style solution. This helps to ensure the robustness and correctness of the plan execution process. Often, an intermediate language is also used to precisely define how the intermediate layer is to behave, and may be constructed from action packages.

The concept of a three-layer architecture is still commonplace today, but it has been extended and built into more useful and mature frameworks. A few of these are outlined below:

3T architecture

The 3T architecture [Bonasso et al., 1997] is a three-layer architecture that is built around RAPs (see Section 3.3.3). It is a very mature architecture and has been in development for over eight years. The architecture has been implemented on a wide variety of very different systems with different operating systems and capabilities. A number of software tools have also been created to allow the use of the 3T architecture on real-world robots.

CLARAty

The CLARAty (Coupled Layer Architecture for Robotic Autonomy) architecture is a two-layer architecture, which makes a deviation from the standard approach of three layers for robot control [Volpe et al., 2001]. The argument for compressing three layers into two emerges from shortfalls proposed in the three-layer approach. These include the fact that most architectures seem to give dominance to particular layers, leaving others redundant. Another problem that led to the development of the CLARAty architecture is the lack of access from the top layer to the bottom layer in the standard three-layer model: access can usually only be gained by interaction through the intermediate layer.

ATLANTIS

The ATLANTIS framework presents itself as "a heterogeneous, asynchronous architecture for controlling autonomous mobile robots which is capable of controlling a robot performing multiple tasks in real time in noisy, unpredictable environments" [Gat, 1992]. ATLANTIS works by attempting to combine a traditional planner with a reactive control mechanism.

The key relationship between these execution architectures is that they all possess different forms and amounts of control and monitoring. These two aspects are critical in any system in which a certain reliability is required. The following chapter will examine one form of execution monitoring and how this may be evaluated.

CHAPTER 4

Introspection

If anything is certain, it is that change is certain. The world we are planning for today will not exist in this form tomorrow.

— *Philip Crosby*

IF AN EXECUTIVE is required to perform a task autonomously then it must be able to provide some degree of robustness to ensure the completion of that task correctly and in a timely manner. When robustness cannot be guaranteed, the usefulness of an executive decreases rapidly. The first step towards robustness is an ability to correctly detect failure of the task being executed. Once failure has been detected and identified, steps can be taken to correct the failure.

There are many different types of controlling mechanisms that may be used for executives. These range from completely reactive control systems, dealing with low-level sensor data, through to abstracted control systems, dealing with high-level tasks with no reactive behaviour. In reactive control systems, the executive chooses the best course of action given the current sensor values. The executive simply reacts to the sensor values and does not attempt to analyse them. At the other end of this scale is an abstracted control architecture, where the robot is given high-level commands (possibly from a planner) and does not react to the environment in any way. These commands are executed strictly at fixed timepoints and there is no flexibility in the schedule. An introspective control system is a compromise between these two extremes. On one level, it reacts to sensor values, but also simultaneously attempts to analyse and reason with these to make judgements about the best course of action to take. These judgements are viewed in the context of the overall result that is required for the task being carried out.

Introspective monitoring can be used for failure detection and prediction. For this, the executive must have the ability to examine itself throughout the course of an action so that it can monitor its progress towards completion. The current state of the action, as well as success or failure, can be tracked to detect if the action is proceeding correctly, or if failure is imminent.

Failure situations can be notoriously difficult for an executive to detect because its internal representation of the environment may be inconsistent with reality. For example, a simple wheeled robot may bottom-out if traversing over a rugged terrain, causing the wheels to leave the ground and the robot to become stuck. The sensors will still be reporting that the wheels are rotating and therefore the robot will be updating its location accordingly using dead-reckoning. Simply given this sensor data, it is not possible to detect if the robot is moving or not. This is because the robot's internal response to the sensor data rests entirely on the assumption that the wheels are in contact with the ground, which in this case they are not. Adding some form of position localisation to the robot may cause the robot to correct its location continually and minimise errors [Dellaert et al., 1999], but will not detect the fact that something has gone wrong — the task of localisation is to simply update the robot's position as required. What is needed is a layer of abstraction on top of this with the ability to analyse the current sensor readings and detect any situations in which these deviate from expected values. This layer allows the executive to determine the state of the world (to a certain level of accuracy) according to some model. In the case of the robot, this may be a routine which detects if there is a consistent anomaly between the wheel-encoding values and localisation from sonar sensors: such a routine could detect if the robot was stuck and could pass on this information to a planner to form a strategy to free the robot. This ability to examine sensor data to determine the current state of the executive and its surroundings using a model is called *introspection*. The monitoring provided by introspection is not confined to easily apparent physical aspects of the system (such as if the robot is blocked and cannot move), but also other less-obvious properties (such as if one of the robot's actuators is operating at 50% efficiency, causing it to move slowly).

In any executive, there is always a model that describes how the fluctuations in sensor values reflect the changes in the physical executive. This model may be one that is only present in the mind of the programmer who constructed the task, and then hardcoded into the action. In such a case, the model is not explicitly available to the executive and therefore cannot be used for reasoning. In the example of the wheeled robot bottoming-out, as above, there is a model which relates the rotations of the wheels to the physical movement of the robot. This model in this situation simply indicates that when the wheels rotate, the robot moves by the distance of the number of rotations times the diameter of the wheel. In the situation above, the wheels had left the ground, causing the assumptions underlying the model to become invalid and therefore inaccurate. The assumptions could also have be invalidated in other ways, such as if the robot were travelling across a sandy surface, resulting in wheel-slippage.

An executive is usually controlled by a set of programmed behaviours that determine the choices it should make to guide it towards the completion of a task. Throughout the task, the executive will pass through various states where different behaviours are observed. For example, a robot with a task of picking up an object might exhibit the following behaviours:

- Object recognition
- Robot localisation
- Robot positioning
- Gripper localisation
- Gripper positioning
- Grasping
- Raising arm

Each of the above behaviours may be further subdivided into states representing fine-grained aspects. For example, there may be several states associated with "Robot positioning" that correspond to the different amounts of positioning required. Even subtler differences are possible, including differences between states relating to the distance to the nearest obstacle, or the current angle of the robot to the object. During execution of the task, the robot transitions between these states in complex ways. Figure 4.1 is an illustration of a purely theoretical arrangement of behaviours, states within these and the transitions between them.

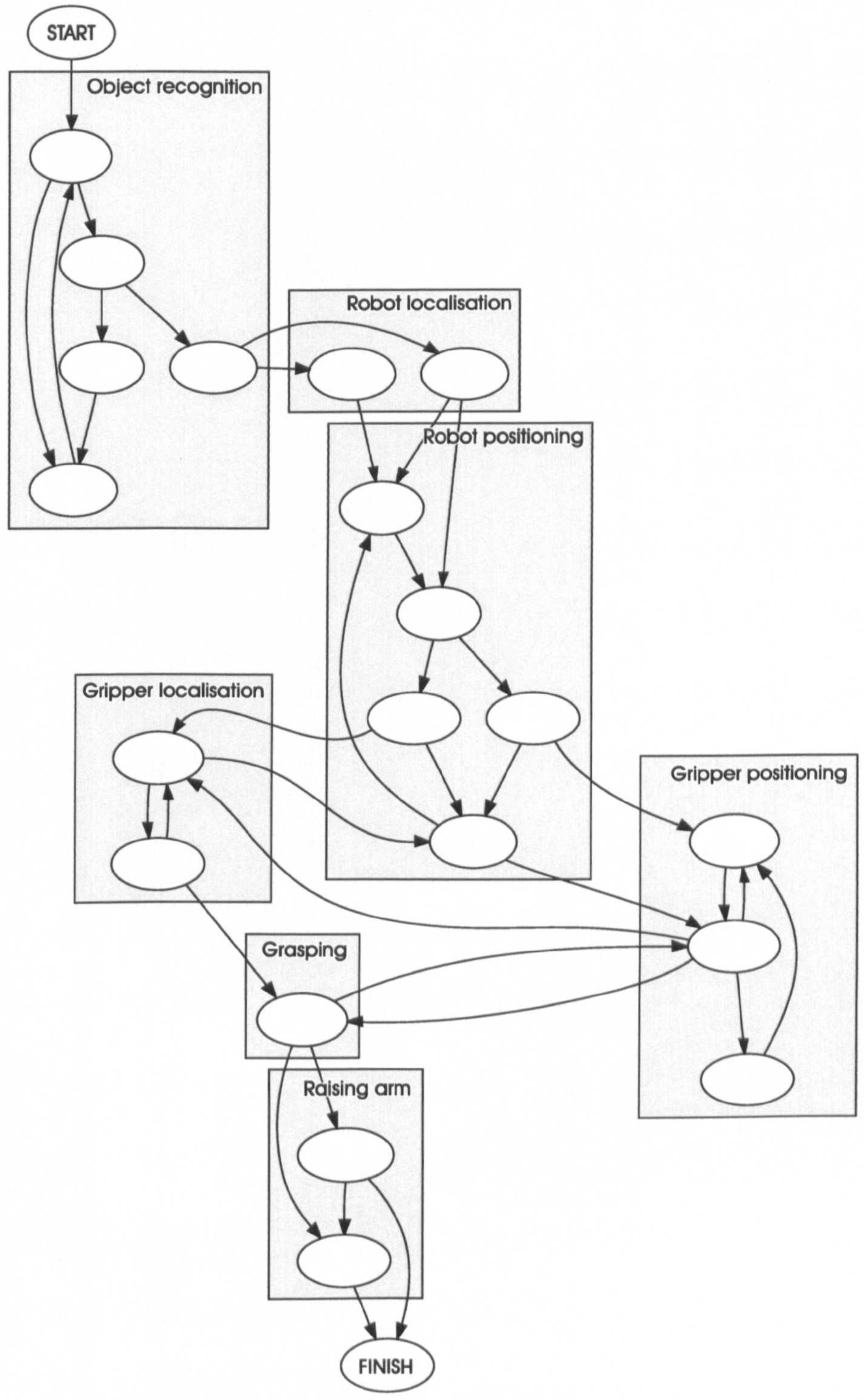


Figure 4.1: *An example theoretical model of states and behaviours representing the "pickup" action*

Introspection can be defined as the recognition of which particular behaviour or system state is manifesting itself at a specific instant during execution. With a small number of simple behaviours, the identification of these might be easily hand-coded. Even with a simple model though, the robot may exhibit states involving combinations of behaviours. With the above gripper robot example, it is possible to imagine states that involve simultaneously localising and positioning the gripper for added accuracy. These multi-state combinations would especially be present in a robot with a sophisticated multi-layered control architecture where behaviours interact, overlap or subsume one another. Such a complex model may have emergent behaviours that are resultant from combinations and interactions of the simple behaviours. These emergent behaviours cannot be classified in the same sense as the above list, and are often too complex to identify or classify manually. It is therefore not possible to detect such complex behaviours for introspection, and another method to identify these is required.

Models can be used for the detecting when execution does not proceed in a normal manner. The execution of a task can be traced through a model and any transitions or states that are 'out of character' for the particular action can be flagged and used as evidence that the action is not executing correctly. A large deviation from the expected behaviour predicted by the model should therefore be construed as a failure of that task.

A model must be generic enough to deal with all the different outcomes and possible executions of a particular task, but specific enough that it will not misinterpret a failure as normal execution. If the model is too specific then any small deviation from the norm will incorrectly be reported as a failure. On the other hand, if the model is too relaxed then the executive may find itself progressing into a situation where failure has occurred but has not been detected by the model. Getting a balance between these two cases is critical if the model is to be of use during execution.

There are many possible representations for models, as well as ways of acquiring these. Techniques include (amongst others) neural networks, dynamic Bayesian networks or Kohonen networks. In [Fox et al., 2006], a process is described in which models can be generated from data collected from a robot through the use of a Kohonen network. This work has been adapted for the rest of this chapter, which explores and builds on this to show how a particular type of model can be learnt automatically from data collected during a series of executions. This is followed by Chapter 5 in which the learnt models are analysed in depth. Chapter 6 expands on this and examines how these models can be used to predict failure.

4.1 Learning HMMs for Tasks

At the beginning of this chapter, some example behavioural states for a gripper robot were introduced. Note that these states may not be directly observable from the view of the robot because of limitations in sensors and inaccuracies of the model. These states can be imagined as having probabilities of transitions to other states. In essence, there is a Hidden Markov Model (HMM) with these states and the transition probabilities between them.

It might be possible to construct a HMM for a task by hand, perhaps by examining the source-code and building up structures in the HMM based on the control structures in the code. Such a model might give a very good representation of a single, simple task with no external interactions, but be highly inaccurate for real-world systems. Another factor to take into account is that when multiple behaviours are present, there will also be subtleties between the interactions of these that cannot be predicted or planned for. The robot's model of the world at any timepoint may also be incorrect (but recoverable from), leading to a deviation between the physical world and the predicted HMM state. In all of these cases, there are hidden states that cannot be guaranteed to be captured by hand-coded models. These states are not directly observable from the executive's point of view but can be represented through the use of HMMs. A representation at this higher level enables the executive to make judgements about these hidden states that would otherwise not be accessible. Since models possessing these hidden states cannot be easily hand-coded, it therefore makes sense to try to automatically learn these models so that many more aspects and eventualities of a task are covered.

In [Fox et al., 2006] there is a description of a way of automatically learning an HMM for a robot task. This process includes automatically classifying states and learning the most probable HMM from these states, given data collected from the robot. Figure 4.2 shows an overview of the technique. Raw data is collected from the robot, which is then processed, clustered and converted using into a series of *observations*. An in-depth explanation of this work is beyond the scope of this thesis, but an outline summary of this technique is provided in the next section with additional relevant details.

To assess the quality of the HMMs and their scope for predictive power, learnt models have to be evaluated in terms of how well they capture the actual behaviour of an executive. For this, a series of tasks were required from which to collect data for evaluation. These tasks were carried out by an ActivMedia AmigoBOT robot. This is a small, inexpensive robot with basic functionality including autonomous navigation and eight sonar sensors for obstacle avoidance and localisation from a pre-programmed map. The AmigoBOT is a two-wheeled robot that can turn on the spot, making it particularly

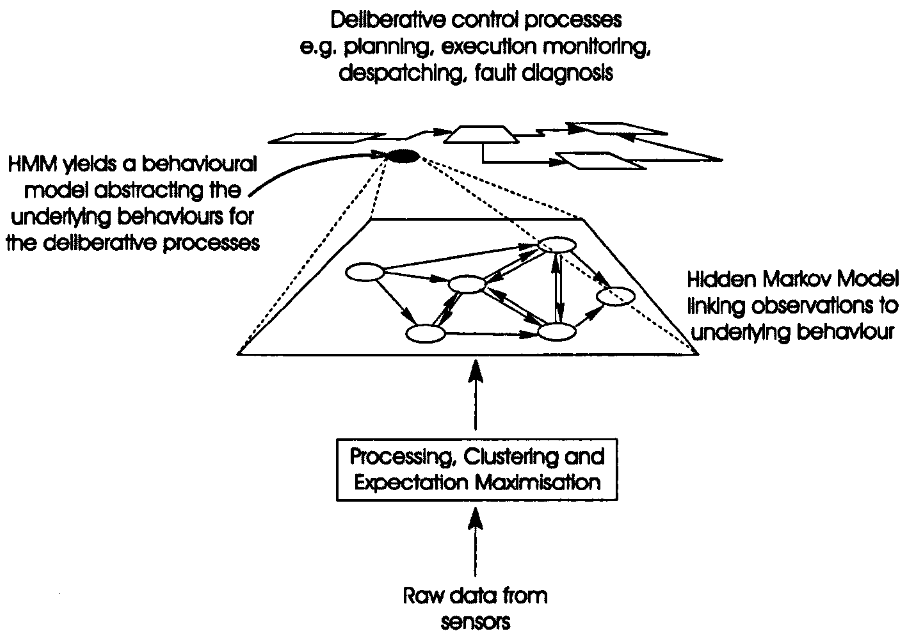

Deliberative control processes
e.g. planning, execution monitoring,
despatching, fault diagnosis
HMM yields a behavioural
model abstracting the
underlying behaviours for
the deliberative processes
Hidden Markov Model
linking observations to
underlying behaviour
Processing, Clustering and
Expectation Maximisation
Raw data from
sensors

Figure 4.2: *Learning an HMM (Figure reprinted from Fox et al. AIJ vol 170, no 2)*

suited for simple navigation tasks in small- to moderately-sized indoor environments. To test the learning process effectively, careful decisions had to be made when identifying tasks: for gathering useful information, a task has to be one that is sufficiently complex so that execution is not straightforward. It is also required to have a significant duration so that enough data can be collected for analysis. With this in mind, three tasks were chosen for analysis:

- Simple Navigation task
- Gradient Navigation task
- Panoramic Photo task

4.1.1 Simple Navigation

The Simple Navigation task was carried out in a cluttered environment with boxes used as obstacles. The task involved a simple autonomous navigation over a distance of approximately four metres with no map for guidance. This meant that localisation was not possible and the wheel-encoding data had to be used to determine the location of the robot using dead-reckoning. The robot was required to use its sonar sensor array to avoid the obstacles whilst traversing from the start to the finish location. The boxes were placed so that multiple routes existed between the start and finish location and the robot could follow any of these to complete the navigation. Dead-ends were also

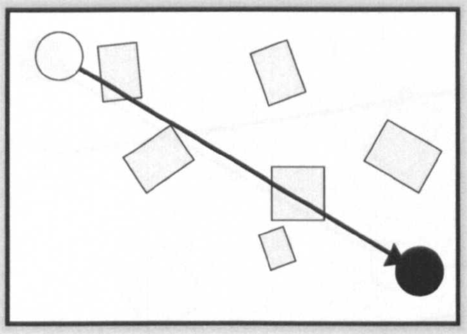


Figure 4.3: *The Simple Navigation task*

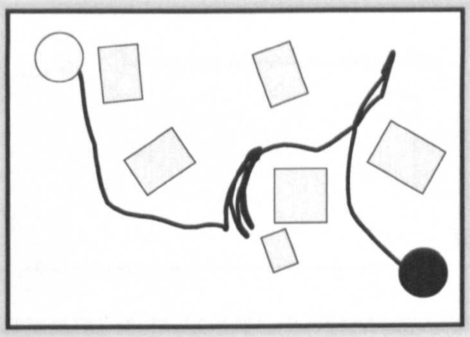


Figure 4.4: *An actual execution of the Simple Navigation task*

included so that the robot may have to backtrack to reach the goal. The locations of the boxes were varied after each execution so that the learnt model would not be dependent on a particular layout of the environment. Figure 4.3 shows a sample layout of the task, and Figure 4.4 shows a typical example of an execution of this task, using real data that was collected from the robot. The white circle is the start and the black circle is the finish. In the middle of the room, indecision can be seen where the robot tries to navigate around a pair of boxes. In this situation, the robot is seeking, trying to find the best route around the boxes.

4.1.2 Gradient Navigation

This task was similar to the above task except that a map of the walls of the room was used for position estimation and localisation. As before, boxes were used as obstacles, but the locations of these were not marked on the map. The robot could use the sonar data reflected from the walls to perform localisation to correct positional errors. Because of this, a much longer navigation task could be used without the risk of the robot losing its position and failing to complete the task successfully.

The task that was used was a simple navigation of a room approximately 10 metres long and 3 metres wide. The room was cluttered with eight large boxes positioned randomly, as in Figure 4.5. The robot had a map of the outline of the room that it could use for localisation, but the positions of the boxes were unknown to the robot. The task was to navigate between six waypoints positioned around the room. Although the robot can gain little information from the flat walls for use in localisation, they contained slightly recessed doorways (not shown) that were marked on the map and could be used for localisation purposes. The navigation between the final two waypoints involves a traversal across the middle of the room, which does not give many opportunities for accurate localisation. Because of this, the final waypoint is located in

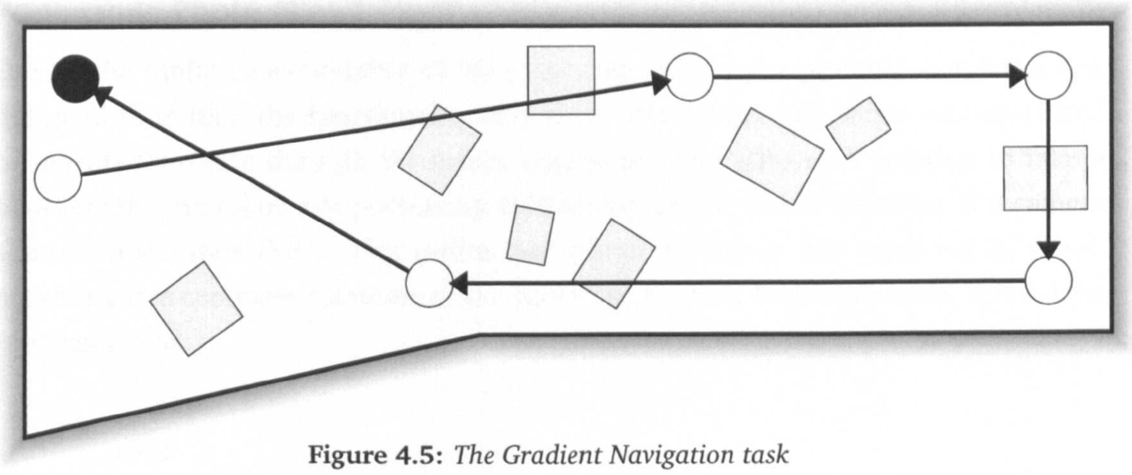


Figure 4.5: *The Gradient Navigation task*

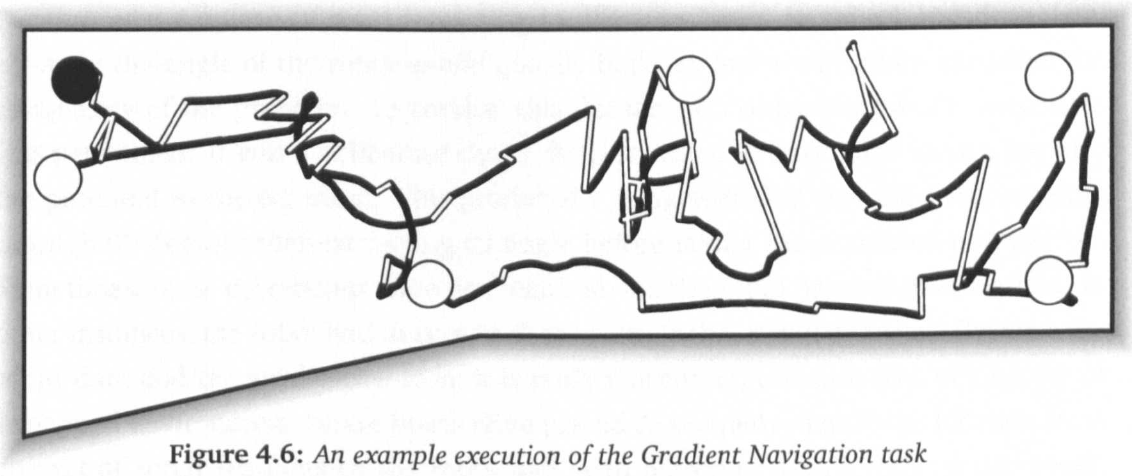


Figure 4.6: *An example execution of the Gradient Navigation task*

the top-left corner where localisation is highly effective due to the two adjacent walls. Figure 4.6 shows an actual example of an execution of this task (the locations of the boxes are not shown). In this figure, localisation steps (which are characterised by large jumps in the robot's believed location) are recorded as white lines. These are where the robot updates its location using sonar data. Note that the trajectory drawn on the map is the robot's *perceived* position, and not its actual position. The actual location of the robot is not obtainable from the sensor data available. In this particular execution, the robot does not quite achieve the waypoint in the top-right corner, but it gets close enough to register as having been reached.

4.1.3 Panoramic Photo (Simulation)

Due to the limited functionality of the robot, this task was a partially simulated task designed to imitate the functionality of a more able robot. The robot was instructed to repeatedly rotate through 90-degree angles, pausing after each rotation to take a photograph (the robot, not possessing an imaging device, made the noise of a camera shutter to simulate this). This rotate/take photo behaviour was repeated 20 times, providing five complete rotations of the robot on the spot. In pseudo-code, this can be represented as:

```
n=0;
while (n++ < 20) {
        take_photo();
        turn(90);
}
```

Because of the nature of dead-reckoning localisation based on wheel encoding data, errors in the angle of the robot would quickly build up and so therefore increased the complexity of the problem. To combat this, localisation using the robot's sonar data was performed. It was a deliberate choice for this action to have large errors, but also the potential to correct them. This produced a behaviour that saw the robot rotating through 90-degrees then correcting its angle before taking the simulated photograph. Sometimes, these corrections were not required and the robot proceeded as normal. In other instances, the robot had to turn back to an angle that it had overshot. Because the sonar data and the localisation from it is fairly inaccurate, this task was carried out in a mapped environment where boxes were placed to uniquely identify its location from any set of sonar readings (if the robot had been placed in a uniformly square room, then a set of sonar readings could represent any one of four angles).

4.1.4 Data Collection

To collect data for the automatic learning of models, the tasks were repeated between 40 and 100 times (dependant on the duration of the task and how long it took to collect data), often with slight variations in the environment to prevent over-learning. The raw data from the robot was collected for each execution of the task, and was stored for later processing.

As has already been stated, the AmigoBOT robot is quite primitive in its functionality, meaning that the data that can be collected from it is very limited. The raw data that is available from the robot is listed in Figure 4.7.

Data	Definition
x-coordinate	Measured in mm from the starting location[1]
y-coordinate	Measured in mm from the starting location
θ	Current angle of the robot
Speed	Current speed of the robot
Angular Speed	Current angular speed of the robot
Sonar Data[8]	8 distance readings indicating the distance to the nearest obstacle
Left Stall	Is the left wheel stalled? [Binary, yes or no]
Right Stall	Is the right wheel stalled? [Binary, yes or no]
Camera State	The current state of the camera [Binary, off or on] *(simulated)*

Figure 4.7: *The 16 AmigoBOT data points recorded every 100 ms*

Data is available from the robot every 100 ms, providing 160 readings per second. On their own, these raw values are not very useful as they abstract away some of the physical behaviours of the robot at any particular timepoint. Any model that was learnt from these readings would lack generality. For example, the x-coordinate of the robot is quite meaningless in terms of representation of any particular behaviour; a low or a high value of x does not signify or imply any state. On the other hand, the distance travelled over a short period of time relates to the speed of the robot, whether it is stopped, at full speed or somewhere in between. The distance travelled is a much more useful variable to learn a model from because it is closer to the actual behaviours that are exhibited by the robot. So instead of using the raw data values from the robot, they are converted into nine different readings designed to attempt to provide more useful and discriminatory values for learning a model[2]. The nine readings are described in depth below. During this conversion, the data was averaged over a sliding window to reduce noise in the data. The size of the sliding window proved to be a very difficult variable to choose — the window was required to be large enough to reduce noisy data, but small enough to capture the fast changes in velocity and angle that the robot is capable of. If the window were too large, then the subtleties of the robot's movement could be smoothed out too far and would not be encapsulated by the data. Conversely, if it were too small then greater variations would be registered that are not characteristic of the behaviour occurring at that time. With this in mind,

[1]The x and y coordinates reported are the robot's perceived location, and not the actual physical location. There may be a discrepancy between the two because of the build up of errors. If localisation is enabled then the robot tries to correct any discrepancies between the actual and perceived values using sonar data.

[2]It is worth noting that as an aside to the research, the raw data was fed into a neural-network learning tool to see if a network could be trained to report the predicted time remaining from the data. When the raw data from the robot was used, a useful model could not be learnt, and the resulting network had poor performance. When the converted readings were used, the network reported sensible values for time remaining and provided a good model for prediction. Although this is a potentially an interesting avenue for research, it also is beyond the scope of this thesis. What this does prove, though, is that the raw data alone does not directly contain useful information for predicting the time remaining in a task, whereas the converted data possesses some structure that can definitely be used to make predictions.

smoothing windows of sizes 4–24 were tried (which here represent smoothing the data over periods ranging from 400 ms–2400 ms), but the best was found to be around eight frames.

Once the data had been smoothed, data was sampled from every fourth reading, and the rest of the data was discarded. This reduced an otherwise excessive amount of data and decreased the computational requirements of the learning of models. The number of readings to discard was chosen in the same way as the size of the sliding window. It may be interesting for future work to experiment with the sizes of these values to see how they affect the predictive power of the models.

With the following:

n	Current timepoint
s	Size of smoothing window over which data is averaged
x_t	Value of x at timepoint t

The converted values may be defined:

Origin Distance

The robot's Cartesian distance from its location at the start of the task.

$$\textit{origin distance} = \frac{1}{s} \sum_{t=n-s}^{s} \sqrt{(x_t - x_0)^2 + (y_t - y_0)^2}$$

Curvilinear Distance

How far the robot has moved forwards during the smoothing window, taken as the sum of the Cartesian distances between each successive pair of (x,y) coordinate readings.

$$\textit{curvilinear distance} = \frac{1}{s} \sum_{t=n-s}^{s} \sqrt{(x_t - x_{t-1})^2 + (y_t - y_{t-1})^2}$$

The actual value for curvilinear distance omits any successive pairs that exceed the 95th percentile of the Normal distribution of the current set of pairs. Such values are most likely to be localisations where the robot's position has been significantly updated because of a localisation step. If these values were not ignored, then unwanted spikes would appear in the curvilinear distance measurement.

Angular Change

The total angle through which the robot has turned within the smoothing window, taken as the sum of all the absolute differences between each successive

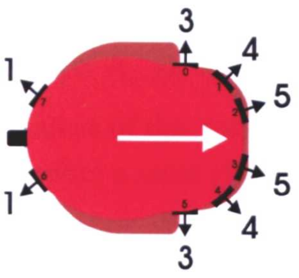


Figure 4.8: *The weighting vector applied to the sonar sensors*

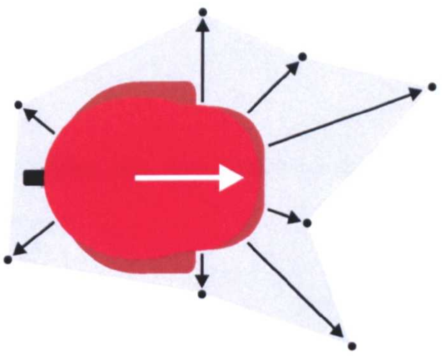


Figure 4.9: *Polygon defined by the locations of the sonar reflection points*

θ reading. A robot that has rotated 90 degrees clockwise and then 90 degrees counter-clockwise within the smoothing window therefore has an angular change of 180 degrees.

$$\mathit{angular\ change} = \frac{1}{s}\sum_{t=n-s}^{s} |\theta_t - \theta_{t-1}|$$

As with Curvilinear Distance, this is also filtered by the 95th percentile in the same manner.

Angular Difference

The absolute difference in angle between the first and last reading inside the smoothing window. A robot that has rotated 90 degrees clockwise and then 90 degrees counter-clockwise within the smoothing window has an angular difference of 0 degrees.

$$\mathit{angular\ difference} = \theta_n - \theta_{n-s}$$

Cluttering

How cluttered the environment is, calculated from the area inside the polygon formed by the sonar reflection points. This is weighted towards the front sensors as these are more important.

The robot reports eight sonar readings, of angle $\sigma_1 \ldots \sigma_8$ and distance $d_1 \ldots d_8$

Sonar values are multiplied by a weighting vector $w = [3,4,5,5,4,3,1,1]$, as shown in Figure 4.8. Therefore weighted distance is defined as $e_i = w_i d_i$. The Cartesian coordinates of the weighted distances are calculated as:

$$sx_i = e_i \sin^{-1}\sigma_i \; sy_i = e_i \cos^{-1}\sigma_i$$

Then the area inside the polygon defined by these points (Figure 4.9) is calculated:

$$\mathit{cluttering} = \tfrac{1}{2}\sum_{i=1}^{8} sx_i sy_{i+1} - sx_{i+1} sy_i$$

(where $sx_9 \cong sx_1, sy_9 \cong sy_1$)

Lower values of this value represent that the robot has less room to manœuvre, and therefore a more cluttered environment.

Speed

The average speed of the robot over the smoothing window

$$\mathit{speed} = \tfrac{1}{s} \sum_{t=n-s}^{s} \mathit{speed}_t$$

Distance Spread

The difference between the maximum and minimum curvilinear distances inside the smoothing window:

$$\mathit{distance\ spread} = \max\left(\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\right) - \min\left(\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\right)$$

where $i = (n-s)\ldots s$

High values of Distance Spread indicate that the robot has changed its speed during the course of the smoothing window, whilst low values show that the speed has remained roughly the same. Extremely high values are indicative of a position localisation step, where the speed has been low and then has increased dramatically. An example of this is shown in Figure 4.10. In Case (a), a typical set of eight coordinate pairs is shown, representing the robot's movement across the smoothing window with low distance spread. Case (b) is an identical execution, except for a position localisation step that corrects the location of the robot in one large jump. It can clearly be seen by visual examination that the robot is unlikely to have physically navigated between these two points within 100 ms.

Angular Spread

The difference between the maximum and minimum angular changes inside the smoothing window:

$$\mathit{angular\ spread} = \max(|\theta_i - \theta_{i-1}|) - \min(|\theta_i - \theta_{i-1}|)$$

where $i = (n-s)\ldots s$

In the same way that Distance Spread can be used to detect position localisation, Angular Spread can be used to detect angle localisation where the robot's perceived angle changes abruptly.

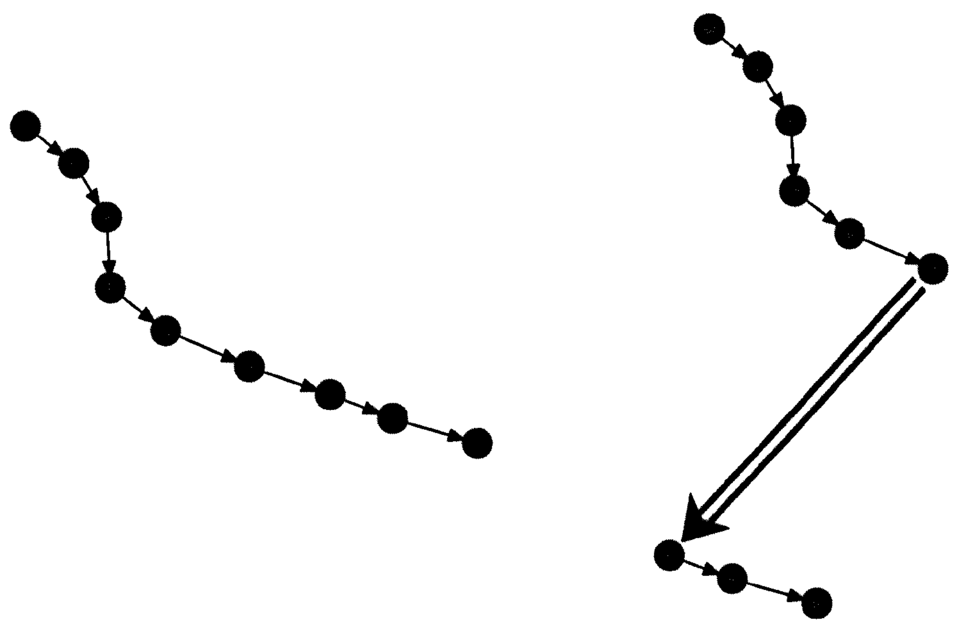

(a) Low Distance Spread (b) Very high Distance Spread

Figure 4.10: *Distance Spread and its relation to localisation steps*

Camera State

Current state of the camera (0 or 1). Because of the binary nature of the camera state, averaging was not used but instead the modally occurring value within the smoothing window was used. In the event of a tie, the last occurring value was used.

Once the raw data had been collected, converted and smoothed as above, the converted data was used to learn an HMM for the task. There are two fundamental steps to this process, clustering and expectation maximisation.

4.1.5 Clustering and Expectation Maximisation

The processed data was clustered to automatically identify a number of *observations*. An observation is a physical state of the system that can be represented by certain values of the processed data. For example, one observation could be identified as a situation where the distance spread was high and the angular spread low. Clustering was done using a Kohonen map [Kohonen, 1990].

Kohonen maps work by mapping a multidimensional input space onto a space of lower dimensions. In this case, the 9-dimensional data collected from the robot is projected onto a 2-dimensional grid, or network. The size of the network affects the complexity of the learnt model, which was varied here between size 10×10 and 30×30. The Kohonen map is initialised with each location in the network being associated with a random vector in the 9-dimensional input space. To avoid problems with initial bias, the vectors were initialised so that no two vectors were within *n* degrees of each other (with *n* being chosen depending on the size of the network).

Training of the Kohonen map is done by presenting each of the input vectors in sequence, and identifying the cell in the grid that has the closest vector to the input vector. This vector, plus the vectors in the surrounding area in the grid (the neighbourhood cells), are aligned towards the input vector by scalar multiplication. The size of the neighbourhood is set initially large, but decreases exponentially over time to facilitate training and avoid local maxima. Decreasing the size allows the network settle into a stable state. This training process is carried out fifty times across the data set, and then the order of the input data is randomised before repeating training a further fifty times. The reordering helps to eliminate any overlearning issues that may occur due to the ordering of the data.

After the training is complete, the result is a series of vectors, each associated with a cell in the network. These are grouped according to the number of input vectors that are attracted to each of the output vectors. The groupings define the clusters that are used in the final model, each internally represented as a normalised vector.

Kohonen maps were chosen because they require no prior knowledge of the number of clusters during the learning process. Some clustering techniques, such as K-means clustering [MacQueen, 1967], require this number as an input to the learning process. Not specifying the number of clusters allows the network to best determine this value, outside of the influence of manual estimates. Since the training data possessed an unknown number of clusters this was the preferred method. An outline of the clustering process carried out by the Kohonen map, as described above, can be seen in Figure 4.11.

The size of the Kohonen map had to be chosen manually to ensure that a reasonable, but not excessive number of observations (clusters) were identified. A larger Kohonen map will produce a greater number of observations, and a smaller map will produce fewer observations. With too few observations, the discriminatory power is lost as all data gets classified into a limited number of observations; with too many observations, the noise in the learning process becomes too significant relative to the signal and training becomes ineffective. Extensive testing showed the optimal size of the map seemed to be one that, after expectation maximisation (see below), produced HMMs possessing in the range of 15–30 states. When learning the models, one of the things that had to be considered was the complexity of the task. For a simple task, a

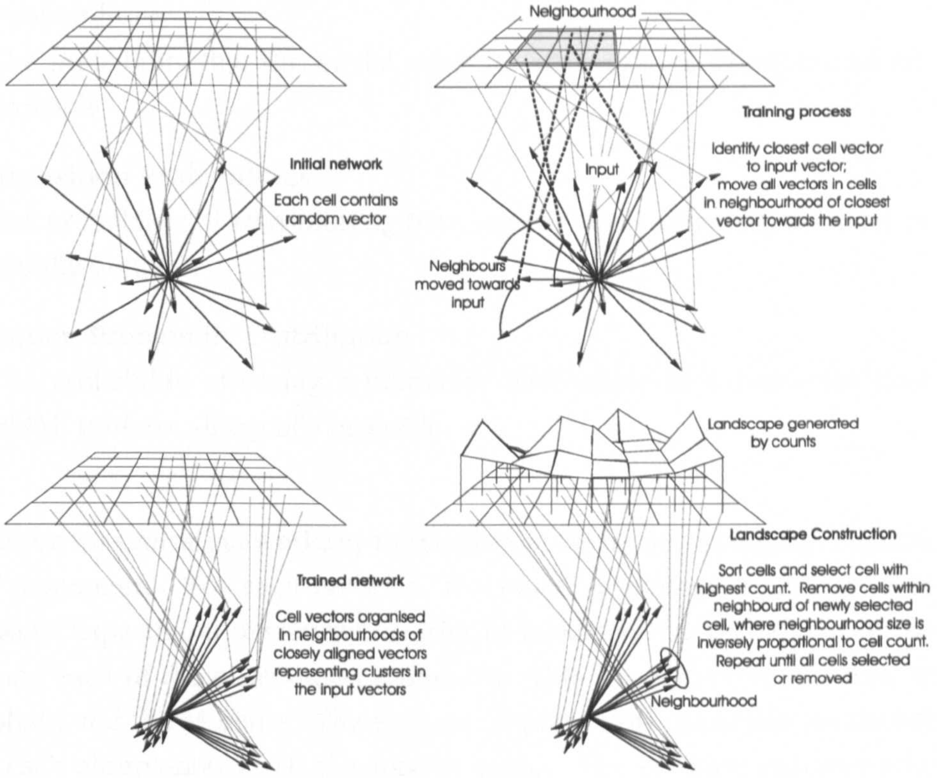

Initial network
Each cell contains random vector
Neighbourhood
Input
Training process
Identify closest cell vector to input vector; move all vectors in cells in neighbourhood of closest vector towards the input
Neighbours moved towards input
Landscape generated by counts
Landscape Construction
Sort cells and select cell with highest count. Remove cells within neighbourd of newly selected cell, where neighbourhood size is inversely proportional to cell count. Repeat until all cells selected or removed
Neighbourhood
Trained network
Cell vectors organised in neighbourhoods of closely aligned vectors representing clusters in the input vectors

Figure 4.11: *Clustering using a Kohonen map (Figure reprinted from Fox et al. AIJ vol 170, no 2)*

model with few states is more preferable, but more complex tasks require more complex models to represent them. In [Fox et al., 2006], dimensions for the Kohonen map ranging from 15–45 were used for a data set of 15,000 data-points. It was found that these values also worked best for the data sets collected within this thesis, which range in size from 15,000 up to 55,000 data-points (depending on the task being learnt).

Once the data had been clustered using the Kohonen map, each recorded piece of data could be classified by the map into a numbered observation by using the normalised vectors. Entire sequences of data from each execution can then be converted into numbered lists of observations. From these, expectation maximisation can be used to learn an HMM that could have produced this data.

The observations as learnt above are decomposed into a series of states using a state-splitting strategy. This works by examining the angles between the vectors and splitting them into subsets. The result of this process is a set of observations and a set of states. Expectation maximisation calculates a Hidden Markov Model that represents the relationship between these two. The HMM is represented by three sets of parameters:

Prior Probabilities
: The probability that the model starts in each state, in initial model all equally probable

State Transition Probabilities
: The probability of transitioning between each pair of states, in initial model all equally probable

Observation Probability Distribution
: The probability of seeing a particular observation in a particular state of the HMM. Initially, all equally probable.

Expectation maximisation works by iterating over the model represented by these three sets of parameters. For each iteration, the model is presented with all observation sequences. Expectation maximisation tries to maximise the probability of the model producing each of the observation sequences. This is a two-phase process, split into the E phase and the M phase. The E phase (expectation) calculates the probability of seeing each observation given the current model. The M phase (maximisation) then updates the model to maximise the probability seen in the E phase. This is a repeated hill-climbing strategy that improves the HMM with respect to the evidence.

After expectation maximisation, the HMM is a model that can predict the internal state of the executive and how it transitions between certain hidden states of execution. In Chapter 6, learnt HMMs will be used to monitor these states during execution and attempt to use this information for controlling the executive.

Figure 4.12 shows an example HMM learnt from the Panoramic Photo task by using the above process. All of the states and the transitions between these are automatically generated. This HMM is included here to give the reader an insight into what a learnt model looks like. The start state is marked in green and the finish is coloured red. Note some of the structure present such as the loops and cycles between states, and the "starting" and "finishing" behaviours. Also note that there are shortcuts in the cycles that allow certain states in the loop to be bypassed. With an automatically learnt model such as this, it is important to note that there is not a simple interpretation of these states — any analysis of automatically learnt models has to be done manually. It is not necessary to understand or interpret this graph at the moment, but this HMM (along with others) will be examined and discussed in depth in Chapter 5.

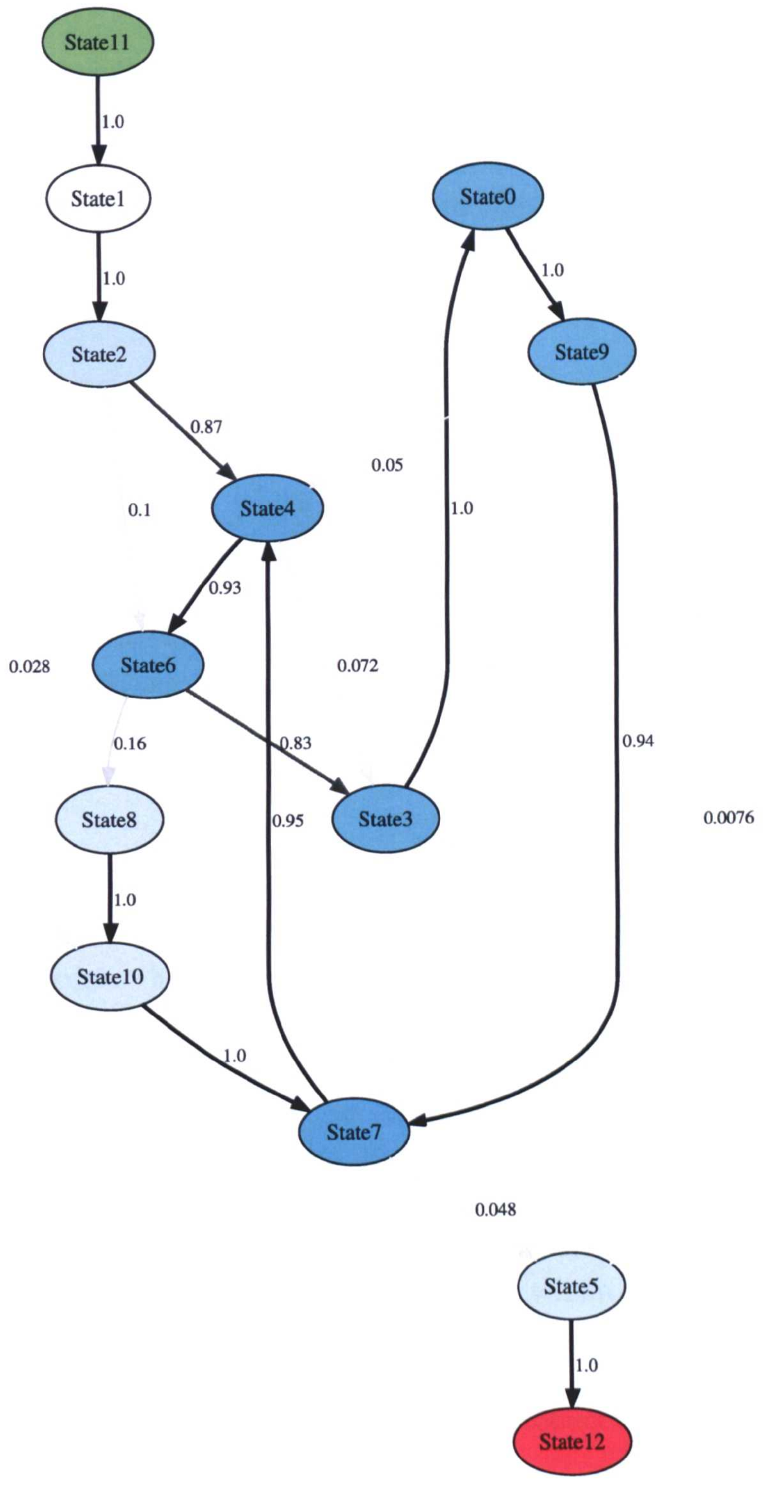


Figure 4.12: *An HMM for Gradient Navigation*

4.2 Error Introduction

To test the ability of the learning process to identify error states, an error-prone version of the Panoramic Photo task was developed. This was a partially simulated action that had a small failure probability. This could be seen as contrived, but it was important that the process was tested on an action that had recognisable failure states and was repeatable under test conditions. Developing a real action that could fail repeatedly and on command was deemed too difficult. The Panoramic Photo with Errors task was implemented roughly as follows:

```
n=0;
while (n++ < 20) {
    turn(90);
    if (rand() < 0.97) { // 3% probability of failure
        take_photo();
    } else {
        shutter_stuck();
        wait(rand()); // random duration to unstick
    }
}
```

The action simulated the camera shutter malfunctioning and becoming stuck. During this period, the robot simulated attempts to free the camera shutter, which took a random duration to unstick. The simulation meant that the execution of the action never failed entirely and could always recover from the failure, but the duration for recovery was uncertain. The duration was in the region of 2–30 seconds. The 20 repetitions of the camera shutter provided an action that would experience a failure in 46% of the cases $(1 - 0.97^{20})$. If the error rate were too low, then a significant number of data points would have to be collected to show meaningful data. In a real-world situation, a sufficient number of data points would have to be sampled to ensure that the action covered all possible executions and the errors contained within these.

Now that suitable tasks have been identified, the following chapter will examine data collected from these as well as the resultant HMMs produced through the learning process that has been outlined above.

CHAPTER 5

Evaluation 1

The cause is hidden; the effect is visible to all

— *Ovid, Roman Poet*

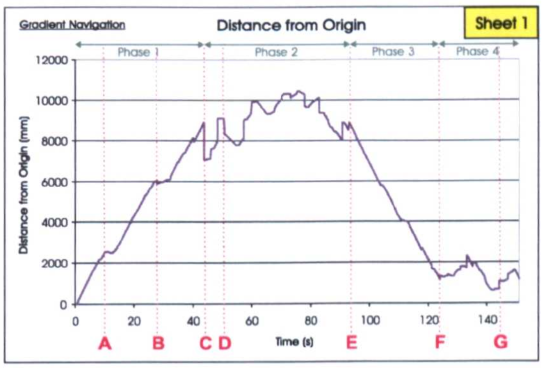
IT IS USEFUL to examine the raw data collected from the robot to understand patterns, trends and features of the data. If there are no such patterns, then attempts to learn models will not produce meaningful results. The first part of this chapter examines data from individual runs to try to identify what is happening with the robot and the states that it may be in. The second part of this chapter investigates the HMMs learnt from the processed data.

5.1 Raw Data Analysis

This section will examine the raw data collected from a single execution for both the Gradient Navigation task and the Panoramic Photo with Errors task. These were chosen for analysis because they possess much more interesting structures than the Simple Navigation or standard Panoramic Photo task. The full-size graphs for this section can be found in Appendix A, but small inline versions are also included in for easy reference. Whilst reading this section, it is advisable to refer to the graphs to examine the correlations described below. A series of points of interest have been identified, and these points are marked on each of the graphs. For both of the tasks, analyses of only the most interesting and relevant variables are shown below — an omission indicates that the variable did not contain any interesting structures deemed necessary for analysis.

5.1.1 Gradient Navigation task

Distance from Origin (Sheet 1)



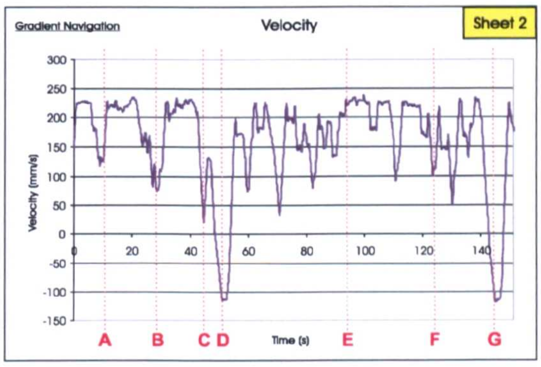
There are four distinct phases that can be seen in this graph, which are marked at the top of the graph.

Phase 1 is characterised by the distance from the origin increasing at a roughly constant rate. This is the robot travelling at maximum velocity, unimpeded by obstacles. There are two small plateaus at A and B where the robot has slowed, perhaps navigating to avoid a box that was in the way, but these were not major deviations from the straight-line path and had little effect on the distance from origin. The third phase is similar to the first, except that the robot is returning to the start, thus the distance from the origin is decreasing.

Phase 2 lies around a distance of 9 m from the origin, where Waypoints 2 and 3 for the task lie. During this phase, the distance fluctuates as the robot navigates around a series of obstacles that are in its way. The robot cannot easily see the route through the obstacles so it takes a short while to complete, trying various options until success.

In the fourth phase, a similar behaviour to the second phase is seen, however the robot's overall behaviour during this phase is quite different. Here, the robot is trying to return to the starting location after a navigation across the middle of the room. Whilst crossing the room, the robot's sensors do not have the range to pick up the locations of the walls and therefore localisation cannot be performed. This causes any error in its location to accumulate. When the robot finally re-detects the wall, it starts to localise itself once more whilst travelling to the final waypoint. The robot regularly updates its position based on the new data, and this may fluctuate for a short while with large errors in its model of up to 2 m or more. The robot continuously aims to reach the waypoint, but its estimated location may alter frequently due to localisation. This may cause the robot to stop, start and turn to where it believes is the goal. Eventually, the error is sufficiently reduced to allow the robot to reach its final waypoint, and the action successfully terminates.

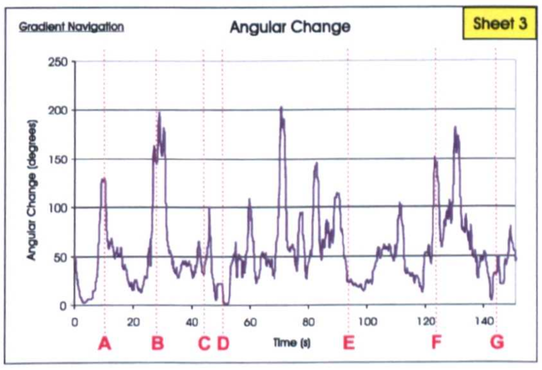
Velocity (Sheet 2)



The main features in this graph are the two points at which the velocity goes negative i.e. the robot is reversing (Points D and G). This is due to the robot having stalled due to a collision with an obstacle, and then trying to recover from this by reversing out of the situation before continuing.

The periods during which the robot is travelling unimpeded at maximum velocity can also clearly be identified, with a maximum speed of around 230 mm/s. If this graph is compared with Sheet 1, the distance from origin, then the plateaus that were identified at A and B can be correlated directly with drops in velocity of the robot.

Angular Change (Sheet 3)

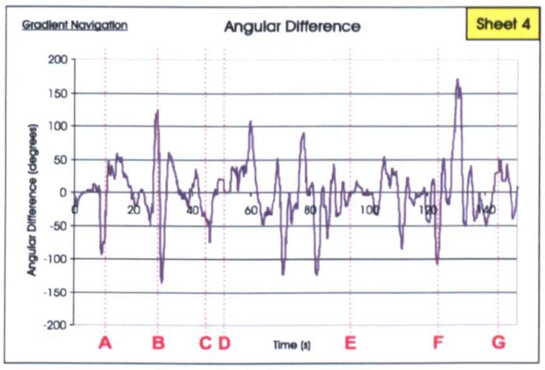


From this graph it can be seen that the robot rarely travels in a relatively straight line for prolonged periods of time. There are three distinct points on this graph where the robot has a very low angular change, and therefore travelling forwards without turning significantly. Very shortly after starting the angular change becomes low. This is because at this point the robot had turned towards the first waypoint and had no obstacles directly in its path, allowing it to proceed in a straight line. Points D and G can be correlated with Sheet 2 to match up with the times when the robot had negative velocity. This means that the robot was not rotating whilst reversing during stall recovery.[1] For a short while after Point E, the angular change is low. This is just after the 3rd waypoint when the robot was travelling towards the finish, and had no obstacles to avoid. Angular change is then significantly higher after Point F when the robot is trying to locate the finishing waypoint.

[1]If the stall recovery attempt had not been successful, the robot would have tried other methods to free itself, including turning back and forth.

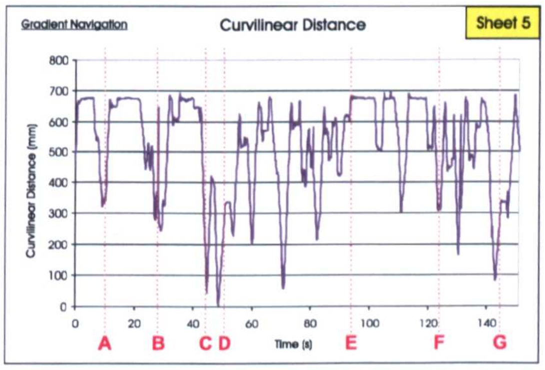
Angular Difference (Sheet 4)

This graph, although hard to interpret at first, can be used to interpret periods of indecision. If this graph is compared to Sheet 3, Points A and C have a high angular change alongside an angular difference that quickly fluctuates between positive and negative values. This means that the robot is performing a lot of rotational movements, both clockwise and anticlockwise, and in this case the robot is searching for a route around an obstacle and cannot decide whether to pass the obstacle to the left or right.



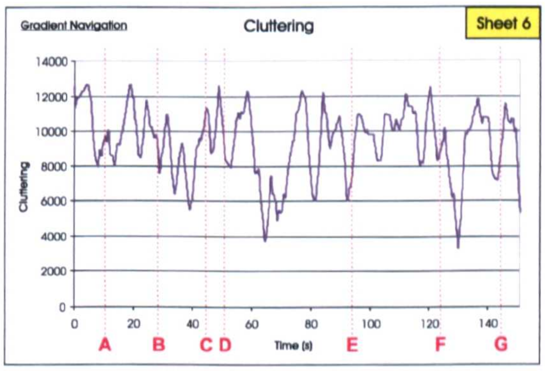
Curvilinear Distance (Sheet 5)

This graph is very similar to Sheet 2, the graph of the robot's velocity. This is as would be expected given the close relationship between velocity and distance travelled. The main deviations between these graphs occur when the robot's velocity went negative, at Points D and G. No additional pertinent information can be derived from this graph that has not been already derived from Sheet 2.



Cluttering (Sheet 6)

The cluttering graph is difficult to interpret and is probably more useful when viewed alongside other graphs. Note again that lower values mean a more cluttered environment. If this graph is compared with Sheet 1 then it can be seen that Phases 2 and 4 contain several times when the environment was very cluttered. This fits in with what has been deduced so far.

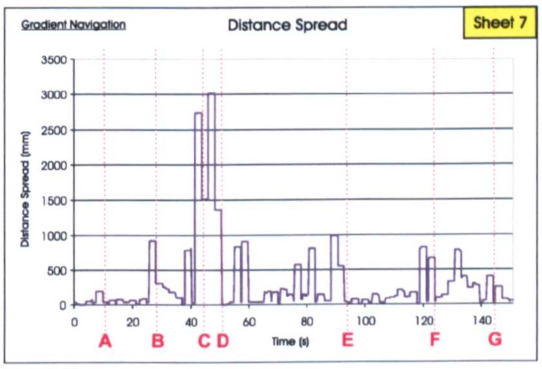


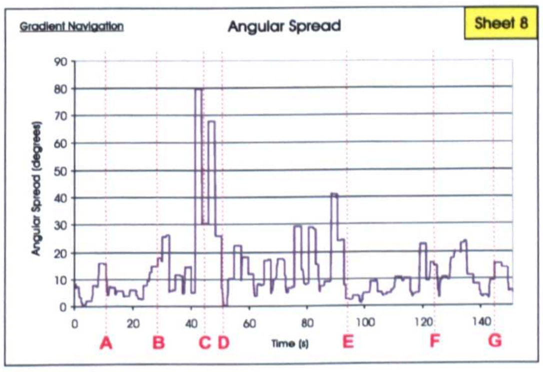
Point	Identified Situation
A	Obstacle avoidance
B	Obstacle avoidance with indecision
C	Localisation step
D	Collision recovery
E	Nominal straight-line navigation
F	Homing-in on Goal
G	Collision recovery

Figure 5.1: *Points identified in the Gradient Navigation task*

Distance and Angular Spread (Sheets 7 and 8)

Peaks in these graphs are indicative of localisation steps in which the robot's position or angle have been significantly updated. There are a couple of well-defined peaks around Point C, and several smaller peaks across the graphs. In the case of Point C, Sheet 1 shows that the robot's position changed by approximately 2 m in a very short space of time, supporting the evidence that this is a localisation step. It is interesting to note that on Sheet 2 the velocity drops down to 20 mm/s after this point. This suggests that the robot may have collided with an obstacle at this point due to the localisation not being accurate. Indeed shortly after this, Point D is reached where the robot has stalled by colliding with an obstacle and can only recover by reversing.



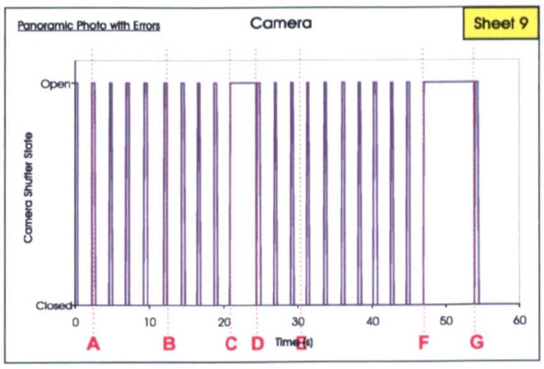


Gradient Navigation Summary

From the graphs, the major events in this particular execution are summarised in Figure 5.1. Assigning names to all these different states is of little importance: the significance is that it is possible to distinctly differentiate behaviours of the robot based on the raw data. If it were not possible to do this manually then it would probably be a futile task to try to learn these states automatically. Ultimately, what is required is a number of distinct states into which can be used to classify the robot's behaviour. A classification tool should be able to take a set of the nine variables and deduce which behaviour the robot is exhibiting at any particular timepoint.

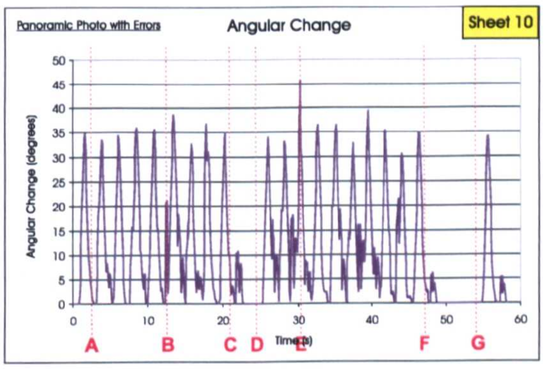
5.1.2 Panoramic Photo with Errors task

Sheets 9–14 show some of the raw data graphs for an execution of a Panoramic Photo with Errors task. This particular execution was chosen because of the fact that two simulated camera failures occurred during execution.



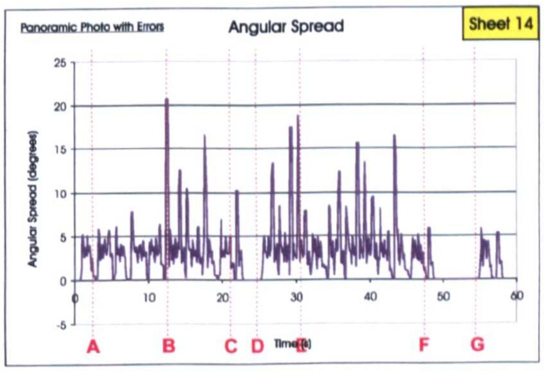
Camera State (Sheet 9)

The periodicity of the task can clearly be seen with the camera shutter alternating between open and closed, with around two seconds between photos. Also, the two failures can also be easily identified when the simulated camera shutter was stuck open, from C to D, and from F to G. Note that once the shutter has been successfully 'unstuck', it still needs to take the photograph at that angle. This means that failures are always followed immediately by another photo.



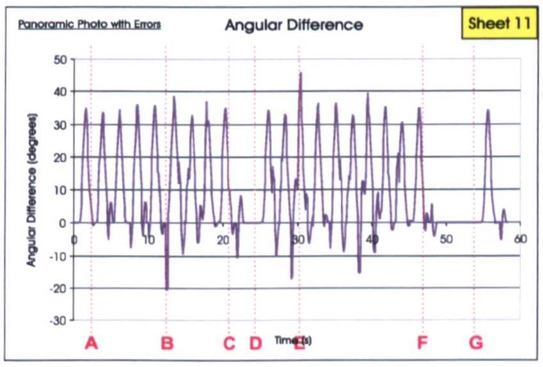
Angular Change (Sheet 10)

Again, the periodicity is easily recognisable in this graph with periods of turning to the required angle, and then remaining still to take the photo. The periods of inactivity whilst trying to fix the camera shutter are also evident from C to D and F to G. These periods do not seem to match up exactly with the periods identified from the camera shutter data, but this is due to the navigation techniques used by the robot. The robot may report



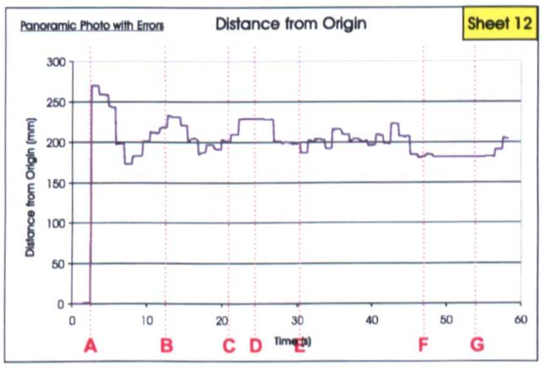
that it has finished the movement, but then make minor corrections to get closer to the desired angle. This may happen if localisation detects small errors in the position and the robot reacts by correcting these. There are also several points on Sheet 10 where the angular change goes significantly higher than usual, for example at Point E. This indicates that that the robot is turning more than usual to take the photo, which initially seems bizarre as the angle through which the robot turns should be constant. When this graph is compared to the angle spread on Sheet 14 a large spread of values is seen at Point E, which is indicative of an angle localisation. What happened here was that the robot turned the 90-degree angle, but part way through the turn it corrected an error in this angle. This meant that it had to turn more to reach the correct heading.

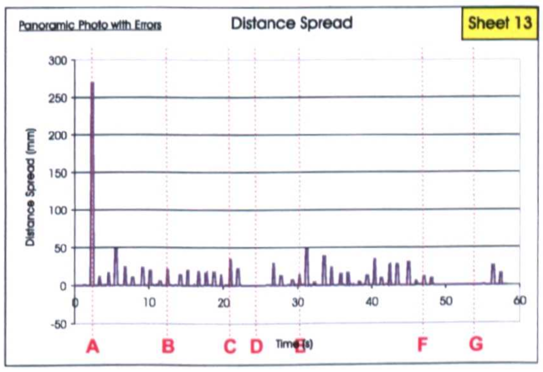
Angular Difference (Sheet 11)



This graph shows several periods during which the angular difference goes negative, for example at Point B and just before Point E. A negative value indicates that the robot is actually turning in the opposite direction to that which it has been instructed to do so. By comparing to Sheet 14 again, these two points also correspond to angle localisation steps. In these cases however, the robot rotated too far and had to turn back to reach the desired heading.

Distance from Origin (Sheet 12)





The main feature in this graph is the huge increase in distance at Point A. Sheet 13 shows that this point corresponds to a large localisation step approximately three seconds after starting the task. This is because when the robot starts the task it has very little data with which to perform localisation. One the robot starts rotating, it receives many more sonar reflections that provide quite detailed information about the location of the robot. With all of this sonar data,

Point	Identified Situation
A	Initial position localisation
B	Over-rotation localisation and correction
C–D	Camera shutter stuck
E	Under-rotation localisation and correction
F–G	Camera shutter stuck

Figure 5.2: *Points identified in the Panoramic Photo with Errors task*

it can accurately correct any discrepancy between the believed location (at the origin) and the actual location. The actual location in this case is around 20 cm from the origin and is continuously updated throughout the task. The robot does not change its position throughout the task, only its idea of its location. The fluctuations over the course of the task are mostly due to the imprecise nature of the sonar data.

Panoramic Photo with Errors Summary

From the graphs, the major events that have been determined in this particular execution are summarised in Figure 5.2.

5.2 HMM Analysis

To analyse the learnt HMMs, a tool called ModelViz was produced. This takes an array of transition probabilities between states in the HMM and displays them graphically as a connected network.

The ModelViz tool uses the Graphviz software package [Ellson et al., 2004] to produce the state transition diagrams found later in this Chapter. In simple operation, ModelViz generates a basic monochrome visualisation of the directed graph that constitutes the HMM. If the start and finish states are supplied to ModelViz, then the tool performs a number of simulations of the HMM, tracing through executions of the model. In this case, the output diagram is coloured to indicate more clearly the structure of the HMM. The states that get visited most (and therefore are most important to the HMM) are coloured with darker blue colours. Those that are visited infrequently are coloured white or pale blue. Additionally, the start and finish states are coloured green and red respectively. The thickness and darkness of the transition lines is determined by the probability that the transition will occur.

The ModelViz tool has a further option with which transitions to the same node are ignored. This can be useful for HMMs that produce long repeating sequences of the same state. When this option is enabled, the self-transition probability is removed, and the rest are normalised to equal 1.0. With this, the probabilities adjacent to the transitions represent the probabilities of the next *different* state occurring. Normalisation in this manner is useful in some situations to simplify an HMM, but in others it can mask structure by discarding important self-transitions. In the analyses below, self-transitions have been ignored as the extra clarity of the models produced outweighs the structure lost in the process.

In most HMMs produced by the learning process, there are a large number of transitions, which would make a graph overly complex. To reduce the complexity, transitions below a certain probability (0.05) are omitted by default. One of the drawbacks of this simplification technique is that it can leave the graph unconnected and result in dead-ends and never-ending loops. The ModelViz tool was therefore altered to recognise unconnected parts of the graph and search these for the highest probability transition that links it to a connected part of the graph. The result is a visualisation of the HMM that is fully connected and will always eventually terminate.

In the next section, there will be a visual examination of some of the HMMs learnt for each of the tasks:

- Simple Navigation
- Gradient Navigation
- Panoramic Photo
- Panoramic Photo with Errors

An attempt to identify the characteristics of particular states has been made, but it must be noted that there is some guesswork involved in this process. This is because a state in the HMM may represent more than one aspect of behaviour, and interpretation of the values is required. A state can be identified by finding the observations that are most characteristic of that particular state, and then examining the normalised vectors associated with the observation to see the relative values of the variables. Figure 5.3 shows an example normalised vector. The values represent the number of standard deviations from the mean that is associated with the observation. In the example, note there is a large positive angular change alongside a large negative angular difference, indicating that this vector is associated with the robot turning anticlockwise. Also worth noting is that the angular spread is reasonably large, which may either indicate a localisation step (characterised by a big jump in angular change) or it could be that the robot has

Data Type	Normalised Value
Origin Distance	0.19
Curvilinear Distance	-0.16
Angular Change	0.64
Angular Difference	-0.63
Cluttering	-0.08
Speed	0.21
Distance Spread	-0.08
Angular Spread	0.41
Camera State	-0.22

Figure 5.3: *Identifying characteristics of an observation*

just started turning very suddenly. To establish which of these is the case, it is necessary to examine the other normalised vectors and attempt to infer which is the case.

For each of the tasks, several different sized Kohonen networks were used for the learning process. These varied from a small 10×10 network up to a large 30×30 network — inside the range identified in [Fox et al., 2006]. A smaller network will produce an HMM with fewer states. If there are too few states to accurately describe the task then there is the possibly of dissimilar states being grouped together into one state. With a larger network, structure may be lost due to the presence of excess noise. The different sizes of Kohonen networks were learnt to find what made an acceptable balance in HMM size.

Because of the huge amount of time taken to learn these HMMs it was not possible to do a comprehensive study of all of the variables that affected the output of the learnt models. These were set through much trial and error based on values used in [Fox et al., 2006].

5.2.1 Panoramic Photo

Size 10 (Figure 5.4)

Because of the limited size of the Kohonen network used to learn this HMM, there are very few states and therefore not much discrimination between these. The main visible feature in this HMM is the prominent looping behaviour of $\overline{0} \rightarrow \overline{5} \rightarrow [\overline{3}|\overline{6}] \rightarrow \overline{2} \rightarrow \overline{1} \rightarrow \overline{0}$. The robot's behaviour within this loop alternates between states of turning and the camera shutter being active at $\overline{5}$, reflecting the TAKE PHOTO, TURN behaviour that comprises the Panoramic Photo task. $\overline{2}$ can be identified as an angle localisation step, and this is often preceded by $\overline{3}$ in which the robot appeared to be adjusting its position. There are no states that appear to have even a weak association with position localisation.

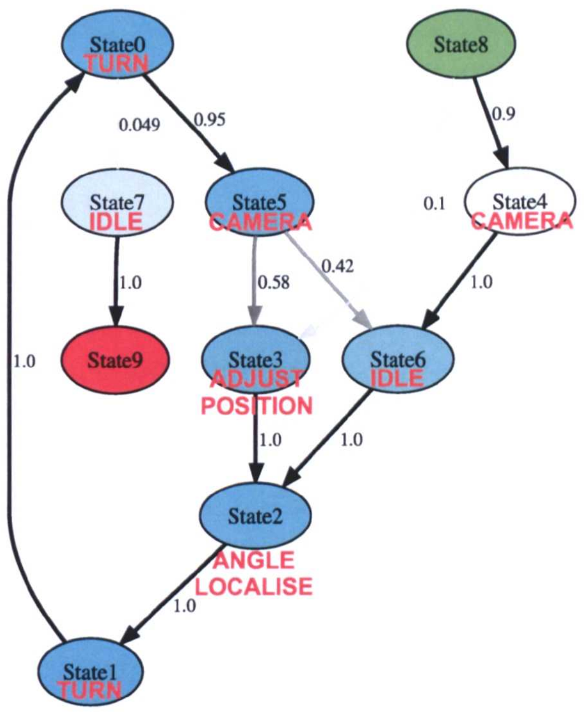


Figure 5.4: *HMM for Panoramic Photo with size 10 network*

At the start of the model, insertion into the loop is usually through $\overline{4}$, associated with the camera shutter being open. This is the first thing that is expected to happen in the Panoramic Photo task. Interestingly, there is a small probability that the loop will be entered at $\overline{3}$, which permits the robot to adjust its position, localise itself and turn before taking the first photo.

At the end of the model, $\overline{7}$ corresponds to the robot being idle, which was a programmed behaviour of the Panoramic Photo task to allow the task to complete and the sensor values to settle because of the smoothing window. The probability of entering this idle state (and therefore terminating) is 0.049. An ideal model should have a probability that results in an average of 20 cycles around the loop, the number of turns that were programmed into the task. This ideal probability can be calculated:

$$\sum_{i=0}^{\infty} i.p(1-p)^i = 20$$

and then solving this for p gives:

$$p = 0.05$$

This value is approximately the same as the probability that has been learnt in this model, indicating that this aspect of the task is represented well.

A network of this size seems to produce an HMM that has too few states to capture the intricacies of the task, as can be seen by the lack of states corresponding to position localisation. It is interesting however to see the model as a reference point for more complex models.

Size 20 (Figure 5.5)

As with the size 10 network, the main feature in this HMM is the looping behaviour in the centre of the diagram, except that here there are several loops visible with multiple paths. The most prominent loop is $\overline{4} \rightarrow \overline{6} \rightarrow \overline{3} \rightarrow \overline{0} \rightarrow \overline{9} \rightarrow \overline{7} \rightarrow \overline{4}$. This corresponds to a camera shutter event, followed by a turning behaviour with localisation steps in between. There are also shortcuts and side branches to this loop, for example the transition from $\overline{9} \rightarrow \overline{4}$ that bypasses the need for a localisation step within the loop. This could correspond to the robot having turned accurately, removing the need to localise at this point.

Also within this HMM is the possibility of taking the transition $\overline{6} \rightarrow \overline{8}$, which corresponds strongly to an angle localisation step followed by fast turning, before entering into the main loop again. This seems to be a learnt option that could allow the robot to correct its angle if it deviated significantly from the expected values.

At the start and end of the model there are states that correspond to the robot being idle. $\overline{5}$ is an idle behaviour much like the one seen at the end of the size 10 network above. The combined probability of entering this state is once again very close to the ideal probability of 0.05 (as calculated above).

Size 25 (Figure 5.6)

More complex still than the previous models, this HMM again has a prominent main loop, but this time much more complex. There are many paths of traversal through the HMM, as well as shortcuts bypassing localisation steps. For example, there is an interesting feature at $\overline{10}$ where there are transitions to $\overline{0}$ or $\overline{3}$ with almost equal probability. If the transition to $\overline{3}$ is taken, then there is a sequence of localisation steps, but if the transition to $\overline{0}$ is taken then no localisation occurs. Examples such as this can show how the learnt HMM has adapted to take into account physical uncertainties as aspects of the task. This is in contrast to the size 10 HMM, which does not have states to represent these different uncertainties. The size 20 HMM possessed some of these characteristics.

Although similar in many ways to the size 10 and size 20 models, this model does not have the expected probability close to the ideal for exiting the loop and transitioning

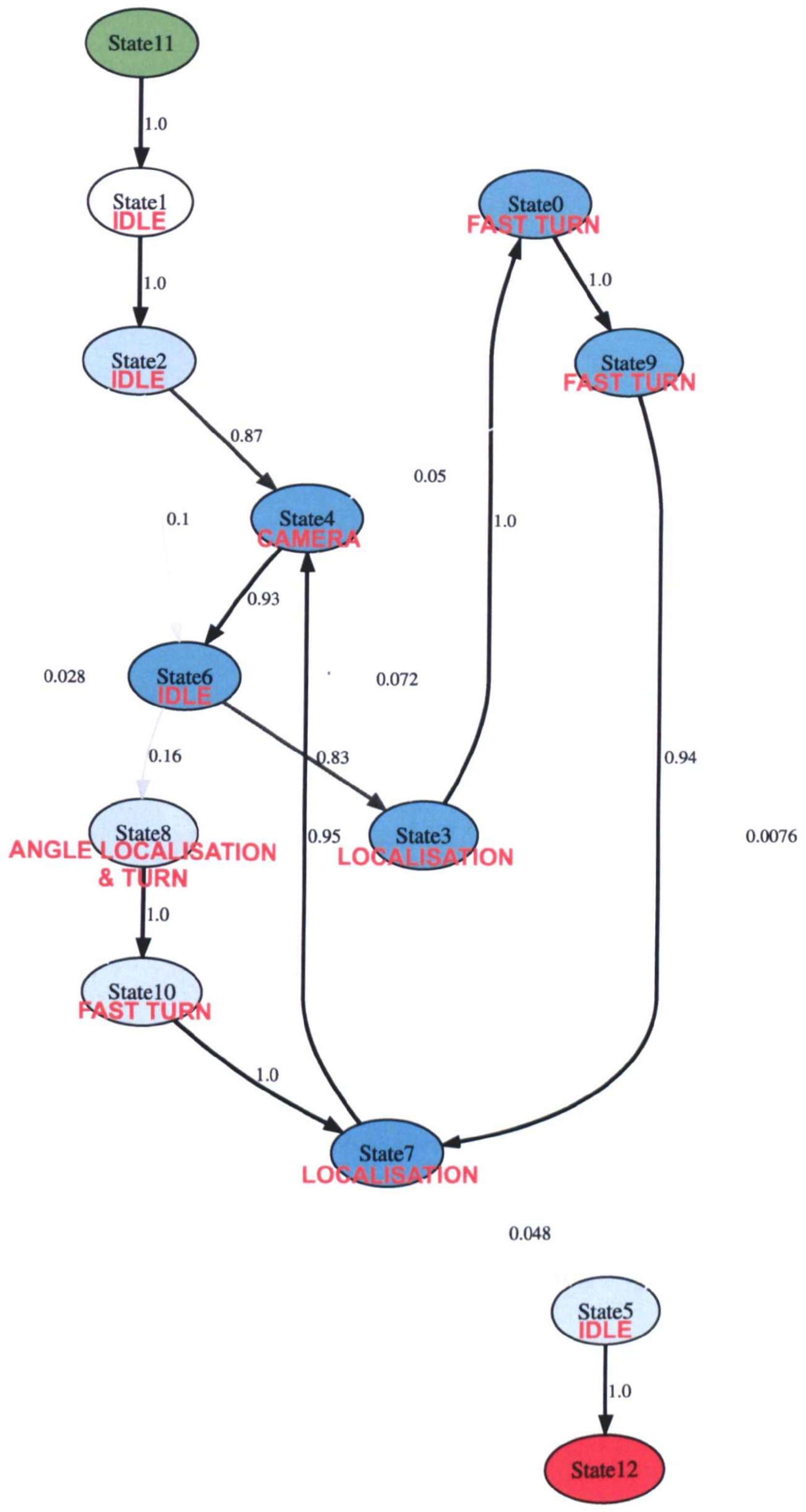


Figure 5.5: *HMM for Panoramic Photo with size 20 network*

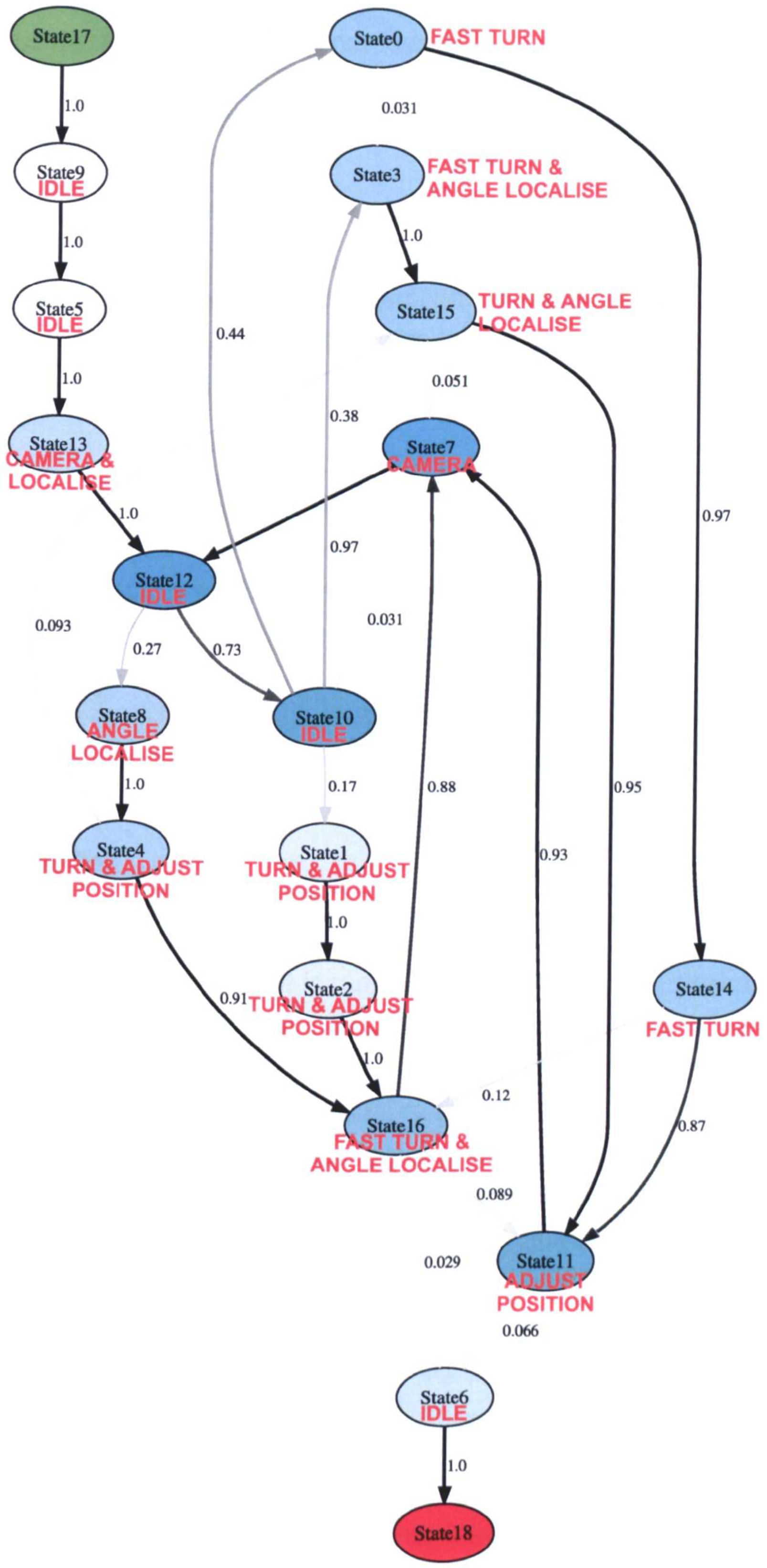


Figure 5.6: *HMM for Panoramic Photo with size 25 network*

through $\overline{6}$ to the finish state. This is not a major issue as it is of the correct magnitude, but could indicate that this model has not learnt this feature correctly.

Size 30 (Figure 5.7)

At this size, the network becomes very complex and difficult to analyse in depth. There are micro-structures that can be identified within the model, but comprehending the structure of the HMM as a whole is very difficult. It can be seen that the structure is very similar to that of the size 25 model with loops and transitions that bypass parts of these. Because of the complexity of this model, not all of the states in this diagram have been labelled.

In this model there are not one, but two states that correspond to the camera shutter being open. This seems to suggest that either the model has learnt some distinction between the camera states to classify them differently, or a size 30 model is too large and has split this state into two unnecessarily. It is not possible to tell which is the case.

As with the size 25 model, the exiting probability to $\overline{12}$ is not close to the ideal value of 0.05, but again is around the correct order of magnitude.

5.2.2 Simple Navigation

Size 10 (Figure 5.8)

The size 10 Simple Navigation HMM is comprised mostly of a small cluster of five states in the middle. This cluster is highly interlinked with transitions from nearly every state to every other state within the cluster. Although it is not shown here, each state possesses a very high self-transition probability of around 0.7, with the exception of $\overline{3}$, which does not transition to itself. These high self-transition probabilities indicate that states mostly occur several times in succession before transitioning to another state. Most of the states here have been identified as *cruising*, that is moving forwards without turning and no obstructions. The cruising behaviour here can be strongly associated with being close to the origin (near the start of the task in this case), or far from the origin near the finishing waypoint. There are also different speeds of cruise observed within this model.

$\overline{0}$ seems to correspond to a slow movement behaviour, which is always seen on the robot when it is trying to finalise its position whilst reaching a waypoint. In this case, the slow movement will be the robot trying to find the finishing waypoint for the navigation task. For this reason, this can be seen as a 'finishing' behaviour.

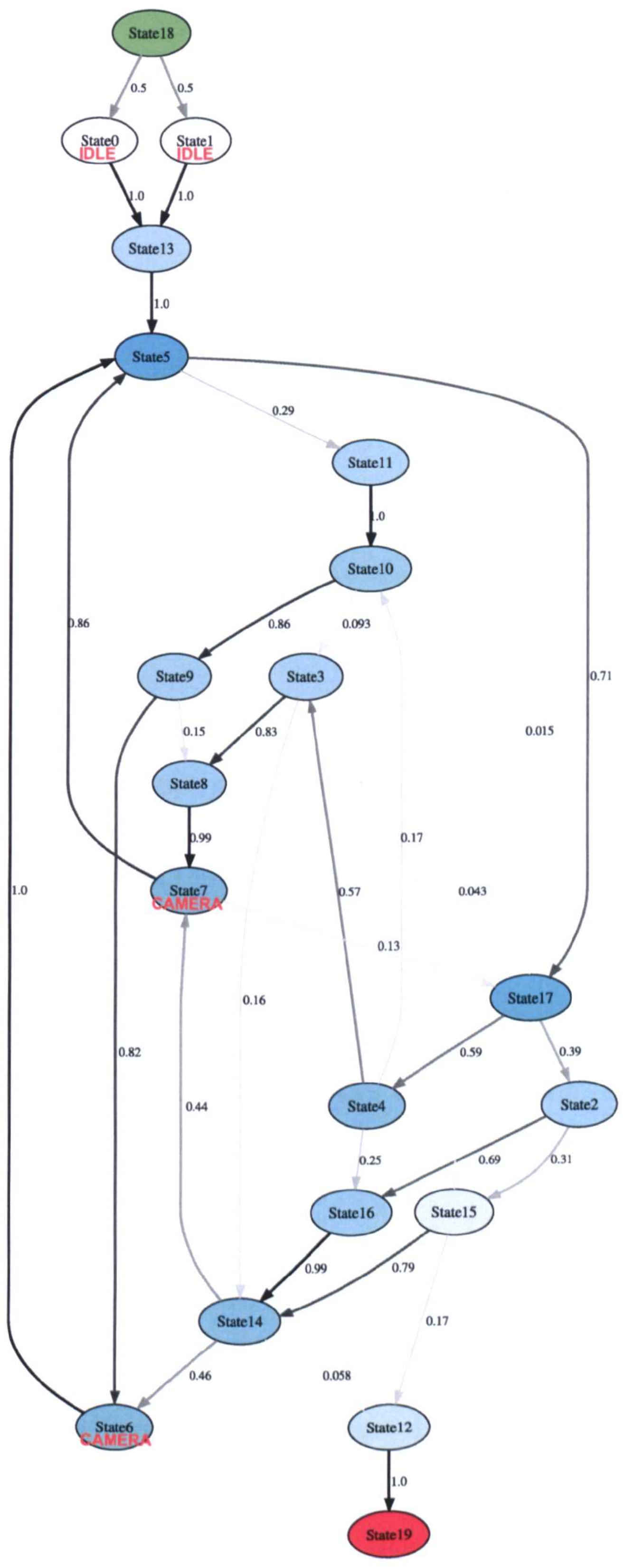


Figure 5.7: *HMM for Panoramic Photo with size 30 network*

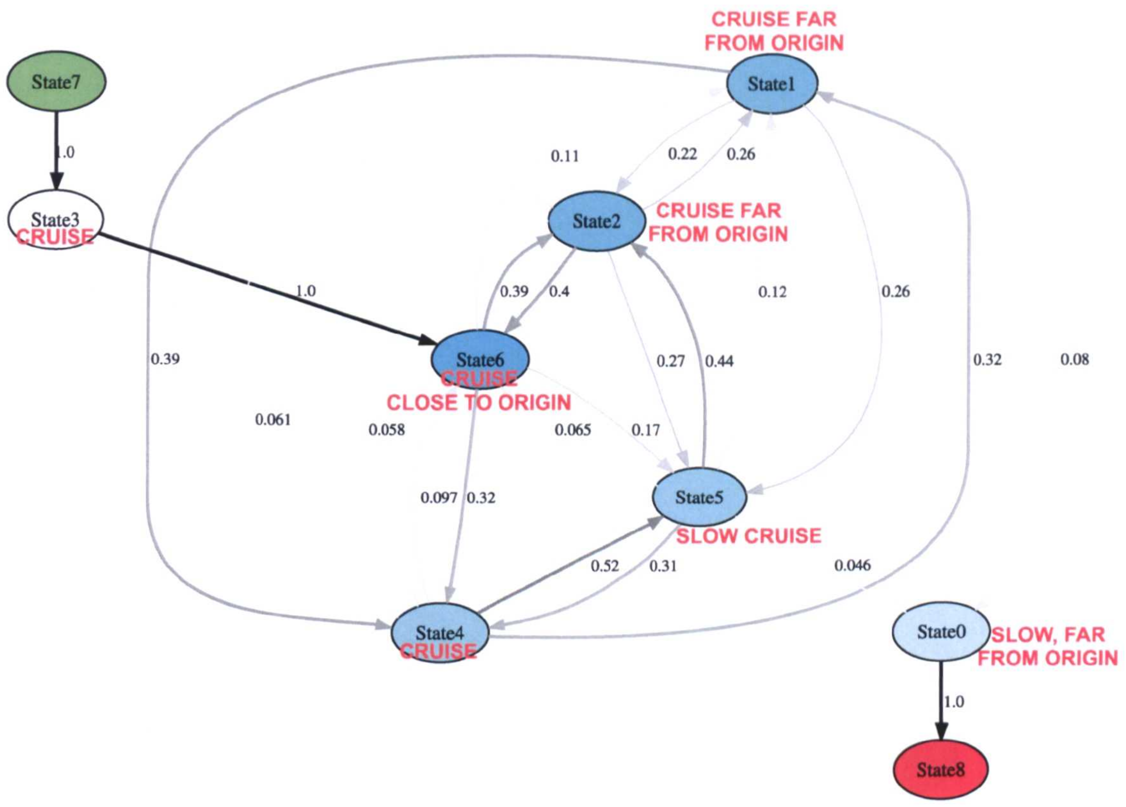


Figure 5.8: *HMM for Simple Navigation with size 10 network*

The only state in this HMM that is associated with turning is $\overline{2}$, which only weakly corresponds with turning right and is therefore not marked. There are no states in this model that are associated with turning left. This lack of a left-turning behaviour suggests that either the robot did not turn left during the main part of the execution, or the Kohonen Map used to learn the HMM was too small. Since the robot *did* turn left during execution, it can be seen that a size 10 model is too small for capturing this task effectively.

Size 20 (Figure 5.9)

Although the Kohonen network used here has twice the dimension length as the size 10 network, there is only one more state present. The "cluster of states" feature identified above is also here, consisting of states $\overline{0}$, $\overline{3}$, $\overline{7}$, $\overline{1}$, $\overline{4}$ and $\overline{6}$. Once again, omitted from the diagram are the self-transition probabilities that are very high for all of the states except $\overline{2}$ and $\overline{5}$.

In this network there is a lot more discrimination between the states. For example in the size 10 HMM, there was no state associated with turning left, but this is present here in both $\overline{0}$ and $\overline{6}$. Curiously, $\overline{6}$ is also the only state associated with reversing.

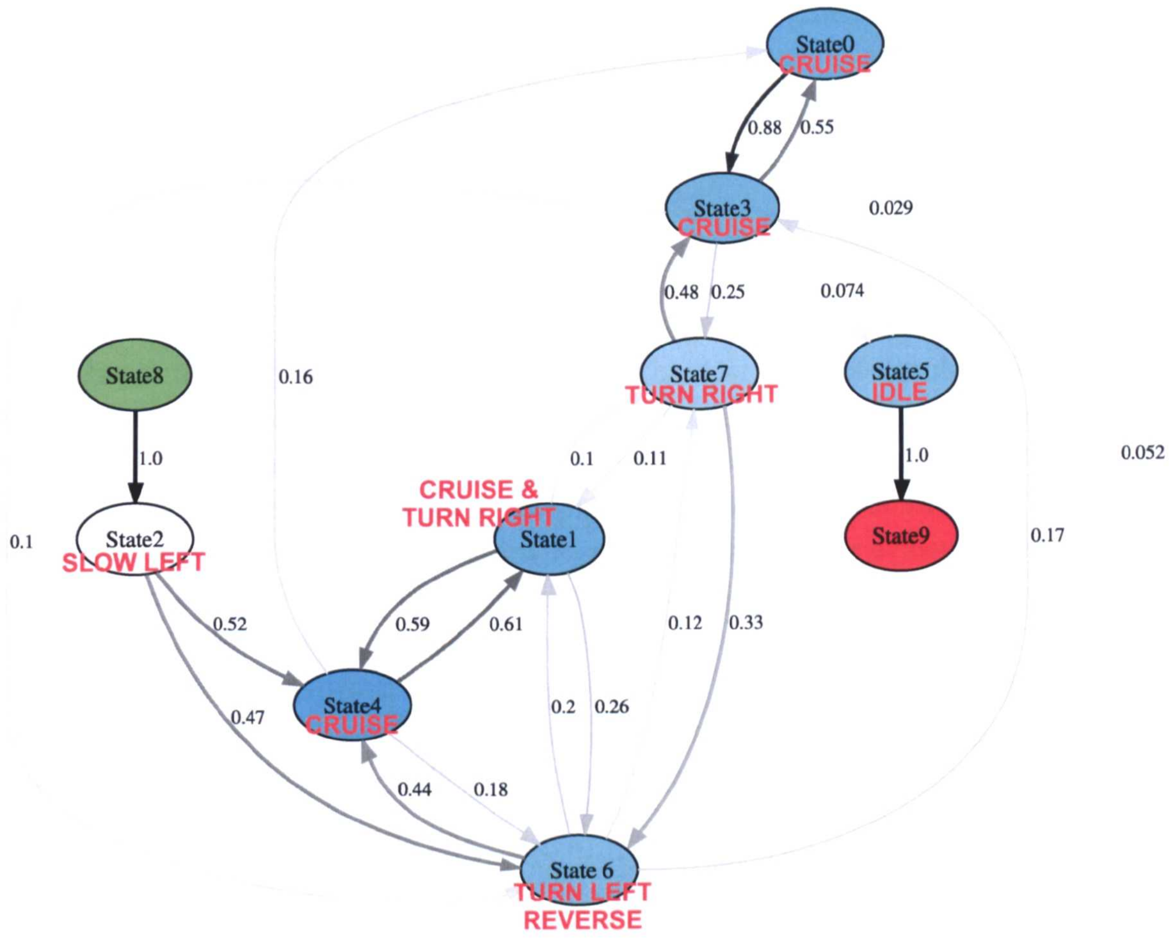


Figure 5.9: *HMM for Simple Navigation with size 20 network*

Size 25 (Figure 5.10)

This HMM is much more complex than the size 20 network with many transitions between states. Because of the size of the network, it is difficult to see patterns in the structure. There are, however, individual states associated with turning left and right for both forwards and backwards motion. This is a structure that was not seen in the smaller networks for the Simple Navigation task, indicating that this network possesses a good discrimination between states.

With this model (and also with other models for the Simple Navigation task) it is necessary to label some of the states with ambiguous names such as 'indecision' or 'slow right' because there is no strong defining characteristic for the behaviour associated with some states. The task did not possess enough recognisable structure to make manual identification of these states simple.

Size 30 (Figure 5.11)

Much larger than the size 25 network, this HMM has lost much of the obvious cluster structure that has been evident in all of the smaller models, and has been replaced with some very different structures. There was a second model learnt from this data with slightly different variables (not shown here) that possessed a linear structure, where states were each visited in turn with few loops or shortcuts. This evidence points towards the fact that the model has become susceptible to over-learning and is identifying structures specific to this task, rather than generic states for Simple Navigation. For this reason, the Kohonen Network used here was probably too large and the HMM is not a useful model for use in a generic situation. If an even larger network were used, it is expected that the structures would break down further.

5.2.3 Gradient Navigation

Size 10 (Figure 5.12)

The main feature in this HMM is the cluster of interlinked states on the left of the diagram. This is very similar to the structures that were seen in the Simple Navigation task. Two of the states in the cluster correspond to cruising far from the origin, and the other two represent reversing plus turning (left or right). $\overline{0}$ is also associated with localisation, but this is not a strong association.

The implementation of the Gradient Navigation task consists of navigation between several waypoints. It starts at the origin, moves to a waypoint far from this and then returns back to a location near the start (see Figure 4.5). Aside from the cluster on the

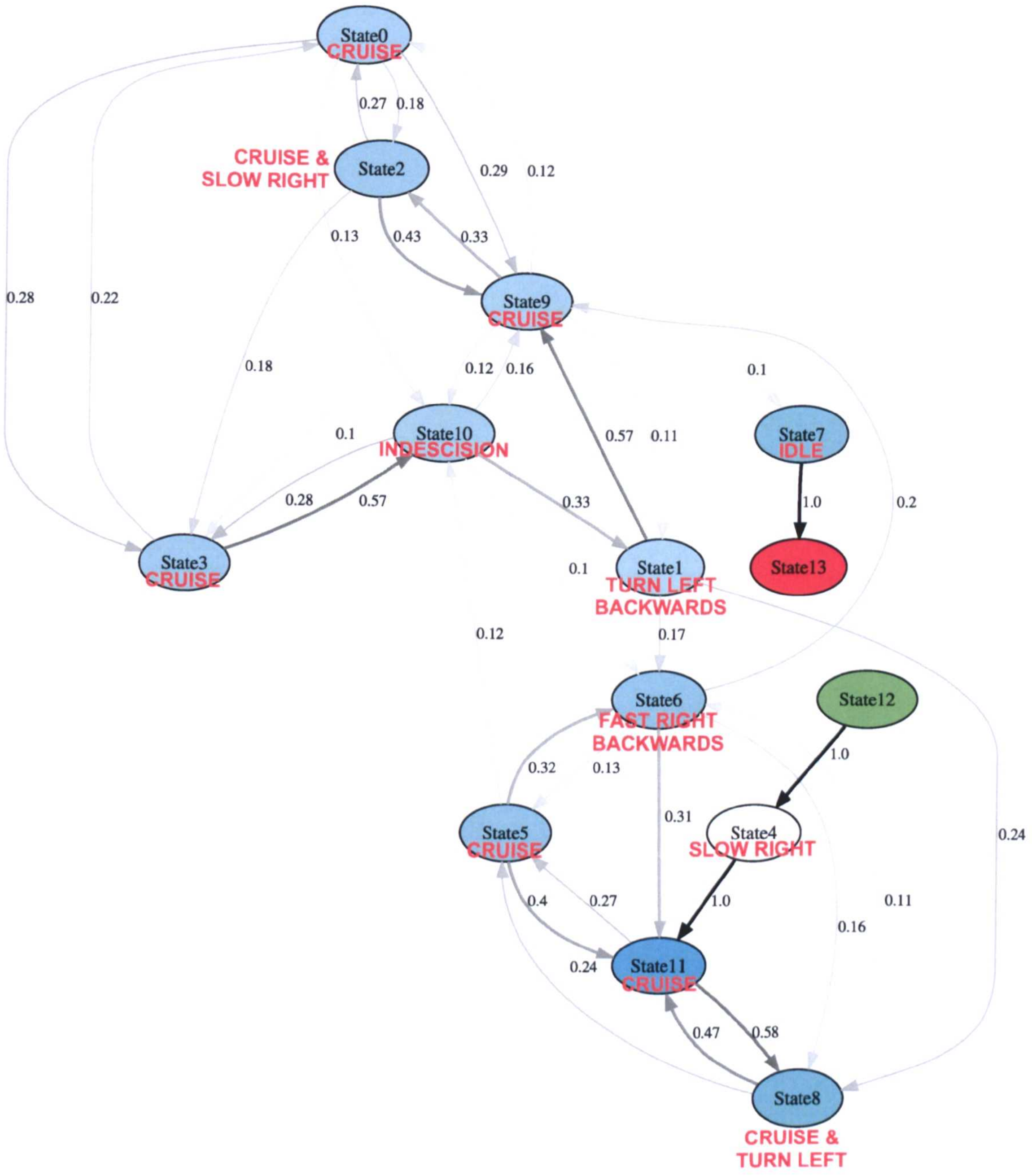


Figure 5.10: *HMM for Simple Navigation with size 25 network*

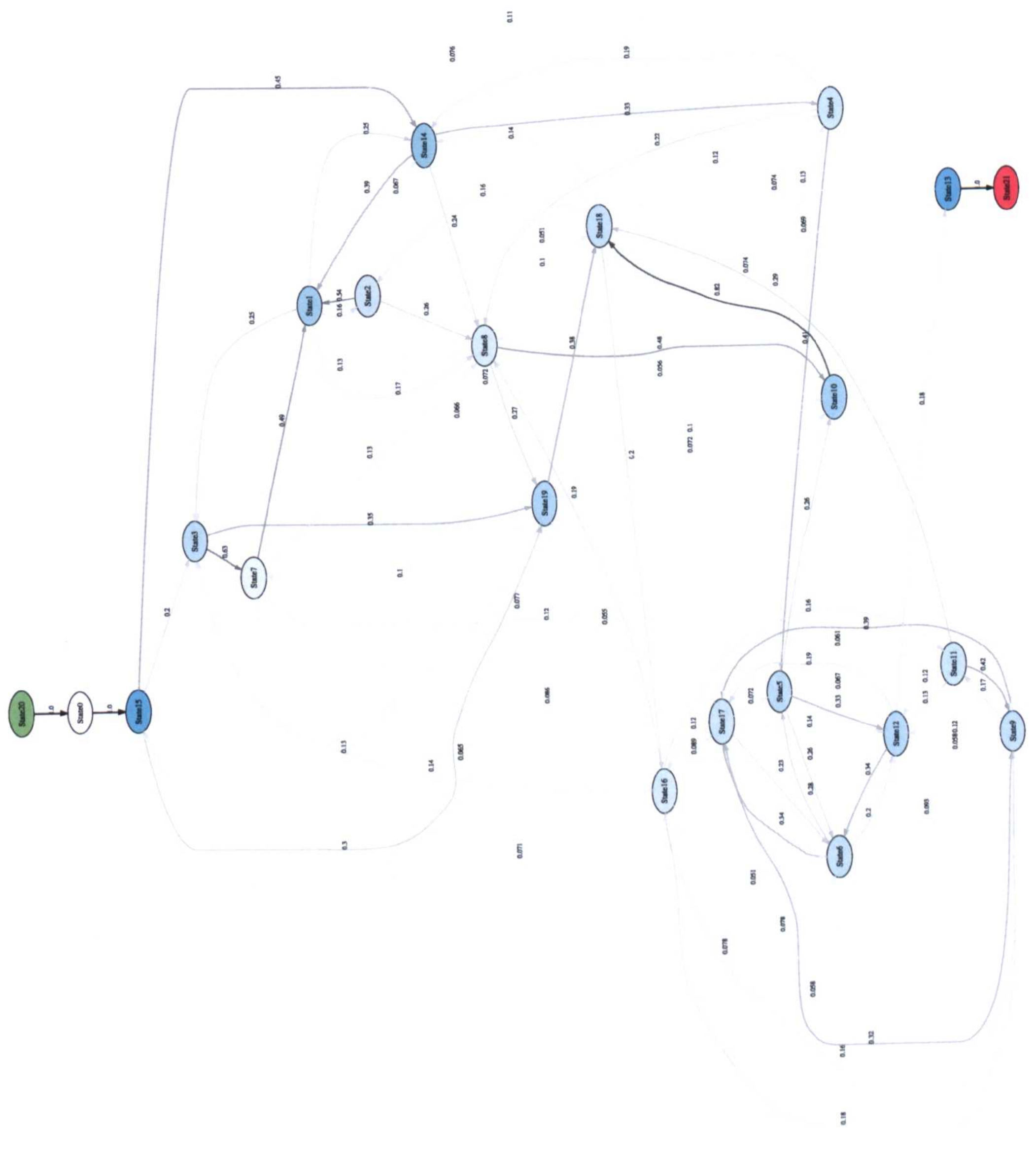


Figure 5.11: *HMM for Simple Navigation with size 30 network*

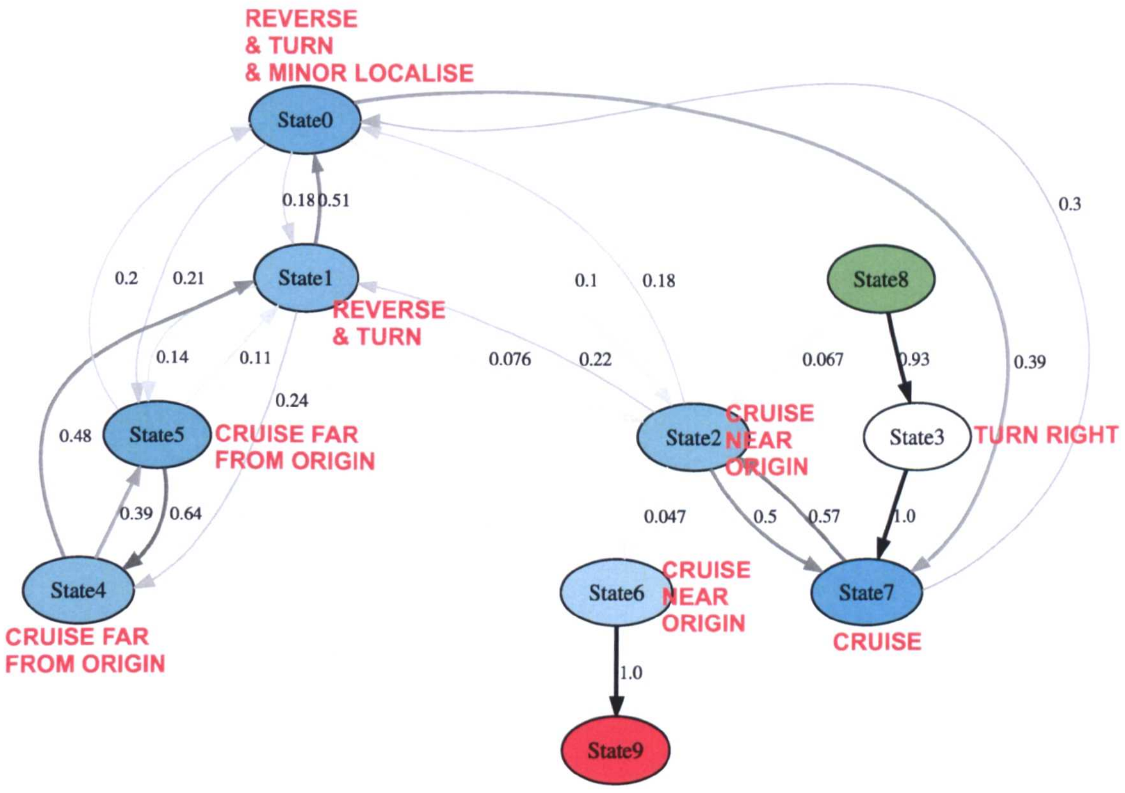


Figure 5.12: *HMM for Gradient Navigation with size 10 network*

left, there are three states associated with cruising close to the origin ($\overline{2}$, $\overline{7}$ and $\overline{6}$). It can be seen that the model can start in these states and then moves into the cluster on the left for the main body of the task. Once the robot returns close to the origin, it can return to this part of the model before exiting through $\overline{6}$.

It should be noted that with the Gradient Navigation task it was difficult to accurately identify the start and finish states, which are both required whilst learning the model. This was the case for all of the HMM sizes used. Identification is difficult due to the fact that the start and finish states for this task are very similar; the states are both close to the origin, both have low speed and both possess low angular change. A certain amount of guesswork and intuition was required in these cases.

Size 20 (Figure 5.13)

Once again, there is a similar structure to the size 10 HMM, with essentially the same structures present. The cluster is still present on the left, but no longer possesses states that correspond to reversing. The right-hand side of the HMM has become simplified and is mostly dominated by $\overline{3}$ and $\overline{2}$ which produce a turn/cruise behaviour along with localisation.

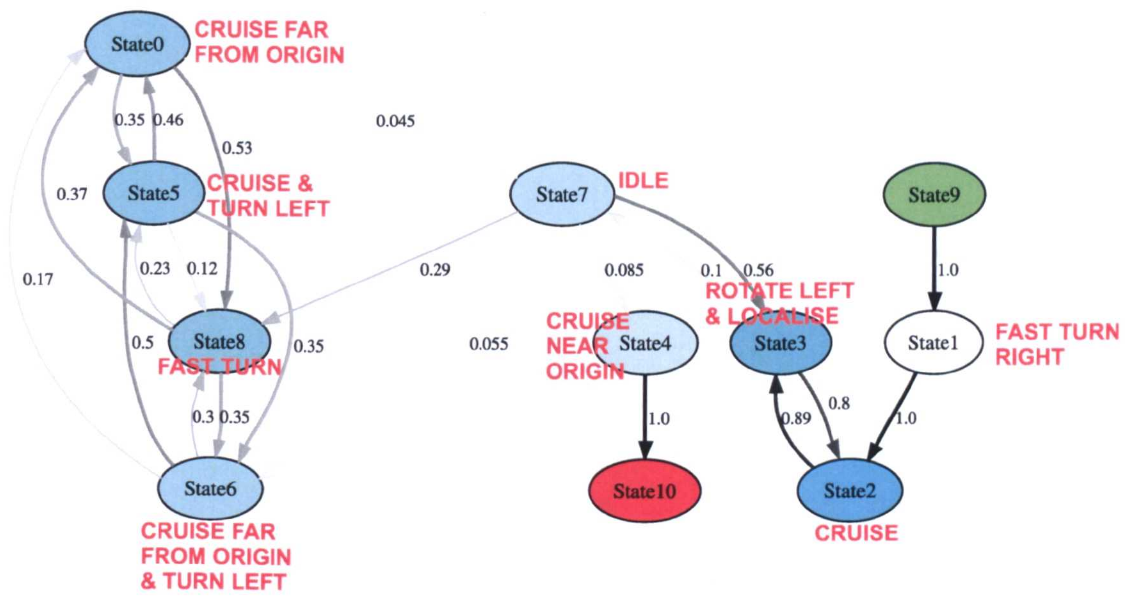


Figure 5.13: *HMM for Gradient Navigation with size 20 network*

Size 25 (Figure 5.14)

For the size 25 network, a much more complex HMM has been learnt. Manually identifying states and structures within this HMM is too difficult.

Size 30 (Figure 5.15)

The size 30 network has been included here for completeness. As with the size 25 network, it is too large to attempt to manually identify features.

5.2.4 Panoramic Photo with Errors

Please note that within this section two HMMs are shown for each model. The first of each omits self-transitions and the second includes these. This has been done to allow the structures (specifically the induced failure states) to be identified with ease. Please note that transitions that are included on one of the diagrams may be omitted from the other due to it being a low-probability transition below 0.05.

Size 10 (Figures 5.16 & 5.17)

This HMM is surprisingly simple: perhaps too simple to encompass the whole structure of the task. There is a state that is associated with the camera ($\overline{2}$) and two states for turning ($\overline{0}$ and $\overline{1}$). The typical execution of this HMM remains mostly in $\overline{0}$ and $\overline{1}$, and oscillates between the two. Occasionally, the transition from $\overline{1}$ to $\overline{2}$ occurs, albeit with very low probability. This probability is too low to be shown in Figure 5.17. The model

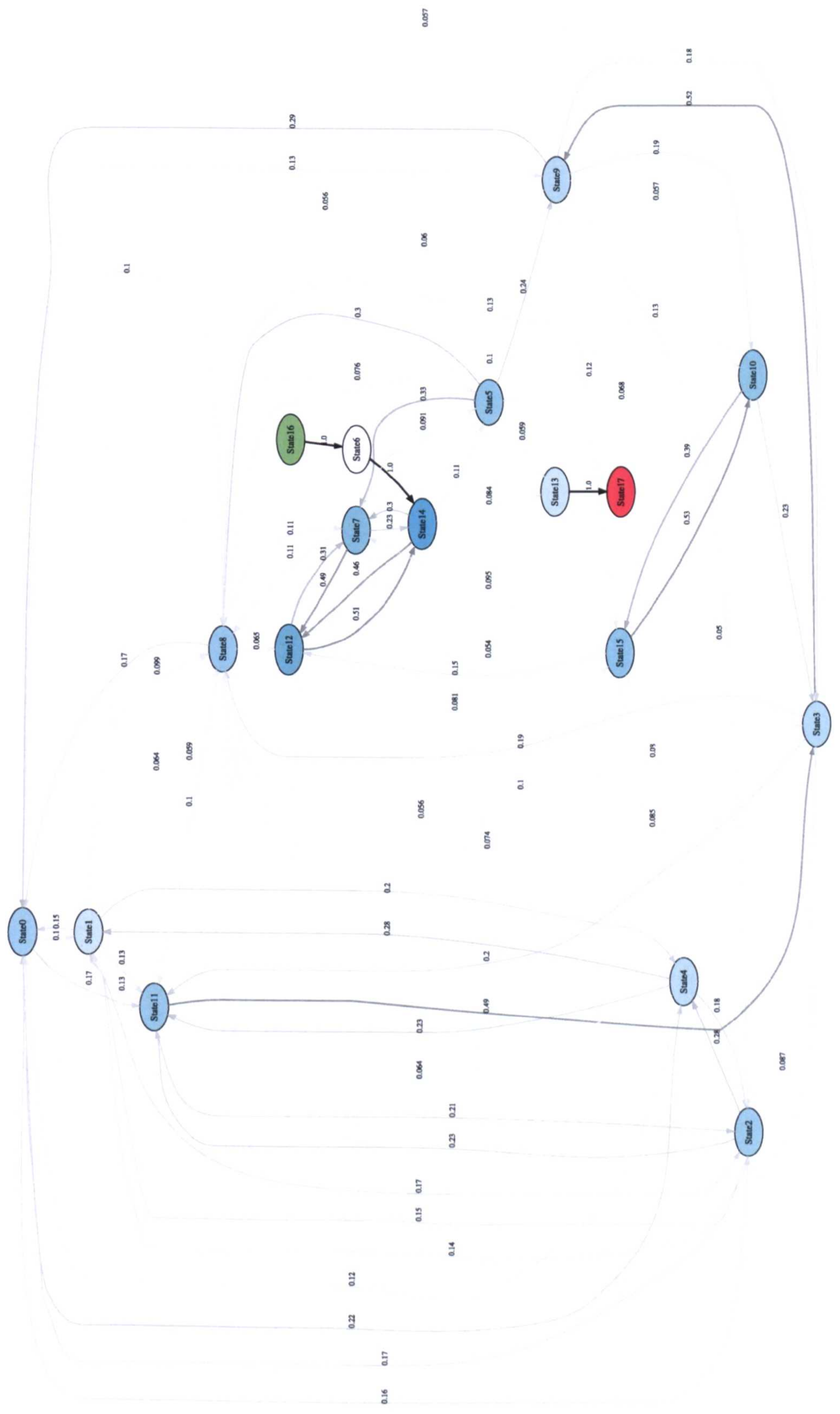


Figure 5.14: *HMM for Gradient Navigation with size 25 network*

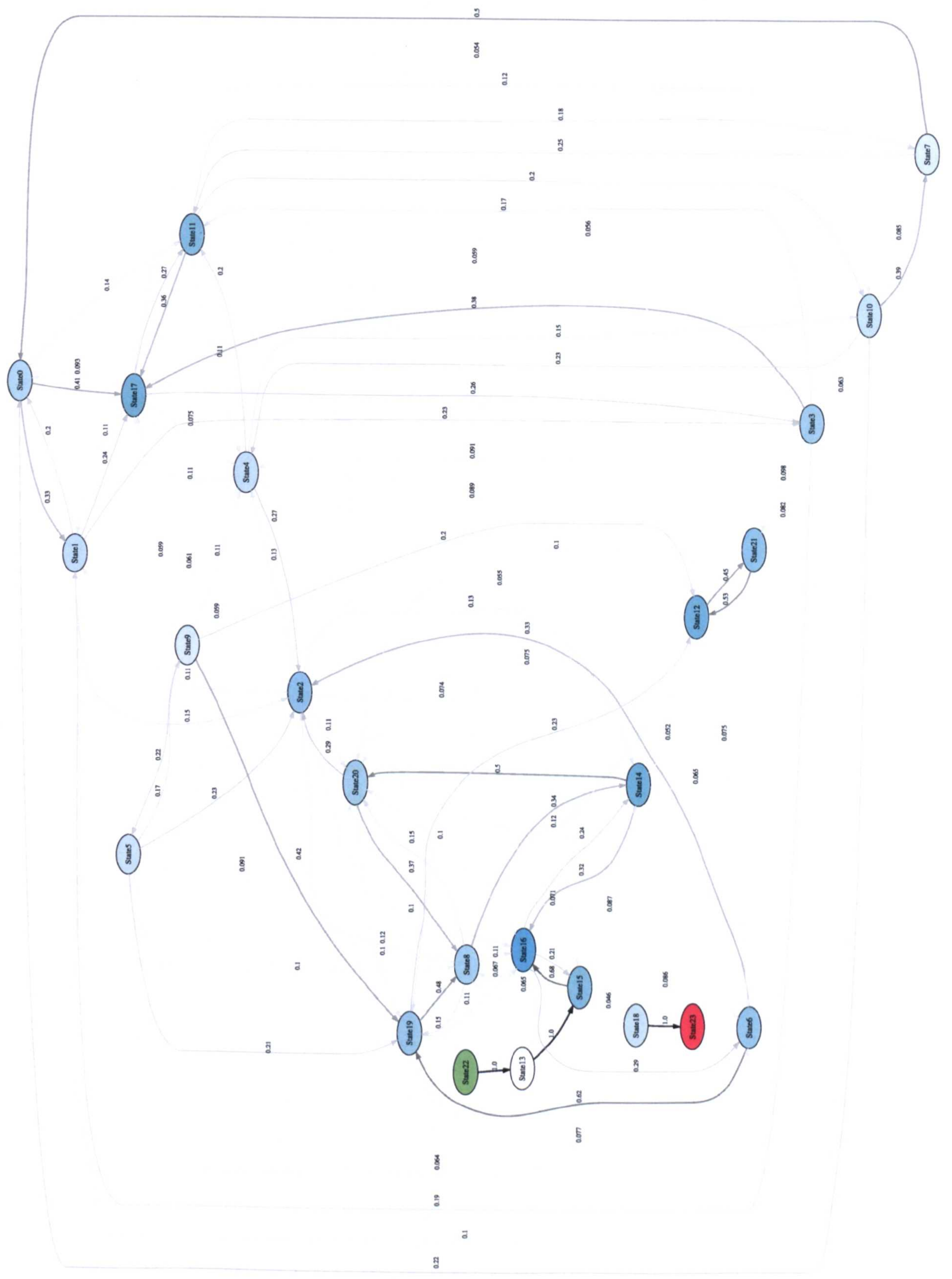


Figure 5.15: *HMM for Gradient Navigation with size 30 network*

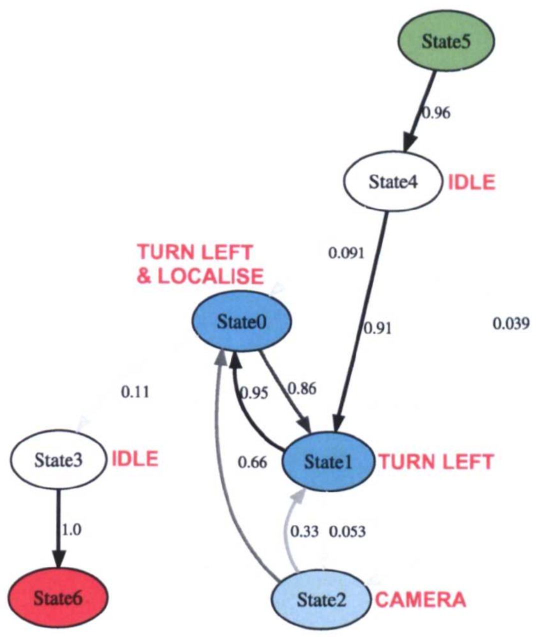


Figure 5.16: *HMM for Panoramic Photo with Errors size 10 network*

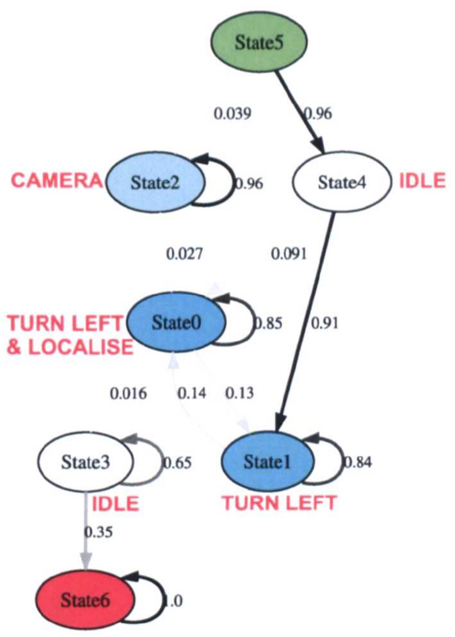


Figure 5.17: *HMM for Panoramic Photo with Errors size 10 network (including self-transitions)*

can exit through $\overline{3}$ (which indicates that the robot is idle), pausing briefly before the end of the task.

Since this HMM represents a task that includes induced errors, a good model should include state(s) that can be traced to these errors. This model does not possess these. Also, there are no states that are associated with any form of localisation behaviour, suggesting that the model is not large enough to represent all the observed behaviours.

Size 20 (Figures 5.18 & 5.19)

This model has a much more defined structure than the size 10 HMM, with an easily identifiable loop between $\overline{0}$, $\overline{1}$ and $\overline{7}$. This corresponds to turning (with angle localisation) and then taking a photo. There is an optional detour to $\overline{4}$ that allows for extra turning and position adjustment if required.

Perhaps the most interesting feature can be identified when self-transitions are included. This shows two states, $\overline{2}$ and $\overline{6}$ that correspond to the camera shutter being open. These states have very low probabilities of being transitioned to, combined with very high self-transition probabilities of around 0.95. This evidence points towards these two states corresponding to the induced error of the camera shutter being stuck open. The HMM has learnt this aspect of the behaviour correctly and has allocated these states that could potentially be used to identify this type of failure.

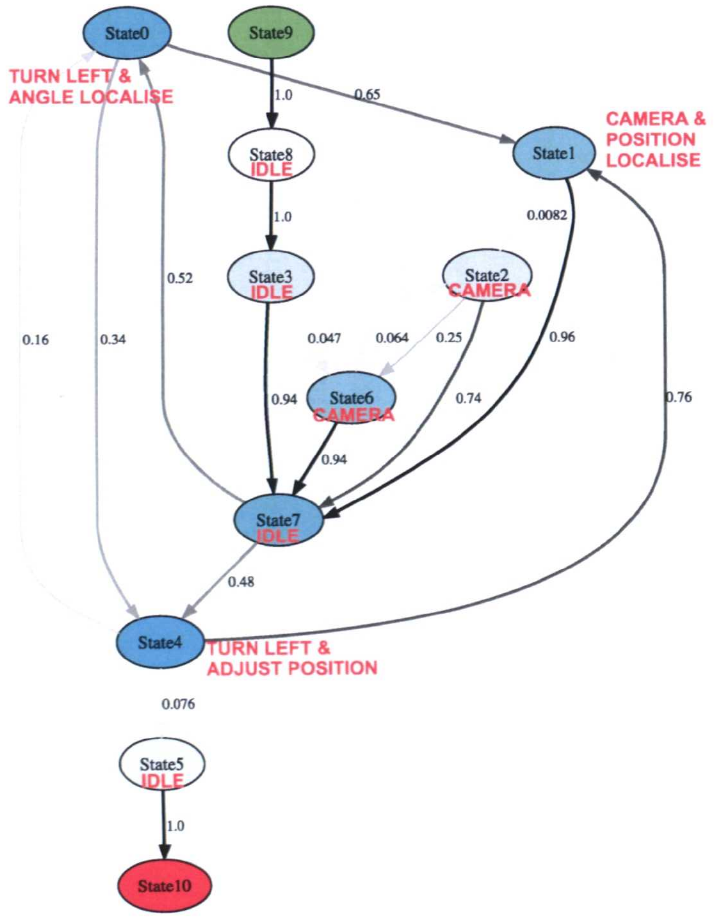


Figure 5.18: *HMM for Panoramic Photo with Errors size 20 network*

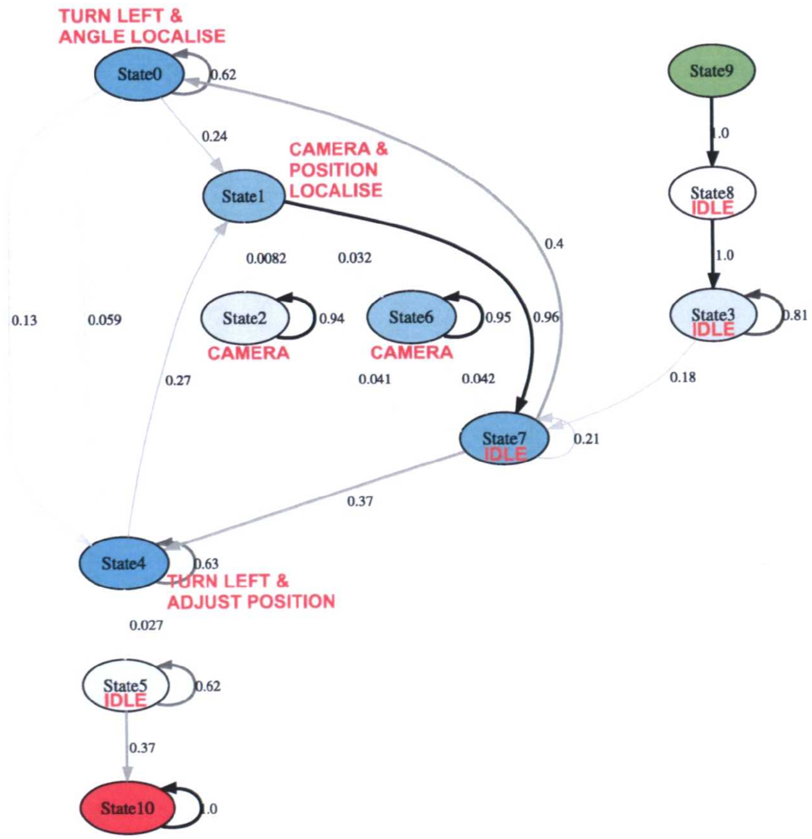


Figure 5.19: *HMM for Panoramic Photo with Errors size 20 network (including self-transitions)*

Size 25 (Figures 5.20 & 5.21)

With this model, the HMM starts to get much more complex, but structures are still visible and identifiable. Although not immediately apparent, there are two loops in this HMM. The first is from $\overline{7} \rightarrow \overline{14} \rightarrow \overline{6} \rightarrow \overline{4} \rightarrow \overline{15} \rightarrow \overline{17} \rightarrow \overline{7}$. The second is from $\overline{2} \rightarrow \overline{1} \rightarrow \overline{0} \rightarrow \overline{18} \rightarrow \overline{3} \rightarrow \overline{5} \rightarrow \overline{2}$. These loops are almost identical and contain similar features, including the camera shutter being active as well as turning and localisation. There are several transitions between the two loops, but these have low probabilities: once a loop is entered into, it is rarely exited. The fact that there are two loops here when one would apparently do suggests that this model may have over-learnt differences between the two. An alternative possibility is that there is a subtle difference between the two loops that is not obviously apparent under visual examination.

Apart from the starting and finishing behaviour states, there are two states that are not in the main loops, $\overline{12}$ and $\overline{13}$. These are both strongly associated with the camera shutter being open and, like before, have low probabilities of occurring as well as high self-transition probabilities. These most likely correspond to the induced error states.

Size 30 (Figures 5.22 & 5.23)

As with the size 25 model, there are two main loops in this diagram that correspond to the TAKE PHOTO, TURN behaviour. In addition, there is a cluster of states consisting of $\overline{10}$, $\overline{1}$, $\overline{11}$, $\overline{2}$ and $\overline{18}$, which all correspond to the camera shutter being open. With the high self-transition probabilities and low probabilities of occurring, these are probably the induced error states. The large amount of repetition in the camera shutter states suggests that this model has overlearnt this feature, where fewer states would have sufficed. This indicates that this model may be too large.

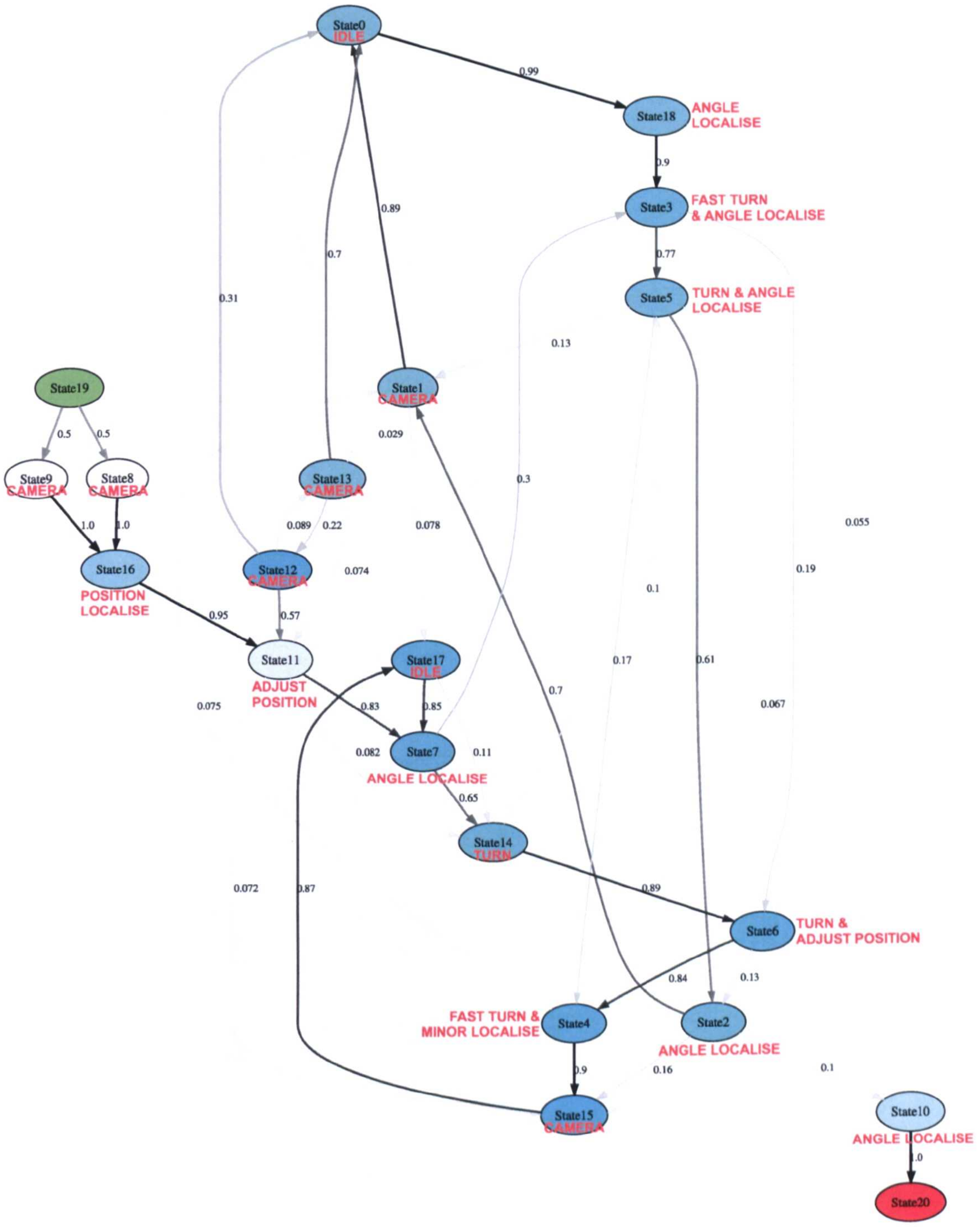


Figure 5.20: *HMM for Panoramic Photo with Errors size 25 network*

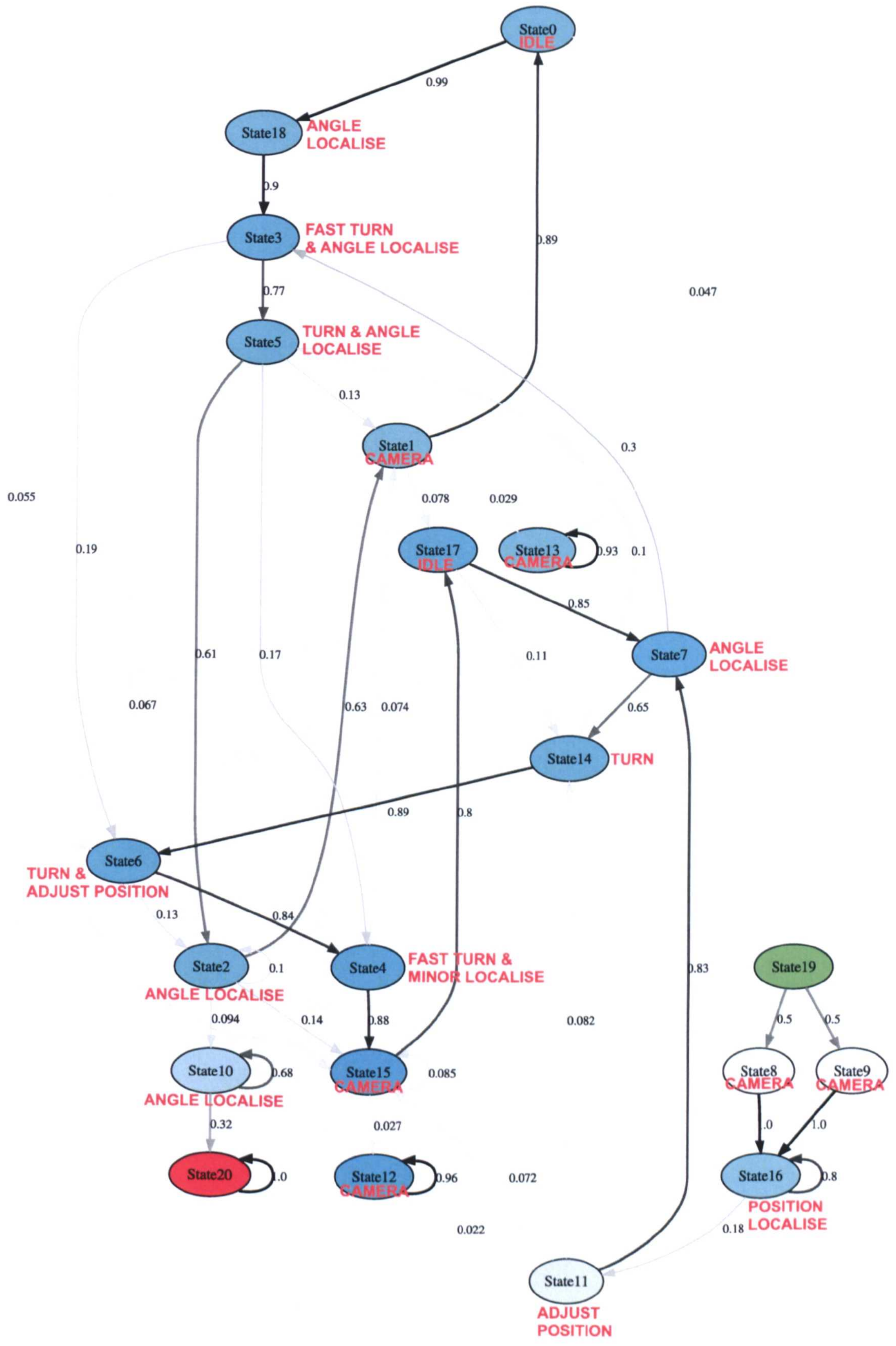


Figure 5.21: *HMM for Panoramic Photo with Errors size 25 network (including self-transitions)*

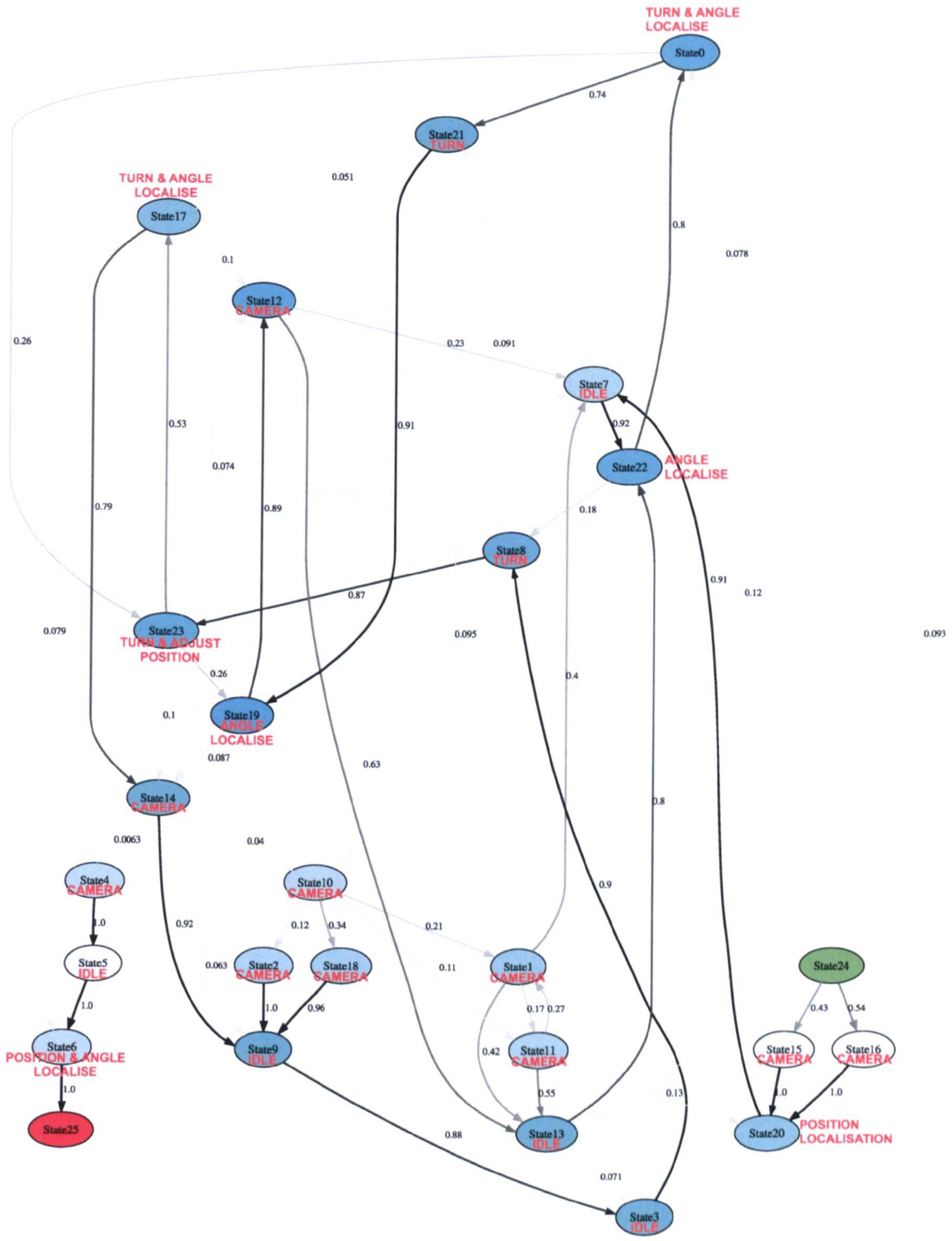


Figure 5.22: *HMM for Panoramic Photo with Errors size 30 network*

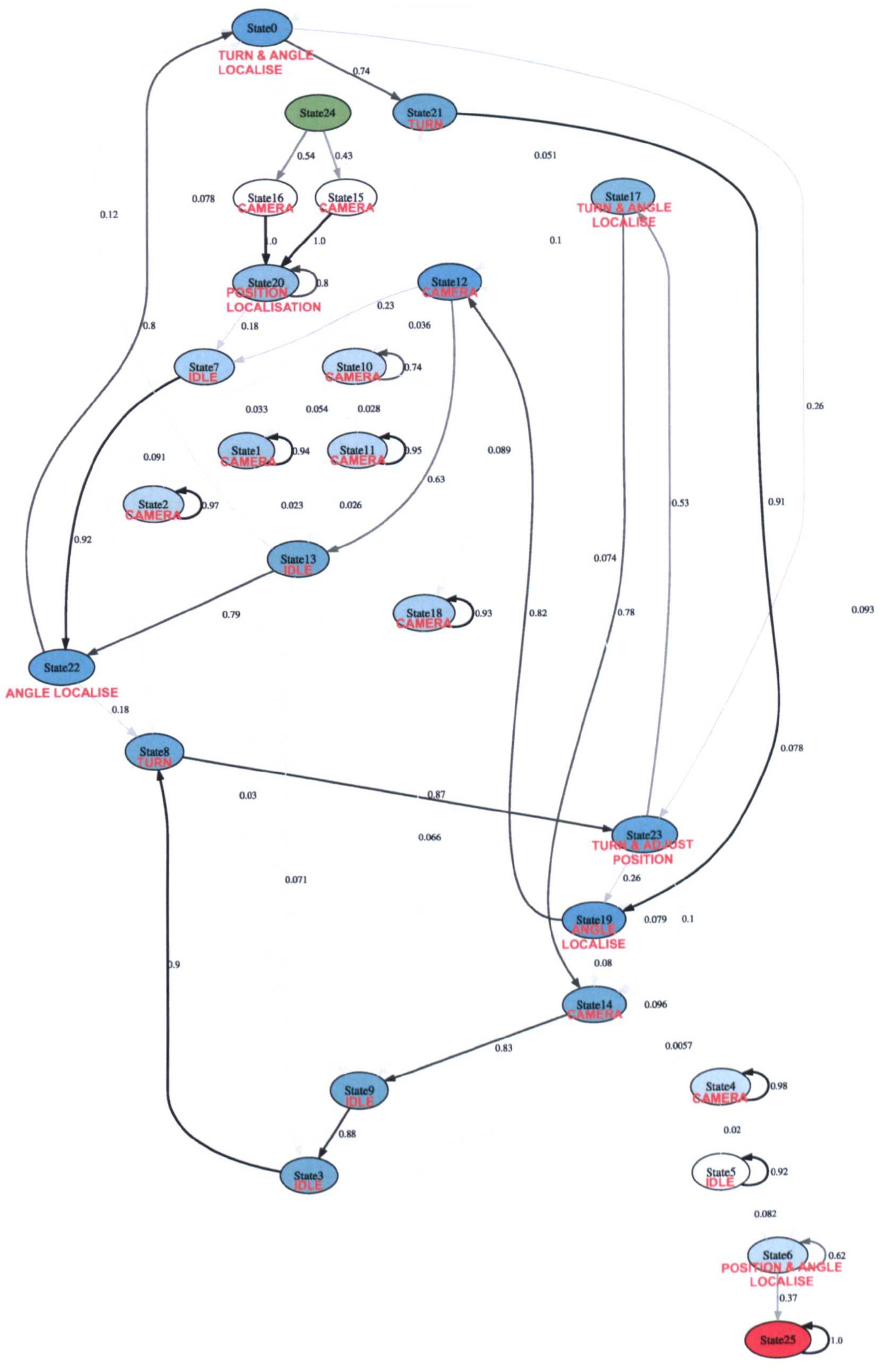


Figure 5.23: *HMM for Panoramic Photo with Errors size 30 network (including self-transitions)*

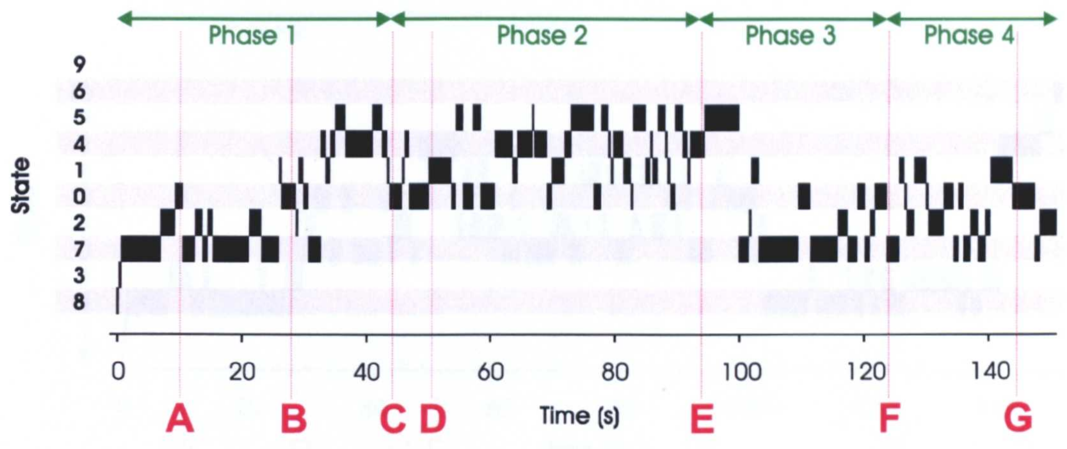


Figure 5.24: *HMM state sequence for Gradient Navigation using Size 10 HMM*

5.3 HMM Trajectory Analysis

In Section 5.1, the raw data from the robot was analysed from a particular execution for two of the tasks. During this process, a number of locations were labelled on the graphs that had been manually identified. Section 5.2 explored the HMMs that were learnt from the raw data, and attempted to identify many of the states present. It must be strongly emphasised that these analyses were done independently of one another, and the HMMs were learnt with no input of the manually identified features. The correlations listed below are an indication of the strong learning capability of the models, and not a by-product of the learning process.

The two analyses performed so far can be correlated by turning the raw data examined earlier into sequences of states by plugging the data into the HMMs. This allows the identified locations in the raw data to be compared directly to the identified HMM states at those locations. The graphs in this section show these HMM state sequences for several models to illustrate how the robot passed through the models during the course of execution. For curiosity, it is possible to directly compare the numbered states below with the raw data graphs and the HMMs to see how execution passes through these, but this is not necessary. The numbered states along the vertical axis in each graph have been rearranged to best illustrate the structure of the task and the transitions between the states. Also note that there are some states that are not visited by the executions analysed below.

5.3.1 Gradient Navigation

Size 10 (Figure 5.24)

Immediately apparent in this graph are the different phases of the task, as identified previously in Sheet 1 of the raw data. Although the structures do not line up perfectly

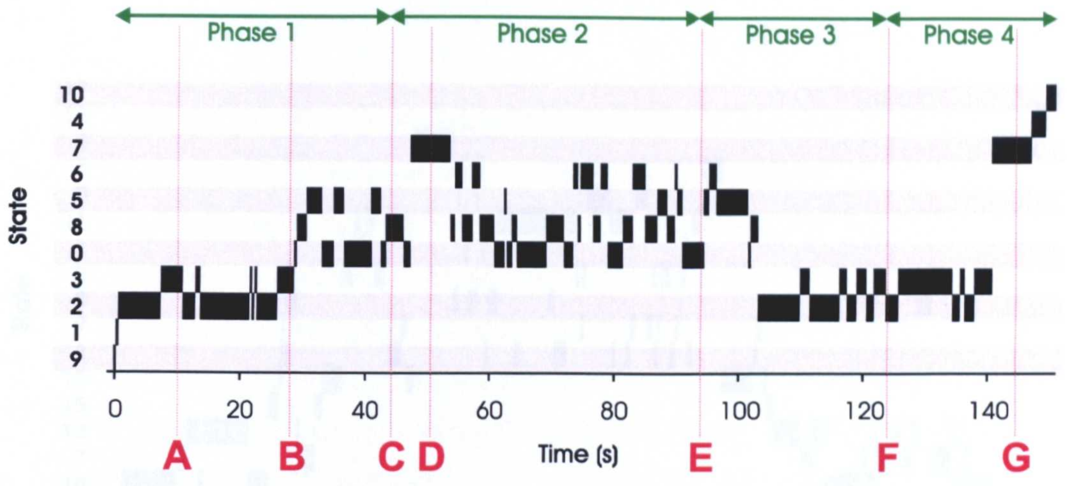


Figure 5.25: *HMM state sequence for Gradient Navigation using Size 20 HMM*

with the phases, the general outline can be seen. The first and third phases loosely correspond to $\overline{2}$ and $\overline{7}$, which were both classified as "cruise" states in the HMM (Figure 5.12, p.58). There is a cluster of $\overline{0}$, $\overline{1}$, $\overline{4}$ and $\overline{5}$ which make up the main body of the task during the second phase. Points A and B, previously identified in the raw data as "obstacle avoidance" are both represented by $\overline{0}$. This state was manually identified in the HMM as "reverse, turn and minor localise", which can easily be seen as an obstacle avoidance behaviour.

Size 20 (Figure 5.25)

Similar to the Size 10 sequence above, this follows the same basic pattern but in a more pronounced manner. There is an oscillation between $\overline{2}$ and $\overline{3}$ at the beginning and end of the task, and then the main body of the task is made up of the cluster $\overline{0}$, $\overline{8}$, $\overline{5}$ and $\overline{6}$. In this model, the "obstacle avoidance" at Points A and B both correspond to $\overline{3}$. The clearest correlations here though are Points D and G which are very obviously associated with $\overline{7}$. These two points were classified in the raw data as "collision recovery", and have been allocated their own state here. During the HMM analysis, $\overline{7}$ was labelled as "idle" (Figure 5.13, p.59), but this discrepancy may be due to the HMM state not having a pronounced association with any particular behaviour on visual examination.

Size 25 (Figure 5.26)

Once again, the same overall structure exists, except with a lot more detail and definition. The "collision recovery" points at D and G have, like before, been classified into one state, $\overline{5}$, which does not occur at any other time. Due to the visual complexity of the learnt model, the HMM states were not labelled when previously examined (Figure 5.14, p.60) but it is now clear that $\overline{5}$ is associated with collision recovery.

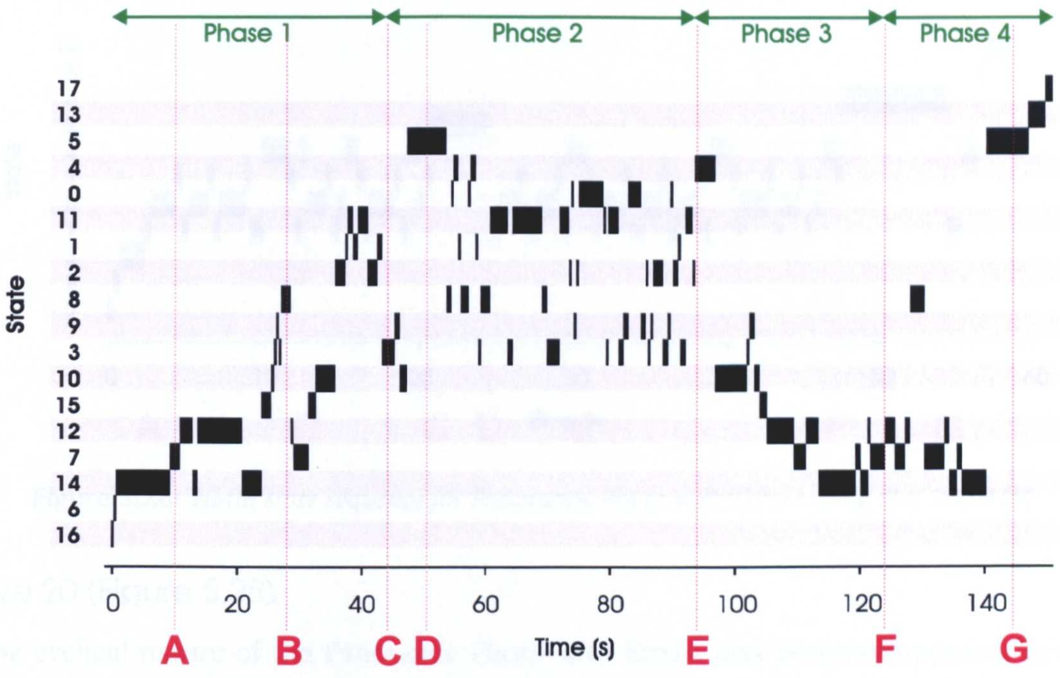


Figure 5.26: *HMM state sequence for Gradient Navigation using Size 25 HMM*

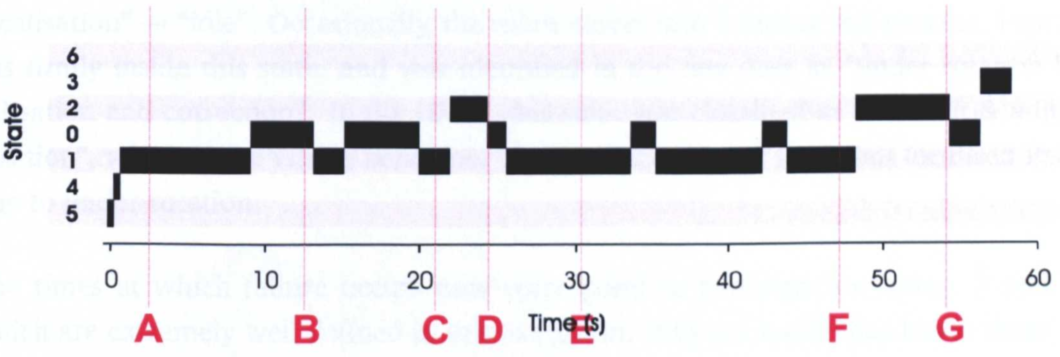


Figure 5.27: *HMM state sequence for Panoramic Photo with Errors using Size 10 HMM*

5.3.2 Panoramic Photo with Errors

Size 10 (Figure 5.27)

In this model, the lack of states and discriminatory power is very apparent. Out of the seven states, only $\overline{0}$ and $\overline{1}$ are regularly visited throughout the course of the task. There are two occurrences of $\overline{2}$, and these correspond to the periods between C and D, and between F and G. When the raw data was examined, these periods were identified as the failure states where the camera shutter was stuck open. This matches with the label "camera" that was assigned to the HMM state when analysed earlier (Figure 5.16, p.62). This model, although small, has successfully captured the ability to detect these error states, but at the expense of not being able to detect the non-error camera states.

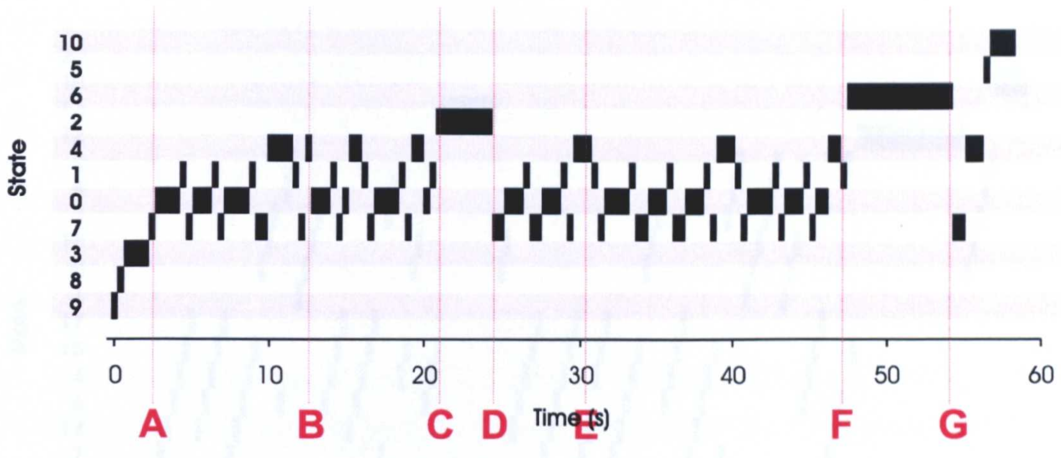


Figure 5.28: *HMM state sequence for Panoramic Photo with Errors using Size 20 HMM*

Size 20 (Figure 5.28)

The cyclical nature of the Panoramic Photo with Errors task becomes apparent here, with a clear cycle of $\overline{7} \rightarrow \overline{0} \rightarrow \overline{1} \rightarrow \overline{7}$. This sequence of states was identified in the HMM (Figure 5.18, p.63) as "idle" → "turn left & angle localise" → "camera & position localisation" → "idle". Occasionally, the robot moves into $\overline{4}$ during this process. Point E sits firmly inside this state, and was identified in the raw data as "under-rotation localisation and correction". In the HMM, this state was classified as "turn left & adjust position", which is the visible behaviour that would appear if the robot localised itself due to under-rotation.

The times at which failure occurs now correspond to two separate states, $\overline{2}$ and $\overline{6}$, which are extremely well defined in this execution. Why the model has learnt these as two separate states is unknown, but it suggests that there may be redundancy present in the model. In this HMM, $\overline{1}$ was previously identified as being associated with the non-error camera shutter. This occurs 18 times in this HMM, plus the two error camera shutter states makes the total of 20 camera events that make up the task.

Size 25 (Figure 5.29)

This model provides an even richer insight into the structure of the task. In the learnt HMM of this size (Figure 5.20, p.65), there were two cycles identified which can be seen in the upper and lower halves of the state sequence graph. Point A, which was identified as "initial position localisation" in the HMM, is associated with $\overline{11}$ in this execution. This state was labelled as "adjust position" in the HMM. There are also the two states that correspond to the failures, $\overline{12}$ and $\overline{13}$. In the HMM, $\overline{1}$ and $\overline{15}$ were identified as the non-error camera shutter states. Between them, these states occur 18

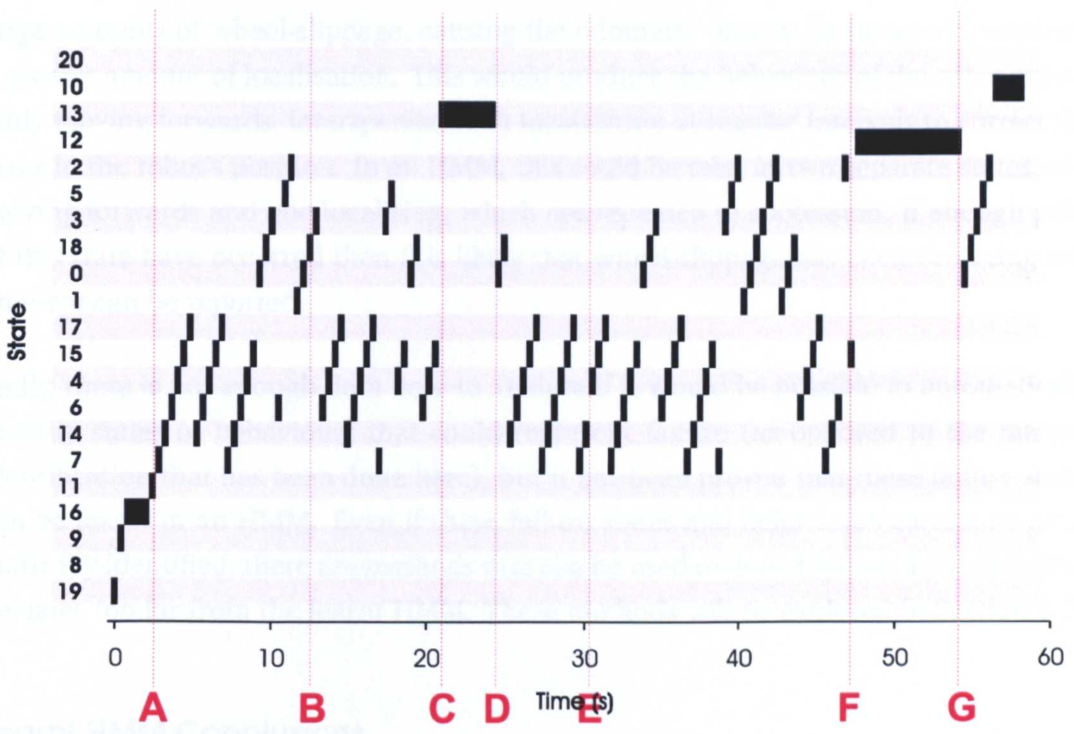


Figure 5.29: *HMM state sequence for Panoramic Photo with Errors using Size 25 HMM*

times, which makes the required 20 camera shutter events when the two error states are included.

In this execution, the "under-rotation localisation and correction" behaviour at Point E is represented not by a state, but by an omission of $\overline{6}$. In all of the other cycles, $\overline{6}$ always follows $\overline{14}$, but it is skipped and goes straight to $\overline{4}$. The transition $\overline{14} \rightarrow \overline{4}$ has an extremely unlikely chance of happening (0.3%), but it has been recognised as the most likely transition in this case. In the HMM, $\overline{6}$ was identified as being a "turn & adjust position" behaviour. The omission of this state suggests that the absence, as well as the presence of a state can be an indication of particular behaviours.

5.4 Failure State Identification

The Panoramic Photo with Errors task and the resultant HMMs show that it is possible to manually identify failure states within the system. In the general case though, it is unlikely that all failures are easily characterised by a particular state, but are more likely represented by a transitional behaviour across several states. In the case of the Panoramic Photo with Errors task, repeated occurrences of a particular state are excellent indicators of the shutter becoming stuck. In a more complex task, the executive may try several steps to recover, producing a set of states indicating failure. For example, a wheeled robot that manoeuvres itself onto a sandy surface may experience large amounts of wheel-slippage, causing the odometry data to be incorrect, requiring a greater amount of localisation. This would produce the behaviour of the robot apparently moving forwards, interspersed with localisation at regular intervals to correct the error in the robot's position. In an HMM, this could be seen as two separate states, one moving forwards and one localising, which are repeated in succession. If enough pairs of this state have occurred then it is likely that wheel-slippage has been occurring and an error can be reported.

Sadly, there is not enough data here to analyse if it would be possible to automatically identify states or behaviours that could represent failure (as opposed to the manual identification that has been done here), but it has been proved that these failure states can be learnt in an HMM. Even if these failure states and behaviours cannot be automatically identified, there are methods that can be used to detect failure if the executive deviates too far from the learnt HMM. These methods will be explored in Section 6.2.

5.5 Learnt HMM Conclusions

In this chapter, HMMs were learnt for the different task by using Kohonen Networks of varying sizes. These were compared back with the raw data by cross-referencing through the HMM state sequences produced by these models. In all cases, the Size 10 networks had problems with missing states, or not possessing a fine-grained representation of the tasks. Such models would not suitably represent the necessary aspects of the tasks. On the other hand, the Size 30 networks appeared to contain too many states and connections between these. This resulted in models that appeared to duplicate states and structures within the model. The ideal size Kohonen Network seemed to be between these values, around Size 20–25 depending on the complexity of the task. Judging the ideal network size is still a trial and error process however, and experimentation with different network sizes (as within this chapter) is probably required when deciding on network sizes for future work.

CHAPTER 6

Failure Detection

If everything seems under control, you're just not going fast enough

— *Mario Andretti*

THE ABILITY of an executive to examine its internal state through introspection, as discussed in Chapter 4 is of no use unless there is some form of feedback available. The state of a system must be analysed and interpreted before making a decision as to how to make sense of the data and best use this for control. The system has to track the progress of the task being carried out by the executive by using introspection, and then identify any anomalies. Once this has been done, it needs to prescribe a course of action that will minimise the disruption to the overall plan and the available resources. These actions may include everything from basic monitoring of general tasks, through to the detection of very specific error cases. There is a huge scope for different types, and indeed different amounts of control at such a level.

Figure 6.1 shows a proposed layout of such a system as described above. The raw data is collected and converted into a series of observations and, finally, states within an

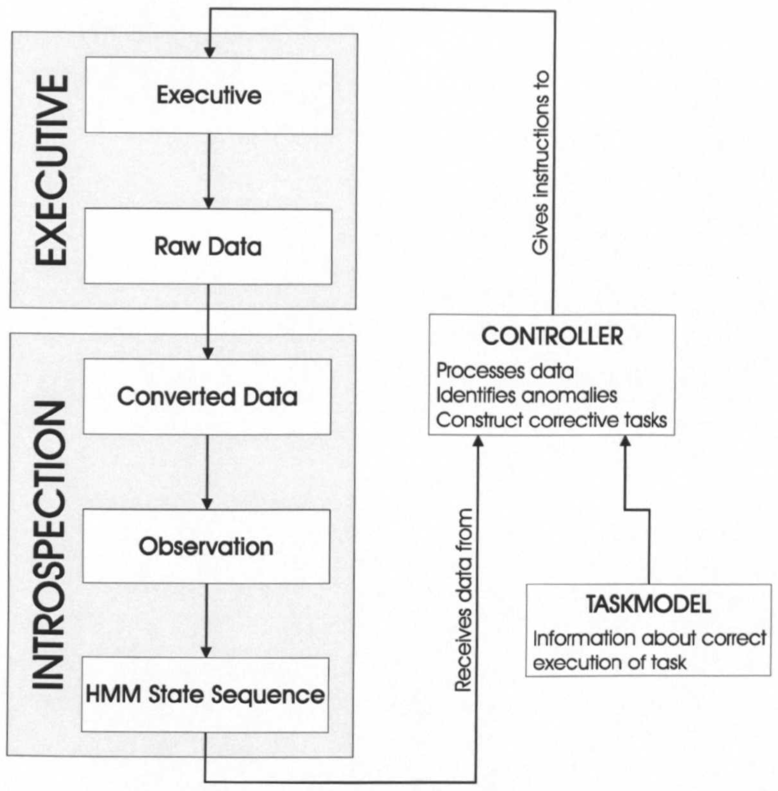


Figure 6.1: *The proposed structure for control*

HMM, as described earlier. This data can then be passed onto a *controller* that has information about how the task should progress, in the form of a *taskmodel*. A taskmodel contains a description of an HMM and also extra information that is useful for determining anomalies. This Chapter will explore what sort of extra information can be contained in a taskmodel that can be used for failure detection. During execution, the data received from the HMM can be compared to the taskmodel to determine deviations and what course of action, if any, should take place. This can then be passed back to the executive to continue execution.

When executing a task, there are two stages to the failure detection procedure:

1. Identify the most likely transitions through the HMM for the task so far
2. Decide if it is anomalous to see this sequence of transitions at this point during execution

This chapter will address both of these points, followed by Chapter 8 which examines the issue of opportunity insertion, which can be used when failure of a task occurs.

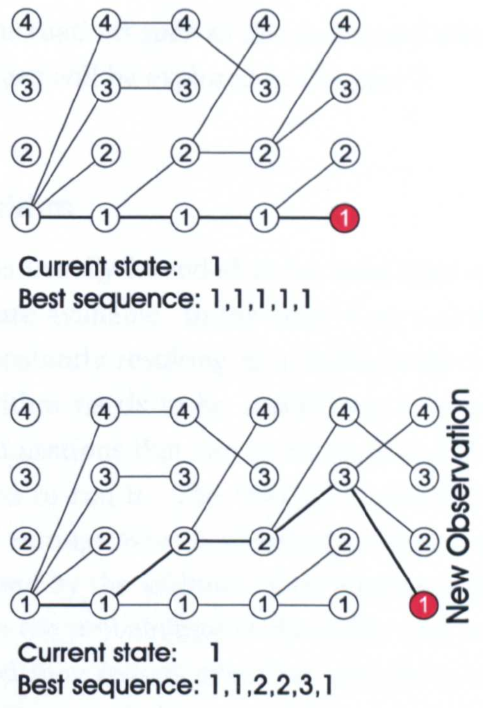


Figure 6.2: *How the Viterbi algorithm can produce fairly large fluctuations over time*

6.1 Determining HMM Sequences

The Viterbi algorithm [Russell and Norvig, 1995] can be used to establish the most likely state of the HMM during execution of the task. This is a dynamic programming algorithm for finding the most likely sequence of states in an HMM, given a series of observations. From the raw data collected, a series of observations is taken and then the Viterbi algorithm converts these into the most likely state sequence within the HMM.

In this chapter so far, the terminology *'indicative of failure'* has been used, rather than failure itself. This is because the algorithms described here are subject to fluctuations that could potentially falsely report a failure, even during a normal execution. For example, one of the disadvantages of using the Viterbi algorithm for state estimation is that the trajectory of states can fluctuate quite wildly over time when extra observations are added. In Figure 6.2, a sequence of:

$$\overline{1} \rightarrow \overline{1} \rightarrow \overline{1} \rightarrow \overline{1} \rightarrow \overline{1}$$

could quickly change to a sequence of:

$$\overline{1} \rightarrow \overline{1} \rightarrow \overline{2} \rightarrow \overline{2} \rightarrow \overline{3} \rightarrow \overline{1}$$

at the next timepoint, with the addition of a single extra observation that provided evidence that the latter sequence of states was more likely. Any mechanism that detects errors must allow for fluctuations such as this and minor errors to avoid reporting false positives. Such techniques will be explored in Chapter 7.

6.1.1 Online Viterbi Algorithm

The Viterbi algorithm is usually intended to be used once on a complete set of data, when all observations are available. In the case of an executive as here, observations are being generated constantly resulting in a dynamically increasing set of data. Because the Viterbi algorithm needs to be applied on a set of data that is continually growing, there are optimisations that can be made to significantly reduce the amount of computation required to run it. The Viterbi algorithm requires the calculation of the most probable path through what is sometimes referred to as a *Viterbi trellis*. Each observation is represented by the addition of an additional layer to the trellis. Rather than entirely recompute the probabilities in the trellis with every new observation, the trellis can be stored and then reused next time the algorithm is called. When a new observation is made, only a single layer is added to the trellis and calculation of the Viterbi algorithm proceeds as normal (see Figure 6.3). Essentially, the Viterbi algorithm is simply paused between the addition of layers, and the final step of tracing the most probable transitions is performed at every layer. This variation on the standard Viterbi algorithm has been termed the *online Viterbi algorithm*, in reference to the fact that it works in real-time with incomplete data sets that are extended once they are known.

The online Viterbi algorithm works because the probabilities of the intermediate states of the trellis are unaffected by the addition of new data points at the end; all that is affected is the path of highest probability, which can be calculated in the usual manner by tracing back through the trellis. As long as the best-transition path is retained between observations, this functions identically to the standard Viterbi algorithm without the overhead of recalculation each time.

6.2 Failure Detection

To detect failure from a list of states, the concept of what is a normal execution looks like needs to be learnt. Fortunately, this is already available in the form of the data that was collected to create the HMMs in Chapter 4. These include every successful execution of the task, and from this can be extracted the information of what is normal and what is abnormal during execution. There are three methods that were developed to identify if an HMM state sequence is abnormal:

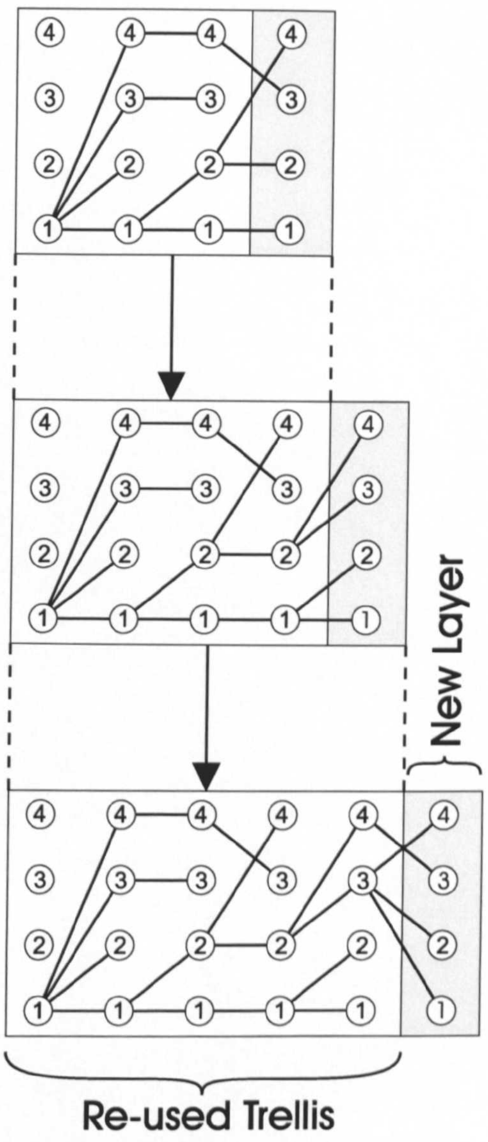


Figure 6.3: *How layers in the Viterbi trellis can be stored and extended later*

Probabilistic Anomaly Detection

The Viterbi algorithm functions by calculating the probability of observing state sequences, and then reports back with the most probable sequence that has occurred. One of the side effects of the Viterbi algorithm is this calculation of these probabilities, and these can be used to evaluate the likelihood that a sequence occurred[1]. By using the data from which the models were learnt, the general trends of how the probability behaves throughout the course of the action can be recorded. A deviation from these general trends is indicative of failure. In Figure 6.4, two actual executions of the Panoramic Photo task are shown. The first is a normal execution, as taken from the training data, and the second is from test data where the robot's radio connection to the controlling computer was deliberately broken after around 17 seconds. It can clearly be seen that the probability from the failed execution deviates from the probability of normal execution at around 35 seconds after the start. Note that the probability values simply represent the *amount of certainty* that the model has the correct sequence. A higher probability does not mean less or greater likelihood of failure. Because methods of detecting deviations such as this are not straightforward, these will be investigated and evaluated in Chapter 7.

Temporal Anomaly Detection

This method detects if the correct number of occurrences of a particular state have been observed, done by counting the number of occurrences of each state at each timepoint, and comparing this to the number of occurrences in the training data. If there are more or fewer incidences of states than in the training data (for the particular timepoint), then this should be interpreted as a failure. For example, if it is known that in all successful executions $\overline{9}$ occurred at least twice by timepoint 10, then any execution that has not seen at least two of $\overline{9}$ at timepoint 10 is indicative of failure. Similarly, if there are at most eight occurrences of $\overline{9}$ at timepoint 10, then anything exceeding eight occurrences is also indicative of failure. By taking a maximum and minimum value of occurrences for each state at each timepoint, upper and lower bounds can be established. These bounds can be represented as a graph, as shown in Figure 6.5. Each state in the action has a similar graph with the range of normal possible statecounts. During execution, the number of occurrences of each state will increase, and must stay in the area between the maximum and minimum statecounts. If the execution wanders into one of the shaded failure areas then the abnormal number of occurrences of

[1]The probabilities calculated are very low because they are the conditional probability that a sequence of states occurred given a specific sequence of observations. For example, with 100 observations, the most probable sequence of states is likely to have a probability around the order of 10^{-100}. This is the probability that a *particular* sequence of 100 states occurred, and hence is very improbable even though it is the most probable sequence.

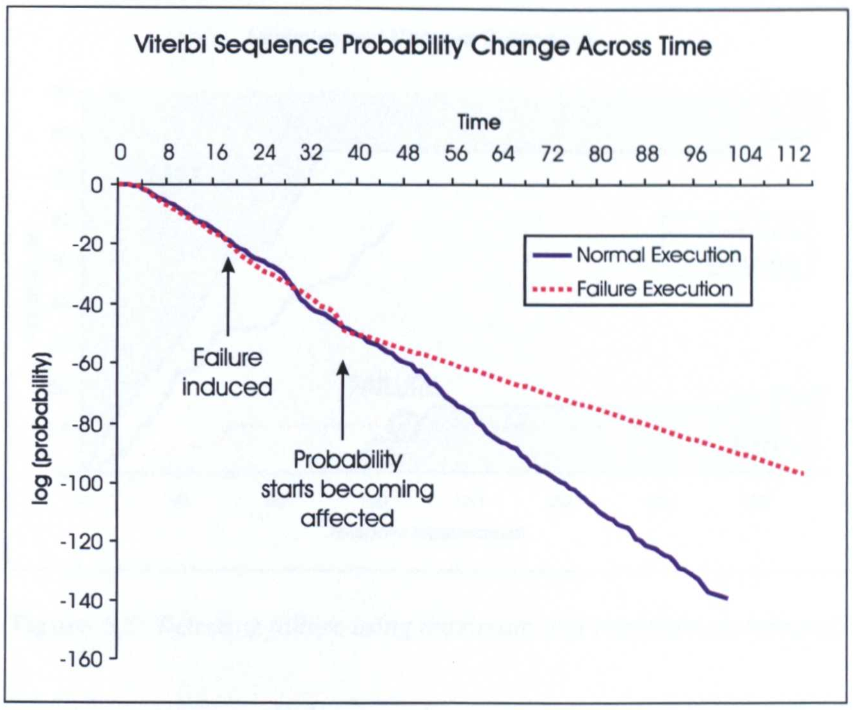


Figure 6.4: *Detecting failure using Viterbi sequence probabilities.*

that particular state indicates that something may have gone wrong. The graph shows two example executions of actions and the paths they take through this graph. Execution A successfully completes without entering the shaded areas. Note that this execution finishes quite quickly in roughly 130 steps, far short of the number of steps that the graph is valid for: it is not necessary for an action to take any particular number of steps as long as the statecounts do not leave the area bounded by the maximum and minimum values. Execution B enters the shaded area at the bottom indicating too few occurrences of the particular state that the graph represents. Note that if the action were left to continue, it would not recover from this failure and it would be beneficial to terminate the action if it remains in this area for a certain amount of time. As stated above, this graph only represents one of the states; in an HMM with thirty states there would be twenty-nine similar graphs tracking the progress of the execution. If *any* of the thirty graphs encounters an abnormal statecount then this is indicative of failure.

State Tardiness Detection (non-temporal)

This works by recording the latest point at which a particular HMM state has been seen. For example, if it is known that $\overline{9}$ in the HMM was never seen in the training data after timepoint 50, then any occurrence of $\overline{9}$ after this timepoint should be construed as a failure. State Tardiness Detection was conceived to detect errors

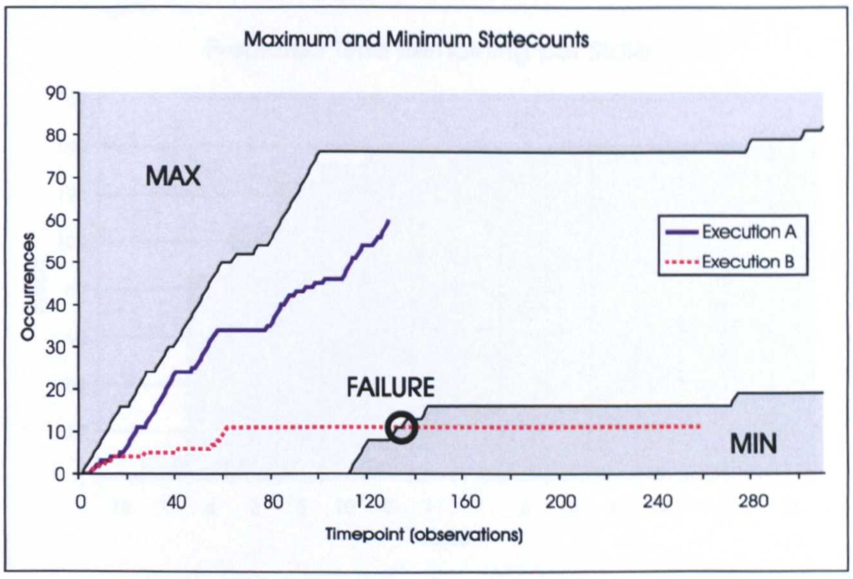


Figure 6.5: *Detecting failure using maximum and minimum statecounts.*

in the cases where a particular state has to occur at some timepoint through the action, but not at the end. If that particular state occurs too late in the action then it is indicative that something has gone wrong.

6.2.1 Bootstrapping

A technique of bootstrapping from the learnt model was tried, albeit with limited success. The aim of this was to produce a prediction of how long the action has remaining given the current state. For example, if the Viterbi algorithm reports that the executive was in State χ, then the prediction could say that there were roughly τ more observations until the action could be expected to finish. This information would be very useful for an executive because it would allow it to determine if the time remaining for the action was greater than the time available. If this were the case, then the task would not be able to complete in time, and the task could be reported as having failed early.

To form the predictions, executions of the HMM were simulated by taking transitions between states as indicated by the probabilities in the model. The average number of timepoints (number of steps) to reach the end state starting from each state was recorded. This value can be taken as an average distance to the end of the action, and can be used during execution to predict how long is remaining until completion.

For this representation to be useful in terms of execution monitoring, each state should have a distinctive time remaining. There should be some states that have high average

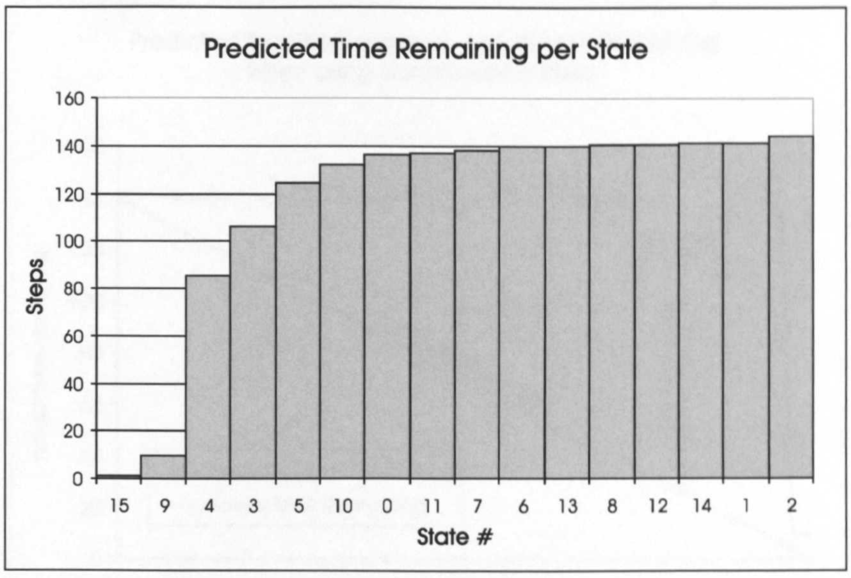


Figure 6.6: *Predicted number of timepoints (HMM steps) remaining per state, sorted by time remaining*

times to completion, and also some that have short times. When the above bootstrapping technique was tried, the states did not have highly discriminatory times. The results for one typical HMM can be seen in Figure 6.6 (the task that produced this HMM was a Panoramic Photo with Errors task). All but four of the states predict completion times within the region of 120–145 steps. These other states predict times that rapidly decrease to zero with no states representing anything from 10–85 steps to completion of the task. It is interesting to note that even though the action is one with deliberate errors inserted, there are no states with higher times to completion, which could be indicative of error states. Other HMMs produce very similar diagrams, and this characteristic distribution of times to completion is seen across all HMMs that were tried. Also, the failure states that were manually identified earlier did not have unusually high durations to completion, and so this technique could not be used to automatically identify these.

To illustrate the lack of predictive power of this method, a typical execution using real data from the Panoramic Photo with Errors task was taken and converted into a state sequence for the above HMM. By plotting a graph of time remaining to completion versus the predicted time for each state in sequence, Figure 6.7 is produced. If the predicted time remaining accurately represented the actual time remaining then it should decrease at a constant rate, as indicated by the line in the figure. Unfortunately this does not happen and the predicted time remaining using this method proves to be a very poor estimate of the actual time remaining. It constantly over-estimates the time remaining and is wildly inaccurate with high errors. Towards the end of the action

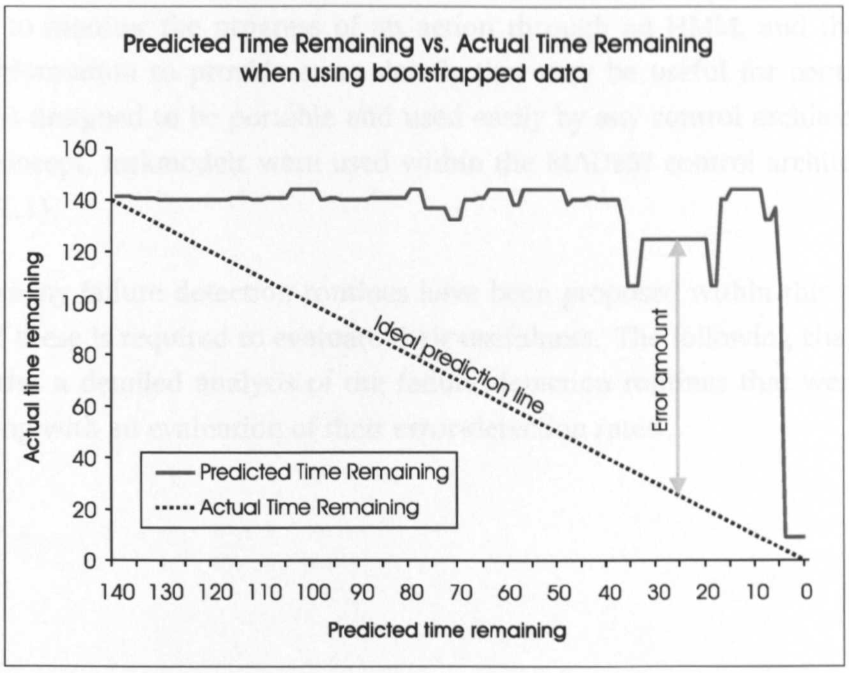


Figure 6.7: *Using the bootstrapped data to predict time remaining for an real execution*

there are a few points at which the prediction decreases, but it generally remains high until the very end.

It is probably the case that the loops and cycles in the HMMs caused any distinctions between states to be watered down. Indeed, the HMMs are learnt as non-temporal structures where states are not assigned times. Instead, they are probability based and have no memory of the number of times a particular state has been encountered. Loops will always have a fixed exiting probability within the HMM, rather than altering based on previous observations. For future work, there is the possibility of learning a second higher-order HMM, with the new observations being represented by the state of the original HMM paired with time. If such a model were learnt, it would lose some of the generality as it would now be tied to actions of a specific duration. It seems that there may be a choice here between having a generic model with low predictive power, or having a highly specialised model with high predictive power.

6.2.2 Taskmodels

In Chapter 4, details of how to learn an HMM for a task were discussed, and this has been expanded on by providing methods for detecting failure by using the HMM. If all of this information is collated then the result is a reusable model of an action that can be used for failure prediction. For this purpose, a type of file called a *taskmodel* was developed. A taskmodel contains all of the information required for an executive controller to monitor the progress of an action through an HMM, and then how to use this information to provide extra details that may be useful for controlling the action. It is designed to be portable and used easily by any control architecture. As a proof of concept, taskmodels were used within the MAD$\overline{\text{BOT}}$ control architecture (see Section 8.1.1).

Although many failure detection routines have been proposed within this chapter, an analysis of these is required to evaluate their usefulness. The following chapter therefore provides a detailed analysis of the failure detection routines that were outlined above, along with an evaluation of their error-detection rates.

CHAPTER 7

Evaluation 2

Malum consilium quod mutari non potest
(It's a bad plan that can't be changed)

— Publilius Syrus

EVALUATION OF THE error detection methods discussed in the previous chapter is essential for discovering their practicality. This chapter evaluates these techniques, alongside measurements of their ability to detect or predict errors. The Panoramic Photo task was chosen for this evaluation as it produced the HMMs with the clearest structures out of the learnt models. This ensures that successful detection methods are not masked by poor task models. To evaluate these methods of error detection, three sets of data were collected:

- Training (50 executions)
- Verification (20 executions)
- Error (50 executions, split into five 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 times where a specific type of error was induced during the execution. 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 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. Simply put, 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 believes that it is still rotating, however there may be an extreme number of localisation steps as it tries to correct the errors. Eventually the robot's localisation cannot keep up with the errors and becomes wildly inaccurate.

Unknown Environment
This is perhaps the hardest type of error to detect, as it was induced by placing the robot in an unknown environment; one which the internal map did not match, causing localisation attempts to produce potentially incorrect and inconsistent results. The robot was still able to complete the task, but the angles through which it turned were incorrect.

Throughout this chapter, analysis was done by using the size 20 HMM for the Panoramic Photo task, as in Figure 5.5. This was selected for its relative simplicity whilst still retaining some structures of the more complex aspects of the task. With larger models, the average number of times a state is visited is also lower (there are more states to visit, but the same number of observations). This makes the maximum and minimum statecount values lower and therefore less distinctive.

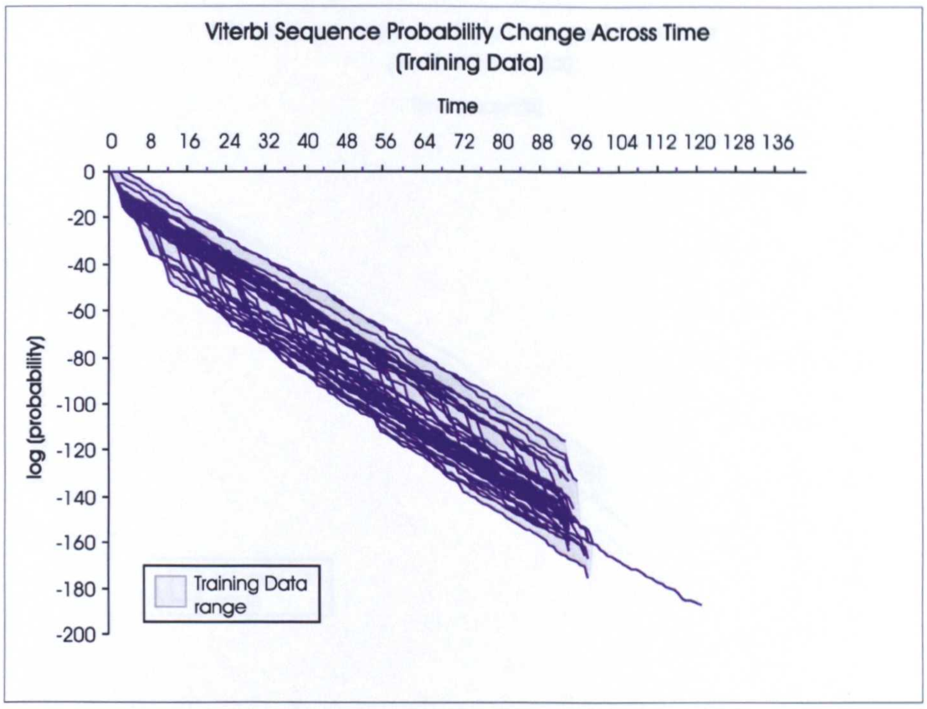


Figure 7.1: *The Viterbi sequence probabilities for the training data*

7.1 Probabilistic Anomaly Detection

As briefly mentioned in the previous chapter, irregular executions can be detected from identifying anomalies in the Viterbi sequence probabilities. Determining a strategy for identifying these anomalies requires examination of the executions to look for patterns and features that could be used to indicate failure. Figure 7.1 shows the probabilities obtained from the fifty Viterbi sequences used to learn the Panoramic Photo task. Time is plotted horizontally, and log(probability) vertically. Logarithmic values are used due to the fact that the probabilities are multiplied at each timepoint, and therefore decrease exponentially. In this figure, the probabilities of the training data decrease at a relatively uniform rate, and are contained within a range of about 40 magnitudes in size. 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 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.

The following section contains an analysis of the HMM sequence probabilities for each of the remaining data sets.

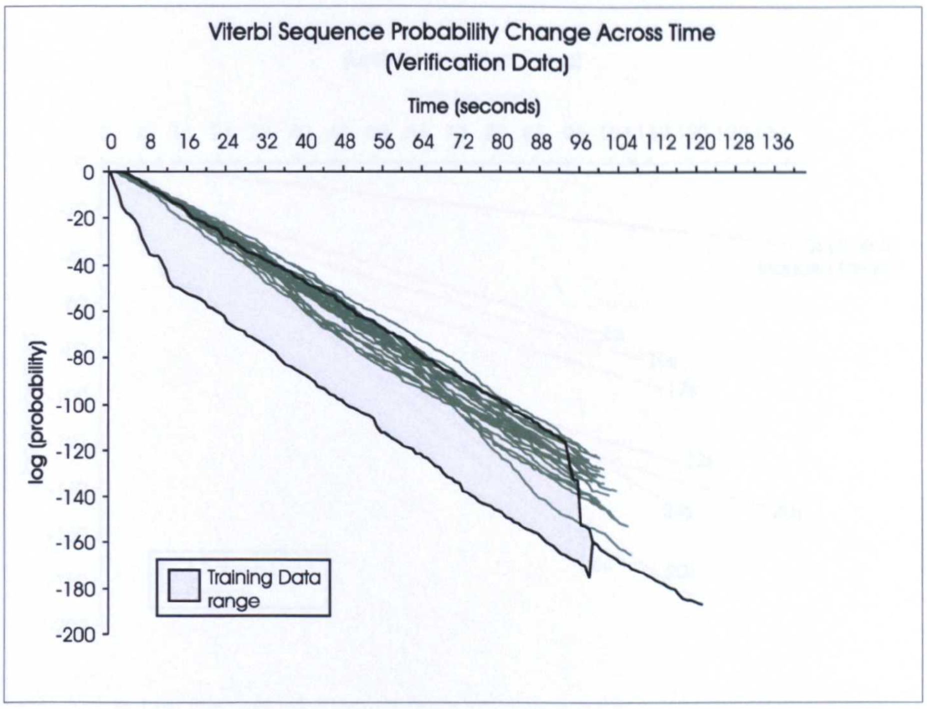


Figure 7.2: *The Viterbi sequence probabilities for the verification data*

7.1.1 Manual Examination of Viterbi Probabilities

Verification Data (Figure 7.2)

This graph shows the probabilities from twenty executions that were gathered for verifying the model. Almost all of the probability lines lie inside the shaded area that represents the range of probabilities observed from the training data. Most of the verification executions stay within or around the same area as the training executions, but there are several executions that deviate slightly. The difference is relatively minimal though, and never more than around 8 orders of magnitude. Note that all of the executions take longer than 49 out of the original 50 executions in the training data. Because of the amount of time it took to collect data, the training data and the verification data were collected on different occasions. In hindsight, it would have been better to have collected all of the data and then partitioned it randomly into training and verification data, but time pressures meant that this was not done.

Lost Connection Errors (Figure 7.3)

The cases where the connection was deliberately terminated can be seen in this figure. The upshot of terminating the connection is that the robot cannot report back new sensor data, and the computer uses the last known values instead. These values are repeated until the action is manually terminated. In the graph, the times at which

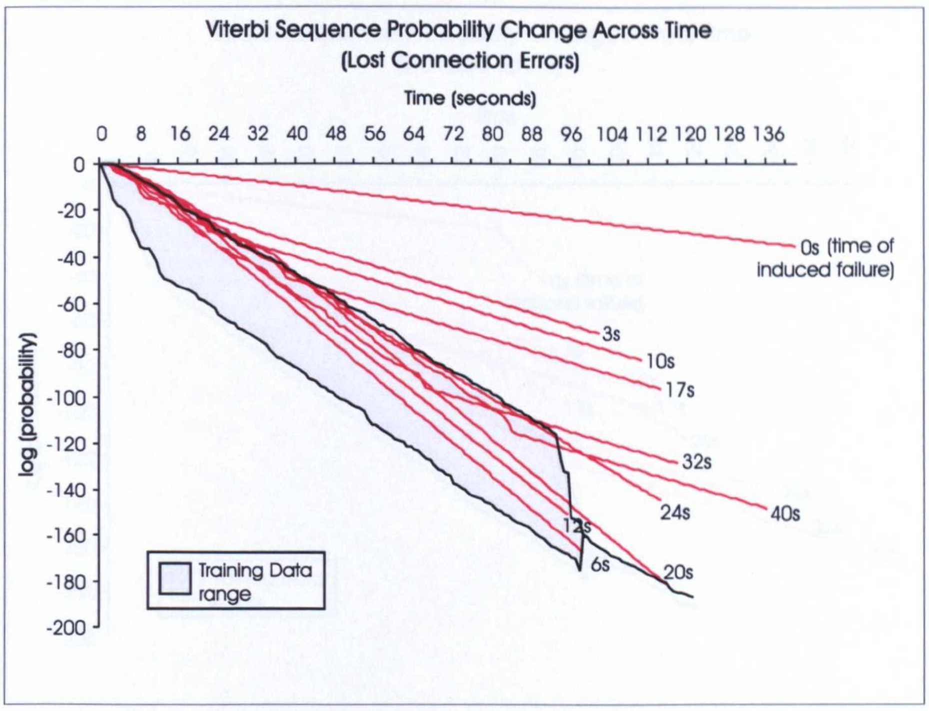


Figure 7.3: *The Viterbi sequence probabilities for the 'Lost Connection' error data*

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 right from the beginning. 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. For the size 20 model that was used (which can be seen in Figure 5.5), 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, it seems that the HMM is seeking out the best place to stay to maximise the probability of the sequence.

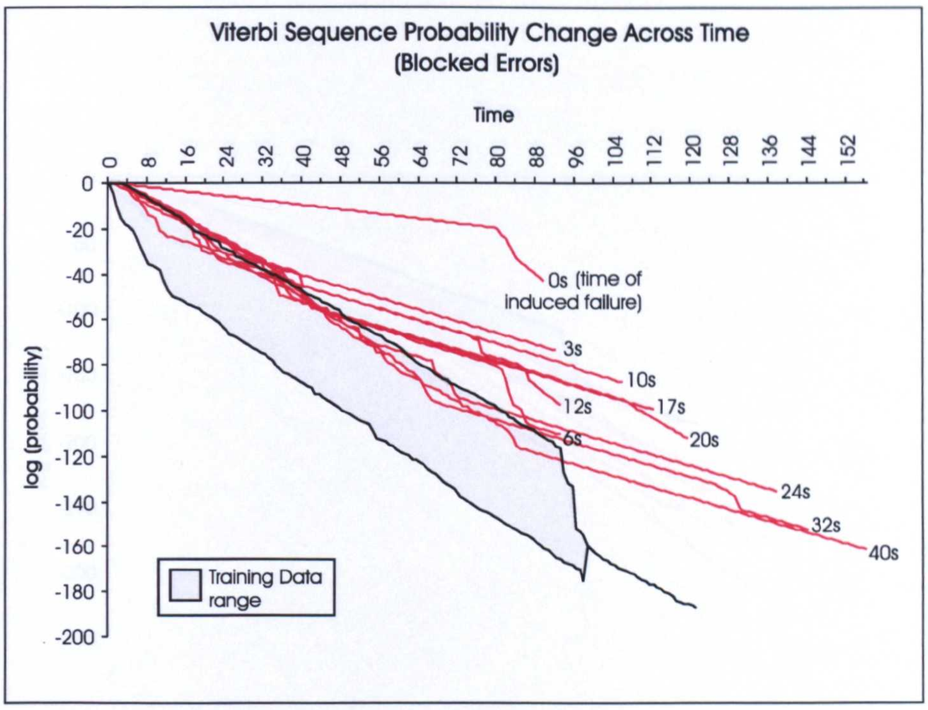


Figure 7.4: *The Viterbi sequence probabilities for the 'Blocked' error data*

Because the last received sensor data is repeated when the connection is terminated, the robot could report that it is continuously stuck in any one of the states through which it progresses. This means that some of the states may be easily detected, such as states that that rarely repeat in succession. Others states may be much harder to identify, for example those that repeat many times in succession during normal executions. The fact that only six of the ten sequences display obvious anomalies could be attributed to this fact.

Blocked Errors (Figure 7.4)

Like the error cases in which the connection was terminated, these executions have probabilities that deviate upwards from the training data. The difference here is that all of the sequences exhibit this behaviour, and not just the majority. One explanation for this could be down to the sensor values reported from the robot; as opposed to the case where the radio connection was severed, the data reported back here will always indicate that the robot is barely moving. The sensor values may change slightly as the robot tries to free itself, but it never manages to do so and angular change remains very close to zero.

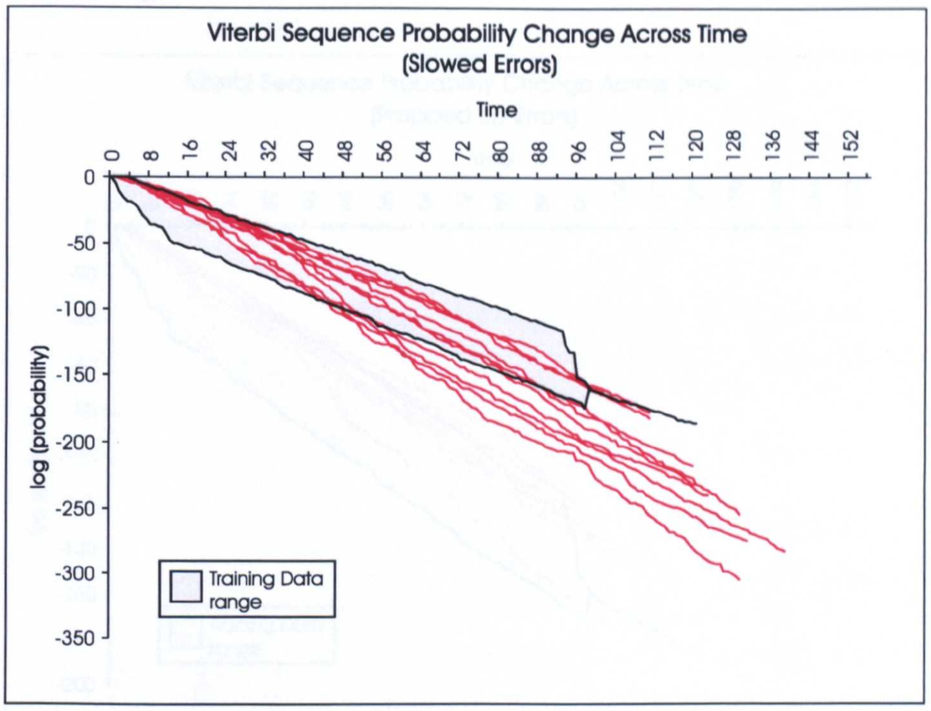


Figure 7.5: *The Viterbi sequence probabilities for the 'Slowed' error data*

Slowed Errors (Figure 7.5)

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-probability sequences could be due to the fact that the most appropriate HMM sequences in these cases loop on states that have low self-transition probabilities. Such a trajectory is highly unlikely to occur.

Propped Up Errors (Figure 7.6)

The data collected from these error executions does not show any apparent difference to the training or verification data. The probabilities reported back are within acceptable limits and it is impossible to distinguish these executions as possessing any anomalies.

Unknown Environment Errors (Figure 7.7)

Again, there is no apparent distinction between errors of this type and normal executions. All of the executions progressed with sensible probabilities that do not indicate the occurrence of any failure.

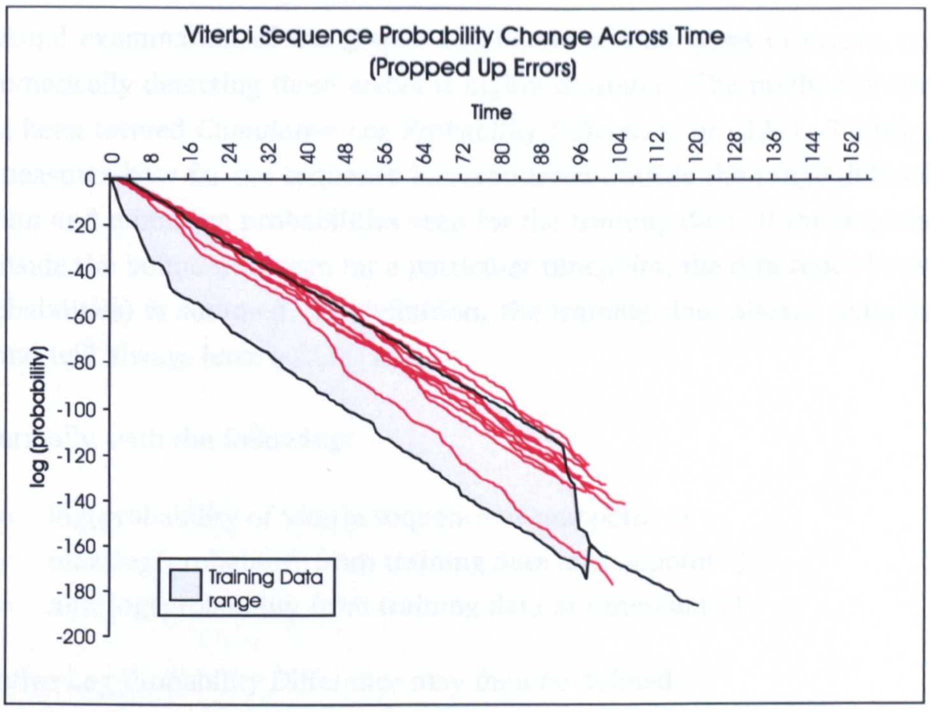


Figure 7.6: *The Viterbi sequence probabilities for the 'Propped Up' error data*

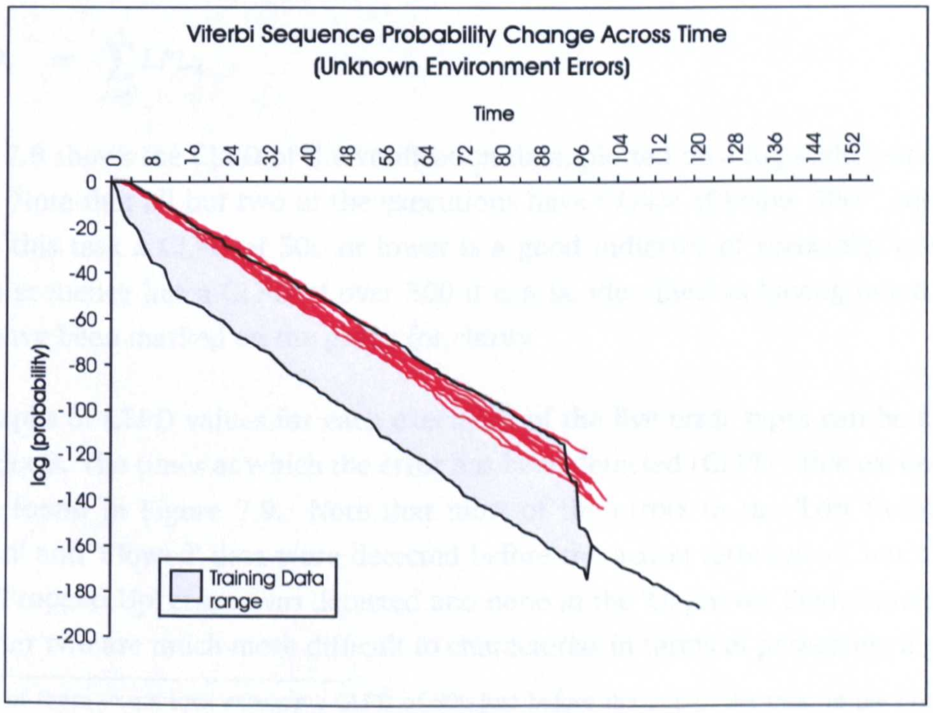


Figure 7.7: *The Viterbi sequence probabilities for the 'Unknown Environment' error data*

7.1.2 Automatic Error Detection

Since visual examination of the graphs can detect certain types of errors, a method for automatically detecting these errors is highly desirable. The method proposed below has been termed *Cumulative Log Probability Difference*, or CLPD. Simply put, 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:

$$\begin{aligned} p_x &= \text{log(probability of Viterbi sequence at timepoint } x) \\ m_x &= \text{max(log(probability from training data at timepoint } x)) \\ n_x &= \text{min(log(probability from training data at timepoint } x)) \end{aligned}$$

Cumulative Log Probability Difference may then be defined:

$$\begin{aligned} LPD_x &= \begin{array}{ll} 0 & [p_x < m_x \wedge p_x > n_x] \\ (p_x - n_x) & [p_x < n_x] \\ (m_x - p_x) & [p_x > m_x] \end{array} \\ CLPD_x &= \sum_{i=0}^{x} LPD_i \end{aligned}$$

Figure 7.8 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 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. These areas have been marked on the graph for clarity.

The graphs of CLPD values for each execution of the five error types can be found in Appendix B. The times at which the error has been detected (CLPD value exceeds 500) can be found in Figure 7.9. 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 and none in the 'Unknown Environment' data. The latter two are much more difficult to characterise in terms of possessing a physical

[1]One of these executions exceeds a CLPD of 500 just before the end of the task, at the point where there was only one training execution of this length. This means there was little evidence for the possible spread of values at this timepoint, and causes the CLPD to rise sharply after this point. As mentioned earlier, selecting the training and verification data randomly from the set of all normal executions would most likely have eliminated this problem, leaving only one normal execution with a CLPD of over 500.

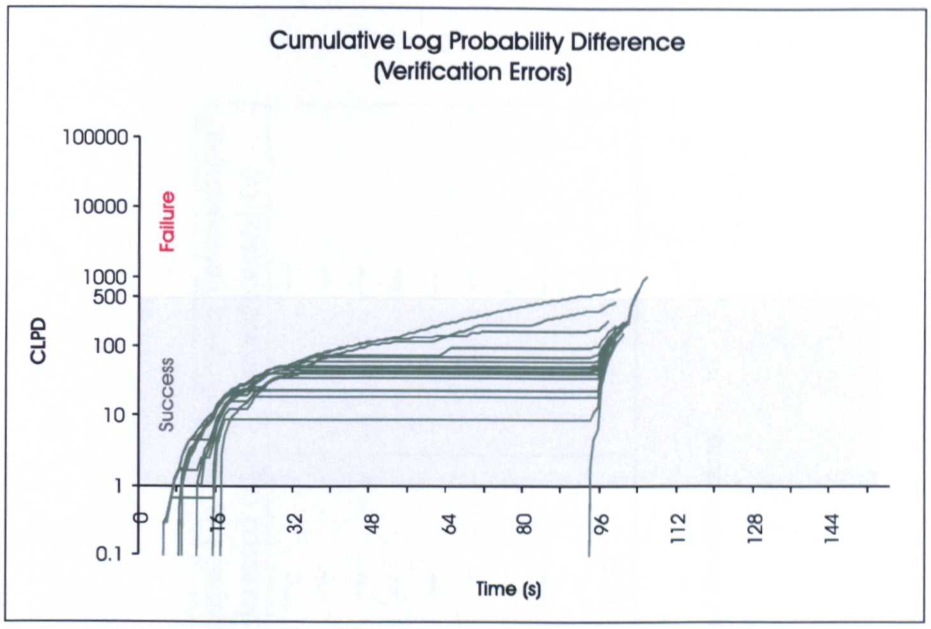


Figure 7.8: *The CLPD values for the verification data*

behaviour that is different to a normal execution. Indeed, a human presented with the raw data of one of these executions and a normal execution would most likely not be able to distinguish between the two. It had been hoped that there would be a difference in the amount of localisation required for these two error types, 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 may have been detected.

7.2 Temporal Anomaly Detection

Temporal Anomaly Detection identifies Viterbi sequences with an anomalous number of occurrences of states (either too few or too many). 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:

$$\begin{aligned}
t &= \text{number of states in HMM} \\
c(s,x) &= \text{number of occurrences of state } s \text{ up to timepoint } x \text{ in current execution} \\
m(s,x) &= \text{max(\# of occurrences of state } s \text{ up to timepoint } x \text{ from training data)} \\
n(s,x) &= \text{min(\# of occurrences of state } s \text{ up to timepoint } x \text{ from training data)}
\end{aligned}$$

Time of induced error (s)	'Lost Connection' error detected (s)	'Blocked' error detected (s)	'Slowed' error detected (s)	'Propped Up' error detected (s)	'Unknown Environment' error detected (s)
0	30.4	30.4	75.2	—	—
3	53.6	64.8	82.4	—	—
6	—	—	70.4	—	—
10	64.0	71.2	111.2	—	—
12	—	68.0	94.4	—	—
17	78.4	80.0	106.4	—	—
20	—	80.0	112.0	—	—
24	106.4	99.2	—	—	—
32	104.0	102.4	—	—	—
40	105.6	106.4	—	96.8	—

Figure 7.9: *Probabilistic Anomaly Detection performance across the error data*

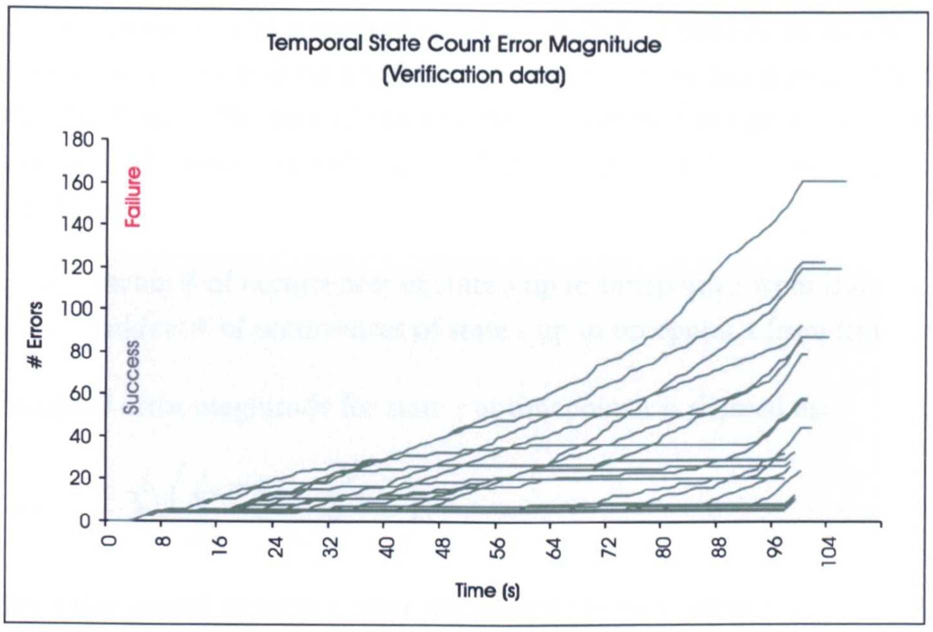


Figure 7.10: *The TSCEM values for the verification data*

The error magnitude for state s at timepoint x is defined as how far the current execution deviates from the training data:

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

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

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

Figure 7.10 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, it shall be taken that a TSCEM value anything over 125 as an indication of failure in this case for this task.

The TSCEM values for each of the error executions can be found in Appendix C. 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’ and ‘Unknown Environment’ errors.

One possible extension to this method of error detection would be to take the Normal 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 statecount varies (z-score) could be taken instead of the TSCEM value. This would be defined as follows:

$$\begin{aligned}\mu(s,x) &= \text{mean(\# of occurrences of state } s \text{ up to timepoint } x \text{ from training data)}\\ \sigma(s,x) &= \text{stddev(\# of occurrences of state } s \text{ up to timepoint } x \text{ from training data)}\end{aligned}$$

The normalised error magnitude for state s at timepoint x is defined as:

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

Using this value would provide a finer distinction between what does and what does not constitute an error. The disadvantage is that calculation times would be slightly greater and more training data would be required to get accurate Normal distributions for each state at each timepoint. Since a great deal of data would be required to analyse this, such a method would be an ideal subject for further research.

7.3 State Tardiness Detection

State Tardiness Detection is the simplest method of error detection here. The table below shows the time in seconds at which errors were reported for the twenty verification executions:

92.0	94.4	93.6	94.4	92.0
92.8	93.6	92.0	92.0	94.4
91.2	92.8	94.4	93.6	94.4
92.0	91.2	91.2	93.6	92.0

Note that *all* of the verification executions are being reported as having errors at around 91–95 seconds into execution i.e. before the task has completed. In a perfect error-detection routine, the verification data should return no errors at all. This means that all of these reported errors are false positives, nullifying any of the predictive power of the algorithm. For completeness, the error detection times for all of the error executions can be found in Figure 7.12. In these results, most errors are detected at around 91–95 seconds with a few being reported at 5.6 seconds into execution.

Time of induced error (s)	‘Lost Connection’ error detected (s)	‘Blocked’ error detected (s)	‘Slowed’ error detected (s)	‘Propped Up’ error detected (s)	‘Unknown Environment’ error detected (s)
0	23.2	23.2	37.6	64.0	—
3	28.0	27.2	50.4	—	—
6	36.0	36.8	47.2	96.8	—
10	40.0	43.2	66.4	—	—
12	48.0	48.8	75.2	—	—
17	57.6	59.2	78.4	—	—
20	62.4	63.2	84.0	—	—
24	74.4	78.4	96.0	—	—
32	51.2	92.0	100.0	—	—
40	—	—	93.6	—	—

Figure 7.11: *Temporal Anomaly Detection performance across the error data*

Time of induced error (s)	'Lost Connection' error detected (s)	'Blocked' error detected (s)	'Slowed' error detected (s)	'Propped Up' error detected (s)	'Unknown Environment' error detected (s)
0	5.6	5.6	5.6	94.4	93.6
3	93.6	—	91.2	94.4	94.4
6	93.6	—	94.4	91.2	93.6
10	93.6	93.6	91.2	93.6	92.8
12	93.6	—	93.6	92.0	94.4
17	93.6	93.6	92.0	92.8	92.8
20	93.6	93.6	94.4	92.8	93.6
24	93.6	93.6	91.2	92.8	92.0
32	93.6	93.6	91.2	94.4	93.6
40	93.6	93.6	91.2	93.6	91.2

Figure 7.12: *State Tardiness Detection performance across the error data*

7.4 Error Detection Evaluation

The data from above is repeated in Appendix D, but grouped by error type instead of detection method. This allows a direct comparison of the Probabilistic Anomaly Detection to the Temporal Anomaly Detection method. The State Tardiness method has not been included here because of the false positives that it produced. From the tables in the Appendix, it can clearly be seen that Temporal Anomaly Detection detects the errors earlier, and detects more of the errors. Temporal Anomaly Detection successfully identified 30/50 errors, while Probabilistic Anomaly Detection identified only 24/50.

CHAPTER 8

Reacting to Uncertainty and Failure

It is far better to foresee even without certainty than not to foresee at all

— *Henri Poincaré*

ONCE FAILURE has been detected, an appropriate response must follow to ensure that the negative impact of failure is minimised. In previous chapters, the issues of introspection and error detection were covered. To complete the system and allow full execution, a high-level controlling mechanism is required. Such a mechanism allows the system to establish how to best react to problems once they have occurred.

8.1 Linkage to High-Level States

So far, the concept of a state has referred specifically to low-level, internal states that may not be directly observable. Examples include the HMM states that were learnt in Chapter 5, and analysed in Chapter 7. In a real-world system there needs to be the capability to deal with high-level physical states, such as a plan generated by an

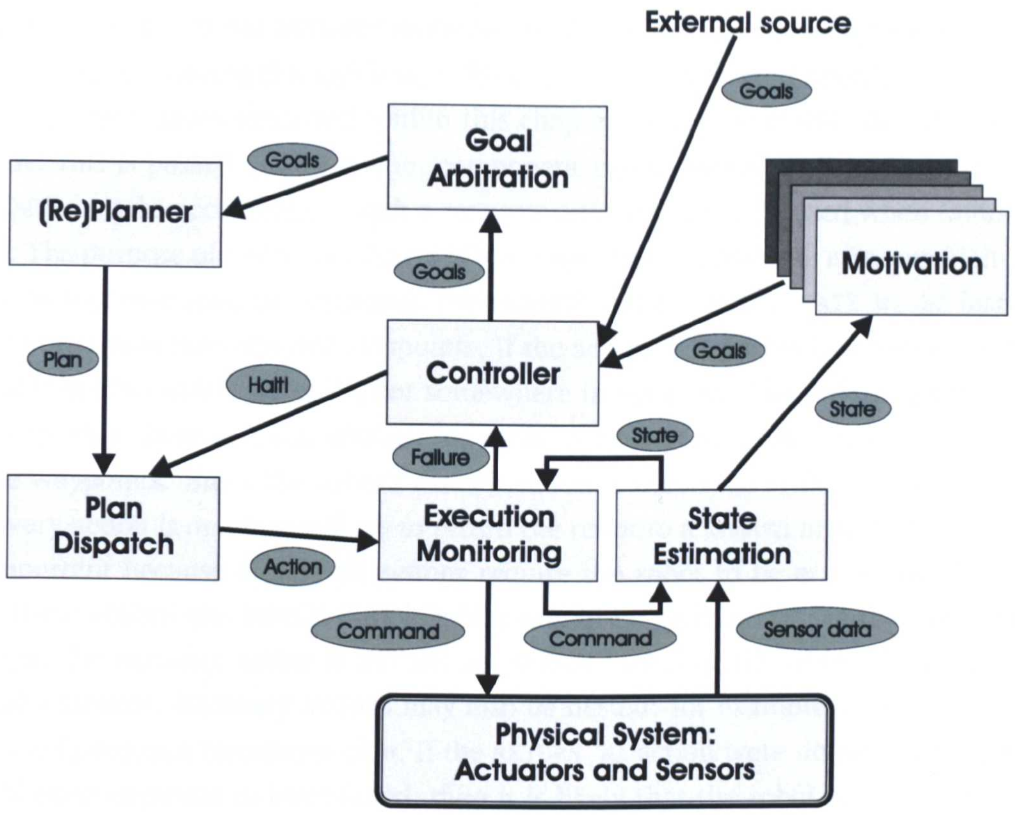


Figure 8.1: *The MADBOT Execution Architecture*

automated planner. The gap between low-level and high-level states means that these need to be linked together to allow useful execution. Although the research topic of linking these levels of reasoning is vast, it is not the subject of this thesis. Regardless of this, the failure analysis methods (as discussed within Chapter 7) were integrated into a robotic architecture called MADBOT.

8.1.1 MADBOT

MADBOT (**M**otivated **A**nd goal-**D**irected ro**BOT**) is a robotic plan execution architecture that is designed to use changing motivations to generate new goals for an executive [Coddington et al., 2005]. It was developed to investigate how changing goals can be integrated into a plan. Figure 8.1 shows an outline of the various system components and how these interact with each other. The architecture uses the automated planner LPG [Gerevini and Serina, 2002] for the generation of plans, and builds on the work on motivations in [Coddington and Luck, 2004]. Motivations allow the executive to be controlled in a much simpler manner than traditional architectures where each goal is explicitly provided to the executive. The executive is allowed to exhibit a much greater autonomy so that it can decide on its own goals during execution.

At the lower end of the MADBOT architecture, there is scope for monitoring and control. Action monitoring through learnt HMMs has been integrated into the architecture, using the techniques described within this chapter. When an HMM sequence reports failure, this is passed onto a system component called the *Controller*. Each action in MADBOT may be accompanied with a *recovery action* that can be used when failure occurs. The purpose of a recovery action is to return the executive into a known high-level state so that execution can continue. For example, an action `NAVIGATE W1 W2` instructs a robot to travel between two waypoints. If the action fails part way through, then the robot is neither at `W1` nor at `W2`, but somewhere in between. The failure of this action suggests that there is some obstacle or other reason preventing navigation between these waypoints. Since the robot's exact location is unknown at this point, a suitable recovery action is one that will try to return the robot to a known high-level state. This is important because almost all actions require the robot to be at a known location, and these actions are invalid if its location cannot be guaranteed. In the case of navigation, the recovery action is `RETURN W1`, which instructs the robot to return to the initial waypoint. Recovery actions may also be nested: for example, a recovery action can itself possess a recovery action. If the `RETURN W1` action were detected through the HMM state sequence to have failed, then it is likely that the robot has become stuck, and unable to reach any waypoints. Since further navigation is unlikely to address the problem, the recovery action for `RETURN W1` is simply `SAFEMODE`. This instructs the robot to terminate execution immediately and then proceed into a safe state to await further instructions from the planner or human operator. Such a response is a suitable recovery action as the robot has been unable to reach the waypoints instructed, and has probably become stuck.

In MADBOT, the HMMs were used to detect failure. There is, however, another execution issue that may be addressed: early success. In the case of a conservative plan, a task may finish early, leaving the executive with extra time or resources available. The evidence based on the models learnt so far suggests that the early detection of success would be much harder than early detection of failure. However, success is easy to detect in nominal execution, and can therefore be hardcoded into the task. Failure is a much more "fuzzy" concept — there are many possible ways in which a task may fail, but only one way in which it may succeed.

This chapter proposes the use of opportunistic plans as a high-level execution control mechanism. This caters for the case in which an action has failed (after a recovery action), as well as the case of uncertainty due to early success. Note that the states and figures within this chapter refer to high-level plan states, and are separate from the low-level HMM states that have been used in previous chapters.

8.1.2 Resource Consumption Uncertainty

When uncertainty is present, the volatile nature of the tasks' resource consumption amounts means that normal timestamped plans are not a viable option. In the research area of planning, a plan is defined as a series of actions for an executive to carry out. This is typically a set of time-stamped actions such as the following:

```
0.0003:  (NAVIGATE BOT1 HOME PRINTER) [40.0000]
40.0005: (LOAD_ROVER PRINTOUT BOT1 PRINTER) [10.0000]
50.0008: (NAVIGATE BOT1 PRINTER DESK1) [30.0000]
```

This simple plan is for a robot (BOT1) to deliver a printout from PRINTER to DESK1. Resources are consumed during the execution of each step of the plan (for example battery power, CPU cycles, hard-disk space and ultimately time).

Since an executive will have limited amounts of each resource, if any of these runs out during execution then this can lead to plan failure, data loss, or in the worst case terminal failure where recovery is not possible. In some situations, the cost of recovery or risk to the executive is too great — this is especially the case where human interaction is impossible, such as unmanned interplanetary robots, or robots working in hazardous or extreme environments. In these situations, risks cannot afford to be taken and any action that might endanger the future of the mission will not be tolerated.

If resource consumption amounts were precisely known beforehand, the task of preventing plan failure due to resource over-allocation would be trivial. Unfortunately, the precise amounts of resources used by an action are not usually predictable and different situations will produce varying consumption levels. For example, the amount of data from a digital camera mounted on a robot will depend on the amount of data compression obtained, which in turn will be affected by the complexity of the subject in the photograph. The time taken to acquire a photograph may also depend on the time of day, i.e. the amount of light varies and hence the duration of the exposure also must change.

Because of the variations in amounts of resource consumption, each action can be modelled as having distributions of resource consumption, for example as illustrated in Figure 8.2. The distributions may be simple, Normal distributions as shown here, or they could be more complex types of distributions due to the nature of the task involved. A more complex distribution could emerge from a robot with the task of picking up a cup; it may require several attempts to succeed, and the distribution for

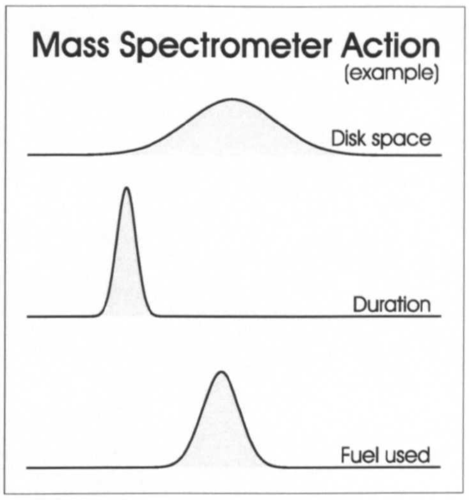


Figure 8.2: *An example of the distributions of consumption of different resources for an action*

this may be the combination of several Normal distributions according to the attempt number on which the action succeeded.

Some methods of responding to uncertainty were mentioned briefly in Chapter 3. In this chapter, instead of focussing on techniques that re-evaluate plans based on knowledge of the resources available, there will be an investigation into the technique called *opportunistic planning*.

Opportunistic planning [Fox and Long, 2002] is a technique that can be seen as an extension to conservative planning (see Section 3.4). It seeks to build on the strengths of conservative planning whilst countering its weakness of low utility plans. Many planning techniques that deal with uncertainty seek to produce a plan accomplishing as much as possible (with a high utility), and then modify this plan to add robustness, for example by adding contingencies. Opportunistic planning is a technique that approaches the problem from the opposite direction, generating a robust conservative plan (with a low utility), and then seeks to add optional plan fragments to this to gain a higher utility. They are usually best considered in environments where resource consumption uncertainty is present, for example where the duration of a particular action cannot be predicted with absolute certainty until it has completed. Other situations where opportunistic plans could be used include oversubscribed planning problems possessing uncertainty, such as the Mars rover domain described in [Joslin et al., 2005]. Although [Fox and Long, 2002] defines the concept of opportunistic plans, the issues of implementation are not addressed. This chapter examines these issues, moving towards a goal of execution of such plans. It does not address the issue of the creation of opportunistic plans, which should be the subject of further research.

8.2 Comparison of Opportunistic Plans to Other Techniques

During the execution of any type of plan, resources can be used to accomplish tasks. Effectively, resources can be traded for an increase in utility: a plan that uses little of the executive's resources will have a low utility; a plan consuming nearly all resources has the ability to achieve a higher utility.[1] The closer to the maximum available resources a plan lies, the more it can potentially achieve. If uncertainty leads to a plan consuming more resource than are available then it fails. There is effectively a boundary between success and failure at which lies the optimal usage of resources. Any small step over the boundary will result in plan failure due to lack of resources. Figure 8.3 is an illustration of how opportunistic planning compares to other plan representations that deal with uncertainty. In traditional planning (a), the planner tries the hardest to place the plan as close to the success / failure boundary as possible without exceeding it. In situations with little or no uncertainty this is the most efficient plan, but once uncertainty is present then this type of plan can lead to an unacceptably high probability of failure. Contingency planning (b) could be seen as a "scattergun method", placing plans that will end up at various points along the line. If resources allow, the choice to take a higher utility plan branch can be taken during execution. With conservative planning (c), only one plan is generated but it lies very far to the left of the boundary. This results in a very reliable plan at the expense of low utility. An opportunistic plan (d) uses the same conservative plan, but includes number of opportunity fragments that can be inserted to approach the boundary safely, with little risk of overstepping this.

To summarise:

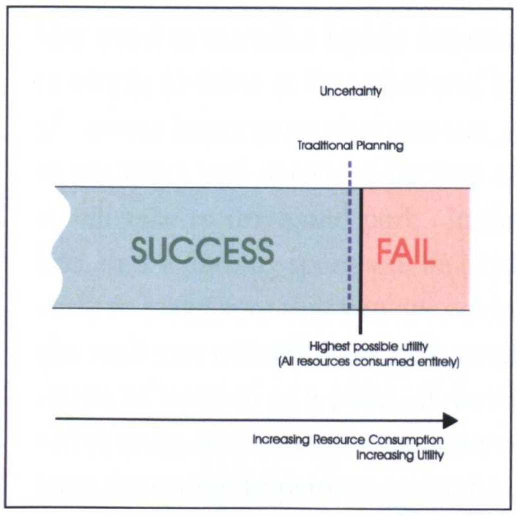
Traditional planning
High (highest) possible utility, high failure probability under resource consumption uncertainty

Contingency planning
Many different possible utilities, exponential plan size

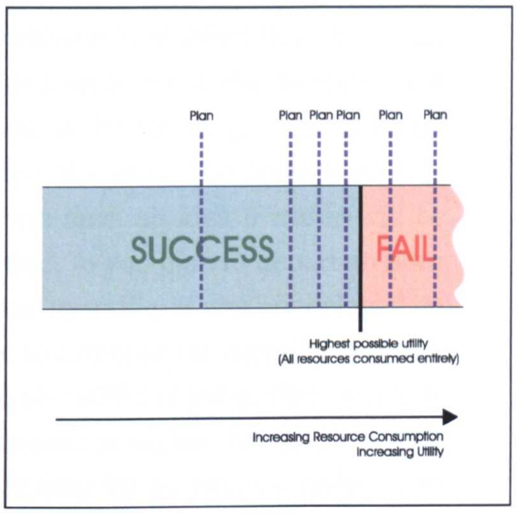
Conservative planning
Low utility, low failure probability under resource consumption uncertainty

Opportunistic planning
Approaches highest utility, low failure probability under resource consumption uncertainty

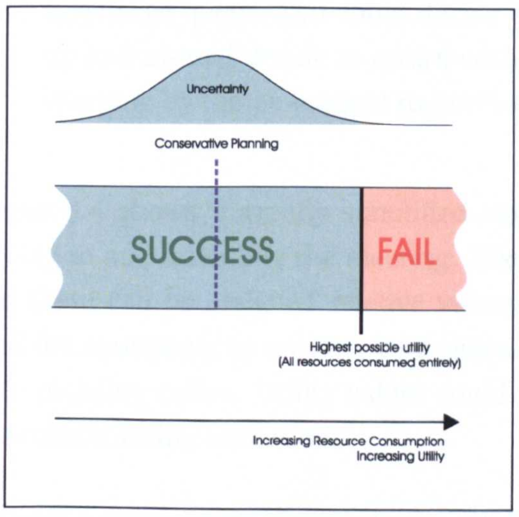
[1]This is assuming sensible plans that are efficient and do not unnecessarily waste resources.



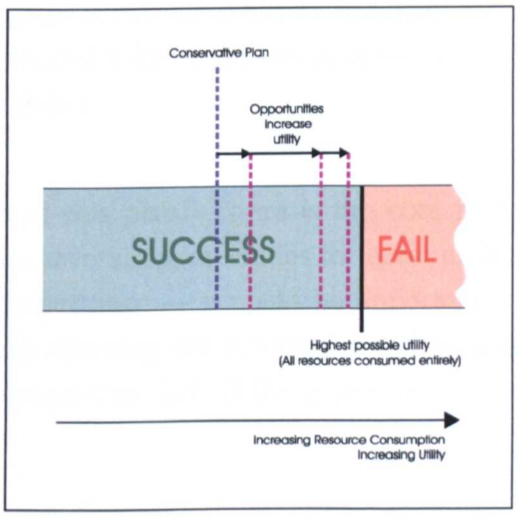
(a) Traditional plan



(b) Contingency plan



(c) Conservative plan



(d) Opportunistic plan

Figure 8.3: *Comparison of different plan representation techniques when uncertainty is present*

8.3 Opportunistic Plan Types

An opportunistic plan consists of a core plan and one or more opportunities, and is perhaps best illustrated by an example. One simple real-world example would be a plan to drive to an important business meeting, as detailed below:

> You need to attend a highly important meeting in another city. The plan is simply to drive to the other city, park and then attend the meeting. It is of utmost importance that you are present at the meeting, so in planning to get there you decide to be very conservative about the amount of time it will take to drive and park. Initially, you have no idea if traffic will be bad, or if a parking space will be hard to find, so you plan to depart an hour early to make sure that you are not late due to traffic. If you are delayed on the road you will still have time available to arrive at the meeting before it starts, at least up to a point. If, however, the traffic is good, then you may arrive early, and have up to an hour of free time available. At this point, you have the option to perhaps go to the coffee shop for an espresso rather than wait in the office for the meeting. On sitting down in the coffee shop, you could decide that there is enough time to purchase today's newspaper too. Another situation could be that you arrive at the coffee shop and discover a very large queue; you could decide that there is not enough time to queue up and instead decide to return early to the office for the meeting, possibly stopping by the newsagent to purchase a paper.

Figure 8.4 shows a greatly simplified version of this plan. There is the core plan: driving to and attending the meeting. There are several opportunities for extra utility, and these can be assigned relative values of importance — it could be important to read the newspaper to see how your shares are performing, but you don't mind missing your morning coffee. Utility values could be assigned to each of the opportunities for evaluation during execution.

There is a second type of opportunistic plan that does not have a core plan. In coreless opportunistic plans, the initial state is returned to after every opportunity. A coreless opportunistic plan does not seek to achieve any particular goal but the executive has free choice as to which opportunities to choose to maximise the utility of the plan given the resources available. This is similar to a simple oversubscribed planning problem where a subset of goals needs to be determined, but in this case the opportunities are chosen during execution rather than at the planning phase. This subtle difference allows the executive to react to uncertainty and make better use of resources. An

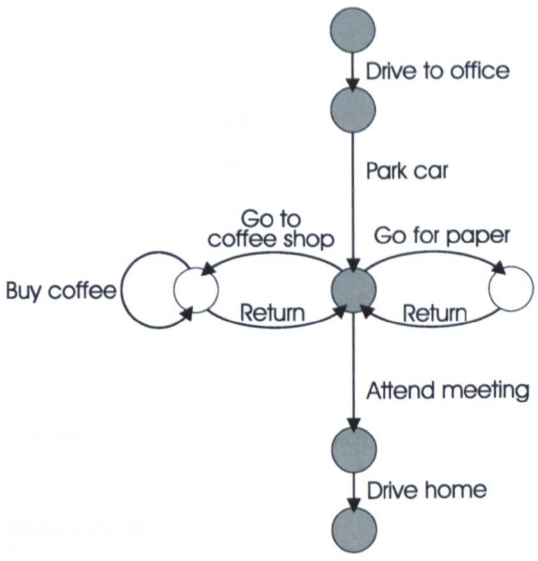


Figure 8.4: *The plan for attending a business meeting*

example of a type of plan that could be represented as above could be planning for a Sunday afternoon — imagine you have no set goals for the afternoon, but there may be some activities that you would like to do, such as the following:

- Go for a walk
- Read the newspaper
- Go to the cinema
- Paint a picture
- Make dinner
- Order a take-away
- Clean the house
- Write another section of your PhD thesis...
- etc.

It is not possible to do all of the tasks in the afternoon, so you need to choose which you would like to attempt. Using associated utilities for each of these you may to pick out which activities you would like to do for the afternoon to maximise the utility from these. Even more than this, it is possible for you to decide in which order the activities should be chosen in case one of them takes longer than initially planned, causing a reshuffle of later activities. A technique for selecting the order in which to execute opportunities during execution is detailed in Section 8.7.

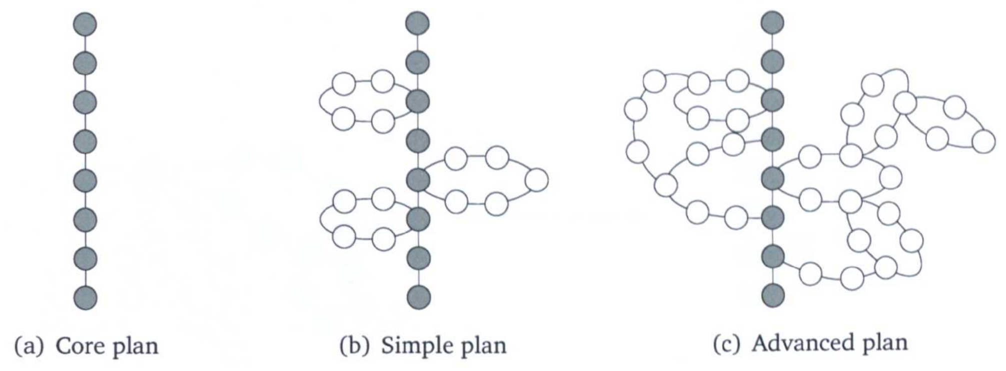

(a) Core plan (b) Simple plan (c) Advanced plan

Figure 8.5: *Examples of opportunistic plans*

8.4 Opportunistic Plan Structure

An opportunistic plan may be seen to consist of two components: a core plan and one or more opportunities. The core plan is an immutable plan consisting of a set of actions that will accomplish the desired goal. Opportunities are sub-plans that link into the plan and loop round to reconnect back to the core plan, forming *opportunity loops*. These must not affect the execution of the rest of the plan or its desired outcome, but can be used to perform additional tasks. Although the core plan cannot be changed, it can be temporarily suspended to make way for other actions in the form of taking an opportunity. Whether or not to take an opportunity would be evaluated during execution; if the executive decides that it has the necessary resources to complete the opportunity then it may be safely inserted. If the resources are not available then the opportunity is passed over and the core plan continues as normal.

Figure 8.5 shows three examples of opportunistic plans.[2] In case (a), an example of a simple linear plan is shown, and (b) introduces three optional opportunity loops that can be executed at run-time. A much more complex opportunistic plan is shown in (c) which has structures that allow many different routes of execution. Note that in all the above plans the core plan is the same.

The concept of a coreless opportunistic plan was introduced in an example above, but in fact these can be converted into opportunistic plans possessing a core plan; as mentioned above, it is possible to determine which opportunity to execute first (which will be explained in Section 8.7). Using this information, a core plan can be created out of the first opportunity and the remaining opportunities may be pushed down to follow on from this. The initial state is split into two, representing the start and end of

[2]Actions are represented in these diagrams (and all those following) by the lines between nodes, and high-level states of the system by the nodes themselves. All plans start at the top of each diagram and continue downwards through the actions. Note again that these states are high-level states and different to the low-level HMM states as discussed in earlier chapters.

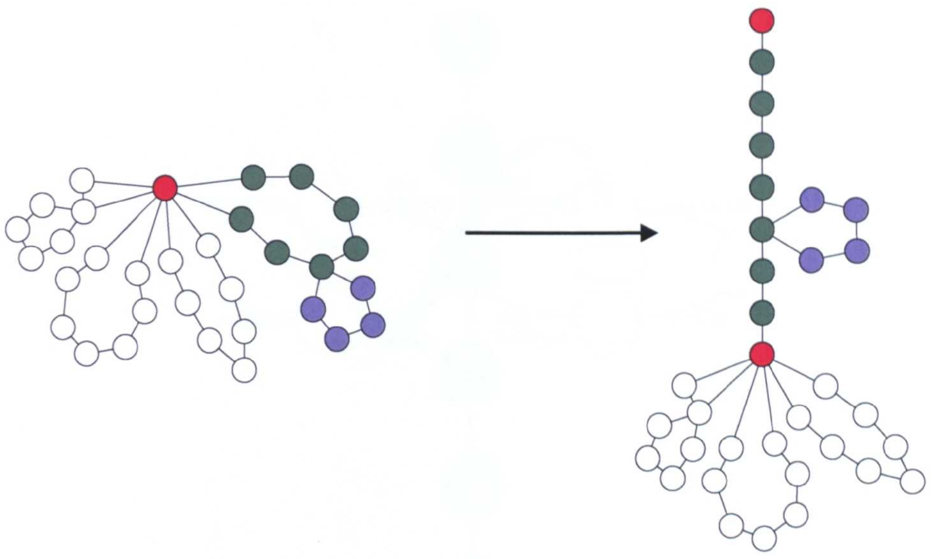


Figure 8.6: *Converting an opportunistic plan with no core into one possessing a core*

the first opportunity. The result is an opportunistic plan that functions in exactly the same manner as before except that the executive does not have to calculate the first opportunity to execute — this has already been done. Figure 8.6 shows an example of this process. The opportunity marked in green has been chosen as the first opportunity to execute. This is then brought to the front of all the other opportunities. Note that any opportunities that branch off the side of the first opportunity (marked in blue) remain unchanged. The advantage of converting a plan without a core into one with a core is that the computation can be done offline by the planner.

8.4.1 Advanced Variants of Opportunistic Planning

The work on opportunistic plans by in [Fox and Long, 2002] lays down a very simple definition of an opportunistic plan. This thesis proposes extensions to this definition, allowing various structures that can be used to improve efficiency and utility. One such structure is shown in Figure 8.7. This plan represents a simple Mars rover situation, and has two opportunities that represent soil analysis tasks. Before a soil analysis is performed, the instrument must be heated and afterwards must be cooled. However, if both opportunities are to be executed, then letting the instrument cool before reheating it would be a waste of resources. In this case, a step can be taken that bridges between the two opportunities, bypassing the unnecessary actions.

One way of loosely defining opportunistic plans is to see them as directed graphs in which every transition may only be used once (this restriction prohibits the creation of infinite loops in plans). If viewed in this way, all simple non-concurrent plans can be seen as directed graphs, including conservative and contingency plans as well as

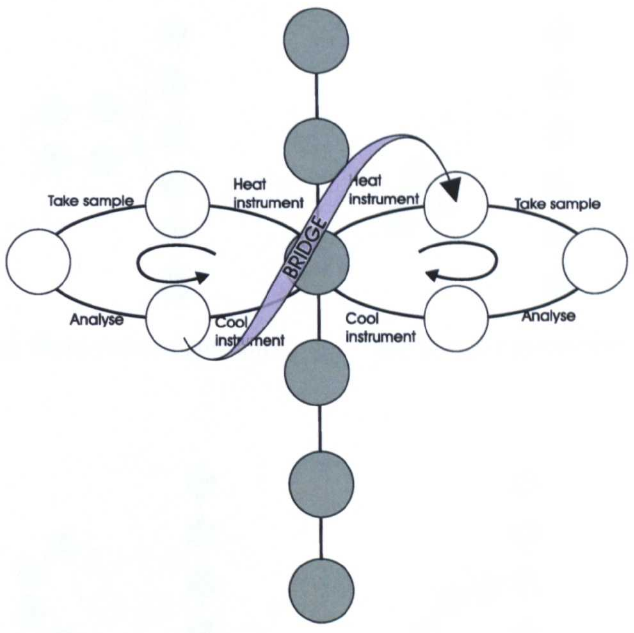


Figure 8.7: *Bridging opportunities to use resources more efficiently*

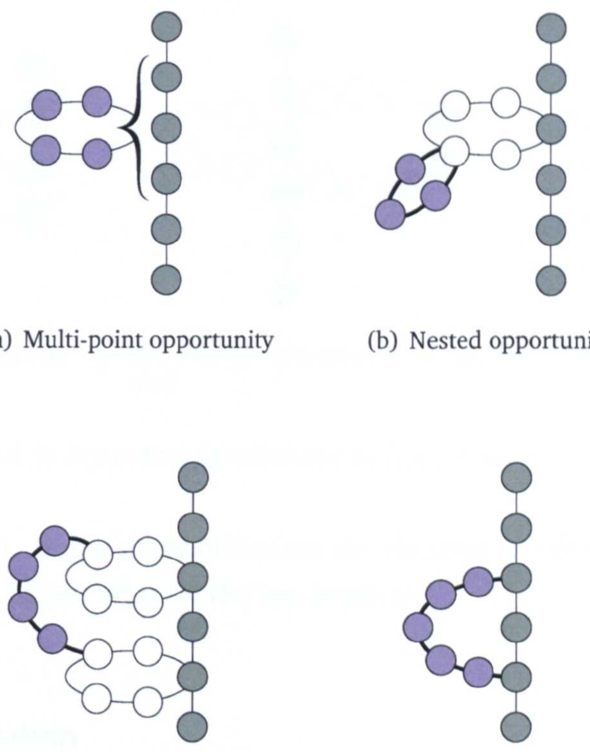
opportunistic plans. If plans are to be represented in this way then execution no longer becomes a simple task — it may be the case that certain choices within the plan are not possible. This will be discussed later in this chapter. With the directed-graph representation in mind, there are several advanced features that may be present in opportunistic plans, as shown in Figure 8.8. These have been defined as follows:

Multi-point opportunity
: This type of opportunity, as shown in case (a) allows an opportunity to be valid across a certain number of states. It is essentially a loop that can be inserted at any one of a choice of states in the plan.

Nested opportunities
: These opportunities that are attached to other opportunities, forming loops off loops, as in case (b). They can be used when certain opportunities have preconditions that are only satisfied by the actions in other opportunities.

Bridges
: Plan fragments that connect between two separate opportunities, joining them together are called bridges (c). These may allow certain actions to be omitted whilst jumping elsewhere in the plan.

Bypasses
: Bypasses are opportunities that do not return to the same location in the plan



(a) Multi-point opportunity (b) Nested opportunity

(c) Bridge (d) Bypass

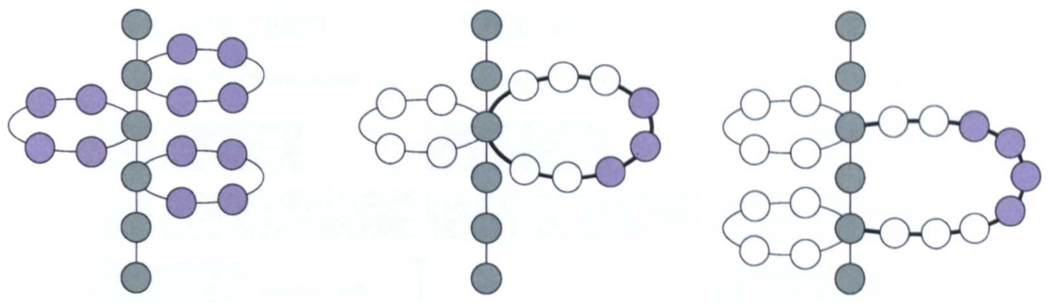
Figure 8.8: *Examples of advanced variants of opportunistic plans*

from which they started, as in (d).[3] A bypass could be used to skip certain parts of a plan to seek other goals with higher utilities.

Multi-point opportunities, nested opportunities and bridges can be converted into simple opportunistic plans by explicitly expanding every possible execution of the plan. Bypasses cannot be eliminated in this way. Figure 8.9 shows examples of the expansion of the above types of opportunities. In the case of multi-point opportunities, a record of the execution of an opportunity needs to be stored so that it cannot be executed twice.

Performing the expansions makes the plans potentially easier to deal with for an executive, although at the cost of loss of expressibility; decisions as to which opportunities to choose must be made at earlier timepoints than if the expansion had not been done. For example, in Figure 8.8(b) there is a choice of executing an opportunity after two actions, leading to a choice after five actions to execute the nested opportunity. In the expanded version in Figure 8.9(b), both those decisions must be made after two actions

[3]It could be argued that this term is deceptive as the plan fragment does not always reduce the length of the plan. It does however accomplish an opportunity before bypassing all of the states in between the start and end state of the opportunity



(a) Multi-point opportunity expansion (b) Nested opportunity expansion (c) Bridge expansion

Figure 8.9: *Expanding the advanced variants of opportunistic plans*

have been executed. In the expanded plans the decision to take a bridge or a nested opportunity cannot be put off until the last moment.

8.5 Opportunity Insertion

To decide on whether to take an opportunity or not, there are two issues that must be addressed. These allow the system to decide whether to insert an opportunity at a certain point in the plan, or whether it should be ignored:

Can the opportunity be inserted here?
Is it possible to insert the opportunity without invalidating the preconditions for the rest of the plan?

Should the opportunity be inserted here, or ignored?
How can the executive best use its resources? Should it take the opportunity now, or save up the resources in the hope of gaining a greater reward later on?

These two issues will be addressed in turn in the remainder of this chapter.

8.6 Can an Opportunity be Inserted?

To carry out an opportunity during plan execution, the executive must be able to determine if it is possible to insert the opportunity at this point. In a simple, STRIPS-type plan with an executive that does not allow concurrent actions, only two checks are required:

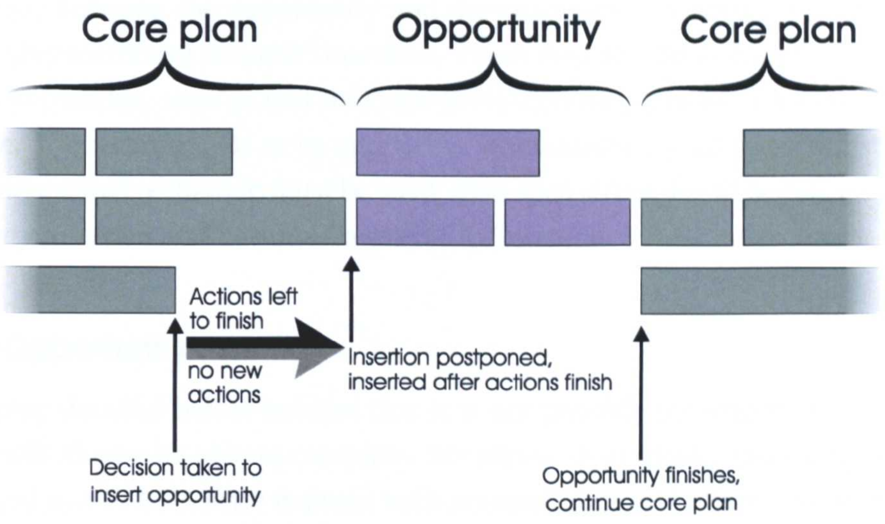

Core plan
Opportunity
Core plan
Actions left to finish
no new actions
Insertion postponed, inserted after actions finish
Decision taken to insert opportunity
Opportunity finishes, continue core plan

Figure 8.10: *Postponing opportunity insertion, waiting for actions to finish before inserting*

- Are the weakest preconditions of the opportunity satisfied?
- Are the weakest preconditions of the remaining plan, after the opportunity has been executed, satisfied?

The set of weakest preconditions is the minimum set of literals that are needed to be true to satisfy the preconditions in the plan fragment. These can easily be determined by identifying which preconditions of actions in the opportunity are not established by earlier effects of actions in the opportunity [Dijkstra, 1976, Hoare, 1983]. Performing the above two checks allows the executive to determine if the opportunity can be safely inserted without violating any conditions.

If simultaneously executing actions are to be permitted, then there must be support for dealing with any actions that may currently be executing at the time of taking the opportunity. These could be actions that monitor battery level or even those that use (potentially conflicting) resources. A very simple method for doing this would be to terminate the dispatch of new actions to the executive and wait for currently executing actions to finish before inserting the opportunity, as in Figure 8.10. In other words, no new actions are started, and those that are currently executing are left to finish. Before this is done, the opportunity's preconditions are verified as above to ensure that that it can be inserted without breaking the plan.

An advantage of this postponing technique is that it is very easy with basic analysis to see if an opportunity will fit in with the existing plan. The disadvantage is that the opportunity is mutually exclusive with the rest of the plan, and prohibits any kind of concurrency between the opportunity and the core plan. Unfortunately, this strategy can be highly inefficient in some situations. There may also be actions for which termination is impossible, such as real-time low-level monitoring tasks or actions that never terminate. If concurrency is to be supported when executing an opportunistic plan, a more sophisticated technique must be used. Two such strategies were conceived, *simple opportunity insertion* and *advanced opportunity insertion*.

8.6.1 Simple Opportunity Insertion

The strategy detailed below ensures that it is not possible for actions to conflict, and the plan will always be able to complete. For simple opportunity insertion, actions are represented in a STRIPS-style manner, with preconditions and effects but no invariants. Start effects and finish preconditions are not used.

The rules for simple opportunity insertion allow concurrency with a limited amount of computation power. If any of the rules are violated, then the opportunity cannot be inserted, and it is ignored. Execution of the core plan continues as normal.

1. *There must be enough time remaining to execute the opportunity and any remaining plan.*
   To calculate this, take the actual amount of each resource remaining in the system, then subtract the conservative estimates (95th percentile) for the opportunity and remaining plan. Then subtract the conservative estimate of resources used by each currently executing action (the executive may be able to provide updated estimates here, otherwise use the initial conservative estimate generated by the planner). If the resulting value is positive, then there are enough resources remaining. The executive may also be able to provide more information about how much of a specific primitive executive action has been completed, allowing these estimates to be more accurate.

2. *The weakest preconditions of the opportunity must be satisfied before dispatching the opportunity.*
   If all the weakest preconditions are satisfied at the current time-point during execution of the core plan, then the opportunity has the necessary preconditions to be executed.

3. *After executing the opportunity, the resulting state must be one in which the remainder of the core plan can execute.*
   To do this, calculate the weakest preconditions for the remainder of the core plan, and check that these will be satisfied in the world state after executing the opportunity. Also the planner, when generating the plan, must guarantee that all

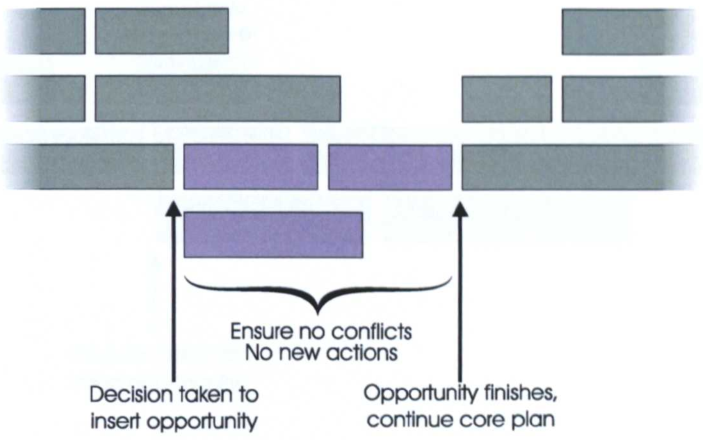

Ensure no conflicts
No new actions
Decision taken to insert opportunity
Opportunity finishes, continue core plan

Figure 8.11: *Simple opportunity insertion*

possible outcomes of the opportunity (no matter how quickly or slowly the individual actions take to finish) have the same effects. This is, of course, bounded by the orderings imposed on actions within the opportunity.

4. *Check that the currently executing actions do not conflict with the opportunity to be inserted.*
   It could be the case that any currently executing action could terminate at any point during the opportunity due to uncertainty. Therefore, to guarantee that there will be no conflicts, the currently executing action must be checked to see if it undoes any precondition of any action in the opportunity.

An illustration of this technique can be seen in Figure 8.11. Although computationally efficient, this strategy can lead to opportunities not being inserted at points where they could otherwise be used. For example, if a currently executing action is highly unlikely to conflict with the opportunity then it cannot be inserted at all, as in Figure 8.12.[4]

8.6.2 Advanced Opportunity Insertion

A second method of opportunity insertion can be considered that can take advantage of conflicts that are very unlikely to occur. This is identical to above with the exception that the system only considers pairs of actions in the core plan and opportunity that are likely to overlap, instead of every possible pair. In this case, the technique has the potential for much less wasted execution time, but at the expense of requiring much

[4]The precondition of P is satisfied internally by one of the later actions in the opportunity, and is not shown here.

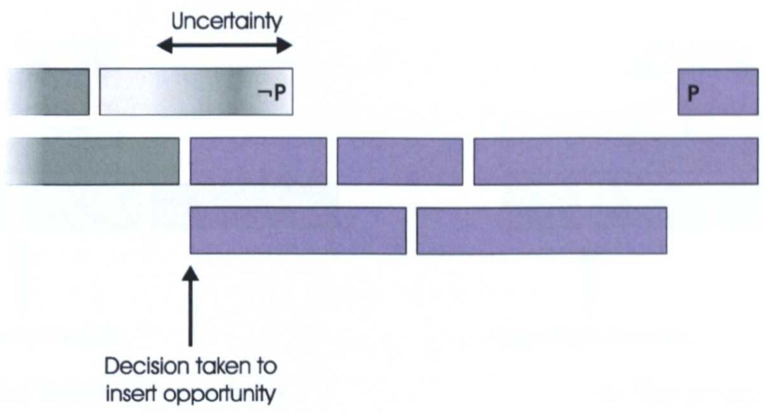


Figure 8.12: *Advanced opportunity insertion*

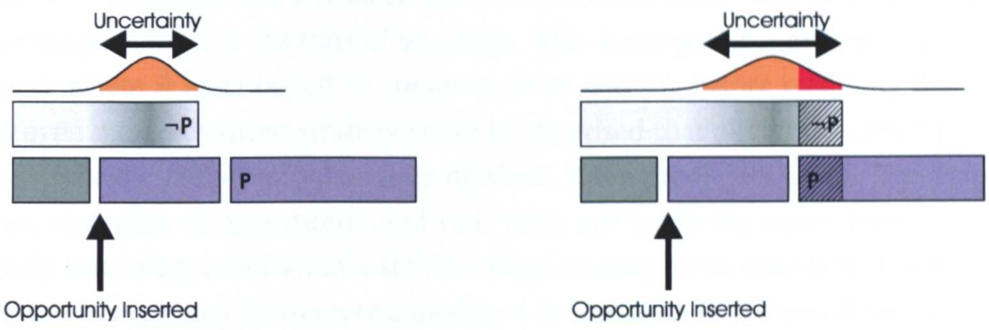
more validation to ensure that the opportunity will fit into the plan. Advanced opportunity insertion also expands on simple opportunity insertion to allow PDDL 2.1-style actions [Fox and Long, 2003], including invariants, pre-conditions and post-conditions.

Advanced opportunity insertion also permits nested opportunities, allowing opportunities to branch off at any point in the current opportunity. This much richer language allows the construction of many more 'useful' plans: those more likely to exist in real-world problems. The disadvantage is that this flexibility is traded off for a much costlier computation time, a very important factor in many executives with limited CPUs.

The advanced set of rules needs to be less restrictive on when opportunity insertion should occur, but at the same time still provide a guarantee that the plan will actually complete. This is much more complex due to the sheer number of interactions, overlappings and execution traces. Advanced opportunity insertion is defined as follows:

1. *There must be sufficient resources remaining to execute the opportunity and any remaining plan in the execution stack.*
   This can be calculated as before (with simple modifications) to the simple opportunity insertion method.

2. *There must be no currently executing actions that* ***are likely*** *to interfere with actions in the opportunity to be inserted.*
   With the simple method, all actions were checked to see if any conflicts existed between *any* two actions, no matter how unlikely that problems would occur. To increase the chance of being able to insert an opportunity, only actions that overlap when the actions run to the 95th percentile duration are considered. In Figure 8.13(a), there is an action in the core plan with the effect of ¬P, and an



(a) Permissible opportunity (b) Non-permissible

Figure 8.13: *Evaluating which actions are likely to interfere*

action in the opportunity with a precondition and invariant of P. The simple opportunity method would prohibit this opportunity from being inserted because of the conflict between these two actions, even though it is highly unlikely that they will interfere. Figure 8.13(b) shows a situation in which the 95th percentile execution duration overlaps with the conflicting action. In such a case the opportunity shouldn't be inserted due to the relatively high probability that the action will fail.

3. *There must be no actions in the opportunity to be inserted that* ***are likely*** *to interfere with any of the currently executing actions.*
   Because PDDL 2.1-style actions can have end conditions, the opportunity's effects have to be considered, as above, to ensure that none interfere with currently executing actions. This can be inferred in a similar manner to above, only checking actions that overlap in the 95th percentile duration.

4. *It must be true that whatever happens, the opportunity will be able to complete after dispatch, regardless of interleavings or orderings due to uncertainty.*
   It may be the case that a certain interleaving of actions produces a conflict. This is incredibly computationally expensive to evaluate, but must be checked so as to ensure that the opportunity can be inserted. The planner may however be able to evaluate this and make these guarantees during plan construction.

5. *It must be true that the remainder of the core plan will be able to complete after the opportunity has finished.*
   The world state when the opportunity has finished must be one in which the core plan is able to complete, no matter what happens. Again, the planner may be able to determine this at the planning stage.

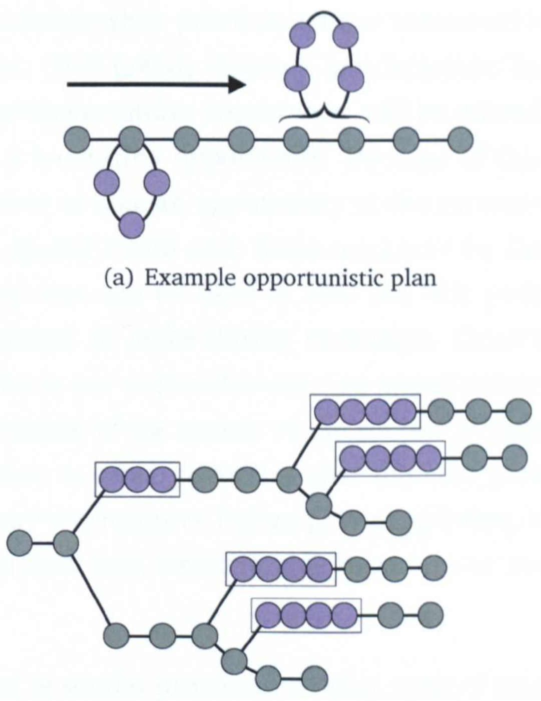
In the cases of the simple and advanced rules, concurrency of the core plan and opportunity is only permitted at the start of an action. This is not possible at then end of the opportunity where it must be left to complete in its entirety before resuming the core plan. An even more advanced strategy could be theorised that allows this concurrency to occur before the opportunity has fully finished, if the predicates allow. The transitions from core plan to opportunity and vice versa are much the same; there is a set of currently executing actions and a set of actions waiting to be executed. Essentially, the executive is switching between two modes of execution, and the transition between these has to be controlled and regulated to ensure that it can happen without need for outside intervention. There has been some research into mode change analysis in the area of scheduling [Tindell et al., 1992], which checks the schedulability of tasks during the change between modes. This could potentially be adapted for use in the mode changes between core plan and opportunities.

8.7 Should an Opportunity Be Inserted?

Aside from the problem of determining if an opportunity can be inserted safely, there is another issue that must be addressed: how to choose when to insert an opportunity, and which opportunities to use when presented with a set of choices. Decisions made at any timepoint may lead to an inefficient use of resources, for example a low-utility opportunity selected at the current timepoint may consume resources that could have been used later for a high-utility opportunity. Because of this, a strategy is required for opportunity selection to ensure the best use of resources.

A successful opportunity selection strategy must be able to provide a guarantee of plan completion, as well as being able to accommodate uncertainty management. It should also be robust to changes in resource consumption amounts, for example if an action consumes more of a resource than initially allocated. A good strategy may also be able to take advantage when actions use less resources than expected, leading to an unexpected amount of extra resources becoming available.

The opportunity selection strategy was built on top a conservative plan execution strategy as described in Section 3.4. To simplify matters, only simple opportunistic plans that do not allow concurrency will be permitted. During execution, the worst-case scenario is initially assumed with each action consuming an amount of resource equal to (for example) the 95th percentile of the distribution. If actions consume less than this amount then they become free to use in opportunities. In the case that an action consumes more than one resource, for example time and fuel, then multiple distributions need to be dealt with. It is likely that the values produced by multiple distributions are dependent on each other, but the worst case scenario is that they are independent



(a) Example opportunistic plan

(b) Expanded opportunity tree

Figure 8.14: *Expansion of an opportunity tree*

variables. To ensure that the action still has a 95% chance of completing given the resources available, the $\sqrt[n]{0.95} \times 100$ percentile must be used, where n is the number of different types of resources that the action consumes. This ensures that the probability of each of the distributions consuming less than the percentile is 0.95. For example with four resources, $\sqrt[4]{0.95} = 0.987$, so the 98.7th percentile is taken as the value to use for creating the conservative plan. The chance of all four resources using less than this amount is 0.9747^4 (i.e. 95%).

8.7.1 Opportunity Trees

It is sometimes useful to expand out the opportunities into an opportunity tree. An opportunity tree is simply the expansion of all possible execution paths through the opportunistic plan. Figure 8.14 shows the example expansion of a simple opportunistic plan (a) into an opportunity tree (b). This expansion makes it easier to see the consequences of particular decisions and how they might benefit or hinder execution later into the plan. An opportunity tree can be executed by starting at the initial node and executing the actions in sequence until a branch is reached. At each branching point, a decision has to be made as to which branch to execute.

During execution of a plan, it may be tempting to execute an opportunity at the first instant that the necessary resources become available, a greedy strategy. Resources would have become available when previous actions consumed less than their allocated 95th percentile amount. This greedy strategy may however lead to bad decisions; it could be the case that a higher-utility opportunity will be missed because the resources were spent earlier on a low-utility opportunity. Because of this, there is a need for a process to decide whether to take an opportunity at the current time-point, or whether it will be beneficial to ignore it and save those resources for later use. To make these choices, an opportunity tree can be used to find the best path through the tree and therefore the best decisions to make during execution. Great care must be taken to ensure that the executive is not required to execute opportunities that could exceed the resources available. Because of the nature of the subject of planning and execution, it is fundamental that there is some guarantee that the core plan will complete. If the executive runs out of any one resource before plan completion, then the plan has failed — possibly resulting in data loss, irrecoverable situations or even irreparable damage to the executive.

There have been studies of similar problems in other areas of research from which experiences can be gained. One possibility is to view the problem as a modified decision-tree problem, or perhaps even as a multi-dimensional vector-packing task [Epstein, 2003] (except that the sizes of the objects to be packed are unknown until selected). A similar problem has been investigated in the area of economics [Buffett and Spencer, 2003], where fluctuating prices influence the decisions of a buyer over time, but this relies on the costs being known before taking actions — in plan execution the "costs" are only known after the action has executed, and anything before is simply a probability distribution. In economics, the exact price is known before the commitment to purchase is made. There has also been research looking at MDPs [Boyan and Littman, 2000] with uncertainty over continuous time distributions, where actions have different outcomes depending on their start time. In [Bresina and Washington, 2000] a technique is discussed that deals with uncertainty in time, but with fixed resource amounts. In many real-world cases where there is uncertainty, it is unlikely to be limited to just one variable, because the uncertainty in one variable will often depend on another. For example, fuel or battery consumption may be a constant amount during the course of an action, and therefore dependent on how long that action takes to execute.

In the problem of uncertain resource consumption as described, the absolute amounts are not known until the action has successfully completed and terminated. In practice, it may be that with some actions it could be possible to predict forwards and continuously improve estimates of values. For example, a robot travelling between two locations will know how fast it has progressed along the first part of its route, allowing it to estimate time remaining. Despite this, the assumption that these estimates are incalculable will be retained for simplicity.

As stated above, the opportunity tree can be used to calculate which opportunities to execute. This is done by calculating the expected utility (on average) of making each choice in the plan, thereby providing a value on which to base decisions. Each branch in the opportunity tree is labelled with an expected utility, and the best decision is therefore the choice from which can be expected the largest payoff — the branch with the highest expected utility.

Expected utility is calculated in a similar manner to how probabilities of outcomes are calculated in decision trees, but there are some subtle differences, which are outlined in the table below.

Decision Tree	Opportunity Tree
Probability associated with each branch is the probability that it will occur.	Probability associated with each branch is established from resource consumption distributions as well as how much of each resource will be available.
Probabilities at each choice-point add up to 1.0.	Probabilities at each choice-point do not add up to 1.0.

8.7.2 Expected Utility Calculation

In an opportunistic plan, the opportunities need to have utility values assigned to them so that decisions can be made about which to execute. These could be manually assigned by a human operator, or perhaps generated by a planner when constructing the plan according to how useful they are. To assess the relative value of the opportunity at a particular point in the plan, the probability of the actions involved completing successfully must be established. If a single action a has a utility of $U(a)$ and a probability $P(a)$ of successfully completing, then the *expected utility* of that action is calculated as $EU(a) = U(a) \times P(a)$. Likewise, with a sequence of actions (a linear plan) consisting of actions $a_{1...n}$, the expected utility is simply equal to

$$EU(a_{1...n}) = \sum_{p=1}^{n} U(a_p) \times P(a_p) \tag{8.1}$$

In the situation where a node a has multiple children $b_{1...m}$, the expected utility from this choice can be calculated as

$$EU(a) = U(a) \times P(a) + \max(EU(b_1), \ldots, EU(b_m)) \tag{8.2}$$

This is the utility of the node itself plus the maximum expected utility from its children. The maximum value of the children is used because, given the choice, it would be illogical to choose any branch other than the one that on average returns the highest utility.

With the knowledge of the algorithms presented above, it is possible to propagate the expected utility values through the whole of the plan tree to calculate the overall expected utility for each choice. These values can then be utilised as the basis for decisions during the plan execution phase (see Section 8.9).

8.7.3 Expected Utility Propagation

To calculate expected utility for each node, a multi-step process is required. This is as follows:

- Propagate cumulative distributions through opportunity tree
- Calculate probability of each node successfully completing
- Calculate expected utility of each node
- Propagate cumulative expected utility through opportunity tree

Propagate cumulative distributions through opportunity tree

In order to calculate the probability of an action completing, $P(a)$, the probability of the preceding actions completing must be considered. This is done by calculating the cumulative resource consumption distributions throughout the opportunity tree. For example, consider a basic plan consisting of five actions $a_1,\ldots,a_5$ in sequence, each Normally distributed with the duration $N(20,5)$ for time taken to complete. The second node would have the cumulative distribution $c(a_2)=N(40,\sqrt{2\times 5^2})$ and the last would be distributed as $c(a_5)=N(100,\sqrt{5\times 5^2})$ (following the standard rules for summing Normal distributions). If a node has several children then the distributions are copied and propagated down each branch, just as if the nodes were in sequence. Distributions are passed down branches of the tree so that the distributions at any node are the sum of those of its parents, right back up to the root node. Each of these new distributions represents the amount of resources that the plan, up to that point, should have consumed.

Calculate probability of each node successfully completing

Once the cumulative distributions are established, it is then possible to calculate the probability of each node completing given a set of starting resources. It is simply a case of reading from the cumulative distribution the probability that the action will complete, given the resources available at commencement. Continuing the above example, the cumulative distribution $c(a_5)$ indicates that actions $a_1,\ldots,a_5$ take on average 100 seconds to complete in sequence. If the plan dictates that these actions must be completed within 150 seconds, then this is a resource constraint. Given this constraint and these distributions, it is possible to calculate the probability of the plan finishing without exceeding these. If 150 seconds are available, then $p(c(a_5) \leq 150) = 0.977$ (as per the cumulative Normal distribution function). This indicates that there is a probability of 0.977 for successfully completing actions 1–5 all in sequence. This calculation is not just applicable to time, it is also applicable to other resources such as fuel; it could have been the case that the actions consumed fuel instead of time, and that the bound of 150 was the size of the fuel tank used. The probability result is still the same.

Calculate expected utility of each node

As stated earlier, $EU = p(a) \times u(a)$ where $p(a)$ is the probability of the action completing, as calculated in the previous step, and $u(a)$ is the utility assigned to the action.

Propagate cumulative expected utility through opportunity tree

Starting with the leaf nodes, the cumulative expected utility is propagated up through the tree back to the root (current) node. This follows the rule described in equation 8.2.

In the examples above, Normal distributions have been used, but it is likely that real-world actions will have more complicated distributions such as gamma or multi-modal compound distributions. Depending on the structure of these, they could either be approximated with Normal distributions, or represented numerically by using Monte Carlo simulations. The only requirement is that the distribution generated by the addition of two others should be representable and calculable. It is obvious that the quality of the approximation of the real distribution has an effect on the outcome of the calculation, but the cumulative effect will be to smooth down the shape of the distributions. What is more important is that the approximation of the distribution covers the correct range of values. For this reason, it was decided that the Normal distribution was an acceptable distribution for use in the analysis contained within this chapter.

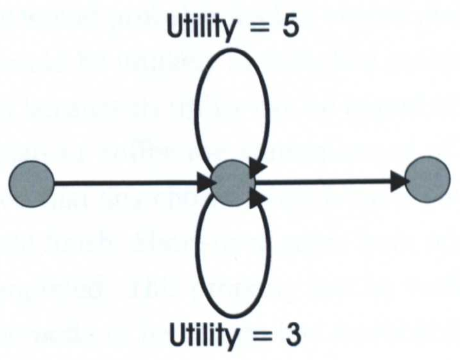
Figure 8.15 shows an example of the cumulative expected utility calculations. In (a), there is a simple plan with three actions in the core plan and two opportunities that branch off from the second of these. One of these has a utility of 3 and another 5. Figure 8.15(b) shows the expansion of this opportunistic plan into an opportunity tree with five different possible execution traces. The graphs below each node represent the cumulative resource consumption distributions, calculated by summing the distributions through the tree, from the root up to that point. The probabilities (P) of each node successfully completing are provided, given the resources available. Expected utility (EU) for each node is also included, as well as the cumulative expected utility (CEU). This tree indicates that at this timepoint, given the available resources, it is better to try executing the opportunity with utility 3 first (with a cumulative expected utility of 3.6) rather than the one with utility 5 (with a CEU of 3.4). If resources permitted, the utility 5 opportunity could always be executed afterwards.

8.7.4 Pruning

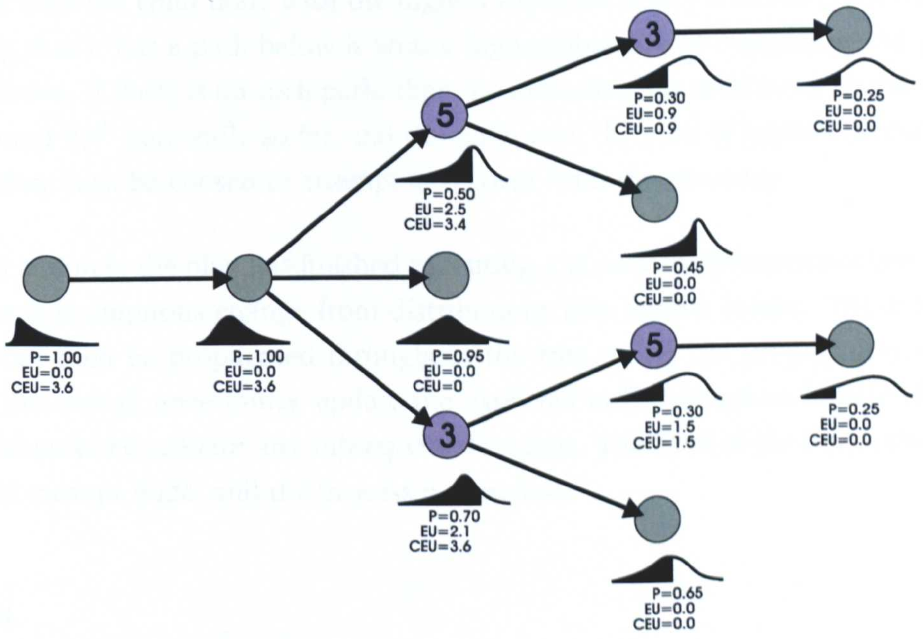
With such a tree as described above, the size of the tree will be exponentially large in relation to the number of branches (opportunities). To counter this exponential increase in size, efficient pruning strategies are required. The problem of pruning decision trees is a well-researched area [Fournier and Crémilleux, 2002] and so this thesis does not attempt to address this issue in depth. Instead, two simple methods are described below to increase the feasibility of working with larger opportunity trees.

The first strategy that can be used relies on the fact that nodes that are highly unlikely to complete can be ignored. This is because the expected utility from them (and any subsequent actions) will be so low as to be insignificant to the overall plan. Any nodes with less than a 5% chance of success are not expanded. If this probability increases (because of extra knowledge gained at the execution stage), then the node can be expanded and its expected utility propagated throughout the tree. This strategy has the potential to prune away large areas of the search tree, but in practice it has not been hugely efficient at doing so when tested. The best result that was achieved was the pruning of a tree with around 5×10^{11} nodes down to just 500,000, but most plans investigated showed little or no reduction in size through pruning in this manner. The technique also suffers from the problems of scale, and plans involving more than around 25 opportunities require vast amounts of processing power.

A second strategy is to employ a dynamic look-ahead process. This is to restrict the number of opportunities expanded in each branch, and keep it bounded to a particular number. This method does not have the same problems of scale as the previous one, but suffers a decrease in obtainable utility in exchange for faster execution and lower memory usage.



(a) Original Plan



(b) Expanded Opportunity Tree

Figure 8.15: *Opportunity tree Expected Utility Propagation*

8.8 Execution

Executing an opportunity tree means using the cumulative expected utilities to make choices as to which opportunity to execute. One possible execution of an opportunity tree would be to always choose the branch with the highest cumulative expected utility. Unfortunately, this strategy would probably lead to overall plan failure as the core plan, excluding opportunities, would be unlikely to complete successfully. The core plan has no utility associated with it because its utility can be regarded as infinite; the executive *must* complete the core plan or suffer the consequences of failure. To counter this problem, it must be ensured that any chosen branch has a path that will lead to a high probability that the plan will finish. Short-term gains from opportunities are not useful unless the plan can be completed. This property can be verified by checking that one of the leaf nodes below the node to be chosen has a probability of completing greater than 0.95.

To execute the opportunity tree, the children of the head node of the tree must be examined. If there is only one node then there is no choice and that node must be moved to, and the action at that location executed. If the head node has multiple children then the child node with the highest expected utility is chosen, but only after ensuring that it has a path below it with a high probability of completing the plan, as stated above. If there is no such path, then the executive has used more resources than the allotted 95th percentile so far, and the node with the path of highest probability of completion must be chosen to attempt to recover from the situation.

Once an action in the plan has finished executing, the previously uncertain first action's resource consumptions change from distributions into known values. These absolute values can then be propagated throughout the tree, using the propagation rules, to reduce the overall uncertainty, update the expected utility values and allow these updated values to be used for any subsequent decisions. The head of the tree is then taken to be the current node, and the process is repeated.

8.9 Results

To collect results, a simulator was written in Java to create random opportunity tree instances and follow execution through in the manner explained above. 10,000 plan instances were randomly generated for each test, varying in number of plan steps (15–30), number of opportunities (5–20) and other minor factors. Each action consumed one of several resources in addition to time. For simplicity, resource consumption distributions were Normally distributed with random parameters. In each case, nodes with a probability less than 0.05 were pruned and assumed to have no expected utility.

Execution strategy	Utility increase	Success rate	Relative CPU time[5]
Conservative	0.0%	100.0%	1.0
1-LA (Greedy)	8.59%	99.98%	18.72
2-LA	12.51%	99.96%	69.52
3-LA	14.90%	99.94%	143.46
5-LA	17.24%	99.96%	315.45
10-LA	18.29%	99.96%	402.53
∞-LA (BEU)	18.31%	99.96%	424.00

Figure 8.16: *Look-ahead Strategy Comparison*

The tests carried out were as follows:

Conservative
: The plan is formed without opportunities, and there are no possibilities to increase utility by executing other actions.

Greedy
: The opportunity tree is executed with a greedy strategy; opportunities are taken at the very first possible point they become available, as long as there is a path that is likely to complete. This is equivalent to a pruning strategy of 1-look-ahead.

***n*-look-ahead**
: Only *n* opportunities into the future were considered, and the rest of the tree was pruned away (but then expanded when required during execution).

Best Expected Utility
: Execution is carried out by choosing the best available node, as explained by the strategy previously discussed. This is equivalent to a pruning strategy of ∞-look-ahead, i.e. no pruning.

The results from the look-ahead strategies detailed above are presented in Figures 8.16 and 8.17. It can be clearly seen that the inclusion of opportunities can increases the utility of a plan greatly. As would be expected, the greater look-ahead that is used, the greater the average utility. This is because the executive can look further into the plan to see the consequences of any decision that it makes. It can also be seen that the time taken to choose an execution path increases very substantially as the amount of look-ahead increases. The success rate drops very slightly if opportunities

[5]Relative CPU time taken to execute plan as compared to Conservative strategy. This is based on the total CPU time taken to calculate the execution path through the plan.

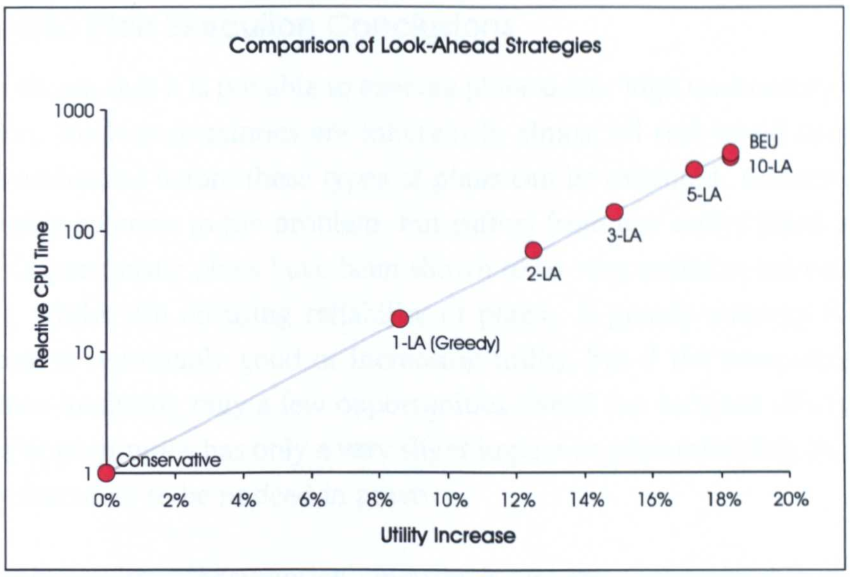


Figure 8.17: *A comparison of the look-ahead strategies*

are used, but the amount is negligible. Planning using a higher percentile, for example the 99th percentile would ensure that this success rate was even higher.

The Best Expected Utility strategy (BEU) resulted in a significantly greater utility than the greedy strategy, the former yielding more than double the utility of the latter. This is, however, at the expense of a huge increase in computation time. With different look-ahead values it is possible to have a compromise between these two strategies. An alternative strategy could be to have a dynamic look-ahead (not discussed here) so that it would become an "anytime" algorithm, offering the best found option at any point during calculation. Such an algorithm would have overheads due to the necessary recalculations that would occur with this.

As would be expected, the results that can be obtained from the simulator are very dependent on the ratio of the size of the core plan to the number of opportunities. A short plan is unlikely to benefit from the option of opportunities, but a long plan is much more likely to have spare resources available to execute opportunities. Longer plans also possess larger uncertainties meaning that the plans are less likely to fail when using the 95th percentile. The results presented above were obtained with plans of 25–35 actions, with up to 15 opportunities available to choose from. Had more actions been used, a higher utility increase would have been seen. In most situations, between two and four of the 15 opportunities were automatically selected by the executive and inserted into the plan during execution. Real-world problems are likely to have more structure than these artificially generated random plans, so it is expected that the utility increase would be even larger in practice than the results seen here.

8.10 Opportunistic Plan Execution Conclusions

It has been shown that it is possible to execute plans under high uncertainty of resource consumption. Such uncertainties are inherent in almost all real-world situations, but need to be addressed before these types of plans can be executed. Conservative planning is a viable solution to the problem, but suffers from low utility plans and wasted resources. Opportunistic plans have been shown to be very useful at increasing overall plan utility, whilst still ensuring reliability of plans. A greedy strategy for opportunity selection is reasonably good at increasing utility, but if the computing power is available then analysing only a few opportunities ahead can increase this further. The insertion of opportunities has only a very slight impact on plan reliability, but this effect is probably too small to be noticed in practice.

Uncertainty in resource consumption amounts is just one of the possible uncertainties that an executive will have to deal with in the real-world. If the techniques presented here were to be combined with other solutions for dealing with incomplete information and probabilistic effects, then it is expected that this would lead to an executive with a very high tolerance to uncertainty. Plans with contingencies could be easily included in this strategy, and would be simply inserted after the opportunistic plan has been expanded into a tree.

CHAPTER 9

Conclusion

It always takes longer than you expect, even when you take into account Hofstadter's Law

— *Hofstadter's Law*

THIS THESIS presented an investigation into the subject of plan execution, introspection and control, and how opportunities can be used to deal with uncertainty and failure. This chapter is a brief summary of the conclusions that have been determined throughout the course of this thesis. In Chapter 1, a list was presented containing the contributions of this thesis. This list is repeated below, with specific references to the relevant sections in which the points were addressed:

- *Identifies suitable tasks from which to learn models for use in evaluating failure prediction*

  Four different tasks for a mobile robot were developed that would contain interesting and useful structures, especially for use in learning models of these tasks (Section 4.1). Also developed were techniques of converting the raw data collected from the robot into practical values for learning the models (Section 4.1.4).

- *Builds on the work in [Fox et al., 2006] and uses this to construct HMMs for the tasks*

  The HMMs automatically learnt from data collected from the four different tasks were presented. Raw data from two executions was manually examined and features identified (Section 5.1). The HMMs were then analysed in depth to understand the structures involved and how best to use them for prediction (Section 5.2). Finally, the results from the HMMs were cross-referenced back to the raw data (through the use of the actual HMM state trajectories) to provide further evidence of their capabilities in determining the current state (Section 5.3). A strong correlation between the actual behaviour of the learnt models and the manually-identified states was found. This indicates that the models were highly successful in encapsulating the behaviours seen in the tasks.

- *Investigates how these HMMs can be used during execution to track progress*

  A variety of repeatable execution failure types were developed, and data was collected from the robot with simulated failures (Chapter 7). These varied from easily identifiable errors through to subtle errors possessing little observable deviation from normal execution.

- *Proposes and empirically evaluates methods for using HMMs to detect execution failure*

  Techniques for failure detection were developed using the probabilities of Viterbi sequences, CLPD values and TSCEM values (Section 6.2). These were analysed by passing the collected failure data into the learnt HMMs (Chapter 7). Both the CLPD values and TSCEM values were shown to be effective at identifying failure whilst minimising false positives. Detailed analysis showed that Kohonen maps of size 20–25 produced HMMs that were most useful in predicting the state of the robot.

- *Outlines how opportunistic plans may be used in situations of uncertainty, and how this relates to failure*

  The link between low-level HMM states and high-level plan states was explained (Section 8.1). Also outlined was how the MADBOT architecture utilised the failure detection routines to terminate execution and return to a known state through the use of recovery actions (Section 8.1.1).

- *Compares and contrasts opportunistic planning with other plan types where uncertainty is present*

  A comparison of planning techniques along with an evaluation of their approaches to utility maximisation was presented (Section 8.2).

- *Greatly expands the definition of opportunistic plans, allowing for much richer plan structures*

  Many new advanced plan structures were identified, including bridges, nested opportunities, multi-point opportunities and bypasses. Techniques for simplifying these structures were also discussed, as well as the proposal for representing a plan as a directed graph (Section 8.7).

- *Identifies methods to allow opportunities to be inserted safely into a plan*

  Several strategies for opportunity insertion were created and described, including simple and advanced strategies for dealing with concurrency in such plans (Section 8.6).

- *Describes a method that allows the best use of resources to maximise utility during execution of an opportunistic plan*

  Opportunity trees were created to enable an executive to make the best choice as to whether to take an opportunity or leave it. This was through choosing the branch with the Best Expected Utility. Pruning strategies were also identified and evaluated. (Section 8.7). It was shown possible to execute plans reliably under uncertainty of resource consumption in this manner.

9.1 Further Research Possibilities

Although great care has been taken to provide thorough investigations and evaluations, there are still aspects of this research that could be extended and developed further. Time limitations meant that it was not been possible to investigate many of these factors involved in the evaluations throughout this thesis.

9.1.1 HMM Learning and Evaluation

There are many relevant variables that influence the size and complexity of the HMMs produced. These factors will also affect other aspects, such as their associated performance for error detection. A full analysis of all of these is well beyond the scope of the work here, but the chosen variables were determined by experimentation combined with intuition. As well as different variables, other methods of learning the models are possible. For example, different clustering algorithms for learning the observations that make up the HMMs.

The tasks from which HMMs were learnt were simple, basic tasks developed to show the capabilities of the HMMs to their full potential. Further work could include much more complex tasks in more "real-world" scenarios. Such tasks would either require more expensive robots, or data from some other non-robotic executive.

In the evaluation of the HMMs for failure prediction (Chapter 7), tests were only carried out on one of the learnt models. Although these tests were extensive, they did not investigate the effect of different sizes of models on failure prediction ability.

Another extension, as briefly mentioned at the end of Chapter 6, is the possibility of learning a second, higher-order HMM that attempts to capture some of the temporal information missed by the HMMs learnt here. There would be a great deal of work involved in processing the data for this, and it may not yield any more useful results than already established.

One further extension to the work includes developing and investigating further techniques for detecting errors in the HMM sequences, for example the NSCEM (Normalised State Count Error Magnitude) values as mentioned in Section 7.2.

9.1.2 Opportunistic Planning

Although the subject of executing opportunistic plans was discussed in detail, there are no planners yet capable of producing these. A method needs devising to create such plans and put these into practise. Possible techniques for creating plans dealing with resource uncertainty have been investigated [Bresina et al., 2002, Dearden et al., 2003], but have yet to be put into practice.

Some planning problems with this type of uncertainty may involve more complicated scenarios than those discussed within this thesis: it may be the case that a problem consists of a combination of limited and unlimited resources. For unlimited resources, any over-consumption is a penalty rather than a failure — time could be viewed as an unlimited resource in many situations, as it does not "run out" in the same way that fuel does. An example of such a problem could be a lorry driver with a working day of a set number of hours. If traffic is bad then the journey between locations may take longer than expected, taking the driver into double-paid overtime. Obviously, such a planner would want to maximise the number of parcels delivered in a day (as the driver is paid per day regardless of whether or not he is driving the whole time), but also minimise the amount of potential of expensive overtime.

Future extensions to opportunistic plan execution include the use of resource production distributions. For example a generator could be used to recharge a depleted battery. This type of action could have a consumption distribution for fuel and time, but a production distribution for battery power. In this way, it would be possible for an executive to choose to take an opportunity to trade off fuel and time to recharge battery power. This would be very beneficial if a later high-utility opportunity required extra power. The opportunity selection process that was developed could easily be adapted to deal with actions of this type.

9.2 Final Summary

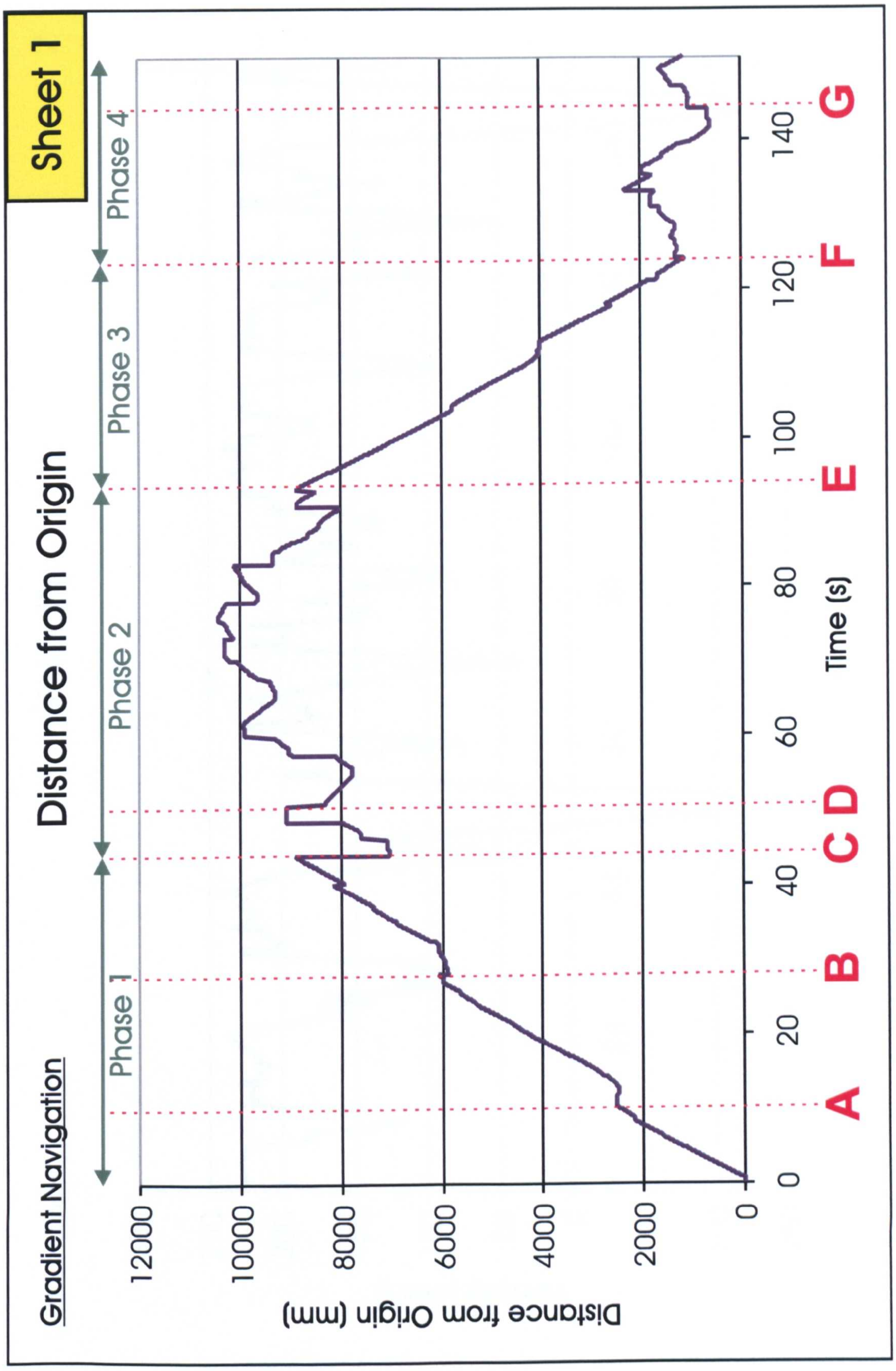
Only through the use of effective monitoring and control can a system be flexible enough to deal with the uncertainties that are present in real-world situations. This thesis has presented and evaluated valid solutions for both of these problems, and has then developed these into useful and effective techniques for use in plan execution monitoring and control.

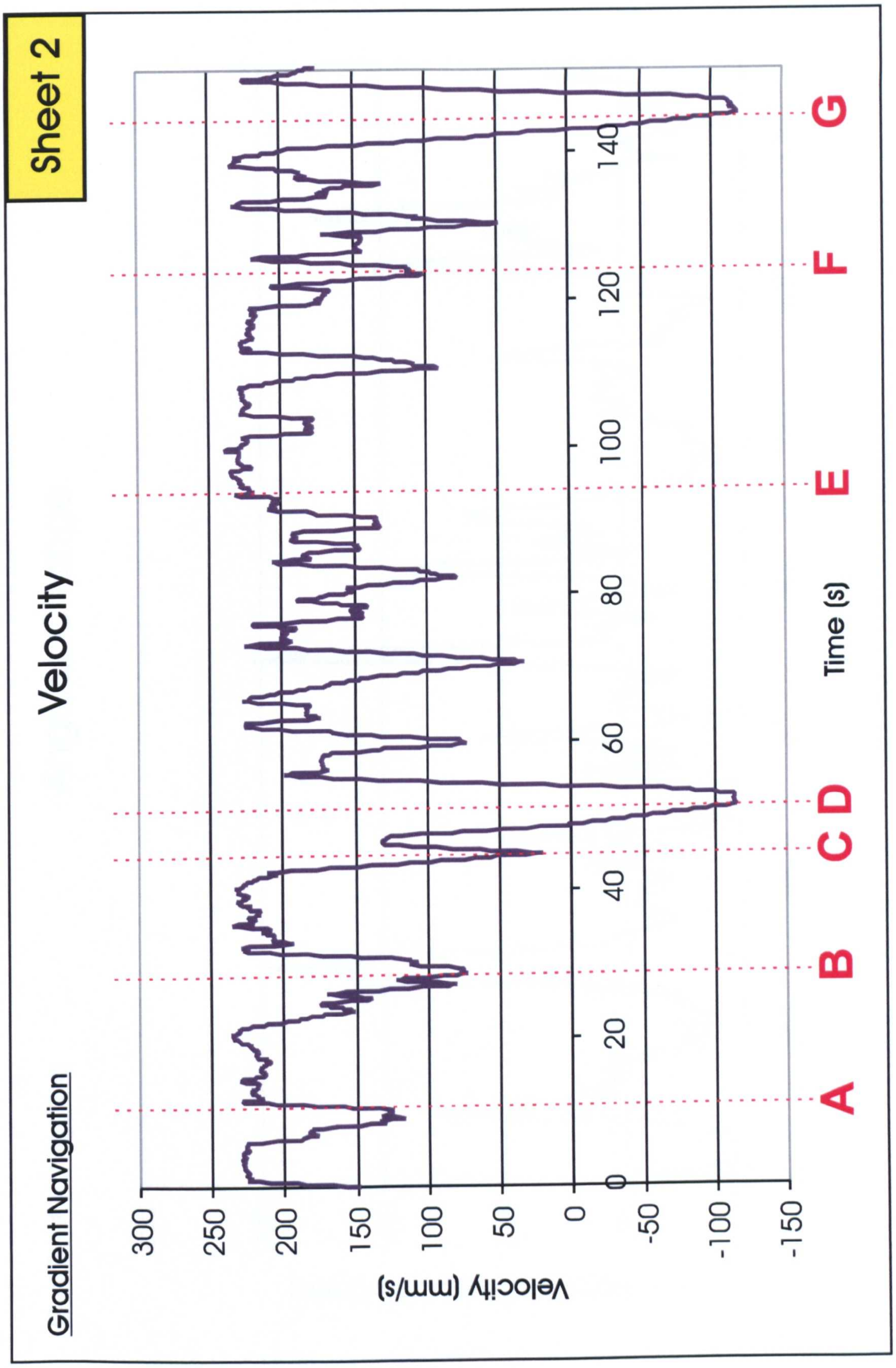
APPENDIX A

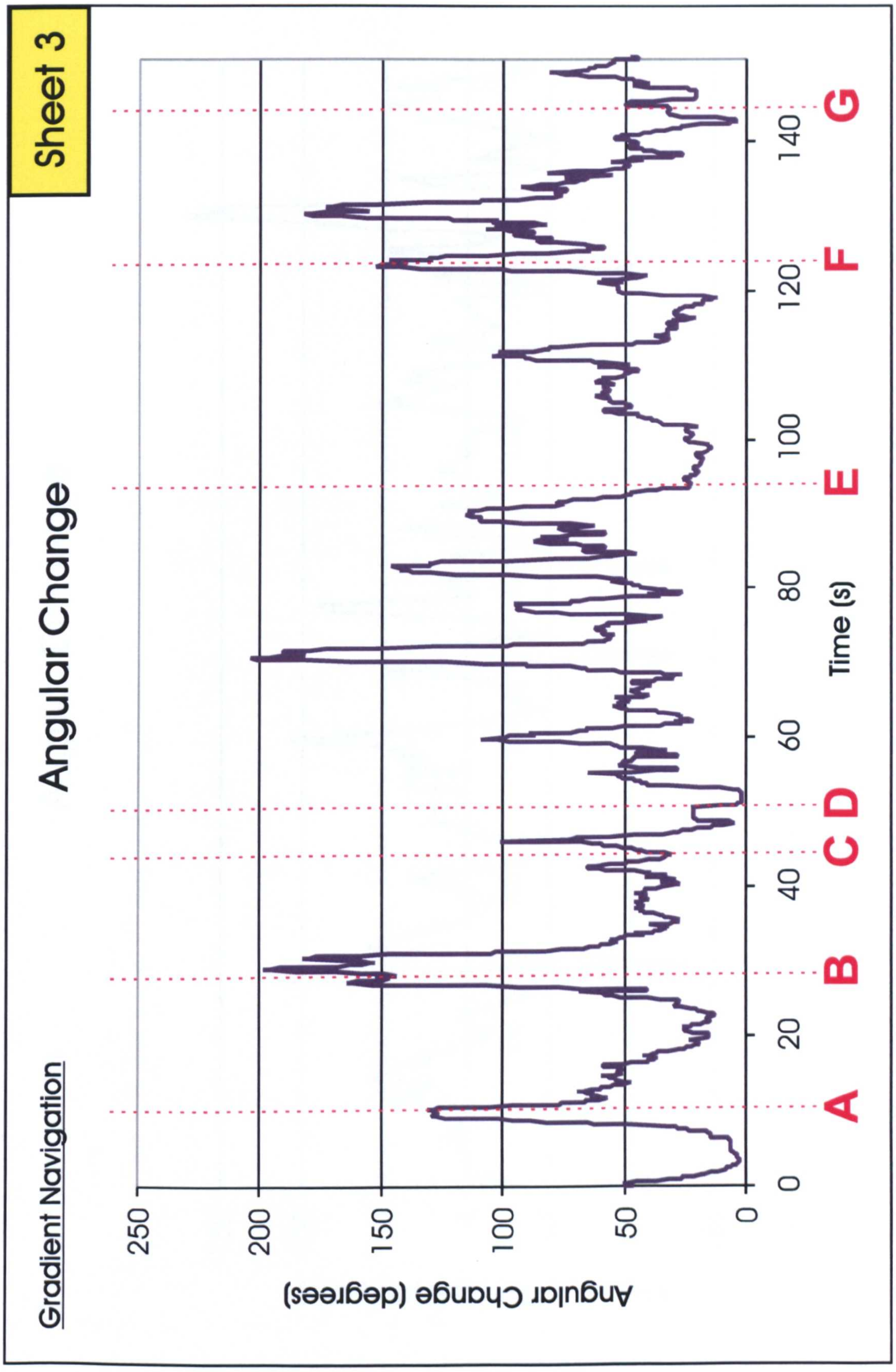
Raw Data Graphs

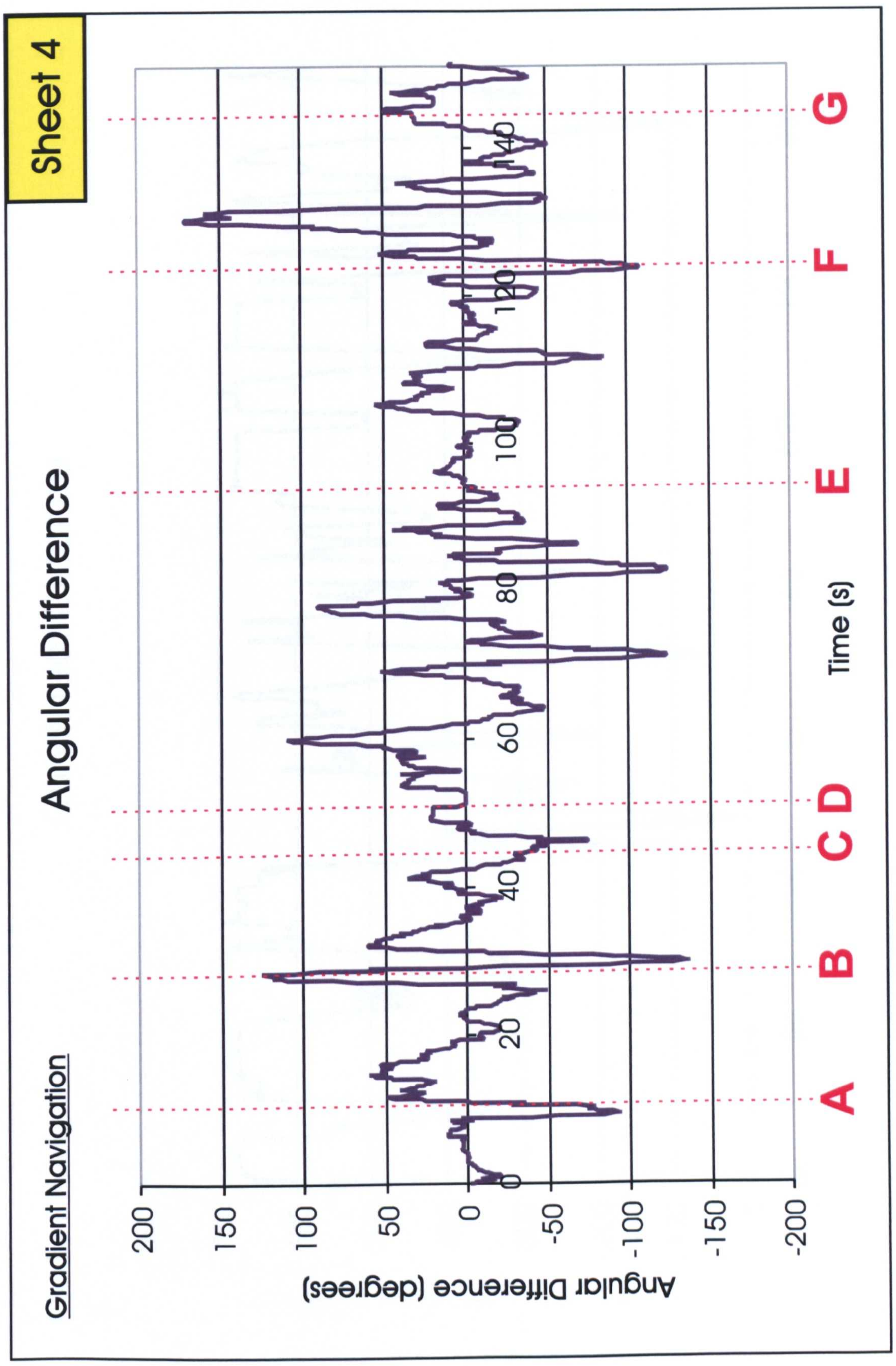
The following pages contain graphs of the raw data for an execution of the Gradient Navigation task (Sheets 1–8) and an execution of the Panoramic Photo with Errors task (Sheets 9–14).

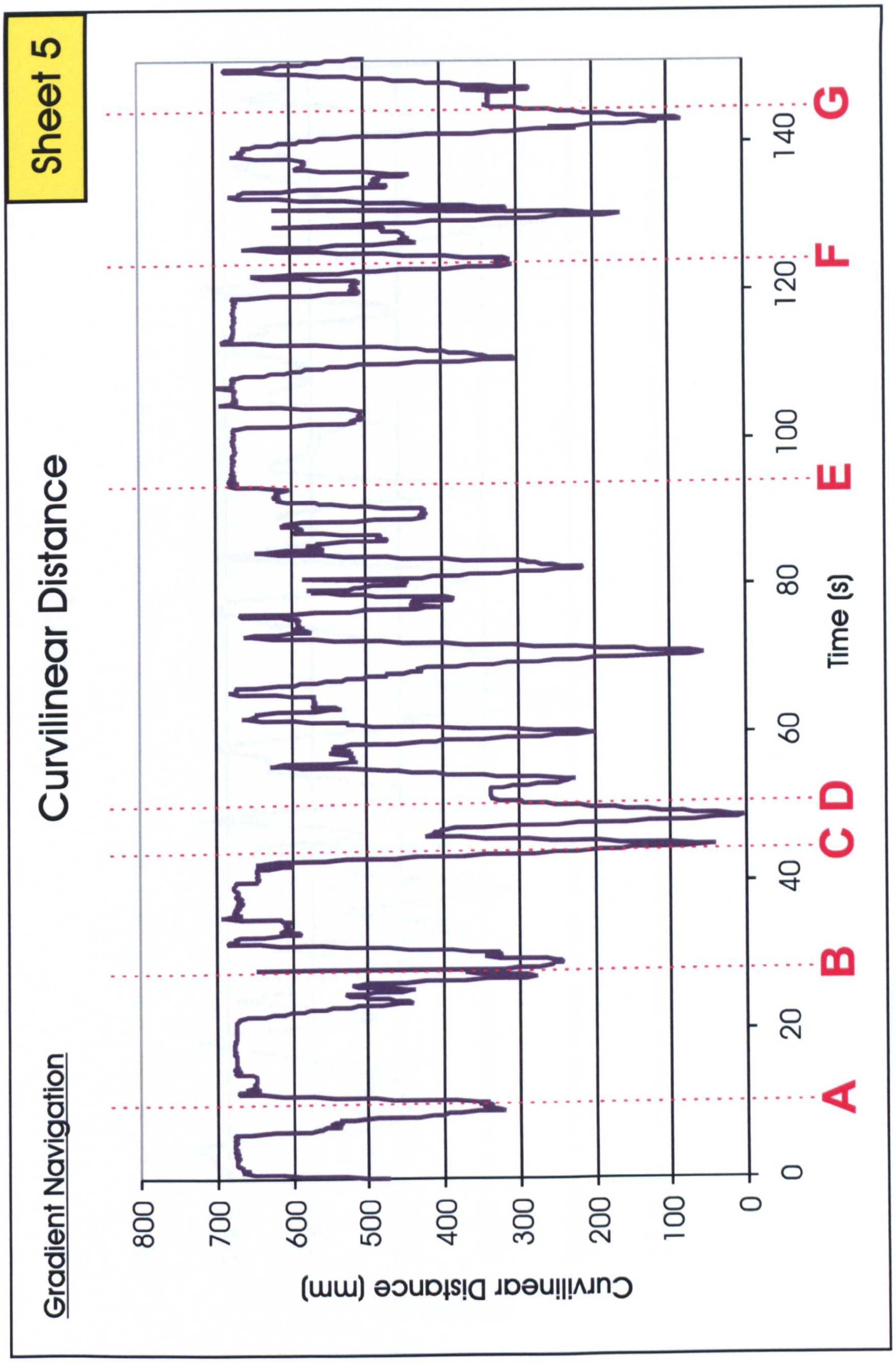
Points of interest have been marked on the graphs with letters and these are referred to extensively in Section 5.1, where these graphs are analysed in depth.

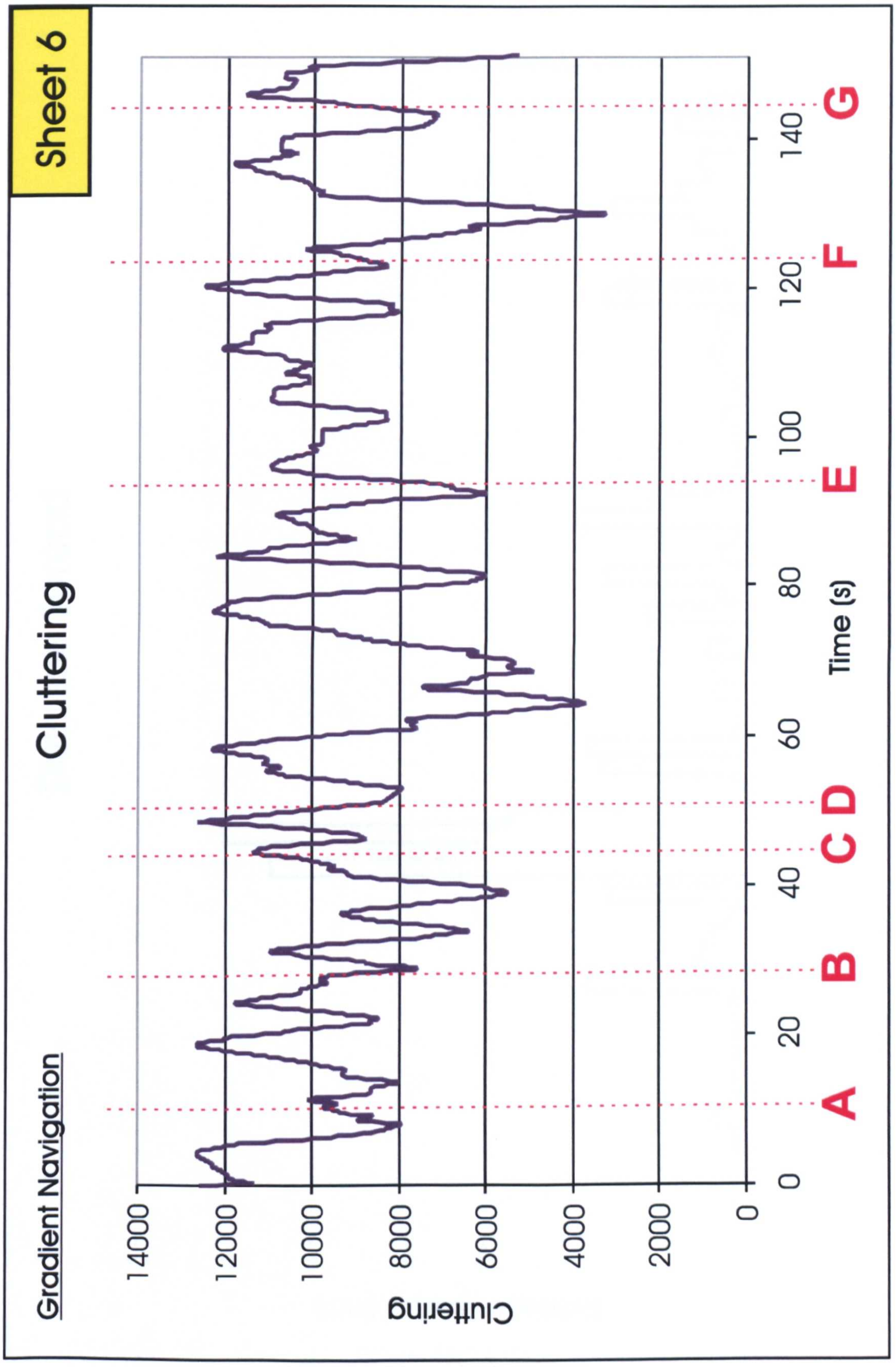

Sheet 1
Gradient Navigation
Distance from Origin
Phase 1
Phase 2
Phase 3
Phase 4
Distance from Origin (mm)
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120
140
Time (s)
A
B
C
D
E
F
G

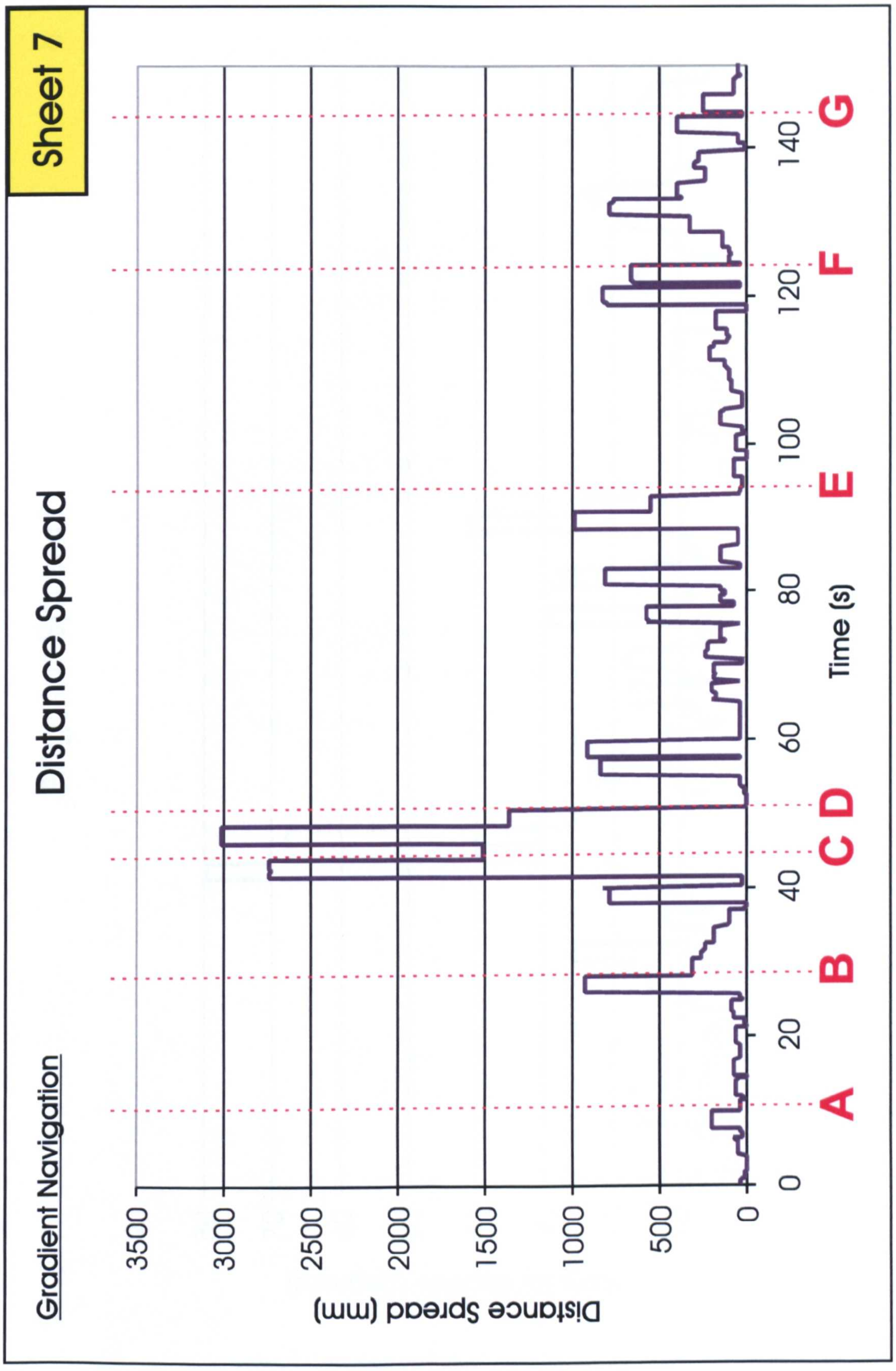

Sheet 2
Gradient Navigation
Velocity
Velocity (mm/s)
300
250
200
150
100
50
0
-50
-100
-150
0
20
40
60
80
100
120
140
Time (s)
A
B
C
D
E
F
G

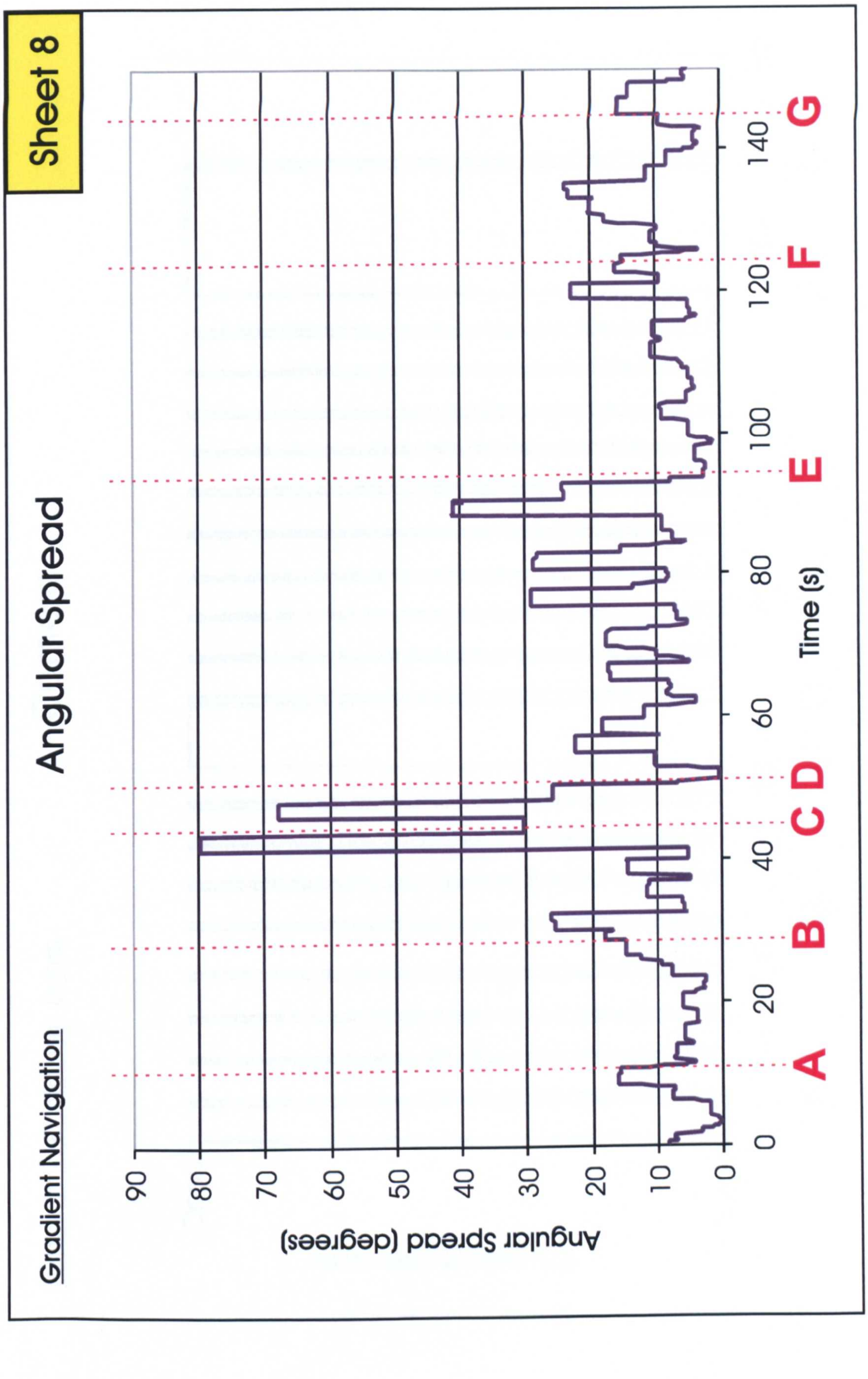

Sheet 3
Gradient Navigation
Angular Change
Angular Change (degrees)
0
50
100
150
200
250
Time (s)
0
20
40
60
80
100
120
140
A
B
C
D
E
F
G

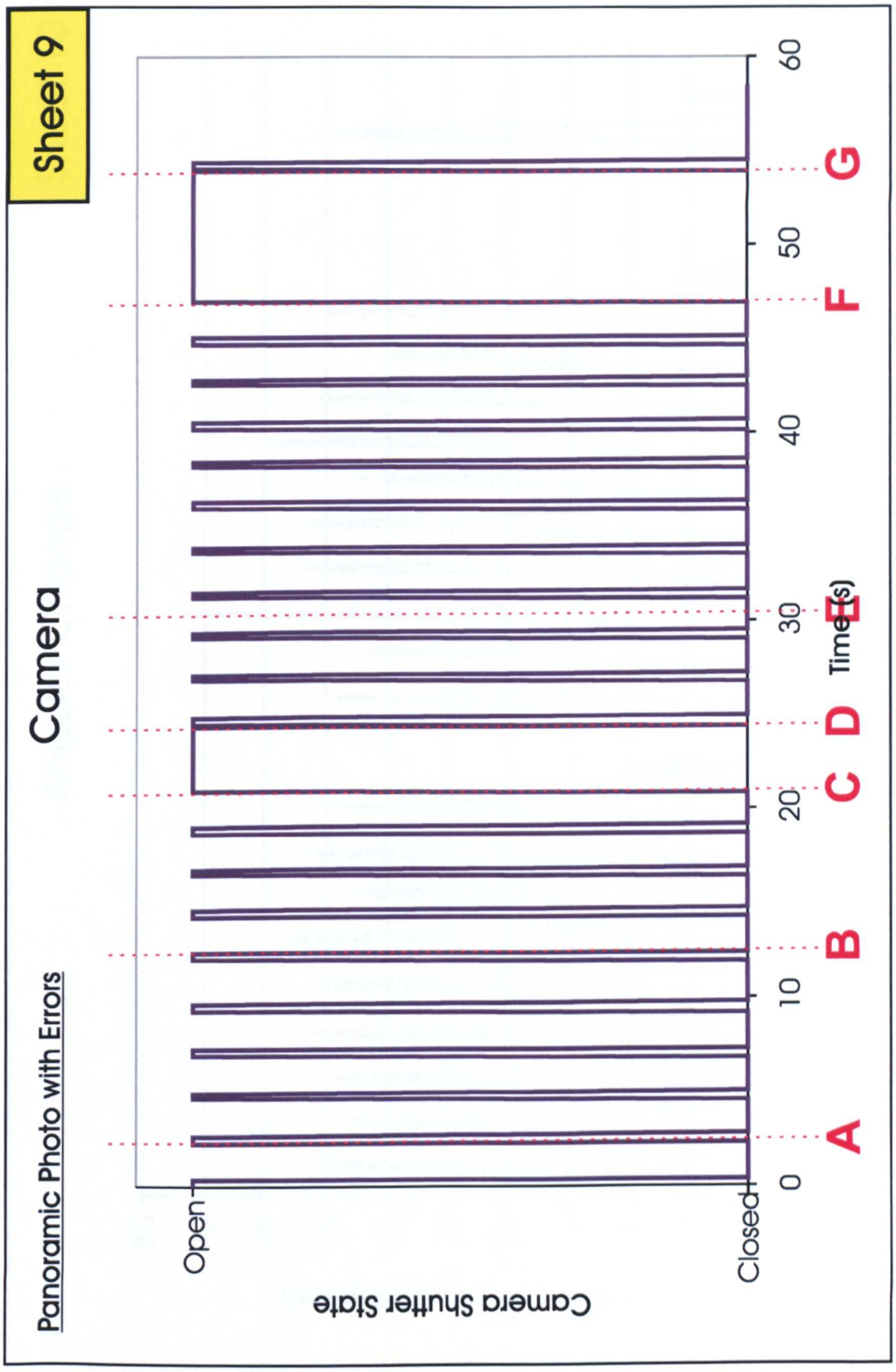

Sheet 4
Gradient Navigation
Angular Difference
Angular Difference (degrees)
200
150
100
50
0
-50
-100
-150
-200
Time (s)
0
20
40
60
80
100
120
140
A
B
C
D
E
F
G

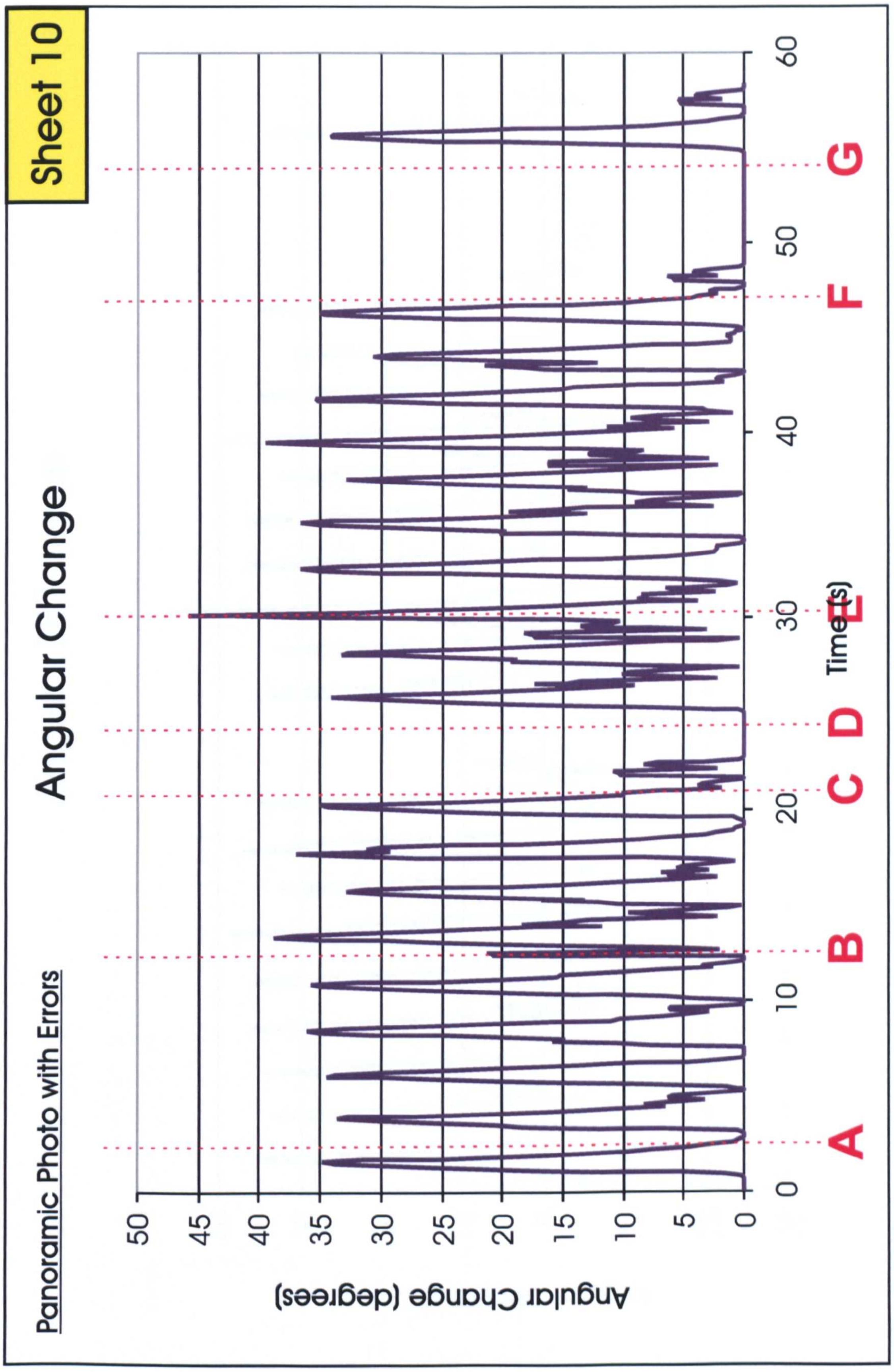

Sheet 5
Gradient Navigation
Curvilinear Distance
Curvilinear Distance (mm)
0
100
200
300
400
500
600
700
800
0
20
40
60
80
100
120
140
Time (s)
A
B
C
D
E
F
G

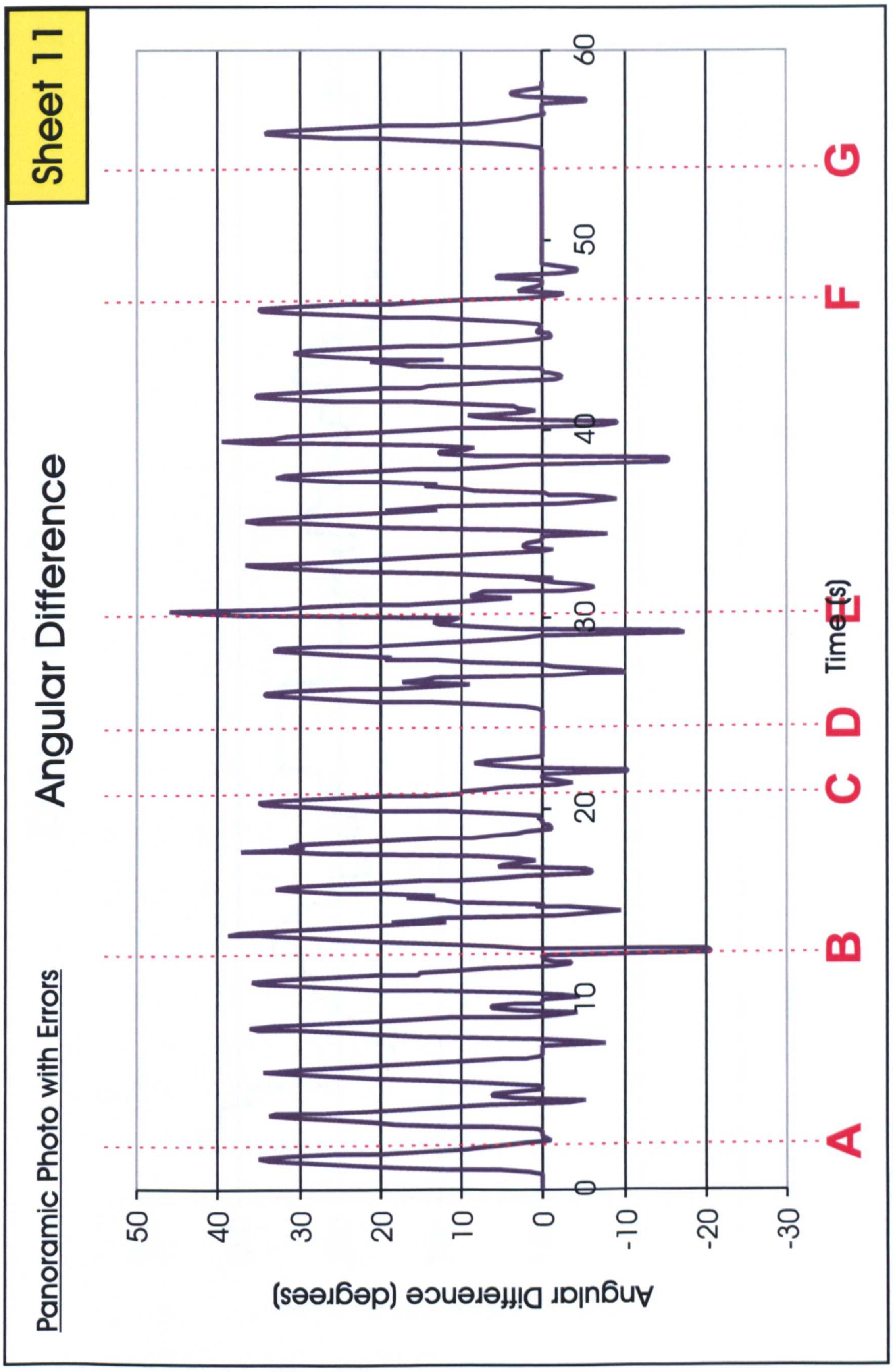

Sheet 6
Gradient Navigation
Cluttering
Cluttering
14000
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120
140
Time (s)
A
B
C
D
E
F
G

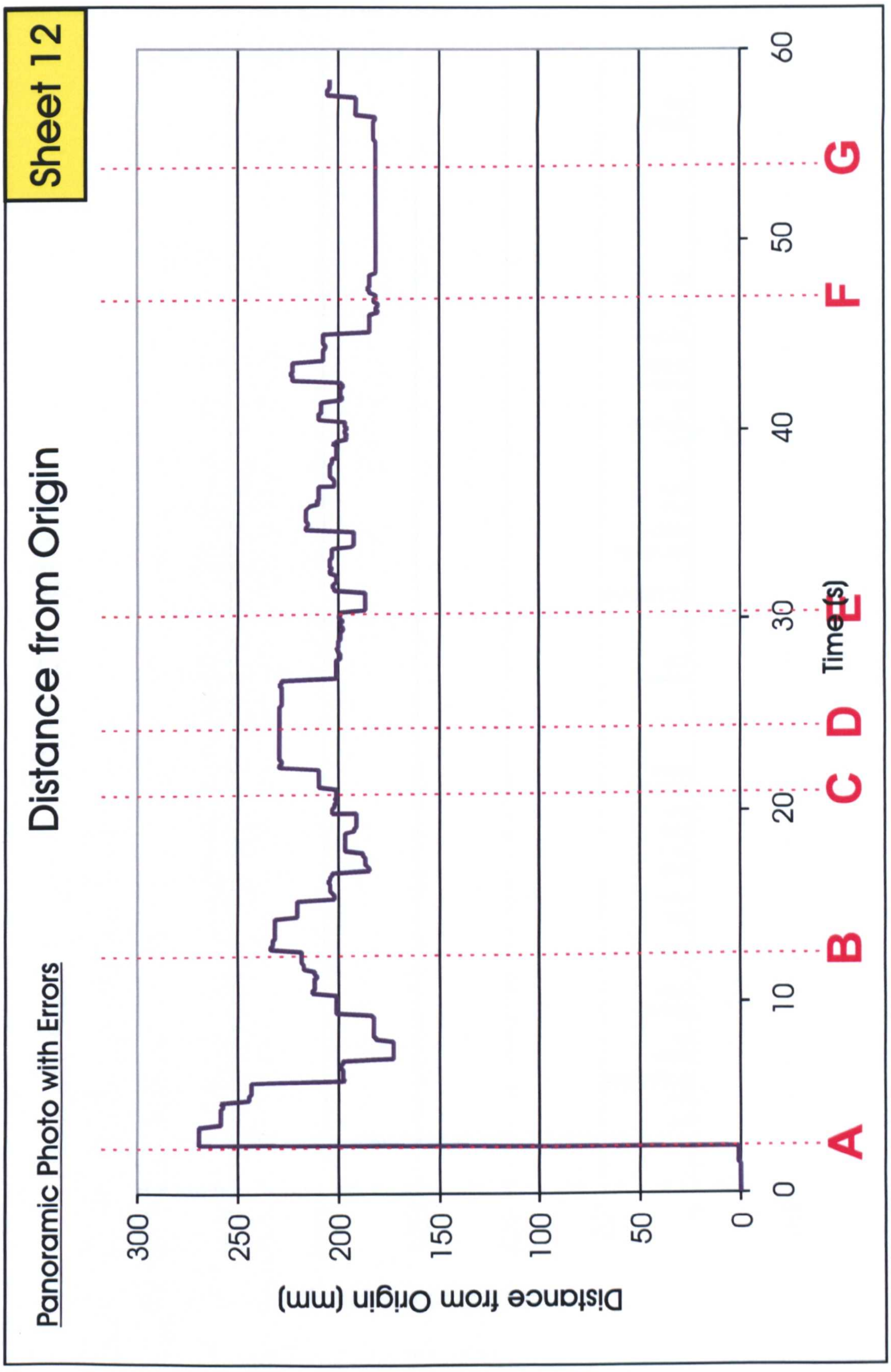

Gradient Navigation
Distance Spread
Sheet 7
Distance Spread (mm)
3500
3000
2500
2000
1500
1000
500
0
0
20
40
60
80
100
120
140
Time (s)
A
B
C
D
E
F
G

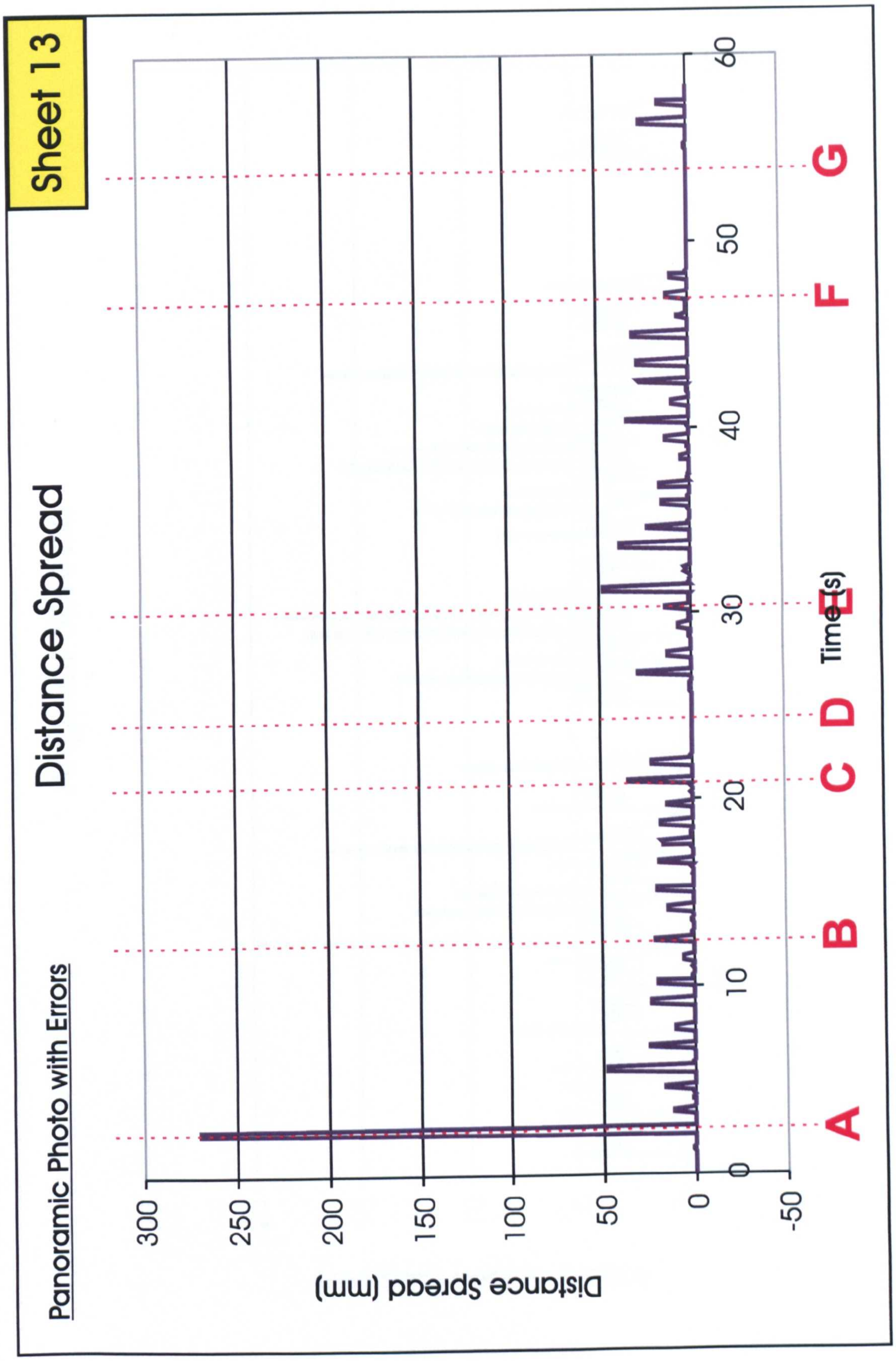

Sheet 8
Gradient Navigation
Angular Spread
Angular Spread (degrees)
Time (s)
0
10
20
30
40
50
60
70
80
90
0
20
40
60
80
100
120
140
A
B
C
D
E
F
G

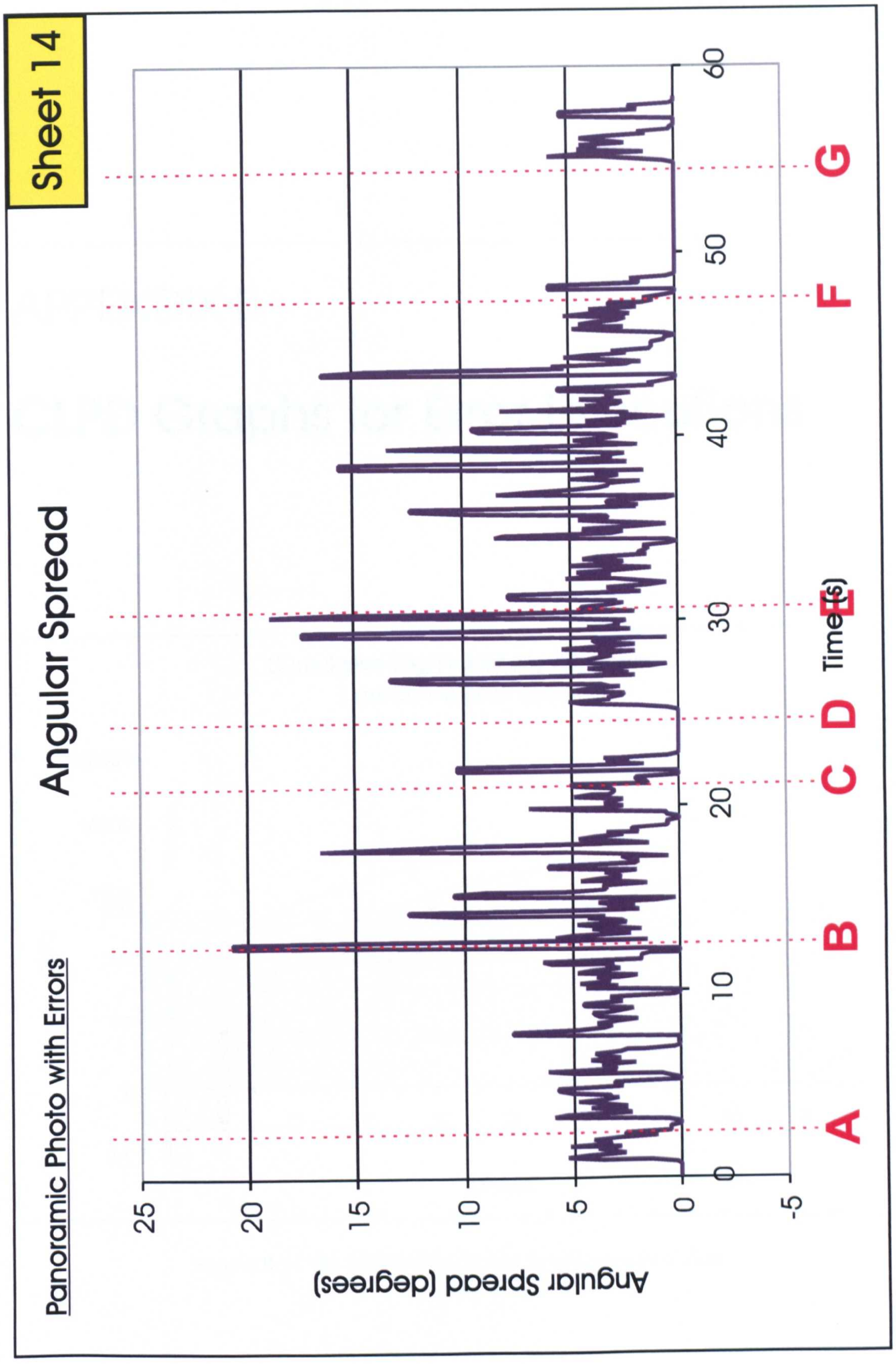

Sheet 9
Panoramic Photo with Errors
Camera
Camera Shutter State
Open
Closed
0
10
20
30
40
50
60
Time (s)
A
B
C
D
E
F
G


Sheet 10
Panoramic Photo with Errors
Angular Change
Angular Change (degrees)
Time (s)
0
5
10
15
20
25
30
35
40
45
50
0
10
20
30
40
50
60
A
B
C
D
E
F
G


Sheet 11
Panoramic Photo with Errors
Angular Difference
Angular Difference (degrees)
50
40
30
20
10
0
-10
-20
-30
0
10
20
30
40
50
60
Time (s)
A
B
C
D
E
F
G


Sheet 12
Panoramic Photo with Errors
Distance from Origin
Distance from Origin (mm)
300
250
200
150
100
50
0
0
10
20
30
40
50
60
Time (s)
A
B
C
D
E
F
G


Sheet 13
Panoramic Photo with Errors
Distance Spread
Distance Spread (mm)
300
250
200
150
100
50
0
-50
Time (s)
0
10
20
30
40
50
60
A
B
C
D
E
F
G


Panoramic Photo with Errors
Angular Spread
Sheet 14
Angular Spread (degrees)
25
20
15
10
5
0
-5
0
10
20
30
40
50
60
Time (s)
A
B
C
D
E
F
G

APPENDIX B

CLPD Graphs for Error Executions

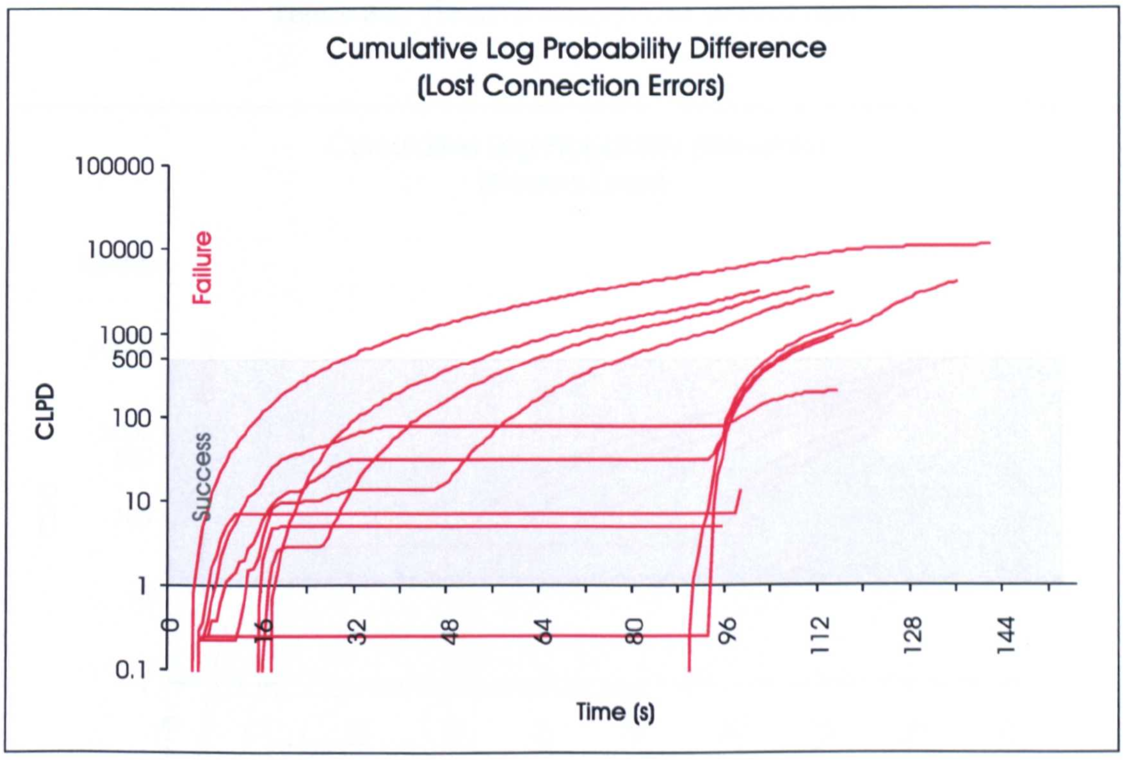


Figure B.1: *The CLPD values for the 'Lost Connection' data*

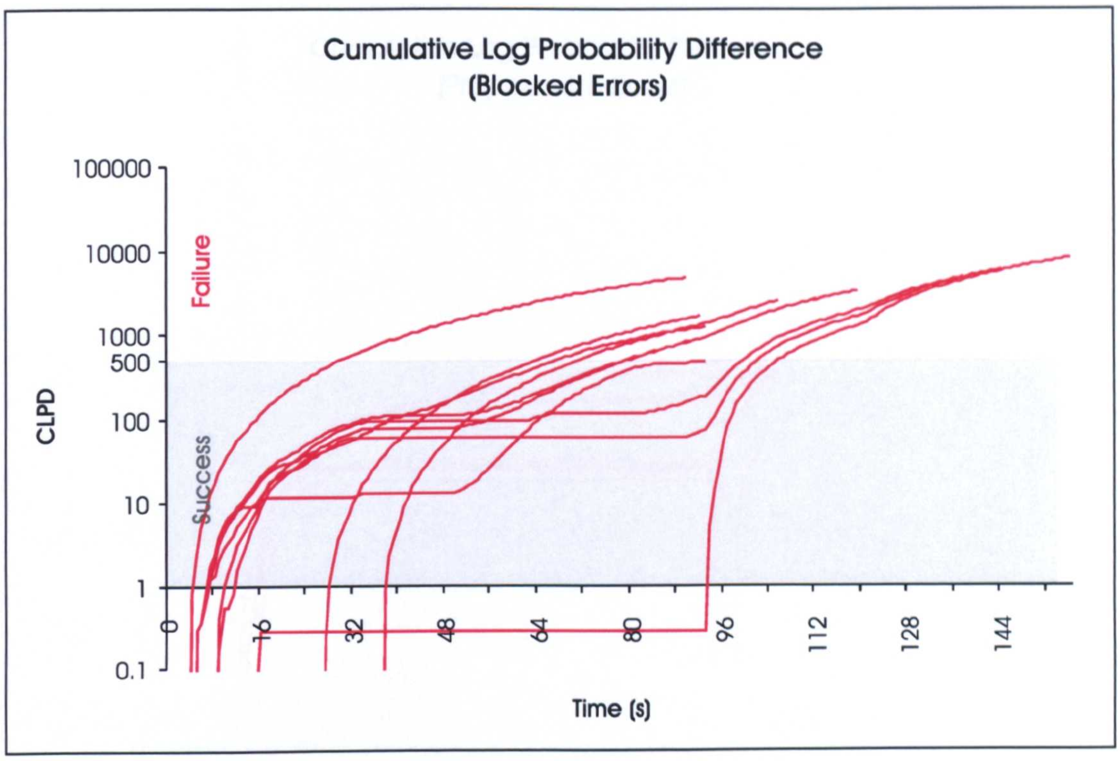


Figure B.2: *The CLPD values for the 'Blocked' data*

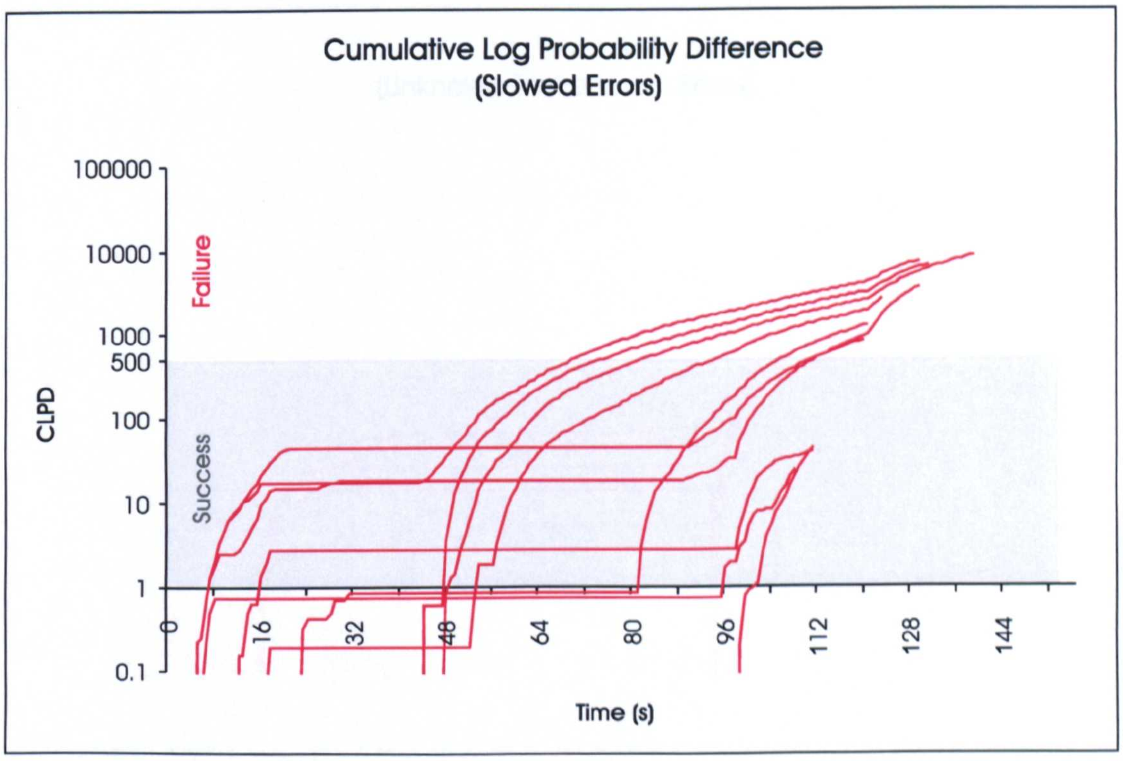


Figure B.3: *The CLPD values for the 'Slowed' data*

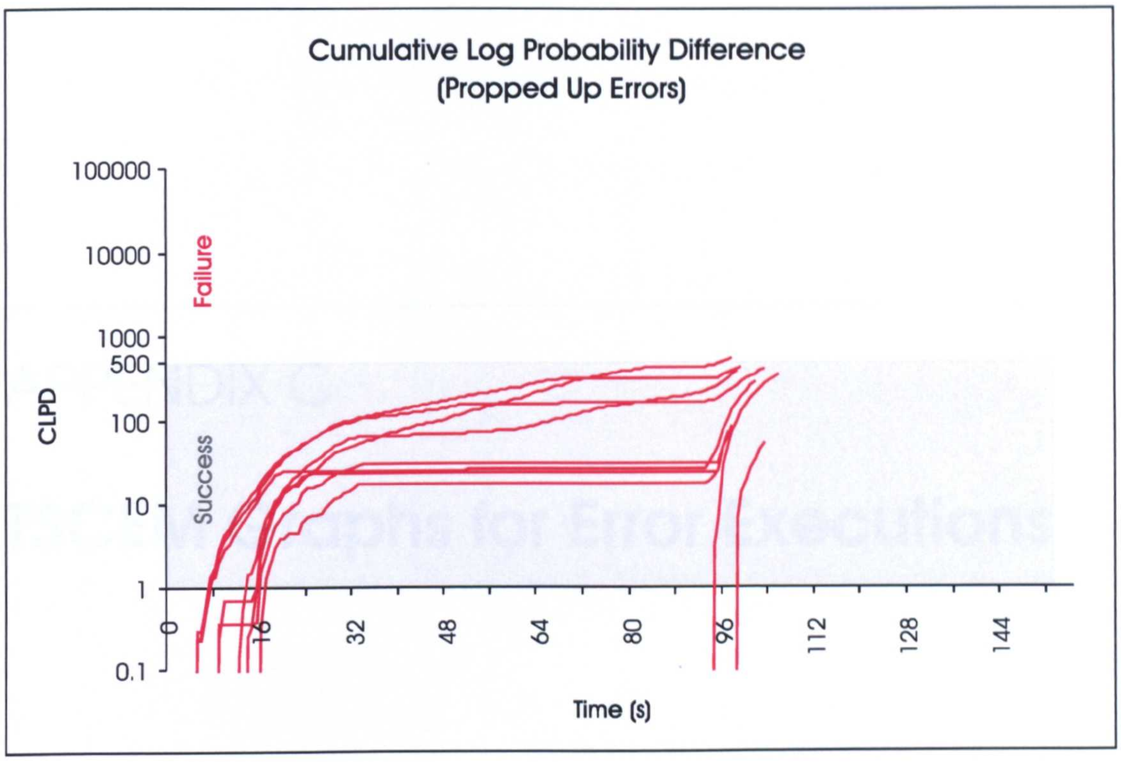


Figure B.4: *The CLPD values for the 'Propped Up' data*

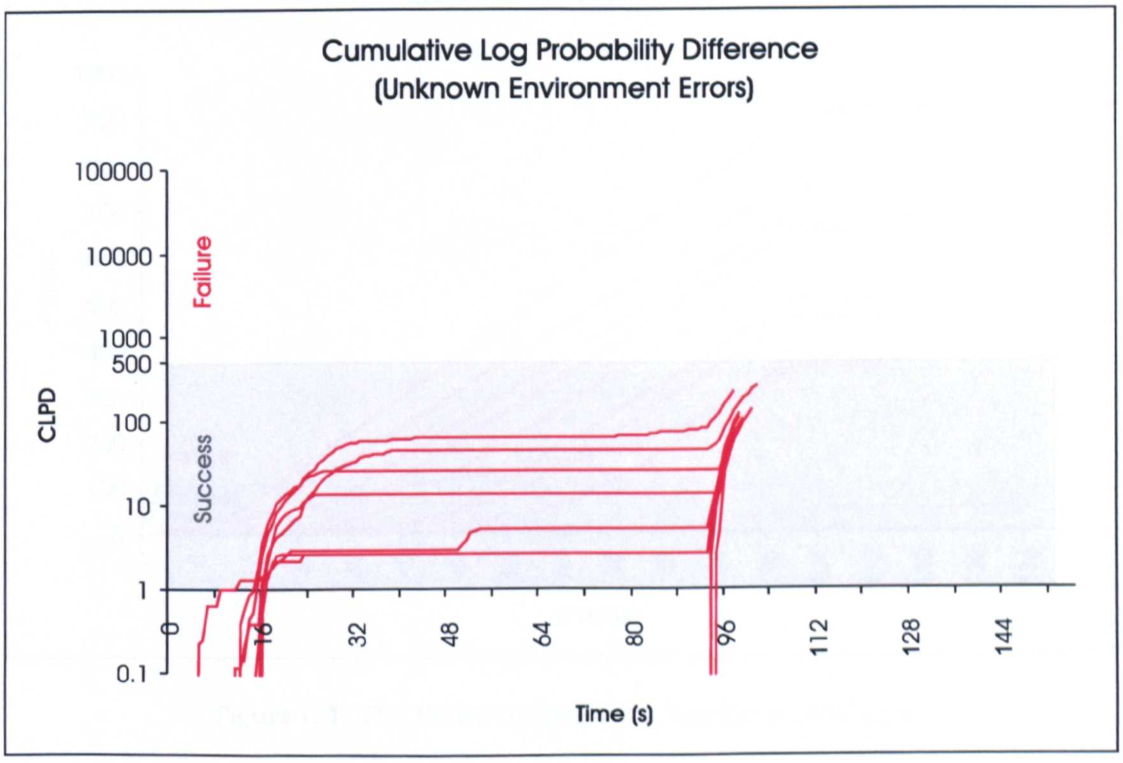


Figure B.5: *The CLPD values for the 'Unknown Environment' data*

APPENDIX C

TSCEM Graphs for Error Executions

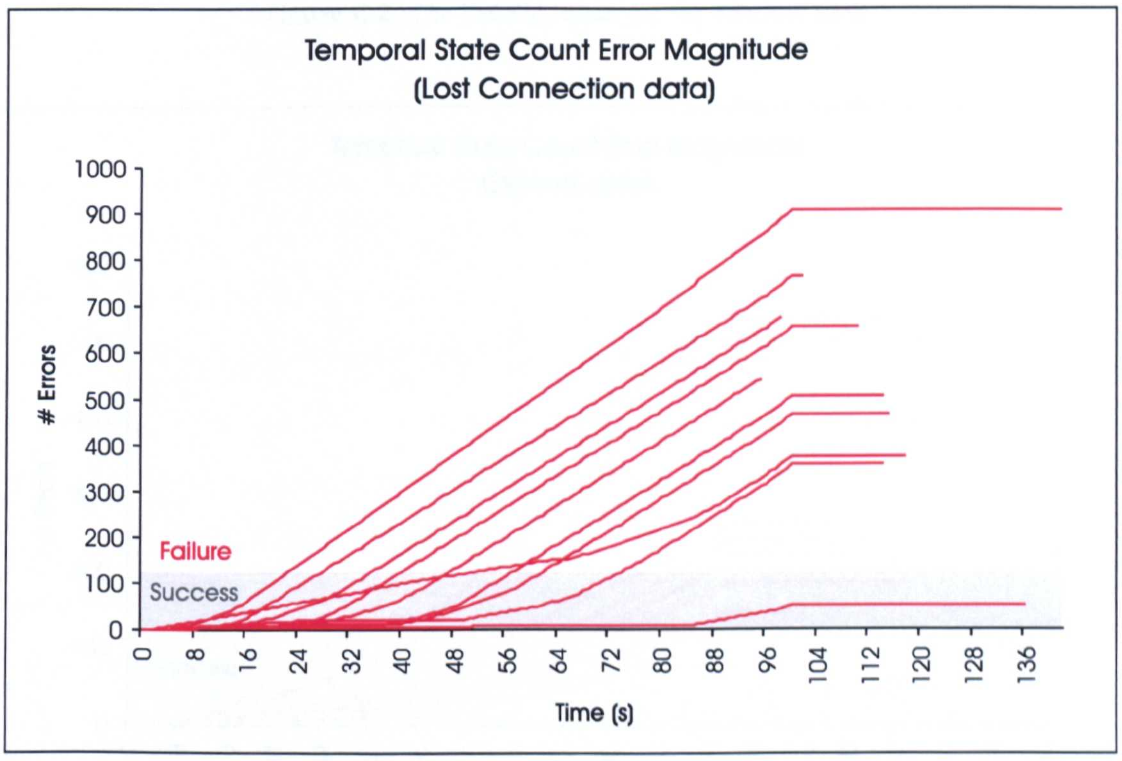


Figure C.1: *The TSCEM values for the 'Lost Connection' data*

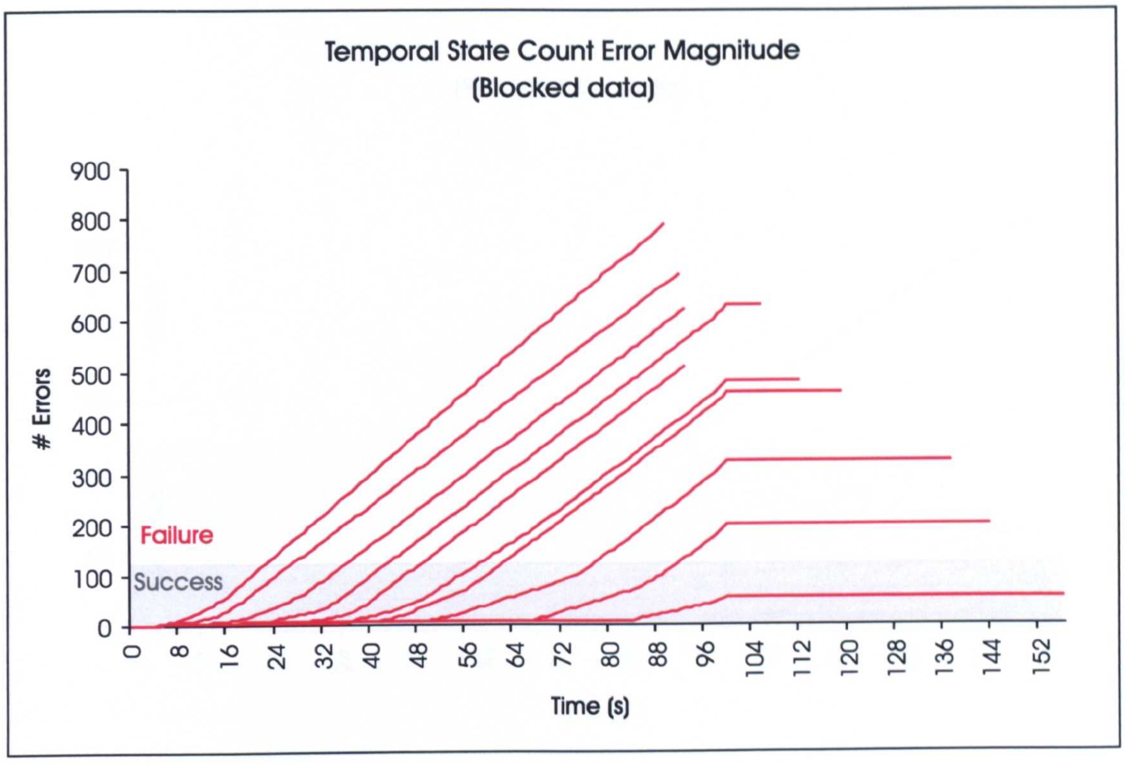


Figure C.2: *The TSCEM values for the 'Blocked' data*

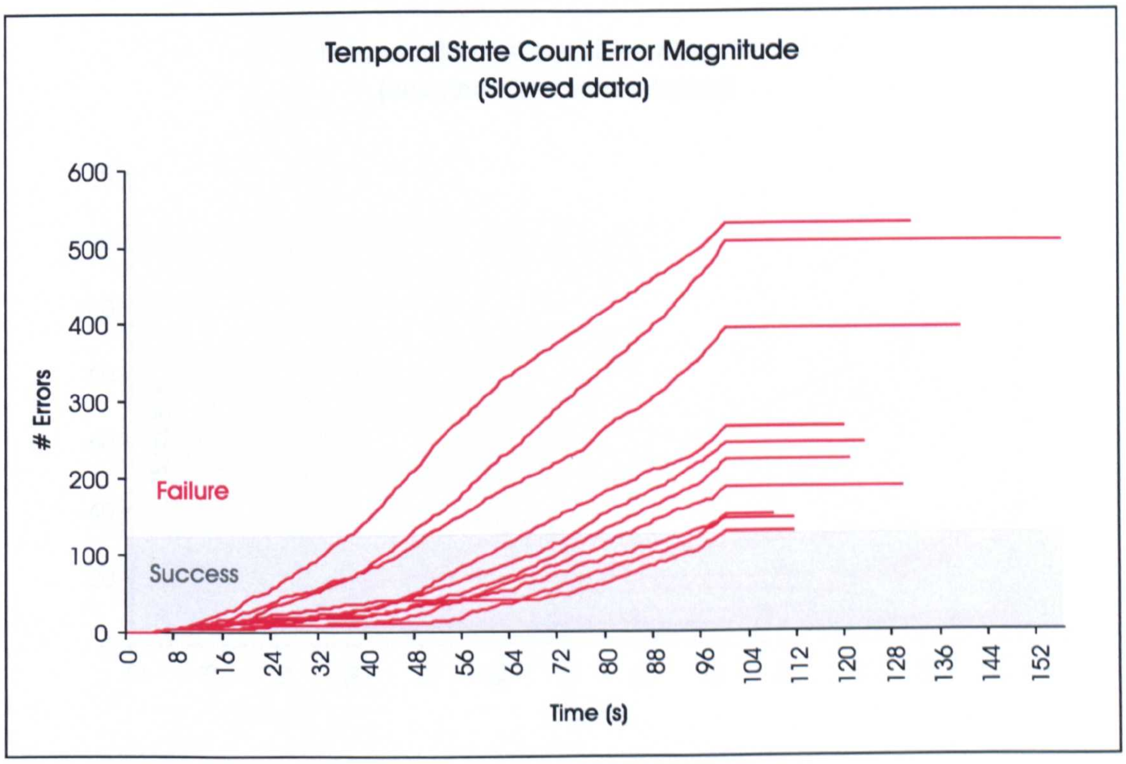


Figure C.3: *The TSCEM values for the 'Slowed' data*

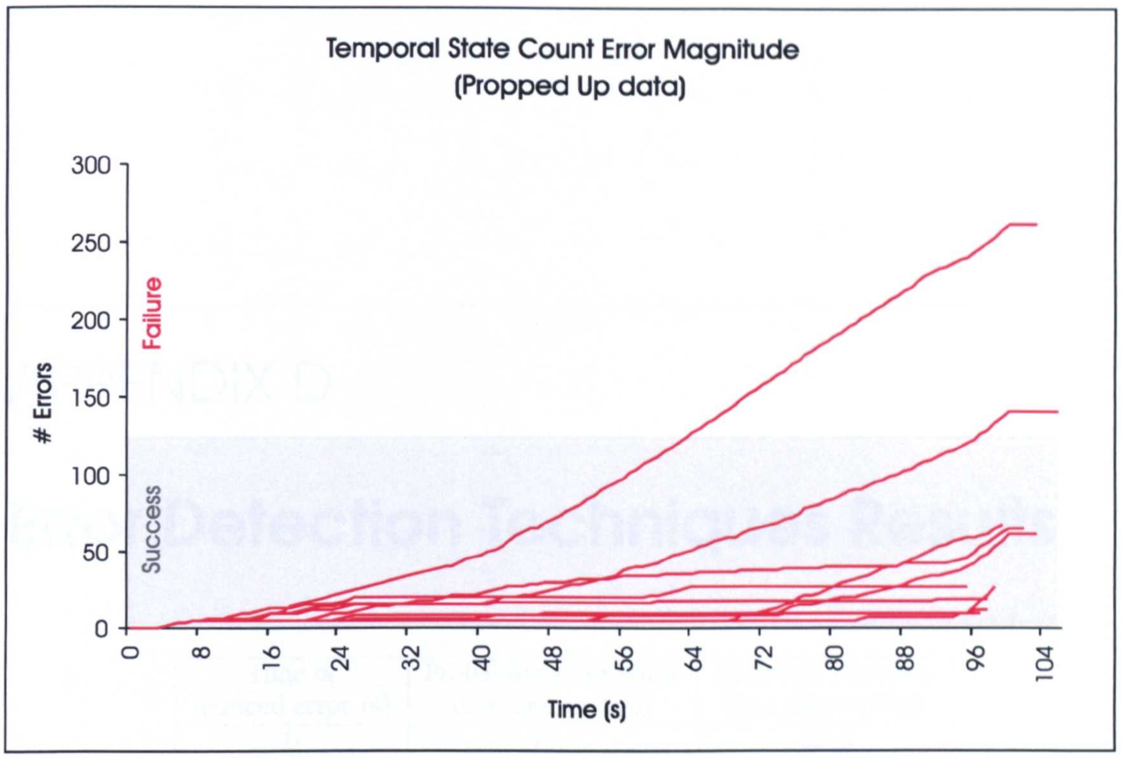


Figure C.4: *The TSCEM values for the 'Propped Up' data*

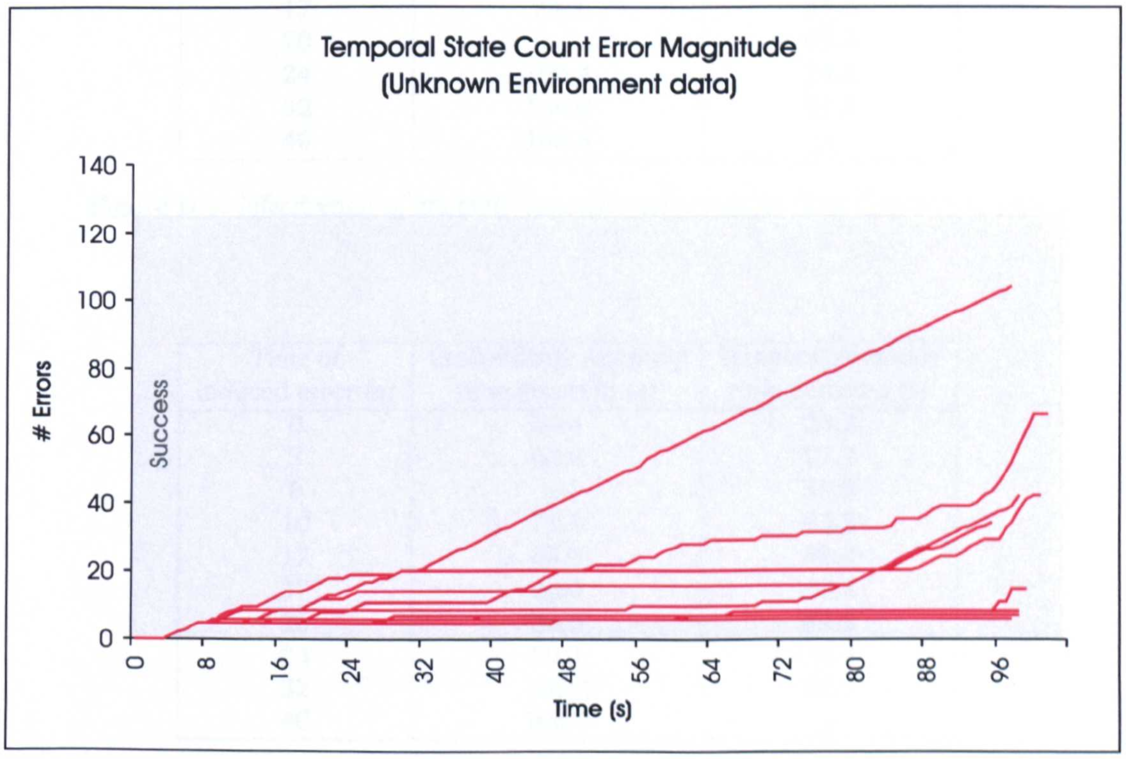


Figure C.5: *The TSCEM values for the 'Unknown Environment' data*

APPENDIX D

Error Detection Techniques Results

Time of induced error (s)	Probabilistic Anomaly time detected (s)	Temporal Anomaly time detected (s)
0	30.4	23.2
3	53.6	28.0
6	—	36.0
10	64.0	40.0
12	—	48.0
17	78.4	57.6
20	—	62.4
24	106.4	74.4
32	104.0	51.2
40	105.5	—

Figure D.1: *Effectiveness of anomaly detection techniques for 'Lost Connection' data*

Time of induced error (s)	Probabilistic Anomaly time detected (s)	Temporal Anomaly time detected (s)
0	30.4	23.2
3	64.8	27.2
6	—	36.8
10	71.2	43.2
12	68.0	48.8
17	80.0	59.2
20	80.0	63.2
24	99.2	78.4
32	102.4	92.0
40	106.4	—

Figure D.2: *Effectiveness of anomaly detection techniques for 'Blocked' data*

Time of induced error (s)	Probabilistic Anomaly time detected (s)	Temporal Anomaly time detected (s)
0	75.2	37.6
3	82.4	50.4
6	70.4	47.2
10	111.2	66.4
12	94.4	75.2
17	106.4	78.4
20	112.0	84.0
24	—	96.0
32	—	100.0
40	—	93.6

Figure D.3: *Effectiveness of anomaly detection techniques for 'Slowed' data*

Time of induced error (s)	Probabilistic Anomaly time detected (s)	Temporal Anomaly time detected (s)
0	—	64.0
3	—	—
6	—	96.8
10	—	—
12	—	—
17	—	—
20	—	—
24	—	—
32	—	—
40	96.8	—

Figure D.4: *Effectiveness of anomaly detection techniques for 'Propped Up' data*

Time of induced error (s)	Probabilistic Anomaly time detected (s)	Temporal Anomaly time detected (s)
0	—	—
3	—	—
6	—	—
10	—	—
12	—	—
17	—	—
20	—	—
24	—	—
32	—	—
40	—	—

Figure D.5: *Effectiveness of anomaly detection techniques for 'Unknown Environment' data*

Bibliography

[Blackburn et al., 2002] Blackburn, M. R., Busser, R., Nauman, A., Knickerbocker, R., and Kasuda, R. (2002). Mars Polar Lander fault identification using model-based testing. In *ICECCS*.

[Bonasso et al., 1997] Bonasso, R. P., Firby, J., Gat, E., Kortenkamp, D., Miller, D. P., and Slack, M. G. (1997). Experiences with an architecture for intelligent, reactive agents. In *Journal of Experimental & Theoretical Artificial Intelligence*, pages 237–256.

[Boyan and Littman, 2000] Boyan, J. A. and Littman, M. L. (2000). Exact solutions to time-dependant MDPs. In *Advances in Neural Information Processing Systems (NIPS) 2000*, pages 1026–1032. MIT Press.

[Bresina et al., 2002] Bresina, J., Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D., and Washington, R. (2002). Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI-02)*.

[Bresina and Washington, 2000] Bresina, J. and Washington, R. (2000). Expected utility distributions for flexible, contingent execution. In *Proceedings of the AAAI-2000 Workshop: Representation Issues for Real-World Planning Systems*.

[Buffett and Spencer, 2003] Buffett, S. and Spencer, B. (2003). Efficient monte carlo decision tree solution in dynamic purchasing environments. In *Proceedings of the 5th international conference on Electronic commerce*, pages 31–39. ACM Press.

[Bui et al., 2002] Bui, H., Venkatesh, S., and West, G. (2002). Policy recognition in the abstract hidden markov models. *Journal of Artificial Intelligence Research*, 17:451–499.

[Coddington et al., 2005] Coddington, A., Fox, M., Gough, J., Long, D., and Serina, I. (2005). MADbot: A motivated and goal directed robot. In *Proceedings of the*

Twentieth American National Conference on Artificial Intelligence (AAAI-05) — Demo Session.

[Coddington and Luck, 2004] Coddington, A. M. and Luck, M. (2004). A motivation-based planning and execution framework. *International Journal on Artificial Intelligence Tools*, 13(1):5–25.

[Dearden et al., 2003] Dearden, R., Meuleau, N., Ramakrishnan, S., Smith, D. E., and Washington, R. (2003). Incremental contingency planning. In *ICAPS-2003 Workshop on Planning under Uncertainty and Incomplete Information*.

[Dellaert et al., 1999] Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA99)*.

[Demeester et al., 2003] Demeester, E., Nuttin, M., Vanhooydonck, D., and Brussel, H. V. (2003). A model-based, probabilistic framework for plan recognition in shared wheelchair control: Experiments and evaluation. In *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*.

[Dijkstra, 1976] Dijkstra, E. W. (1976). *A Discipline of Programming*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Dix et al., 1990] Dix, J., Posegga, J., and Schmitt, P. (1990). Modal logic for AI planning.

[Ellson et al., 2004] Ellson, J., Gansner, E., Koren, Y., Koutsofios, E., Mocenigo, J., and North, S. (2004). Graphviz 2.6 graph visualization software. available from `http://www.graphviz.org/`.

[Epstein, 2003] Epstein, L. (2003). On variable sized vector packing. *Acta Cybern.*, 16(1):47–56.

[Feng, 2004] Feng, Z. (2004). Towards better scalability in solving MDPs and POMDPs. In *ICAPS 2004 Doctoral Consortium*.

[Firby, 1989] Firby, R. J. (1989). *Adaptive Execution in Complex Dynamic Domains*. PhD thesis, Yale University.

[Fournier and Crémilleux, 2002] Fournier, D. and Crémilleux, B. (2002). A quality index for decision tree pruning.

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

[Fox and Long, 2002] Fox, M. and Long, D. (2002). Single-trajectory opportunistic planning under uncertainty. In *UK Planning SIG, Delft*.

[Fox and Long, 2003] Fox, M. and Long, D. (2003). PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of AI Research*, 20:61–124.

[Gat, 1992] Gat, E. (1992). Integrating planning and reaction in a heterogeneous asynchronous architecture for controlling mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI)*.

[Gat, 1998] Gat, E. (March 1998). On three-layer architectures. In *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press.

[Gerevini and Serina, 2002] Gerevini, A. and Serina, I. (2002). Lpg: A planner based on local search for planning graphs with action costs. In *AIPS*, pages 13–22.

[Givan and Parr, 2001] Givan, B. and Parr, R. (2001). An introduction to markov decision processes. Talk given at Dagstuhl, available online at `http://www.ece.purdue.edu/~givan/talks/mdp-tutorial.pdf`.

[Gregory, 2001] Gregory, I. P. (2001). A comparative review of robot programming languages.

[Hoare, 1983] Hoare, C. A. R. (1983). An axiomatic basis for computer programming. *Commun. ACM*, 26(1):53–56.

[Horstmann and Zilberstein, 2003] Horstmann, M. and Zilberstein, S. (2003). Automated generation of understandable contingency plans. 13th International Conference on Automatic Planning & Scheduling (ICAPS).

[Ingham et al., 2001] Ingham, M., Ragno, R., and Williams, B. (2001). A reactive model-based programming language for robotic space explorers. In *Proceedings of ISAIRAS–01*.

[Joslin et al., 2005] Joslin, D., Frank, J., Jónsson, A. K., and Smith, D. E. (2005). Simulation-based planning for planetary rover experiments. In *Winter Simulation Conference*.

[Kohonen, 1990] Kohonen, T. (1990). Self-organized formation of topologically correct feature maps. In Shavlik, J. W. and Dietterich, T. G., editors, *Readings in Machine Learning*, pages 326–336. Kaufmann, San Mateo, CA.

[Kosaka and Kak, 1992] Kosaka, A. and Kak, A. C. (1992). Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *CVGIP: Image Underst.*, 56(3):271–329.

[Lane and Kaelbling, 2002] Lane, T. and Kaelbling, L. P. (2002). Nearly deterministic abstractions of markov decision processes. In *Eighteenth national conference on Artificial intelligence*, pages 260–266, Menlo Park, CA, USA. American Association for Artificial Intelligence.

[Liao et al., 2004] Liao, L., Fox, D., and Kautz, H. (2004). Learning and inferring transportation routes. In *Proceedings of the 19th National Conference on AI (AAAI-04)*, San José, CA.

[MacQueen, 1967] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, 1:281–296.

[Matijevic, 1996] Matijevic, J. (1996). Mars pathfinder microrover — implementing a low cost planetary mission experiment.

[Mausam et al., 2005] Mausam, Benezara, E., Brafman, R. I., Meuleau, N., and Hansen, E. (2005). Planning with continuous resources in stochastic domains. In *Proceedings of IJCAI'05*.

[Muscettola et al., 1998] Muscettola, N., Nayak, P., Pell, B., and Williams, B. (1998). Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47.

[NASA, 1999] NASA (1999). Mars Polar Lander Press Kit.

[NASA, 2000] NASA (2000). Report on the loss of the Mars Polar Lander and deep space 2 missions.

[Pedersen et al., 2005] Pedersen, L., Smith, D., Deans, M., Sargent, R., Kunz, C., Lees, D., and Rajagopalan, S. (2005). Mission planning and target tracking for autonomous instrument placement. In *Proceedings of the 2005 IEEE Aerospace Conference, Big Sky, Montana*.

[Pell et al., 1996] Pell, B., Gat, E., Keesing, R., Muscoletta, N., and Smith, B. (1996). Plan execution for autonomous spacecraft. In *Plan Execution: Problems and Issues: Papers from the 1996 AAAI Fall Symposium*, pages 109–116. AAAI Press, Menlo Park, California.

[Pettersson, 1997] Pettersson, L. (1997). Real-time scheduling using fuzzy techniques.

[Russell and Norvig, 1995] Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*, pages 762–766. Prentice Hall, first edition.

[Schaffer et al., 2005] Schaffer, S., Clement, B., and Chien, S. (2005). Probabilistic reasoning for plan robustness. In *19th International Joint Conference on Artificial Intelligence*.

[Simmons and Apfelbaum, 1998] Simmons, R. and Apfelbaum, D. (1998). A task description language for robot control.

[Tindell et al., 1992] Tindell, K., Burns, A., and Wellings, A. J. (1992). Mode changes in priority pre-emptively scheduled systems. In *IEEE Real-Time Systems Symposium*, pages 100–109.

[Volpe, 2003] Volpe, R. (2003). Rover functional autonomy development for the mars mobile science laboratory. In *Proceedings of the 2003 IEEE Aerospace Conference, Big Sky, Montana.*

[Volpe, 2005] Volpe, R. (2005). Rover technology development and mission infusion beyond MER,. In *Proceedings of the 2005 IEEE Aerospace Conference, Big Sky, Montana.*

[Volpe et al., 2001] Volpe, R., Nesnas, I., Estlin, T., Mutz, D., Petras, R., and Das, H. (2001). The claraty architecture for robotic autonomy.

[Williams and Gupta, 1999] Williams, B. and Gupta, V. (1999). Unifying model-based and reactive programming within a model-based executive. Workshop on Principles of Diagnosis.

[Williams and Nayak, 1999] Williams, B. C. and Nayak, P. (1999). A model-based approach to reactive self-configuring systems. In Minker, J., editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland. Computer Science Department, University of Maryland.

Thank you Derek Long and Maria Fox for your supervision, perseverance, much needed nagging and above all, faith in my work.

Many thanks also go to my family, as well as my colleagues and friends, Andrew, Amanda, Pete, Ivan, Sam R., Marion, Sam G., Naomi, Camilla, Jen, the extremely helpful folks at Typophile and the countless others who supported me through difficult times.