REAL TIME FURNACE FROTH STATE DETECTION USING HIDDEN MARKOV MODELS

William Gordon Harker

A dissertation submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, 2013

DECLARATION

I declare that this dissertation is my own unaided work. It is being submitted to the Degree of Master of Science in Engineering to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other University.

.....

(William Gordon Harker)

.....day of.....year.....

ABSTRACT

In this dissertation the feasibility of developing a soft sensor utilising Hidden Markov Models (HMM) was evaluated. Specifically, this methodology was tested for use as a soft sensor to detect furnace froths in a real time environment.

Initially, a review of Hidden Markov Models was undertaken to gain an understanding of the mathematics and algorithms associated with HMM's. A simple HMM example was constructed to highlight practical problems associated with HMM's. One such problem identified was that HMM's are unsuitable for real time use without modification.

Potential modifications were then researched to improve the real time performance of the HMM. This research yielded a real time variant of the HMM Viterbi algorithm, labelled Real Time Viterbi (RTV), as a potential modification. In addition a new hybrid algorithm, labelled the Hidden Markov Model Fixed State Test (HMM FST), was developed by the Author.
Comparative studies of the respective real time performances of the RTV and HMM FST algorithms concluded that the HMM FST algorithm was the most suitable for use in the real world application.

A final HMM FST real time algorithm was developed which incorporated the use of K-Means Clustering techniques. Data files, consisting of electrode positions from real furnace froths, were then replayed into the HMM FST algorithm to evaluate its performance. Four scenarios, incorporating different HMM FST tuning parameters, were then executed to determine the impact of the model parameters on its froth detection ability and false positive response. A final tuning set was recommended for the HMM FST Furnace Froth Detector.

This research proved that this approach can be used as a practical soft sensor to detect furnace froths in electric arc furnaces with any structural or electrode configuration. The HMM FST model could be tuned to various levels of sensitivity and was found to generate low false positives due to its treatment of plant sensors as a collective.

CONTENTS

DECLARATION 2
ABSTRACT 3
CONTENTS 4
LIST OF FIGURES 7
LIST OF TABLES 8
LIST OF SYMBOLS 9
CHAPTER 1 INTRODUCTION 10
1.1 General Introduction 10
1.1.1 Soft Sensors 10
1.1.2 The Difficulty With Froth Detection 11
1.2 Previous Research 11
1.2.1 The Origins of Hidden Markov Models 11
1.2.2 Applications of Hidden Markov Models 12
1.3 Purpose of Research 14
1.4 Aims and Objectives 14
1.5 Limitations of Research 15
1.6 Research Methodology 15
1.7 Dissertation Overview 17
CHAPTER 2 HIDDEN MARKOV MODEL THEORY 18
2.1 Basic Probability Theory Review 18
2.2 HMM Notation 21
2.2.1 Row Stochastic Matrices (A, B and π) 22
2.2.2 HMM Visualisation 22
2.3 Three HMM Problems 24
2.3.1 Solution to Problem 1 (Forward Procedure) 25
2.3.2 Solution to Problem 2 (Viterbi Algorithm) 31
2.3.3 Solution to Problem 3 (Baum Welch Algorithm) 36
2.4 Summary 43
CHAPTER 3 PRACTICAL HMM EXAMPLE 44
3.1 Formulating the Model 44
3.1.1 Practical HMM Design Guidelines 44
3.1.2 The Emission Probability Matrix (B) for the Loaded Dice Example 45
3.1.3 The State Transition Matrix (A) for the Loaded Dice Example 47

3.1.4 The Initial State Distribution (π) for the Loaded Dice Example 48
3.2 Executing the Model 49
3.2.1 Generating Fair and Loaded Dice Throws 50
3.2.2 Model Execution Methodology 51
3.2.3 Model Execution Example 52
3.3 Summary 53
CHAPTER 4 REAL TIME HMM STATE DETECTION 54
4.1 An Introduction to Real Time HMM Detection 55
4.1.1 Fixed State Test (FST) Algorithm 56
4.1.2 Modified Real Time Viterbi (RTV) Algorithm 57
4.2 Qualitative Real Time Algorithm Performance 60
4.2.1 Scenario 1 - The HMM Model Output for a Buffer Length of 10 62
4.2.2 Scenario 2 - The HMM Model Output for a Buffer Length of 5 63
4.2.3 Scenario 3 - The HMM Model Output for a Buffer Length of 8 64
4.3 Quantitative Real Time Algorithm Performance 65
4.3.1 Real Time Model Performance Results 66
4.4 Summary 69
CHAPTER 5 HMM FURNACE FROTH DETECTION 70
5.1 Process Description 70
5.1.1 Furnace Power Control and Electrode Positions 72
5.2 Vector Quantisation 74
5.2.1 Introduction to Vector Quantisation 74
5.2.2 Real Time Considerations for Vector Quantisation 76
5.2.3 A Vector Quantisation Example 78
5.3 A Note on False Positives 81
5.4 Design Of The HMM Froth Detector Algorithm 82
5.4.1 Stage (1) and (2) Codeword Initialisation 83
5.4.2 Stage (3) K-Means Clustering Algorithm Execution 85
5.4.3 Stage (4) Sorting and Storing the Codebook 86
5.4.4 Stage (5) Construction Of The HMM Matrices 88
5.4.5 Stage (6) FST Buffer Length 92
5.4.6 Stage (7) Reading in Stored Data in Real Time 93
5.4.7 Stage (8) and (9) Creating HMM Observations 95
5.4.8 Stage (10) Executing the HMM Fixed State Test (FST) Algorithm 95
5.4.9 Stage (11) Capturing Results 97
5.5 Summary 101
CHAPTER 6 HMM FST FROTH DETECTOR RESULTS 102

6.1 Qualitative Results 102
6.2 Quantitative Results 107
6.3 Tuning the Response of the HMM FST Algorithm 111
6.3.1 Scenario 1 - The Effect of Codeword (Observation) Choice on Response 113
6.3.2 Scenario 2 - The Effect of Buffer Length on Response 115
6.3.3 Scenario 3 - The Effect of Emission Matrix Parameters on Response 116
6.3.4 Scenario Results Summary 118
6.4 Evaluating the HMM FST Detector's False Positive Response 121
6.4.1 Estimating The False Positive Rate With Scenario 2 Parameters ... 124
6.4.2 Estimating The False Positive Rate With Scenario 3 Parameters ... 127
6.4.3 False Positive Test Summary 128
6.5 Summary 129
CHAPTER 7 CONCLUSIONS 130
7.1 The Research Question Answered 131
7.2 HMM FST False Positive Rejection Properties 131
7.3 Secondary Conclusions 133
7.4 Future Work 134
REFERENCES 136
APPENDIX A 139
APPENDIX B 140
APPENDIX C 141
APPENDIX D 143
APPENDIX E 152

LIST OF FIGURES

Figure 2.1 General Structure of a Hidden Markov Model as described by Stamp (2004: 3) 23
Figure 2.2 Visualisation of a Forward Variable at time t and state i 28
Figure 2.3 HMM Lattice Visualisation 31
Figure 2.4 Baum Welch Algorithm Output for Fair/Loaded Dice Example 41
Figure 3.1 An extract of G2 code 49
Figure 4.1 Scenario 1 (buffer length of 10) 62
Figure 4.2 Scenario 2 (buffer length of 5) 63
Figure 4.3 Scenario 3 (buffer length of 8) 64
Figure 5.1 A Froth Detection with Successful Automatic Suppression 71
Figure 5.2 A Froth Detection with a Partially Successful Automatic Suppression 72
Figure 5.3 VQ Example With N = 4 79
Figure 5.4 VQ Example With N = 10 79
Figure 5.5 VQ and HMM Algorithm Flow Charts 82
Figure 5.6 Possible State Transitions of the Froth Detector HMM 89
Figure 5.7 Real Time Output of the Froth Detector HMM 97
Figure 6.1 An Example of a HMM FST Froth Detection 103
Figure 6.2 An Example of a HMM FST Froth Detection 103
Figure 6.3 An Example of a HMM FST Froth Detection 104
Figure 6.4 An Example of a HMM FST Froth Detection 105
Figure 6.5 A Histogram of Froth Events versus Delay Intervals for Each Scenario 119
Figure 6.6 An Example of a False Positive Detection 122
Figure 6.7 A Second Example of a False Positive Detection 123
Figure 6.8 An Example of a Replayed False Positive Data File with Scenario 2 Parameters 124
Figure 6.9 A Second Example of a Replayed False Positive Data File with Scenario 2 Parameters 125
Figure 6.10 An Example of a Replayed False Positive Data File with Scenario 3 Parameters 127
Figure B1 An Octave Script (hmm.m) Executing a Specific HMM model 140
Figure B2 The Octave Output of the Script (1 = Fair State, 2 = Loaded State) 140

LIST OF TABLES

Table 3.1 Probabilities for Fair and Loaded Dice 46
Table 4.1 Monte Carlo Results for Loaded Dice Simulation for FST Algorithm 67
Table 4.2 Monte Carlo Results for Loaded Dice Simulation for RTV Algorithm 68
Table 5.1 Quantitative Output of the Froth Detector HMM 98
Table 6.1 HMM Codebook Generated from K-Means Algorithm 108
Table 6.2 Results of First 10 Data Files Replayed into HMM FST Algorithm 109
Table 6.3 Manually Modified Codebook 113
Table 6.4 Symbolic Summary of Scenario Results 119
Table 6.5 HMM FST False Positive Test Results (Scenario 2 Parameters) 125
Table 6.6 HMM FST False Positive Test Results (Scenario 3 Parameters) 128
Table 7.1 A Summary of Results Per Scenario 130
Table A1 Results of Fair and Loaded Dice Algorithm Test 139
Table D1 Original Scenario Results of HMM FST Algorithm Test with K-Means Generated Codebook and Buffer Length = 10 143
Table D2 Scenario 1 Results of HMM FST Algorithm Test with Manual Codebook and Buffer Length = 10 145
Table D3 Scenario 2 Results of HMM FST Algorithm Test with Manual Codebook and Reduced Buffer Length = 5 147
Table D4 Scenario 3 Results of HMM FST Algorithm Test with Original Codebook, Buffer Length = 10 and Modified B Matrix 149

LIST OF SYMBOLS

Hidden Markov Model	HMM
Fixed State Test	FST
Real Time Viterbi	RTV
A Hidden Markov Model	λ
HMM State Transition Matrix	A
HMM Emission Probability Matrix	B
HMM Initial State Distribution Matrix	π
Buffer Length	L
The Set of Possible Observations	V
Observation at time t	O_t
Observation Sequence	O
The Set of Possible Hidden States	X
Hidden State at time t	q_t
Hidden State Sequence	Q
Forward Variable	α
Partial Probability	δ
Viterbi Algorithm Back Pointer	Φ
Baum-Welch Xi Variable	ξ
Baum-Welch Gamma Variable	γ
Discrete Time Point	t
Megawatts	MW
Kilo amperes	kA
Millimetres	mm

CHAPTER 1 INTRODUCTION

1.1 General Introduction

1.1.1 Soft Sensors

In the discipline of process control, there is a field of study that is focused with developing plant models that aim to describe or explain plant behaviour. While the instrumentation field is a natural complement to process control there is an aspect of plant control that requires more than basic instrumentation to execute its objectives. This field of process control is concerned with the development of soft sensors that utilise software (as well as plant instrumentation) to provide either a new process variable (where an instrument cannot do the job) or to provide higher level information on the state of the plant. A simple example of a soft sensor that provides a new process variable is one of a water tank (eg a header tank). Often a pressure transmitter is utilised to translate water pressure (in kPa) into level (in metres) using the equation: Pressure = density * gravity * height where density and gravity are constants and the height of the water is derived from the pressure transmitter.

As mentioned above, the second output of a soft sensor is one of a plant state. The concept of a state is one that is well known in the process control industry. Alarming on plant variables (eg a high pressure) is a practice that transforms a process variable (usually fed from a plant instrument) into an alarm state (eg high, low, ok) - this usually happens in the Operator Interface System.

In this dissertation, the output of the soft sensor is a plant state. The soft sensor is required to detect whether a furnace is frothing based upon the movement of the electrodes. There is no instrument that can directly measure whether a furnace is in the froth state. As such, software is required to infer this state from other process variables - in this case these are electrode positions.

1.1.2 The Difficulty With Froth Detection

In most furnaces a froth results in a rapid expansion of the volume of the slag bath (containing metal oxides). Once a froth has started, some intervention is required to the furnace operation to suppress this froth. If left untreated, the froth should result in some form of extreme intervention (from either the plant operator or the plant PLC [Programmable Logic Controller]) such as the cutting of power to the furnace. In extreme cases a froth could result in molten metal being deposited into the furnace scrubbers which could cause an explosion.

This research is thus aimed at detecting the froth event before it is too late. The difficulty with this type of soft sensor is the trade off between early detection and flagging a false positive. A secondary problem, that this research attempts to solve, is the issue of minimising the false positive ratio of the detector while still flagging the froth in enough time for the suppression system to take effect. Depending on the furnace this window is in the order of 5 to 10 minutes.

1.2 Previous Research

1.2.1 The Origins of Hidden Markov Models

According to Rabiner (1989: 257-9) the basic theory of Hidden Markov Models was published by Baum and his colleagues [refer to Baum and Petrie (1966: 1554-1563) for further information] in the late 1960's. This work built upon the research undertaken by Andrei Andreyevich Markov in the early 1900's who pioneered the statistical process known as Markov Chains. In a first order Markov Chain the future variable is only probabilistically dependent on the current variable. Applying this rule over a number of repetitions therefore creates a memoryless chain of variables. The implication of a memory less system is that a sequence of hidden states can be identified or re-created by only keeping the previous state thus minimising data storage.

From Forney (1973: 268-278) the Viterbi algorithm, another major algorithm associated with HMM's, was developed by Andrew J. Viterbi in 1967 as a maximum likelihood decision algorithm for any symbolic sequence that can be modelled as a

Markov process. Today the Viterbi algorithm is an international standard in cellular data decoding used to compress data over noisy channels - Viterbi (2006: 142).
Hidden Markov Models were implemented in speech processing applications in the 1970's but today, due to their robust mathematical structure, are used in varied applications. In this dissertation, the application of Hidden Markov Models is applied to the field of Process Control.

1.2.2 Applications of Hidden Markov Models

As introduced by Rabiner (1989: 257) above, one significant application of Hidden Markov Models is that of speech recognition. In this case the speech of a person is configured as the HMM observation set via the use of a codebook. The concept of using a codebook with codewords (in the furnace froth detector) was partly inspired by the above mentioned work. HMM's are also heavily utilised in the communications field. As introduced above in Viterbi (2006: 120) the developer, Andrew Viterbi, was heavily involved in their implementation in CDMA (Code Division Multiple Access) communication technology which is used in mobile phones. CDMA employs spread spectrum technology and utilises Viterbi decoding of transmitted data. The Viterbi algorithm is one of the main algorithms associated with the use of HMM's - it identifies the most probable sequence of hidden states. The limitation with its original form is with its applicability to a real time environment. This algorithm as well as its hybrids was thus evaluated in this dissertation for its suitability in the real time froth detector algorithm.

Campos (2010: 81) introduced a Bioinformatics HMM application that utilised Hidden Markov Models to identify viral genomes in DNA sequences which indicates the diversity of HMM applications.

Bruckner, Sallans and Russ (2007: 989) presented work that utilised HMM's in discriminating traffic behaviour in tunnels. In their model, multiple cameras detected vehicle movements which then utilised HMM's to identify traffic states (eg a traffic jam). This work introduced the concept of a "model of normality". Their algorithm initiates alarms for situations that deviate from the "normal" state. This approach was originally considered for the Furnace Froth Detector but was later discarded as detecting for normality versus detecting for an abnormal process state yielded no advantage.

Savage, Marquez and Lepe-casillas (2005: 1) utilised HMM's in an underwater robot position locator. A vector of sonar readings provided the HMM observations while the hidden states were configured as the actual robot position. The work presented here played a significant role in the ideas incorporated into the Froth Detector. The concept of packaging input sensors into a vector and then utilising vector quantization techniques to determine the closest codeword in a trained codebook contributed significantly to the Froth Detector algorithm. The idea of packaging electrode positions into a vector and generating a codebook of vector codewords was thus obtained from this work.

In spite of the varied applications of HMM's no articles directly using HMM's in real time, online, process control applications were found. A process control application of HMM's was presented in a paper by Tai, Ching and Chan (2006:1) who proposed a strategy for the detection of machine failure using Hidden Markov Models. In this application bottle filling machines represented the hidden states while bottle levels (arranged in a batch) represented the observations of the HMM. The problem with this approach was that the model was not developed for real time use and presented offline findings.

In addition to the above literature survey, additional reviews of prior work are included at the beginning of some of the chapters in this dissertation. The literature reviewed in these chapters focuses on work specific to the topic discussed in that chapter.

1.3 Purpose of Research

The purpose of this research dissertation is to determine whether Hidden Markov Models are a suitable tool to use in developing a soft sensor to detect the froth state in any furnace (providing the furnace's electrodes respond to the froth condition).
Previous research in the real time use of Hidden Markov Models is discussed in CHAPTER 4, REAL TIME HMM STATE DETECTION.

Formally, the research question can be written follows:

Can Hidden Markov Models be used, in a real time environment, to detect Furnace Froths?

1.4 Aims and Objectives

In order to answer the research question the following aim was defined:

The aim of this research is to develop a soft sensor that utilises Hidden Markov Models to detect furnace froths in a real time environment.

The following objectives were formulated in order to achieve the aim:

- To research the topic of Hidden Markov Models with specific attention to the three algorithms that are associated with the topic.
- To translate these algorithms into software for use in a basic example and ultimately in the final algorithm**.**
- To apply the above research to a practical example that can be used to demonstrate the practical use of Hidden Markov Models in a real time environment.
- The final objective was to develop the froth detector soft sensor and test its performance by replaying saved froth data through the algorithm

1.5 Limitations of Research

The limitations of this research were as follows:

- The algorithm could not be tested on the live plant. The main reason for this was the excessive engineering time required to deploy a robust algorithm suitable for even attended online use - the risk to the plant was deemed too great without the requisite testing and commissioning phases. As such prior plant data was obtained and replayed through the froth detector in real time. The data was replayed in such a manner as to accurately mimic the model's online response.
- This research is aimed at proving whether Hidden Markov Models can be utilised in a real time process control environment to detect the furnace froth state. Although the final algorithm developed may be ready for online testing, a fully developed algorithm exceeds the scope of this research.
- A useful benchmark that was used to measure the performance of the HMM algorithm was provided by an existing rule based fuzzy froth detector that has already been deployed on the plant. It is not the specific aim of this research to specifically improve upon the performance of the existing system.

1.6 Research Methodology

The following experimental technique was utilised in this research:

- Initially a number of HMM detection algorithms, suitable for real time deployment, were researched. The algorithms that showed potential were translated into software and tested on a simple detection application (a Fair/Loaded dice system). The best performing algorithm was then selected for use in the real world engineering application.
- The above mentioned (best performing) algorithm was then integrated into a Hidden Markov Model developed (in a software development environment called G2) to test real world data (data obtained from actual furnace froths). In addition to the standard HMM components, the model utilised a technique called K-Means clustering to obtain the HMM observations by identifying clusters of data. 100 data files (separate from the 100 data files used to test the model) of genuine froths were then utilised to train the K-Means

Clustering Algorithm to obtain codewords (centroids) for use in the HMM model.

- Thereafter another 100 data files of genuine froth events were selected for testing the model's detection capabilities. The data contained within these files was replayed in real time into the HMM model in such a way as to accurately mimic online plant events and conditions. An additional 30 files, containing false positives, were also selected specifically for testing the model's false positive rejection properties. True positive versus false positive data was thus separated to reduce the degrees of experimental freedom. The 130 data files, utilised to test the model's detection capabilities, were considered to contain sufficient variance in plant conditions to test the HMM Froth Detector.
- In the case of the 100 data files, 4 separate scenarios or models (the same model structure with a different set of tuning parameters is deemed a new model) were created to test the effect of the different tuning parameters on model performance and false positive creation. The results of these scenarios were analysed with basic statistics which included a histogram analysis. These statistics were used to create a performance indicator of each scenario that was used to determine the best performing tuning parameter set.
- In the case of the 30 files, the two top performing models (differing only by their tuning parameters) were utilised to specifically test the false positive rejection capabilities of the HMM model. Considering that the data in all 30 of these files represented previous false positives, the number of non-detections by the HMM model were considered as the performance indicator.
- The results obtained from the respective true positive versus false positive test regimes were used to quantify model performance and ultimately prove or disprove the research question.
- All of the software developed for this dissertation has been implemented in an Expert System called G2. G2 is a real time, online development environment created by the Gensym corporation. Many advanced control applications throughout the world have been developed and deployed onto the G2 platform which runs on most of the popular operating systems. There however is one reference to the Octave application. Octave was used to validate some of the HMM algorithms developed in G2 as it (Octave) contains a library of Hidden Markov Model functions.

1.7 Dissertation Overview

This dissertation discusses the steps undertaken to develop a soft sensor that detects the furnace froth state. Hidden Markov Models are employed as a framework for the soft sensor which aims to detect real time froth events on an electric arc furnace. Leading up to the development of the final algorithm utilised in the soft sensor, a number of investigative phases and test stages had to be completed. These stages and phases have been grouped into a logical progression of chapters. The overview of these chapters is as follows:

- CHAPTER 2, HIDDEN MARKOV MODEL THEORY, contains a review of the relevant Hidden Markov Model theory that pays specific attention to the three algorithms that are associated with the topic. The mathematical nature of the HMM is discussed in this chapter along with the basic probability theory required to understand this HMM theory.
- CHAPTER 3, PRACTICAL HMM EXAMPLE, deals with a specific application of Hidden Markov Models. A basic example model was created to detect the change from Fair to Loaded dice in a dishonest casino. This chapter discusses the construction of the HMM model as well as its outputs as they apply to the Fair/Loaded dice model.
- CHAPTER 4, REAL TIME HMM STATE DETECTION, discusses the difficulties with using the HMM model, developed in CHAPTER 3, in a real time environment. Proposed modifications to the HMM algorithms are discussed and tested to determine the best algorithm to use in the Furnace Froth Detector.
- CHAPTER 5, HMM FURNACE FROTH DETECTION is dedicated to the design of the final HMM FST algorithm utilised in the Furnace Froth Detector. The structure of the final model is discussed along with each stage in the algorithm.
- CHAPTER 6, HMM FST FROTH DETECTOR RESULTS discusses the results of 4 different scenarios created to test the response space of the final algorithm.
- The above research and development is concluded in CHAPTER 7, CONCLUSIONS where the answer to the research question is discussed.

CHAPTER 2 HIDDEN MARKOV MODEL THEORY

In this chapter all the theory required to understand Hidden Markov Models is presented. A review of basic statistical probability theory is presented first and is followed by the derivations of the respective Hidden Markov Model algorithms (also known as the three HMM problems). Finally a practical test of the Baum Welch algorithm will be presented.

2.1 Basic Probability Theory Review

Since the topic of Hidden Markov Models is one that relies heavily on statistics, the following is a short review on the theorems and corollaries that hold true for event probabilities. This section provides a background as to the origins of some of the equations introduced in the following section discussing HMM theory.

From Walpole and Myers (1990: 27) Conditional Probability is the probability of some event A, given the occurrence of some other event B. Conditional probability is written:

$$P(A \mid B) \tag{1}$$

The intersection of two events is the probability of both events occurring. This probability is symbolically written as:

$$P(A \cap B) \tag{2}$$

The union of two events is the probability of one or the other event occurring. This probability is symbolically written as:

$$P(A \cup B) \tag{3}$$

From Walpole and Myers (1990: 27), Intersectional and conditional events are related as follows:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \quad (4)$$

From Walpole and Myers (1990: 30), if the two events A and B can both occur, then the Multiplicative Rule is applied to determine this conditional probability:

$$P(A \cap B) = P(A).P(B \mid A) \quad (5)$$

According to Walpole and Myers (32), if the two events A and B are independent (each event has no influence on the other), then:

$$P(A \cap B) = P(A).P(B) \quad (6)$$

For any two events, calculating the probability of one event or the other occurring is as follows - this theorem (the Addition Rule) is available from Walpole and Myers (22):

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \quad (7)$$

If events A and B are mutually exclusive (there is no chance that they can both occur ie P(A∩B) = 0) then **Equation (7)** is reduced to:

$$P(A \cup B) = P(A) + P(B) \quad (8)$$

Extending **Equation (6)** for the general case of mutually exclusive events the following statistical equation (one that is used widely in HMM theory) is obtained:

$$P(A_1 \cup A_2 \cup ... A_n) = P(A_1) + P(A_2) + ... + P(A_n) \quad (9)$$

Extending **Equation (8)** for the general case of independent events, yields another important statistical theorem in understanding HMM theory:

$$P(A_1 \cap A_2 \cap A_3 .. A_n) = P(A_1)\, P(A_2)\, P(A_3) ... P(A_n) \quad (10)$$

Equation (9) and **Equation (10)** are respectively available from Walpole and Myers (1990: 23) and Walpole and Myers (32).

A simple example of **Equation (9)** is with a single fair dice. The probability of rolling a particular number is = 1/6. The probability of rolling the numbers 1 or 2 in a single throw becomes P(rolling a 1) + P(rolling a 2) = 2/6. These are mutually exclusive events as two dice faces cannot occur in the same dice throw.

Applying the same dice throw example of **Equation (10)**, the probability of rolling the numbers 1 and 2 in two successive throws becomes P(rolling a 1)P(rolling a 2) = 1/36. These are independent events as one dice throw has no influence on the other.

In the following chapters (that deal with the derivation of the respective HMM algorithms) a sequence of events is considered. In particular a general sequence of observations $O = \{O_0, O_1, O_2 \dots O_{T-1}\}$ is considered for T discrete time intervals. This sequence is a chain of independent events (one observation has no influence on another) and as such, **Equation (10)** can be used to determine the probability of that Observation sequence O existing (by multiplying the respective probabilities of the individual events together). This concept is important for understanding the derivation of the Forward Algorithm presented in Section 2.3.1 and the Viterbi algorithm discussed in Section 2.3.2. **Equation (9)** is important for understanding the step in **Equation (16)** that sums over all possible state sequences to find $P(O| \lambda)$. The key point being that the existence of a specific state sequence (from any other state sequence) is a mutually exclusive event as only one state sequence can exist in an HMM model execution cycle.

2.2 HMM Notation

The most challenging aspect of Hidden Markov Models (HMM's) is the notation. One such challenge is to grasp the difference between the current state variable (or observation) at a point in time and a state (or observation) label. The following list of symbols should illustrate this difference:

λ = the symbol denoting the entire model. The compact model specification is usually denoted as $\lambda = (A, B, \pi)$.
T = the length of the observation sequence.
N = the number of states in the model.
M = the number of observation symbols defined within the HMM.
$X = \{x_0, x_1, x_2 \ldots x_{N-1}\}$ which refers to the set of N possible states defined within the HMM.
$V = \{v_0, v_1, v_2 \ldots v_{M-1}\}$ and refers to the set of M possible observations defined within the HMM where each element represents a discrete observation symbol.
$Q = \{q_0, q_1, q_2 \ldots q_{T-1}\}$ and refers to the sequence of hidden states that generates the observation sequence O. At any discrete time stamp between t = 0 to t = T - 1, q_t is an element of X.
$O = \{O_0, O_1, O_2 \ldots O_{T-1}\}$ and refers to an observation sequence where O_t is an element of V.
A = the state transition matrix where A is always N by N and a_{ij} denotes the probability of state i (at time t) transitioning to state j (at time t + 1)
B = the observation (or emission) probability matrix where B has N rows and M columns. The probability of the model emitting observation k at time t and being in state q_j is denoted as $b_j(k)$. The variable k is thus the index of the observation contained within V.
π = the initial state distribution vector which has N elements.

In general, the notation of Hidden Markov Models is such that t always refers to a time stamp and i, j or k always refer to state or observation indices. A distinction must thus be made between any hidden state existing at time t (q_t) and the particular index (x_i) of that state as it exists in X.

2.2.1 Row Stochastic Matrices (A, B and π)

Although the specific nature of these matrices will be discussed in detail later on, it is important to note that the matrices: A, B and π are all row stochastic. This means that their row probabilities must add up to 1.
In real world terms this means that in terms of the probability space, each state (with non zero probabilities) has a finite possibility of existing under the prevailing model conditions.

Consider the arbitrary stochastic vector (unrelated to any specific HMM matrix) of state probabilities of (S_1, S_2, S_3) = (0.2, 0.3, 0.5). In the real world implementation of this system, this means that each state has a finite probability of existing and that S_3, for example, should appear 2.5 more times than S_1 (on average). It also means that only these 3 states feature in the model and that at least one of them must exist at any point in time.

If for example the sum of these probabilities was less than 1, this would imply that there would be a chance of none of the 3 states existing at any point in time.

This property of each respective matrix provides a useful cross-check for the learning algorithms described in subsequent sections.

2.2.2 HMM Visualisation

Following from the section on notation, it is often useful to visualise a Hidden Markov Model. In the visualisation provided below (see **Figure 2.1**), it is important to note that in the execution of an HMM model, events unfold as a sequence of observations that infer a corresponding sequence of hidden states. The relationship between the observation and the corresponding hidden state is one to one. The actual hidden state that is output (for a particular observation) depends on the respective probabilities in the A, B and π matrices. Algebraically, an observation O_t at time t, infers the presence of the most probable state $q_t = x_i$ (also at time t) where x_i is an element of the possible states vector.

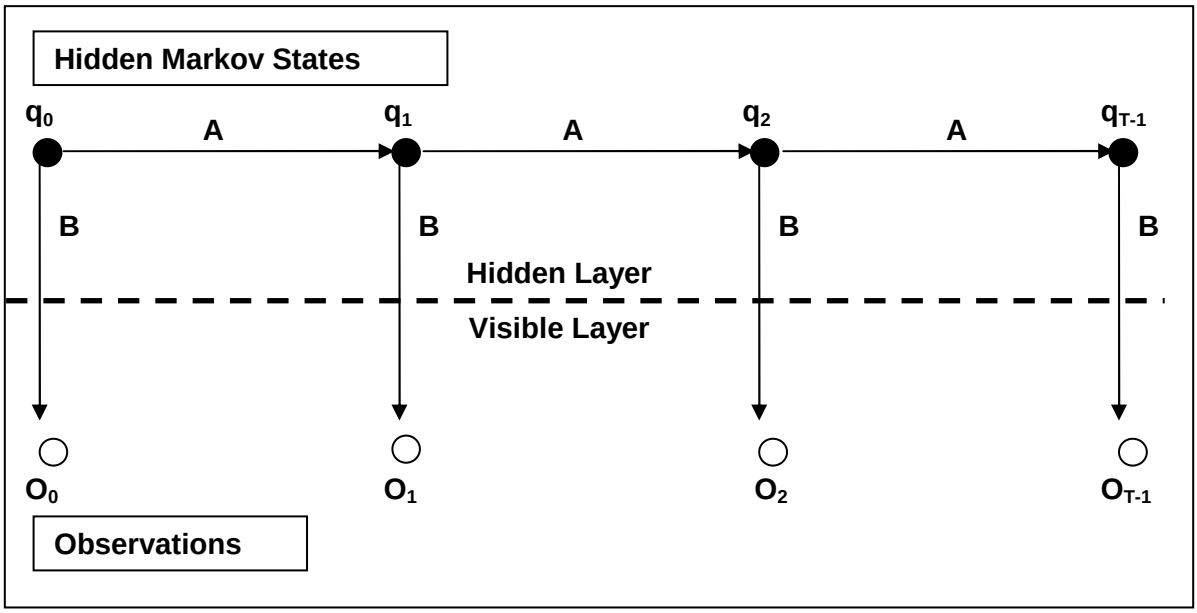

Figure 2.1 General Structure of a Hidden Markov Model as described by Stamp (2004: 3)

As can be observed in **Figure 2.1**, consecutive states (hidden layer) are dependent on the A (state transition) matrix. The observation that is emitted for a particular state is a function of the B (emission) matrix. The actual state pointed to by the first state in a state sequence: q_0 will be a function of the initial state distribution vector: π.

2.3 Three HMM Problems

When working with HMM's there are three common problems that need to be addressed. These are:

- **Problem 1**. Given the model λ = (A, B, π) and a sequence of observations O, find P(O| λ) This means that given a specific model, the first problem is how to find the probability of a specific observation sequence existing. This problem is also known as the Evaluation Problem.
- **Problem 2**. Given the model λ = (A, B, π) and a sequence of observations O, find an optimal state sequence for the underlying Markov process. The requirement is thus to find the sequence of hidden states that best explains or accounts for the observed sequence. This problem is also known as the Optimisation or Decoding Problem.
- **Problem 3**. Given an observation sequence O and the dimensions N (number of states in **λ**) and M (number of observations in λ), find the model λ = (A, B, π) that maximises the probability of O. This problem is also known as the Training Problem.

Each of the problems mentioned above, have dedicated algorithms that provide a solution. These solutions make intensive reference to Baye's theorem and in some cases are computationally intensive. The following sections describe the algorithm used to solve the respective problem. In the case of Problem 1 and 2, some derivation proof of the respective algorithm is also provided.

2.3.1 Solution to Problem 1 (Forward Procedure)

Given an observation sequence $O = O_0, O_1, O_2, O_3 \ldots O_{T-1}$ (where each observation is an independent event) the aim of this section is to calculate $P(O|\lambda)$ ie the conditional probability of observing the above observation sequence given the model λ.

Rabiner (1989: 262) suggested that a possible way of achieving this is (a brute force approach) by enumerating every possible state sequence of length T.
Thus consider a fixed state sequence of length T:

$$Q = q_0, q_1, q_2, q_3 \ldots q_{T-1} \tag{11}$$

where q_0 represents the initial state. The conditional probability for observing the observation sequence O given the state sequence Q and the model is thus:

$$P(O \mid Q, \lambda) = \prod_{t=0}^{T-1} P(O_t \mid q_t, \lambda) \tag{12}$$

Equation (12) describes the joint probabilities (probability product of independent events) of the probabilities of hidden state q_t emitting observation O_t given the model λ, for all t. The probability of the model emitting observation O_1, for example, with a hidden state q_1 is denoted as $b_{q_1}(O_1)$ and is a parameter of the HMM B matrix (which will be introduced later on). **Equation (12)** translates into the product of the respective elements in matrix B for each q:

$$P(O \mid Q, \lambda) = b_{q_0}(O_0).\, b_{q_1}(O_1).b_{q_2}(O_2) \ldots b_{q_{T-1}}(O_{T-1}) \tag{13}$$

The probability of the hidden state sequence in (11) existing (given the model) is thus:

$$P(Q \mid \lambda) = \pi_{q_0} a_{q_0 q_1}\, a_{q_1 q_2}\, a_{q_2 q_3} \ldots a_{q_{T-2} q_{T-1}} \tag{14}$$

Where $a_{q_0q_1}$, for example, represents the probability of the Markov process transitioning from state q_0 to state q_1 and π_{q_0} is the probability of the Markov process initially being in state q_0.

From Bayes formula [presented in **Equation (5)**], the joint probability that the Observation sequence O and the State sequence Q occur simultaneously is the product of **Equations (13)** and **(14)**.

$$P(O,Q \mid \lambda) = P(O \mid Q, \lambda).P(Q \mid \lambda) \tag{15}$$

It is important to note that **Equation (15)** refers to the joint probability of the observation sequence O being observed for a specific sequence Q. Extending this probability to all possible permutations of state sequences Q (which are mutually exclusive events), it is necessary to sum **Equation (15)** over all possible combinations of Q to obtain **Equation (16)**. The statistical theorem applied here is the event union theorem in **Equation (9)** which calculates the probability of O existing for State Sequence 1 or State Sequence 2 or State Sequence 3 or ... or State Sequence n, given the model. The resultant quantity is thus the probability of observation sequence O given the entire model ie P(O| λ) as opposed to the probability of observation sequence O given the model and a fixed state sequence as in **Equation (11)**

P(O| λ) is thus the solution to problem 1 and can be written as follows:

$$P(O \mid \lambda) = \sum_{AllQ} P(O \mid Q, \lambda).P(Q \mid \lambda)$$

$$P(O \mid \lambda) = \sum_{AllQ} \pi_{q_0} b_{q_0}(O_0) a_{q_0q_1} b_{q_1}(O_1) a_{q_1q_2} b_{q_2}(O_2) a_{q_2q_3} \ldots b_{q_{T-1}}(O_{T-1}) a_{q_{T-2}q_{T-1}} \tag{16}$$

Where $b_{q_{T-1}}(O_{T-1}) a_{q_{T-2}q_{T-1}}$, for example, refers to the product of the emission probability (a probability obtained from the HMM B matrix that specifies the probability of an observation being emitted for a specific state) for state q_{T-1} and the probability of state q_{T-2} transitioning to state q_{T-1} at time t = T – 1 (the last datapoint).

Rabiner (1989: 262) demonstrated that the calculation of **Equation (16)** involves in the order of $2T.N^T$ binary computations. For relatively few states (say N = 3) and observations (say T = 10) the approximate number of calculations required would

amount to 1.1million. This is computationally infeasible even for the above small values for N and T respectively.

A more efficient procedure exists and is credited to Baum and Welch. It is a recursive procedure called the Backward-Forward Procedure to solve problem 1. The procedure described by Rabiner (1989: 262) is as follows:

Firstly a variable (called the Forward Variable and given the symbol **α**) is defined as the probability of a partial observation sequence ($O = O_0, O_1, O_2, O_3 \dots O_t$) being observed up to time t and for state $q_t = x_i$. This means that the Markov Process (hidden layer) is specified to be in state x_i (where state x_i is an element of the possible states vector X) at time t.

The Forward Variable can thus be expressed as the following equation:

$$\alpha_t(i) = P(O_0, O_1, O_2, O_3 \dots O_t, q_t = x_i \mid \lambda) \quad (17)$$

Equation(17) can thus be interpreted as the probability of an observation sequence up to time t and the Markov process being in state x_i at time t, given the model λ. It is important to note that this is a probability and that all possible state sequences up to time t need to be considered. The state at time t is known and is: x_i. This means that the permutation of all state sequences must end (or intersect) with state x_i at time t.

To understand this concept better, a more visual approach to the explanation of this procedure is proposed. Consider a lattice structure of all possible states at each discrete time period (see **Figure 2.2**). The Forward Variable is thus a probability calculation for an observation sequence at each time period t and for each state i. In essence each forward variable is the joint probability described in **Equation (16)** for all Q up to time t -1 and the intersecting probability of being in state q_i at time t. By including all possible state sequences, the above sentence can be reworded as: the Forward Variable is the intersection of the probability of an observation sequence up to time t -1 ie $P(O_0, O_1, O_2, O_3 \dots O_{t-1} \mid \lambda)$ and the probability of the model being in state x_i while simultaneously observing O_t at time t.

Any forward variable $\alpha_t(i)$ (as can be seen in **Figure 2.2**) is related to the prior alphas (the forward variables as of 1 time period ago) by the state transition matrix A (and

more specifically the individual element a_{ij}). Since all the respective components relate to probabilities and $\alpha_t(i)$ can be "reached" by any of the prior alpha variables, $\alpha_t(i)$ = the probability union (sum) of (the prior alphas * the respective transition (A matrix) probabilities) * O_t (emission probability at time t). Not that the state labels i and x_i are congruent - the shortened version i is convenient when referring to a state subscript in the alpha variable [eg $\alpha_t(1)$]

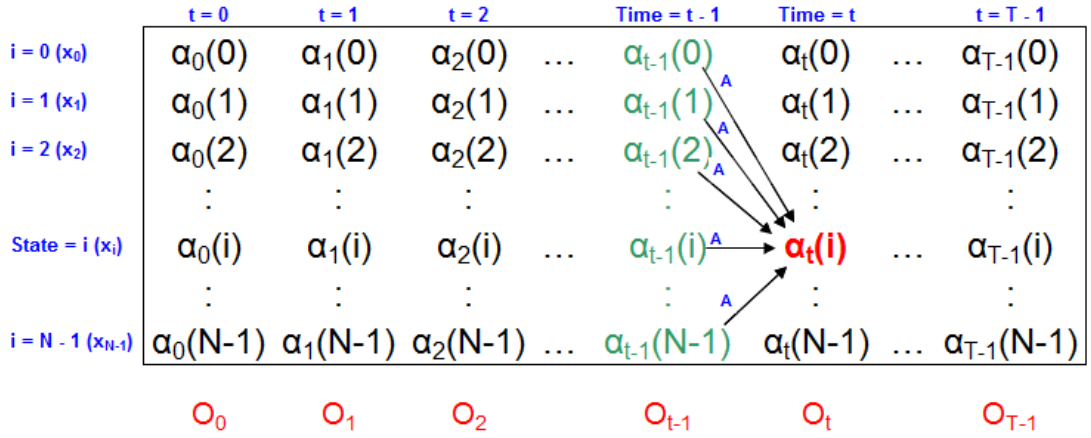

Figure 2.2 Visualisation of a Forward Variable at time t and state i

Formally, the Forward Variable is expressed as the following:

Initialisation:

$$\alpha_0(i) = \pi_i b_i(O_0) \tag{18}$$

Equation(18) applies for *i = 0, 1, 2 to N – 1*.

Recursion:

$$\alpha_t(i) = \left[\sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji}\right] b_i(O_t) \tag{19}$$

Equation(18) applies for i = 0, 1, 2 to N – 1 and t = 1, 2, 3 … T – 1.

Termination:

$$P(O \mid \lambda) = \sum_{i=0}^{\text{N-1}} \alpha_{\text{T - 1}}(\text{i}) \tag{20}$$

A discussion on **Equations (18), (19)** and **(20)** follows:

Equation (18) is the product of the initial state probability: π_i and the emission probability for the first observation: $b_i(O_0)$ for each state: i. $\alpha_0(i)$, by definition, thus provides the probability of observing the partial observation sequence (O_0) and any state sequence of length 1 (the most trivial of state sequences), given λ . $\alpha_0(i)$ over all Q [as described in **Equation (16)**] will thus provide $P(O_0| \lambda)$ – this is thus the probability of observing the very first observation O_0 given the model. It should be noted that "all Q" means all possible state sequences (i = 0 to i = N-1). This is an important observation as **Equation (20)** is essentially proven inductively using the above reasoning.

Moving on to **Equation (19)**, it is important to consider the specific case for t = 1 and i = 0 (or x_0) before discussing the generic variables for time t and state i respectively. For t = 1, **Equation (19)** is reduced to:

$$\alpha_1(0) = \left[\sum_{j=0}^{N-1} \alpha_0(j) a_{j0} \right] b_0(O_1) \tag{21}$$

Equation (21) shows that $\alpha_1(0)$ (t = 1 and terminating state with index = 0) is the sum of the products of [$\alpha_0(i)$ and the transition probabilities from states i = 0 to N – 1 to state 0 (or x_0) respectively] multiplied by the observation emission probability of O_1 for state i = 0. Thus for t = 1 observing the partial observation sequence (O_0,O_1) and any state sequence (of length 2) ending with state x_0, has an associated probability of $P(O_0,O_1,q_t = x_0| \lambda) = \alpha_1(0)$.

$\alpha_1(i)$ is thus an extension of the above reasoning for any state sequence, of length 2, ending with state: x_i. The probability of this event is expressed symbolically as $P(O_0,O_1, q_t = x_i| \lambda)$ and numerically as $\alpha_1(i)$.

Noting that $\alpha_1(i)$ has been "reached" [see **Figure 2.2** for a graphical explanation of this concept for $\alpha_t(i)$] from all possible states in t = 0 then the sum (probability union)

of $\alpha_1(i)$ (for i = 0 to N – 1) is, intuitively, a probability comprising all possible state sequence permutations of length = 2. This probability is thus: $P(O_0,O_1|\ \lambda)$ ie the probability of observing the partial observation sequence (O_0,O_1) given λ. Note that this probability is now independent of being in a particular state at time t = 1.

This is the power of the recursive Alpha Variable as all state sequence permutations are calculated for incremental values of Alpha (and incremental observation sequence lengths) as opposed to calculating all state sequence permutations for the complete observation sequence as in **Equation (16)**

Extending this reasoning, inductively, to $\alpha_{T-1}(i)$, it follows that $\alpha_{T-1}(i) = P(O, q_t = x_i|\ \lambda)$ where $O = O_0, O_1, O_2, O_3 \ldots O_{T-1.}$ Summing $\alpha_{T-1}(i)$ over (i = 0 to N -1) [as described in **Equation (20)**] effectively sums $P(O, q_t = x_i|\ \lambda)$ over all Q (all possible permutations of state sequences) which ultimately provides $P(O|\ \lambda)$ (a probability now independent of being in a particular state at time t = T - 1) and efficiently solves HMM problem 1. Since the Forward Algorithm is an integral part of the final HMM algorithm, pseudo code is provided in APPENDIX E.

With respect to the efficiency of the Forward (Alpha) Variable, Rabiner (1989: 263) noted that $P(O|\ \lambda)$ requires in the order of N^2T calculations as opposed to $2T.N^T$ calculations in the brute force method. This means that for N = 3 states and T = 10 observations the calculations required to calculate $P(O|\lambda)$ are reduced from 1.1million (as in the brute force method) to 90.

2.3.2 Solution to Problem 2 (Viterbi Algorithm)

Problem 2 can be stated as follows:

Given the model **λ** = (A, B, π) and a sequence of observations: O, find an optimal state sequence for the underlying Markov process. The requirement is therefore to find the sequence of hidden states that best explains or accounts for the observed sequence.

The solution to this problem, as described in (University of Leeds. 2010), is similar to that of Problem 1 and involves defining the new variable: δ (Greek symbol Delta – lower case). δ represents a partial probability and is recursively used to find a partial best path (of states) through an HMM model.

In describing the solution to this problem (University of Leeds. 2010) a trellis diagram (displaying all possible state paths) was used to describe the most likely state path routing for each time step given a specific observation. The approach (University of Leeds. 2010) was adapted into a lattice diagram (see **Figure 2.3** HMM Lattice Visualisation) to visualise the solution to this problem.

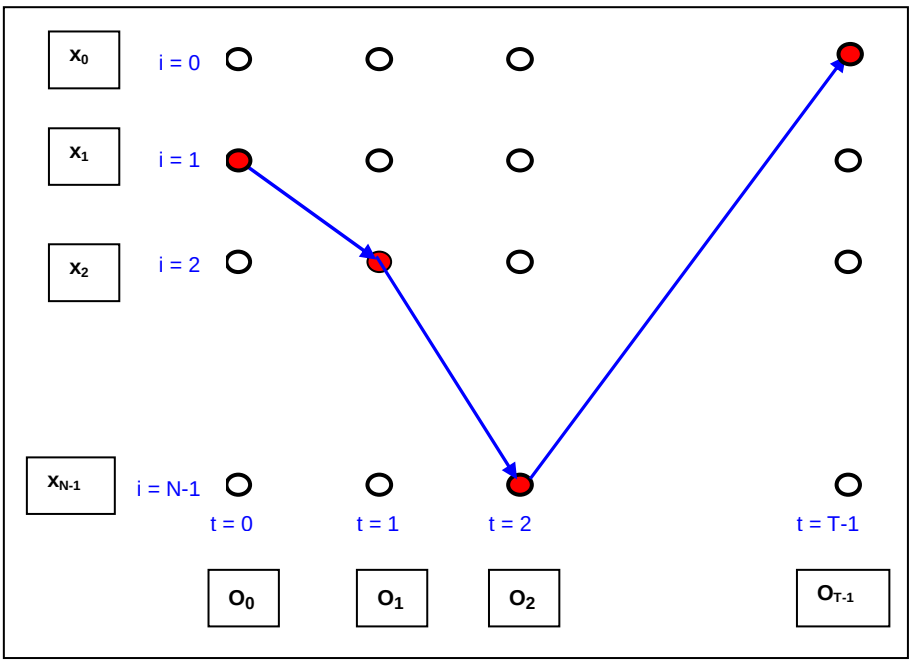

Figure 2.3 HMM Lattice Visualisation

From **Figure 2.3**, the HMM can be considered as a grid (lattice) of possible states (vertical axis) and observations (horizontal axis). For any observation sequence O = O_0, O_1, O_2, O_3 … O_{T-1}, there is a corresponding state sequence Q = {q_0, q_1, q_2 … q_{T-1}} that best fits or explains this Observation sequence. For each observation symbol O_t at time t, the most probable hidden state, q_i, is an element of the state vector X = {x_0, x_1, x_2 … x_{N-1}}. Hence depending on the model λ = (A, B, π), any permutation of state sequences can "explain" a given observation sequence. The aim of this section is to discuss the algorithm that explains how the most likely hidden state is calculated for each consecutive observation symbol. At this state the fundamental property of an HMM (of first order) should be recalled: the current state is given by the probability function that only considers the previous state. **Equation (22)** provides the most fundamental (recursive) probability calculation with respect to HMM's as follows:

Pr (of being in state i at time t) = Pr(of being in state j at time t – 1).Pr(i given j).Pr(Observing O_t) (22)

Referring to **Figure 2.3**, the partial probability $\delta_t(i)$ is expressed for a specific time and state (ie for any point in the lattice). The Viterbi algorithm starts with t = 0 and finds the state i that maximises $\delta_0(i)$. Thereafter it proceeds to time t = 1 to find $\delta_1(i)$ and eventually calculates $\delta_{T-1}(i)$ for time t = T – 1 with the overall aim to find the most probable sequence of hidden states given a sequence of observations. The calculation of $\delta_t(i)$ is such that part of the calculation includes selecting the maximum partial probability from the previous time stamp: $\delta_{t-1}(i)$ to ultimately find the partial best path to state i at time t. The idea is that the highest probability state is the state that best fits the observation sequence observed to time t.

Formally the partial probabilities are expressed as follows:

The initial partial probabilities, as seen in **Equation (23)**, are the product of the initial state distribution vector (for the respective state **i**) and the 1st observation probability (for state **i**):

$$\delta_0(\mathrm{i}) = \pi_i b_i(O_0) \tag{23}$$

$\delta_0(i)$ represents the probability of the model being in state i for t = 0 and for observation O_0. Note that for the time stamp t = 0, a partial probability is calculated for each state i where i = 0 to N-1. Anticipating the next step, the state i that maximises the partial probability $\delta_0(i)$ for t = 0 and for the given observation O_0 is thus the most likely initial state of the model and must be saved in order to calculate $\delta_1(i)$

Following on to the next time stamp: t = 1, the partial probability for state i and t = 1 and Observation O_1 can be expressed in words as:

- The most probable path getting to state i for time stamp t = 1. At this stage there are only two time stamps to consider (t = 0, 1).
- Starting with the partial probabilities $\delta_0(j)$ of time stamp t = 0 for each state j (note the use of j to refer to a state in the previous time stamp), the algorithm finds the highest product of this partial probability $\delta_0(j)$ and the transition probability a_{ji} where i represents the state in the previous point.
- The state that maximises the above probability product is again saved.
- The maximised probability product multiplied by the emission probability $b_i(O_1)$ gives the partial probability for $\delta_1(i)$ where i = 0 to N-1.
- Thus for each state in time t = 1, a partial probability $\delta_1(i)$ has been calculated and the state that maximises this probability has been identified.
- Note that no determination on the optimal state can be undertaken as the Viterbi algorithm requires a final observation (in the sequence) in order to work backwards to identify the states that best explain all these observations. This is one of the problems of the Viterbi algorithm with respect to its real time use - in real time there is no final observation ie the observation sequence is continuous.

The above can be described algebraically as:

$$\delta_1(\mathrm{i}) = \max_{j\in\{0,1\ldots N-1\}} \left[\delta_0(\mathrm{j})\mathrm{a}_{\mathrm{ji}}\right] b_i(O_1) \qquad (24)$$

It should be noted that the term: $b_i(O_1)$ is specifically excluded from the maximum function because this observation probability is a function of i and not j.
Extending the above for the general case, the following general **Equation (25)** can be expressed for the partial probability $\delta_t(i)$ where t = 0,1,2 … T -1.

$$\delta_t(\mathrm{i}) = \max_{j \in \{0,1 \ldots N-1\}} \left[\delta_{t-1}(\mathrm{j})a_{ji}\right] b_i(O_t) \quad (25)$$

Equation (25) can be visualised as a partial probability value for each point in the lattice described in **Figure 2.3**. It should be noted that the value of j that maximises the probability: $\delta_{t-1}(j)a_{ji}$ is in fact the optimal or most likely state for time t = t – 1 that resulted in the model arriving at state i at time t. The second part of the algorithm entails, for each state i at time t, storing a back pointer $\Phi_t(i)$ (*Greek phi symbol*) which stores the optimal preceding state [j from **Equation (25)**].

Formally, this back pointer can be written as:

$$\phi_t(\mathrm{i}) = \arg\max_{j \in \{0,1 \ldots N-1\}} \left[\delta_{t-1}(\mathrm{j})a_{ji}\right] \quad (26)$$

From **Equation (26)** the following applies:

- To calculate the partial probability $\delta_t(i)$ at time t and for state i, the delta variable for time (t – 1) was required.
- From **Equation (25)** it can be seen that the value for j that maximises the product: $\delta_{t-1}(j)a_{ji}$ determines the value of $\delta_t(i)$.
- Once the last observation O_{T-1} is observed, the final partial probability $\delta_{T-1}(i)$ can be calculated for each state i.
- The state i that maximises $\delta_{T-1}(i)$ is thus the last optimal state in the sequence of T - 1 states. At this stage only one optimal state is known - the last state: i_{T-1} of the state sequence.
- Using **Equation (26)** the 2nd to last optimal state: i_t (down to the 1st optimal state in the sequence) can be deduced by accessing the back pointer as described in **Equation (27)** for t = T-2, T-3 ... 0.

$$i_t = \phi_{t+1}(i_{t+1}) \quad (27)$$

From **Equation (27)** it can be observed that the Viterbi algorithm is one where the final state (that optimises the final observation in a sequence) is calculated first. The algorithm then backtracks to find the optimal preceding states. Note that pseudo code for the Viterbi Algorithm is contained in APPENDIX E

In real time there is no final observation which means that some modification to the Viterbi algorithm is required to initiate the backtracking stage. This modification is known as the Real Time Viterbi algorithm and is discussed in Section 4.1.1.

2.3.3 Solution to Problem 3 (Baum Welch Algorithm)

The Baum Welch Algorithm can be considered a methodology to re-estimate or train the Hidden Markov Model. Specifically, this algorithm is used to re-estimate the matrices defining the Hidden Markov Model λ = (A, B, π) where

- **A** refers to the state transition matrix - the probabilities of a specific hidden state transitioning to another hidden state.
- **B** refers to the observation probability matrix - the probabilities of a specific hidden state emitting a specific probability.
- **π** refers to the initial state distribution matrix - the probabilities of the model initially being in the respective hidden states.

The model parameters are re-estimated in such a way that P(O| λ) [the calculation of this probability is called the Forward-Backward algorithm and is provided in **Equation (20)**] is maximised and the output is a new HMM model λ'. It should be noted that there is no algorithm available that produces optimal parameter values for the respective HMM matrices. The best available technique is the iterative Baum-Welch algorithm that iterates over the parameters to find a local maximum of P(O| λ).

While the derivation of this algorithm is out of the scope of this dissertation, the basic concept behind the re-estimation procedure is as follows - Rabiner (1989: 264):

Considering the state transition matrix A and the general element a_{ij} where a_{ij} is the probability of the model transitioning from state i to state j, the re-estimated probability for this parameter $\overline{a_{ij}}$ is given by:

$$\overline{a_{ij}} = \frac{Number_of_transitions_from_state_q_i_to_state_q_j}{Number_of_transitions_out_of_state_q_i} \qquad (28)$$

Equation (28) applies the standard probability theory which states that the probability of an event occurring is the number of occurrences of that events over the total number of events. In this case, the events are the hidden states that describe a given observation sequence.

Similarly for the Observation Probability Matrix, the probability of the model emitting observation k at time t and being in state q_i is denoted as $b_i(k)$. The variable k is thus the index of the observation contained within the set of possible observations. A similar approach is applied to calculating the re-estimated observation probability $\overline{b_i(k)}$:

$$\overline{b_i(k)} = \frac{Number_of_times_in_state_q_i_while_observing_O_k}{Number_of_times_in_state_q_i} \quad (29)$$

Equation (29) applies similar logic in that the observation probability for a particular state with that observation symbol is the number of times that this particular observation with this state has been counted (in a given Observation sequence) divided by the number of times that this hidden state has existed.

In order to calculate the numerator and denominator of **Equation (28)** and **Equation (29)** respectively, the following new variables are defined (Greek symbol xi and gamma respectively): ξ(i,j) and γ(i).

ξ(i,j) is a variable that provides the probability of being in state q_i at time t and state q_j at time t + 1 given the model λ and observation sequence O.
γ(i) is a variable that provides the probability of being in state q_i at time t given the model λ and observation sequence O.

Summing γ(i) over a time period (with a number of observations) can be interpreted as the number of times that this specific state is to be expected for that observation sequence. For example, if the probability of being in a specific state is 0.5 then over an observation sequence of length 10, the expected number of these states being present is thus 5.
Similarly, summing ξ(i,j) over a time period can be interpreted as the number of times that a transition from state i to state j can be expected for a specific observation sequence.

Applying the above, **Equation (28)** becomes:

$$\overline{a_{ij}} = \frac{\sum_{t=0}^{t=T-2} \xi(i, j)}{\sum_{t=0}^{t=T-2} \gamma(i)} \tag{30}$$

which is iterated for all states i and states j.

Similarly **Equation (29)** becomes:

$$\overline{b_i(k)} = \frac{\sum_{such_that:Ot=k}^{t=T-1} \gamma(i)}{\sum_{t=0}^{t=T-1} \gamma(i)} \tag{31}$$

which is iterated for all states i and observations k.

Applying **Equations (30)** and **(31)** respectively, it has been proven by Baum and his colleagues that the probability P(O| λ') is either greater to or equal to P(O| λ) where λ' represents the re-estimated model [with the new parameters estimated in **Equations (30)** and **(31)**] . If the respective probabilities are equal then the model has converged to a local maximum which is also known as a critical point - Rabiner (1989: 265). If not the procedure can continue iterating until the difference between P(O| λ') and P(O| λ) is less than some termination threshold. Note that pseudo code for the Baum Welch Algorithm is contained in APPENDIX E

Although the Baum-Welch procedure has not been used in developing the Froth Detector algorithm (the problem is adapting it into a real time environment) an example will be presented as follows:

Consider the Fair/Loaded Dice model presented in CHAPTER 3. The system is such that a single fair dice is used for a few throws after which a loaded dice is introduced for a few throws. The dice is then switched back to the fair dice for the last few throws. When the fair dice is used, each number has a 1/6 probability of being

thrown. When the loaded dice is used, the number six is given a 50% probability of being rolled.

Creating a Hidden Markov Model of the above, the following applies:

- The observations of this model are the dice faces 1 to 6.
- The hidden states are the types of dice used: fair and loaded.
- The switch from fair to loaded dice and vice versa should be a relatively rare event and once a particular dice is being used it is likely to be used for a number of throws.
- The initial state distribution vector would be something like (1.0, 0) indicating that the fair dice is initially utilised.
- The State Transition Matrix and Observation Probability matrices should look something like in **Equations (32)** and **(33)** respectively.

$$A = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} \tag{32}$$

$$B = \begin{pmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/10 & 1/10 & 1/10 & 1/10 & 1/10 & 5/10 \end{pmatrix} \tag{33}$$

In order to generate an observation sequence that conforms to the above specification, the algorithm discussed in Section 3.2.1 was utilised. This algorithm actually generates a random observation (dice throw) sequence based upon the specified start and end of the loaded dice sub-sequence (the rest of the sequence utilises the fair dice which applies the probability of 1/6 of rolling each dice face). The loaded dice is always biased on the number 6 but the probability of throwing a six can be adjusted by adjusting a parameter. Once the probability for a loaded six is specified the remaining probability space is shared equally for the dice faces one to five. In this example the loaded six probability was set to 50% and the simulation was selected for 50 dice throws in total with the loaded dice introduced at throw 20 and removed after throw 35 - the first and last throws using the loaded dice are thus 20 and 35 respectively.

The observation sequence (of 50 dice throws) that was generated by the fair/loaded dice algorithm is as follows (the bold sub-sequence indicates the specified switch to the loaded dice) in **Equation (34)**:

$$O = (4, 3, 2, 3, 6, 2, 2, 3, 4, 1, 2, 3, 1, 1, 4, 5, 3, 4, 4, \mathbf{1, 6, 6, 5, 6, 3, 6, 6, 4, 6, 6, 6, 6, 6, 3, 6}, 6, 5, 6, 1, 3, 4, 4, 3, 1, 5, 5, 6, 5, 1, 2) \quad (34)$$

In reality, the details of the system will not be known to the observer. The only information that will be available is the observation sequence provided in **Equation (34)**.

Before executing the Baum-Welch algorithm, initial values need to be loaded for the A, B and π matrices respectively.

The following equations [**Equation (35) to (37)**] summarise the initial HMM parameters loaded in the HMM model for this test:

$$A = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix} \quad (35)$$

$$B = \begin{pmatrix} 0.167 & 0.167 & 0.167 & 0.167 & 0.167 & 0.167 \\ 0.15 & 0.15 & 0.15 & 0.15 & 0.15 & 0.25 \end{pmatrix} \quad (36)$$

$$\pi = (0.5, 0.5) \quad (37)$$

Note that the B matrix is the only HMM matrix that was modified with an initial guess of the parameters - this was done to speed up the convergence of the Baum Welch algorithm. The rest of the HMM matrices (A and π) were configured with arbitrary values.

The Baum Welch algorithm was then executed for 50 iterations and the results were pasted (directly from the logbook) into **Figure 2.4**

```
#10108   9:55:02 p.m.   Completed BW after 50 iterations. Start probability was:
6.132e-39. Final probability was: 5.469e-35. This is a change of 8.917e5 % and took
5 minutes to compute.
 RE-EST-INI-STATE-SEQ = sequence (1.0,
  4.243e-57), RE-EST-STATE-SEQ = sequence (sequence (0.917,
    0.083),
  sequence (0.112,
    0.888)), RE-EST-OBS-PROB-SEQ = sequence (sequence (0.249,
    0.142,
    0.248,
    0.251,
    0.07,
    0.039),
  sequence (1.846e-5,
    6.613e-26,
    0.097,
    0.046,
    0.193,
    0.664)).
```

Figure 2.4 Baum Welch Algorithm Output for Fair/Loaded Dice Example

From **Figure 2.4** the re-estimated HMM matrices are thus:

$$\bar{A} = \begin{pmatrix} 0.917 & 0.083 \\ 0.112 & 0.888 \end{pmatrix} \tag{38}$$

$$\bar{B} = \begin{pmatrix} 0.249 & 0.142 & 0.248 & 0.251 & 0.07 & 0.039 \\ 1.846\text{e-}5 & 6.613\text{e-}26 & 0.097 & 0.046 & 0.193 & 0.664 \end{pmatrix} \tag{39}$$

$$\bar{\pi} = (0.917, 0.083) \tag{40}$$

Referring to **Equations (38)** to **(40)** the following applies:

- The probability P(O| λ) (obtained from **Figure 2.4)** improved by 8.917e5 % over the 50 iterations. This is the probability of observing the observation sequence in **Equation (34)** given the model λ. P(O| λ) is calculated using the Forward Algorithm in this case.
- The re-estimated Initial State Distribution vector (π), **Equation (40)**, indicates that the probabilities of the first hidden state in the model being Fair and Loaded are 0.917 and 0.083 respectively. Considering that the first portion of the observation sequence in **Equation (34)** was generated by a Fair dice, the re-estimated initial state parameters are a very good estimation.
- The re-estimated matrix parameters for the state transition matrix (A), **Equation (38)**, indicate that there is a 0.917 probability of the next throw being from a fair dice if the current throw was also from a fair dice. There is thus a 0.112 probability of the next throw being from a fair dice if the current dice is loaded. These parameters are also a very good re-estimation.
- The re-estimated emission matrix (B) parameters in **Equation (39)** are a reasonable estimate but the effects of a relatively short observation sequence are obvious in the 2nd row. The 2nd row refers to the Loaded state while the columns refer to the observations (dice faces one to six). The number six was observed often in the loaded dice section, of the observation sequence in **Equation (34),** hence the high probability of 0.664. The numbers one and two did not feature much in this Loaded sub-sequence hence their low respective observation probabilities of 1.846e-5 and 6.613e-26.

2.4 Summary

This chapter presented the main three algorithms of Hidden Markov Models and their derivation. These problems (and respective solutions) are summarised as follows:

- The problem of calculating the probability of an observation sequence given the model. The solution to this is known as the Forward-Backward algorithm.
- The problem of calculating the sequence of hidden states that best fits a given observation sequence. The solution to this is known as the Viterbi algorithm.
- The problem of training the HMM and obtaining the A, B and π matrices from a training sequence of observations. The solution to this is known as the Baum-Welch algorithm.

Since the Baum Welch algorithm was not utilised in the final model and is not discussed further in this document, a practical example of its use was presented where its strengths and weaknesses were evaluated. One of the problems that can occur with the Baum Welch re-estimation procedure is evident in **Equation (39)** - an observation sequence that is too short may not contain sufficient occurrences of the respective observations. The net result is a low observation probability for that observation.

A possible solution to this problem would be to run a number of re-estimation calculations on different observation sequences. The resulting parameters could then be averaged to give the final re-estimated matrix set.

Finally, the above example indicates that it is possible to populate the respective HMM matrix parameters using the Baum Welch algorithm without much prior knowledge of the system. The Baum Welch algorithm was not included in the HMM Froth Detector algorithm as fixed HMM matrix parameters provide consistent model performance as well as control over the aggressive/conservative tuning of the model. The fixed HMM parameters, in the case of the Froth Detector algorithm, were derived using prior experience of the system and are discussed in more detail in Section 5.4.

CHAPTER 3 PRACTICAL HMM EXAMPLE

In this chapter the theory in the preceding chapter will be utilised to analyse a test HMM case. The various algorithms have been developed into software and will be tested in the example of detecting loaded dice in an occasionally dishonest casino. In this example a relatively simple HMM will be utilised to demonstrate the use of the Viterbi algorithm. A normal set of fair dice (singular) has six sides with each side having an equal probability of showing. The probability of throwing a particular number (one to six) is thus 1/6.

In the case of a loaded dice, the probability of throwing a specific number is biased higher than the standard 1/6. It is important to note that although other numbers might appear, the biased number will appear statistically more often.
In this example, an HMM is used to detect the case whereby only one of the numbers of the dice is biased (number six).

3.1 Formulating the Model

There are three main components to every HMM:

- The Emission Probability matrix (B)
- The State Transition matrix (A)
- The Initial State Distribution matrix (π)

Constructing the above matrices will be discussed in the following sections.

3.1.1 Practical HMM Design Guidelines

At this stage it is important to note that there are no fixed rules for constructing the various HMM components. Like most algorithms the final HMM parameters will require tuning - either using the Baum Welch algorithm (discussed in Section 2.3.3) or manually as was the case in the final HMM Froth Detector algorithm.

There are however rough guidelines that can be followed to create the basic HMM infrastructure. Such a guideline is available from Stamp (2004: 13) and contains the following general tips:

- Choose the number of states N to define the model.
- Choose the possible state transitions in terms of what possible state transitions exist (ie this merely implies deciding which state transition probabilities a_{ij} are non-zero)
- Choose the number of observations M that the model can possibly emit.
- Let $\pi_i \approx 1/N$ for all i states
- Ensure that the dimension of π is *1 x N*.
- Let aij ≈ 1/N for all i and j state transitions.
- Noting that A = { aij } and the dimension of A is *N x N*.
- Let $b_j(k) \approx 1/M$ for all j states and k observations.
- B = { $b_j(k)$ } and the dimension of B is *N x M*.
- If some empirical knowledge of the model is available then insert intuitive values for A, B and π respectively.
- Ensure that the sum of all rows of every matrix sums to 1 (see note on Row Stochastic Matrices in Section 2.2.1).
- Ensure that the elements of each matrix are not the same value.

The following section will thus apply these guidelines in developing the Hidden Markov Model for a simple model consisting of a Fair/Loaded dice system.

3.1.2 The Emission Probability Matrix (B) for the Loaded Dice Example

It should be intuitive to note that there are two main hidden states in this model:

- The **L**oaded state
- The **F**air state

In reality these states will not be directly observable and require the model to infer which state is present at any particular time.
In order to do this, a sequence of observations is required. In this particular case, the observation symbols will be any combination of dice faces. Formally:

X = {fair, loaded}
V = {one, two, three, four, five, six}

Where N represents the possible hidden states (2 in this case) and M contains the possible observations (6 faces of the dice in this case).

An example of an observation sequence, O, is thus:

O = {1,2,4,3,5,6,6,6,5,4,6}

The corresponding state sequence that best explains the observation sequence might look something like:

Q = {F,F,F,F,F,L,L,L,F,F,F}

Where F and L refer to the Fair and Loaded states respectively (consecutive sixes might indicate a switch to loaded dice).

Since the model variables are defined, the only task left to complete is to define the probabilities of each observation symbol appearing while the respective hidden state prevails. It is thus necessary to define the probabilities of throwing a particular number for a fair and loaded dice respectively. From the introduction it can be seen that the probability of throwing a number in a fair dice is 1/6. In the case of a loaded dice, it is necessary to define a different set of probabilities. For the purposes of this example, the following probabilities are defined in **Table 3.1**:

Table 3.1 Probabilities for Fair and Loaded Dice

Dice Face	Probability of throwing for a Fair Dice	Probability of throwing for a Loaded Dice
One	1/6	1/10
Two	1/6	1/10
Three	1/6	1/10
Four	1/6	1/10
Five	1/6	1/10
Six	1/6	5/10

From **Table 3.1** it can be observed that in the case of the loaded dice, the bias is on the number six. This means that when throwing this type of dice, a six is 5 times more likely to appear than any other number. In addition, an extremely long sequence of loaded dice throws should result in sixes observed for approximately 50% of the total number of throws.

Since the contents of **Table 3.1** are probabilities of observing the respective dice faces, the emission (observation) probability matrix B can be derived directly from it as follows:

$$B = \begin{pmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/10 & 1/10 & 1/10 & 1/10 & 1/10 & 5/10 \end{pmatrix} \tag{41}$$

From **Equation (41)** it is important to note that the rows refer to the model states (fair and loaded respectively) while the columns refer to the observations (one, two, three, four, five, six). For example, the element at row 2 and column 6 (5/10) refers to the probability (50%) of observing a six for the loaded dice (hidden) state.

3.1.3 The State Transition Matrix (A) for the Loaded Dice Example

Formulating the State Transition Matrix for this example requires more experience than with the Emission Probability Matrix. From the problem definition the following assumptions can be made:

- Switching to loaded dice is a rare event. This means that the fair dice is used most of the time and the switch to a loaded dice (and vice versa) is made occasionally. This implies that the transition probability (say 10%) from fair to loaded (and vice versa) is going to be low. This can be visualised by considering a 100 dice throws. A 10% state change probability implies that there should be roughly 10 transitions from fair to loaded in a 100 throws.
- Once the switch is made, that particular dice is likely to be used for a while. This implies that the transition probability to the same state is going to be high (say 90%).
- Once the switch to loaded dice has been made, the probability of transitioning back to the fair dice has been made the same as in the case of the fair to

loaded dice transition. In reality, the longer the loaded dice is used, the more suspicious the players will become. As such the probability of switching from the loaded dice to the fair dice has also been made 10%. This means that the probability of the loaded dice being used in the next throw is 90%.

From the above discussion, the State Transition matrix has been constructed as follows:

$$A = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} \tag{42}$$

Equation (42) is thus a 2 by 2 matrix with both rows and columns pointing to a particular state with the respective element representing the probability of transitioning between the state defined in the rows to the state indicated in the column. The first row and column points to the Fair state while the second points to the Loaded state. For example the probability of transitioning from the loaded state to the fair state is 0.1 (row 2, column 1).

3.1.4 The Initial State Distribution (π) for the Loaded Dice Example

The Initial State Distribution is a vector of probabilities that specify the probability of the model being in each state. In this example, the assumption is that initially the Fair dice set would be used and that the Loaded dice set would be introduced later into the game. Formally this is written as follows:

$$\pi = (1.0, 0.0) \tag{43}$$

Equations (41) to **(43)** thus fully describe the Hidden Markov Model **λ** = (A, B, π).

3.2 Executing the Model

As part of this dissertation, all of the relevant algorithms (Forward/Backward, Viterbi. Baum-Welch and others) have been developed, *from scratch*, in the expert system G2. Since the final vehicle to answer the research question will be within a G2 environment, all of the relevant software has been coded into this development environment.

A sample snippet of code (a G2 procedure) is as follows:

```
hmm-compute-forward-probability(HMMNAME: text,INITSTATESEQ:
   sequence, STATEMATRIX: class hmm-matrix, OBSPROBMATRIX:
   class hmm-matrix, OBSIDXSEQ: sequence) = (quantity)
HMMFV: class hmm-matrix;
MAXOBS, MAXSTATES,t,i,j,OBSIDX: integer;
PI,OBSPROB,ALPHA,ALPHA-T-1,A-PROB,B-PROB,FINAL-PROB:
   quantity;
RESULT: symbol;
DEBUG: truth-value = false;
begin
  {--- Initialisation Stuff}
  {--- Calc the number of time periods/observations}
  MAXOBS = the number of elements in OBSIDXSEQ;
  {--- Calc the number of states}
  MAXSTATES = the number of elements in INITSTATESEQ;
  {--- Create the empty matrix for the forward variable}
  create a hmm-matrix HMMFV;
  call hmm-create-matrix (HMMFV, MAXSTATES, MAXOBS);

  if DEBUG then
    post "Starting Forward Calculation for [HMMNAME].";

  if DEBUG then
    post "Computing Alpha 0 (i)";

  {--- Compute Alpha 0 (i)}
  for i = 0 to (MAXSTATES - 1) do
    {--- We are dealing with first column of the forward variable matrix}
    t = 0;
    {--- Alpha0(i) = pi(i)bi(O0)}
    {--- We are also dealing with the 1st observation ie O0 so get the index
       of it and find column to use in Observation matrix}
    {--- Obtain the actual observation prob as a function of the state i}
    OBSPROB = hmm-matrix-get-value (the matrix of OBSPROBMATRIX,
       i, OBSIDXSEQ [0]);
    PI = INITSTATESEQ [i];
```

Figure 3.1 An extract of G2 code

3.2.1 Generating Fair and Loaded Dice Throws

When it comes to simulating the respective Fair and Loaded dice throws, it is important to note that this is not something that would be needed in practice. In reality the Hidden Markov Model (HMM) would not be aware of the status of the dice and would also not be aware of the probabilities of throwing a specific number on a loaded dice. For the purposes of this example, it is given that the loaded dice is biased on the number six which has a 50% probability of being thrown. The other five numbers of the loaded dice thus have a 10% probability of being thrown. This section describes how the fair and loaded dice respectively were simulated.

Simulating Fair and Loaded dice throws was achieved as follows:

- A random number (float) between 0 and 100 was generated.
- Ranges were created for each number on a dice. For the fair dice, the range for ONE would be (0,16.667] while the range for TWO would be (16.667,33.333] and continuing up to (83.333,100] for the number SIX.
- For the loaded dice, the range for ONE would be (0,10.0] while the range for TWO would be (10.0,20.0] and continuing up to (50,100] for the number SIX.
- The result of the random number would thus fall into a particular range which would generate the symbol specific to the name of that region (ONE, TWO etc) which corresponds to the number of the dice face.

This algorithm was tested with 1 million iterations (throws) and the percentages of each number thrown were observed. For the fair dice scenario each number (one to six) should appear approximately 16.6% of the time - there will be a small statistical variance. For the loaded dice, numbers one to five should appear 10% of the time while the number six should be recorded for half of the throws (50%), again with a small variance with each test. **Table A1** in APPENDIX A provides tabular results of this test and confirms the expected results.

3.2.2 Model Execution Methodology

When it comes to testing the HMM model and evaluating its effectiveness, a number of scenarios were considered. Additional software was developed to execute thousands of Monte Carlo iterations to measure the performance of the HMM model. Before considering the scenarios, the performance indicators of the system will be discussed below:

- Fair to Loaded Transitions. Once the fair dice is replaced with a loaded dice, the perfect detector will detect this event on that throw. The more throws before or after this transition that the detector requires to detect the loaded dice, the worse its performance.
- Loaded to Fair Transitions. Once in a loaded state, another important event will require detection. This is the return to the fair dice state. Again, the perfect model will detect this event at exactly the right iteration. The more dice throws that appear between the actual event and the model's detection of the event, the poorer the performance of the model.

3.2.3 Model Execution Example

In order to test the Viterbi algorithm, the following dice throw sequence, consisting of 30 consecutive throws was constructed:

(1,6,3,3,4,3,3,5,5,1,**6,6,6,2,6,5,6**,4,3,3,2,1,5,1,6,3,4,6,2,5)

With a fair dice, a reasonable expectation would be to observe one six every six throws. The sub-sequence highlighted in blue above, would thus indicate a statistical anomaly (from the perspective of a normal dice throw sequence) and would thus indicate the usage of a Loaded dice.

When presenting this 30-throw observation sequence through the model and executing the Viterbi algorithm, the results indicate a loaded dice portion as follows:

(F,F,F,F,F,F,F,F,F,F,**L,L,L,L,L,L,L**,F,F,F,F,F,F,F,F,F,F,F,F,F)

To verify the results of the Viterbi algorithm, the same model was configured and the same sequence was executed in (GNU Octave 2009). The results of this exercise are available in **Figure B2** in APPENDIX B and are identical to those presented above. It should be noted that the above example refers to the offline use of an HMM. In other words the entire sequence of observations is available for use by the Viterbi algorithm to determine the state sequence that best explains the observation sequence. One of the strengths of the offline use of HMM's is its accuracy and ability to detect state transitions simultaneous to changes in observation patterns. It achieves this by recursively calculating (using the Viterbi algorithm) over the entire state sequence which state sequence gives the highest probability of existing given the observation sequence.

3.3 Summary

This chapter introduced a practical example of an HMM model. The HMM constructed was utilised to detect the switch from a Fair dice to a Loaded dice in a dishonest casino. The main algorithm utilised in this chapter was the Viterbi algorithm which was used to detect the hidden states that describe an observation sequence of dice throws. In this example the hidden states referred to the type of dice utilised (Fair/Loaded) in the various parts of the observation sequence (the dice faces rolled).

It is important to note that this example was not conducted in real time. The observation sequence was input into the model after the completed sequence of dice rolls was generated. The axis of time was thus not considered in this section.

CHAPTER 4 REAL TIME HMM STATE DETECTION

In the previous chapter a simple fair/loaded dice example was used to introduce the basic concepts of Hidden Markov Models. Of particular importance was the tools used to detect the hidden state of any Hidden Markov Model (HMM) - the most important tool being the Viterbi algorithm.

An important distinction was made between an offline HMM and a real time HMM. An offline HMM is presented with the full observation sequence before the hidden state sequence is calculated.

An online HMM is required to work with an unknown number of observations and must calculate the hidden states as the observations arrive in real time. The two main concepts emanating from the online HMM space are thus:

- **Accuracy**. This is the ability of the online HMM to accurately detect the hidden states as the observations arrive without the benefit of the complete observation sequence. The real time HMM has an inherent accuracy that can at best, be the same as the offline variant. The accuracy of the online HMM is thus dependent on its design.
- **Latency**. An inherent weakness of the online HMM is the need to observe as much of the observation sequence as possible before calculating the hidden states that best fit the observations. To improve accuracy more observations need to be observed before back calculating the states that produced them. This look ahead action has the effect of introducing latency into the detection algorithm. A lag is therefore introduced between observing observations and calculating the hidden state that best explains these observations.

This chapter continues with the Fair/Loaded dice example and explores the above two concepts for various real time HMM algorithms. The ultimate outcome is to make a decision on the best real time HMM algorithm to use for the real world engineering problem.

4.1 An Introduction to Real Time HMM Detection

One of the major problems with respect to utilising HMM's for detection is designing a strategy for real time detection. The model is therefore required to provide an output, as to the current hidden state (*or a sequence of the last n optimal states*), after each observation.
In many applications, the HMM detection is only required some time after a sequence of events has occurred as opposed to while these events are unfolding. The former allows the collection of complete data while the latter requires decision making as each observation is observed.

Ardö (2007: 109) proposed an Infinite Viterbi algorithm to calculate a real state sequence in the case whereby T tends to infinity. Bloit and Rodet (2008: 2) developed a similar Short Time Viterbi algorithm that calculates a fusion point in the past observation history that guarantees that the local Viterbi paths up to this fusion point are identical to the global Viterbi path. In both cases, the principle is the same: a past observation in history is found such that future observations do not affect the optimal state sequence to this observation point.
In Cho and Bello (2009: 3) a real time modification of the classic Viterbi algorithm that utilises a FILO (First In, Last Out) observation buffer (of length L) that contains the observation sequence to be decoded into hidden states, is proposed. The main difference between the classic Viterbi and the real time Viterbi is that the delta variables are re-used for future observations. One of the specifics of their algorithm revolves around identifying the state at the beginning of the LIFO (Last In, First Out) buffer. Their decoded state sequences are stacked into an L x L matrix with the estimated hidden state sequence for time t being derived from a number of potential candidates using the statistical mode function.
Zhang, Liu and Run-Sheng (2002: 2) developed a time-synchronous Viterbi Beam Search algorithm that tests all hypotheses in parallel but one that eliminates very low probability partial paths (as evidenced by the Delta variable). The action of eliminating these low scoring paths is dubbed as "pruning". This research was in the field of speech recognition where each state represents a spoken word. In fact most of the above citations refer to work performed in analysing speech (or music) and deal with models containing a large number of states.
Research undertaken by the Author has not yielded any real time Viterbi algorithms developed for and applied to process control. The ideas presented above have been

integrated into the algorithms presented in the following sections. The Fixed State Test algorithm is one that was conceived and developed by the Author. These real time algorithms have been developed (also in the Gensym G2 development environment) and tested on the same Fair/Loaded Dice system that has been described in the previous chapter. The difference in this case is that the algorithm is required to determine the hidden state of the dice (fair or loaded) in real time as each new dice throw is observed. In the case of the classic Viterbi algorithm, the algorithm has the entire sequence of throws available prior to processing. In this case, the algorithm has a short local sequence of throws available to determine the hidden states.

4.1.1 Fixed State Test (FST) Algorithm

This algorithm was conceived and developed by the Author (pseudo code for the HMM FST Algorithm is contained in APPENDIX E) and is one that tests for a specific sequence of states. To achieve this a slight deviation from the Forward algorithm was adapted. From **Equation (15)** the dependent variable $P(O, Q| \lambda)$ refers to the probability of observing an observation sequence O and a state sequence Q given the model λ. It is important to note that Q is a specific state sequence. Using this probability it is thus possible to develop a model λ, observe an observation sequence associated with the model and (most importantly) test for a specific state sequence **-** the state sequence returning the highest probability is one that is most likely present. In practical terms the following detection strategy was developed:

- Create a buffer of L observations (dice throws in this example) that are packed in a FILO (First In Last Out) manner.
- Test for a number of specific state sequences $Q_1, Q_2 \ldots Q_n$ where $Q_i = \{x_{i0}, x_{i1}, x_{i2 \ldots} x_{iL-1}\}$ that could include any permutation of the respective states (Fair and Loaded in this case). The most obvious states would be a sequence of states with all Fair and all Loaded elements respectively.
- Apply **Equation (15)** to calculate $P_1(O, Q_1| \lambda)$, $P_2(O, Q_2| \lambda) \ldots P_n(O, Q_n| \lambda)$ respectively where $P_i(O, Q_i| \lambda)$ is the probability of observing the observation sequence O (of length L) and the hidden state sequence Q_i given the model λ.
- In this chapter n (the number of state sequences to test) has been chosen as 2. The state sequences (of varying buffer length L) are comprised of entirely

the state Fair and the state Loaded respectively. Specifically, Q_1 = {*Loaded, Loaded, Loaded, Loaded, Loaded*} and Q_2 = {*Fair, Fair, Fair, Fair, Fair*} for L = 5. Thus after each observation $P_1(O, Q_1| \lambda)$ and $P_2(O, Q_2| \lambda)$ is calculated (*actually log[$P_1(O, Q_1| \lambda)$] and log[$P_2(O, Q_2| \lambda)$] are calculated due to the small probabilities involved*)

- The state sequence with the highest (log) probability is then deemed the optimal state sequence. It should be noted that this is not the optimal state sequence as determined by the Viterbi algorithm but is the most likely state sequence of the sequences tested (2 in this case). In essence this algorithm is testing that there are L consecutive hidden fair or loaded states respectively. For short buffer lengths (low L) the algorithm should react quickly to change but will react slowly for larger values of L.
- Since the algorithm has been designed to test for a state sequence of all FAIR or LOADED states respectively, the latency becomes L - 1. Theoretically it is possible to make L = 1. The risk of the model outputting false positives increases with a reduced number of observations - this will be demonstrated later on in this chapter. Thus a balance needs to be maintained between minimising latency (achieved with low values of L) and minimising false positives (achieved with large values of L).

4.1.2 Modified Real Time Viterbi (RTV) Algorithm

The second algorithm developed is a modified Viterbi algorithm as proposed by Cho and Bello (2009: 3) The main components of this algorithm are as follows:

- Create a buffer of L observations that create a FILO sequence of dice throw observations.
- The hidden state sequences that best explains the observation sequence is calculated by the Viterbi algorithm.
- The very first observation (t = 0) utilises the initial state distribution to calculate the very first optimal state. Thereafter a modified Viterbi algorithm is used to calculate the optimal state sequence.
- As observation t = L - 1 arrives, the Viterbi observation calculates the optimal state sequence (of L elements) for the first time. This means that from t = 0 to t = L - 2 there is no output from the model - an obvious latency issue.

- At time t = L, the observation first captured falls out of the observation sequence. At this stage all the partial probabilities (Delta Variables) shift left in the buffer of length L to create space for a new Delta Variable at the most recent element in the Observation Buffer of length L. It should be noted that the Delta Variables are the partial probabilities of the state sequences up to time t. The reason why the state sequence cannot be enumerated at this point is that the classic Viterbi algorithm calls for a back tracking process once the last observation has been observed. This means that the end observation can theoretically affect the first optimal state.
- One of the practicalities not considered in Cho and Bello (2009: 3) is the ever decreasing magnitude of the Delta Variable. With an infinite observation sequence, there will come a time when the precision of the Delta Variable falls out of the range of float variables. The author thus proposes a scaling phase for all observations t > L.
- Referring to the above scaling stage, it is important to understand that only the delta variable at time t will be re-scaled (to the maximum value in the vector). This does not introduce a mathematical problem as the latest Delta Variable is only used to calculate the back pointer for that time stamp and future Delta variables. There will of course be a mathematical discrepancy at the interface between the first scaled vector and the last unscaled vector. As explained above, the back pointers remain unaffected which is the ultimate output of the Viterbi algorithm.
- Once the new Delta Variable has been calculated, the maximum partial path probability (and the state in the buffer that resulted in this) can be determined at this point in time. This is now deemed the last observation as per the classic Viterbi algorithm and a back tracking process (from this state) is used to create an optimal state sequence of length L.
- The final task is to identify the newly calculated optimal state for the new observation observed. As mentioned above, future observations can influence the current state. This means that with each new observation added to the buffer, the elements of the new optimal state sequence: 0, 1, 2 .... L - 2 are not necessarily the same as the following state elements of the old optimal state sequence: 1, 2 .... L -1. The new optimal state sequence is not necessarily going to be one whereby the right most state elements get shifted to the left by one time stamp with a new right most state appended to the sequence - it is possible for the entire state sequence (of length L) to change with a new observation.

- Contrary to the algorithm described in Cho and Bello (2009: 3) which proposes a fairly complex procedure of applying L phase shifted state sequences to calculate an overlapping state (deemed the newest state), the proposed methodology to choose the newest state is one whereby the first state in the buffer (ie the left most state) is deemed the newest state. The reason for this is that the left most state in the state sequence of length L is the least likely state to be influenced by future observations.
- The latency of this algorithm is thus L - 1 (ie one time interval less than the buffer length)

4.2 Qualitative Real Time Algorithm Performance

To test the above mentioned algorithms, the following methodology was used:

- A sequence of 100 dice throws was generated and presented to each algorithm in real time with the algorithms producing outputs with each new observation.
- The first 19 throws use a simulated fair dice (the algorithm to simulate fair and loaded dice throws respectively is discussed in Section 3.2.1). In this example, the probability of throwing a six for a loaded dice was increased to 95% with the probability of the other dice faces being thrown = 1%. This was changed to easily discriminate between fair and loaded dice for the qualitative study only.
- From throw 20 to throw 35 a loaded dice was simulated (again refer to Section 3.2.1 for details on how a loaded dice is simulated). From throw 36 to 59 a fair dice is once again simulated.
- From throw 60 to throw 80 a loaded dice is once again simulated with the last 20 throws changing back to a fair dice.
- Note that the choice of the loaded dice regions was subject to the following set of rules: To be a realistic example the number of throws using loaded dice could not be too large and should be a fraction of the number of throws using fair dice. In addition the positioning of the fair/loaded dice transitions should be sufficiently far apart to allow a relatively long period of fair dice usage - 25 throws in this example. The regions discussed above were thus deemed to be realistic and at the same would provide sufficient variation to test the respective HMM models.
- After each throw, each algorithm outputs the hidden state (fair or loaded) that it deems the system to be in at that stage.
- A real time graph was created to capture the respective events as they happen and a few scenarios are presented below.

In the following scenarios the following colour codes apply for the respective plots in each graph:

- **LOG-FAIR-PROB**. This is the log probability that the hidden state sequence, of length L, being tested is a vector with all elements = FAIR. The reason that

this is constant is that all observation probabilities are 1/6 for FAIR dice throws. Thus no matter what number is thrown, the same emission probability $b_j(k)$ will be used in the calculation.

- **LOG-LOADED-PROB**. This is the log probability that the hidden state sequence, of length L, being tested is a vector with all elements = LOADED. This will increase with more sixes thrown as the loaded dice is biased on this number.
- **ACTUAL-DICE-IS-LOADED**. This trace (with triangle markers) represents the times when the fair dice algorithm or the loaded dice algorithm is in use. As mentioned above, there are specific times in the 100 throw test when each dice is used where (1 = loaded dice, -1 = fair dice). The perfect detection algorithm will be one that follows this trace exactly.
- **LOADED-DICE-DETECTED**. This trace represents the output of the Fixed State Test (FST) algorithm where (1 = loaded dice, -1 = fair dice)
- **AVG-DICE-VAL**. This trace provides the average value of all dice numbers thrown in the buffer (of length L). For a fair dice this value should average around 3.5. The more sixes that are thrown, the higher this number will be. This parameter was included to give some indication of the actual numbers that are being thrown.
- **HMM-LOADED-DICE-DETECTED-BY-RT-VITERBI**. This trace represents the output (1 = loaded, -1 = fair) of the 2nd algorithm: the Modified Real Time Viterbi (RTV) algorithm.

4.2.1 Scenario 1 - The HMM Model Output for a Buffer Length of 10

In **Figure 4.1**, the stages at which the dice is switched from fair to loaded can be clearly seen (black trace with triangle markers). A few throws later the Fixed State Test (FST) algorithm **(**LOADED-DICE-DETECTED trace with a square marker**)** detects this switch and flags that the hidden state is now Loaded. The Real Time Viterbi (RTV) algorithm (HMM-LOADED-DICE-DETECTED-BY-RT-VITERBI trace with no marker) only flags the Loaded stated a few throws later still.

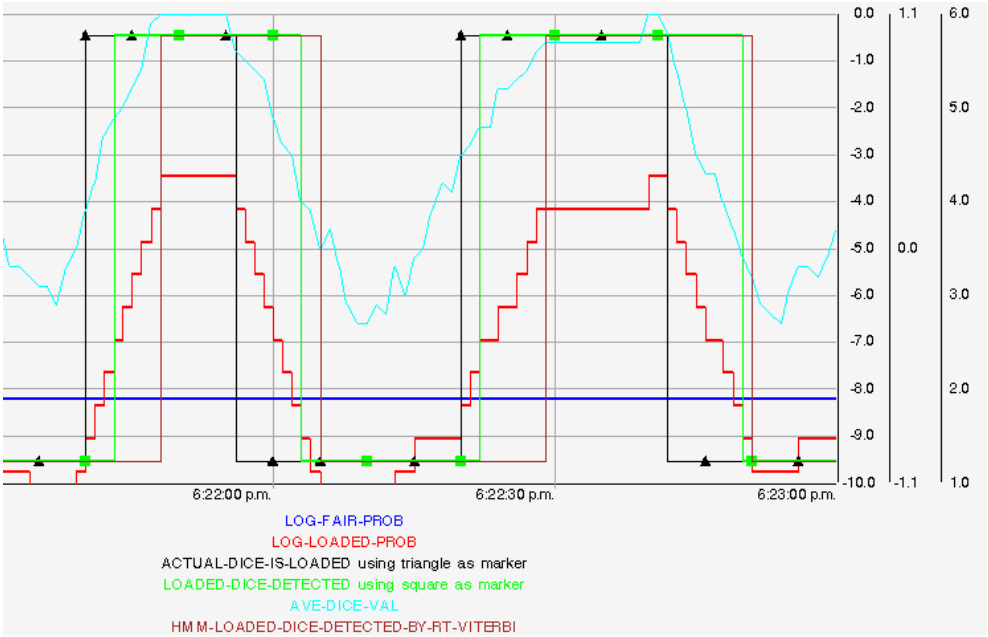

Figure 4.1 Scenario 1 (buffer length of 10)

It is important to note that in this scenario there are no false positives - these would be indicated by the respective RTV or FST algorithms outputting the Loaded state when the Fair dice is actually used.

4.2.2 Scenario 2 - The HMM Model Output for a Buffer Length of 5

Referring to **Figure 4.2** the first important point to note is the shorter delay between the actual dice being changed to loaded (black trace with triangle markers) and the FST and RTV algorithms detecting this change. The green trace with green markers refers to the Fixed State Test algorithm while the brown trace refers to the Real Time Viterbi algorithm.

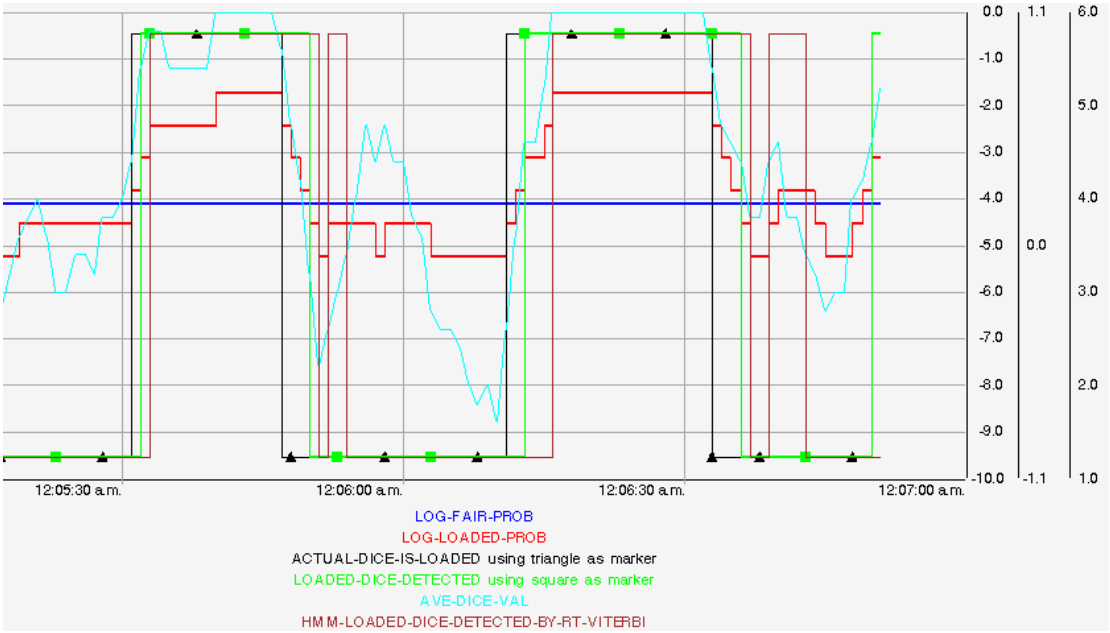

Figure 4.2 Scenario 2 (buffer length of 5)

The second important point to note is the oscillation of brown trace (Real Time Viterbi algorithm) in the beginning and towards the end of the graph. In this case, the algorithm is throwing out false positives as the actual state of the dice is FAIR. As mentioned in the introduction to this section, the shorter the buffer length, the higher the likelihood of false positives. Towards the end of the graph the Fixed State Text algorithm also outputs a false positive (green trace with square markers) - this is probably more a case of too many sixes being thrown by the fair dice (see high average dice value in the cyan plot).

The final point to note is that the Fixed State Test (FST) algorithm is detecting the transitions (fair to loaded and loaded to fair) quicker than the RTV algorithm. This provides some qualitative indication of the performance of each algorithm. In the next section a more qualitative analysis will be undertaken to compare algorithms.

4.2.3 Scenario 3 - The HMM Model Output for a Buffer Length of 8

In this scenario a buffer length of 8 has been used. This is roughly midway between the buffer length of 10 in Scenario 1 and the buffer length of 5 used in Scenario 2.

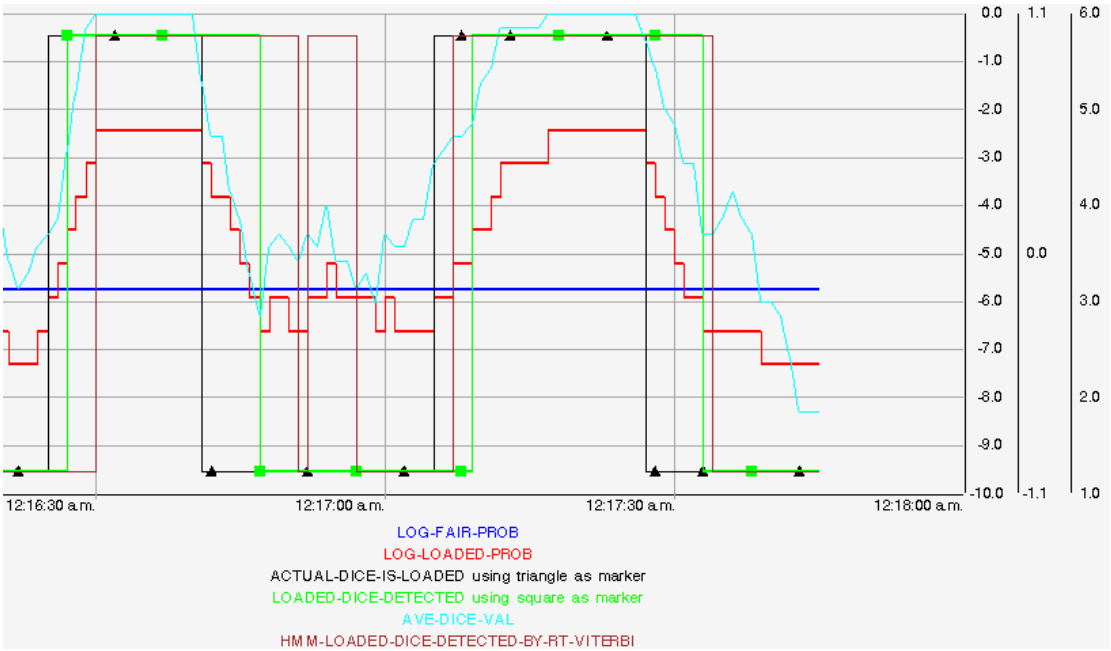

Figure 4.3 Scenario 3 (buffer length of 8)

Referring to **Figure 4.3** it can be observed that the Fixed State Test algorithm (green trace with square markers) responds a little slower than in Scenario 2 but does not suffer from any false positives. The Real Time Viterbi algorithm still suffers from false positives and also experiences a greater latency than the FST algorithm. When testing for the two states in the Fixed State Test algorithm, a relative probability (of one state sequence versus another) is being tested as opposed to finding the optimum Viterbi path as in the RTV algorithm. The former algorithm is thus only concerned with 2 state sequences while the latter attempts to find the optimum sequence which requires a greater convergence in all the possible state sequences to a particular state.

The qualitative observations that can be made in all scenarios above are:

- The FST algorithm has a lower model latency than the RTV algorithm - it detects the dice transitions sooner.
- The FST algorithm experiences less false positives than the RTV algorithm.

4.3 Quantitative Real Time Algorithm Performance

In this section the abovementioned FST and RTV algorithms will be tested quantitatively to assess their respectively performances against the following criteria:

- A varying **buffer length**. As mentioned above, an increasing buffer length increases the latency of the algorithm. The ideal is thus to have a very short buffer length. It should be noted that shorter buffer lengths degrade (false positives are introduced) the detection performance of the HMM so a trade off needs to be established.
- A **varied loaded dice probability**. The HMM model expects the number six to appear with a probability of 50%. Ideally this means a dice sequence with a six as every second number should be observed for the periods when the loaded dice is used. In this performance test, the probability of a six being thrown was varied (between 50% and 90% - with the remaining probability evenly distributed amongst the other dice faces) to observe the performances of each algorithm. The more consecutive sixes that are thrown, the easier the HMM model should discriminate between the fair and loaded dice states respectively.

To achieve the above, software was developed to execute a Monte Carlo simulation (of 1000 iterations with each iteration comprising a sequence of 100 dice throws) of the same sequence of dice throws as described in Section 4.2. What this means is that the start and end indices of the respective loaded dice stages, are the same. The actual dice faces thrown will be different as the algorithm throws the dice based upon a probability. The same rough pattern of sixes versus other dice faces should be observed with each iteration.

The output of the Monte Carlo simulation provides the following *(each output report is thus based upon 1000 iterations*):

- Total number of transitions from Fair to Loaded. This should be 2 given the design of the test.
- A false positive estimate. As mentioned above the dice throw sequence (of 100 throws) has been designed with 2 fair to loaded transitions. If the

algorithm detects more than this, then these additional edge detections represent false positives.

- The number of times (as a percentage) that the (Fixed State Test) FST model detects the fair to loaded transition within 5 throws (of the designed transition).
- The number of times (as a percentage) that the FST model detects the fair to loaded transition within 10 throws (of the designed transition).
- The number of times (as a percentage) that the FST model detects the fair to loaded transition within 15 throws (of the designed transition).
- The number of times (as a percentage) that the (Real Time Viterbi) RTV model detects the fair to loaded transition within 5 throws (of the designed transition).
- The number of times (as a percentage) that the RTV model detects the fair to loaded transition within 10 throws (of the designed transition).
- The number of times (as a percentage) that the RTV model detects the fair to loaded transition within 15 throws (of the designed transition).

4.3.1 Real Time Model Performance Results

The results in Table 4.1 and in Table 4.2 summarise the outputs of the Monte Carlo runs of the Fixed State Test (FST) and Real Time Viterbi (RTV) models respectively for varying buffer length and loaded six probability. Each Monte Carlo run comprises of 1000 iterations, with each iteration consisting of a sequence of 100 dice throws.

Referring to Table 4.1 and in Table 4.2 the following points apply to both tables:

- The FST FALSE POS% column is a performance measure as to the number of times the model reports a loaded dice when in fact a fair dice was in use. A false positive percentage of, say, 100% means that there were, on average, 4 fair to loaded dice transitions per iteration where 2 were true detections and 2 where false positives. There are only 2 actual transitions (from fair to loaded dice states) per iteration which is why the TOTAL TRANS column is consistently populated with the number: 2000 (2 fair to loaded transitions per run with 1000 runs in total).
- The FST WITHIN 5%, FST WITHIN 10% and FST WITHIN 15% columns are also performance measures but are more an indication of the latency

performance of the respective models The FST WITHIN 5% column details how many times (as a percent of the 2000 transitions from fair to loaded in any Monte Carlo run) the respective models detected the transition (fair to loaded dice) within 5 dice throws. The FST WITHIN 10% and FST WITHIN 15% columns provide similar information but report on a detection window (the number of throws after the actual transition) of 10 and 15 throws respectively.

Table 4.1 Monte Carlo Results for Loaded Dice Simulation for FST Algorithm

DATETIME	ITERATION	BUFFER LENGTH	LOADED SIX PROB%	TOTAL TRANS	FST FALSE POS%	FST WITHIN 5%	FST WITHIN 10%	FST WITHIN 15%
2012/11/21 23:48	1000	5	50	2000	58.75	64.45	24.45	7.5
2012/11/21 17:44	**1000**	**8**	**50**	**2000**	**8.5**	**48.4**	**32.85**	**9.9**
2012/11/20 21:58	1000	13	50	2000	0.15	16.65	36.7	20.7
2012/11/20 13:34	1000	18	50	2000	0	4.1	20.05	29.2
2012/11/20 00:50	1000	5	90	2000	22.65	99.2	0.75	0
2012/11/19 15:59	**1000**	**8**	**90**	**2000**	**4.1**	**92.7**	**3**	**0.35**
2012/11/19 21:47	1000	13	90	2000	0	69.65	16.95	0
2012/11/20 07:40	1000	18	90	2000	0	25.05	48.7	0.6

Referring to Table 4.1 the following applies:

- The False Positive Rate increases with a decrease in buffer length.
- The False Positive Rate also increases with a lower probability of a loaded six dice face being thrown. The reason for this is that the fair to loaded boundaries would be more unclear as the number six appears less often with loaded dice. For example a dice sequence of (1,3,4,5,6,5,3,6,6,6,6,6,6,6, 6,6,1,2,5) provides a much more obvious fair/loaded interface than a sequence of (1,3,4,5,6,5,3,6,5,6,4,6,4,6,1,6,1,2,5).
- As the Buffer Length increases, the latency increases. For example (3rd row from the bottom) with a buffer length of 8 (with Loaded Six Prob% = 90%) 92.7% of the model outputs (detecting the fair to loaded dice transition) were detected within 5 throws of the actual dice change (to loaded). Increasing the buffer length to 13 (refer to the 2nd row from the bottom) significantly reduces the detections (within 5 throws) to 69.65%.
- Row 2 and row 6 (the rows in bold) highlight the Buffer Length that provides the best model performance for the respective biased six probability of 50% and 90%.

Table 4.2 Monte Carlo Results for Loaded Dice Simulation for RTV Algorithm

DATETIME	ITERATION	BUFFER LENGTH	LOADED SIX PROB%	TOTAL TRANS	RTV FALSE POS%	RTV WITHIN 5%	RTV WITHIN 10%	RTV WITHIN 15%
2012/11/21 23:48	1000	5	50	2000	130.2	62.05	26.85	8
2012/11/21 17:44	1000	8	50	2000	72.3	29.5	49.4	11.8
2012/11/20 21:58	**1000**	**13**	**50**	**2000**	**21.25**	**10.1**	**18.9**	**45.65**
2012/11/20 13:34	1000	18	50	2000	7.55	4.75	6.35	17.35
2012/11/20 00:50	1000	5	90	2000	74.25	97.3	1.55	0.1
2012/11/19 15:59	1000	8	90	2000	37.7	40.2	57.9	0.55
2012/11/19 21:47	**1000**	**13**	**90**	**2000**	**7.65**	**14.4**	**24.05**	**57.85**
2012/11/20 07:40	1000	18	90	2000	1.5	6.85	6.25	24.5

Referring to Table 4.2, a similar pattern of model behaviour is observed except that the False Positive rate is significantly higher than that of the FST model. In addition this model suffers from a higher latency (as indicated by the lower values in the RTV WITHIN 5% column for example) than the former.

Note that row 3 and row 7 (the rows in bold) highlight the Buffer Length that provides the best model performance for the respective biased six probability of 50% and 90%. In the RTV model a longer buffer length (13) provides the best performance compared with a buffer length of (8) in the FST algorithm.

4.4 Summary

This chapter discussed the use of Hidden Markov Models in real time. As discussed above, the main problem with the Viterbi algorithm is that it requires the full observation sequence to be present before it can output the corresponding hidden states. In real time the observation sequence is continuous rendering the original Viterbi algorithm unsuitable for this purpose.

In order to adapt the Viterbi algorithm for use in real time, a number of modifications were researched and presented. One such modification, labelled the Real Time Viterbi (RTV), was coded and tested against an algorithm developed by the Author (known as the Fixed State Test (FST) algorithm). The same Fair/Loaded dice HMM (as used in the previous chapter) was utilised in this chapter. The respective RTV and FST algorithms were then engaged to detect the Fair/Loaded states of the dice as each new dice face was rolled.
A Monte Carlo simulation was conducted to determine which algorithm detects the Fair to Loaded dice transitions sooner in real time while simultaneously minimising the number of false positives (dice state transitions that were not genuine). The performance indicators were thus the delay (latency) between the actual event occurring and the time the algorithm detected it as well as the number of false positives output by the respective algorithms.

From the results in Section 4.3.1, is it clear that the FST (Fixed State Test) algorithm is superior in all performance aspects (false positive rate and latency) to the Real Time Viterbi (RTV) algorithm.
As such the FST algorithm was chosen as the detection algorithm, to be used in conjunction with a Hidden Markov Model, to answer the research question.

CHAPTER 5 HMM FURNACE FROTH DETECTION

In this chapter conclusions from previous chapters are applied to solve a real world engineering problem. In order to protect the confidentiality of a client, no direct references to the identity of the client are made and, in some cases, process specifics have been withheld. This chapter introduces the problem but focuses on the design of the Hidden Markov Model Froth Detector and discusses the typical model outputs.

5.1 Process Description

The furnaces on which this model was applied are AC arc furnaces (total furnace powers in excess of 70MW) that smelt slag and metal. One of the by-products of this smelting process is CO Gas. When the slag layer is not sufficiently viscous the CO gas becomes trapped in this layer resulting in its volume increasing rapidly. The electrodes (6 in a 3 phase star formation) rise to maintain the active power draw of the respective phase to setpoint - power control is achieved by controlling phase current in this case.
As there is limited travel, the electrodes can only rise to a maximum limit at which point the secondary phase current draw increases until the phases trip on over-current (currents in the order of 30kA per phase).
The result is a loss of production as well as wear and tear on the high voltage switchgear.

The only real indicator of a froth is the upward travel of the electrodes. Apart from the many other reasons that could result in the electrodes rising, the nature of the system is such that the arcing process (surface bath arcing) is very unstable. As such the electrodes are continuously rising and falling as the power controller regulates the power being supplied to each phase. Thus even in stable operation there is a fair risk of detecting false positives.
Thus to detect a furnace froth state, a soft sensor is required to discriminate between normal furnace operation and the frothing state based only the movement of the 6 electrodes.

Early detection of a froth allows the opportunity to initiate froth suppression procedures which, if initiated early enough, can prevent a scenario whereby a phase (or all of them) trips on over-current. The time constant of the system is in the order of a few minutes which means that there is not much time available between froth detection and froth suppression respectively. Taking control action a few seconds earlier can make all the difference between tripping on over current and suppressing the froth.

What makes this particular process unique is that the Author has recently developed a soft sensor to detect froths on the same furnace. The soft sensor utilises a hybrid fuzzy model to detect froths and the system has been allowed to take autonomous control action, for the last year, to suppress these froths.

Figure 5.1 provides an example output of this system and a furnace froth. As can be observed, the froth builds up until such point that the detector detects this event (vertical black line) and then immediately proceeds to suppress the froth (vertical red line). The electrodes (*their respective positions as a function of time are plotted on the brown purple, cyan, dark blue, yellow and pink traces*) hit the upper limit and plateau at maximum travel limit until such time that the suppression procedures have taken effect and the process recovers.

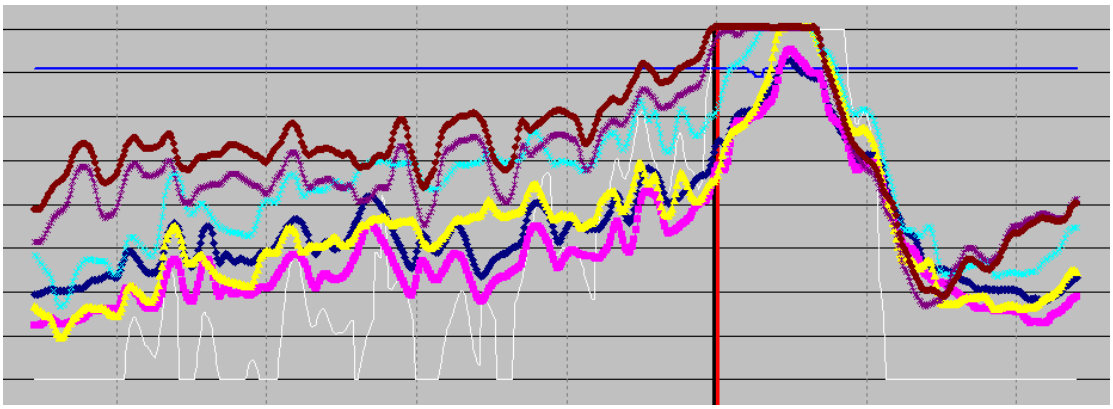

Figure 5.1 A Froth Detection with Successful Automatic Suppression

In some cases, the froth is detected too late or automatic suppression takes too long in which case, a phase may trip on over current. **Figure 5.2** provides a graphical example of when a froth was detected but the automatic froth suppression procedures failed to prevent at least one phase from tripping on over current (the stepped blue plot is the total furnace power setpoint which reduces soon after the

froth detection event indicating a phase trip). It should be noted that the other 2 phases recovered from the froth and continued to operate normally thereafter.

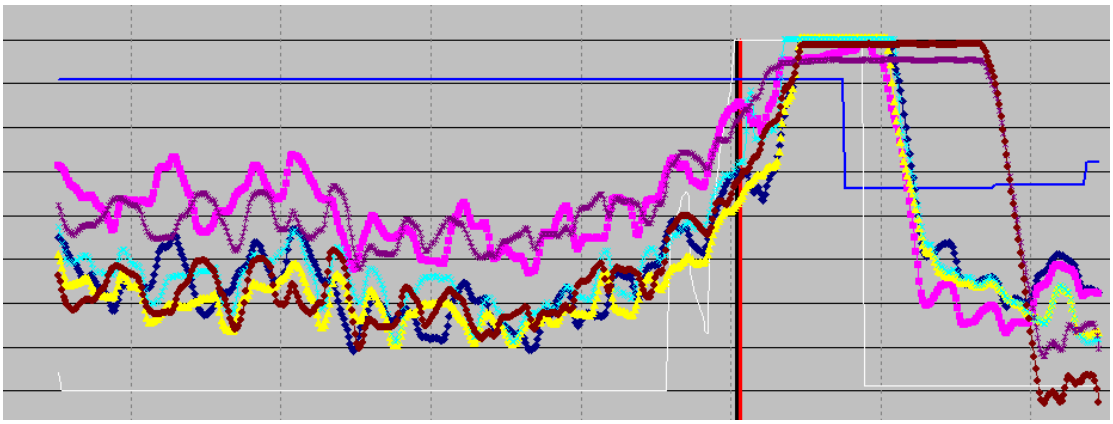

Figure 5.2 A Froth Detection with a Partially Successful Automatic Suppression

The existing soft sensor thus provides a base with which to compare the HMM algorithm's performance (although the aim of this dissertation is to prove that an HMM can detect froths and not to improve upon the above mentioned soft sensor). More importantly the existing system provides access to 100's of past froth events (neatly packaged) that can be played (in real time) into the new HMM algorithm to answer the research question.

Although the above algorithm enjoys a low false positive rate, it does suffer from false positives during unstable furnace operation which is one area that the HMM model attempts to improve upon.

5.1.1 Furnace Power Control and Electrode Positions

One of the main reasons for the need for this type of a soft sensor is that, in any furnace, there is always a limited volume available for the froth to expand. A common analogy of a froth is that of milk boiling in a pot on a stove. A shallow pot will require more input to stop the frothing milk from boiling over while a deep pot will provide more time to observe whether the froth is serious enough to warrant any input action (to prevent the milk boiling over). Preventative action in this case is to immediately remove the heat or move the pot off the stove.

In electric arc furnaces, the same applies:

- The depth of the pot is synonymous with the maximum electrode travel length (the maximum length the electrodes can continue arcing, while still under automatic power control, before tripping the power). This travel length can be roughly anywhere between a few hundred millimetres to nearly 2 metres depending on the type of furnace. In a submerged arc nickel-copper furnace, the maximum electrode travel is reported as 1.22m by Mostert and Roberts (1973:293). Additionally Boulet, Vaculik and Wong (1997:6) provided results on a non-ferrous, 3 electrode, arc furnace controller that regulates electrodes between approximately 400mm to 700mm. As discussed above, the greater this travel distance, the less urgent control action will be in the event of a furnace froth. If furnaces were built with very high freeboards, then a froth detector would be entirely superfluous.
- The speed (in millimetres per minute) of electrode upward movement is a good observation metric to determine whether any input (control action) is required to suppress the froth. This upward speed (when the furnace power is still under automatic control) is usually a large fraction of the maximum upward electrode travel rate. This speed can be anywhere between tens of millimetres per minute to a few thousand millimetres per minute depending on the type of furnace. Panoiu, Panoiu, Sora and Iordan (2008: 1632) reported electrode speeds of between 300mm/min up to 2100mm/min in their 73MVA Electric Arc Furnace control application.
- In the current application, the maximum electrode travel is limited to 900mm and the upward speed during a froth (under automatic control) is between 50mm/min and 100mm/min.

Despite the above mentioned variances in furnace type (AC vs DC), material type (steel, non-ferrous, chrome etc), electrode configurations (6 in line, 3 triangular), arc positions (surface vs submerged), electrode travel limits and electrode speeds, the algorithm developed (for this research dissertation) to detect furnace froths can be utilised (with minor adjustments to its configuration) with the main rider being that the electrode behaviour must be such that in a froth condition they rise.

5.2 Vector Quantisation

5.2.1 Introduction to Vector Quantisation

One of the main differences between a real world process and the real time fair/loaded dice example in CHAPTER 4 is that the observations are continuous plant variables and are no longer discrete symbols (eg dice faces). This means that an additional stage is required to convert the six electrode positions (or the state that they are in) into some form of symbolic representation. The concept of a codeword is one that needs to be introduced whereby a codeword represents one element of a dictionary of n words that contains all the possible observation symbols in a Hidden Markov Model (HMM). A codeword is typically a vector (of p dimensions) that best represents a sub-region of continuous data around its Euclidean distance to all the raw vectors contained within this space - finding clusters of data in a scatter plot is a typical use for Vector Quantisation. Ramachandran (1999: 4) proposed a number of Vector Quantization design algorithms that work with a concept called distortion (a measure of the difference between the codeword and original vector to be sampled). These algorithms include the Linde-Buzo-Gray as well as the Lloyd-Max Quantizers and aim to sample and compress data in preparation for lower bandwidth data transmission. In the field of digital communications the distortion measure becomes important as the encoded and compressed vector has to be re-created at the decoder.

Savage, Marquez and Lepe-casillas (2005: 3) proposed using the Linde-Buzo-Gray vector quantization algorithm in identifying centroids which map a robot position based upon sonar readings (encoded into a vector). The sonar input vectors are then assigned to the closest centroid which becomes the current observation input into the Hidden Markov Model. A sequence of observations is then input into the HMM to find regions (of 2 dimension space) in which the robot is located. These regions are thus the hidden states of the HMM.

The above mentioned references point to different applications of Vector Quantization (VQ). In the case of Ramachandran (1999: 4) the VQ algorithm is compressing data which needs to be decoded once transmitted. In the context of this process, there is no decoding process which means a simpler algorithm can be used to extract symbols from real time vectors. The k-means clustering algorithm was thus investigated and found to be simple, efficient and most importantly, guaranteed to

converge. This algorithm, as described by Teknomo (2006: Numerical Example of K-Means Clustering), is summarised in the following procedure:

- Choose the number of clusters or codewords: N. Note that each codeword is a vector of i elements.
- Select N random codewords. Many improvements to this algorithm focus on this process to prevent the algorithm from converging to a local optimum. Arthur and Vassilvitskii (2006: 1) reported on utilising an initial random seeding algorithm to address this issue in their K-Means++ algorithm which was adopted for specific scenarios in the Froth Detector algorithm.
- After the first iteration compute the Euclidean distance, as in **Equation (44)**, between the new vector and every codeword.
- Compute a new set of codewords based upon the centroid, see **Equation (45)**, of all the vectors deemed to be part of this cluster.
- The termination condition is based upon whether the centroid of the new codeword changes within some tolerance.
- The collection of codewords, once the algorithm has identified all the relevant centroids, is known as the codebook.

The Euclidean distance is calculated as follows:

$$E_d(x, y_i) = \sqrt{\sum_{j=1}^{k} (x_j - y_{ij})^2} \tag{44}$$

Where:

- x represents the current input vector with k elements.
- y_i represents the codeword (or centroid) that the input vector is being tested against.
- x_j is thus the j^{th} element of the input vector x.
- y_{ij} is the j^{th} element of the codeword/centroid vector $y_{i.}$

The Centroid is calculated (each element of the vector at a time) as follows:

$$y_i = \frac{1}{M} \sum_{j=1}^{M} x_{ij} \tag{45}$$

Where:

- y_i represents the i^{th} element of the new codeword calculated.
- M is the number of vectors assigned to the cluster (the number of vectors contributing to the current codeword).
- x_{ij} is the i^{th} element of the j^{th} vector in the cluster.

For vectors of one dimension the centroid calculation reduces to

$$y = \frac{1}{M}(x_1 + x_2 + x_3 + ... + x_M) \quad (46)$$

Teknomo (2006: Weakness of K-Mean Clustering) indicated a number of weaknesses of the K-Means Clustering algorithm. The main weakness that could potentially affect this research is that of the sensitivity of the clustering behaviour to the initial centroids (codewords). For example, choosing initial codewords too close together could result in poor clustering - which could result in the algorithm calculating local optima. The other weakness is the requirement to choose the number of centroids upfront. The Linde-Buzo-Gray as described in Ramachandran (1999: 4) overcomes this weakness by adjusting the codebook size to minimise distortion.

Despite the above mentioned weaknesses, the K-Means algorithm still remains the algorithm chosen for this research dissertation due to its simplicity and speed.

5.2.2 Real Time Considerations for Vector Quantisation

Since Vector Quantisation (VQ) is heavily used as a compression algorithm in the digital communications field, there is the understanding that all of the data (vectors) is usually present before transmission.

In the context of this research dissertation, one of the main issues is that of having incomplete data available in real time. This means that as time progresses, new data may arrive that can change the model.

Referring to the K-Means clustering model, the real time consideration applies to the training of the model - the part whereby the codewords are calculated from raw data to provide the possible observation symbols for the HMM model. This specifically

applies in the case of Infinite Training - where every new vector contributes to the clustering as well as to the state detection of the model.

In the case of infinite training, the real time complexity arises where the Centroid calculation is required to store all of the vectors contributing to that cluster. The large amount of data that needs to be stored as well as the computational overhead of averaging over so many vectors prompted the Author to develop a recursive modification to the centroid calculation described in **Equation (45)**. Research on this topic (of reducing real time VQ complexity) yielded a paper by (Chen, Koh, Soon 1998: 99) who proposed three approaches to reduce the real time complexity of various aspects of vector quantisers. Their main approach was to reduce the number of input vectors considered for the centroid calculation step using a pre-processing stage.

As mentioned above, a different approach (for minimising the data storage requirements and computational overhead) has been developed for this dissertation. This approach revolves around the centroid calculation and proposes that every element of a centroid be calculated in a differential format that makes use of the prior centroid.

The proposed modified algorithm for calculating the i^{th} element: $C(i)_N$ for the centroid calculated at time t = N, is as follows:

$$C(i)_N = \frac{N-1}{N} C(i)_{N-1} + \frac{x(i)_N}{N} \tag{47}$$

The derivation of **Equation (47)** can found in APPENDIX C.

The Author was unable to find any articles or publications using this methodology to calculate the new centroid.

The implications of **Equation (47)** are as follows:

- No storage of the input vector set is required.
- The calculation requires no iterating over a large set of vector elements, as in **Equation (45)**, and should therefore be less computationally intensive.

5.2.3 A Vector Quantisation Example

To illustrate the performance of a K-Means Vector Quantisation (VQ) algorithm a test system, for formulating the codebook (training the model) has been created with the following parameters:

- The number of elements in each input vector is 2 (a 2 axis co-ordinate system).
- The values of each element in each vector are limited to [0,1]. Visually this represents a two axis system with all x-values between 0 and 1 and all y-values also between 0 and 1.
- The input vectors have been created in such a way that they create two clusters - see black rectangles in **Figure 5.3** for an example.
- The initial centroids are calculated randomly within the above constraints: [0.1] for both x and y axes respectively.
- In each simulation, 5000 input vectors are presented to the Vector Quantisation algorithm (K-Means).
- The number of codewords or centroids (N) is varied to illustrate the various concepts.
- A codeword is deemed to have converged when the average change over all the elements between the last centroid and the new centroid are less than 0.01%. A codeword is also deemed unconverged if less than 100 input vectors are assigned to it.

In the VQ example in **Figure 5.3** the number of codewords has been chosen as 4 (the yellow squares represent the co-ordinates of the initial codewords). As can be observed only 2 (the red squares represent converged codewords) out of the 4 Codewords actually converge.

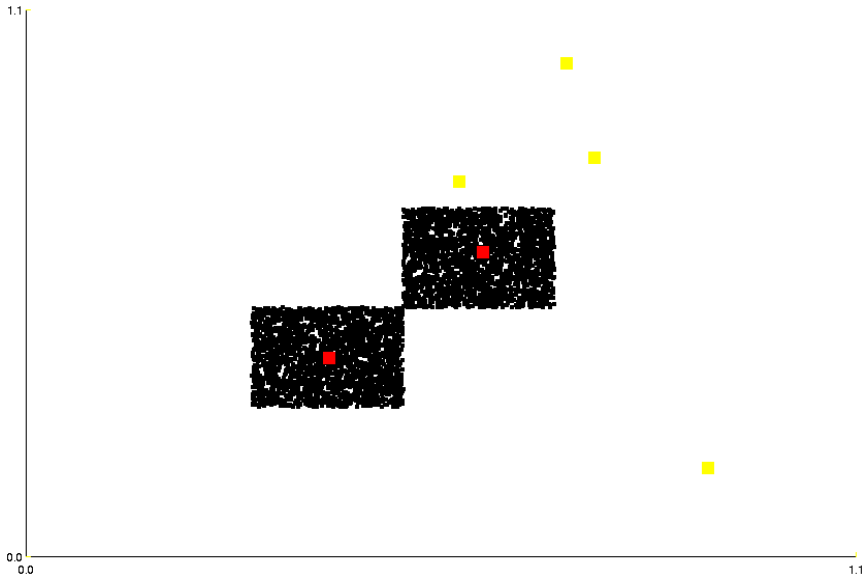

Figure 5.3 VQ Example With N = 4

In the VQ example in **Figure 5.4** the number of codewords has been chosen as 10. As can be observed 4 converged codewords result. Note that theoretically all 10 codewords can converge.

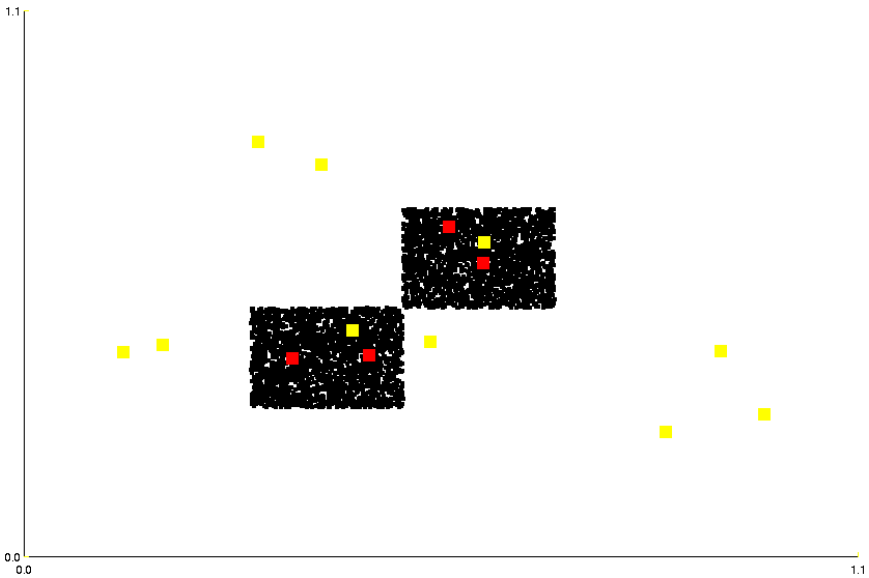

Figure 5.4 VQ Example With N = 10

When considering the above examples and the K-Means algorithm, the following summarises some of the lessons learned when developing a practical algorithm:

- **Unconverged Vectors**. The algorithm automatically deems a code word as unconverged when it has less than 100 input vectors assigned. These codewords are then removed from the code word dictionary.
- **Codeword Movement**. Not much attention is paid to this issue in the literature surveyed. As such, an algorithm developed for this purpose is described in **Equation (48)** which calculates an average percentage change of each element (from the old to the new centroid/codeword) in the respective centroid.
- **Codeword Convergence**. Following from the above point, any absolute value in **Equation (48)** less than 0.01% is thus deemed a converged codeword and no further processing is expended on this codeword in the training mode.

$$AveChange\% = \frac{1}{k}\sum_{i=1}^{k}\left[\frac{\Delta_i}{y'_i}\right]*100 \quad (48)$$

Where:

- Δ_i = y_i - y'_i (the new value of centroid element **i** minus the old value for **i**)
- k = the total number of elements in each centroid vector.

5.3 A Note on False Positives

Based upon a number of years of the Author's professional experience in developing real time soft sensors, in the process control arena, the main criterion that determines the success of any soft sensor is that of its false positive rate - in any model a low false positive rate is preferable.

A false positive, in the context of this research, is one where the model outputs a detected state that is not true. In Section 4.2, Qualitative Real Time Algorithm Performance, the concept of a false positive was introduced (albeit without explanation). Referring to the Loaded Dice model above, a false positive would be the scenario whereby the model detected the loaded dice state when, in reality, a fair dice was in use. Conversely, the false positive label would also apply to where the model deemed the dice to be fair when in fact it was loaded.

In real world applications, a model would usually be developed to detect an abnormal process state. A model output that deems the process to be in a normal state when it was in the abnormal state is usually less severe than one where it deems the process to be in the abnormal state when it is really in the normal state. The above comparison could possibly be labelled as a comparison between a false negative versus a false positive. In the Author's professional experience it is often more important for the model to not output false positives than it is to successfully detect true positives as confidence is quickly eroded in any soft sensor producing false positives. It is thus preferable for the model to detect fewer genuine abnormal conditions than outputting more false positives.

In some cases systems have been disabled after literally a dozen false positives have warned plant personnel of abnormal states that were not actually present. This example was for a soft sensor utilised for decision support only. In the case of soft sensors that will eventually take control action based upon their outputs, a very low false positive rate is ultimately required (approximately 10% is a tolerable ball park figure).

With reference to the above discussion, the HMM model designed for the real world application has had as much emphasis placed upon minimising false positives as that of successfully detecting the furnace froth state.

5.4 Design Of The HMM Froth Detector Algorithm

The basic algorithms for the Vector Quantization (VQ) and Hidden Markov Model Fixed State Test (HMM - FST) are described in flow charts in **Figure 5.5**. Each stage in the algorithm is numbered in brackets and will be discussed below.

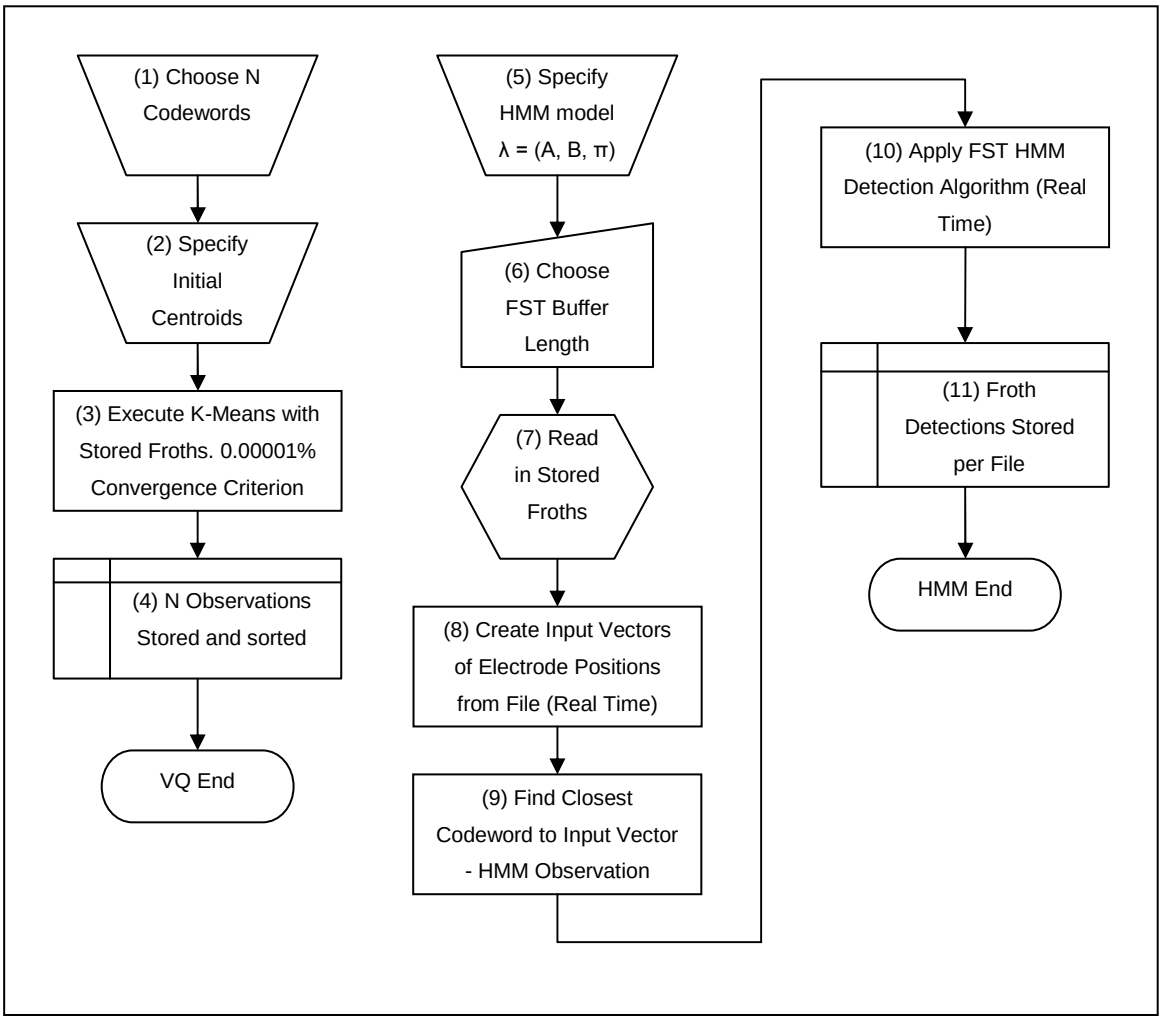

Figure 5.5 VQ and HMM Algorithm Flow Charts

5.4.1 Stage (1) and (2) Codeword Initialisation

The very first stage of the Froth Detector algorithm is to develop the codebook (as described in Section 5.2.1). The codebook, in the HMM context, is a collection of possible observations that can be presented to the model.

In the context of the furnace froth detector, the structure of the codeword is a vector of 7 elements. The last six elements are the electrode positions (electrode 1 to 6) while the first element is the average electrode rate of change over the last 5 minutes. This rate of change was calculated as follows:

- At each new datapoint 6 electrode positions are collected.
- The average electrode position is then calculated and stored for the last 5 minutes - relative to the time of the current datapoint.
- For every datapoint collected within this 5 minute window the difference or change in consecutive values is then calculated and totalled.
- This sum (of all the delta values) is then divided by the total number of stored values to finally provide the average rate of change. Algebraically this algorithm is described in **Equation (49)**

$$AveRateofChange = \frac{1}{k}\sum_{i=1}^{k} \Delta EP_{ave\,i} \tag{49}$$

Where:

- ΔEP_{avei} refers to the i^{th} value of the stored delta value (the difference between the i^{th} value and the value as of 1 datapoint before this) of the average electrode position.
- k refers to the number of stored delta values within the prescribed time period (5 minutes in this case).
- The final rate of change then has to be scaled (divided) according to the interval between datapoints (2 seconds in this case) and again to obtain a rate per minute.
- The final rate will be of units: mm per minute or % per minute.

In order to produce the codewords, 100 saved data files of past froths were re-played into the K-Means Clustering algorithm. It is important to note that each of the 100 froth files were chosen based upon a "genuine froth rating". This means that only obvious and genuine froths (like the ones depicted in **Figure 5.1** or in **Figure 5.2**) were chosen and no marginal froth trends were included. Each froth file contains about 20 minutes worth of electrode position data with a 2 second resolution (600 input vectors per file). This means that approximately 60000 input vectors contributed to forming the codebook.

In Section 5.2.1, a reference was made to previous work by Arthur and Vassilvitskii (2006: 1) who proposed initial seeding of the initial centroids in order to improve clustering. This concept was incorporated into the K-Means algorithm developed for use in this research project. The initial centroids were thus chosen with the following considerations:

- From past experience it is known that a froth has almost certainly developed when most of the furnaces electrodes are at approximately 95% of their maximum upward travel (over 850mm) and are still rising at a rate of approximately 50mm per minute. At this point it will be too late to take any control action as the electrodes are too close to their upper limits.
- As such the first initial centroid was chosen with as much early warning space as possible and is as follows (110, 650, 650 ,650, 650 ,650 ,650)
- Another 5 initial centroids/codewords were chosen with their electrode positions simultaneously decreasing by 50mm and the rise rates decreasing in logical decrements of between 50mm to 100mm. For example, the 3rd initial vector was thus (10, 500, 500 ,500, 500, 500, 500). The idea here is to use the rate of change (1st element) as a partial discriminator between the frothing states and the intermediate states.
- 6 codewords were thus chosen (and by implication 6 HMM Observations) for this system. The choice was based upon a balance between codebook diversity and limiting the HMM observations to a reasonable number. It should be noted that more HMM observations result in greater HMM computational complexity.

5.4.2 Stage (3) K-Means Clustering Algorithm Execution

The next stage of the algorithm is to execute the K-Means clustering algorithm. As mentioned above, the number of codewords was chosen at 6. Thereafter initial values for the respective centroids were chosen. These initial values draw on previous experience with the system and aim to partition the codewords as follows:

- Two clusters of where there is definitely no froth indicated. The input vectors in these two clusters will typically be ones were the electrode positions are very low and ones where the average electrode rate of change is also low. These two codewords will be the main HMM observations that decode the No-Froth HMM hidden state.
- Another two clusters that are positioned in some middle ground space. This means that the electrode positions will be half of maximum travel and the average electrode rate of change will also be specified as half of maximum. The corresponding observations (in an HMM context) will play a mediation type role and will mainly be used to flag the No-Froth HMM state as well as the Poss-Froth HMM state.
- A final set of two clusters of where a furnace froth is easily observable. The input vectors in these clusters will have a high rising rate of average electrode travel as well as higher electrode positions. The most obvious electrode positions that indicate a furnace froth are those that are on the upper limit of travel. The aim of these two clusters is to provide a grouping of vectors that indicates a high probability of a froth but still provides time for froth suppression to prevent a phase trip on over current. The resulting observations from these two clusters (or codewords) play the role of flagging the Def-Froth (definite froth) HMM hidden state.
- Thus X, the vector of possible states, is: X = {No-Froth, Poss-Froth, Def-Froth}

As mentioned above, past data of 100 clear cut frothing events was re-played into the K-Means algorithm to train the clustering model to ultimately provide a codebook. As mentioned above, 60000 input vectors contributed to developing the final codewords.

In order to allow each centroid to contain a sufficient number of contributing vectors, the convergence percent (introduced in **Equation (48)**) was set at 0.000005%. A centroid is deemed converged if a new input vector moves or adjusts the new centroid by less than the stipulated convergence percent.
The above value (0.000005%) was obtained by trial and error. A value too low will result in less codewords actually converging. Too high a convergence percent will result in less vectors influencing the final codeword (centroid).
With the convergence criterion of 0.000005% specified, a few thousand input vectors contributed to each codeword/centroid and all 6 clusters converged.

Finally it is important to note that the first 5 minutes of every replayed froth file do not contribute to the clustering algorithm. The reason for this is due to the need for the average electrode position rate of change to collect data for the initial 5 minutes of each replayed file. While 5 minutes of rate information is being collected, the average electrode rate of change is suppressed at 0. To include these vectors into the clustering algorithm would introduce flawed input vectors into the system as, in reality, this rate of change is very rarely 0. Once the first 5 minutes of rate information has been collected, a new average electrode position rate of change is calculated with every new datapoint to provide a new rate of change value over the last 5 minutes.

5.4.3 Stage (4) Sorting and Storing the Codebook

Once the codebook has been compiled and there are no unconverged vectors, the final task, with respect to the Vector Quantization (VQ) algorithm, is to convert the codewords in the codebook into observations suitable for use in the Hidden Markov Model (HMM).
As mentioned above, the reason for the VQ stage is to create a discrete number of observations that can be used in an HMM model. As mentioned in Section 5.4.2, 6 codewords were chosen for the Froth Detector algorithm. In this stage, the requirement is to label the codewords for use as observations and sort the code vectors into some order.

The sorting procedure is thus as follows:

- A vector (V_{max}) was firstly created with the absolute maximum limits. This vector contained the maximum average rate of change of the electrode position as well as the maximum travel of the 6 respective electrode positions. Specifically, $V_{max} = (ROC_{max}, EP1_{max}, EP2_{max}, EP3_{max}, EP4_{max}, EP5_{max}, EP6_{max})$. In the case of a 3 electrode furnace this vector would become: $V_{max} = (ROC_{max}, EP1_{max}, EP2_{max}, EP3_{max})$ - apart from tuning the HMM matrices, this would be the only change required to adapt the algorithm to a 3 electrode furnace.
- The 6 codewords were then sorted by their Euclidean distance from V_{max} in ascending order (*smallest Euclidean distance to highest*). The vector closest to V_{max} is thus the first vector in the ordered vector sequence.
- Each codeword in the ordered codebook sequence was provided a simple symbolic name: O_1 to O_6. These names now identify the HMM observations and provide the set of possible HMM observations, labelled as V. Formally $V = \{O_1, O_2, O_3, O_4, O_5, O_6\}$
- Referring to the Fair/Loaded dice example in Section 0, the above mentioned observation set V is synonymous to the set of dice faces: 1 to 6 which comprise the possible set of observations in the Fair/Loaded Dice HMM model. It is pure co-incidence that the number of possible observations in each model is 6.
- The Observation set V is thus the final output of the Vector Quantisation stage.

5.4.4 Stage (5) Construction Of The HMM Matrices

This stage is a once off process that requires the creation of 3 matrices that define the Hidden Markov Model (HMM), **λ**. Formally λ = (A, B, π) where A represents the state transition matrix, B represents the observation probability or emission matrix and π represents the initial state distribution matrix (vector).
The first point to note is that 3 hidden states were chosen - refer to Section 5.4.2 for more information. In HMM terminology the vector X = {No-Froth, Poss-Froth, Def-Froth} represents the set of possible states.

The hidden states are thus labelled as:

- Def-Froth (short for definite froth).
- Poss-Froth (short for possible froth).
- No-Froth.

The easiest matrix to define was the initial state distribution matrix which provides the HMM with the probability of the model being in each respective state at time t = 0.
The logical choice of this matrix was one where each state has an equal initial probability as depicted in **Equation (50)**

$$\pi = \left(1/3, 1/3, 1/3\right) \tag{50}$$

The implications of the above (equal probability) distribution are such that the probability of the model initially being in a particular hidden state is thus 0.333.

The next matrix to populate is the State Transition Matrix, A. This matrix represents the probabilities of one state transitioning to another. It was decided that the state model was NOT to be ergodic (each state can transition to all other states and itself). Instead the following state transition paths (see **Figure 5.6**) were decided upon:

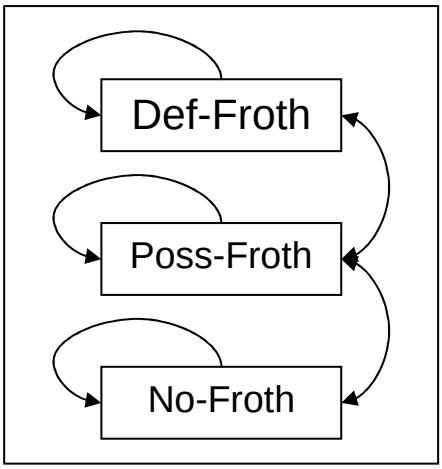

Figure 5.6 Possible State Transitions of the Froth Detector HMM

The implications of the state transitions depicted in **Figure 5.6** are that the *No-Froth* state cannot directly transition to the *Def-Froth* state without going through the intermediate state of *Poss-Froth*. The reason for this model design is to reduce the probability of a false positive triggered by a spike in the raw data (the source of which is from field instrumentation - which can attract noise).

$$A = \begin{pmatrix} 0.7 & 0.3 & 0 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.1 & 0.9 \end{pmatrix} \tag{51}$$

With respect to **Equation (51)** the following details apply:

- The elements (containing probabilities) of the State Transition Matrix are always interpreted as: a_{ij} where i (the row) represents the original state and j (the column) represents the new state. In this case, the 1st row and 1st column represent the No-Froth state. Row 2 and column 2 represent the Poss-Froth state while row 3 and column 3 refer to the Def-Froth state.
- With the above in mind, there is a 70% probability of the No-Froth state transitioning (remaining) in the same state while there is a 30% of the No-Froth state transitioning to the Poss-Froth state.

- When the model is in the Poss-Froth state there is a 25% of it transitioning to either the Def-Froth or No-Froth states respectively. In addition there is a 50% of it remaining in the same state.
- When the model is in the Def-Froth state, there is only a 10% chance of it transitioning to the Poss-State and a 90% of it remaining in the Def-Froth state.
- The reasoning behind these asymmetric probabilities is that a froth is a rare event and the system should spend most of its time in the No-Froth state. Once a Furnace Froth is initiated there is a high probability of it remaining in this state (the model expects this to be a 90% probability). The above probabilities inserted into the State Transition matrix above are thus a starting point and can be adjusted to fine tune the HMM.
- It should be noted that this matrix is row stochastic which means that only the 3 defined states must occupy the probability space at any time.

The final matrix to configure is the emission probability matrix (B). This matrix assigns probabilities of the HMM being in a specific hidden state and emitting a specific observation. **Equation (52)** provides an example of the Observation Probability matrix developed for this specific Hidden Markov Model (HMM).

$$B = \begin{pmatrix} 0 & 0 & 0.1 & 0.2 & 0.3 & 0.4 \\ 0.05 & 0.4 & 0.5 & 0.05 & 0 & 0 \\ 0.6 & 0.4 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{52}$$

With respect for the Emission Probability Matrix above, the following details apply:

- The elements (containing probabilities) of the Emission Probability Matrix are always interpreted as: b_{ij} where i (the row) represents the state (*No-Froth, Poss-Froth, Def-Froth*) and j (the column) represents the observation list: V = (q_1, q_2, q_3, q_4, q_5, q_6). Note that V is not an observation sequence but is a list of the possible observations defined within the model. Thus b_{ij} is the probability of hidden state i emitting observation j.
- The top row applies to the state: No-Froth. There is zero chance of observing q_1 and q_2 while the model is in this hidden state. The reason for this is that the q_1 and q_2 observations are respectively the closest observations to the vector with the highest average electrode rise rate and the maximum travel of all

electrodes (V_{max}). The probabilities of the model emitting observations q_3 to q_6 while in the No-Froth hidden state thus increases with the observed symbol's index.

- The middle row applies to the state: possible froth. The most probable observations while in this state are thus q_2 and q_3 while the HMM model has been configured with no chance of observing q_5 and q_6 while in this state.
- The bottom row applies to the Def-Froth state. The only observations that will be observed with the model in this state are thus q_1 and q_2 (ie the observations or codewords closest to V_{max}).
- It should be noted that this matrix is row stochastic which means that the entire probability space is covered by the 6 observations for any given hidden state at any one time.

5.4.5 Stage (6) FST Buffer Length

The Fixed State Test (FST) algorithm is a variation of the HMM Forward Algorithm and was introduced in Section 4.1.1. Using the Fair/Loaded Dice model it was shown as the most suitable algorithm to use for real time decoding of the hidden states.
This algorithm functions by evaluating the probabilities of specific state sequences of a specific length (buffer length L). The sequence returning the highest probability is thus deemed as the state sequence that best fits the sequence of observations (the observation sequence is thus the same length as the state sequence).

The state sequences that are specifically tested are simply sequences padded with each of the defined states. In the context of this HMM the state sequences tested (in real time) would comprise of the following (assuming a buffer length of L = 5 in this example):

- Q_1 = (*No-Froth, No-Froth, No-Froth, No-Froth, No-Froth*)
- Q_2 = (*Poss-Froth, Poss-Froth, Poss-Froth, Poss-Froth, Poss-Froth*)
- Q_3 = (*Def-Froth, Def-Froth, Def-Froth, Def-Froth, Def-Froth*)

Thus after each observation, the following probabilities are tested:

- $P_1(O, Q_1| \lambda)$
- $P_2(O, Q_2| \lambda)$
- $P_3(O, Q_3| \lambda)$

Where O represents a sequence of the most recently observed observations in a FILO buffer of length L (same length as the tested state sequence) and $P_1(O, Q_1| \lambda)$, for example, represents the probability of observation sequence O and state sequence Q_1 co-existing given the HMM model λ.

In order to test the performance of the Froth Detector HMM, the buffer length was varied between L = 5 and L = 10.

5.4.6 Stage (7) Reading in Stored Data in Real Time

The saved data from the plant comprises of hundreds of saved froths. Each froth file (in comma separated format) contains 20 minutes of data with a 2 second resolution. This means that there are 600 datapoints per variable per froth file. Some of the variables contained in each froth file are:

- The positions of electrodes 1 to 6. Note that these positions are filtered by an 8 second averaging filter.
- The total furnace power over the 20 minute period.
- The froth detection time as determined by the existing algorithm.
- The output of the existing froth detection algorithm.

From the available data in the above mentioned files, only the electrode positions and froth detection time (as per the existing algorithm) are used in the HMM Froth Detector.
Given the above, the following methodology was adopted with respect to reading the above data into the respective Hidden Markov Model (HMM) and Vector Quantization (VQ) algorithms:

- 100 data files were chosen to train the VQ codebook and another 100 (different) data files were also used to test the HMM detection algorithm. Each froth contained in this list (of 200 files) is one that is a definite true positive event (a genuine froth). Therefore there is no data in this list of 200 files that could be interpreted as a false positive. As mentioned in Section 5.1 a prior froth detection system was recently commissioned on the plant to detect (and autonomously suppress) furnace froths. The data collected in these 200 files has been obtained via this system and, more importantly, provides a useful benchmark for testing the HMM approach to froth detection.
- During the VQ training phase, each file is opened, stored into an internal G2 buffer and presented to the VQ algorithm. The electrode positions and average electrode rate of change are packaged into an input vector of 7 elements and input into the VQ algorithm (in order to calculate the respective centroids or data clusters). It should be noted that this is not a real time process. The VQ algorithm is only used to define clusters of data and, as

such, has no interest in the timing of the data. The VQ input vectors were thus processed in "as fast as possible" mode.

- During the HMM test phase, each file was opened and replayed into the HMM algorithm. The difference between this process and the VQ training process is that this data (the respective 6 electrode positions) was played in real time. As mentioned above, the data has a resolution of 2 seconds. This means that there is a 2 second real time wait between each data point that is presented to the HMM algorithm. The net result is that replaying each file consumes 20 minutes of time but very accurately simulates the sequences of events that played out in the original froth event. Replaying 100 froth files is thus a time consuming process taking over 33 hours.
- An important point to note is that the average electrode rate of change (in mm per minute) is a not a variable calculated previously - it has been defined and calculated exclusively for the HMM Froth Detector. In both the VQ and HMM algorithms this rate of change is calculated over the previous 5 minutes worth of data (due consideration was thus required in order to calculate a rate of change in real time versus a rate of change in "as fast as possible" mode). This means that the first 5 minutes of data is required for data gathering (specifically for loading the initial 5 minutes of the FILO buffer) and requires both the VQ and HMM algorithms to ignore this period of data for each froth file - in an online system this would only be required when the system is first booted. Fortunately this is not a problem as the actual frothing event, within each file, always occurs 13 minutes into each file (by design).

The output of this stage, is thus 6 electrode positions and their average rate of change, read in from a csv (comma separated values) file, presented to the HMM every 2 seconds.

5.4.7 Stage (8) and (9) Creating HMM Observations

At this stage of the overall algorithm, the Vector Quantization (VQ) codebook has been created along with the 6 codewords. In addition, there is previous data, stored in a text file, feeding the HMM algorithm with electrode positions every 2 seconds. Specifically, the HMM algorithm is presented with:

- 6 Electrode positions.
- An average electrode rate of change calculated over the last 5 minutes.
- The above are thus combined into a 7 element vector (every 2 seconds) and presented to the HMM algorithm.

The above mentioned vector is then sent to the Vector Quantization algorithm to test for the closest codeword to this input vector. The Euclidean distance measure, as presented in **Equation (44)**, is used to calculate the closest codeword to the new input vector. The Euclidean distance is a scalar value which is calculated for each input vector and each of the 6 codewords respectively. The codeword with the lowest Euclidean distance measure is thus deemed as the latest Observation in the HMM context. There are 6 codewords configured for the overall system which means there are 6 possible HMM observations. Every 2 seconds, one of the 6 Observations is thus presented to the HMM FST (Fixed State Test) algorithm in order to detect the current hidden state (No-Froth, Poss-Froth or Def-Froth).

5.4.8 Stage (10) Executing the HMM Fixed State Test (FST) Algorithm

As mentioned above, an observation is presented to the HMM FST algorithm every 2 seconds. The output of this particular stage encompasses the entire purpose of this research dissertation: to detect the hidden states of the current process.

The details of this algorithm can be found in Section 4.1.1 and in Section 5.4.5 (a discussion on buffer length). In short the FST algorithm is a modification of one of the main HMM algorithms, the Forward Algorithm. The main difference is that the FST algorithm has been developed for real time use and tests for specific state sequences. The state sequence with the highest probability is thus deemed as the most probable state (but not the optimal sequence as output by the Viterbi algorithm)

sequence. Since every element of the tested state sequence is the same, the current hidden state is thus deemed that same state.

In essence the FST algorithm is testing for a consistent period when the system has been in a specific state for at least the time period equal to the buffer length. The longer the buffer length, the longer the system has to be consistently in a particular state. From a latency perspective a long buffer length is not ideal as the system has to consistently reside in the Def-Froth state before the overall algorithm will flag a froth. On the other hand, a buffer length too short will result in a higher number of false positives.

5.4.9 Stage (11) Capturing Results

The final stage of the Hidden Markov Model (HMM) is the results. The results are split between the following:

- An internal G2 trend chart. This is a normal graph of the 6 electrodes positions and the average electrode rate of change plotted against the HMM output. The output of the current froth detector (previously developed for the plant) has also been included for benchmark purposes.
- An Excel report. One of the attractive features with using G2 is its capability to connect to OLE (Object Linking and Embedding) compliant applications (Visual Basic, Word, Excel etc). Since the original plant system already has such an interface to G2, an additional Excel worksheet was developed to capture the quantitative outputs of the individual froth simulations (event times, electrode positions etc)

The results introduced above thus provide a qualitative as well as a quantitative output of the HMM algorithm. An example of each will be presented below.

In **Figure 5.7** the qualitative output of the HMM FST algorithm is displayed for a specific froth - replayed into the HMM algorithm.

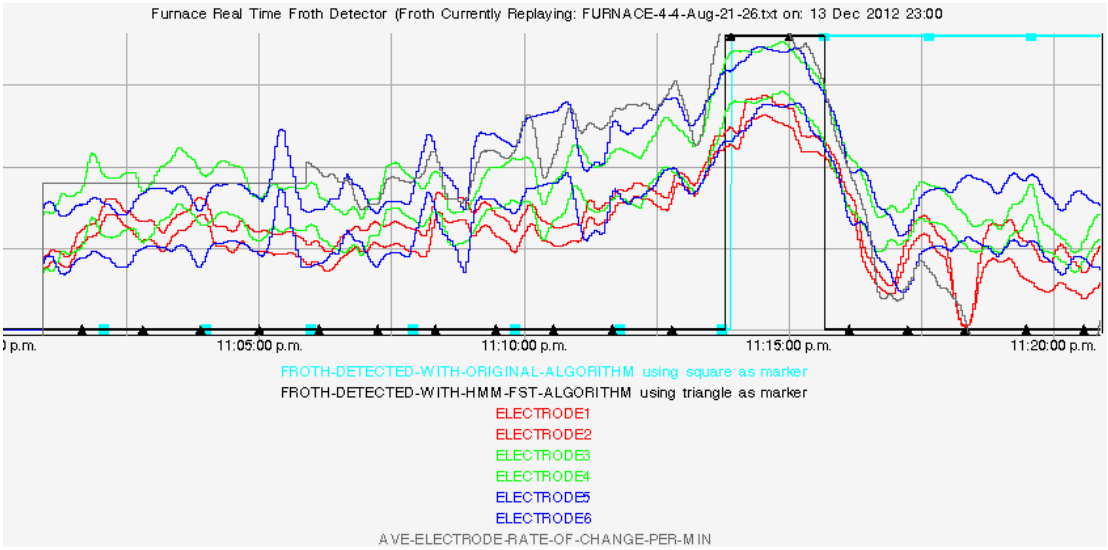

Figure 5.7 Real Time Output of the Froth Detector HMM

With respect to the plots in **Figure 5.7** the following is a (colour coded) explanation of their meanings:

- **FROTH-DETECTED-WITH-ORIGINAL-ALGORITHM**. This provides a qualitative indication of when the froth event was detected by the original froth detection algorithm.
- **FROTH-DETECTED-WITH-HMM-FST-ALGORITHM**. This provides a qualitative indication of when the froth event was detected by the new HMM FST algorithm. In this example the HMM FST algorithm detected the froth almost simultaneously to the original algorithm. Note that the leading edge of this plot is only of interest as, in an online environment, another leading edge within a minimum time of this one will be ignored.
- **ELECTRODE1**. This plot provides a real time plot of Electrode 1's position.
- **ELECTRODE2**. This plot provides a real time plot of Electrode 2's position.
- **ELECTRODE3**. This plot provides a real time plot of Electrode 3's position.
- **ELECTRODE4**. This plot provides a real time plot of Electrode 4's position.
- **ELECTRODE5**. This plot provides a real time plot of Electrode 5's position.
- **ELECTRODE6**. This plot provides a real time plot of Electrode 6's position.
- **AVE-ELECTRODE-RATE-OF-CHANGE-PER-MIN**. The plot provides a real time indication of the average electrode rate of change. It is important to note that it is zero for the first 5 minutes of every froth as it gathers FILO (first in last out) data for the rate of change calculation.

In Table 5.1 an example of the quantitative output of the HMM FST algorithm is displayed for a collection of froths (in most cases 100 froth data files are replayed in one batch - mainly after a parameter from the algorithm has been changed)

Table 5.1 Quantitative Output of the Froth Detector HMM

Froth File Index	Replay Start Time	Replayed Froth File	Old Dection Time	Old Detection Offset	New Detection Time	New Detection Offset	Buffer Length
1	12 Dec 2012 12:41:21	FURNACE-2-13-Aug-12-00.txt	13 Aug 2012 12:00:29	781	13 Aug 2012 11:59:54	746	8
2	12 Dec 2012 13:02:21	FURNACE-2-17-Jun-07-54.txt	17 Jun 2012 07:54:49	781	17 Jun 2012 07:54:40	772	8
3	12 Dec 2012 13:23:21	FURNACE-2-20-Aug-11-42.txt	20 Aug 2012 11:42:50	782	20 Aug 2012 11:43:06	798	8
4	12 Dec 2012 13:44:23	FURNACE-2-20-Aug-12-17.txt	20 Aug 2012 12:17:46	782	20 Aug 2012 12:17:52	788	8
5	12 Dec 2012 14:05:25	FURNACE-2-23-Aug-13-26.txt	23 Aug 2012 13:26:12	782	23 Aug 2012 13:26:34	804	8
6	12 Dec 2012 14:26:27	FURNACE-2-24-Sep-02-07.txt	24 Sep 2012 02:07:45	781	24 Sep 2012 02:07:38	774	8
7	12 Dec 2012 14:47:27	FURNACE-2-25-Aug-03-14.txt	25 Aug 2012 03:14:14	782	25 Aug 2012 03:14:26	794	8
8	12 Dec 2012 15:08:29	FURNACE-2-25-Aug-03-38.txt	25 Aug 2012 03:38:48	782	25 Aug 2012 03:38:54	788	8
9	12 Dec 2012 15:29:31	FURNACE-2-25-Aug-11-34.txt	25 Aug 2012 11:34:02	782	25 Aug 2012 11:34:18	798	8
10	12 Dec 2012 15:50:33	FURNACE-2-26-Aug-19-16.txt	26 Aug 2012 19:16:10	782	26 Aug 2012 19:17:30	862	8
11	12 Dec 2012 16:11:35	FURNACE-2-27-Aug-21-51.txt	27 Aug 2012 21:51:16	782	27 Aug 2012 21:51:30	796	8
12	12 Dec 2012 16:32:37	FURNACE-2-28-Jun-05-33.txt	28 Jun 2012 05:33:48	782	28 Jun 2012 05:26:10	324	8
12	12 Dec 2012 16:32:37	FURNACE-2-28-Jun-05-33.txt	28 Jun 2012 05:33:48	782	28 Jun 2012 05:38:56	1090	8
13	12 Dec 2012 16:53:39	FURNACE-2-29-Aug-05-52.txt	29 Aug 2012 05:52:54	782	29 Aug 2012 05:52:06	734	8
14	12 Dec 2012 17:14:41	FURNACE-2-29-Aug-07-15.txt	29 Aug 2012 07:15:18	782	29 Aug 2012 07:11:58	582	8
15	12 Dec 2012 17:35:43	FURNACE-2-29-Jul-19-57.txt	29 Jul 2012 19:57:34	782	29 Jul 2012 19:57:44	792	8

With respect the quantitative output in Table 5.1 the following is an explanation of the various columns:

- **Froth File Index**. This is the index of the current froth (1 to 100 in most cases). It is merely a numeric identifier.
- **Replay Start Time**. This is time (in real time) that the first value from the respective froth data file was played in to the HMM.
- **Replayed Froth File**. This column provides the name of the current froth data file being replayed.
- **Old Detection Time**. This is the detection time that the original froth was detected by the original algorithm.
- **Old Detection Offset**. This is an interval time (in seconds) that the original froth was detected (by the original algorithm), scaled to the first data point. As previously mentioned, all of the original froths will be placed 13 minutes into the 20 minute data file. Thus all of the values in this column will be 781 or 782 seconds.
- **New Detection Time**. This is the time that the new HMM FST algorithm detected the froth but scaled to the original time axis. This provides an indication as to the time that the froth would have been detected had the HMM FST algorithm been running at that time.
- **New Detection Offset**. This is an interval time (in seconds) that the froth was detected, by the new HMM FST algorithm, scaled to the first data point. This is the most important output column as it provides a detection time compared with the old algorithm. In most cases a lower number is preferable as early froth detection will most likely lead to successful froth suppression.
- **Buffer Length**. This is the current buffer length of the HMM FST algorithm.

With respect to each row in Table 5.1 it is important to note that the mere existence of the row indicates that a froth was detected for that particular replayed froth. Should a replayed froth remain undetected by the HMM FST algorithm a record (row) in this output table will not be created.

Secondly, there can be duplicate rows. A duplicate row (for the same replayed froth file) indicates that a second froth event was detected. In most of the results (discussed in the next chapter) the froths input into the HMM FST algorithm will be from genuine froths that have only one froth event per data file. As such, more than one row per replayed froth file could indicate a false positive. If this time is a few

minutes after the previous detection event then this is not classified as a false positive as an online HMM FST algorithm will be constrained to outputting a froth event (on a particular furnace) a minimum of 10 minutes apart - this interval will be used throughout the following chapters.

When considering that 100 froth data files are replayed in one simulated batch, the ideal output (of Table 5.1) is as follows (in decreasing order of desirability):

- 100 rows (records). More than this indicates that false positives were generated. Less than this indicates that the froth was not detected at all.
- The value in the New Detection Offset column for every row is equal or less than the value in the Old Detection Offset for that particular row. As mentioned on many occasions, the sooner a froth is detected, the more successful froth suppression is likely to be and the less likely the furnace will experience a phase trip on over current. A later froth time (as detected by the HMM FST algorithm) does not necessarily indicate inferior performance as some of the slower froths are detected too soon by the existing plant system - a situation potentially lending itself to a higher false positive rate.

5.5 Summary

In this chapter, the design of the final Froth Detector algorithm was discussed. One of the major design considerations becomes apparent when comparing the differences between the real time Fair/Loaded dice model and the Froth Detector. This difference is the discrete versus continuous nature of the HMM observations. In the dice example the dice faces are discrete observations. In the Froth Detector model, all plant inputs are continuous.

The above difference required the introduction of an intermediate stage to translate the process variables into observations. The concept of Vector Quantisation (VQ) was thus introduced which ultimately finds clusters (using the K-Means Clustering algorithm) or centroids of vector data while in the training mode. In the run time mode, the VQ algorithm matches up input vectors to the closest cluster using the Euclidian distance technique. Using this methodology the VQ algorithm can thus translate vectors into symbols which is the methodology the algorithm used to translate grouped plant data into observations. The first part of this chapter was thus dedicated to discussing the various Vector Quantisation considerations.

The HMM Fixed State Test (FST) algorithm was selected, in the previous chapter, as the best performing algorithm for use in real time. The rest of this chapter discussed the algorithm's flow chart and how each stage contributes to the overall soft sensor. The end user outputs of the HMM FST Detector were discussed with sample outputs presented in Section 5.4.9.

CHAPTER 6 HMM FST FROTH DETECTOR RESULTS

In this chapter the results of the HMM FST (Fixed State Test) algorithm are presented. The results are compared to the existing Hybrid Fuzzy Froth Detector algorithm (currently deployed on the plant) to gauge the performance of the detector.

6.1 Qualitative Results

As mentioned previously, with exception of the results in Section 6.4, the results in this chapter are derived from a collection of 100 genuine froth data files replayed into the HMM FST Froth Detector algorithm. In order for the HMM FST detector to be deemed reasonably effective at detecting these froth events, it should detect all 100 events as froths. In addition, the time at which it detects these events should be within close proximity to that of the original froth detector system (currently deployed on the Smelter). In some cases an HMM FST detection event that is later than the original froth event is not necessarily indicative of poor performance.

Note that Section 5.4.9 provides the descriptions of the traces in the plots presented in this chapter.

Figure 6.1 to **Figure 6.4** are thus qualitative results (presented in trends) of data files that have been replayed through the HMM FST algorithm.

In **Figure 6.1**, the HMM FST froth detector detected the froth event at almost exactly the same time as the current online system. The froth in this event was not as aggressive, compared to the other examples, as some of the electrodes did not hit their upper limits before the froth was successfully suppressed.

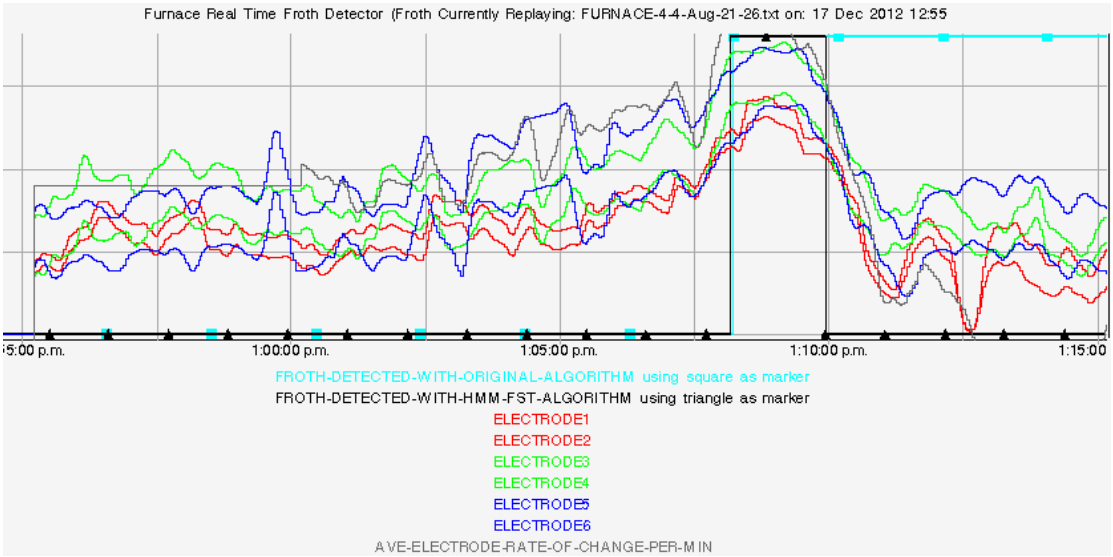

Figure 6.1 An Example of a HMM FST Froth Detection

In **Figure 6.2**, the HMM FST detector, detected the froth event slightly later than the current algorithm. This froth was particularly violent as is evidenced by the rapid electrode rise rate just before the electrodes hit their upper limits. In this case the froth suppression was only partially successful as only phase 1 (electrode 1 and 2) recovered. The other electrodes remained at their top limit which generally indicates a phase trip on over current. The problem with this type of event is that the froth evolved too quickly. A minute prior to the original detection (vertical cyan plot) the electrode movement did not display much of an upward trend - initiating suppression at this stage would have been a decision based upon luck more than experience.

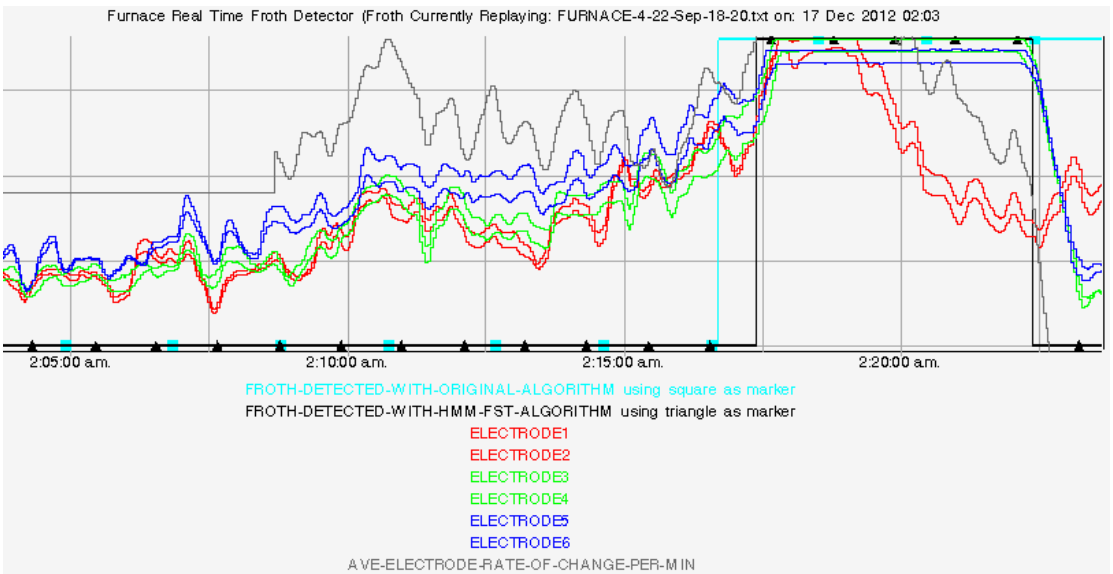

Figure 6.2 An Example of a HMM FST Froth Detection

Figure 6.3 shows a similar trend to that in **Figure 6.2** except that the HMM FST detector detected the froth sooner than the existing algorithm. Due to the aggressive nature of the froth, all 3 phases tripped on over current as a result of this event. It is possible that the ~30 second earlier HMM FST detection could have prevented one or all of the phases from tripping. The final point to note is that the electrode behaviour, literally a minute before the event, indicated no significant evidence of a froth.

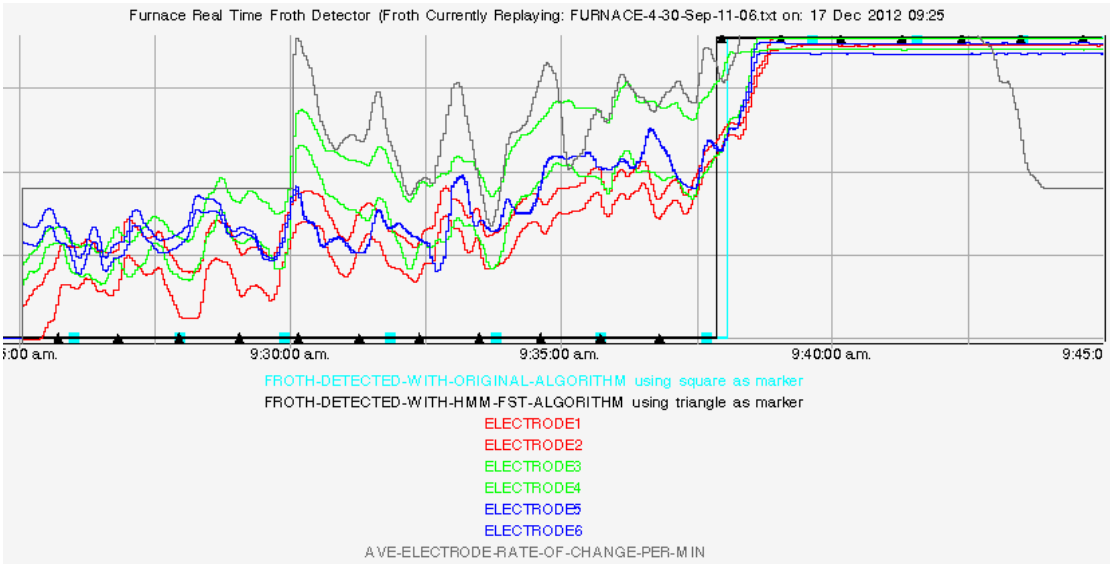

Figure 6.3 An Example of a HMM FST Froth Detection

Figure 6.4 provides another example of where the HMM FST algorithm detects the froth event significantly earlier than the existing algorithm. One of the main contributing factors to this detection is the rapid rise rate of the average electrode position (refer to the light grey plot intersecting the vertical black plot). This could be a potentially useful early detection property of the HMM FST algorithm for sustained rise rates.

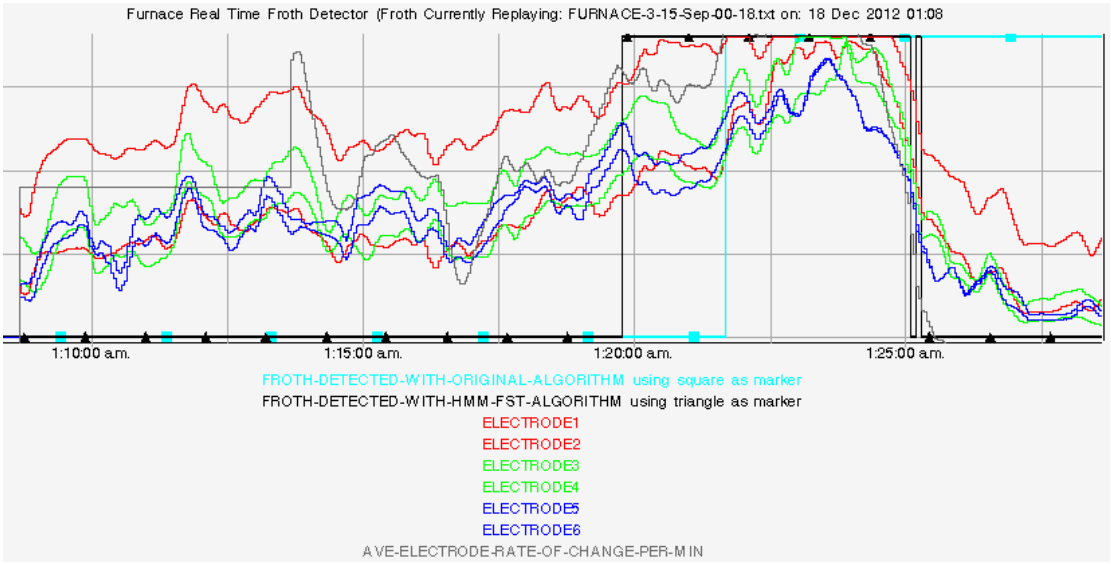

Figure 6.4 An Example of a HMM FST Froth Detection

With respect to the performance of the HMM FST algorithm in the examples above, the following are some general observations:

- The algorithm initially exhibited a low false positive detection rate which is a desired performance measure. It should be noted that a froth (as detected by the HMM FST algorithm) is only prevented from detecting a froth (rising edge only) within 5 minutes of a previous one and also for the first 5 minutes when gathering data for the average electrode rise rate. Thereafter the algorithm is free to detect froths. In **Figure 6.1** to **Figure 6.4** there are no instances whereby the algorithm triggered an unreasonable froth event. The topic of a false positive rate is discussed in detail in Section 6.4.
- The HMM FST algorithm appeared to handle electrodes that are over-slipped with respect to the others (Electrodes 1 and 2 in **Figure 6.4** provide a good example of this). From the Author's professional experience over-slipped electrodes have always created difficulty for froth detection as one of the

electrodes is always riding close to the upper limit which by definition is in the froth zone. By employing observations that contain a vector of 7 elements, this appears to mitigate the problem of over slipped electrodes due to the process whereby the Euclidean distance of the entire vector (from the maximum vector V_{max}) is obtained rather than that of the individual electrodes.

- The final observation concerns the algorithm remaining in the froth state once triggered. The algorithm prevents another froth from triggering 5 minutes after the previous one but there is nothing preventing another froth from being flagged afterwards - **Figure 6.4** is an example of this. Generally this is problematic behaviour (multiple rising and falling edges of the froth state) but, in the results presented in this dissertation, these events are a small minority of the results and usually occur after a previous genuine froth. When deploying the HMM FST algorithm online, these events can easily be filtered by prohibiting a subsequent froth detection for a time period (of 10 minutes).

In summary the above trends indicate that the HMM FST froth detector can successfully detect furnace froths. The performance of the algorithm will be quantitatively analysed in the following section.

6.2 Quantitative Results

The first test of the HMM FST Algorithm was conducted under the following conditions:

- The HMM (Hidden Markov Model) Codebook (provided in **Table 6.1**) was generated by executing the K-Means Clustering algorithm for specific initial vectors (the details on how these were specified are provided in Section 5.4.1). Column 3 (*Obs*) refers to the observation symbol assigned to the particular codeword; Column 4 (E_{dist}) refers to the codeword's respective Euclidean distance from the maximum vector (V_{max}); Column 5 (*Vectors*) lists the number of input vectors contributing to the respective codeword centroid; Columns 6 to 12 detail the contents of the respective codewords where Y_1 refers to the average electrode rate of change while Y_2 to Y_7 refer to the electrode positions.
- The State Transition Matrix (A) and Observation Probability Matrix (B) utilised in the Hidden Markov Model are specified in **Equation (51)** and **Equation (52)** respectively in Section 5.4.4.
- The HMM FST (Fixed State Test) Buffer Length utilised in this scenario was 10.
- As discussed in Section 5.4.9 above, 100 new froth data files were played into the HMM FST algorithm (note that these data files were **not** used to construct the codebook in **Table 6.1**) and refer to froths that have been previously rated as genuine froths with zero chance of them being interpreted false positives.
- Throughout this chapter a froth detected within 10 minutes of a prior froth is not considered a genuine detection nor is it considered a false positive. The reason for this is that the time constant of this type of system is between 10 to 20 minutes. In order to protect the furnace, froth suppression will only be initiated at a minimum interval of 10 minutes after a prior froth. Thus any identified froth state is "held" for 10 minutes before being reset. This rule is not built into the software and is only applied after the results.
- The only other constraint (developed into the actual detector software) is a 5 minute latch that prevents another rising edge trigger within 5 minutes of a previous detection. Together with the fact that the first 5 minutes of every data file is dedicated to gathering data for the average electrode rate of change

calculation (thus suppressing any froth detections) a maximum of 4 froth detections can be flagged per 20 minute data file. Note that out of the 460 data files replayed through the algorithm, there were two instances whereby more than 2 froth detections were observed - in this case 3 froth detections were observed (refer to Froth Index File 12 in **Table D2** and **Table D3** respectively in APPENDIX D).

Table 6.1 HMM Codebook Generated from K-Means Algorithm

Datetime	IDX	Obs	E_{dist}	Vectors	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
2012/12/26 03:17	1	O1	268.2	4043	49	789	801	815	775	768	803
2012/12/26 03:17	2	O2	718.1	5739	10	603	600	623	603	591	627
2012/12/26 03:17	3	O3	1018.9	2138	-4	529	504	471	399	471	553
2012/12/26 03:17	4	O4	1232.5	3394	-16	377	365	416	479	369	390
2012/12/26 03:17	5	O5	1296.8	3665	-26	380	317	309	344	410	492
2012/12/26 03:17	6	O6	1637.6	2545	-44	250	225	234	217	208	265

With the above codebook and respective HMM parameters, the froth data files were replayed into the HMM FST algorithm. The full table (complete with all 114 rows) is published in **Table D1** in APPENDIX D. The data in **Table 6.2** provides a sample of the first 10 data files replayed into the HMM FST froth detector algorithm. The New Detection Offset column refers to the time (in seconds) that the HMM FST algorithm detected the froth relative to the first data point of the data file.

An additional column (Delay) has been inserted to provide the delay (in seconds) between the HMM FST algorithm's detection time and the event detection time of the original algorithm currently deployed on the plant. A negative delay indicates that the HMM FST algorithm detected the froth before the original algorithm. Note that the column (Old Detection Offset) refers to the time (in seconds) that the froth was detected (using the current plant algorithm) relative to the first data point of the simulation - this number will always be 782 as the current plant algorithm always collects the last 13 minutes (780 seconds) of data prior to the froth and 7 minutes of data after the froth event.

Table 6.2 Results of First 10 Data Files Replayed into HMM FST Algorithm

Froth File Index	Replay Start Time	Old Detection Offset	New Detection Time	New Detection Offset	Delay
1	05 Feb 2013 12:45:46	781	13 Aug 2012 11:57:50	622	-159
2	05 Feb 2013 13:06:46	781	17 Jun 2012 07:53:52	724	-57
3	05 Feb 2013 13:27:46	782	20 Aug 2012 11:41:44	716	-66
3	**05 Feb 2013 13:27:46**	**782**	**20 Aug 2012 11:47:16**	**1048**	**266**
4	05 Feb 2013 13:48:48	782	20 Aug 2012 12:10:14	330	-452
4	**05 Feb 2013 13:48:48**	**782**	**20 Aug 2012 12:15:20**	**636**	**-146**
5	05 Feb 2013 14:09:50	782	23 Aug 2012 13:23:58	648	-134
6	05 Feb 2013 14:30:52	781	24 Sep 2012 02:05:44	660	-121
7	05 Feb 2013 14:51:52	782	25 Aug 2012 03:13:20	728	-54
8	05 Feb 2013 15:12:54	782	25 Aug 2012 03:37:38	712	-70
9	05 Feb 2013 15:33:56	782	25 Aug 2012 11:29:18	498	-284
10	05 Feb 2013 15:54:58	782	26 Aug 2012 19:08:22	314	-468
10	**05 Feb 2013 15:54:58**	**782**	**26 Aug 2012 19:14:04**	**656**	**-126**

Referring to **Table D1** in APPENDIX D, the following observations are relevant:

- Firstly, the HMM FST algorithm failed to detect froths (froth file index 12) in one of the replayed data files. The reasons for this relate to a software design issue - at the start of the replayed file, the initial state is already Def-Froth due to the aggressive tuning of the codebook. This state does not change throughout the 20 minutes resulting in no detection (edge trigger). In a continuous environment the Def-Froth state would have been triggered earlier than the start time of the data file.
- Given that there are 114 rows (and 1 non-detection) the algorithm produced 15 false positives in total (the bold rows in **Table D1** in APPENDIX D highlight these false positives).
- With respect to the data files that have produced more than 1 row (for example the last 2 rows in **Table 6.2**), it should be noted that the 2nd detection by the HMM FST algorithm was within 10 minutes of the first for all but 1 of the false positives (eg in the last 2 rows in **Table 6.2** the 2nd detection occurred within approximately 6 minutes of the first detection). After excluding these instances, only one genuine false positive detection (file index 13 in **Table D1** where the 2nd froth was detected 10 minutes and 6 seconds after the first) is left for this scenario.

- The average delay (after excluding the false positives) was -111.9 seconds - rounded to -112 seconds. This means that, on average, the HMM FST algorithm detected the froth event 112 seconds before the existing algorithm.
- In 12 instances, the HMM FST algorithm detected the froth within 5 seconds (before or after) of the original froth detection time.
- Out of the 12 instances above, there is 1 event whereby both algorithms flagged a froth at exactly the same time (to the second).
- Extending the above window to 20 seconds, there were 32 instances whereby the HMM FST algorithm detected the froth within this window - with respect to the original detection time.

6.3 Tuning the Response of the HMM FST Algorithm

As mentioned in the previous chapter, there is always a trade-off between a responsive algorithm and one that has a low false positive rate. In the context of the HMM FST Froth Detector the importance of flagging a froth event early is the second most important performance aspect of this algorithm. The most important performance criterion of this algorithm is that of a low false positive rate. A fine balance has to thus be found between a responsive algorithm and an algorithm that minimises this false positive rate. This section qualitatively explores the first part of the problem ie parameters that increase the algorithm's responsiveness.

In a few of the earlier chapters there has been some discussion on the various parameters of the HMM FST algorithm. In this section a number of these parameters are quantitatively evaluated as to their influence on the HMM Froth Detector's overall responsiveness. The following parameters were considered:

- **Buffer Length**. In Section 5.4.5 the concept of buffer length was discussed. An additional concept of latency was also discussed along with its relationship to the buffer length. It was postulated that a long buffer length increased the system latency and ultimately reduced the detector's responsiveness. The opposite scenario was also discussed whereby a short buffer length reduced the system latency and improved the system response time but increased its false positive rate.
- **Codeword/Observation Choice**. Section 5.4.3 provided a discussion on how the system codebook was created and how each individual codeword (6 in total) was sorted in ascending Euclidean distance from the maximum vector. The final sequence of sorted codewords then became the observations for the Hidden Markov Model (HMM). The first two observations (O_1 and O_2) relate to the Def-Froth hidden state while the last four observations contribute (with various probabilities) to the Poss-Froth and No-Froth hidden states respectively. Although not specifically discussed in Section 5.4.3, this chapter will demonstrate how it is possible to improve the algorithm's response by respectively adjusting the O_1 and O_2 codewords to vectors with higher Euclidean distances from the maximum vector. The net result of this action is that these two HMM observations (that relate to the Def-Froth hidden state)

will be generated sooner (as the electrodes rise to the upper limit) thus increasing the responsiveness of the overall algorithm.

- **Observation Probability Matrix**. In Section 5.4.4 details on the formation of the HMM Observation Probability Matrix (B) were provided. This section explores the algorithm's responsiveness as a function of the B matrix parameters.

6.3.1 Scenario 1 - The Effect of Codeword (Observation) Choice on Response

One of the factors that affects the HMM FST algorithm's response is the choice of codewords. By adjusting the codeword vectors, the corresponding observations are modified which ultimately changes when hidden states are flagged. Instead of allowing the k-means clustering algorithm to find the respective codewords, the option exists to manually configure some of the codewords or even the entire codebook.

In this scenario only the first 3 observations of the codebook were modified (refer to the bold rows in **Table 6.3**) while the buffer length remained at 10 and the respective HMM matrices were left untouched. By modifying the code vector elements (Y_1 to Y_7), in such a way that the Euclidean distance from the maximum vector is reduced, the respective observations are emitted for larger vector values ie higher electrode positions and higher rates of change. The net effect is that the Def-Froth HMM hidden state is flagged for higher electrode positions and later into the build up of a froth thus rendering the algorithm more conservative.

Table 6.3 Manually Modified Codebook

Datetime	IDX	Obs	E_{dist}	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
2012/12/28 03:32	**1**	**O1**	**248.2**	**100**	**800**	**800**	**800**	**800**	**800**	**800**
2012/12/28 03:32	**2**	**O2**	**489.9**	**60**	**700**	**700**	**700**	**700**	**700**	**700**
2012/12/28 03:32	**3**	**O3**	**686.1**	**40**	**620**	**620**	**620**	**620**	**620**	**620**
2012/12/28 03:32	4	O4	1085.2	-2	514	355	451	626	440	410
2012/12/28 03:32	5	O5	1331.0	-25	297	328	330	384	370	452
2012/12/28 03:32	6	O6	1644.2	-45	247	204	236	219	224	251

Comparing the first 3 rows of **Table 6.3** with those of **Table 6.1** the main difference to note is the respective codeword values for column E_{dist} which are lower in **Table 6.3**. This means that the vectors have higher values for the same respective observation symbol and should result in froths being detected later by the HMM FST algorithm. The main implication of this is that the new codebook in **Table 6.3** will ultimately make the model less aggressive (conservative) in its froth detection ability.
The above mechanism is confirmed when considering the resulting output of the HMM FST algorithm (tested with a different set of 100 froth data files). The results for this scenario are published in **Table D2** in APPENDIX D.

The observations that can be drawn from these results are as follows:

- In all 100 data files a froth was detected. There were no events that were missed in this scenario.
- There was only one data file whereby a false positive (refer to the bold rows in **Table D2** in APPENDIX D) resulted. The data file with index 12 produced 2 false positives. If the same false positive criterion as the scenario in Section 6.2 is applied (a froth event detected within 10 minutes of a prior froth detection is discarded), then there is only 1 false positive for this entire scenario - in this case, the first event was deemed the false positive which means that the third detection event was discarded. The net result is that this scenario recorded one false positive out of 100 data files.
- With the above 2 rows removed, the average delay was 102 seconds which means that (on average) the HMM FST algorithm detected froth events 102 seconds later than the current algorithm deployed on the smelter. In the previous scenario this delay was -112 seconds.

The results of this scenario prove that the HMM FST algorithm's codebook plays a major part in its performance. The less the codeword's Euclidean distance from the maximum vector V_{max}, the higher the electrodes will need to rise in order to trigger a froth detection event. This action made the algorithm less aggressive but at the same time reduced raw false positives (the false positive detections observed before applying the 10 minute consecutive froth rule).

6.3.2 Scenario 2 - The Effect of Buffer Length on Response

The next parameter tested for its impact on the Hidden Markov Model (HMM) Fixed State Test (FST) algorithm's performance was the buffer length (Section 5.4.5 provides more detail on this parameter). As such, the HMM FST model's parameters were left as in Scenario 1 with the only change being the Buffer Length reduction from L =10 to L = 5.
The expectation was that by reducing the buffer length (in this case to L = 5) the algorithm would become more aggressive and detect froths quicker as the latency was halved (the buffer lengths in the above scenarios were fixed at 10).

A further expectation was that this change would be relatively linear i.e. the froth detections would be phase shifted 10 seconds earlier.

When considering the results published in **Table D3** in APPENDIX D, the above concepts were confirmed. Specifically, the following observations can be made:

- In all 100 data files a froth was detected. There were no events that were missed in this scenario.
- As in Scenario 1 there was only one data file (with index 12) that produced false positives (refer to the bold rows in **Table D3** in APPENDIX D). The specifics of these false positives are very similar to those reported in Scenario 1 (in Section 6.3.1) except that the detection time was 10 seconds sooner. As such they were dealt with in the same manner as the previous section (1 event was removed and the other was flagged as a false positive). The net result is that this scenario recorded one false positive out of 100 data files.
- About 80% of all the froth event detections have detection times that occurred 10 seconds earlier than those published in Section 6.3.1.
- With the false positives removed, the average detection delay was reduced to 2.4 seconds.

The results of this scenario provide a good indication as to the impact of buffer length on the overall model. In most cases the buffer length only affected the model's latency as these detection times were advanced by 10 seconds. In a minority of cases the entire model's detection regime was altered resulting in detections much

earlier. The net result was that by reducing the buffer length makes the model more aggressive.

6.3.3 Scenario 3 - The Effect of Emission Matrix Parameters on Response

The final scenario in this section is one whereby the effect of the HMM Emission Matrix (B) on model performance was evaluated. In this example, the emission matrix [the original B matrix is given in **Equation (52)**] was changed to the matrix described in **Equation (53)**. The rest of the model's parameters remain as described in the Original Scenario - refer to Section 6.2 for details.
These parameters are: a Buffer Length of L = 10 and the HMM observation Codebook as described in **Table 6.1** (the codebook was generated using the K-Means Clustering algorithm in this case).
Note that only the third row in the matrix in **Equation (53)** was adjusted. This row corresponds to the Def-Froth state.

$$B = \begin{pmatrix} 0 & 0 & 0.1 & 0.2 & 0.3 & 0.4 \\ 0.05 & 0.4 & 0.5 & 0.05 & 0 & 0 \\ 0.8 & 0.2 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{53}$$

The implications of changing this row to the values described above are that the HMM model requires more O_1 observations to flag a froth state. The contribution from the O_2 observation is thus reduced from a probability of 0.4 to 0.2 which means there is a lower chance of this observation being associated with the Def-Froth state.

The expected overall impact of this change was twofold:

- An increase in the froth delay times (relative to the Original Scenario discussed in Section 6.2) as more O_1 observations must be output before the Def-Froth state is flagged.
- A reduction in false positives for the same reason.

The results published in **Table D4** in APPENDIX D confirm the expected findings as follows:

- The algorithm produced 5 false positive rows (compared with 15 in the Original Scenario) - the bold rows in **Table D4** in APPENDIX D highlight these false positive events. When applying the 10 minute window rule (a froth detection is discarded if it proceeds a prior froth detection within a 10 minute window) all 5 of the above events can be discarded. The net result is that this scenario produced 0 false positives.
- The algorithm detected every froth event and there were no "true negatives" output (in the Original Scenario one instance was recorded whereby a froth event went undetected).
- The average delay (excluding the false positives) was 7.2 seconds (-112 seconds was recorded for the Original Scenario).

Summarising the above, the results indicate that the parameters loaded for this scenario render the HMM FST algorithm more conservative (delayed detections) but also significantly reduced the false positive detections to zero.

6.3.4 Scenario Results Summary

Section 6.3 provided results for four different scenarios. They are as follows:

- **Original Scenario**. In this scenario the observation codebook was generated using the K-Means Clustering Algorithm (as specified in **Table 6.1)**. The Buffer Length was set at 10 and the Emission Matrix (B) was configured as in **Equation (52)**.
- **Scenario 1**. In this scenario half of the codewords in the observation codebook were modified manually (to more conservative values). The Buffer Length remained at L = 10 and the Emission Matrix was left unchanged.
- **Scenario 2**. This scenario tested the response of the algorithm to a reduced Buffer Length which was reduced from L = 10 to L = 5. The rest of the HMM FST model's parameters were left as in Scenario 1.
- **Scenario 3**. The final scenario tested the impact of a more conservative Emission Matrix (B) which is provided in **Equation (53)**. The rest of the model's parameters were returned to the values configured in the Original Scenario (Codebook generated by K-Means Clustering and a Buffer Length L = 10).

With respect to the four scenarios above, **Table 6.4** provides a symbolic summary of the results of the various model parameters. Using the symbolic states in **Table 6.4** the following observations can be inferred:

- An aggressive codebook with any other parameter tuned aggressively results in a model that is too aggressive which in turn results in too many false positives.
- A conservative codebook results in a very unresponsive model. Making both of the other 2 parameters (Buffer Length and B Matrix) aggressive improves this response.
- The final observation from **Table 6.4** is that the tuning parameters from Scenario 2 or Scenario 3 are the most desirable. It should be noted that the false positive test described in Section 6.4 is a more realistic measure of model's false positive output and the results from this test contribute to the values reported in the column "False Positive Ratio".

Table 6.4 Symbolic Summary of Scenario Results

Scenario	Codebook	B Matrix	Buffer Length	False Positive Ratio	Delay
Original Scenario	Aggressive	Aggressive	Conservative	Moderate	Large Negative
Scenario 1	Conservative	Aggressive	Conservative	Low	Large Positive
Scenario 2	Conservative	Aggressive	Aggressive	Low	Close to Zero
Scenario 3	Aggressive	Conservative	Conservative	Very Low	Small Positive

In the respective sections discussing each of the scenarios, the average delay time (the delay between the HMM FST algorithm detecting a froth and the time of the original froth detection as given by the existing algorithm deployed on the furnaces) was used as one of the main performance indicators. Due to the large variance of the delay times, a more appropriate technique (to compare scenarios) is that of using a Histogram. **Figure 6.5** provides the distribution of froth events over the respective x-axis intervals for each of the four scenarios discussed above.

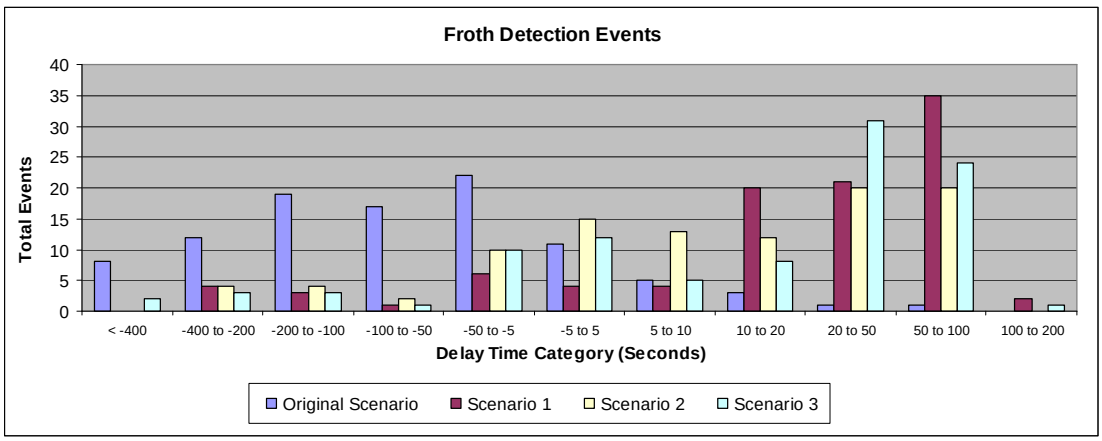

Figure 6.5 A Histogram of Froth Events versus Delay Intervals for Each Scenario

Referring to **Figure 6.5**, the following observations apply:

- The Original Scenario rendered the HMM FST algorithm the most aggressive (the distribution of events are clustered in the negative delay intervals).

- The HMM FST algorithm parameters utilised in Scenario 1, resulted in the most conservative response of the model (out of the four scenarios). This is evidenced with the events clustered to the right of the distribution.
- The response of the model with Scenario 2's parameters was more towards the middle of the road.
- The last paragraph of Section 5.4.9 describes the ideal output of the Hidden Markov Model (HMM) Fixed State Test (FST) algorithm to be one where there are zero false positives with the froth delay time being close to 0 or even slightly negative. As can be observed in **Figure 6.5**, together with **Table 6.4**, the parameters configured in Scenario 2 best achieve this output.
- The final decision as to which model parameter set should be adopted was made after determining which tuning set minimises the number of false positives or invalid detections. In this test (refer to Section 6.4 for details) 30 new data files (that have been judged as disputable or invalid froth detection events) were replayed through the model to gauge whether the HMM FST model also flagged these same events as a froth.

6.4 Evaluating the HMM FST Detector's False Positive Response

In this section, the false positive response of the model is analysed. The methodology employed to achieve this is similar to that used above ie a number of froth data files (30 in this scenario) were replayed into the HMM FST Froth Detector Algorithm. The difference in this case is that these data files contain false positives or at least froths that are not rated as genuine froths. The aim was to determine whether the HMM FST algorithm is robust enough to reject events that are not legitimate froths.
Note that the perfect algorithm is one that is aggressive (detects froths early) but one that does not experience any false positives (all the events it detects are genuine froths).

The latter part of this section reports on the results obtained when the HMM FST model was tested with the parameters used in Scenario 2 and Scenario 3 respectively. The details of these parameters are as follows:

- **Scenario 2**. In this scenario (a more conservative) observation codebook was manually configured with values presented in **Table 6.3** The Buffer Length was set at 5 and a (relatively aggressive) Emission Matrix (B) was configured as in **Equation (52)**.
- **Scenario 3**. A more conservative Emission Matrix (B) [provided in **Equation (53)**] was used in this scenario. The Codebook used was the one generated by K-Means Clustering (as specified in **Table 6.1)**. A Buffer Length of L = 10 was used.

Before discussing the quantitative results, a few qualitative results are discussed below. The first example of a false positive is provided in **Figure 6.6**. It is important to note that both froth detector algorithms only have the benefit of hindsight up to the point of detection (identified by the vertical cyan line in **Figure 6.6**) - they cannot look into the future to see whether their respective decision at that point in time was justified.

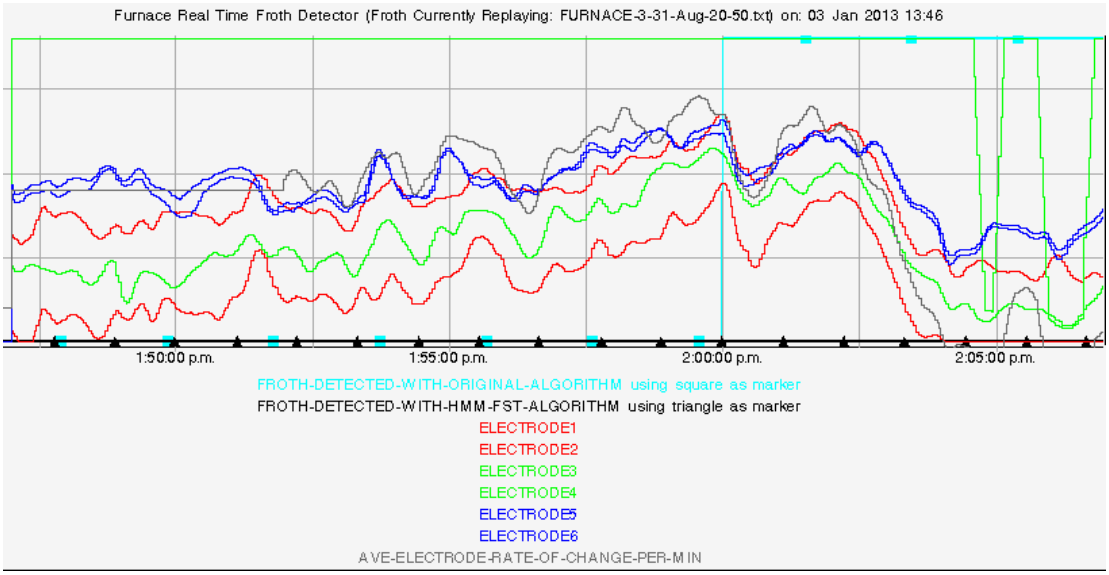

Figure 6.6 An Example of a False Positive Detection

With respect to the froth event in **Figure 6.6** it is important to note that the HMM FST algorithm did not flag this event (note the absence of a vertical black line) as a froth and thus rejected the false positive in this case. This is despite the fact that Electrode 4 (green plot) resided in the top limit for the duration of the replayed data. One aim of the HMM FST algorithm is to reduce the reliance on individual electrode positions and look at the electrode groupings as a whole. By applying the Euclidean distance to a vector of 7 elements, a spike or a faulty electrode position transducer (as in the case of **Figure 6.6**) should not result in the model flagging a froth detection event.

Figure 6.7 is another example of a marginal call with respect to flagging a froth. The electrodes were well grouped and none were over-slipped. A short term upward moment of all the electrodes triggered the froth detection in the current plant system (vertical cyan line) as well as within the HMM FST algorithm (vertical back line). This froth event was more genuine than the one above but its genuine froth rating would have been higher had the algorithm waited for a minute or two. In this case a genuine froth was building but the existing detector probably flagged this froth due to the short term rise rate of all electrodes.

An experienced plant operator might have chosen to wait for a minute or two to observe the behaviour of the electrodes to improve his or her confidence level that a froth was genuinely in the making - however this is nearly always a gamble as the opportunity for suppression could be lost even if the decision is delayed for a minute.

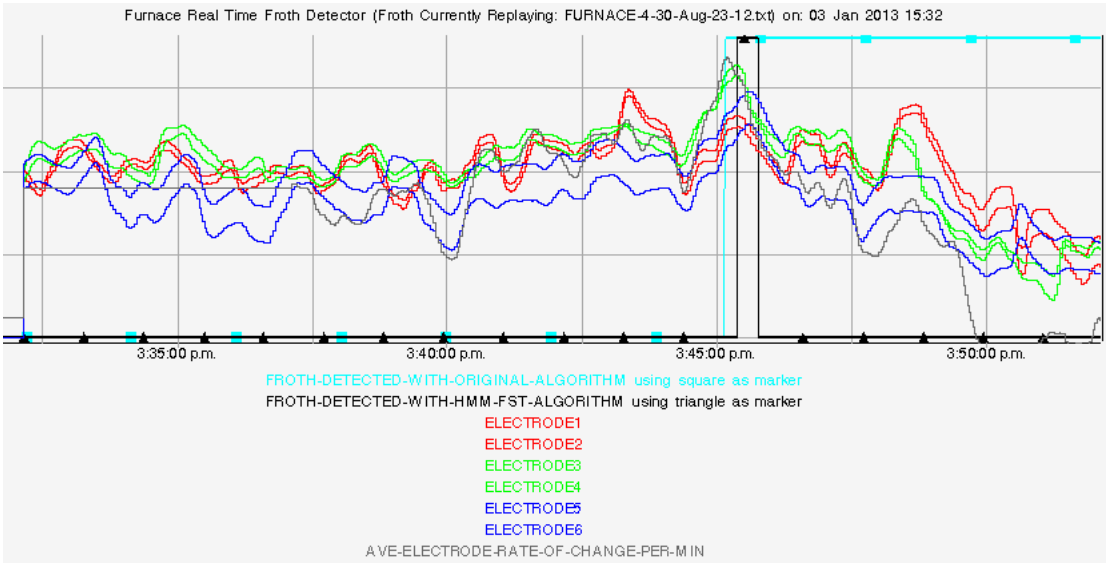

Figure 6.7 A Second Example of a False Positive Detection

6.4.1 Estimating The False Positive Rate With Scenario 2 Parameters

Figure 6.8 (as do **Figure 6.6** and **Figure 6.7)** provides a qualitative example of a false positive event replayed into the HMM FST froth detector. As can be observed, this event was flagged as a false positive by the existing Smelter froth detector soft sensor. However it was not flagged as a froth by the HMM FST algorithm - the reason for this is due to the design of the algorithm that looks at a collection of process data (contained in a 7 element input vector every 2 seconds) before concluding the process status.

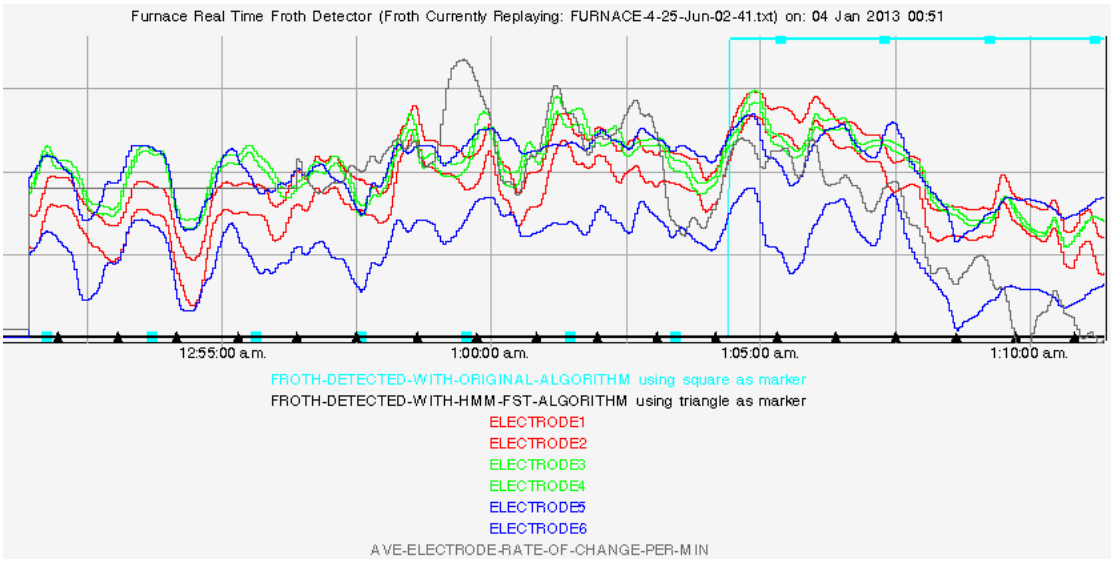

Figure 6.8 An Example of a Replayed False Positive Data File with Scenario 2 Parameters

Figure 6.9 provides a another qualitative example of a false positive event replayed into the HMM FST froth detector (using the model parameters from Scenario 2) that was rejected as a false positive by the HMM FST algorithm. This example provides an instance of where the average electrode rise rate clearly indicated that a froth was building. However the electrode positions were not high enough for the HMM FST algorithm to flag it as a froth event. This is another example of how the HMM FST algorithm is able to reduce false positives by looking at the big picture rather than the behaviour of individual electrodes.

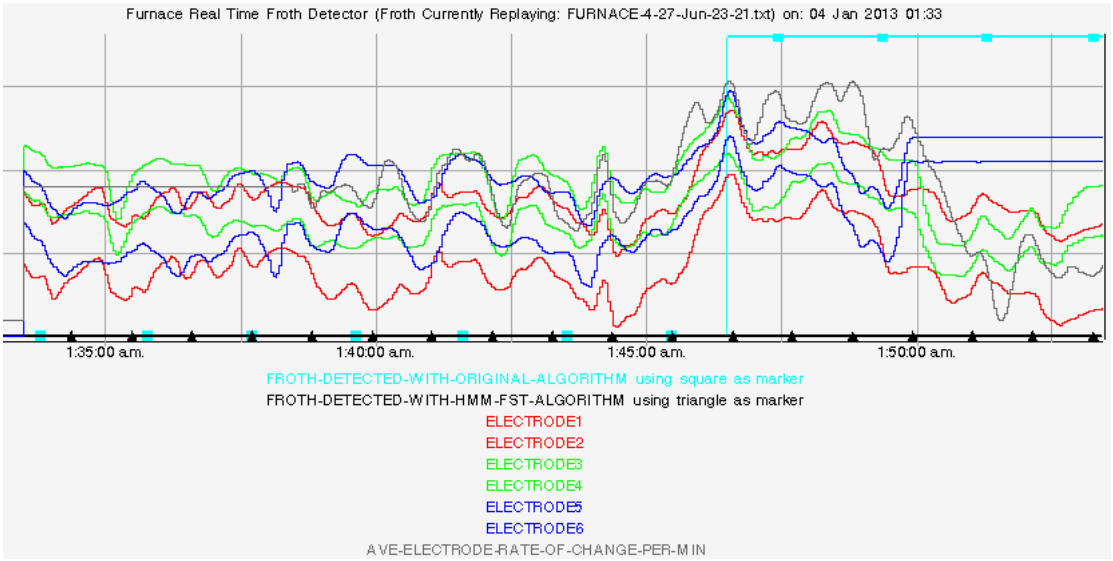

Figure 6.9 A Second Example of a Replayed False Positive Data File with Scenario 2 Parameters

The following table (**Table 6.5)** provides quantitative results of the false positive test utilising the model parameters associated with Scenario 2. In this scenario, 17 events (1 event per row) were captured as froth events. Given the fact that all 30 data files, replayed in this section, were taken from froth detection events that were rated as false positives, the 17 froth events detected below were also false positive detections.

Table 6.5 HMM FST False Positive Test Results (Scenario 2 Parameters)

Froth File Index	Replay Start Time	Old Detection Offset	New Detection Time	New Detection Offset	Delay
5	03 Jan 2013 17:50:18	781	29 Sep 2012 09:38:48	784	3
7	03 Jan 2013 18:32:18	782	31 Aug 2012 03:15:28	536	-246
7	03 Jan 2013 18:32:18	782	31 Aug 2012 03:21:48	916	134
8	03 Jan 2013 18:53:20	781	04 Oct 2012 00:00:48	552	-229
9	03 Jan 2013 19:14:20	781	04 Oct 2012 00:43:56	782	1
14	03 Jan 2013 20:39:20	781	01 Oct 2012 13:20:38	312	-469
14	03 Jan 2013 20:39:20	781	01 Oct 2012 13:26:30	664	-117
15	03 Jan 2013 21:00:20	781	01 Oct 2012 13:38:58	460	-321
15	03 Jan 2013 21:00:20	781	01 Oct 2012 13:44:14	776	-5
18	03 Jan 2013 22:03:22	781	01 Oct 2012 21:17:26	618	-163
20	03 Jan 2013 22:45:22	781	03 Oct 2012 23:10:54	896	115
24	04 Jan 2013 00:09:22	781	30 Sep 2012 11:19:10	348	-433
24	04 Jan 2013 00:09:22	781	30 Sep 2012 11:26:24	782	1
29	04 Jan 2013 01:54:30	781	29 Sep 2012 15:45:52	324	-457
29	04 Jan 2013 01:54:30	781	29 Sep 2012 15:51:56	688	-93
29	04 Jan 2013 01:54:30	781	29 Sep 2012 15:58:10	1062	281
30	04 Jan 2013 02:15:30	782	30 Aug 2012 23:12:48	786	4

From **Table 6.5** it should be noted that out of the 17 events detected there were a number of multiple detections per file. Compensating for this, the above data reflects false positives from 11 out of the 30 replayed data files. Expressing the 17 detection events as a percentage (out of 30), the false positive rate is 57% ie 57% of the tested events resulted in false positives.

6.4.2 Estimating The False Positive Rate With Scenario 3 Parameters

Figure 6.10 provides a qualitative result of how the plant conditions indicated that a froth was building but this process state was rejected as a false positive by the HMM FST algorithm. In this replayed data file the current plant algorithm flagged a froth event while the HMM FST detector did not. It appears that Electrode 3 and Electrode 4 (green plots) were both offline until roughly 10 minutes from the end of the file. As soon as they acquired arc they retracted. This high rate of change resulted in the original algorithm flagging a froth event which is an obvious false positive. Similarly to the event in **Figure 6.6** the HMM FST algorithm did not flag an event although there were individual electrodes that exhibited froth type behaviour (upward rise rates). This provides further evidence of the strength of the HMM FST algorithm which can discard individual electrode behaviour and flag a froth event based upon group behaviour.

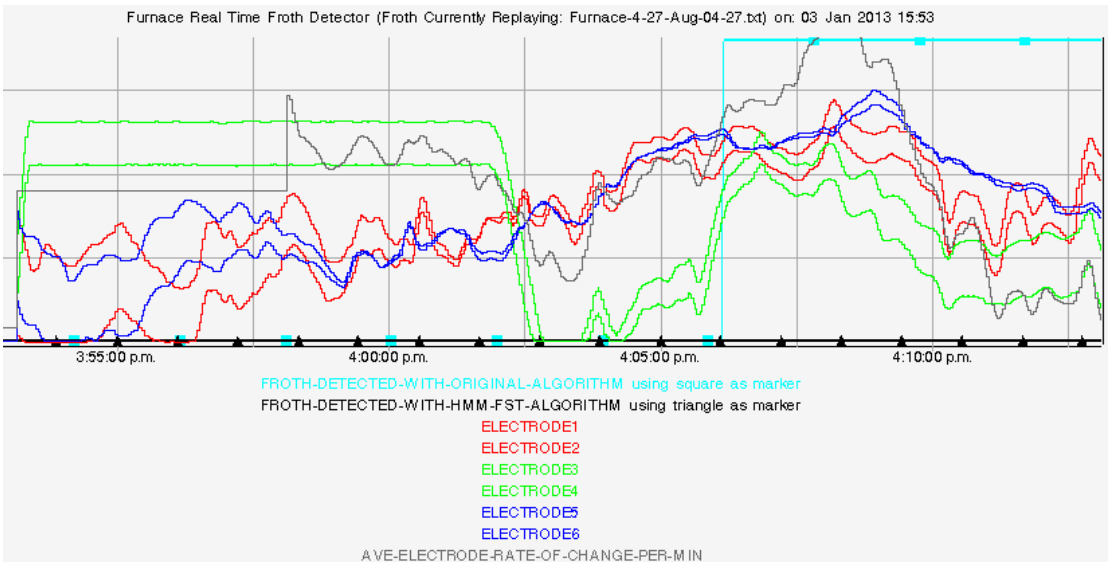

Figure 6.10 An Example of a Replayed False Positive Data File with Scenario 3 Parameters

The following table (**Table 6.6**) provides quantitative results of the false positive test utilising the model parameters associated with Scenario 3.

Table 6.6 HMM FST False Positive Test Results (Scenario 3 Parameters)

Froth File Index	Replay Start Time	Old Detection Offset	New Detection Time	New Detection Offset	Delay
7	14 Feb 2013 04:22:54	782	31 Aug 2012 03:19:34	782	0
8	14 Feb 2013 04:43:56	781	04 Oct 2012 00:00:36	540	-241
8	14 Feb 2013 04:43:56	781	04 Oct 2012 00:06:40	904	123
9	14 Feb 2013 05:04:56	781	04 Oct 2012 00:42:44	710	-71
12	14 Feb 2013 05:47:56	781	01 Oct 2012 10:10:58	334	-447
14	14 Feb 2013 06:29:56	781	01 Oct 2012 13:26:28	662	-119
15	14 Feb 2013 06:50:57	781	01 Oct 2012 13:47:16	958	177
18	14 Feb 2013 07:53:57	781	01 Oct 2012 21:17:22	614	-167
20	14 Feb 2013 08:35:56	781	03 Oct 2012 23:10:52	894	113
21	14 Feb 2013 08:56:56	781	30 Sep 2012 05:28:34	964	183
24	14 Feb 2013 09:59:56	781	30 Sep 2012 11:26:26	784	3
29	14 Feb 2013 11:45:04	781	29 Sep 2012 15:48:54	506	-275
30	14 Feb 2013 12:06:04	782	30 Aug 2012 23:12:56	794	12

Although the ideal result (in the false positive test scenario) would be a table with zero rows, the total number of froth detection events totals **13.** The potential number of events that could be detected as false positives is not necessarily limited to 30 (the 2nd row in **Table 6.6** indicates that two, or more, false positive events can be detected per replayed froth file). In this scenario the 13 rows translate to a false positive rate of about 43%. It is important to note that all 30 files were flagged as respective froth events (by the existing plant froth detector) and resulted in unnecessary froth suppression procedures being initiated in each respective event.

6.4.3 False Positive Test Summary

From the results published in Section 6.4.1 and Section 6.4.2, it can be concluded that the HMM FST algorithm parameters that result in the lowest false positive rate are those associated with Scenario 3 (43% versus 57% for Scenario 2). In this scenario the codebook was configured aggressively while the buffer length and emission matrix were respectively configured with conservative values. From **Table 6.4** it can be noted that the highest number of false positives was observed with an aggressive codebook as well as an aggressive B matrix. Thus in order to keep false positive events (non-genuine froth detections) to a minimum, the HMM emission matrix (B Matrix) should contain a conservative configuration.

6.5 Summary

In this chapter the results of the dissertation were presented. Four scenarios were executed to test the performance of the HMM FST algorithm with various tuning parameters.

The data used to test the HMM FST algorithm consisted of 100 new data files (totally different data to what was used in the K-Means clustering training procedure) containing genuine froth events. An additional 30 data files (of false positive events) were used to test the algorithm's false positive response. In determining the performance of the algorithm the following criteria were used:

- **The delay time**. The data files utilised in this dissertation originate from an existing soft sensor that detects froths using a hybrid fuzzy logic algorithm. This provided a benchmark that was used to measure the performance of the HMM FST algorithm. The difference in detection time between the two systems, per replayed froth file, was deemed the delay time and formed the first performance indicator in this chapter.
- **False positive output**. The algorithm flagging a froth detection, when in reality no froth existed, would be classified as a false positive. With respect to the 100 data files, there was only one froth event contained per file. Therefore more than one froth detection per file was deemed as a false positive.

With regard to the detection performance of the HMM FST algorithm it was found that an aggressive codebook plays a significant role in the detection delay. The average delay ranged from a large negative time difference (indicating that the HMM FST algorithm was detecting the events sooner than the existing plant algorithm) to a large positive difference - implying that the HMM FST algorithm was more conservative in performance.

When considering the false positive ratio of the HMM FST algorithm, the HMM FST algorithm flagged 43% of the replayed false positives as froth detection events. This implies that the HMM FST algorithm contains better false positive rejection properties that the existing plant soft sensor.

This chapter concluded with a proposal on the best tuning set to use for this application of the HMM FST algorithm.

CHAPTER 7 CONCLUSIONS

The results in CHAPTER 6 present findings based upon 4 scenarios. Each scenario tests a different set of HMM FST algorithm tuning parameters. A set of 100 data files (of genuine froth events) was replayed through the HMM FST algorithm for each tuning parameter set. An additional set of 30 data files were also replayed into the HMM FST algorithm. These 30 data files contained non-genuine froth events or false positives. Only Scenario's 2 and 3 were tested with these input files. In total 230 (the training of the VQ algorithm accounts for the other 100 data files) unique data files were thus utilised in this research dissertation.

A summary of all the relevant results (per scenario) is presented in **Table 7.1**.
The conclusions discussed in the following sections will thus refer to this performance summary.

Table 7.1 A Summary of Results Per Scenario

	100 Genuine Froth Data Files				**30 Data Files of False Positives**
Scenario	**Froths Detected**	**Average Delay Times (seconds)**	**False Positive Detections**	**Histogram Observations**	**False Positives (%)**
Original Scenario	99	-112	(15) 1	Delay times clustered in -200 to 5s range. No delay times longer than 20s.	N/A
Scenario 1	100	102	(2) 1	Delay times mainly clustered in 10 to 100s range but with 35% of delays in 50 to 100s range	N/A
Scenario 2	100	2	(2) 1	Moderate variance of delay times with main cluster in 20 to 100s range	57%
Scenario 3	100	7	(5) 0	Larger variance of delay times with main group in 20 to 100s range	43%

7.1 The Research Question Answered

The question that this research intended to answer was whether Hidden Markov Models are suitable for use as a real time soft sensor and specifically suitable for use as a real time furnace froth detector.

From the results in **Table 7.1** it can be observed that in each scenario all 100 froth events (399 out of 400 simulations in total) were detected by the HMM FST algorithm. The one exception to this was in the Original Scenario that failed to detect a froth event in 1 of the replayed data files - which was shown to be as a result of a testing procedure deficiency rather than a genuine "true negative".

Based upon the above, it can be concluded that Hidden Markov Models can be used, in real time, to detect furnace froths.

7.2 HMM FST False Positive Rejection Properties

Following from the original test of 100 data files, the ideal result was that no false positives (non-genuine froth detections) should have been generated by the model. From **Table 7.1** it can be observed that, at most, one false positive detection was output by the HMM FST model, per scenario, after the 10 minute consecutive froth detection rule was applied.

Considering the raw false positive outputs (ie the bold rows in Table D1 to Table D4 in APPENDIX D) which are summarised in brackets in the False Positive Detections column in **Table 7.1**, 24 raw false positives were detected from 400 data files. This equates to a false positive ratio of 6%.

A more rigid test of the model's ability to reject non-genuine froth detections was devised by replaying 30 data files that contained detections that were not genuine froths (ie false positives**)** through the HMM FST model. In this test Scenario 3 yielded the lowest raw false positive rate of 43% (13 events out of 30 files) which is a significant improvement on the current plant system which produced a 100% false

positive rate for the same 30 files. In this case the 10 minute rule between subsequent froths was not applied as each event was deemed significant.
From Section 6.4.3 it was shown that to keep false positives to a minimum, the B matrix should be tuned more conservatively.

The conclusion that can be made is that the HMM FST algorithm displays improved false positive rejection qualities (compared with the existing Smelter algorithm) and in general exhibits a low false positive generation rate. As discussed in Section 6.4 this ability can be ascribed to the design of the model whereby plant data is processed as a group (vector data with Euclidean distances) to ultimately determine whether the furnace is frothing or not.

7.3 Secondary Conclusions

The following secondary conclusions can be drawn from the results summary in **Table 7.1**:

- The delay times reported against each froth demonstrated a variance of model tuning from very aggressive to very conservative. The conclusion that can be made from this is that the HMM FST model has a wide tuning range that can be utilised to tune the HMM FST model for different process requirements.
- Considering the model's design, the only component specific to the structure of the furnace utilised to generate the froth data is the codebook. In this dissertation the structure of the codewords were vectors of 7 elements comprising a rise rate and 6 electrode positions. To apply this to a furnace of 3 electrodes, the codeword design would need to change to 4 elements. The varying electrode travel distance from furnace to furnace could be catered for by adjusting the V_{max} vector. The conclusion that can be drawn from this is that the HMM FST algorithm can be utilised to detect froths in any type of furnace (provided that the froth is evidenced by a general rise of the electrodes)

7.4 Future Work

The nature of the work presented in this dissertation is mainly that of research although a small development component has been completed. In the Author's professional experience, to successfully transition any soft sensor application to a successful online product, a development process has to be completed. In some cases there are process specialists on the plant that can short-cut the research process by sharing their knowledge. However there are no short cuts for the development process as there are usually many process specifics and real world considerations that require attention before a product can be installed onto a plant.

In parallel to deploying this algorithm onto a live plant, some future research is proposed to further explore the relationships between the model parameters. There are many degrees of freedom that could complicate the tuning of this model. Different model parameters result in different algorithm responses that are not necessarily linear. The following lists some specific proposals for future work:

- The state transition matrix (A) remained constant throughout all of the above simulations. Future work could include adjusting the parameters of the state transition matrix (A) to explore its effect on the FST HMM model performance.
- The results from the four scenarios tested in this dissertation demonstrate that the buffer length (L) parameter has a significant impact on model performance. One of the proposed activities for future work is to explore, in greater detail, the impact of buffer length on model performance. A particular focus could be to determine a specific range of buffer lengths that deliver good model performance while maintaining a low false positive rate. A more advanced study could be undertaken to explore the relationship between this parameter and the natural time constant of the system to assist with tuning on new furnaces.
- The final proposal for future work is to utilise the Baum-Welch algorithm in some way to determine the feasibility of training the HMM model with existing data. The Baum-Welch algorithm outputs optimal parameters for the A, B and π matrices in the classical use of Hidden Markov Models. Researching the ability to utilise this algorithm to determine optimal parameters for $\lambda = (A, B, \pi)$ could yield performance improvements and even lead to a methodology for

unsupervised training of the model to compensate for longer term process variations.

In the case of the practical application of this research dissertation, the next step would be to install the respective algorithms onto a suitable online control system. A G2 Expert System is not necessarily required to deploy this application. The algorithm would thus require significant online, real time testing for a few months before its outputs could be alarmed to operations. Thereafter a further stage of online testing would be required before the HMM FST algorithm can be allowed to make autonomous decisions with respect to froth suppression. The above periods thus comprise the development stage and form the basis for any future work on the Hidden Markov Model Fixed State Test algorithm.

REFERENCES

Ardö, H. 2007. *Multi-Target Tracking Using On-line Viterbi Optimisation and Stochastic Modelling*. Unpublished Doctoral Thesis, Lund University, Sweden. Available from www.maths.lth.se/vision/publdb/reports/pdf/ardo-phd-09.pdf [Accessed: 5 October 2012].

Arthur D. & Vassilvitskii S. 2006. *K-Means++: The Advantages of Careful Seeding*. Stanford InfoLab Publication Server. [n.p.] Available from http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf [Accessed 4 January 2012].

Baum, L.E. & Petrie T. 1966. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *Annals of Mathematical Statistics*, 37 (6),1554-1563.

Bloit, J. & Rodet, X. 2008. *Short-time Viterbi for online HMM decoding : Evaluation On a Real-time Phone Recognition Task*. ICASSP Conference proceedings, March 2008, Las Vegas, IRCAM, France. Available from http://articles.ircam.fr/textes/Bloit08a/index.pdf [Accessed: 5 October 2012].

Boulet B., Vaculik V. & Wong G. 1997. *Control on Non-Ferrous Electric Arc Furnaces*. IEEE Canadian Review, Summer Issue, 4-8.

Bruckner, D., Sallans, B. & Russ, G. 2007. *Hidden Markov Models for Traffic Observation*. 5th IEEE International Conference on Industrial Informatics. 23-27 June 2007 . Vienna, Austria. IEEE Xplore Digital Library, 989-994

Campos, J. 2010. *Hidden Markov Models - Applications in Bioinformatics*. Universidad de Zaragoza, Spain. [n.p.]. Available from: http://webdiis.unizar.es/asignaturas/SPN/material/HMM.pdf [Accessed: 8 April 2012].

Chen C.-Q., Koh S.-N. & Soon I.-Y. 1998. *Fast codebook search algorithm for unconstrained vector quantisation*. IEE Proceedings - Vision, Image, and Signal Processing, 145 (2), 97–102.

Cho, T. & Bello, J.P. 2009. *Real-Time Implementation Of Hmm-Based Chord Estimation In Musical Audio*. New York University, USA. [n.p.]. Available from https://files.nyu.edu/jb2843/public/Publications_files/Cho_Bello_ICMC2009.pdf [Accessed: 15 October 2012].

Forney, G.D. 1973. *The Viterbi algorithm*. Proceedings of the IEEE, 61 (3), 268–278.

GNU Octave, software, version 3.2.4, John W. Eaton, USA

Mostert J. & Roberts P. 1973. Electric Smelting At Rustenburg Platinum Mines Limited Of Nickel-Copper Concentrates Containing Platinum-Group Metals. *Journal of the South African Institute of Mining and Metallurgy*, 73 (9), 290–299.

Panoiu M., Panoiu C., Sora I. & Iordan A. 2008. *Study about the Process Control of an Electric Arc Furnace Using Simulations based on an Adaptive Algorithm*. WSEAS Transactions on Information Science and Applications. 11 (5), 1628-1637.

Rabiner, L.R. 1989. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*. Proceedings of the IEEE, 77 (2), 258-286.

Ramachandran, R.P. 1999. Quantization of Discrete Time Signals. In: Madisetti V.K., Williams D.B. ed. *Digital Signal Processing Handbook*. Boca Raton. CRC Press LLC. Available from http://www.dsp-book.narod.ru/DSPMW/06.PDF [Accessed: 2 January 2012].

Savage J., Marquez E. & Lepe-casillas O. 2005. *Hidden Markov Models and Vector Quantization for Mobile Robot Localization*. IASTED Proceedings - Robotics and Applications, 31 October 2005 to 2 November 2005. Cambridge, MA, USA: ACTA Press, 498-032.

Stamp, M. 2004. *A Revealing Introduction to Hidden Markov Models*. Department of Computer Science San Jose State University. [n.p.]. Available from: www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf [Accessed: 28 May 2008].

Tai, A., Ching, W. & Chan, P. 2006. *Hidden Markov Model for the Detection of Machine Failure*. Proceedings of the 36th International Conference of Computers and Industrial Engineering. June 2006. Taiwan. Available from

http://hkumath.hku.hk/~imr/IMRPreprintSeries/2006/IMR2006-07.pdf [Accessed 7 January 2013].

Teknomo K. 2006. *K-Means Clustering Tutorials*. Kardi Teknomo's Page. [n.p.] Available from http://people.revoledu.com/kardi/tutorial/kMean/Algorithm.htm [Accessed: 21 November 2012].

University of Leeds. 2010. *Hidden Markov Models*. Available from: http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html [Accessed: 4 May 2011].

Viterbi, A. 2006. *A Personal History of the Viterbi Algorithm*. IEEE Signal Processing Magazine, 23 (4), 120–142.

Walpole, R. & Myers, R. 1990. *Probability And Statistics For Engineers And Scientists*. 4th Edition. New York, USA: Macmillan Publishing Company.

Zhang, J., Liu, J. & Run-Sheng, L. 2002. *Real Time Viterbi Searching For Practical Telephone Speech Recognition Systems*. International Symposium on Chinese Spoken Language Processing (ISCSLP 2002) Conference proceedings, August 2002, Taipei, Taiwan. Available from http://www.isca-speech.org/archive_open/archive_papers/iscslp2002/clp2_104.pdf [Accessed: 24 September 2012].

APPENDIX A

Table A1 Results of Fair and Loaded Dice Algorithm Test

Date & Time	Dice Type	Throws	Ones %	Twos %	Threes %	Fours %	Fives %	Sixes %
2012/07/23 02:07	FAIR	1000000	16.67	16.687	16.606	16.683	16.708	16.645
2012/07/23 02:10	FAIR	1000000	16.623	16.653	16.684	16.717	16.663	16.659
2012/07/23 02:10	FAIR	1000000	16.657	16.627	16.705	16.636	16.688	16.688
2012/07/23 02:10	FAIR	1000000	16.712	16.714	16.693	16.656	16.611	16.614
2012/07/23 02:10	FAIR	1000000	16.631	16.659	16.664	16.681	16.658	16.706
2012/07/23 02:10	FAIR	1000000	16.644	16.678	16.66	16.673	16.64	16.706
2012/07/23 02:10	FAIR	1000000	16.653	16.665	16.678	16.661	16.7	16.644
2012/07/23 02:12	FAIR	1000000	16.64	16.703	16.6	16.741	16.686	16.629
2012/07/23 02:12	FAIR	1000000	16.668	16.629	16.713	16.683	16.643	16.664
2012/07/23 02:13	FAIR	1000000	16.719	16.667	16.659	16.645	16.637	16.672
2012/07/23 02:07	LOADED	1000000	9.99	10.029	10.061	10.026	9.948	49.956
2012/07/23 02:10	LOADED	1000000	9.963	10.006	10.036	10.005	9.978	50.021
2012/07/23 02:10	LOADED	1000000	10.013	9.948	9.986	10.03	10.035	49.998
2012/07/23 02:10	LOADED	1000000	10.031	9.992	10.026	10.02	9.987	49.955
2012/07/23 02:10	LOADED	1000000	9.964	10.014	10.026	10.015	9.961	50.03
2012/07/23 02:10	LOADED	1000000	9.975	10.041	10.001	10.019	9.973	50.002
2012/07/23 02:10	LOADED	1000000	9.995	9.991	10.035	9.962	10.02	50.007
2012/07/23 02:12	LOADED	1000000	9.96	9.987	10.059	10.056	10.018	49.929
2012/07/23 02:12	LOADED	1000000	9.982	9.996	10.019	9.978	10.009	50.025
2012/07/23 02:13	LOADED	1000000	10.025	10.001	10.088	9.948	9.997	49.949

APPENDIX B

```
# An Octave script using HMM functions

transprob = [0.9, 0.1; 0.1, 0.9];
outprob = [0.167, 0.167, 0.167 0.167, 0.167, 0.167; 0.1, 0.1, 0.1, 0.1, 0.1, 0.5];
sequence = [1,6,3,3,4,3,3,5,5,1,6,6,6,2,6,5,6,4,3,3,2,1,5,1,6,3,4,6,2,5]
vpath = hmmviterbi (sequence, transprob, outprob)
```

Figure B1 An Octave Script (hmm.m) Executing a Specific HMM model

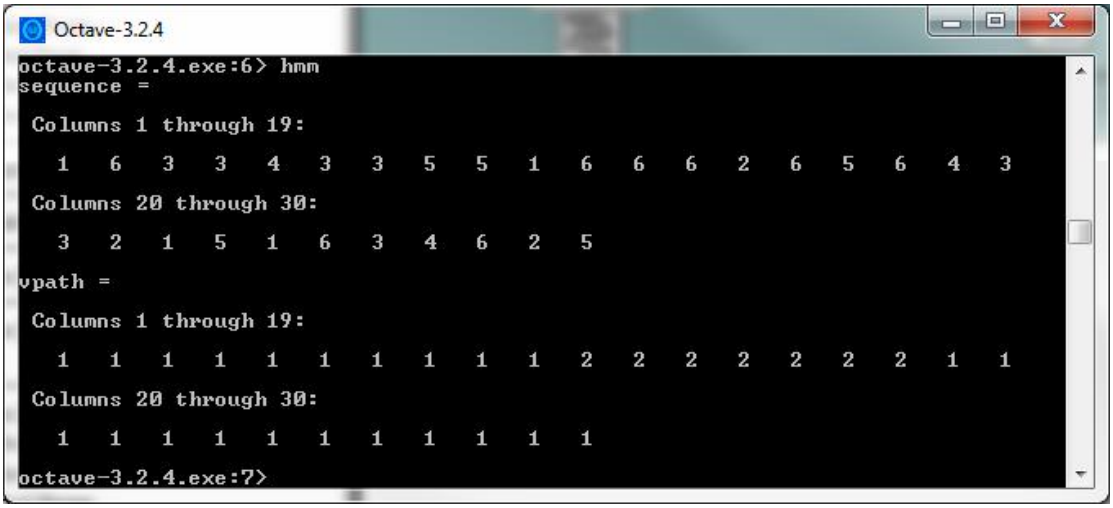

Figure B2 The Octave Output of the Script (1 = Fair State, 2 = Loaded State)

APPENDIX C

Starting with the centroid calculation for vectors of one dimension at time t = M where it is assumed that a new input vector is received with every time stamp. The other assumption is that all vectors contribute to the new centroid C_M:

$$C_M = \frac{1}{M}(x_1 + x_2 + x_3 + ... + x_M) \tag{54}$$

Consider the next centroid calculation for the next data point: M + 1:

$$C_{M+1} = \frac{1}{(M+1)}(x_1 + x_2 + x_3 + ... + x_M + x_{M+1}) \tag{55}$$

Factorise and multiply both sides by (M+1):

$$C_{M+1}(M+1) = (x_1 + x_2 + x_3 + ... + x_M) + x_{M+1} \tag{56}$$

Divide by M on both sides:

$$\frac{C_{M+1}(M+1)}{M} = \frac{1}{M}(x_1 + x_2 + x_3 + ... + x_M) + \frac{x_{M+1}}{M} \tag{57}$$

Substitute C_M from **Equation (54)** :

$$\frac{C_{M+1}(M+1)}{M} = C_M + \frac{x_{M+1}}{M} \tag{58}$$

Make C_{M+1} the subject of the formula:

$$C_{M+1} = \frac{M}{M+1}C_M + \frac{x_{M+1}}{M+1} \tag{59}$$

Let N = M + 1 where N represents the current time stamp:

$$C_N = \frac{N-1}{N} C_{N-1} + \frac{x_N}{N} \tag{60}$$

Expanding **Equation (60)** to cater for all dimensions i at time t = N (where $x(i)_N$ represents the i^{th} element of the input vector at time t = N) the final centroid calculation (for the i^{th} element of the new centroid at time t = N) is:

$$C(i)_N = \frac{N-1}{N} C(i)_{N-1} + \frac{x(i)_N}{N} \tag{61}$$

The implications of **Equation (61)** are:

- The i^{th} element of the new centroid is a function of the i^{th} element of the previous centroid.
- No storage of the input vector set is required.
- The calculation requires no iterating over a large set of vector elements and should therefore be less computationally intensive.

APPENDIX D

Table D1 Original Scenario Results of HMM FST Algorithm Test with K-Means Generated Codebook and Buffer Length = 10

Froth File Index	Replay Start Time	Old Detection Offset	New Detection Time	New Detection Offset	Delay
1	05 Feb 2013 12:45:46	781	13 Aug 2012 11:57:50	622	-159
2	05 Feb 2013 13:06:46	781	17 Jun 2012 07:53:52	724	-57
3	05 Feb 2013 13:27:46	782	20 Aug 2012 11:41:44	716	-66
3	**05 Feb 2013 13:27:46**	**782**	**20 Aug 2012 11:47:16**	**1048**	**266**
4	05 Feb 2013 13:48:48	782	20 Aug 2012 12:10:14	330	-452
4	**05 Feb 2013 13:48:48**	**782**	**20 Aug 2012 12:15:20**	**636**	**-146**
5	05 Feb 2013 14:09:50	782	23 Aug 2012 13:23:58	648	-134
6	05 Feb 2013 14:30:52	781	24 Sep 2012 02:05:44	660	-121
7	05 Feb 2013 14:51:52	782	25 Aug 2012 03:13:20	728	-54
8	05 Feb 2013 15:12:54	782	25 Aug 2012 03:37:38	712	-70
9	05 Feb 2013 15:33:56	782	25 Aug 2012 11:29:18	498	-284
10	05 Feb 2013 15:54:58	782	26 Aug 2012 19:08:22	314	-468
10	**05 Feb 2013 15:54:58**	**782**	**26 Aug 2012 19:14:04**	**656**	**-126**
11	05 Feb 2013 16:16:00	782	27 Aug 2012 21:50:00	706	-76
12	**No Detection**	**No Detection**	**No Detection**	**No Detection**	**No Detection**
13	05 Feb 2013 16:58:04	782	29 Aug 2012 05:49:34	582	-200
13	**05 Feb 2013 16:58:04**	**782**	**29 Aug 2012 05:59:40**	**1188**	**406**
14	05 Feb 2013 17:19:06	782	29 Aug 2012 07:11:40	564	-218
15	05 Feb 2013 17:40:08	782	29 Jul 2012 19:55:04	632	-150
15	**05 Feb 2013 17:40:08**	**782**	**29 Jul 2012 20:01:40**	**1028**	**246**
16	05 Feb 2013 18:01:10	781	03 Oct 2012 11:23:08	676	-105
16	**05 Feb 2013 18:01:10**	**781**	**03 Oct 2012 11:31:42**	**1190**	**409**
17	05 Feb 2013 18:22:10	781	03 Oct 2012 12:15:20	328	-453
18	05 Feb 2013 19:47:56	781	03 Oct 2012 19:44:22	368	-413
19	05 Feb 2013 20:08:56	782	30 Aug 2012 13:35:30	536	-246
19	**05 Feb 2013 20:08:56**	**782**	**30 Aug 2012 13:43:00**	**986**	**204**
20	05 Feb 2013 20:29:58	782	30 Jul 2012 15:41:46	614	-168
21	05 Feb 2013 20:51:00	782	30 Jul 2012 22:19:42	494	-288
22	05 Feb 2013 21:12:02	782	31 Jul 2012 07:47:06	724	-58
23	05 Feb 2013 21:33:04	782	31 Jul 2012 11:58:08	636	-146
24	05 Feb 2013 21:54:06	781	04 Oct 2012 17:32:14	566	-215
24	**05 Feb 2013 21:54:06**	**781**	**04 Oct 2012 17:39:34**	**1006**	**225**
25	05 Feb 2013 22:15:06	782	09 Jun 2012 06:00:14	758	-24
26	05 Feb 2013 22:36:08	782	01 Sep 2012 12:00:50	670	-112
27	05 Feb 2013 22:57:10	782	14 Sep 2012 08:31:46	788	6
27	**05 Feb 2013 22:57:10**	**782**	**14 Sep 2012 08:38:30**	**1192**	**410**
28	05 Feb 2013 23:18:12	782	14 Sep 2012 09:01:16	710	-72
29	05 Feb 2013 23:39:14	782	14 Sep 2012 09:35:20	770	-12
30	06 Feb 2013 00:00:16	781	15 Sep 2012 00:16:32	670	-111
31	06 Feb 2013 00:21:16	781	15 Sep 2012 03:51:32	344	-437
32	06 Feb 2013 00:42:16	782	19 Sep 2012 21:46:08	632	-150

33	06 Feb 2013 01:03:18	781	21 Sep 2012 18:28:20	770	-11
34	06 Feb 2013 01:24:19	781	21 Sep 2012 18:59:34	794	13
35	06 Feb 2013 01:45:19	781	21 Sep 2012 19:42:08	800	19
36	06 Feb 2013 02:06:19	781	22 Sep 2012 16:15:12	650	-131
37	06 Feb 2013 02:27:19	781	22 Sep 2012 22:14:18	312	-469
38	06 Feb 2013 02:48:19	781	24 Sep 2012 13:01:50	488	-293
39	06 Feb 2013 03:09:19	782	25 Jun 2012 23:31:28	712	-70
40	06 Feb 2013 03:30:21	782	26 Jun 2012 07:38:56	668	-114
41	06 Feb 2013 03:51:23	782	26 Jun 2012 21:38:44	714	-68
42	06 Feb 2013 04:12:25	781	27 Sep 2012 05:39:40	308	-473
42	**06 Feb 2013 04:12:25**	**781**	**27 Sep 2012 05:44:54**	**622**	**-159**
43	06 Feb 2013 04:33:25	781	27 Sep 2012 18:04:18	740	-41
43	**06 Feb 2013 04:33:25**	**781**	**27 Sep 2012 18:11:22**	**1164**	**383**
44	06 Feb 2013 04:54:25	782	28 Jun 2012 07:07:50	788	6
45	06 Feb 2013 05:15:27	782	29 Jul 2012 08:35:48	704	-78
46	06 Feb 2013 05:36:29	782	29 Jul 2012 13:15:22	458	-324
47	06 Feb 2013 05:57:31	782	30 Jul 2012 22:22:54	672	-110
47	**06 Feb 2013 05:57:31**	**782**	**30 Jul 2012 22:29:36**	**1074**	**292**
48	06 Feb 2013 06:18:33	782	30 Jul 2012 22:59:10	634	-148
49	06 Feb 2013 06:39:35	782	30 Jun 2012 08:04:22	778	-4
50	06 Feb 2013 07:00:37	781	04 Oct 2012 12:48:12	766	-15
51	06 Feb 2013 07:21:37	781	07 Sep 2012 14:08:12	390	-391
52	06 Feb 2013 07:42:37	782	01 Jul 2012 13:26:26	640	-142
53	06 Feb 2013 08:03:39	781	01 Oct 2012 02:33:12	698	-83
54	06 Feb 2013 08:24:39	781	01 Oct 2012 10:12:02	690	-91
55	06 Feb 2013 08:45:39	781	01 Oct 2012 19:57:10	750	-31
55	**06 Feb 2013 08:45:39**	**781**	**01 Oct 2012 20:04:36**	**1196**	**415**
56	06 Feb 2013 09:06:39	781	01 Oct 2012 22:39:54	434	-347
57	06 Feb 2013 10:28:37	781	01 Oct 2012 23:22:12	758	-23
58	06 Feb 2013 10:49:36	781	15 Jul 2012 04:00:12	776	-5
59	06 Feb 2013 11:10:36	781	02 Oct 2012 01:08:52	766	-15
60	06 Feb 2013 11:31:38	781	02 Oct 2012 01:41:12	728	-53
61	06 Feb 2013 11:52:38	781	02 Oct 2012 11:54:12	772	-9
62	06 Feb 2013 12:13:38	781	02 Oct 2012 14:12:22	768	-13
63	06 Feb 2013 12:39:30	781	02 Oct 2012 17:43:26	754	-27
64	06 Feb 2013 13:00:30	782	20 Jul 2012 07:36:52	848	66
65	06 Feb 2013 13:21:32	782	21 Aug 2012 14:15:08	774	-8
66	06 Feb 2013 13:42:34	781	21 Sep 2012 16:31:48	494	-287
66	**06 Feb 2013 13:42:34**	**781**	**21 Sep 2012 16:40:54**	**1040**	**259**
67	06 Feb 2013 14:03:34	781	22 Sep 2012 15:56:40	790	9
68	06 Feb 2013 14:24:34	781	22 Sep 2012 18:20:04	766	-15
69	06 Feb 2013 14:45:34	782	24 Jul 2012 16:07:06	766	-16
70	06 Feb 2013 15:06:36	781	24 Sep 2012 05:40:30	784	3
71	06 Feb 2013 15:27:36	782	25 Apr 2012 06:04:26	644	-138
72	06 Feb 2013 15:48:38	781	25 Sep 2012 14:35:12	788	7
73	06 Feb 2013 16:09:38	781	25 Sep 2012 17:25:30	746	-35
74	06 Feb 2013 16:30:38	781	25 Sep 2012 18:08:06	712	-69
75	06 Feb 2013 16:51:38	781	26 Sep 2012 00:10:14	786	5
76	06 Feb 2013 17:12:38	781	26 Sep 2012 05:03:08	782	1
77	06 Feb 2013 20:38:45	781	26 Sep 2012 09:13:30	772	-9
78	06 Feb 2013 20:59:45	781	26 Sep 2012 18:01:02	812	31
79	06 Feb 2013 21:20:45	781	26 Sep 2012 18:22:52	794	13
80	06 Feb 2013 21:41:45	781	26 Sep 2012 18:40:18	782	1

81	06 Feb 2013 22:02:46	782	27 Apr 2012 18:28:32	788	6
82	06 Feb 2013 22:23:48	782	29 Jul 2012 02:03:28	686	-96
83	06 Feb 2013 22:44:50	782	29 Jun 2012 19:28:18	382	-400
84	06 Feb 2013 23:05:52	781	29 Sep 2012 13:36:32	712	-69
85	06 Feb 2013 23:26:52	781	03 Oct 2012 14:05:02	698	-83
86	06 Feb 2013 23:47:52	781	03 Oct 2012 15:25:06	784	3
87	07 Feb 2013 00:08:52	781	03 Oct 2012 23:18:52	622	-159
88	07 Feb 2013 00:29:52	782	30 Jul 2012 02:08:16	782	0
89	07 Feb 2013 00:50:54	781	30 Sep 2012 11:03:24	606	-175
90	07 Feb 2013 01:11:54	781	30 Sep 2012 13:07:52	782	1
91	07 Feb 2013 01:32:54	781	30 Sep 2012 13:47:40	762	-19
92	07 Feb 2013 01:53:54	781	30 Sep 2012 14:23:26	648	-133
93	07 Feb 2013 02:14:54	781	30 Sep 2012 17:05:00	786	5
94	07 Feb 2013 02:35:54	781	30 Sep 2012 19:48:14	744	-37
95	07 Feb 2013 02:56:54	782	31 Aug 2012 01:24:58	780	-2
96	07 Feb 2013 03:17:56	782	31 Jul 2012 02:57:40	742	-40
97	07 Feb 2013 03:38:58	782	31 Jul 2012 03:52:02	786	4
98	07 Feb 2013 04:00:00	782	31 Jul 2012 06:26:10	746	-36
99	07 Feb 2013 04:21:02	782	04 Aug 2012 21:26:10	768	-14
100	07 Feb 2013 04:42:04	781	04 Oct 2012 02:50:40	446	-335
100	**07 Feb 2013 04:42:04**	**781**	**04 Oct 2012 02:55:50**	**756**	**-25**

Table D2 Scenario 1 Results of HMM FST Algorithm Test with Manual Codebook and Buffer Length = 10

Froth File Index	Replay Start Time	Old Detection Offset	New Detection Time	New Detection Offset	Delay
1	28 Dec 2012 03:32:07	781	13 Aug 2012 12:00:48	800	19
2	28 Dec 2012 03:53:07	781	17 Jun 2012 07:55:02	794	13
3	28 Dec 2012 04:14:07	782	20 Aug 2012 11:43:00	792	10
4	28 Dec 2012 04:35:09	782	20 Aug 2012 12:17:32	768	-14
5	28 Dec 2012 04:56:11	782	23 Aug 2012 13:26:36	806	24
6	28 Dec 2012 05:17:13	781	24 Sep 2012 02:07:56	792	11
7	28 Dec 2012 05:38:13	782	25 Aug 2012 03:14:44	812	30
8	28 Dec 2012 05:59:15	782	25 Aug 2012 03:37:50	724	-58
9	28 Dec 2012 06:20:17	782	25 Aug 2012 11:35:12	852	70
10	28 Dec 2012 06:41:19	782	26 Aug 2012 19:17:32	864	82
11	28 Dec 2012 07:02:21	782	27 Aug 2012 21:52:32	858	76
12	**28 Dec 2012 07:23:23**	**782**	**28 Jun 2012 05:26:20**	**334**	**-448**
12	28 Dec 2012 07:23:23	782	28 Jun 2012 05:31:26	640	-142
12	**28 Dec 2012 07:23:23**	**782**	**28 Jun 2012 05:38:58**	**1092**	**310**
13	28 Dec 2012 07:44:25	782	29 Aug 2012 05:52:22	750	-32
14	28 Dec 2012 08:05:27	782	29 Aug 2012 07:12:46	630	-152
15	28 Dec 2012 08:26:29	782	29 Jul 2012 19:57:54	802	20
16	28 Dec 2012 08:47:31	781	03 Oct 2012 11:24:22	750	-31
17	28 Dec 2012 09:08:31	781	03 Oct 2012 12:22:56	784	3
18	28 Dec 2012 09:29:31	781	03 Oct 2012 19:46:48	514	-267
19	28 Dec 2012 09:50:31	782	30 Aug 2012 13:39:30	776	-6
20	28 Dec 2012 10:11:33	782	30 Jul 2012 15:44:48	796	14
21	28 Dec 2012 10:32:35	782	30 Jul 2012 22:22:34	666	-116

22	28 Dec 2012 10:53:37	782	31 Jul 2012 07:49:18	856	74
23	28 Dec 2012 11:14:39	782	31 Jul 2012 12:02:52	920	138
24	28 Dec 2012 11:35:41	781	04 Oct 2012 17:36:56	848	67
25	28 Dec 2012 11:56:41	782	09 Jun 2012 06:01:02	806	24
26	28 Dec 2012 12:17:43	782	01 Sep 2012 12:02:44	784	2
27	28 Dec 2012 12:38:45	782	14 Sep 2012 08:32:44	846	64
28	28 Dec 2012 12:59:48	782	14 Sep 2012 09:03:24	838	56
29	28 Dec 2012 13:20:50	782	14 Sep 2012 09:36:48	858	76
30	28 Dec 2012 13:41:52	781	15 Sep 2012 00:18:42	800	19
31	28 Dec 2012 14:02:52	781	15 Sep 2012 03:53:46	478	-303
32	28 Dec 2012 14:23:53	782	19 Sep 2012 21:49:06	810	28
33	28 Dec 2012 14:44:54	781	21 Sep 2012 18:29:16	826	45
34	28 Dec 2012 15:05:54	781	21 Sep 2012 19:00:22	842	61
35	28 Dec 2012 15:26:54	781	21 Sep 2012 19:42:56	848	67
36	28 Dec 2012 15:47:54	781	22 Sep 2012 16:17:54	812	31
37	28 Dec 2012 16:08:55	781	22 Sep 2012 22:18:14	548	-233
38	28 Dec 2012 16:29:55	781	24 Sep 2012 13:07:02	800	19
39	28 Dec 2012 16:50:55	782	25 Jun 2012 23:32:50	794	12
40	28 Dec 2012 17:11:57	782	26 Jun 2012 07:41:56	848	66
41	28 Dec 2012 17:32:59	782	26 Jun 2012 21:40:44	834	52
42	28 Dec 2012 17:54:02	781	27 Sep 2012 05:47:42	790	9
43	28 Dec 2012 18:15:01	781	27 Sep 2012 18:06:14	856	75
44	28 Dec 2012 18:36:02	782	28 Jun 2012 07:09:28	886	104
45	28 Dec 2012 18:57:03	782	29 Jul 2012 08:37:06	782	0
46	28 Dec 2012 19:18:05	782	29 Jul 2012 13:20:36	772	-10
47	28 Dec 2012 19:39:07	782	30 Jul 2012 22:24:58	796	14
48	28 Dec 2012 20:00:09	782	30 Jul 2012 23:02:20	824	42
49	28 Dec 2012 20:21:11	782	30 Jun 2012 08:06:02	878	96
50	28 Dec 2012 20:42:13	781	04 Oct 2012 12:49:48	862	81
51	28 Dec 2012 21:03:13	781	07 Sep 2012 14:14:58	796	15
52	28 Dec 2012 21:24:13	782	01 Jul 2012 13:29:26	820	38
53	28 Dec 2012 21:45:15	781	01 Oct 2012 02:36:06	872	91
54	28 Dec 2012 22:06:15	781	01 Oct 2012 10:14:50	858	77
55	28 Dec 2012 22:27:15	781	01 Oct 2012 19:57:56	796	15
56	28 Dec 2012 22:48:16	781	01 Oct 2012 22:45:14	754	-27
57	28 Dec 2012 23:09:16	781	01 Oct 2012 23:22:54	800	19
58	28 Dec 2012 23:30:16	781	15 Jul 2012 04:01:10	834	53
59	28 Dec 2012 23:51:16	781	02 Oct 2012 01:09:54	828	47
60	29 Dec 2012 00:12:16	781	02 Oct 2012 01:42:20	796	15
61	29 Dec 2012 00:33:16	781	02 Oct 2012 11:55:10	830	49
62	29 Dec 2012 00:54:16	781	02 Oct 2012 14:12:50	796	15
63	29 Dec 2012 01:15:16	781	02 Oct 2012 17:44:52	840	59
64	29 Dec 2012 01:36:17	782	20 Jul 2012 07:37:04	860	78
65	29 Dec 2012 01:57:19	782	21 Aug 2012 14:16:10	836	54
66	29 Dec 2012 02:18:21	781	21 Sep 2012 16:37:34	840	59
67	29 Dec 2012 02:39:22	781	22 Sep 2012 15:57:40	850	69
68	29 Dec 2012 03:00:21	781	22 Sep 2012 18:21:16	838	57
69	29 Dec 2012 03:21:21	782	24 Jul 2012 16:08:18	838	56
70	29 Dec 2012 03:42:23	781	24 Sep 2012 05:41:30	844	63
71	29 Dec 2012 04:03:23	782	25 Apr 2012 06:06:56	794	12
72	29 Dec 2012 04:24:25	781	25 Sep 2012 14:35:20	796	15
73	29 Dec 2012 04:45:25	781	25 Sep 2012 17:26:04	780	-1
74	29 Dec 2012 05:06:26	781	25 Sep 2012 18:10:16	842	61

75	29 Dec 2012 05:27:26	781	26 Sep 2012 00:11:24	856	75
76	29 Dec 2012 05:48:26	781	26 Sep 2012 05:04:12	846	65
77	29 Dec 2012 06:09:26	781	26 Sep 2012 09:14:30	832	51
78	29 Dec 2012 06:30:26	781	26 Sep 2012 18:01:06	816	35
79	29 Dec 2012 06:51:26	781	26 Sep 2012 18:23:34	836	55
80	29 Dec 2012 07:12:26	781	26 Sep 2012 18:41:56	880	99
81	29 Dec 2012 07:33:26	782	27 Apr 2012 18:28:50	806	24
82	29 Dec 2012 07:54:28	782	29 Jul 2012 02:05:24	802	20
83	29 Dec 2012 08:15:30	782	29 Jun 2012 19:30:42	526	-256
84	29 Dec 2012 08:36:32	781	29 Sep 2012 13:37:58	798	17
85	29 Dec 2012 08:57:32	781	03 Oct 2012 14:07:00	816	35
86	29 Dec 2012 09:18:32	781	03 Oct 2012 15:26:16	854	73
87	29 Dec 2012 09:39:32	781	03 Oct 2012 23:22:12	822	41
88	29 Dec 2012 10:00:32	782	30 Jul 2012 02:09:16	842	60
89	29 Dec 2012 10:21:34	781	30 Sep 2012 11:06:36	798	17
90	29 Dec 2012 10:42:34	781	30 Sep 2012 13:08:16	806	25
91	29 Dec 2012 11:03:34	781	30 Sep 2012 13:48:32	814	33
92	29 Dec 2012 11:24:34	781	30 Sep 2012 14:25:46	788	7
93	29 Dec 2012 11:45:34	781	30 Sep 2012 17:05:20	806	25
94	29 Dec 2012 12:06:34	781	30 Sep 2012 19:49:32	822	41
95	29 Dec 2012 12:27:34	782	31 Aug 2012 01:25:16	798	16
96	29 Dec 2012 12:48:36	782	31 Jul 2012 02:59:32	854	72
97	29 Dec 2012 13:09:38	782	31 Jul 2012 03:52:24	808	26
98	29 Dec 2012 13:30:40	782	31 Jul 2012 06:27:20	816	34
99	29 Dec 2012 13:51:42	782	04 Aug 2012 21:26:32	790	8
100	29 Dec 2012 14:12:44	781	04 Oct 2012 02:56:36	802	21

Table D3 Scenario 2 Results of HMM FST Algorithm Test with Manual Codebook and Reduced Buffer Length = 5

Froth File Index	Replay Start Time	Old Detection Offset	New Detection Time	New Detection Offset	Delay	Delay L = 10
1	29 Dec 2012 14:29:19	781	13 Aug 2012 12:00:38	790	9	19
2	29 Dec 2012 14:50:19	781	17 Jun 2012 07:54:52	784	3	13
3	29 Dec 2012 15:11:19	782	20 Aug 2012 11:42:30	762	-20	10
4	29 Dec 2012 15:32:21	782	20 Aug 2012 12:17:22	758	-24	-14
5	29 Dec 2012 15:53:23	782	23 Aug 2012 13:26:26	796	14	24
6	29 Dec 2012 16:14:27	781	24 Sep 2012 02:07:46	782	1	11
7	29 Dec 2012 16:35:27	782	25 Aug 2012 03:14:34	802	20	30
8	29 Dec 2012 16:56:29	782	25 Aug 2012 03:37:40	714	-68	-58
9	29 Dec 2012 17:17:31	782	25 Aug 2012 11:35:02	842	60	70
10	29 Dec 2012 17:38:33	782	26 Aug 2012 19:16:18	790	8	82
11	29 Dec 2012 17:59:35	782	27 Aug 2012 21:52:22	848	66	76
12	**29 Dec 2012 18:20:37**	**782**	**28 Jun 2012 05:26:10**	**324**	**-458**	**0**
12	29 Dec 2012 18:20:37	782	28 Jun 2012 05:31:16	630	-152	-142
12	**29 Dec 2012 18:20:37**	**782**	**28 Jun 2012 05:38:48**	**1082**	**300**	**0**
13	29 Dec 2012 18:41:39	782	29 Aug 2012 05:52:12	740	-42	-32
14	29 Dec 2012 19:02:41	782	29 Aug 2012 07:12:36	620	-162	-152
15	29 Dec 2012 19:23:44	782	29 Jul 2012 19:57:44	792	10	20
16	29 Dec 2012 19:44:45	781	03 Oct 2012 11:24:12	740	-41	-31

17	29 Dec 2012 20:05:45	781	03 Oct 2012 12:22:46	774	-7	3
18	29 Dec 2012 20:26:45	781	03 Oct 2012 19:45:44	450	-331	-267
19	29 Dec 2012 20:47:45	782	30 Aug 2012 13:38:06	692	-90	-6
20	29 Dec 2012 21:08:48	782	30 Jul 2012 15:44:38	786	4	14
21	29 Dec 2012 21:29:49	782	30 Jul 2012 22:22:24	656	-126	-116
22	29 Dec 2012 21:50:51	782	31 Jul 2012 07:48:08	786	4	74
23	29 Dec 2012 22:11:55	782	31 Jul 2012 12:01:06	814	32	138
24	29 Dec 2012 22:32:57	781	04 Oct 2012 17:36:46	838	57	67
25	29 Dec 2012 22:53:57	782	09 Jun 2012 06:00:52	796	14	24
26	29 Dec 2012 23:14:59	782	01 Sep 2012 12:02:34	774	-8	2
27	29 Dec 2012 23:36:01	782	14 Sep 2012 08:32:06	808	26	64
28	29 Dec 2012 23:57:03	782	14 Sep 2012 09:03:14	828	46	56
29	30 Dec 2012 00:18:05	782	14 Sep 2012 09:36:38	848	66	76
30	30 Dec 2012 00:39:08	781	15 Sep 2012 00:18:32	790	9	19
31	30 Dec 2012 01:00:08	781	15 Sep 2012 03:53:36	468	-313	-303
32	30 Dec 2012 01:21:08	782	19 Sep 2012 21:48:56	800	18	28
33	30 Dec 2012 01:42:10	781	21 Sep 2012 18:29:06	816	35	45
34	30 Dec 2012 02:03:10	781	21 Sep 2012 18:59:38	798	17	61
35	30 Dec 2012 02:24:10	781	21 Sep 2012 19:42:46	838	57	67
36	30 Dec 2012 02:45:10	781	22 Sep 2012 16:17:22	780	-1	31
37	30 Dec 2012 03:06:11	781	22 Sep 2012 22:18:04	538	-243	-233
38	30 Dec 2012 03:27:12	781	24 Sep 2012 13:06:52	790	9	19
39	30 Dec 2012 03:48:12	782	25 Jun 2012 23:32:40	784	2	12
40	30 Dec 2012 04:09:15	782	26 Jun 2012 07:41:46	838	56	66
41	30 Dec 2012 04:30:16	782	26 Jun 2012 21:40:34	824	42	52
42	30 Dec 2012 07:54:51	781	27 Sep 2012 05:47:32	780	-1	9
43	30 Dec 2012 08:15:52	781	27 Sep 2012 18:06:04	846	65	75
44	30 Dec 2012 08:36:53	782	28 Jun 2012 07:09:02	860	78	104
45	30 Dec 2012 08:57:55	782	29 Jul 2012 08:36:56	772	-10	0
46	30 Dec 2012 09:18:57	782	29 Jul 2012 13:20:26	762	-20	-10
47	30 Dec 2012 09:39:59	782	30 Jul 2012 22:22:58	676	-106	14
48	30 Dec 2012 10:01:01	782	30 Jul 2012 23:02:10	814	32	42
49	30 Dec 2012 10:22:03	782	30 Jun 2012 08:05:52	868	86	96
50	30 Dec 2012 10:43:05	781	04 Oct 2012 12:49:38	852	71	81
51	30 Dec 2012 11:04:05	781	07 Sep 2012 14:14:48	786	5	15
52	30 Dec 2012 11:25:05	782	01 Jul 2012 13:29:16	810	28	38
53	30 Dec 2012 11:46:07	781	01 Oct 2012 02:34:44	790	9	91
54	30 Dec 2012 12:07:07	781	01 Oct 2012 10:14:40	848	67	77
55	30 Dec 2012 12:28:07	781	01 Oct 2012 19:57:46	786	5	15
56	30 Dec 2012 12:49:07	781	01 Oct 2012 22:45:04	744	-37	-27
57	30 Dec 2012 13:10:08	781	01 Oct 2012 23:22:44	790	9	19
58	30 Dec 2012 13:31:08	781	15 Jul 2012 04:00:24	788	7	53
59	30 Dec 2012 13:52:08	781	02 Oct 2012 01:09:44	818	37	47
60	30 Dec 2012 14:13:08	781	02 Oct 2012 01:42:10	786	5	15
61	30 Dec 2012 14:34:08	781	02 Oct 2012 11:55:00	820	39	49
62	30 Dec 2012 14:55:08	781	02 Oct 2012 14:12:40	786	5	15
63	30 Dec 2012 15:16:08	781	02 Oct 2012 17:44:42	830	49	59
64	30 Dec 2012 15:37:08	782	20 Jul 2012 07:36:54	850	68	78
65	30 Dec 2012 15:58:10	782	21 Aug 2012 14:16:00	826	44	54
66	30 Dec 2012 16:19:12	781	21 Sep 2012 16:37:24	830	49	59
67	30 Dec 2012 16:40:12	781	22 Sep 2012 15:57:30	840	59	69
68	30 Dec 2012 17:01:12	781	22 Sep 2012 18:21:06	828	47	57
69	30 Dec 2012 17:22:12	782	24 Jul 2012 16:08:08	828	46	56

70	30 Dec 2012 17:43:14	781	24 Sep 2012 05:41:20	834	53	63
71	30 Dec 2012 18:04:14	782	25 Apr 2012 06:06:46	784	2	12
72	30 Dec 2012 18:25:16	781	25 Sep 2012 14:35:10	786	5	15
73	30 Dec 2012 18:46:16	781	25 Sep 2012 17:25:54	770	-11	-1
74	30 Dec 2012 19:07:16	781	25 Sep 2012 18:10:06	832	51	61
75	30 Dec 2012 19:28:16	781	26 Sep 2012 00:11:14	846	65	75
76	30 Dec 2012 19:49:16	781	26 Sep 2012 05:04:02	836	55	65
77	30 Dec 2012 20:10:16	781	26 Sep 2012 09:14:20	822	41	51
78	30 Dec 2012 20:31:16	781	26 Sep 2012 18:00:56	806	25	35
79	30 Dec 2012 20:52:16	781	26 Sep 2012 18:22:50	792	11	55
80	30 Dec 2012 21:13:16	781	26 Sep 2012 18:41:26	850	69	99
81	30 Dec 2012 21:34:16	782	27 Apr 2012 18:28:40	796	14	24
82	30 Dec 2012 21:55:18	782	29 Jul 2012 02:05:14	792	10	20
83	30 Dec 2012 22:16:20	782	29 Jun 2012 19:30:32	516	-266	-256
84	30 Dec 2012 22:37:23	781	29 Sep 2012 13:37:48	788	7	17
85	30 Dec 2012 22:58:22	781	03 Oct 2012 14:06:36	792	11	35
86	30 Dec 2012 23:19:28	781	03 Oct 2012 15:26:06	844	63	73
87	30 Dec 2012 23:40:28	781	03 Oct 2012 23:22:02	812	31	41
88	31 Dec 2012 00:01:29	782	30 Jul 2012 02:08:26	792	10	60
89	31 Dec 2012 00:22:30	781	30 Sep 2012 11:06:26	788	7	17
90	31 Dec 2012 00:43:31	781	30 Sep 2012 13:08:06	796	15	25
91	31 Dec 2012 01:04:31	781	30 Sep 2012 13:48:22	804	23	33
92	31 Dec 2012 01:25:31	781	30 Sep 2012 14:25:36	778	-3	7
93	31 Dec 2012 01:46:31	781	30 Sep 2012 17:05:10	796	15	25
94	31 Dec 2012 02:07:31	781	30 Sep 2012 19:49:22	812	31	41
95	31 Dec 2012 02:28:31	782	31 Aug 2012 01:25:06	788	6	16
96	31 Dec 2012 02:49:33	782	31 Jul 2012 02:59:22	844	62	72
97	31 Dec 2012 03:10:35	782	31 Jul 2012 03:52:14	798	16	26
98	31 Dec 2012 03:31:42	782	31 Jul 2012 06:27:10	806	24	34
99	31 Dec 2012 03:52:46	782	04 Aug 2012 21:26:22	780	-2	8
100	31 Dec 2012 04:13:49	781	04 Oct 2012 02:56:26	792	11	21

Table D4 Scenario 3 Results of HMM FST Algorithm Test with Original Codebook, Buffer Length = 10 and Modified B Matrix

Froth File Index	Replay Start Time	Old Detection Offset	New Detection Time	New Detection Offset	Delay
1	11 Feb 2013 21:10:16	781	13 Aug 2012 12:00:52	804	23
2	11 Feb 2013 21:31:16	781	17 Jun 2012 07:54:48	780	-1
3	11 Feb 2013 21:52:16	782	20 Aug 2012 11:42:46	778	-4
4	11 Feb 2013 22:13:18	782	20 Aug 2012 12:17:20	756	-26
5	11 Feb 2013 22:34:20	782	23 Aug 2012 13:26:24	794	12
6	11 Feb 2013 22:55:22	781	24 Sep 2012 02:08:10	806	25
7	11 Feb 2013 23:16:22	782	25 Aug 2012 03:14:40	808	26
8	11 Feb 2013 23:37:24	782	25 Aug 2012 03:38:46	780	-2
9	11 Feb 2013 23:58:26	782	25 Aug 2012 11:35:12	852	70
10	12 Feb 2013 00:19:28	782	26 Aug 2012 19:08:22	314	-468
10	**12 Feb 2013 00:19:28**	**782**	**26 Aug 2012 19:17:20**	**852**	**70**
11	12 Feb 2013 00:40:30	782	27 Aug 2012 21:52:18	844	62
12	12 Feb 2013 01:01:32	782	28 Jun 2012 05:32:26	700	-82

13	12 Feb 2013 01:22:35	782	29 Aug 2012 05:52:20	748	-34
14	12 Feb 2013 01:43:37	782	29 Aug 2012 07:15:12	776	-6
15	12 Feb 2013 02:04:39	782	29 Jul 2012 19:58:08	816	34
16	12 Feb 2013 02:25:41	781	03 Oct 2012 11:24:18	746	-35
17	12 Feb 2013 02:46:41	781	03 Oct 2012 12:15:20	328	-453
17	**12 Feb 2013 02:46:41**	**781**	**03 Oct 2012 12:22:48**	**776**	**-5**
18	12 Feb 2013 03:07:41	781	03 Oct 2012 19:47:02	528	-253
19	12 Feb 2013 03:28:41	782	30 Aug 2012 13:39:20	766	-16
20	12 Feb 2013 03:49:43	782	30 Jul 2012 15:45:10	818	36
21	12 Feb 2013 04:10:45	782	30 Jul 2012 22:22:18	650	-132
22	12 Feb 2013 04:31:47	782	31 Jul 2012 07:48:08	786	4
23	12 Feb 2013 04:52:49	782	31 Jul 2012 12:02:38	906	124
24	12 Feb 2013 05:13:51	781	04 Oct 2012 17:32:14	566	-215
24	**12 Feb 2013 05:13:51**	**781**	**04 Oct 2012 17:39:34**	**1006**	**225**
25	12 Feb 2013 05:34:51	782	09 Jun 2012 06:01:36	840	58
26	12 Feb 2013 05:55:53	782	01 Sep 2012 12:02:36	776	-6
27	12 Feb 2013 06:16:55	782	14 Sep 2012 08:32:38	840	58
27	**12 Feb 2013 06:16:55**	**782**	**14 Sep 2012 08:38:30**	**1192**	**410**
28	12 Feb 2013 06:37:57	782	14 Sep 2012 09:03:16	830	48
29	12 Feb 2013 06:58:59	782	14 Sep 2012 09:36:40	850	68
30	12 Feb 2013 07:20:01	781	15 Sep 2012 00:18:28	786	5
31	12 Feb 2013 07:41:01	781	15 Sep 2012 03:54:28	520	-261
32	12 Feb 2013 11:20:43	782	19 Sep 2012 21:50:16	880	98
33	12 Feb 2013 11:41:45	781	21 Sep 2012 18:29:08	818	37
34	12 Feb 2013 12:02:45	781	21 Sep 2012 19:00:08	828	47
35	12 Feb 2013 12:23:45	781	21 Sep 2012 19:42:42	834	53
36	12 Feb 2013 12:44:45	781	22 Sep 2012 16:18:06	824	43
37	12 Feb 2013 13:05:45	781	22 Sep 2012 22:20:06	660	-121
38	12 Feb 2013 16:40:09	781	24 Sep 2012 13:06:54	792	11
39	12 Feb 2013 17:01:09	782	25 Jun 2012 23:32:36	780	-2
40	12 Feb 2013 17:22:11	782	26 Jun 2012 07:41:42	834	52
41	12 Feb 2013 18:23:51	782	26 Jun 2012 21:40:32	822	40
42	12 Feb 2013 18:44:53	781	27 Sep 2012 05:48:40	848	67
43	12 Feb 2013 19:05:53	781	27 Sep 2012 18:06:12	854	73
44	12 Feb 2013 19:26:53	782	28 Jun 2012 07:09:20	878	96
45	12 Feb 2013 19:47:55	782	29 Jul 2012 08:37:16	792	10
46	12 Feb 2013 20:08:57	782	29 Jul 2012 13:20:38	774	-8
47	12 Feb 2013 20:29:59	782	30 Jul 2012 22:24:44	782	0
48	12 Feb 2013 20:51:02	782	30 Jul 2012 23:02:20	824	42
49	12 Feb 2013 21:12:04	782	30 Jun 2012 08:05:50	866	84
50	12 Feb 2013 21:33:06	781	04 Oct 2012 12:49:42	856	75
51	12 Feb 2013 21:54:06	781	07 Sep 2012 14:15:28	826	45
52	12 Feb 2013 22:15:06	782	01 Jul 2012 13:29:16	810	28
53	12 Feb 2013 22:36:08	781	01 Oct 2012 02:35:52	858	77
54	12 Feb 2013 22:57:08	781	01 Oct 2012 10:14:38	846	65
55	12 Feb 2013 23:18:08	781	01 Oct 2012 19:57:44	784	3
55	**12 Feb 2013 23:18:08**	**781**	**01 Oct 2012 20:04:36**	**1196**	**415**
56	12 Feb 2013 23:39:08	781	01 Oct 2012 22:45:04	744	-37
57	13 Feb 2013 00:00:08	781	01 Oct 2012 23:22:50	796	15
58	13 Feb 2013 00:21:08	781	15 Jul 2012 04:00:56	820	39
59	13 Feb 2013 00:42:08	781	02 Oct 2012 01:09:40	814	33
60	13 Feb 2013 01:03:08	781	02 Oct 2012 01:42:06	782	1
61	13 Feb 2013 01:24:08	781	02 Oct 2012 11:55:02	822	41

62	13 Feb 2013 01:45:08	781	02 Oct 2012 14:12:34	780	-1
63	13 Feb 2013 02:06:08	781	02 Oct 2012 17:44:38	826	45
64	13 Feb 2013 02:27:08	782	20 Jul 2012 07:36:52	848	66
65	13 Feb 2013 02:48:10	782	21 Aug 2012 14:15:08	774	-8
66	13 Feb 2013 03:09:12	781	21 Sep 2012 16:37:20	826	45
67	13 Feb 2013 03:30:12	781	22 Sep 2012 15:57:26	836	55
68	13 Feb 2013 03:51:13	781	22 Sep 2012 18:21:02	824	43
69	13 Feb 2013 04:12:13	782	24 Jul 2012 16:08:06	826	44
70	13 Feb 2013 04:33:15	781	24 Sep 2012 05:41:22	836	55
71	13 Feb 2013 04:54:15	782	25 Apr 2012 06:06:44	782	0
72	13 Feb 2013 05:15:17	781	25 Sep 2012 14:35:12	788	7
73	13 Feb 2013 05:36:17	781	25 Sep 2012 17:25:58	774	-7
74	13 Feb 2013 05:57:17	781	25 Sep 2012 18:10:10	836	55
75	13 Feb 2013 06:18:17	781	26 Sep 2012 00:11:10	842	61
76	13 Feb 2013 06:39:17	781	26 Sep 2012 05:04:08	842	61
77	13 Feb 2013 07:00:17	781	26 Sep 2012 09:14:18	820	39
78	13 Feb 2013 07:21:17	781	26 Sep 2012 18:01:02	812	31
79	13 Feb 2013 07:42:17	781	26 Sep 2012 18:23:24	826	45
80	13 Feb 2013 08:03:17	781	26 Sep 2012 18:41:42	866	85
81	13 Feb 2013 08:24:18	782	27 Apr 2012 18:28:44	800	18
82	13 Feb 2013 08:45:20	782	29 Jul 2012 02:05:10	788	6
83	13 Feb 2013 09:06:22	782	29 Jun 2012 19:32:02	606	-176
84	13 Feb 2013 09:27:24	781	29 Sep 2012 13:38:26	826	45
85	13 Feb 2013 09:48:24	781	03 Oct 2012 14:07:08	824	43
86	13 Feb 2013 10:09:24	781	03 Oct 2012 15:26:02	840	59
87	13 Feb 2013 10:30:26	781	03 Oct 2012 23:22:18	828	47
88	13 Feb 2013 10:51:26	782	30 Jul 2012 02:09:02	828	46
89	13 Feb 2013 11:12:29	781	30 Sep 2012 11:06:34	796	15
90	13 Feb 2013 11:33:28	781	30 Sep 2012 13:08:06	796	15
91	13 Feb 2013 11:54:29	781	30 Sep 2012 13:48:22	804	23
92	13 Feb 2013 12:15:28	781	30 Sep 2012 14:25:38	780	-1
93	13 Feb 2013 12:36:28	781	30 Sep 2012 17:05:22	808	27
94	13 Feb 2013 12:57:28	781	30 Sep 2012 19:49:16	806	25
95	13 Feb 2013 13:18:28	782	31 Aug 2012 01:25:10	792	10
96	13 Feb 2013 13:39:31	782	31 Jul 2012 02:59:20	842	60
97	13 Feb 2013 14:00:32	782	31 Jul 2012 03:52:14	798	16
98	13 Feb 2013 14:21:34	782	31 Jul 2012 06:27:20	816	34
99	13 Feb 2013 14:42:36	782	04 Aug 2012 21:26:34	792	10
100	13 Feb 2013 15:03:38	781	04 Oct 2012 02:56:30	796	15

APPENDIX E

This section contains pseudo code for the main algorithms discussed in this dissertation.

The Forward Algorithm

// compute $\alpha_0(i)$
for i = 0 to N − 1
    $\alpha_0(i) = \pi_i \, b_i(O_0)$
next i

// compute $\alpha_t(i)$
for t = 1 to T − 1
    for i = 0 to N − 1
        $\alpha_t(i) = 0$

        for j = 0 to N − 1
            $\alpha_t(i) = \alpha_t(i) + \alpha_{t-1}(j)a_{ji}$
        next j

        $\alpha_t(i) = \alpha_t(i) \, b_i(O_t)$
    next i
next t

//Sum up all alpha variables for last time stamp to get the forward probability
for i = 0 to N - 1
    $P(O|\lambda) = P(O|\lambda) + \alpha_{T-1}(i)$
next i

The Viterbi Algorithm

```
//Compute δ0(i) ie t = 0
for i = 0 to N - 1
        δ'0(i) = -1                                   // Initialise Delta max
        δ0(i) = πi bi(O0)

        if δ0(i) > δ'0(i) then                        // Store the back pointer
                δ'0(i) = δ0(i)
                ϕ0(i) = i
        end if
next i

//Compute δt(i)
for t = 1 to T - 1
        for i = 0 to N - 1
                //Compute
                for j = 0 to N - 1
                        // Store values of j that maximise the product
                        ϕt(j) =  argmax[δt-1(i) aji]
                next j
        next i
next t

// Backtrack to get optimum sequence
// Start with t = T - 1
for i = 0 to N - 1
        i'T-1 =argmax[δT-1(i)]                        // Store optimum last state
next i

//Compute the rest of the hidden state sequence that optimises the observations
for t = T - 1 to 1 by -1 do
        //Find previous state that optimises current state
        i't-1= ϕt(i't)
next t
```

The Baum Welch Algorithm

//Initialise the re-estimated matrices to the current HMM matrices

$\pi' = \pi$

$A' = A$

$B' = B$

Const P_{Min} //Define a minimum probability

Const Max_x //Define maximum iterations

for x = Max_x

Calculate $P_x(O|\lambda)$ //Use Forward Algorithm

$\pi' = \gamma_0(i)$

$A' = a_{ij}$ iterated over all i and j //Using **Equation (30)**

$B' = b_i(k)$ iterated over all i and k //Using **Equation (31)**

//Copy the re-estimated matrices back to original

$\pi = \pi'$

$A = A'$

$B = B'$

//Exit if model's probability improvement is too small

Exit if $P_x(O|\lambda) - P_{x-1}(O|\lambda) <= P_{Min}$

next x

The HMM FST Algorithm

//Initialisation
//Build the test sequence for all states and for buffer length L
for i = N - 1
    Store $Q_i(i,L)$
next i

//In real time
//At time = t, calculate the probability of observing each test state for the observation sequence O_L
for i = N - 1
    $P_{max} = \max[P_i(O_L, Q_i| \lambda)]$ //Test for maximum probability
    //Record the state sequence that provides P_{max}
    $i' = \arg[P_{max}]$
next i

$i'_t = i'$ //Optimal state at time t = i'