Log mining to develop a diagnostic and prognostic framework for the MeerLICHT telescope

by

Timothy Brian Roelf

Minor dissertation presented in partial fulfilment of the requirements for the degree of

Master of Science

(Data Science)

in the Faculty of Science, University of Cape Town

Supervisor:
Prof P. Groot
Department of Astronomy

Co-supervisor:
Dr R. Rakotonirainy
Department of Statistics

June 2022

The financial assistance of the National Research Foundation (NRF), under SARChI project 111692, towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

University of Cape Town · iYunivesithi yaseKapa · Universiteit van Kaapstad

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

Plagirism Declaration

I, Timothy Brian Roelf, declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated). And, that I have not previously in its entirety or in part submitted it for obtaining any qualification at this or any other University. I authorise the University to reproduce for the purpose of research either the whole or any portion of the contents in any manner whatsoever.

Date: 2022/06/29 .....

Copyright © 2022 University of Cape Town
All rights reserved.

Abstract

Log mining to develop a diagnostic and prognostic framework for the MeerLICHT telescope

T. B. Roelf

Department of Astronomy,
University of Cape Town,
Private Bag X3, Rondebosch 7701, Republic of South Africa.

Minor Dissertation: MSc

June 2022

In this work we present the approach taken to address the problems anomalous fault detection and system delays experienced by the MeerLICHT telescope. We make use of the abundantly available console logs, that record all aspects of the telescope's function, to obtain information. The MeerLICHT operational team must devote time to manually inspecting the logs during system downtime to discover faults. This task is laborious, time inefficient given the large size of the logs, and does not suit the time-sensitive nature of many of the surveys the telescope partakes in. We used the novel approach of the Hidden Markov model, to address the problems of fault detection and system delays experienced by the MeerLICHT. We were able to train the model in three separate ways, showing some success at fault detection and none at the addressing the system delays.

Acknowledgements

I would like to offer my sincere and deep gratitude to my supervisor Professor Paul Groot for all his generous support throughout my research project, which in no small way proved crucial in my finishing this minor dissertation. Next, I would like to offer my thanks to the Astronomy Department as a whole. Thanks to emeritus Professor Iain McDonald for his time to answer my question via email correspondence and patience to help someone who asked for advice.

To my parents and brother, thank you for all the love and support. Thank you for having my back. To my friends, who motivated me when I could not motivate myself, thanks for always standing with me. To my colleagues in office 4.33, I will never forget the willingness to listen and understand the what I was dealing with at any point during my research.

Again, I would like to thank the National Research Foundation (NRF) for their financial contributions towards my degree.

Notation

This list summarises the notation that is introduced in the text. Matrices and vectors are in bold, with the matrices being uppercase. Transposition of either the matrices or vectors is denoted with a prime $\prime$. All vectors are row vectors unless other wise stated.

C_t ..... state occupied by the Markov chain at time t

$\mathbf{C}^{(T)}$ ..... $(C_1, C_2, \ldots, C_t)$

$\mathbf{e}_i$ ..... $(0, ..., 1, ..., 0)$, with 1 in the ith position

L_T ..... the likelihood

m ..... number of states for the Markov chain

$p_i(d)$ ..... state duration density

$p_i(x)$ ..... probability mass being in state i

$\mathbf{P}(x)$ ..... diagonal matrix with element $p_i(x)$

$\mathbb{N}$ ..... the set of all positive integers

T ..... the length of the sequence

X_t ..... observation at time t

$\mathbf{X}^{(T)}$ ..... $(X_1, X_2, \ldots, X_t)$

$\boldsymbol{\alpha}_t$ ..... row vector of forward probabilities

$\alpha_t(i)$ ..... forward probability element of $\boldsymbol{\alpha}_t$

$\boldsymbol{\beta}_t$ ..... row vector of backward probabilities

$\beta_t(i)$ ..... backward probability element of $\boldsymbol{\beta}_t$

$\boldsymbol{\delta}$ ..... initial distribution of the Markov chain

$\boldsymbol{\phi}_t$ ..... normalized vector of forward probabilities

$\boldsymbol{\Gamma}$ ..... transition probability matrix of the Markov chain

γ_{ij} ..... probability of transition from state i to state j in the Markov chain

ω_t ..... $\boldsymbol{\alpha}_t \mathbf{1}' = \sum_i \alpha_t(i)$

$\mathbf{1}$ ..... row vector of ones

Abbreviations

SAAO South African Astronomical Observatory

HMM Hidden Markov Model

GW Gravitational Wave

TCS Telescope Control System

HM Hardware Manager

SH Sensor Hub

BH Broker Hub

DH Device Hub

OH Observing Hub

FfH Flatfielding Hub

FITS Flexible Image Transport System

ALMA Atacama Large Millimetre/sub-millimeter Array

Contents

List of Figures

List of Algorithms

List of Tables

Introduction

This is an age where on-site operations that have been traditionally supervised by personnel are increasingly being automated, in favour of reducing operational expenditure and improving productivity (Starovasnik, 2012). The same can be said for the field of astronomical observation. Astronomers share some of the reasons they choose to automate operations with industry such as, fewer operational costs and increased productivity. Other reasons may include a decreased availability to conduct observations (resulting from commitments to academic institutions); the ability to conduct projects that are simply unattainable through manual observation; improvements in the quality of data produced by the telescope (Eaton *et al.*, 2003).

Typically when an astronomer would want to collect data, observations of the night sky (images, spectra, etc.), the only option would be to visit an observing site and operate a telescope themselves. However, this may not be feasible for the aforementioned reasons. This has resulted in an increasing push towards the remote operation of these telescopes, or arrays of telescopes (Bloemen *et al.*, 2016), for the purposes of autonomous observation. One such endeavour is the MeerLICHT optical telescope: located at the Sutherland station of the South African Astronomical Observatory (SAAO). With the increasing adoption of these remote observatories and overall push towards autonomous observatories (Sybilski *et al.*, 2014), the need for autonomous diagnosis of the telescope's system increases in tandem.

In this project we aim to develop a framework for automatically detecting anomalous behaviours in MeerLICHT's function, with the potential to be adopted by future telescopes. This framework utilises the telescope's operational logs (or console logs), in addition to techniques from data mining and machine learning (ML), to process the raw logs thereby extracting the relevant information and to perform analysis respectively. Although we treat this problem in an offline manner, there has been work on utilising console logs to perform online fault detection. Prominent examples make use of a Google production system (Xu, 2010) and demonstrate a security application (Du *et al.*, 2017) respectively.

In order to distinguish between a system anomaly and regular behaviour, we

must first accustom ourselves with the telescope itself: how does it operate, and what the contents of its console logs are.

As such, the subsequent chapter (Chapter 1) will introduce the MeerLICHT telescope. The chapter will discuss some of the novel design characteristics of the telescope, offer a brief background into its scientific goals, and provide a description of the software that operates the telescope itself. Chapter 2 will describe what a typical night of observing entails and how it is represented in the console logs. Log analysis as a whole will be expanded upon, as well as what makes performing it a challenge in general and specifically in this work. Chapter 3 introduces the concept of Hidden Markov Models (HMM), detailing the necessary theoretical groundwork. Chapter 4 discusses the implementation of the HMM framework and other aspects such as data preprocessing, parameter error estimation, and model validation.

The penultimate chapter (Chapter 5) discusses how the research questions were addressed using the HMM framework and explores the relevant results. Finally, Chapter 6 will put forward whether the framework meets its stated aims, or not, discuss noteworthy points accumulated throughout the research procedure, and offers suggestions on how to build on this work into the future.

Chapter 1

The MeerLICHT telescope

Although MeerLICHT itself is not the direct object of study in this work, it is the main benefactor of the progress detailed herein. As such, we attempt to familiarise one with the telescope in this opening chapter.

Section 1.1 begins by presenting the unique design aspects of MeerLICHT. A few of the achievable scientific goals, worked towards by the project's collaborators, are elaborated on in Section 1.2. Presented in Section 1.3 are the levels of automatic function for an observatory; level progression enabling software; the production of console logs by both the telescope and software.

Concluding the chapter with Section 1.4, a discussion is offered on the necessity of automated fault detection.

1.1 Introducing MeerLICHT

The optical telescope MeerLICHT (Bloemen *et al.*, 2016) was conceived by a consortium led by Radboud University (NL),joined by the University of Cape Town (SA); SAAO (SA); Oxford and Manchester University (UK) an the University of Amsterdam, for the purpose of studying transient phenomena, gravitational wave (GW) counterparts, and variable stars. The full telescope may be seen in Figure 1.1.

1.1.1 The design of the telescope

The telescope is purpose-built to meet the needs of studying transient phenomena (astronomical objects that vary in time) with the aim of co-observing with MeerKAT — South Africa's Square Kilometre Array (SKA) precursor radio telescope array. A real-time, simultaneous, optical view of the radio sky is provided by MeerLICHT (Bloemen *et al.*, 2016).

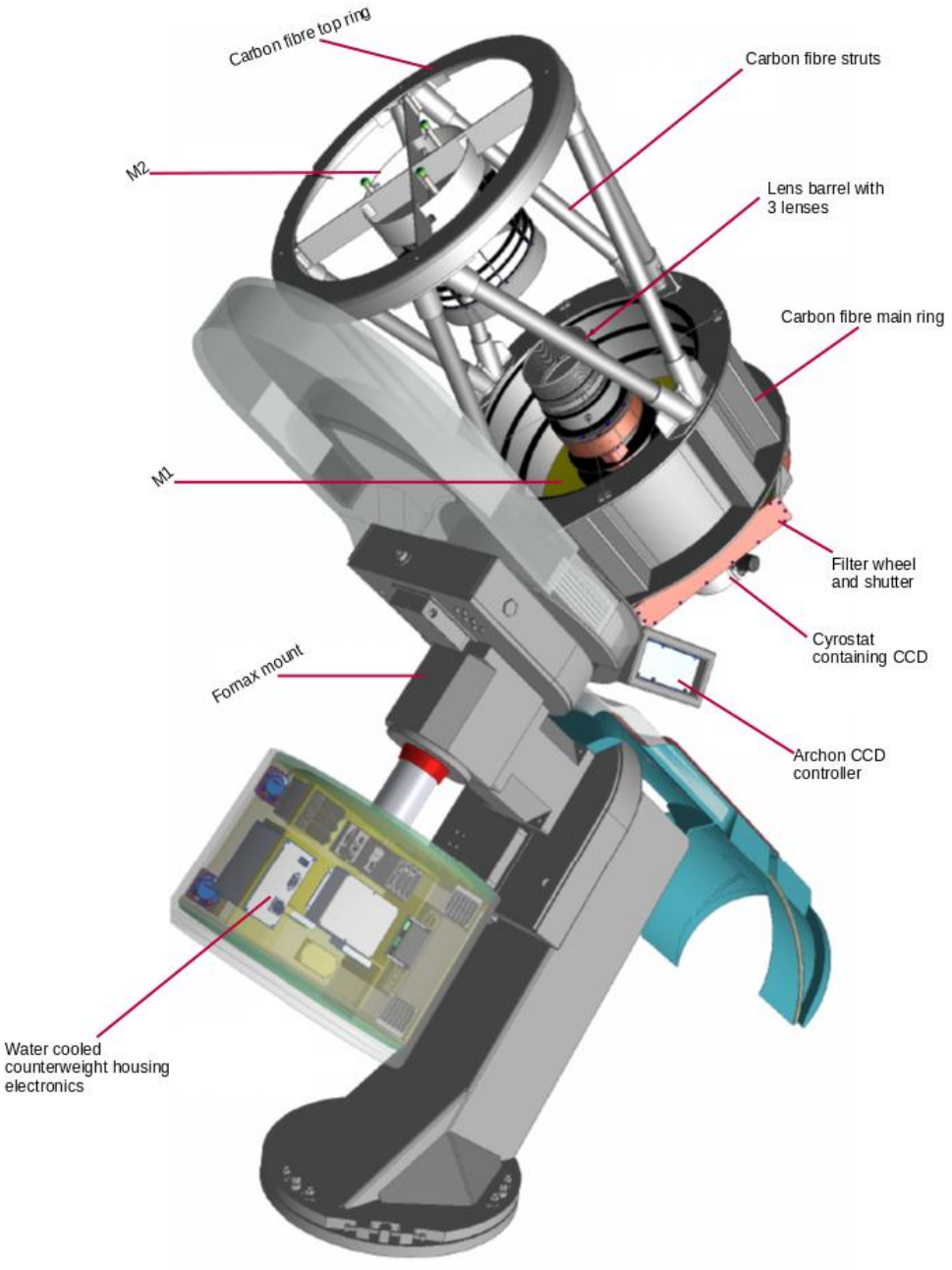

Figure 1.1: Model of the MeerLICHT/BlackGEM telescope. Adapted from Bloemen *et al.* (2016).

Bloemen *et al.* (2016) describe the telescope as being constructed of aluminium and carbon fibre components. For instance, the main ring around the primary mirror (M1), top ring around the secondary mirror (M2), and the struts connecting the two are composed of carbon fibre, thus reducing weight and increasing rigidity. Employing a modified Dall-Kirkham design, with a paraboloidal primary mirror (M1) and a spherical secondary mirror (M2) (both made of fused silica), MeerLICHT is a wide-field optical telescope. Vignetting is reduced by having a primary mirror (65 cm) that is larger than the photometric aperture (60 cm) and a 2.7 square degree field of view (FoV), with a 0.56 arcsec/pixel sample scale, is produced. Several piezo actuators, offering guidance and focusing capabilities, actively control MeerLICHT's secondary mirror.

A flat focal plane, of 9.5 cm by 9.5 cm, is achieved using a triple-lens field corrector. The novel concept of using the third lens as a first order atmospheric dispersion corrector (ADC) is incorporated into the design, this is a bonus, as no extra optical components to achieve this effect are added (ter Horst *et al.*, 2016). This corrects for differential refraction, resulting from interactions with the atmosphere experienced by photons of differing wavelengths traveling to the camera.

Mounted on a specially requested FORNAX 200 equatorial mount (developed by Fornax, in consultation with the BlackGEM team), with the counterweight that houses the electronics able to be cooled to reduce the optical turbulence experienced by MeerLICHT (Bloemen *et al.*, 2016). The function of the equatorial mount is to allow the motion of the telescope in a single direction (in this case the right ascension, RA), while tracking stars. Before the camera a shutter wheel and a filter wheel with 6 positions is located. The filter set having been derived from the Sloan Digital Sky Survey (SDSS), with an additional wide q filter are used by MeerLICHT. The filter wavelength ranges are shown in Table 1.1 below:

Table 1.1: The filter set used, and the range of wavelengths they represent. The same filters will be used by MeerLICHT/BlackGEM (Bloemen *et al.*, 2016).

Filter	Wavelength range (nm)
u	350 - 410
g	410 - 550
r	563 - 690
i	690 - 840
z	840 - 990
q	440 - 720

Lastly, we discuss the heart of the telescope the detector. A charge coupled device (CCD) semiconductor, made of silicon, consisting of a grid of photosensitive

pixels is used as a detector by MeerLICHT. Interacting with the silicon, an electron-hole pair is created (via the photo-electric effect) by an arriving photon. While the hole is diffused into the bulk of the silicon, the electron remains stored in the pixel to be read-out by electrodes as a signal (Pieterse, 2019).

As mentioned above, a 2.7 square degree FoV with a 0.56 arcsec/pixel sample scale is produced by the CCD. The specific CCD used is a single large-format CCD, the STA1600 chip, with an area of 10560 by 10560 of 9 μm by 9 μm pixels. The choice of a single chip, as opposed to building a mosaic of smaller chips, was due to a reduction in complications and a more reasonable cost-per-pixel expense. The chip was acquired together with an Archon controller, allowing the chip to be read-out in 7 s at a frequency of 1 MHz. To aid in the reduction of noise, generated by heat creating electron-hole pairs, the CCD is cooled by building it into a cryostat which is operated at a temperature of 170 K.

1.2 MeerLICHT's science goals

The study of astrophysical transients, falling under the purview of time domain astronomy, is the core scientific objective of the MeerLICHT project (Paterson, 2019). However, there are numerous astrophysical objects that vary in time and not all of them would be called transients. Some of these objects would be identified as variable stars (or just variables)[1]. The key difference between a variable and a transient would be that a transient is a once-off phenomena, whereas a variable is an object that has a sustained or repeated variation over a span of time.

Examples of transient phenomena include: supernovae, gamma-ray bursts, fast radio bursts, and gravitational wave (GW) sources. An object that has a sustained variation of brightness (even though this variation *itself* may vary), for example would include pulsars and dwarf novae. It is important to note that the time-scales of variability (as a whole) may be as short as a few milliseconds or last several decades. To investigate these phenomena thoroughly, a number of programs and surveys will be performed by MeerLICHT. Elaborated as follows are those surveys, and its secondary purpose as a prototype for the BlackGEM array.

1.2.1 Full sky survey

The nature of observing phenomena that are constantly changing, necessitates that comparisons of older observations are made with newer ones in order to observe that change (Pieterse, 2019). As such, the southern sky that is visible (DEC $< 30°$) is

[1] See Figure A.1 for a detailed view.

repeatedly scanned when a catalogue of **reference** images (as they are referred back to for comparison) is built by MeerLICHT.

These images will be taken in all the filters (u, g, r, i, z, and q), utilising the same 60 s exposure time for each image. The method by which the observations will be performed is known as **snap-to-grid**. This involves dividing the sky visible to MeerLICHT into a grid, where each block of that grid (called a **field**) is the same size as MeerLICHT's FoV, then observing any particular field of that grid. This is only possible due the consistent image quality over MeerLICHT's entire FoV, which allows objects at any position in the field to be observed with similar clarity.

1.2.2 Co-observing with MeerKAT

The shadowing of the SKA precursor array MeerKAT, is MeerLICHT's primary goal (Bloemen *et al.*, 2016). MeerKAT consists of 64 antennae, each with a 13.5 m diameter, with a maximum baseline of 8 km; where 70% of the collecting area falls within a kilometer of the core as described by Booth and Jonas (2012). As a result, the MeerKAT array is the most sensitive centimetre-wavelength radio telescope in the world (SARAO, 2016). Currently, multi-wavelength projects are performed by follow-up observations and for the study of certain transients this is insufficient. By the time these follow-ups are triggered the transient may have completely passed, or important information may be lost. MeerKAT has a FoV of 1.75 square degrees (Taylor and Jarvis, 2017), enabling MeerLICHT (FoV 2.7 square degrees) to observe the entire area MeerKAT is capturing in a single pointing. Their close proximity to each other ensures the same area of the night sky will be available to both of them simultaneously, easing scheduling and making co-observing even more practical.

Several transient surveys namely: **ThunderKAT**, studying slow radio transients (several seconds and longer in duration); **Pulsar Timing**, exploring the Galactic Centre and the Galactic Plane conducting precision pulsar timing; **TRAPUM**, aiming to increase pulsar database and investigate unusual pulsating objects, are undertaken by MeerKAT (Booth and Jonas, 2012). Where possible all MeerKAT observations will be co-observed by MeerLICHT. The main limitation being that, as an optical telescope, co-observing will be done at night during clear conditions by MeerLICHT.

1.2.3 Local transient program

Scheduled as a back-up, in the event that co-observing with MeerKAT is not feasible, a local transient program has been arranged for MeerLICHT (Pieterse, 2019).

Here, nearby galaxies (at a distance < 100 Mpc) in the filters u, q, and i in the attempt to find faint and rare transients, but to also better understand the transients in stellar populations, will be observed by MeerLICHT. As the limiting factor for this local transient program is not the sky background, but the surface brightness of the observed galaxy (the galaxy light sets the sensitivity, not the background), a half-hour period has been reserved between dusk and full dark in the evenings, along with the reverse in the mornings.

1.2.4 BlackGEM Prototype

MeerLICHT is serving as the prototype to the BlackGEM array based at the La Silla observatory in Chile (Bloemen *et al.*, 2016). The goal of the BlackGEM project is to find electromagnetic counterparts of GW sources, following triggers from the Advanced LIGO and Advanced VIRGO interferometers.

Identical telescopes are used for both the BlackGEM and MeerLICHT projects, although there will be more for BlackGEM. Due to the poor sky localisation provided by the interferometers, BlackGEM will need to survey areas from 100 to several hundred square degrees. As such, an array of several smaller telescopes was deemed most appropriate, with the first 3 having been built and commissioned at La Silla.

Several surveys will be performed by BlackGEM. **BlackGEM Southern All Sky survey**, to build a good reference image catalogue a full scan of the southern sky (in all filters) and will comprise 50% of the observing time during the first year. **BlackGEM Fast Synoptic Sky survey**, two weeks around each full moon will be used to perform a high-cadence (1 min) synoptic survey of a variety of fields to improve the understanding of the fast transient and variable sky. **BlackGEM GW Trigger follow-up**, the highest priority program for BlackGEM and will have the array image all the visible fields with the highest probability of finding the GW source. **BlackGEM Qscan survey**, BlackGEM will map a > 1000 square degrees area of the southern sky in the $q-$band to locate slowly evolving transients (BlackGEM, 2022*b*). **BlackGEM Local Transient survey**, a survey of 50 square degrees localised on mass over-densities < 100 Mpc, performed in the u, q, & i filters with the aim of monitoring the population of massive luminous binary stars and detecting low luminosity transients (BlackGEM, 2022*a*).

1.3 Operational software

There are three levels of automation and independence, as defined by Sybilski *et al.* (2014), each differentiated by the amount of human intervention or presence that is

required for full functional operation, which is elaborated on as follows:

Typical observatory (TO) - the entirety of the observatory's control system is not exposed to the internet, an observer is required to perform some function to ensure an observation is executed safely.

Remote observatory (RO) - all instruments and functionality are available online. The scheduling, conducting, and supervision of observations is able to be performed remotely. With the majority of observing procedures automated, human presence will only be required for network malfunctions and hardware failures.

Autonomous observatory (AO) - the system is capable of operating with errors (up to some threshold), is able to independently plan action, and to prioritise targets. Redundant sensors and hardware components will keep the system available for use, with human presence needed for quicker hardware and software upgrades.

Conducting observations, ensuring system safety during normal conditions and errors, making independent decisions about planned tasks (using available data and logic), and performing maintenance are the key areas to autonomous observing. Currently, it would be difficult to classify MeerLICHT. It would fall somewhere between a remote observatory and autonomous observatory. The reasons being, that MeerLICHT has only partial hardware redundancy, fault tolerant for software, and only has semi-independent decision making, so it may fall into the category of "semi-autonomous".

1.3.1 The telescope control system

The telescope control system (TCS) that is being used for both MeerLICHT and BlackGEM's remote operation is called ABOT (**A**stronomical Ro**bot**), developed and maintained by the team at Sybilla Technologies. Enabling the remote operation of MeerLICHT, ABOT was developed with the intention of fulfilling the needs of a network of distributed autonomous observatories (Sybilski *et al.*, 2014).

Autonomous observing presents numerous challenges to software engineers: the remoteness of a typical observatory's location; the safety of the equipment; as well as reliability, fault tolerance, and extensibility (Drzał *et al.*, 2016). Their solution to this was to design the system in a distributed way, isolating key functionalities, and by layering the structure of the software, with increasing abstraction, to keep the problem manageable (see Figure 1.2). This isolation allows the different devices (components) of the observatory to function with a level of independence, and ensures the observatory keeps functioning despite an error in the operation of some components.

The first layer (referred to as Level Zero)[2] is that of the actual devices themselves, along with the default firmware and API (Application Programming Interface) that is loaded on to each of them by the manufacturer.

The second layer (Level One), is the Hardware Abstraction Layer (HAL). At this layer, Sybilla provides the software drivers needed for an astronomer to interact with the devices through a Graphical User Interface (GUI) for single devices (e.g. the dome). This provides a unified way of accessing all the devices, and in the instance that the manufacturer included tools are inadequate or buggy an alternative is provided to ensure this unified experience.

The third layer (Level Two) consists of the various robotic services needed for autonomous observation, implemented in the Microsoft Robotics Framework. At this level the individual devices are able to interact with each other and are capable of providing aggregated functionality called a *service*. A service that requires other services to perform tasks, and function properly, is called a *hub*. The observatory is also capable of connecting to the internet at this level enabling astronomers to schedule observations for that night, or even take remote control of the observatory.

The fourth layer (Level Three) is the cloud services, hosted on an Azure cloud. A common point of safe and secure interaction with the robotic observatory; referred to as ABOT Tutor, developed using Aurelia (JavaScript client framework for web and mobile)[3].

1.3.2 The hubs and logging with ABOT

When a service requires another service to perform its function, it becomes a hub. There are several hubs, and some provide the basis for the domain isolation of individual components. Components do not have direct communication with those outside their domain, this isolation is achieved using separate processes and separate memory (Drzał *et al.*, 2016).

There are three domains, with the first isolated domain being the *Hardware Manager* (HM). Its domain only contains other hubs, with a number of those hubs interacting with components in the remaining domains. This domain contains the means for higher layers of software to access the intermediate layer via the Proxy Hub (or Broker Hub) (BH) and the Scheduler. The Proxy Hub is used for on-demand interaction with the observatory, and the Scheduler handles all scheduled observations. The Observing Hub, Focusing Hub, and Flatfielding Hub connect to components directly, partitioning the work the HM hub needs to perform.

[2] See Figure A.2 for conceptual overview.
[3] Aurelia Homepage: http://aurelia.io

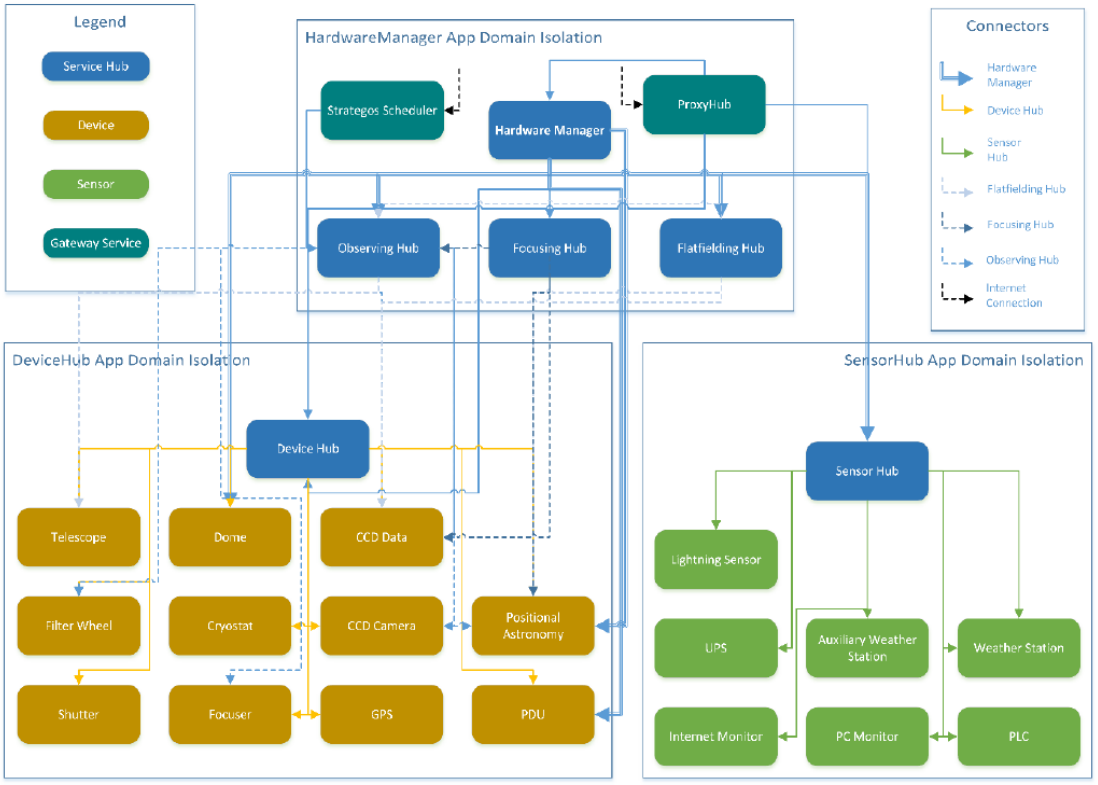

Figure 1.2: Current software architecture of ABOT for MeerLICHT/BlackGEM, showing the domain isolation, and grouping of components by function Drzał *et al.* (2016).

The remaining two domains are the *Device Hub* (DH) and the *Sensor Hub* (SH). The DH domain isolation contains components such as: the telescope, dome, CCD, and shutter etc. While the SH domain isolation contains all the sensor components: the lightning sensor, and weather station etc. The HM hub is able to connect to several components within the DH domain, this enables the independent functioning of components during an observation. For instance, the HM hub can move the dome itself, while the DH hub moves the telescope. It does not need to communicate through the DH hub. This independence speeds up observing time, and in the case the HM hub cannot communicate with the dome (e.g. network malfunction) the DH hub can do it (fault tolerance).

An aspect of making the software more manageable, via the use of isolation, is that it makes debugging a simpler task. Each component leaves a record of a task executed, what it did, whether the execution failed or succeeded; this record of single tasks is then aggregated over a period of time to a single sink. A sink could be a file (flat file, or a rolling flat file), a cloud service, or a database. This process is known as *logging*, and if the sink is a file it is referred to as a log file (console log). Proper logging is crucial to aiding debugging, performing adequate maintenance of a system, and to making improvements to it (Drzał *et al.*, 2016).

Sybilla built their own logging library (based on Microsoft's Enterprise Library) following the principles: each component uses the same logging library; different sinks must be available; components that are used together (i.e. within a single domain isolation) are aggregated to the same sink; multiple sinks may be active simultaneously for a single component (e.g. file and database); different levels of verbosity will be available; sinks/verbosity are provided in a configuration file and may be manually or dynamically changed when necessary.

As this work was performed in an offline setting, the sinks used for the log information were flat files, formatted as a plain text files, and unsurprisingly there are four separate sinks for each the HM, DH, BH, and SH. The contents of each sink are discussed in the next section.

1.4 Necessity of automated fault detection

Due to the time sensitive nature of the goals that MeerLICHT aims to achieve, as the timescales that transients occur over are several seconds and longer (SARAO, 2016), and that co-observing with MeerKAT may only happen during the night — the ability to quickly diagnose faults in MeerLICHT's operation is crucial.

As it stands, when an error occurs that causes the telescope to cancel the current observation an error message is displayed in the web-interface of the telescope control system, but nothing further to help diagnose the problem. This may in turn lead to further problems, such as the telescope not returning to the correct starting position before it moves onto the next observation. Or if a physical component (device) of the telescope is operating slower than is expected an astronomer has to go and directly assess that particular service's log entry to find the issue. This process is time inefficient, and compromises the ability of MeerLICHT to achieve its goals.

For this reason we explore the console logs. To analyse the information they provide on the workings of the telescope's system, so an automated tool may be developed to diagnose system faults and bottlenecks in the task duration.

Chapter 2

Logs, log analysis, and log mining

Console logs, and logging, were briefly touched upon in the previous chapter. However, only in so far as describing the mechanism that generates the logs and how it relates to the general architecture of the software. The content of the logs will now be discussed, along with the topic of analysing those logs.

Chapter 1 aimed to make the telescope more familiar. The goal of this chapter is to make the contents of the logs, and what can be done with that raw data, familiar. Section 2.1 discusses MeerLICHT's logs detailing: a typical night of observing as represented in the logs, the content of the logs, and their structure. Section 2.2 provides an introduction to log mining, what it is typically used for, illustrating methods of its implementation, including contemporary tools, while also discussing current research utilising console logs. Section 2.3 specifically looks at projects within astronomy utilising console logs. Section 2.4 tackles the challenges when working with logs.

2.1 MeerLICHT's operational logs

As mentioned earlier, MeerLICHT's TCS (ABOT) provides an extensive logging feature. This function centres around the service hubs described in Section 1.3, the most notable having their own domains of isolation: the HM, DH, BH, and SH. These hubs in particular provide the best overall picture of the state of the observatory, a consequence of them being connected to all services the observatory has.

Together they facilitate the observatories remote connection (BH); interact with all the observatories environmental and other sensors (SH); monitor the connections between the various sub-components that make up the hardware services (DH); manage all the hardware services by checking if they are functioning within bounds (HM).

2.1.1 Contents of MeerLICHT logs

Each domain isolation has its own flat file log — formatted as a plain text file. These files contain the, human-readable, output of the various services that fall within that particular domain isolation. Although, since some of the services are hubs too and can request an action performed by a service outside its domain (see Figure 1.2), some actions are logged more than once.

There is some structure to the text files, however, that structure differs slightly for each domain. This makes them more manageable to work with, as opposed to being completely free text (the exception being the BH). Examples of that shared structure would be: whenever a new action is logged, as opposed to the output of a single action across multiple lines, a timestamp[1] is given; then the verbosity of the output; the service/hub performing the action; and lastly the output of the action performed.

What follows is an investigation of some the outputs that would be recorded in each log, with the intention of providing some more detail about their contents.

Hardware Manager

```
2019-11-30T18:39:05.096Z INFO Abot.Hubs.HardwareManager2.Y2013.M06: Calling Dome.Rotate with 8.85173999841
2019-11-30T18:39:05.098Z VERBOSE Abot.Hubs.HardwareManager2.Y2013.M06: Pending commands
2019-11-30T18:39:05.098Z VERBOSE Abot.Hubs.HardwareManager2.Y2013.M06: HardwareManager2::Dome.Rotate: { nam
2019-11-30T18:39:10.490Z INFO ObservingHubService[Base]: Filterwheel Move (1e00dca1-88ff-49e6-a2dc-8494a492
ObservingHub->ObservingHub
2019-11-30T18:38:46.0367430+00:00 600.00 s, 575.55 s
2019-11-30T18:39:10.490Z INFO ObservingHubService[Base]: Filterwheel Move (1e00dca1-88ff-49e6-a2dc-8494a492
ObservingHub->ObservingHub
2019-11-30T18:38:46.0367430+00:00 600.00 s, 575.55 s
2019-11-30T18:39:10.492Z INFO ObservingHubService[Base]: Camera Configuration Check (913e00db-740a-4f6f-8ff
ObservingHub->ObservingHub
2019-11-30T18:38:46.0377448+00:00 600.00 s, 575.54 s
2019-11-30T18:39:10.493Z INFO Abot.Hubs.ObservingHub.Y2013.M07: Observing Hub PostProcessCommand: Camera Co
2019-11-30T18:39:10.494Z VERBOSE Abot.Hubs.ObservingHub.Y2013.M07: Checking for subframe
```

Figure 2.1: An excerpt from the HM domain, showing the actions of the HM and OH.

Since the HM sits at the top of the domain isolation hierarchy, reviewing its contents is able to provide a rather detailed account of what the observatory as a whole does throughout the day. This detailed account includes keeping track of, and enforcing, the daytime transitions, loading of the observing schedule, or checking for messages from the web interface.

When it comes to observations, besides processing the observing programs for an evening, the HM controls the dome and gives the go ahead to begin observing. Looking at Figure 2.1, the Observing hub (OH) is responsible for coordinating most

[1]Formatted as: yyyy-MM-dd'T'HH:mm:ss.SSS'Z' e.g. 2019-12-02T02:33:07.033Z

of the actions taken related to observing like: issuing observation requests, checking the camera, moving the telescope to the correct set of coordinates, taking an image, saving the image, or updating metadata. Since the OH is a hub, it does not perform all these actions itself, but rather it puts the requests to the relevant services it is connected to. For instance it issues the move telescope request to the telescope service, which then performs the action.

Similarly the Flatfielding Hub (FfH), which coordinates the taking of images used for making corrections to science images, can move the telescope, capture an image, and process that image for saving, because it is connected to the OH which enables it use services it is not directly connected to.

```
2019-11-30T18:15:16.841Z VERBOSE Abot.Hubs.HardwareManager2.Y2013.M06: <ObservingProgram>
  <signature>http://sybillatechnologies.com/observingprogram</signature>
  <programName>BGLT</programName>
  <programId>bglt</programId>
  <validFrom>2019-11-30T23:15:40.00Z</validFrom>
  <validTo>2019-12-01T02:32:16.00Z</validTo>
  <timeBasedConstraint>false</timeBasedConstraint>
  <observationFields>
    <observationField>
      <observationFiledName>16026</observationFiledName>
      <ra>6.752208666666666</ra>
      <dec>-26.88394</dec>
      <observationAtoms>
        <observationAtom>
          <observationAtomName>u_60</observationAtomName>
          <filter>u</filter>
```

Figure 2.2: The image shows a portion of the observing program for that evening.

The HM log does not just contain the recorded output from requests of services (and hubs) within the HM domain, there is also a copy of the full observing program, in XML format, that has been scheduled for the observatory to perform that night (Figure 2.2). The program has information on: the fields to be observed, the filters the observations are to be performed in, the exposure time, the RA and DEC coordinates, how long a scheduled observation is valid for etc.

Device Hub

Moving onto the DH, which contains a wealth of information, since its domain contains almost all the observatory services — except the environmental sensors. There are several reoccurring (unrelated to taking an exposure) actions that are present in this log. The first of which is the connection made with the mount to get its properties (e.g. position, is it parked, is it slewing, etc.), as seen in Figure 2.3, this happens every second with the properties displayed in XML format as well.

```
2019-12-03T02:40:15.992Z VERBOSE Fornax.WSCommunicator: Received: <?xml version='1.0' encoding='
<mountProperties xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aos="http://sybilla
  <name>Sutherland</name>
  <rightAscension>9.35787883251</rightAscension>
  <declination>-26.0547298472</declination>
  <altitude>79.9994771275</altitude>
  <azimuth>44.970009609</azimuth>
  <epoch>J2000</epoch>
  <targetRightAscension xsi:nil="true"/>
  <targetDeclination xsi:nil="true"/>
  <targetAltitude xsi:nil="true"/>
  <targetAzimuth xsi:nil="true"/>
  <isConnected>true</isConnected>
  <isHomed>false</isHomed>
  <isParked>false</isParked>
  <isSlewing>false</isSlewing>
```

Figure 2.3: A portion of the mount properties that are returned every second.

Next is the dome where every 5 seconds, a communications check is performed (Figure 2.4a) and its watchdog is reset (Figure 2.4b). The watchdog is a fail-safe that several of the devices have (dome, mount, focuser, etc.). If the connection between a device, and the program supervising it fails the watchdog is not reset properly indicating a failure (Drzał *et al.*, 2016). Figure 2.5 shows that there is another communications check, with the dome's controller, that provides a dome update regarding the weather sensor, dome shutters, dome power, etc. This check appears to occur every 10 seconds.

```
2019-12-03T02:40:15.431Z VERBOSE MeerLichtDome.Communicator: COM1 sent: :01061065800004\0D\0A
2019-12-03T02:40:15.485Z VERBOSE MeerLichtDome.Communicator: COM1 received: :01061065800004\0D\0A
```

a

```
2019-12-03T02:40:15.020Z VERBOSE MeerLichtDome.Abot: Resetting watchdog
2019-12-03T02:40:15.020Z VERBOSE MeerLichtDome.Controller: Resetting wachdog
2019-12-03T02:40:15.020Z VERBOSE MeerLichtDome.Abot: Watchdog reset
```

b

Figure 2.4: Excerpts from the Device Hub log: (a) the communications checks that are performed with the dome, (b) how the dome's watchdog is reset.

Along with those recurring actions, there are also those actions that follow from observing. Because of the way the software architecture is structured, with some services in the HM domain able to communicate with services in the DH domain, a request will start within the HM and will be responded to from within the DH. The build-up to and after, taking an exposure is where most of the observatory's functionality is used. A consequence of the nature of this work, is the importance of having a clear sequence of actions to analyse — made more difficult when that full sequence is documented in separate locations.

```
2019-12-03T02:40:24.770Z VERBOSE MeerLichtDome.Controller: Start of UpdateAsync
2019-12-03T02:40:24.770Z VERBOSE MeerLichtDome.Controller: Starting update
```

Figure 2.5: Updates from sensors inside the dome regarding its position, power, etc.

What assists in tracking how requests are fulfilled is that they have an unique ID. Figure 2.6a gives the most clear example of an ID, shown in the first line of the screenshotted text. The 'Acquire Data' request is given an ID[2], however this ID will only be used with the HM domain (log). The actual image will have a name (given based on the observing program, filter, and field) and a unique ID, assigned by the OH. However, depending on what type of image is taken either the OH will issue it an ID or the camera service which is within the DH domain (Figure 2.6b).

```
2019-12-03T02:40:50.325Z INFO ObservingHubService[Base]: Acquire Data (868c9b02-fbda-4f03-a30e-d5bdfc86ef6e) Pending.
ObservingHub->ObservingHub
2019-12-03T02:40:50.2793989+00:00 600.00 s, 599.95 s
2019-12-03T02:40:50.325Z INFO Abot.Hubs.ObservingHub.Y2013.M07: Observing Hub PostProcessCommand: Acquire Data. (INFO: 008OBH)
2019-12-03T02:40:50.326Z VERBOSE Abot.Hubs.ObservingHub.Y2013.M07: Acquire data OH: expTime: 60 ccdReadout: 90 (VERBOSE: 879OH)
```

a

```
2019-12-03T02:40:50.329Z INFO Abot.Services.Camera.Y2012.M09: OnBeginAcquireData. 29731e3a-0ec3-4243-bea5-070a73e485fb Pending
```

b

Figure 2.6: The completion of a request is fulfilled across separate isolated domains: (a) a request for an exposure to be taken (Hardware Manager), (b) the same request as the camera service acts to fulfill it (Device Hub).

This makes it difficult to track the images in the DH, since it assigns an exposure a unique ID that is not shown in the HM log until the image is ready to saved. Similarly, within the DH log, the name assigned by the OH is only connected to the camera service issued ID once the image ready to be saved.

Broker Hub

The BH is a specialised gateway that maintains the observatory's secure connection to the hosted cloud services Drzał *et al.* (2016) (see Figure 2.7). This is a separate communication channel to the one used for scheduling (using Azure queue) observing programs, and is used to remotely access the observatory or receive status updates in real-time. As a gateway service the BH logs do not contain much information outside the time it takes for services to send status updates.

Sensor Hub

Lastly is the SH, to which all of the environmental sensors are connected (see Figure 2.8). There are several sensors within the dome, and on the exterior of the

[2]The ID shown in the figure is: 868c9b02-fbda-4f03-a30e-d5bdfc86ef6e.

```
2019-12-02T00:01:27.125Z VERBOSE Abot.Hubs.BrokerHub.Y2014.M10: Sent HardwareManagerState for hardwaremanager01 in 177.9998ms.
2019-12-02T00:01:28.907Z VERBOSE Abot.Hubs.BrokerHub.Y2014.M10: Sent FocusingHubState for focusinghub01 in 177.9938ms.
2019-12-02T00:01:28.908Z VERBOSE Abot.Hubs.BrokerHub.Y2014.M10: Sent SensorGroupState for observatorywatch01 in 178.9954ms.
2019-12-02T00:01:28.908Z VERBOSE Abot.Hubs.BrokerHub.Y2014.M10: Sent FocuserState for focuser01 in 178.9954ms.
2019-12-02T00:01:28.909Z VERBOSE Abot.Hubs.BrokerHub.Y2014.M10: Sent TelescopeState for telescope01 in 179.9964ms.
2019-12-02T00:01:28.910Z VERBOSE Abot.Hubs.BrokerHub.Y2014.M10: Sent ObservingHubState for photometrichub01 in 180.9944ms.
2019-12-02T00:01:28.910Z VERBOSE Abot.Hubs.BrokerHub.Y2014.M10: Sent PduManagerState for pdumanager01 in 180.9944ms.
```

Figure 2.7: Various services are continuously sending updates for the devices they manage via the secure connection managed by the Broker hub.

observatory, to monitor if the weather (rain, etc.) conditions fall within the pre-defined operational limits. These sensors are continually sending updates to the observatory's web interface in real-time to keep users up-to-date, and provide the environmental information for an image's metadata.

```
Update for .WxtWindSpeedMax.value channel rejected, current 7.7@2019-12-04T12:55:13.5320000+00:00, incoming 8.2@2019-12-04T12:5
Update for .WxtHeatingTemp channel rejected, current 21.5@2019-12-04T12:55:13.5500000+00:00, incoming 21.6@2019-12-04T12:55:13.
Update for .WxtWindSpeedMin.value channel rejected, current 4@2019-12-04T12:55:13.6050000+00:00, incoming 6@2019-12-04T12:55:13
Update for .WxtWindSpeedAvg.value channel rejected, current 6.9@2019-12-04T12:55:13.6360000+00:00, incoming 7.2@2019-12-04T12:5
Update for .WxtWindDirMin.value channel rejected, current 104@2019-12-04T12:55:13.7920000+00:00, incoming 111@2019-12-04T12:55:
```

Figure 2.8: Updating the various wind-speed related values such as, the max-speed, average speed, direction etc.

2.1.2 The typical sequence of events while observing

The actions of the hubs are logged on a regular basis, some more than others, and one can use this information to gain an understanding of how the telescope functions and to help identify the different characteristics of its operation. The record the logs keep is able to be interpreted as a sequence of events that has a particular direction or flow to it. During periods when the performance of the telescope is normal, this flow will be different to when the performance is abnormal. That is the crucial insight needed for performing anomaly detection, being able to differentiate between the normal and abnormal flow of events.

This section presents the typical flow of events during a single day and night of normal telescope operation, and provides context as to whether they would be normal or abnormal. For clarity, the process for taking an exposure was given a separate diagram.

Figure 2.9 shows this process the telescope follows in-order to capture single exposure. There are several different instances when MeerLICHT takes exposures, the first is a science image when the telescope is observing a particular field that contains for example a transient. The next instance is when the telescope takes the **flatfields**. A flatfield image is taken of a uniformly lit surface, and is used to correct for errors introduced via differences in response to photons of the CCD's pixels or for imperfect optics (Pieterse, 2019). The final instance is when the telescope captures

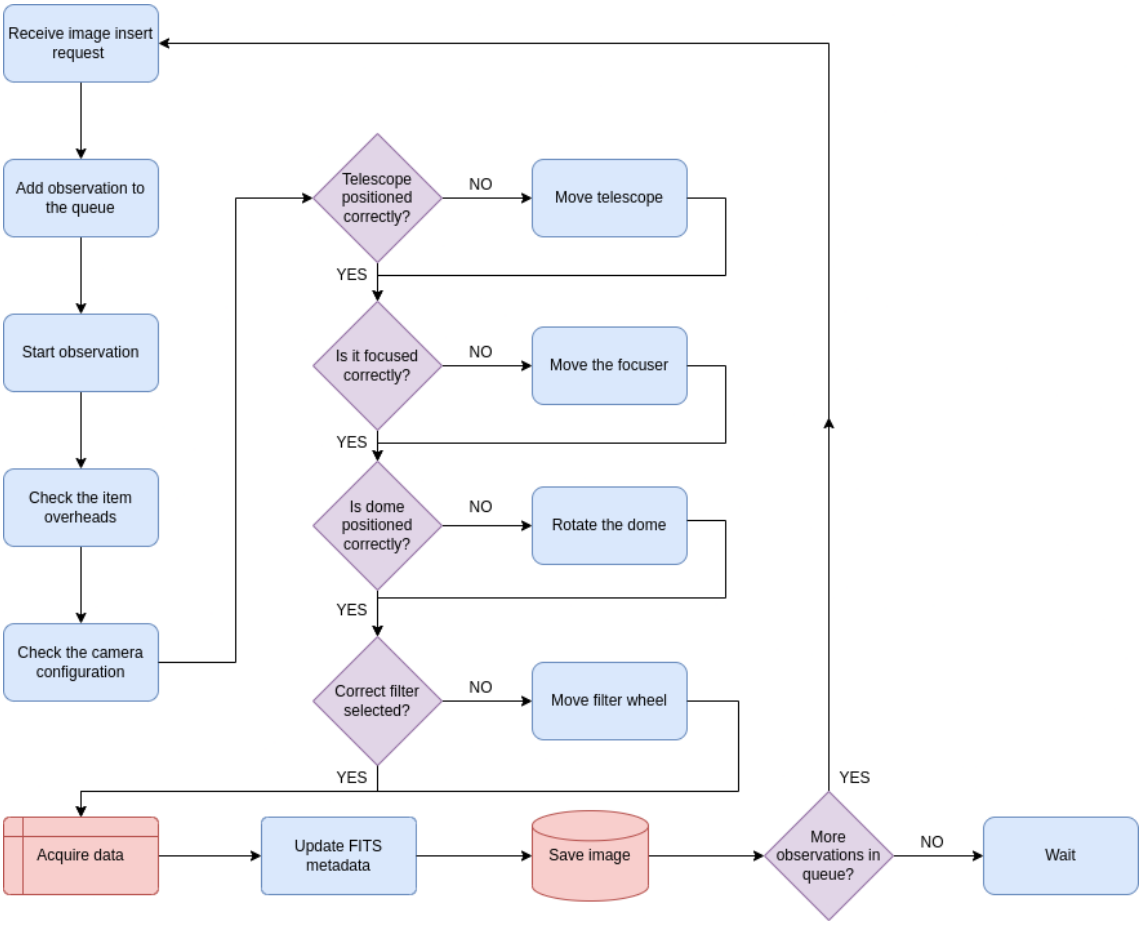

Figure 2.9: The lead up to capturing an exposure, the various calibration checks, and how the image is then stored.

the **bias** and **dark** images (or frames). These are also used to correct the science images, for different types of noise introduced into the image. The bias frames are used to correct for noise individual pixels introduce, they are taken with the shutter closed and zero integration time (length of time shutter is open). While the dark frames correct for any thermal noise generated, due to heat of the components, and have a longer 60 s integration time.

Each image is stored as a **FITS** (**F**lexible **I**mage **T**ransport **S**ystem) file, a format designed for storing large images. Images are stored along with their metadata, in the form of an human-readable ASCII header. The headers contain environmental information (temperature, humidity, etc.) and observational information (the field, the filter, etc.), and are updated just before the image is saved in an offsite database.

Day till Dusk

This is the period of time during which the telescope is least active. Due to the presence of abnormal events the previous night, a backlog of events could occur

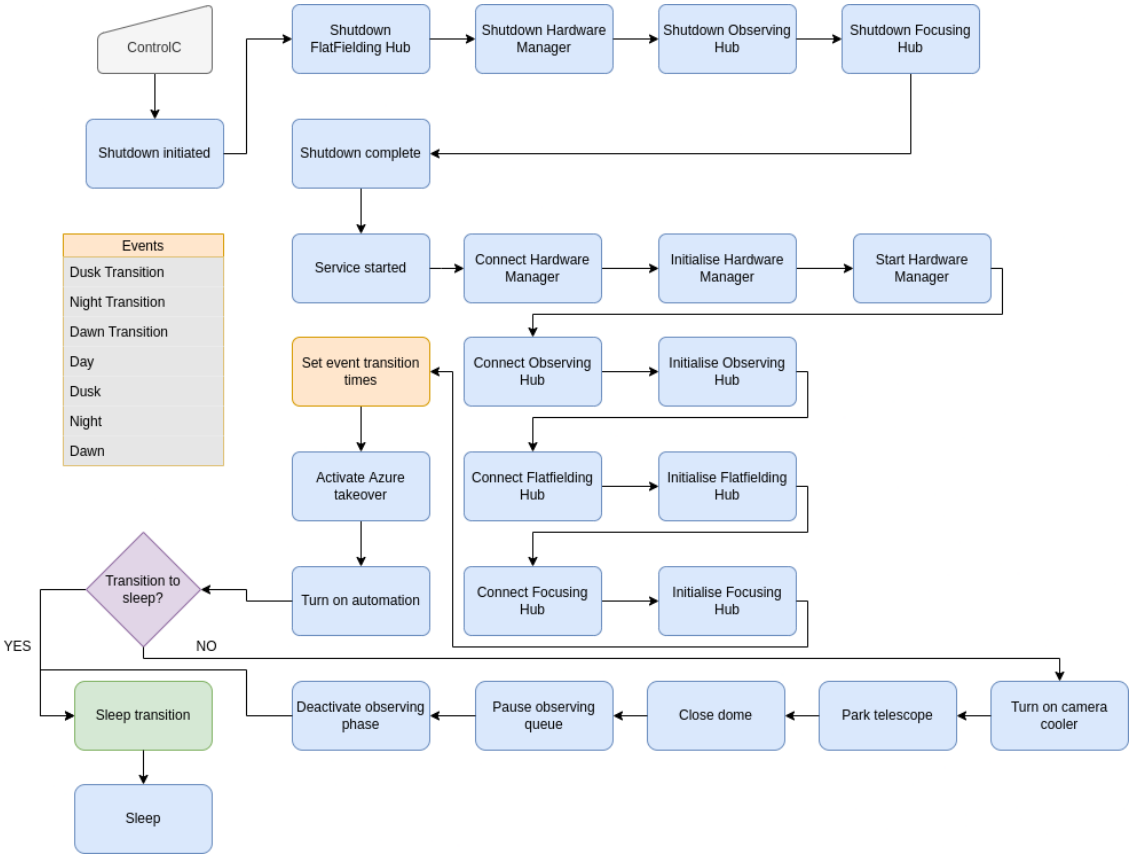

Figure 2.10: The restart of the telescope in the morning, and how it transitions into the sleeping state.

within the TCS. To prevent this from delaying the scheduled observations for the following day the TCS was manually rebooted, depicted as the grey block at the start of Figure 2.10. In the context of this work, that was considered an abnormal event and anything that triggered a reboot was an anomaly. It was found, that this was most often caused by a failure to transition into the appropriate state.

An abnormal event that would occur during this reboot period, and force a reboot, was a failure of the Azure takeover. This would result in a the HM not being able to receive the observing program correctly. Once MeerLICHT had rebooted, the telescope then transitions into its **Sleeping** state. This state allows the telescope to remain on, able to receive instructions and send device updates, yet idle. Acting as the catch-all state, if a malfunction occurs and cannot be rectified by trying the operation again, MeerLICHT looks to transition into the Sleeping state while waiting for human intervention (Drzał *et al.*, 2016).

Dusk till Night

After sitting idle for most of the daytime, the Dusk transition occurs (see Figure 2.11). During the dusk period of the day the telescope takes the first set of flatfields

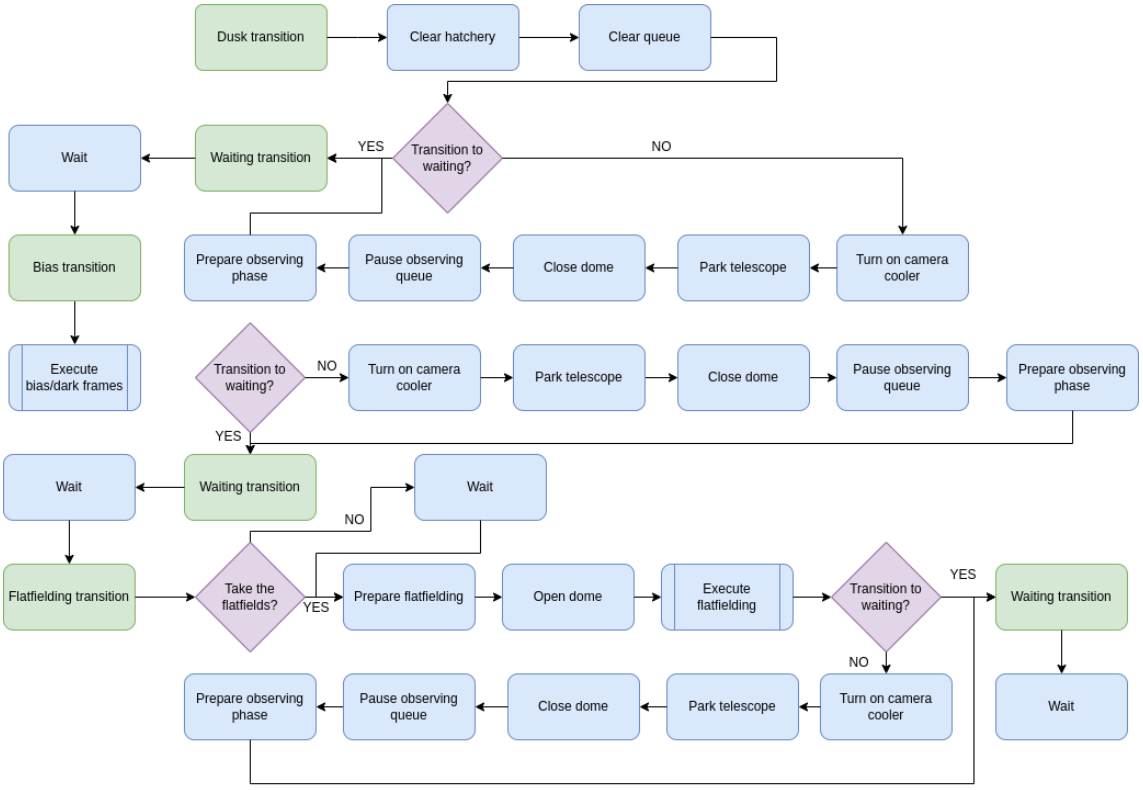

Figure 2.11: The telescope activating for the evening observations, and taking the evening flatfields.

and bias/dark frames. If these actions are not performed or performed incorrectly, it is an indication that the telescope is malfunctioning.

Something that occurred often in the analysed logs, was the cancellation of the **Flatfielding** state due to a delay in taking the bias/dark frames. After waking up and performing the Dusk transition, the telescope transitions into the **Waiting** state. This is the state the telescope assumes between different phases of operation, like taking the bias/dark frames or observing. The difference between this state and the Sleeping state, is that the Waiting state is primarily used during nighttime hours. While the Sleeping state is used during daytime hours. The Waiting state also plays a role in the malfunction recovery process, if failure occurs the telescope will first look to transition back into the Waiting state, before trying the failed action again. If the malfunction persists, the telescope then moves into the Sleeping state (Drzał *et al.*, 2016).

Night till Dawn

The Night transition signals the time for the HM to receive the observing program for the evening. MeerLICHT transitions into the **Observing** state to perform the science exposures.

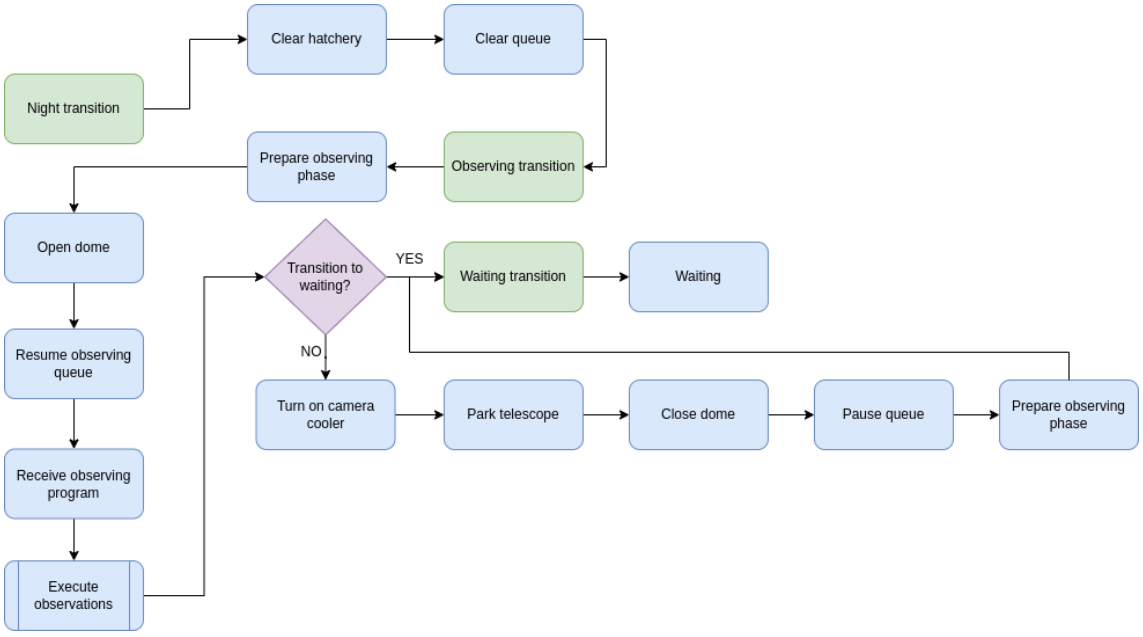

Figure 2.12: The period during which the science exposures are taken.

While Figure 2.12 appears simpler by comparison, it is this phase of operation that can cause the most issues. For example, as the telescope is slewing most often during this phase what could happen is the telescope slews too far unintentionally and stops in the incorrect position. This would have a knock-on effect, as the next observation is expecting the telescope to be positioned differently. When the telescope slews again it will still stop in the incorrect position. The observing program must then be cancelled, and the telescope returned to its home (neutral) position.

Another example involved the dome, when it had closed due to adverse weather and the primary weather station would give the go ahead to return to observing. However, the emergency backup weather station did not give the go ahead. This results in the dome staying closed, and MeerLICHT requiring a reboot. These malfunctions would then affect the taking of the dawn flatfields and bias/dark frames.

Dawn till Day

The usage of this period of time is the same as for the dusk period. The reason for this second flatfielding process, is that the observing conditions closer to dawn are not identical to those closer to dusk (see Figure 2.13). While the dusk flatfields could be used solely, they offer the benefit of redundancy. The dawn and dusk flatfield and bias/dark images may then be stacked and the averages of the pixels are used to create a "master" image. This also reduces any noise that may have been introduced in either set of images (Pieterse, 2019).

Figure 2.13: The second round of flatfields and bias/dark frames are taken in the morning after observing.

The unfortunate thing about this period of operation, is that if there were any malfunctions during the observing phase they can persist to this phase. An example from the analysed logs was that, if there were sufficient failed attempts to transition into the Observing state, this exceeds MeerLICHT's predefined limit for failed transitions and it would not then be able to transition into the **Bias** state nor subsequent Flatfielding state. This would then require a reboot to fix the issue.

2.2 Working with logs to obtain insight

Console logs are a ubiquitous feature of programming that date back to the earliest days (Xu, 2010). According to Oliner *et al.* (2012), many developers still make use of well placed print statements, to log to console or the local disk. Making use of a combination of manual inspection, and regular expressions (regex), to find specific messages or patterns. This practice is not suited to environments that produce an excessive volume of logs, such as large scale distributed systems commonly found today, and thus necessitates automated tools to be built in order to handle the volume of information (Fu *et al.*, 2009).

Within the literature, it becomes apparent that this process of analysing console logs, to find patterns or to make decisions, gets referred to as either log *analysis* or log *mining*.

Log analysis, manually or otherwise (via use of a tool), refers to transforming log information (raw or cleansed) into actionable insights for solving various problems (Alspaugh *et al.*, 2014). This transformation could be an aggregation, or a filtering of certain events. It could even be a visualisation, or a statistical test performed to validate a hypothesis made. Log mining employs techniques from data mining, and/or text mining, to cleanse, structure (e.g. create a table of interesting values), and finally model the log data to uncover, and explore, interesting hidden patterns (Chuvakin, 2007).

2.2.1 Learning from commercial tools

Logs are most often used for troubleshooting. When something goes wrong they are able to provide a wealth of information that can aid a developer in patching the bug that caused the issues (Jayathilake, 2012).

There are a number of commercial analytic tools available, such as Splunk (Splunk, 2020) and Elastic (ELK) stack (Elastic, 2020). They provide solutions to log management, monitoring, online dashboards to access the information, en-

abling visualisations of specific metrics, as well as providing tools with which to model and gain insight from the information.

A metric is a quantity, such as CPU or RAM usage, that a developer might want to monitor during a systems operation, along with the logs, network usage (traffic over TCP/UDP ports), and web applications (HTTP/HTTPS) among others. The management tool ingests these recorded events, indexes individual events for later use, and sends the indexed events to a store; this store may be local, or cloud-based, depending on the package purchased. Once stored, the data is then mined, with interesting discoveries then able to be visualised. An interesting scientific application came from Bagnasco *et al.* (2015), where they were able to utilise the ELK stack to setup a monitoring framework for the INFN-Torino computing centre. The centre which provides Infrastructure-as-a-Service (IaaS), and Platform-as-a-Service (PaaS), services to scientific projects notably the ALICE Virtual Analysis Facility (VAF).

What becomes clear is that there are a couple of important aspects to working with logs — mining (analysis) and ingestion.

Parsing

Sticking strictly to logs, the ingestion, likely the most important part in the entire process, is known as *parsing*. This takes the unstructured logs, and formats them, giving them a structure that makes mining the logs possible.

Nagappan and Vouk (2010) refer to this as 'abstracting the log lines to log events' and as mentioned earlier developers used regex traditionally, but also in cases when an automated solution is not used, to perform this abstraction. It is a non-trivial task to manually create, and maintain, these regex. Especially as the system constantly evolves and older parsing rules become obsolete (He *et al.*, 2016), which they state has resulted in a high demand for automated log parsing systems. This task of automating log parsing is also greatly hindered by the fact that logs are unstructured and that there is no standard for developers to follow when it comes to logging, choosing to log what is useful to them (Jayathilake, 2012).

Even commercial solutions like Splunk (Splunk, 2020), and ELK (Elastic, 2020) (specifically Logstash), require users to provide their own rules in order to parse logs. Logstash does provide some patterns out of the box and a filter plugin called 'Grok' which provides users with a syntax, built on top the Oniguruma[3] regex library, with which to match patterns that are not native.

He *et al.* (2016) evaluated several 'widely-employed' log parsers including SLCT (Simple Logfile Clustering Tool) (Vaarandi, 2003) and logSig (Tang *et al.*, 2011).

[3]Oniguruma: https://github.com/kkos/oniguruma/blob/master/doc/RE

SLCT makes use of a clustering algorithm that was developed by Vaarandi (2003), which was designed to exploit two observations they had, the majority of words appear only a few times and there is a strong correlation between words that appeared frequently. It is a three-step algorithm, similar to the CACTUS algorithm (Ganti *et al.*, 1999). It makes an initial pass to build a data summary; then builds the clusters by assigning sequential lines to candidate clusters based on the similarity of frequent words; finally the candidates are inspected to see if they meet the criteria to be labelled a cluster.

LogSig (2011), attempts to identify a log event type by matching it with a given signature (based off the calculated match score). However, as the signatures are not always known, Tang *et al.* (2011) attempt to find the signatures and the events that match them simultaneously. First the log lines are partitioned into pairwise groups; next these pairs are grouped with as many common pairs as possible; lastly each group is scanned and a signature is constructed, such that it has a high match score with all the pairs in the group. These found signatures can then be used to parse more logs from the same system in future.

Though automated log parsers exist, only SLCT is ready-to-use. This coupled with the need to create their own rules to use the commercial solutions mentioned, and lack of logging standards, motivate developers to implement custom solutions (He *et al.*, 2016).

Mining

According to Jayathilake (2012), some features that a log analysis system should have are the ability to handle log files of varying structure and syntax, while recognising common constructs in the log files (e.g. timestamps, IP addresses, port numbers etc.); next it should be adaptable enough to detect, then bypass, randomly appearing file corruptions without terminating the analysis procedure prematurely; it is highly important that the system is able to automate recurring pattern analysis for an analyst.

Building on their previous work (Jiang *et al.*, 2006), where they used a flow dynamic approach to model transactional data within a distributed system, Guo *et al.* (2006) then use Gaussian Mixture Models (GMMs) to perform fault detection on that transactional data.

Xu *et al.* (2009) realised when working with a highly distributed system, that when log messages were grouped properly they had a strong correlation and that a group of messages was a better indicator of a problem than a single message. They used the novel approach noting that a single log message contains a constant part,

called a 'message type', and a variable part 'message variable'. They then created various features using the message types and variables, using them to perform a principle-component analysis (PCA) based anomaly detection.

Applications of log mining have been found in network failure detection, as well as in diagnosing network-wide traffic anomalies. By introducing a methodology for dynamically mining syslog event sequences Yamanishi and Maruyama (2005) addressed the concerns of: network failure detection, identifying the syslog pattern that emerged during a failure, and to dynamically correlate events across devices.

Taking a different approach Lakhina *et al.* (2004) focused on network traffic anomalies, as opposed to a component of the network. Specifically, they looked at 'volume anomalies', sudden increases (or decreases) in volume of traffic between an origin point and destination. Splitting their methodology into three parts, first they detected a traffic anomaly by performing a PCA subspace separation, next they identified the anomaly by choosing the one[4] that minimised the projection of normal traffic conditions onto the 'abnormal' subspace. Lastly, they quantified the anomaly by estimating the number of bytes it consisted of.

Another use for log mining is for security. A paper by Suh-Lee *et al.* (2016), describes their interest in improving the effectiveness, and utilisation, of already existing data gathered by Security Information and Event Management (SIEM) systems. They tested 12 different classifiers, attempting to find the best performing classifier. Some of the best performing models were a voted perceptron, bayes net, and a random forest.

The work by Du *et al.* (2017) leveraged the volume of the log data in a deep learning approach. They built a Long Short-Term Memory (LSTM) deep neural network, attempting to model sequences of log messages as a natural language that follow certain patterns and grammar; that follow a strict logic, and control flow, akin to a natural language — albeit with less structure and diminished vocabulary.

2.3 Research involving log mining in astronomy

While there has been much research into utilising logs, for the most part focused on anomaly detection, to assist with security (threat detection), network faults, and fault detection within distributed systems, comparatively little research has been done about their use in astronomy. Of the work that has been done, arguably the most influential research has been performed by the group at the Atacama Large Millimetre/sub-millimeter Array (ALMA) telescope in Chile (shown in Figure 2.14).

[4] In their work, they constructed an as complete set of all anomalies as possible.

Attempting to improve how ALMA's software support team dealt with problems Gil *et al.* (2016) implemented the tools available in the ELK stack, with its aggregation capabilities and ease of administration mentioned as reasons for its implementation.

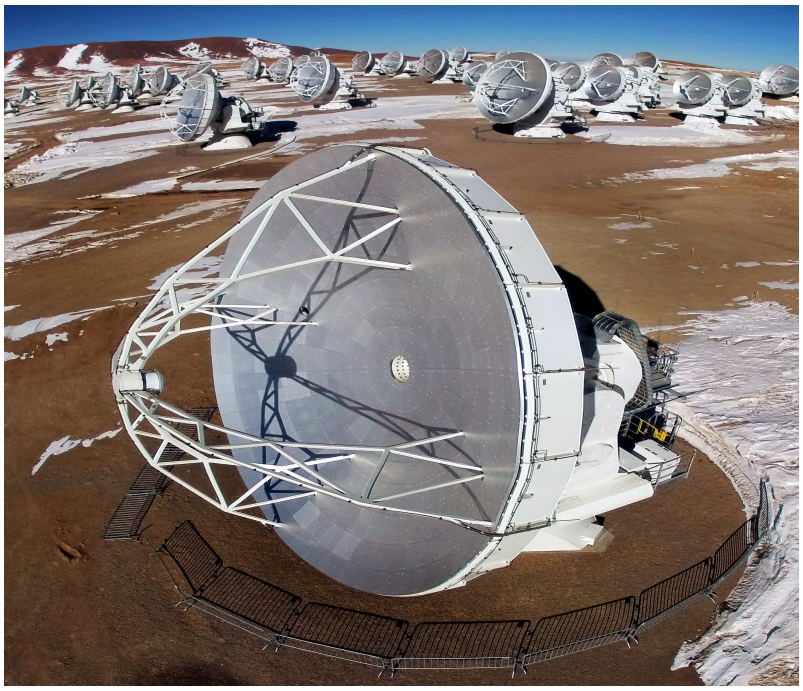

Figure 2.14: An aerial image taken of ALMA. Credit: EFE/Ariel Marinkovic (Marinkovic, 2013).

The author of the previous paper (Gil *et al.*, 2016) was involved with another project that attempted to go a step further.

Pettinato *et al.* (2019) investigated whether the sequences of tasks performed by ALMA could be accurately re-constructed, the intention then being to associate the tasks with various code components — performed in a semi-automated fashion. They approached this problem from the perspective topic modelling, performing Latent Dirichlet Allocation (LDA) to group log messages belonging to the same topic (task) together. Since each log event, recorded at ALMA, is a XML document, with the log message being an element of that, it also contained information on the code file and routine. Therefore, they were able to associate the various code components to the log messages, and by extension the topics, by referencing the log event that messages belongs to.

Another working group in Chile that have been exploring the use of logs, those at the Paranal Observatory, using the VLT[5] Survey Telescope (VST) (Schipani *et al.*, 2020) shown in Figure 2.15.

[5]Very Large Telescope.

Beginning in 2015, the authors describe their efforts made in systematic log file collection and analysis, aiming to identify long-term trends. They present a case study of the calibrated open-loop active optics corrections performed on the telescope's secondary mirror. The active optics system is meant to correct aberrations measured in the optics before an exposure is taken, for the secondary mirror this means correcting its position. The researchers were able to confirm that the position of the mirror did indeed linearly depend on the temperature and altitude, which is important as the VST's TCS implements corrections based on a linear dependence to temperature. They also able to revealed two interesting issues. When the open-loop correction for a linear astigmatism was applied to the x-axis and tilt along the y-axis, it was wrong by a non-negligible amount. And the linear astigmatism corrections were disabled for long periods of time.

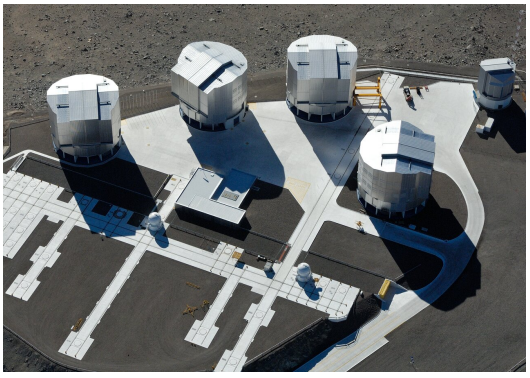

Figure 2.15: VLT and, at far right, VST telescopes at Cerro Paranal Credit: ESO/G.Hüdepohl (Hüdepohl, 2005)

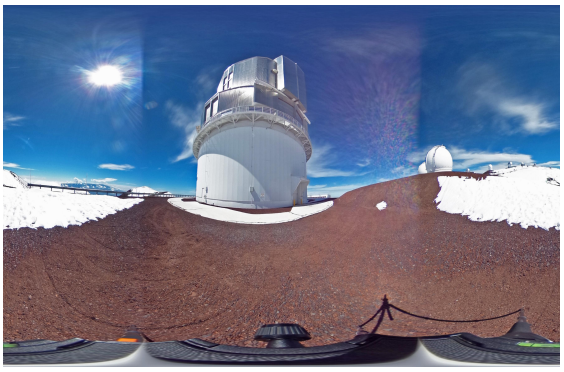

Figure 2.16: Subaru telescope at Hawai'i. Credit: NAOJ/Dr. Hideaki Fujiwara (Fujiwara, 2014)

Not only in Chile are astronomers benefiting from log collection and mining, Nakamura *et al.* (2020) have prototyped their own log analysis infrastructure using the ELK stack, citing the work done by the group at ALMA as inspiration. Working with the Subaru Telescope (shown in Figure 2.16), located at the Mauna Kea Observatory, Hawai'i, the group is looking to make use of 20 years worth of log data to establish a feasible cause for defects and to predict future errors. As of publication, they were able to import a year's worth of log data into an Elasticsearch database, however, they were not able to graph any of the data due to an indexing problem: searching the database takes too long.

2.4 Challenges working with logs

The group at the Subaru Telescope is not alone when it comes to facing challenges presented by working with logs, although their problems may be uniquely their own.

The most prominent of these challenges being that, the amount of logs is too much to examine manually — 15 GB of log data is generated daily at ALMA (Gil *et al.*, 2016). Another source of frustration results from the lack of structure that is inherent to many logs with formats and semantics varying dramatically between systems (Du *et al.*, 2017).

Additionally log files generally represent a single stream of events, however, a challenge arises when messages from multiple sources may interleave at runtime (e.g. multiple threads) or statically (different modules in a program) (Oliner *et al.*, 2012). They also mention that implementing extensive logging might pose a risk to a system's performance. Anomaly detection brings its own difficulties, as it can be difficult (if not impossible) to know beforehand what would be a feature that could represent a fault (Du *et al.*, 2017). Also anomaly detection should occur timeously, thus methods that require many passes over data are not applicable in many situations.

Challenging as it may be working with system logs, it proves to be worthwhile as they contain a rich amount of information that can provide useful insight.

MeerLICHT's logs are the same, having explored them earlier in the chapter it is clear that they contain enough information to identify the key processes the behind the telescope's function and where potential faults may occur. Investigating commercial tools uncovered two key aspects to working with system logs: parsing and mining. Being able to ingest logs in a manner that enables analysis is crucial, however, as noted by He *et al.* (2016) there is a lack of ready-to-use parsers. Mining was most commonly performed to implement fault detection (Xu *et al.*, 2009; Guo *et al.*, 2006), with a common technique being PCA subspace separation (Xu *et al.*, 2009; Lakhina *et al.*, 2004). While log analysis is more prevalent in applications of security, networking, and distributed system management, successes have been achieved in astronomy (Gil *et al.*, 2016; Pettinato *et al.*, 2019; Schipani *et al.*, 2020). The next chapter will detail the methodology used to analyse MeerLICHT's logs and will more fully examine the briefly mentioned HMM.

Chapter 3

The Hidden Markov Model

This chapter provides a motivation, and description, of the modelling technique and methodology used in this work to investigate the questions of anomaly detection, and state duration as they pertain to MeerLICHT.

Section 3.1 provides a brief introduction to HMMs, detailing their basic structure and some terminology. The section also introduces the basic problems of an HMM, while providing descriptions of several important quantities that an HMM calculates (e.g. likelihoods). Section 3.2 elaborates on previous work using HMMs and motivates their use in this work. Next, Section 3.3 details the algorithms that form the backbone of this work: the forward and backward algorithms, also explaining how numerical underflow (or overflow) is prevented.

3.1 Introducing HMMs

Hidden Markov Models (HMMs) are an extension of the theory of Markov chains (see Rabiner (1989); Jurafsky and Martin (2020); Zucchini *et al.* (2016)). A discrete-time Markov chain is a sequence of random variables $\{C_t : t \in \mathbb{N}\}$ that satisfy the Markov property

$$\Pr\left(C_{t+1} \mid C_t, \ldots, C_1\right) = \Pr\left(C_{t+1} \mid C_t\right), \tag{3.1}$$

for all $t \in \mathbb{N}$, i.e. the probability of the appearance of a random variable C_{t+1} depends only on the appearance of the previous random variable C_t.

Satisfying the Markov property means it is only necessary to consider the most recent value C_t, rather than the entire sequence up to time t. Equation 3.1 describes a *first-order* Markov chain; if Equation 3.1 was conditional on C_{t-1} too it would be a *second-order* Markov chain and so on.

Following the convention as laid out by Zucchini *et al.* (2016), it is more compact to define the history of the sequence $(C_1, C_2, \ldots, C_t)$ as $\mathbf{C}^{(t)}$. As a result, Equation

3.1 can equivalently be written as

$$\Pr(C_{t+1} \mid \mathbf{C}^{(t)}) = \Pr(C_{t+1} \mid C_t). \tag{3.2}$$

Zucchini *et al.* (2016) point out: the Markov property relaxes the assumption of independence between the random variables $\{C_t\}$ and introduces a specific dependency to them that is mathematically convenient.

There are several quantities that are required to be able to formally specify a Markov chain. The first of these are the conditional probabilities, that define the probability of observing a random variable j (the state) at a future point $s+t$ in time, given that the current observed state is i at time s. These probabilities, known as the **transition probabilities**, are denoted as follows:

$$\gamma_{ij}(t) \equiv \Pr(C_{s+t} = j \mid C_s = i), \tag{3.3}$$

and are commonly used to define the (i,j)th element of a matrix $\mathbf{\Gamma}(t)$ (called the transition probability matrix or t.p.m). In this work the assumption is made that the Markov chains under investigation are homogeneous, therefore, the transition probabilities $\gamma_{ij}(t)$ do not depend on the time s when the transition takes place.

To make a single transition one uses $\mathbf{\Gamma}(1)$ which represents the one-step t.p.m, i.e. its elements are the conditional probabilities from Equation 3.2 at $t = 1$, and is abbreviated as $\mathbf{\Gamma}$ for convenience. The important properties of $\mathbf{\Gamma}$ are that its elements must always be positive and its rows sum to 1. Since all finite state-space homogeneous Markov chains will satisfy the Chapman-Kolmogorov equations[1], which imply that for all $t \in \mathbb{N}$

$$\mathbf{\Gamma}(t) = \mathbf{\Gamma}^t = \mathbf{\Gamma}(1)^t, \tag{3.4}$$

thus $\mathbf{\Gamma}(t)$ is the tth exponent of $\mathbf{\Gamma}$.

This leads into the next quantity needed to define a Markov chain the **states**. The states are the m values the random variables $\{C_t\}$ can take. As such, this makes $\mathbf{\Gamma}$ a $(m \times m)$ square matrix.

The final quantity needed to specify a Markov chain is its **initial distribution**, which is a row vector of the unconditional probabilities $\Pr(C_t = j)$, and is denoted as follows:

$$\boldsymbol{\delta}(t) = (\Pr(C_t = 1), \ldots, \Pr(C_t = m)), \quad t \in \mathbb{N}; \tag{3.5}$$

at $t = 1$, $\boldsymbol{\delta}(1)$. Importantly, it is possible to show that

$$\boldsymbol{\delta}(t)\mathbf{\Gamma} = \boldsymbol{\delta}(t+1); \tag{3.6}$$

the proof of which is shown in (Appendix A.3).

[1] $\mathbf{\Gamma}(t+u) = \mathbf{\Gamma}(t)\mathbf{\Gamma}(u)$

3.1.1 The traditional HMM

What separates a normal Markov chain from a HMM, is whether one can directly observe the Markov chain in question or not. In the context of HMMs, 'observe' means to take a measurement of the process that relies on the value of the hidden/latent random variable $C_t : t \in \mathbb{N}$ although C_t itself cannot be measured (see Equation 3.1).

An HMM, $\{X_t : t \in \mathbb{N}\}$, has two parts to consider. The first is a hidden/latent 'parameter process' $\{C_t : t = 1, 2, \ldots\}$ that satisfies the Markov property, and in the context of this work represents the states that MeerLICHT occupies e..g. 'Observing', 'Sleeping', 'Waiting', 'Calibrating', etc. To be clear, this is a choice made in the context of this work. The states in an HMM might not have a clear meaning and interpretation of them should be done with caution. Secondly is a 'state-dependent process' $\{X_t : t = 1, 2, \ldots\}$, which are the unique lines of text present in a log file in this instance, where the distribution of X_t only depends on the current state C_t. If the Markov chain C_t has m states, then X_t is called a m-state HMM (Zucchini *et al.*, 2016).

Letting $\mathbf{C}^{(t)}$ and $\mathbf{X}^{(t)}$ be the histories, from time 1 to t, of the parameter and state dependent process' respectively, one can represent the simplest model of this kind by:

$$\Pr(C_t \mid \mathbf{C}^{(t-1)}) = \Pr\left(C_t \mid C_{t-1}\right), \quad t = 2, 3, \ldots \tag{3.7a}$$

$$\Pr(X_t \mid \mathbf{X}^{(t-1)}, \mathbf{C}^{(t)}) = \Pr\left(X_t \mid C_t\right), \quad t \in \mathbb{N}. \tag{3.7b}$$

Equations (3.7a) and (3.7b) illustrate that $\{C_t\}$ satisfies the Markov property and that any observation made (X_t) of the state-dependent process only depends on the current state respectively.

Equation (3.7b) can be extended to the case where the random variables X_t and C_t take on discrete values,

$$p_i(x) = \Pr\left(X_t = x \mid C_t = i\right), \quad \text{for } i = 1, 2, \ldots, m; \tag{3.8}$$

with p_i being the probability mass function of X_t given that the Markov chain is in a specific state i at a time t. The m probability distributions p_i are referred to as **state-dependent distributions**. The state-dependent distributions and the total number of unique observation are the required additional specification for an HMM, over those required for a Markov model.

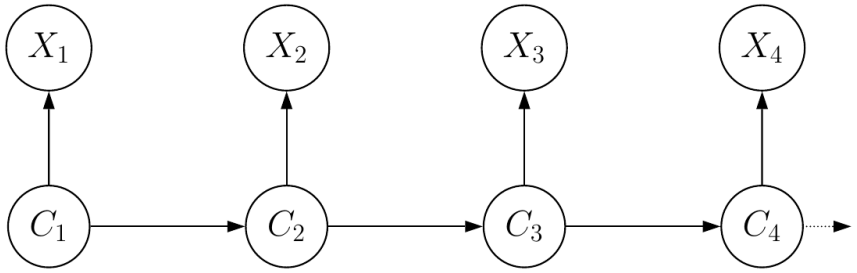

Figure 3.1: This illustration of a basic HMM shows the 'hidden' parameter process, the sequence $\{C_t\}$, and how the observed sequence, $\{X_t\}$, depends only on the current state (Zucchini *et al.*, 2016).

3.1.2 Marginal distributions for an HMM

Two quantities frequently used in HMM applications: the marginal distribution of X_t, often including the higher order marginals (X_t, X_{t+k}) too, and the likelihood of consecutive observations.

It is useful to note that the joint distribution of a set of random variables Φ_i represented by a directed graphical model is given by:

$$\Pr(\Phi_1, \Phi_2, \ldots, \Phi_N) = \prod_{i=1}^{N} \Pr(\Phi_i \mid \text{pa}(\Phi_i)); \tag{3.9}$$

where $\text{pa}(\Phi_i)$ is the 'parent' of Φ_i in the set $\Phi_1, \Phi_2, \ldots, \Phi_N$[2].

To demonstrate, consider a directed graph with four random variables[3] X_t, X_{t+k}, C_t, C_{t+k} (refer to Figure 3.1 if needed). Following Equation 3.9 the joint probability is given by:

$$\Pr(X_t, X_{t+k}, C_t, C_{t+k}) = \Pr(C_t) \Pr(X_t \mid C_t) \Pr(C_{t+k} \mid C_t) \Pr(X_{t+k} \mid C_{t+k}); \tag{3.10}$$

since C_t has no parent and $\text{pa}(X_t) = \text{pa}(C_{t+k}) = C_t$ etc. Using this convenient way to express the joint probabilities, all that needs to be done to calculate the marginal $\Pr(X_t, X_{t+k})$ is to sum over all values the states C_t and C_{t+k} can take.

For discrete-valued observations X_t and X_{t+k}, defining $\delta_i(t) = \Pr(C_t = i)$ for

[2]The notation of Equation 3.9 is mirrored from Zucchini *et al.* (2016), with a fuller proof offered by Jordan (2004).

[3]For positive k.

$t = 1, \ldots, T$, the marginal $\Pr(X_t, X_{t+k})$ is given by:

$$\begin{aligned}
&\Pr(X_t = v, X_{t+k} = w) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} \Pr(X_t = v, X_{t+k} = w, C_t = i, C_{t+k} = j) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} \underbrace{\Pr(C_t = i)}_{\delta_i(t)} \overbrace{\Pr(X_t = v \mid C_t = i)}^{p_i(v)} \underbrace{\Pr(C_{t+k} = j \mid C_t = i)}_{\gamma_{ij}(k)} p_j(w) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{m} \delta_i(t) p_i(v) \gamma_{ij}(k) p_j(w).
\end{aligned}$$

Expressed as the double sum as a product of matrices,

$$\Pr(X_t = v, X_{t+k} = w) = \boldsymbol{\delta}(t)\mathbf{P}(v)\boldsymbol{\Gamma}^k\mathbf{P}(w)\mathbf{1}'. \tag{3.11}$$

Or equivalently, by making use of Equation 3.6 as,

$$\Pr(X_t = v, X_{t+k} = w) = \boldsymbol{\delta}(1)\boldsymbol{\Gamma}^{(t-1)}\mathbf{P}(v)\boldsymbol{\Gamma}^k\mathbf{P}(w)\mathbf{1}'. \tag{3.12}$$

Unpacking Equations 3.11 and 3.12: $\mathbf{P}(v)$ is defined as a diagonal matrix whose ith element is $p_i(v)$ (similarly for $\mathbf{P}(w)$); Γ^k the kth multiple of t.p.m Γ; $\boldsymbol{\delta}(t)$ are the starting unconditional probabilities, or just the initial distribution $\boldsymbol{\delta}(1)$ post-multiplied by the t.p.m $t-1$ times, where $\delta_i(t)$ is a single element in the vector; lastly $\mathbf{1}'$ which is a column vector of ones.

Following this demonstration, on obtaining expression for the marginal probability for two random variables, it is relatively straightforward to obtain similar ones for higher-order marginals.

3.1.3 The likelihood of a sequence

With a straightforward way to obtain expressions for marginal probabilities, we now detail the expression for the likelihood L_T of T consecutive observations $x_1, x_2, .., x_T$.

Although Rabiner (1989) and Zucchini *et al.* (2016) mention that to fully compute the likelihood requires $O(Tm^T)$ operations, Baum *et al.* (1972) show that it is possible to use $O(Tm^2)$ operations. This opens the way to estimating parameters via numerical maximization.

Proposition 1 *We now propose the likelihood is given by*:

$$L_T = \boldsymbol{\delta}\mathbf{P}(x_1)\boldsymbol{\Gamma}\mathbf{P}(x_2)\boldsymbol{\Gamma}\mathbf{P}(x_3)\ldots\boldsymbol{\Gamma}\mathbf{P}(x_T)\mathbf{1}' \tag{3.13}$$

Proof. For the case of discrete observations, the likelihood is given by the following expression:

$$L_T = \Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)}) = \sum_{c_1,c_2,\ldots,c_T}^{m} \Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)}, \mathbf{C}^{(T)} = \mathbf{c}^{(T)}).$$

As a result of applying Equation 3.9, it can be shown that

$$\begin{aligned} L_T &= \sum_{c_1,\ldots,c_T}^{m} (\delta_{c_1}\gamma_{c_1,c_2}\gamma_{c_2,c_3}\cdots\gamma_{c_{T-1},c_T})(p_{c_1}(x_1)p_{c_2}(x_2)\cdots p_{c_T}(x_T)) \\ &= \sum_{c_1,\ldots,c_T}^{m} \delta_{c_1}p_{c_1}(x_1)\gamma_{c_1,c_2}p_{c_2}(x_2)\gamma_{c_2,c_3}\cdots\gamma_{c_{T-1},c_T}p_{c_T}(x_T) \\ &= \boldsymbol{\delta}\mathbf{P}(x_1)\boldsymbol{\Gamma}\mathbf{P}(x_2)\boldsymbol{\Gamma}\mathbf{P}(x_3)\ldots\boldsymbol{\Gamma}\mathbf{P}(x_T)\mathbf{1}' \end{aligned}$$

which is exactly Equation 3.13. □

Seemingly innocuous, it is this expression of the likelihood Equation 3.13 which gives rise to the so-called **forward algorithm**, a means of recursively computing the likelihood. This drastically reduces the computational expense when, for example, addressing parameter estimation or calculating the likelihood itself (Zucchini *et al.*, 2016). As its name implies, the forward algorithm requires a forward pass through the data. However, it is also possible to make a backwards pass through the data (**backwards algorithm**) which aids in the task of decoding.

Calculating the likelihood (evaluating Equation 3.13); parameter estimation (estimating the elements of $\boldsymbol{\Gamma}$, $\boldsymbol{\delta}(t)$, p_i); decoding (finding the state sequence) — these are the questions that Rabiner (1989) refers to as the 'three basic problems for HMMs'. In the context of MeerLICHT, this means calculating the most likely state sequence from a sequence of log messages (decoding), estimating the probabilities of a specific log message given the state (p_i) and the state transitions.

3.1.4 The basic problems of an HMM

To conclude this introduction to HMMs, it is necessary to touch upon the 'basic' problems that need to be addressed so that a HMM can be useful in real world applications (Rabiner, 1989). The first problem to be addressed, is how to efficiently calculate the likelihood (L_T) that an observing sequence $x_1, x_2, \ldots, x_T$ was generated by a given m-state HMM parameterised by: $\boldsymbol{\delta}$ (initial distribution), $\boldsymbol{\Gamma}$ (t.p.m), and p_i (state-dependent distributions). This is the most important problem that must be addressed, as the likelihood is required to calculate several quantities (e.g. state predictions, and decoding) and used to train the model.

The next problem, known as **decoding**, is to discover the best state (or sequence of states), given a sequence of observations $x_1, x_2, \ldots, x_T$ and an m-state HMM. This problem comes in two different variations, the first being **local** decoding, this is to find the most probable state i at time $t \leq T$ (i.e. at a specific instance in time). While the other, **global** decoding, finds the most probable state sequence $c_1, c_2, \ldots, c_T$.

Lastly, the model parameters need to be estimated and re-estimated (i.e. $\boldsymbol{\delta}$, $\boldsymbol{\Gamma}$, and p_i). The model parameters were estimated using 3 different procedures. The first was by using the labelled training set to calculate the frequencies of either the emission (p_i) and transmission ($\boldsymbol{\Gamma}$) probabilities, which we refer to as the **frequency** method. The second was an implementation that made use of the **self-train** algorithm, and was referred to as such (Tamposis *et al.*, 2018). The last approach directly maximises the likelihood function (Zucchini *et al.*, 2016), as opposed to using the popular expectation maximisation (EM) algorithm as proposed by Rabiner (1989), referred to as the **MLE** (maximum likelihood estimation) method.

The ability to efficiently calculate the likelihood is the crux to solving the later problems (and others). This is achieved, as mentioned above, by the use of the forward algorithm, which offers an efficient means of calculating the likelihood.

3.2 Motivation for using HMMs

Having detailed what an HMM is, it is prudent to offer a motivation for their use in this work. The previous chapter mentioned a number of different ways that log analysis could be practically implemented in an anomaly detection problem. Ranging from building an LSTM neural network (Du *et al.*, 2017), to incorporating a GMM (Guo *et al.*, 2006), or the more popular use of performing PCA to create separate subspaces (Xu *et al.*, 2009; Lakhina *et al.*, 2004). Notably, Yamanishi and Maruyama (2005) incorporated HMMs in their dynamic mining of syslogs.

Accordingly, this begins with an elaboration on some of the previous work that makes use of HMMs to accomplish a variety of aims, and concludes by commenting on how those previous works motivate the use of a HMMs for the MeerLICHT project.

3.2.1 Research involving HMMs

Typically HMMs have been applied in speech-recognition and speech processing, with much success (Bilmes, 2006; Rabiner, 1989). Not only that, they have also proved useful in a much wider variety of fields. An early adaption of HMMs to

perform system monitoring came from Heck and McClellan (1991), where they used an HMM to model tool wear in automatic machining systems. They were able to identify several characteristic states that a given tool would progress through, before becoming worn, which helped them predict when the tool would hit the failure point.

Around the same time Smyth and Mellstrom (1991), demonstrated the use of HMMs for performing fault detection on an antenna pointing system. This particular system belonged to the Deep Space Network (DSN), used to provided telecommunications between Earth and various interplanetary spacecraft. The authors note that component fault may manifest indirectly via changes in observed sensor readings; thus causal relationships can be difficult to establish between sensor readings and faults. The approach they took was to estimate the state-dependent probabilities via a feed-forward neural network, or a Gaussian classifier, and to use those to estimate the probability of being in state i at time t with an HMM. They offer a more in-depth explanation in a follow-up paper (Smyth, 1994*a*), while also expanding on the online monitoring scenario. Another follow-up (Smyth, 1994*b*), builds on their previous work, contributing to the detection of unknown states using a 'hybrid' model: where the state-dependent probabilities are estimated via either a 'generative' or 'discriminative' model[4].

Panuccio *et al.* (2002), made use of multiple HMMs to simplify the task of clustering together similar signals obtained from an electroencephalogram (EEG). They trained N individual HMMs (λ_i), one for each sequence of the dataset (D). This enabled them to build a measure matrix L, whose elements were the probability that the model generated a sequence S_j ($\Pr(S_j|\lambda_i)$) which they then performed the clustering on. The simplification being that, rather than clustering together the individual sequences in D which is difficult, they are able to cluster together the elements of L (which are points) more easily. Another interesting use case is the work done by Michelot *et al.* (2016) where they built an **R** (R Core Team, 2021) package to perform the modelling of animal movement data.

HMMs have also been applied in a 'process mining' application, to build a process model from mined system logs (Gadler *et al.*, 2017). 'Process mining' is a set of techniques that aim to extract knowledge, of a particular process, from the logs that particular process generates during its execution. Firstly, they generate a process model from the available event logs (called 'process discovery'). They then compare this 'discovered' process model to an ideal model of the same process (called 'conformance checking'). These graphs are usually directed graphs, with nodes representing logs and edges representing transitions between them respectively. The

[4]Discriminative models directly estimate $\Pr(\mathbf{C}|\mathbf{X})$ (e.g. feed-forward neural networks e.t.c.), generative models model $\Pr(\mathbf{X}|\mathbf{C})$ (e.g. Gaussian classifiers, kernel density estimators e.t.c.).

case study analysis in this work was performed on an Italian company that manufactures precision turning parts, milling parts, and technical parts. The aim of the research was to develop a way to automatically build process models that represent the actual intents (or uses) of users while interacting with their software.

3.2.2 The reason for their use

While the previous research using HMMs provided above is certainly not exhaustive, it demonstrates that HMMs are a useful multi-purpose model for time-series analysis. Given that one of the core aims of this work is to perform system-monitoring/fault detection on MeerLICHT the previous research cited above — and others Ying *et al.* (2000); Hovland and McCarragher (1998) etc. — suggest that using an HMM to perform fault detection is a valid choice.

Using system logs as a data source for a vanilla HMM appears to be fairly uncommon, and even then it appears to be restricted to applications of workflow management (e.g. Damevski *et al.* (2016)). Although, there have been instances of a special type of HMM being used for fault detection (e.g. Tan and Xi (2008); Min and Shun-Zheng (2006)), called a Hidden Semi-Markov model (HSMM).

A vanilla HMM has an implicit duration density that is geometric, which may be unsuitable for many applications (Yu, 2010; Langrock and Zucchini, 2011). The HSMM makes up for this and allows for an arbitrary duration density to be specified, however this is above the scope of this work and so the implicit duration is assumed (the topic of HSMMs will be addressed in the concluding chapter).

3.3 The forward & backward algorithms

The forward algorithm is a recursive evaluation of the matrix expression of the likelihood Equation 3.13. This forward recursion is arrived at immediately, following the appropriate definition of the vector $\boldsymbol{\alpha}_t$, for $t = 1, 2, \ldots, T$ and whose elements are referred to as **forward probabilities**. Its reverse, the backward algorithm, similarly follows with the definition of the vector $\boldsymbol{\beta}_t$, for $t = 1, 2, \ldots, T-1$, with elements called **backwards probabilities**.

How to arrive at the recursion relations is tackled first and then followed by a description of the algorithms themselves.

3.3.1 Recursive relations

Starting with the forward recursion, we define the vector $\boldsymbol{\alpha}_t$, for $t = 1, 2, \ldots, T$.

This is the vector of forward probabilities as follows (with the convention that empty products are treated as the identity matrix):

$$\boldsymbol{\alpha}_t = \boldsymbol{\delta}\mathbf{P}(x_1)\boldsymbol{\Gamma}\mathbf{P}(x_2)\cdots\boldsymbol{\Gamma}\mathbf{P}(x_t) = \boldsymbol{\delta}\mathbf{P}(x_1)\prod_{s=2}^{t}\boldsymbol{\Gamma}\mathbf{P}(x_s). \tag{3.14}$$

By this definition, it becomes apparent that:

$$L_T = \boldsymbol{\alpha}_T\mathbf{1}', \quad \text{and} \tag{3.15a}$$

$$\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\mathbf{P}(x_t), \ \text{ for } t \geq 2. \tag{3.15b}$$

They also note, the forward recursion ($\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\mathbf{P}(x_t)$) then makes it convenient to arrange the computation of the likelihood Equation 3.13 in the following way:

$$\begin{gathered}\boldsymbol{\alpha}_1 = \boldsymbol{\delta}\mathbf{P}(x_1);\\ \boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}\mathbf{P}(x_t), \quad \text{for } t = 2, 3, \ldots, T;\\ L_T = \boldsymbol{\alpha}_T\mathbf{1}'.\end{gathered}$$

The final algorithm takes steps to prevent the underflow (or overflow) of the elements of $\boldsymbol{\alpha}_t$ and the estimated parameters too.

The backward recursion also becomes apparent in a similar way. Define the vector $\boldsymbol{\beta}_t$, for $t = 1, 2, \ldots, T$, of backwards probabilities by,

$$\boldsymbol{\beta}_t' = \boldsymbol{\Gamma}\mathbf{P}(x_{t+1})\boldsymbol{\Gamma}\mathbf{P}(x_{t+2})\cdots\boldsymbol{\Gamma}\mathbf{P}(x_T)\mathbf{1}' = \left(\prod_{s=t+1}^{T}\boldsymbol{\Gamma}\mathbf{P}(x_s)\right)\mathbf{1}'. \tag{3.16}$$

Again empty products are treated as the identity by convention, with the case $t = T$ therefore yielding $\boldsymbol{\beta}_T = \mathbf{1}$, and the recursion in this instance is

$$\boldsymbol{\beta}_t' = \boldsymbol{\Gamma}\mathbf{P}(x_{t+1})\boldsymbol{\beta}_{t+1}', \ \text{ for } t = 1, 2, \ldots, T-1. \tag{3.17}$$

These recursion relations are the reason that the computation of likelihood evaluation, and other related calculations, is so inexpensive using the forward and backward algorithms.

3.3.2 Onto the algorithms

A known problem is underflow or overflow, of the parameter estimates. This can be prevented by **scaling** the estimates. Both are neutralised in the same way, so the focus will rest on underflow.

For the case of discrete observations, the elements of $\boldsymbol{\alpha}_t$ get progressively smaller, resulting from being the product of many probabilities[5]. Naturally, to prevent the

[5] A similar thing happens to $\boldsymbol{\beta}_t$, however, in the opposite direction.

underflow issues, one would want to take the logarithm[6] of the likelihood, however, since the likelihood is a product of matrices its logarithm is not the sum of the logarithms of its factors. Instead the method of preventing underflow used in this work comes from Zucchini *et al.* (2016), and involves scaling the vector $\boldsymbol{\alpha}_t$ at each time step t so that its elements sum to 1. One then keeps track of the sum of the logarithms of the scaled factors to calculate the log-likelihood, and each factor separately so as to record them at each t.

The scaling can be applied quite simply. Define, for $t = 1, 2, \ldots, T$, the vector

$$\boldsymbol{\phi}_t = \frac{\boldsymbol{\alpha}_t}{\omega_t} \tag{3.18}$$

where $\omega_t = \sum_i \alpha_t(i) = \boldsymbol{\alpha}_t \mathbf{1}'$. Zucchini *et al.* (2016) note several consequences that result from this definition, most relevantly for this work:

$$\omega_t \boldsymbol{\phi}_t = \omega_{t-1} \boldsymbol{\phi}_{t-1} \boldsymbol{\Gamma} \mathbf{P}(x_t); \tag{3.19a}$$

$$L_T = \boldsymbol{\alpha}_T \mathbf{1}' = \omega_T (\boldsymbol{\phi}_T \mathbf{1}') = \omega_T. \tag{3.19b}$$

Hence, $L_T = \omega_T = \prod_{t=1}^{T} (\omega_t / \omega_{t-1})$. Making use of Equation 3.19a,

$$\omega_t = \omega_{t-1} (\boldsymbol{\phi}_{t-1} \boldsymbol{\Gamma} \mathbf{P}(x_t) \mathbf{1}'), \tag{3.20}$$

leads to the following conclusion

$$\log(L_T) = \sum_{t=1}^{T} \log(\frac{\omega_t}{\omega_{t-1}}) = \sum_{t=1}^{T} \log(\boldsymbol{\phi}_{t-1} \boldsymbol{\Gamma} \mathbf{P}(x_t)), \tag{3.21}$$

an expression for the log-likelihood in terms of the scaled forward probabilities.

The forward variable at $t = 1$ ($\boldsymbol{\delta} \mathbf{P}(x_1)$) is calculated; then divided by its sum over all states ($\omega_1 = \sum_i \boldsymbol{\delta} \mathbf{P}(x_1)$) to get the first scaled forward variable $\boldsymbol{\phi}_1$. The log-likelihood at this step (L_1) is calculated by then taking the logarithm of that sum. Since they will be needed for future calculations, the logarithms of each vector of forward probabilities is stored in an $m \times T$ matrix $\log(\boldsymbol{A})$. The recursion relation from earlier is used to repeat these steps for the entire length (T) of the given observing sequence. Utilising $\boldsymbol{\phi}_{t-1}$, from the previous time-step, the next ω_t is then calculated. That is promptly followed by the log-likelihood L_t at that step and the next $\boldsymbol{\phi}_t$; allowing for the next $\log(\boldsymbol{A})$ entry to be calculated.

The algorithm returns the value of the log-likelihood at $t = T$, a vector of scaled forward probabilities for $t = T$, and vectors of the log-forward probabilities for each time t (stored in a matrix).

Algorithm 3.1 summarises the forward algorithm.

[6] All logarithms in this work use base e.

Algorithm 3.1 The Forward Algorithm

Input: t.p.m $\boldsymbol{\Gamma}$, initial dist. $\boldsymbol{\delta}$, state-dep dist. $\mathbf{P}(x_t)$, observed sequence $\mathbf{X}^{(T)}$
Output: log-likelihood L_T, scaled fwd prob $\boldsymbol{\phi}_T$, log-fwd prob $\log(\boldsymbol{A})$

$\omega_1 \leftarrow \sum_i \boldsymbol{\delta}\mathbf{P}(x_1)$
$L_1 \leftarrow \log(\omega_1)$; $\boldsymbol{\phi}_1 \leftarrow \boldsymbol{\delta}\mathbf{P}(x_1)/\omega_1$ ▷ Log-likelihood/Scaling
$\log(\boldsymbol{\alpha}_1) \leftarrow L_1 + \log(\boldsymbol{\phi}_1)$ ▷ Get first $\log(\boldsymbol{A})$ entry
for $t = 2, \ldots, T$ **do**
    $\omega_t \leftarrow \sum_i \boldsymbol{\phi}_{t-1}\boldsymbol{\Gamma}\mathbf{P}(x_t)$ ▷ Recursion
    $L_t \leftarrow L_{t-1} + \omega_t$; $\boldsymbol{\phi}_t \leftarrow \boldsymbol{\phi}_{t-1}\boldsymbol{\Gamma}\mathbf{P}(x_t)/\omega_t$ ▷ Update
    $\log(\boldsymbol{\alpha}_t) \leftarrow \log(\boldsymbol{\phi}_t) + L_t$ ▷ Get next $\log(\boldsymbol{A})$ entry
end for

The backwards algorithm is similar, although starting at $t = T$ and moving backwards. First $\boldsymbol{\beta}'_T$ is scaled, divided by its sum ($\omega_T = \sum_i \boldsymbol{\beta}'_T$), to get the scaled backward variable $\boldsymbol{\phi}'_T$; then the log of ω_T. Using the backwards recursion, and $\boldsymbol{\phi}'_T$, the intermediary variable at $t = T - 1$ is calculated ($\boldsymbol{\theta}'_{T-1}$). The entry at $t = T - 1$ in the matrix of the logarithm of backwards probabilities is then, $\log(\boldsymbol{\theta}'_{T-1})$ plus ω_T. The backward variable at this time-step $\boldsymbol{\phi}_{T-1}$ and the sum ω_{T-1}, are then calculated using $\boldsymbol{\theta}'_{T-1}$. This repeats till $t = 1$.

Algorithm 3.2 summarises the backward algorithm.

Algorithm 3.2 The Backward Algorithm

Input: t.p.m $\boldsymbol{\Gamma}$, final prob. $\boldsymbol{\beta}'_T$, state-dep dist. $\mathbf{P}(x_t)$, observed sequence $\mathbf{X}^{(T)}$
Output: log-bkd prob $\log(\boldsymbol{B})$

$\omega_T \leftarrow \sum_i \boldsymbol{\beta}'_T$
$\boldsymbol{\phi}'_T \leftarrow \boldsymbol{\beta}'_T/\omega_T$; $\omega_T \leftarrow \log(\omega_T)$ ▷ Scaling
for $t = T - 1, \ldots, 1$ **do**
    $\boldsymbol{\theta}'_t \leftarrow \boldsymbol{\Gamma}\mathbf{P}(x_{t+1})\boldsymbol{\phi}'_{t+1}$ ▷ Intermediary/Recursion
    $\log(\boldsymbol{\beta}_t) \leftarrow \log(\boldsymbol{\theta}'_t) + \omega_{t+1}$ ▷ Get next $\log(\boldsymbol{B}$ entry
    $\boldsymbol{\phi}'_t \leftarrow \boldsymbol{\theta}'_t/\sum_i \boldsymbol{\theta}'_t$; $\omega_t \leftarrow \omega_{t+1} + \log(\sum_i \boldsymbol{\theta}'_t)$ ▷ Update
end for

The algorithm is then able to return the logarithm of the backwards probabilities, $\log(\boldsymbol{B})$. The algorithms also store the next logarithmic probability at different points during their respective recursive loops, which is another difference. In the backward algorithm it happens right after the recursive step, while in the forward algorithm it first updates the likelihood and scaled probability.

While not explicitly shown in this work, it is proved in Zucchini *et al.* (2016) (specifically pages $65 - 69$) that the elements of $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$ are indeed probabilities.

Chapter 4

Implementation

This chapter covers the implementation of an HMM to perform fault diagnosis on the MeerLICHT telescope. Section 4.1, explains how the model was built using the information from the previous sections (basic structure, likelihood, forward algorithm etc.) and how the model parameters were estimated. Section 4.2 details the process of estimating error values for the parameters and how they were used to build approximate confidence intervals. Then the procedures for evaluating the validity of the model, and choosing the correct one, are explained in Section 4.3. The final section, Section 4.4, provides more details on the anomaly detection procedures used.

4.1 Building the model

This section will address key issues of its construction, for example the choice of state-dependent distribution used or how the parameter estimation (re-estimation) was performed, or calculating errors for the estimated parameters. It also discusses how the number of hidden states was selected.

In this work the HMM is not used in a traditional way. The latent states are treated as known. While it may seem strange to use a latent state model in this way, the upside of using an HMM is that the model is not a black box and produces descriptive probability distributions.

4.1.1 Data and data preprocessing

Raw data

The raw data is formatted as seen in Chapter 2 and makes it extremely difficult to work with.

Section 2.4 makes mention of some of the challenges researchers or developers face when it comes to parsing data into a working environment, specifically in a format that makes working with the data less challenging, with many of them implementing custom parsing solutions.

The raw data for this work is taken from the Hardware Manager log files. The reason being that due to the way the domain isolation is structured, the HM log files provide the best overview of the whole of the telescope's operation.

The full dataset consists of seven days of MeerLICHT's HM log files, captured over the period from the morning of December 06 to the morning of December 13 2019. The raw log data was captured in 5 plain text files, the format of which can be seen in Figure 2.1. The largest of these text files was 8.3 Mb and had 100960 lines of text (recorded log messages). In total there was 24 Mb of text data and it was comprised of 251758 lines of text.

Preprocessing

To use an HMM to model the sequence of messages appearing in the logs not all the information present in a single line is needed. Only the order of the sequence and the type of message are really required. The order of the sequence is fairly simple to extract, as each line contains a timestamp, although there are limitations due to the finite precision available for the timestamps. Difficulty arises when it comes to ascertaining the type of an individual log line. This is partly due to the fact that the logs used in this work are not exhaustive (i.e. they do not contain every possible message), and partly to not being able to access the source code that creates the logs (it is proprietary). Although the logs were not exhaustive, the message types were still inferred from the data.

Taking inspiration from Xu (2010) and using the constant parts of each log line as markers of a specific message type, the message types were able to be inferred by comparing those constant parts of each log line and grouping those lines with the same constant parts together as a single type.

The logs were parsed using a regular expression which extracted the timestamp, date, verbosity, and the tokenized constant part of the message. Considerations for the constant part of each log line were arrived at after a period of visual inspection and are displayed in Table 4.1.

Certain error messages that occurred in the logs had printouts that spanned more than a single line, however since the first line contained most (if not all) of the relevant information, only the first line was used. This also helped reduce the volume of the data.

Table 4.1: Tokens that were considered constant or variable in this work.

Constant Part	**Example**
Single word	"Observation"; "correction", etc.
Combined word	"ComplexCommandItemOverhead"; "PostProcessCommand", etc.
Alphanumeric	"OH11", "X1", "Y1", etc.
Variable Part	**Example**
Numeric	1.83460666265773
Command ID	d3f3a6f2-dcc3-432c-a8bb-e65dc717e3c8

Once each line was stripped of their variable parts, using only the tokenized message, the lines were then grouped together by the contents of those tokenized messages. Each group was then given a label ("OBS_1", "OBS_2", etc.), resulting in a sequence of those labels that was then used by the HMM instead of the entire tokenized message. A dictionary of all the individual label/message pairs is then kept, so as to maintain consistency between the pairs throughout this work. In total 560 unique log lines were identified. Once each log line had been assigned an observation symbol, the total data set was split in two.

The last preprocessing step was to label each log line present in the dataset with a state (with states as seen in Table 4.2). This means treating the latent states as if they were not hidden. The reason for doing so is two-fold.

Firstly Drzał *et al.* (2016) have already predefined several suggested states that the system can transition into (Chapter 2) which were used and the intention of this work was always to have at least two predefined states (normal/abnormal). This will help avoid the problem of the states in an HMM not having any operational meaning (as mentioned in Section 3.1.1).

The second reason, is that it becomes easier to be more rigorous with model checking. Comparing the model outputs with the labels in the test set allows for measures such as the accuracy or recall to be calculated.

Clean data

Once the preprocessing had been completed, getting rid of excess lines of text, tokenizing the log lines, and giving them an observing label, what remained was 111359 tokenized log lines.

This set of 111359 tokenized log lines was then split into a training data set (used to build the model) and a test data set (used to test the model). The training set contained 98199 log lines ($\approx$ 12% of the data), while the test set contained 13240 log lines.

Since each log line was represented by an observing symbol (assigned during the

preprocessing), the final data that was to build the model was a sequence of 98119 observing symbols.

4.1.2 The state-dependent distribution

With the data preprocessed and in a format that is compatible with an HMM, the next big decision to be made is which state-dependent distribution would be the most suitable to be used with this data.

Given the nature of the data, a likely choice of a state-dependent distribution would be the multinomial distribution, with n_t trials at time t and $q \geq 2$ outcomes to each trial (where q is the number of unique observations). However, the general multinomial distribution was not used due to unresolved concerns about the appropriate number of trials per time-step, and due to the fact that windowing the messages together may cause valuable information about the order of the messages to be lost.

While the general multinomial distribution may not be viable, by allowing only a single trial at time t ($n_t = 1$) the **categorical distribution** is used. This is beneficial, as a single log line would represent the current state of the telescope (not one-to-one though). This enables a higher degree of interpretability as no information about message ordering would be lost.

4.1.3 Working & natural parameters

The last aspect that needs to be attended to, before performing the maximum likelihood estimation (MLE) of parameter values, is that of the difference in **working** and **natural** parameters.

In Section 3.1.4, when talking about the basic problems of an HMM, it was noted that there were two ways to perform the MLE. Using the Baum-Welch (EM) algorithm, as proposed by Rabiner (1989). Or via the direct maximisation of the likelihood using a numerical optimiser, as described by Zucchini *et al.* (2016), which is the method preferred in this work.

It is preferred for two reasons: the wide availability of numerical optimisation packages and the fact that it only requires the forward probability to perform MLE, – within the context of HMMs this represents a significant ease of use–; another noted advantage is that a number of these packages contain functions that can use several optimisation algorithms, which can be straightforwardly interchanged.

The parameters, the elements of $\mathbf{\Gamma}$, $\boldsymbol{\delta}$, and the state-dependent probabilities $p_i(x)$ are subject to various linear constraints (e.g. non-negativity). Therefore, the esti-

mates of these parameters should also satisfy those constraints; it is a constrained maximisation problem, not an unconstrained one. However, Zucchini *et al.* (2016) note that depending on the implementation and data, performing a constrained optimisation can be slower. Thus, they chose to focus on an unconstrained implementation of the MLE.

They do this by transforming the constrained (natural) parameters, to unconstrained (working) parameters. The likelihood is then maximised with respect to the working parameters; following that the natural parameters are obtained by performing the inverse transformation. And while not part of their reasoning, the unconstrained problem makes it convenient to use the **R** (R Core Team, 2021) optimr (Nash, 2016), which is one of the aforementioned packages that can implement several unconstrained optimisation routines — individually or simultaneously.

In the case of this work, the relevant constraints are:

- the rows of the transition probability matrix $\mathbf{\Gamma}$ must sum to 1, with all its entries γ_{ij} being non-negative.
- the initial probability vector $\boldsymbol{\delta}$ must sum to 1; with non-negative entries.
- the vector(s) of the state-dependent probabilities must sum to 1, and all the entries $p_i(x)$ must be non-negative.

Zucchini *et al.* (2016) (pages 50−51) detail how to arrive at the transformations, also noting that there may be several appropriate transformations that can be applied. The expression for the natural parameter (an element of the t.p.m), in terms of the working parameter (τ) is as follows

$$\gamma_{ij} = \frac{\exp(\tau_{ij})}{1 + \sum_{k \neq i} \exp(\tau_{ik})}, \quad \text{for } i \neq j. \tag{4.1}$$

This allows one to solve for the working parameter and obtain an expression for it, in terms of the natural parameter

$$\tau_{ij} = \log\left(\frac{\gamma_{ij}}{1 - \sum_{k \neq i} \gamma_{ik}}\right) = \log(\gamma_{ij}/\gamma_{ii}), \quad \text{for } i \neq j. \tag{4.2}$$

Using these expressions allows one to perform the unconstrained MLE as a means of estimating the working parameters and then transform them into their constrained natural counterparts. The same transformation was applied to all the parameters.

4.1.4 Parameter estimation

In the previous chapter, it was mentioned that there were 3 different techniques that were used to perform the parameter estimation. As mentioned earlier in the chapter,

this work assumes the latent states are known. These states were normal/abnormal and those taken from Drzał *et al.* (2016).

The first was the frequency method that was used by Boussemart *et al.* (2011) in their work on learning the state of an operator of an unmanned vehicle, in which they also made use of HMMs. As the states are known (see Table 4.2), it is possible to directly calculate $p_i(x) = \Pr(X_t = x|C_t = i)$ and $\gamma_{ij} = \Pr(C_{t+1} = j|C_t = i)$, as frequencies of appearances.

This was done by counting the number of appearances of a particular observing given a specific state then dividing by the total number of times any observing symbol appeared given that state, and by counting the number of times a state transitions into another state (could be a self-transition) then dividing by the total number of transitions the initial state performs respectively. Before dividing by total, a small offset was added to the appearance counts ($\mathbf{\Gamma}$ and $p_i(x)$). A detailed reason is given in Section 5.1.

The initial state probabilities $\boldsymbol{\delta}$ were similarly calculated, the appearances of each unique state is counted then divided by the total number of states that appeared in the training sequence.

Table 4.2: The state labels, assigned to the log-lines, that were used during the frequency and self-training estimation.

	Name of the states		
	9-State	3-State	2-State
State 1	Sleeping	Observing	Normal function
State 2	Waiting	Not observing	Abnormal function
State 3	FlatFielding	Malfunction	
State 4	Calibrating		
State 5	Observing		
State 6	Malfunction		
State 7	Exposure		
State 8	Shutdown		
State 9	Startup		

Next the parameters were estimated via a technique known as **self-training** (Tamposis *et al.*, 2018). Using the typical formulation of the algorithm, a classifier (in this case the frequency method) is trained using a small amount of the labelled training data. This is then used to classify (give a state label to) the remaining unlabelled data. Once the entire set is labelled, the model is then retrained using all the labelled data.

The difference in this work was that, a small amount of the labelled training data is used to re-label the remaining training data. Then all the labelled training data

is used to retrain the model. Numerous iterations of this procedure were performed and the training procedure was stopped when the log-likelihood of the training data converged.

The ML parameter estimation is slightly more involved, as it not only requires choosing an optimisation routine but also choosing a set of initial parameters for the optimiser. As a reminder Equation 3.13 is the likelihood function to be maximised. One of the default **R** (R Core Team, 2021) optimisation packages optimr, with the Nelder-Mead algorithm implemented, was used. The reason for using Nelder-Mead is that it does not need any derivatives to be calculated. Other algorithms, such as the conjugate-gradient (CG) or Broyden, Fletcher, Goldfarb and Shanno (BFGS) algorithms, require a function for the gradient to be given for the optimisation to be performed efficiently and accurately. As for the choice of initial parameters, the final estimated parameters from the frequency method was used.

4.1.5 Parameter simulations

A simulated data set was created as a means of checking the behaviour of the estimated parameters.

Before a dataset can be simulated the initial state distribution ($\boldsymbol{\delta}$), the transition probability matrix ($\boldsymbol{\Gamma}$), and the state-dependent distribution $\mathbf{P}(x_t)$ must be populated.

Values for $\boldsymbol{\delta}$ and $\boldsymbol{\Gamma}$ were randomly chosen. However, the values chosen for $\boldsymbol{\delta}$ must add up to one and each row in $\boldsymbol{\Gamma}$ must add up to one. Then for each state in the model 560 random values were chosen and divided by the sum of all the values for that state, these were then used to populate the $\mathbf{P}(x_t)$.

Using $\boldsymbol{\Gamma}$, $\boldsymbol{\delta}$, and the $\mathbf{P}(x_t)$ a sequence of observing symbols can now be generated. An initial state is chosen by sampling from $\boldsymbol{\delta}$. The next state is then chosen by sampling from the row in $\boldsymbol{\Gamma}$ that corresponds with the initial state. This continues until a state sequence is arrived at that is the same length as the full original sequence (111359).

Then by sampling from the row in $\mathbf{P}(x_t)$ that corresponds to a given state in the sequence, one can simulate the observing sequence.

This simulated sequence is then used to estimate parameters in the same way as the original parameters.

4.2 Bootstrapped confidence intervals

Once the parameters have been estimated, it is natural to want to evaluate the estimates via a chosen measure. Frequently the point estimate of the parameter is used in conjunction with an approximated interval estimate, to provide an indication of the best guess and how far in error that guess might be (Efron and Tibshirani, 1993).

4.2.1 The parametric bootstrap

Efron and Tibshirani (1986) use the example of calculating the mean ($\overline{x}$) of a random sample of size n, taken from an unknown distribution F; then wanting to find out how accurate of an estimator $\overline{x}$ is of the true mean $\theta = E_F\{X\}$, as a way of demonstrating the bootstrap principle. Essentially they want to calculate the standard error of $\overline{x}$, its standard deviation.

They note that this standard error is given by a convenient formula,

$$\sigma(F) = [\mu_2(F)/n]^{\frac{1}{2}}, \tag{4.3}$$

although it cannot actually be used since the distribution F is unknown, therefore $\mu_2(F)$ is incalculable. However, Efron and Tibshirani (1986) describe two methods of getting around the unknown distribution F, although only one is relevant in the context of this work. That method is by replacing F in Equation 4.3 by the empirical distribution $\hat{F}$, with probability mass $1/n$ on the random sample $x_1, x_2, .., x_n$. This replacement of $\sigma(F)$ in Equation 4.3 with $\hat{\sigma}(\hat{F})$ is what they call the *bootstrap estimate*.

In most instances there is no neat functional form for calculating the standard error of an estimator, such as Equation 4.3.

Nevertheless, Efron and Tibshirani (1986) note there is a straightforward way to numerically evaluate $\hat{\sigma}$ by means of a Monte Carlo algorithm. This involves taking n independent draws from $\hat{F}$, called a *bootstrap sample*. Since $\hat{F}$ is the empirical distribution of the data, they point out that this is the same as a random sample of size n drawn *with replacement* from the original sample $\{x_1, x_2, ..., x_n\}$. They elaborate on the Monte Carlo algorithm, noting that after drawing a large number of independent bootstrap samples; calculating the desired estimator for each bootstrap sample, $\hat{\sigma}$ is just the sample standard deviation of the bootstrap estimator.

The notions of replacing the sample distribution F by the empirical distribution $\hat{F}$ and creating a bootstrap sample, by independently sampling $\hat{F}$ (randomly sampling the data), are the key points to take away from this example. What was

described in the example is what called a **non-parametric** bootstrap estimate of the standard error.

This differs from the **parametric** bootstrap used in this work which creates a bootstrap sample by drawing from a parametric estimate of F (in this case the HMM), rather than the empirical distribution (Efron and Tibshirani, 1993). This involves using the fitted HMM to generate a sequence the same length as the original. The parametric bootstrap samples may then be used to estimate the standard error as in the example. The parametric standard error or the parametric bootstrap samples themselves are what can then be used to calculate the desired approximate interval estimates.

4.2.2 Constructing percentile intervals via parametric bootstrap

While Efron and Tibshirani (1986) note that standard errors are crude measures of statistical accuracy, they do mention they can be useful by allowing approximate *confidence intervals* for the unknown parameters to be calculated. A confidence interval (CI) has a particular coverage probability (typically $1 - 2\alpha$), which implies that $100 \cdot (1 - 2\alpha)\%$ of the time a random CI constructed in this way will contain the true value of a parameter estimate (Efron and Tibshirani, 1993).

While in their paper Efron and Tibshirani (1986) demonstrate how to use the bootstrap standard error estimate to calculate an approximate standard normal confidence interval, they also describe several methods that they developed for constructing approximate confidence intervals which just make use of the bootstrap sample; notably the **percentile method**, used by Zucchini *et al.* (2016) in their book, as well as in this work.

Algorithm 4.1 Bootstrap percentile intervals

Input: sequence of observations T, B number of bootstrap repetitions, $0 < \alpha < 1$ helps set percentile boundary

Output: $1 - 2\alpha$ percentile interval estimates for each parameter

1: Fit the HMM i.e. calculate $\hat{\boldsymbol{\theta}}$, the estimate of all model parameters $\boldsymbol{\theta}$.
2: Use $\hat{\boldsymbol{\theta}}$ to generate $B \gg 1$ independent bootstrap samples
   of observations, each of length T.
3: Estimate $\hat{\boldsymbol{\theta}}^*$, the parameters for each of the B bootstrap samples
4: For each individual parameter list the B replications, $\hat{\theta}^*$, in ascending order
5: Let $\hat{\theta}_B^{*(\alpha)}$ be the $B \cdot \alpha$th value in the ordered list
6: Let $\hat{\theta}_B^{*(1-\alpha)}$ be the $B \cdot (1 - \alpha)$th value in the ordered list
7: The $1 - 2\alpha$ percentile interval for the parameter estimate $\hat{\theta}$ is then $[\hat{\theta}_B^{*(\alpha)}, \hat{\theta}_B^{*(1-\alpha)}]$

In their book Efron and Tibshirani (1993) have a discussion about how to use the percentiles of the bootstrap histogram define confidence limits.

They show that the $1 - 2\alpha$ *percentile interval* (see Algorithm 4.1) for each parameter estimate can, for a large enough number of bootstrap replications, be approximated by the interval enclosed by the $100 \cdot \alpha$th (that is the $B \cdot \alpha$th value) and $100 \cdot (1 - \alpha)$th empirical percentiles of an ordered list[1] of bootstrap estimates.

If the result of $B \cdot \alpha$ is not an integer, Efron and Tibshirani (1993) adopted the following convention. Let $k = (B + 1) \cdot \alpha$, then they use the interval of the kth largest and $(B + 1 - k)$th largest values obtained from an ordered list of the bootstrap parameter estimates.

The use of a percentile interval also provides a convenient way to avoid a pitfall associated with the use of a standard interval. If the underlying distribution of the bootstrap parameter estimates is not approximately standard normal, they would need to be mapped to a standard normal scale (via the use of an appropriate transformation) to calculate the standard interval and that interval must then be mapped back to the original scale. While Efron and Tibshirani (1993) show that the percentile method performs this transformation and mapping back to the original scale automatically.

4.3 Model selection & checking

As seen in Section 4.1.4, there are several different models, predicated on the number of hidden states that were investigated. This inevitably leads to a discussion how the appropriate model is chosen, and as with the parameter estimates, there is also a need for a method of evaluating how well the model as a whole is performing.

4.3.1 Model selection

Assuming that some sequence of observations $x_1, x_2, \ldots, x_T$ has been generated by the 'true' model f, the sequence is then fitted using models from differing approximating families $h_1 \in H_1$ and $h_2 \in H_2$. Model selection aims to choose the most appropriate.

Zucchini *et al.* (2016) (pages 97−100) outline two of the more popular approaches to model selection. The first selection criteria they detail is the Akaike information criterion (AIC), in which the value of an estimator of the difference between the true model and the families calculated (they choose the Kullback-Leibler divergence). The chosen model is then which ever estimator value is closer to the true model.

[1] Ordered in ascending fashion.

AIC criteria is given by:

$$AIC = -2\log(L) + 2p, \tag{4.4}$$

where the $\log(L)$ is the log-likelihood and p is the number of model parameters. The first term is a measure of the fit and decreases with increasing the number of states m, while the second term is a penalty term that increases with increasing m.

The second criteria Zucchini *et al.* (2016) detail is that of the Bayesian information criterion (BIC). In this method a prior probability that f belongs to a model is specified for each $h_1 \in H_1$ and $h_2 \in H_2$. Then the posterior probabilities that f belongs to either family, given the observed sequence, are calculated. The family that has the high posterior probability is then closer to the true model, this is summarised by the BIC criteria as follows:

$$BIC = -2\log(L) + p\log(T), \tag{4.5}$$

the $\log(L)$ and p are the same as for the AIC, with the T being the length of the observing sequence. When compared to the AIC, the BIC often favours models with fewer parameters, because the BIC penalty weighs more for $T > e^2$ which is often the case. The model that produces the lowest value of either the AIC or BIC (they need not agree, but often do) is the one that should be chosen.

4.3.2 Model checking

While it is all good and well that the 'best' model can be selected by some criterion, there still remains the question of whether the chosen model is indeed adequate. There needs to be an approach to assessing the general goodness of the fit and to identify outliers relative to the model — the latter of which is a subject of the next chapter.

Standard classification metrics

The use of the **precision**, **recall**, and **F1-score** measures, has been widely applied in multi-class machine learning classification problems (Sokolova and Lapalme, 2009).

A model's precision is the number of examples allocated to the positive class by the model divided by the number of examples allocated to that class by the model as a whole (i.e. $\frac{tp}{(tp+fp)}$)[2], and a model's recall is the number of examples allocated to the positive class by the model divided by the total number of examples that belong to that class in the data (i.e. $\frac{tp}{(tp+fn)}$). The F1-score is an aggregate of the precision

[2] tp = true positive, fp = false positive, tn = true negative, and fn = false negative.

and recall (with the choice to weigh them differently depending on importance), so that a single value can represent the performance of the model.

These metrics are often preferred over the accuracy ($\frac{tp+tn}{tp+tn+fp+fn}$) in many multi-class classification problems, due to an inherent imbalance of class representation in the training set (Grandini *et al.*, 2020). This is due to the accuracy hiding the effect of misclassifying classes with smaller representation, as the more prominent classes will have a more noticeable affect.

Table 4.3: This table shows the percentage of each class within the training set.

Name of the states			
9-State		3-State	2-State
Sleep = 2.4%	Malfnc = 27.0%	Obs = 45%	Norm = 73%
Wait = 10.2%	Exp = 10.8%	Not obs = 28%	Abnorm = 27%
FlatF= 2.3%	Shutd = 0.1%	Malfnc = 27%	
Calibr = 9.3%	Startup = 1.6%		
Obs = 36.2%			

Looking at the balance of classes in Table 4.3, it is noticeable for each model that there is an imbalance of classes within the training set. In the context of the MeerLICHT project as a whole, it is a better outcome for the models to predict a malfunction (or abnormal) state but the actual state is functioning. This means that false negatives are more tolerable than false positives, and thus a lower recall is more tolerable than a lower precision.

Another choice is how the metrics will be averaged, in a multi-class problem the usual averaging procedures are: *macro-averaging* in which the mean of a particular metric (calculated per individual class) is taken, or *micro-averaging* where the cumulative *tp, tn, fp, fn* values are found and then the metric is calculated Sokolova and Lapalme (2009). Macro-averaging treats all the classes equally and could result in an inaccurate F1-score, so the micro-averaging scheme was preferred.

When considering a state (class) C_i, the tp_i value (tp for class C_i) was incremented (by 1) when the predicted class was C_i and the actual class (from the test set) was C_i. The tn_i was incremented when the predicted class was any other than C_i and the actual class was any other than C_i. The fp_i was incremented when the predicted class was C_i but the actual class was not. Lastly, fn_i was incremented when the predicted class was not C_i but the actual class was.

All the per class *tp, tn, fp, fn* values were then summed up to check if it returned the total number of responses in the test set.

4.4 Implementing anomaly detection

This sections details the theoretical foundation for each of the anomaly detection methods.

4.4.1 Anomaly detection via changes in the entropy

The use of an HMM (or the variant HSMM) to perform anomaly detection is a task that has been the focus of several previous works, notably in Tan and Xi (2008); Xie and Tang (2012); Min and Shun-Zheng (2006). Each of these previous works implemented a similar method to perform the anomaly detection. They each calculated a value for the 'entropy' of the sequence, all using a similar definition

$$H = \ln(\Pr(o_1^T|\lambda))/T, \tag{4.6}$$

at each point in time as the sequence grew larger.

Here T is the length of the sequence at some point in time, and o_1^T is the sequence at that point in time. Their focus was to perform intrusion detection on the studied network, via the use of either network traffic loads or request types along the network. Using the established definition Equation 4.6 to obtain a value of entropy for 'normal' behaviour; the understanding that the entropy of 'abnormal' behaviour will be larger than that of normal behaviour. They then compared this value for the normal sequence of events (traffic or requests) against one from a suspect sequence of events. And if the value of the entropy was larger for the suspect sequence of events, they would be able to tell the traffic is indeed abnormal.

4.4.2 Anomaly detection via decoding

Decoding is one of Rabiner's basic problems (Rabiner, 1989). It is the ability to discover the best state (or state-sequence), using N observations and an m-state HMM, that belongs to an observation (or sequence of observations), known as local and global decoding respectively.

Of the two decoding implementations, global (entire state sequence) or local (individual states), the local decoding was chosen. The reason being that the results of either decoding scheme are very similar (Zucchini *et al.*, 2016), for the purposes of comparing against other methods local decoding will be more than sufficient, and it is simpler to implement. When defining local decoding as a conditional probability, Zucchini *et al.* (2016) use the following proposition (the proof of which is on page 68):

$$\alpha_t(i)\beta_t(i) = \Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)}, C_t = i). \tag{4.7}$$

Where $\alpha_t(i)$ is the i^{th} entry in the vector of forward probabilities (similarly with $\beta_t(i)$). Then using Equation 4.7, Zucchini *et al.* (2016) show that the conditional distribution of C_t, given the observations, for $i = 1, 2, \ldots, m$, is

$$\begin{aligned} \Pr(C_t = i \mid \mathbf{X}^{(T)} = \mathbf{x}^{(T)}) &= \frac{\Pr(C_t = i, \mathbf{X}^{(T)} = \mathbf{x}^{(T)})}{\Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)})} \\ &= \frac{\alpha_t(i)\beta_t(i)}{L_T}. \end{aligned} \tag{4.8}$$

Using Equation 4.8, the probabilities for each state are calculated and the one with the largest probability is then chosen as the given state for that instance in time.

As mentioned in Section 4.3.2, in this work the F1-scores were calculated using a micro-averaging scheme:

$$\text{F1-score} = \frac{(\beta^2 + 1)Precision_\mu Recall_\mu}{(\beta^2 \times Precision_\mu) + Recall_\mu}. \tag{4.9}$$

Here β is a positive real factor that is chosen to weigh the recall β times more than the precision. As this work considers the precision to more important, β is chosen to be < 1. The $Precision_\mu / Recall_\mu$ refer to:

$$Precision_\mu = \frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l}(tp_i + fp_i)}, \tag{4.10a}$$

$$Recall_\mu = \frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l}(tp_i + fn_i)}, \tag{4.10b}$$

which calculates the precision and recall within the micro-averaging scheme. The anomaly detection procedure is to just decode the entire test sequence, compare the results (F1-score, etc.) of decoding to the actual test set states.

4.4.3 Anomaly detection via predictions

This method of performing anomaly detection is very similar to decoding, but instead of assigning a state (or states) to an observation (or sequence) that has been observed, one uses all the observations observed until time T to predict the state that will occur at time $T + h$.

In their book Zucchini *et al.* (2016) offer a convenient expression to calculate the conditional distribution of C_{T+h}, given prior observations, for $i = 1, 2, \ldots, m$:

$$\Pr(C_{T+h} = i | \mathbf{X}^{(T)} = \mathbf{x}^{(T)}) = \boldsymbol{\alpha}_T \boldsymbol{\Gamma}^h \mathbf{e}_i' / L_T. \tag{4.11}$$

The proof of Equation 4.11 is in the Appendix A.8. In Equation 4.11 the quantity h is known as the 'horizon', and by changing its value one can predict the state h-steps

ahead. In this instance $h = 1$, the one-step-ahead prediction was made, and instead of using the entire test sequence all at once, a single test observation was added to the sequence T at each time-step. I.e. at $t = 1$ the sequence T has one observation, at $t = 2$ T has two observations, etc. The prediction was performed this way, as it was thought to more closely resemble an online scenario.

Chapter 5

Results and Discussion

Having detailed the foundational theoretical framework throughout the previous chapter, it is now time to present and discuss the results of using that framework, and an offline dataset, to address the research questions. This chapter is structured as follows.

Firstly in Section 5.1 the results of the model selection, parameter estimation, and simulated parameter estimation are presented. Next, Section 5.2 discusses the results of applying an HMM to the problem of anomaly detection.

5.1 Parameter estimates and model selection

As mentioned in Section 4.1.4 when the parameters for $\mathbf{\Gamma}$ and $\mathbf{P}(x_t)$ are estimated using the frequency of appearance, a small offset (10^{-9}) is added to that frequency of appearance value before dividing by the total. This was done to ensure that any estimation method that makes use of the transformation between natural and working parameters (self-train and ML estimates) converges, as the transformation used involves taking the logarithm of the natural parameter, and therefore this cannot be zero.

5.1.1 Parameter estimates with percentile intervals

After performing the parameter estimation one needs to check the estimated parameters. In this work a percentile interval, for each estimated parameter value, was calculated by performing a parametric bootstrap (Efron and Tibshirani, 1993). A more detailed account of the parameter and standard error estimation is given in Section 4.2. The confidence intervals give one an indication of the variability of the estimates, not the accuracy.

It should be noted at this point that having accurately estimated parameters does not guarantee that the model to which those parameters belong will perform well i.e. predicting system faults consistently and correctly.

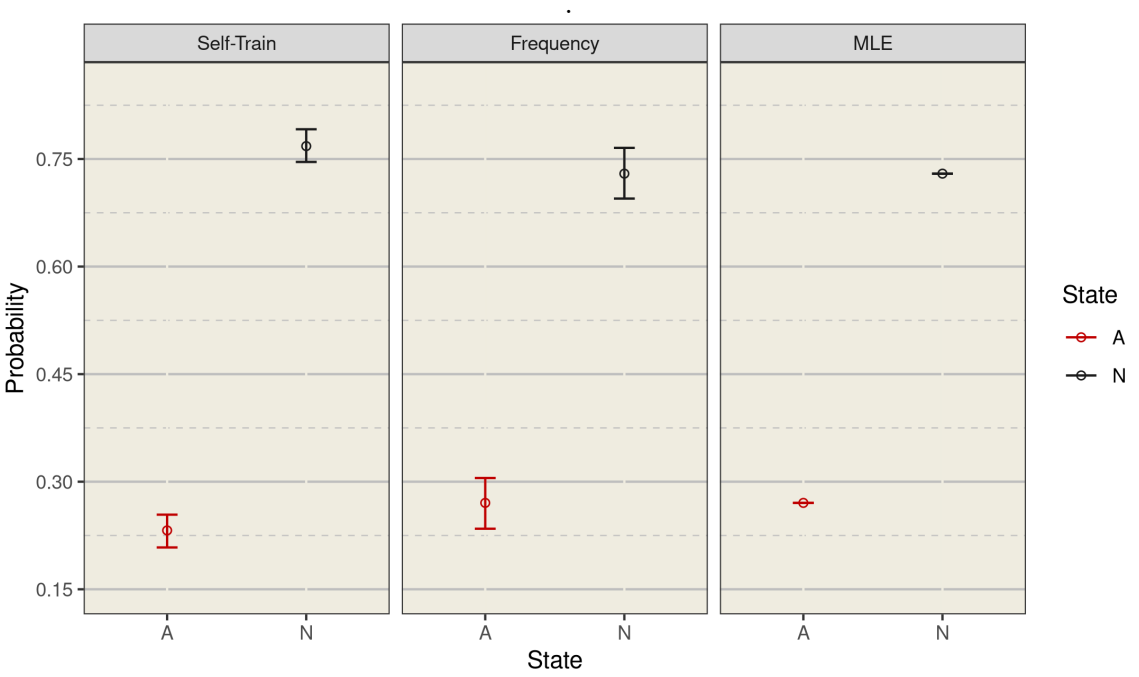

Figure 5.1: Estimates of the initial two-state probabilities with a 90% percentile interval. The left panel shows the self-train, the centre panel the frequency, and the right panel the MLE estimates. A is the abnormal state and N is the normal state.

The intervals in Figure 5.1 are 90% CIs, i.e. if the procedure to estimate the parameter and calculate an interval was repeated numerous times then 90% of those CIs would contain the true value (Chernick, 2008). Stated otherwise, one can be 90% confident that a particular interval is one of those that does contain the true value (Petty, 2012). Each of the intervals in Figure 5.1 is fairly narrow which suggests that one can be fairly confident that each estimate could be the exact value. With one being the most confident that the ML estimate could the true value of the parameter, as the MLE appears to have produced the least variable estimates. The MLE used the estimates generated from the frequency parameter estimation procedure as initial values for the optimr routine (see Section 4.1.4), and since their estimates agree so closely it appears that those initial values allow the likelihood to be maximised (more likely a local maximum, optimr does not perform global maximisation).

It is also unexpected that the self-training parameter estimates appear to be less variable than the frequency estimates (narrower percentile interval).

This is because the self-training estimate makes use of a frequency estimate of the parameter values, using a much smaller percentage of the training data ($\approx 20\%$)

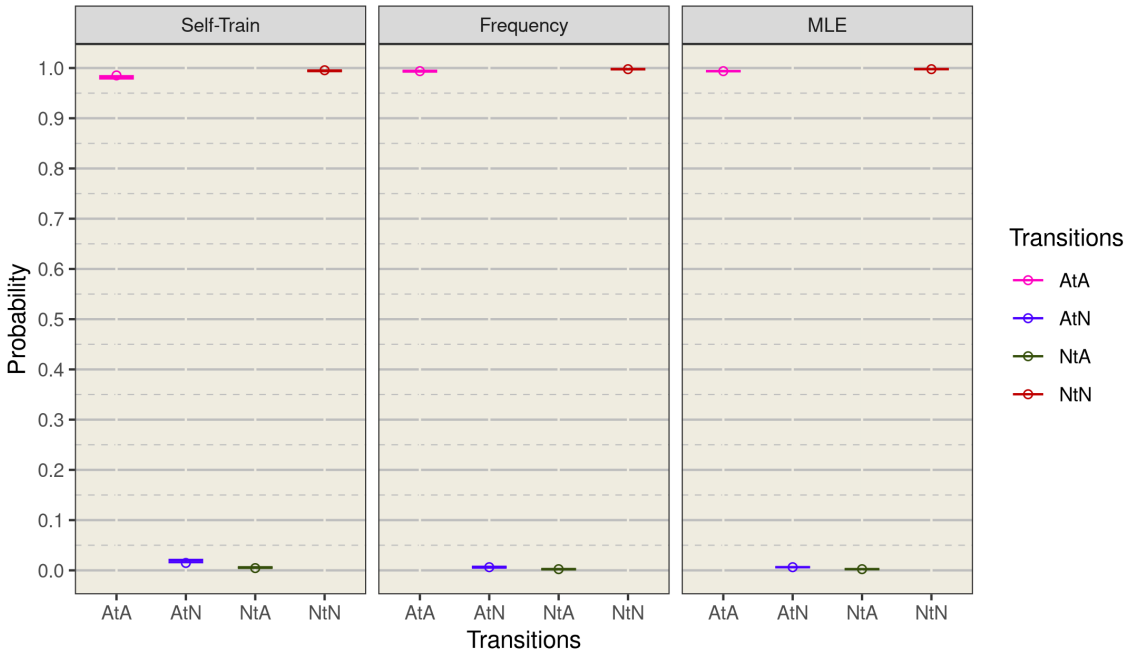

Figure 5.2: Estimates of all the two-state transition probabilities with a 90% percentile interval. The left panel shows the self-train, the centre panel the frequency, and the right panel the MLE estimates. The transitions are AtA = abnormal to abnormal, AtN = abnormal to normal, NtA = normal to abnormal, and NtN = normal to normal respectively.

initially. So the estimate of the parameter was expected to be either larger or smaller than the pure frequency estimate, as seen in Figure 5.1, but with larger intervals as there would be the same effect on the bootstrap samples.

Figure 5.2 shows the state transition probabilities, with the most prevalent transitions being the same-state transitions. It appears that each of the parameter estimation methods agree with each other closely. With the transitions into, and remaining in, state N being slightly more prevalent than those of state A for each estimation procedure. The only method that appears to differ slightly are the self-trained parameters, with certain estimates that are closer to the boundaries of the intervals (the state A to state A and state A to state N transitions) and which have wider intervals too. Since the transitions shown in Figure 5.2 are evenly split between the top and bottom ends of the y-axis scale, it makes it more difficult to judge the intervals.

As such the transitions shown in Figure 5.3 are plotted on a logarithmic scale making the evaluation easier, although it only shows those transitions close to 1 (the other half is shown in Figure A.3). Considering Figure 5.3 it is immediately apparent that the percentile intervals for the self-trained estimates are poor, they do not even contain the estimates and that means that one does not have any measure of confidence that these estimates contain the true value of the parameter.

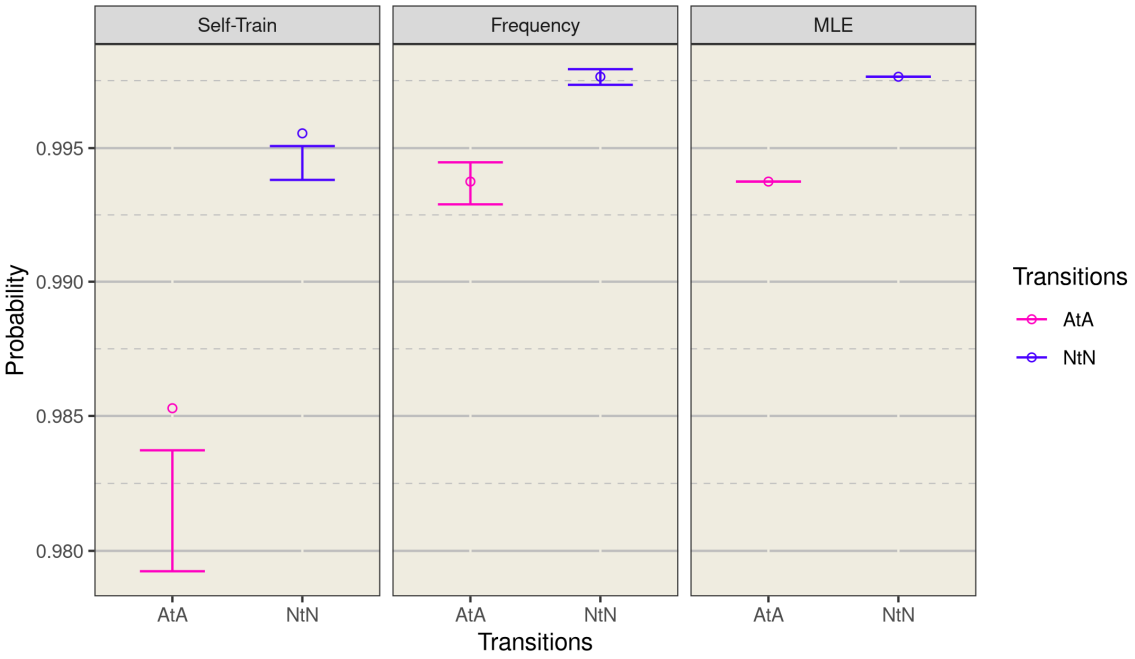

Figure 5.3: Estimates of the two-state transition probabilities that are > 0.9. Each estimate is shown with a 90% percentile interval and plotted on a logarithmic scale. These are the same state transitions. The transitions are AtA = abnormal to abnormal, and NtN = normal to normal respectively.

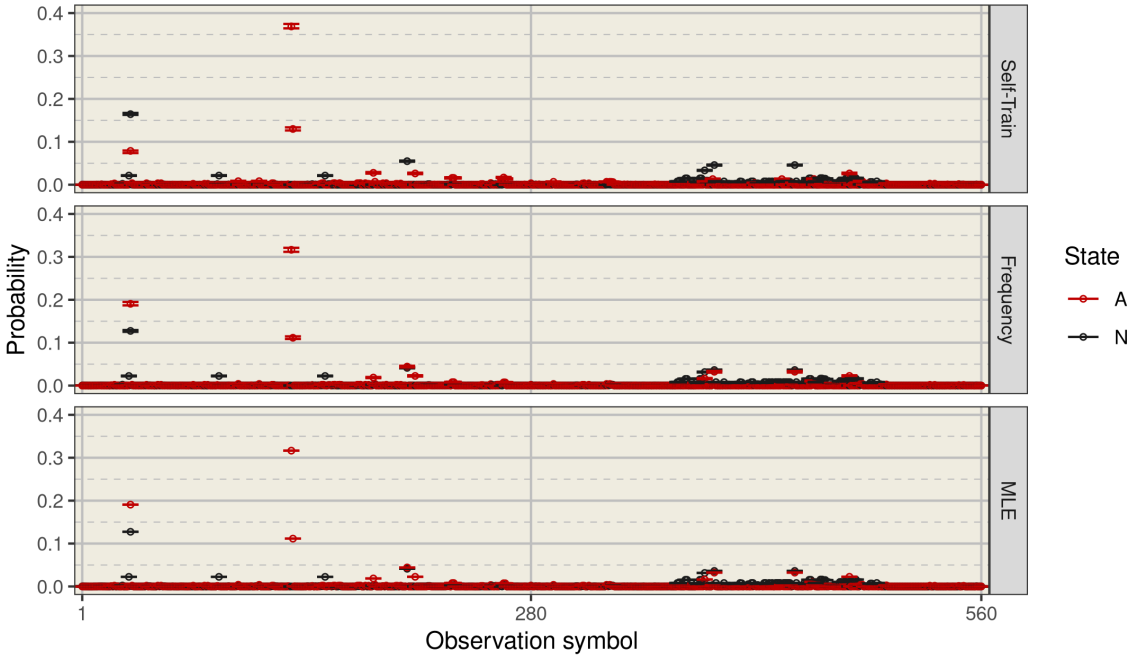

Figure 5.4: Estimate of the two-state state-dependent probabilities with a 90% percentile interval. In the top panel are the estimates from the self-train procedure, in the middle panel are those from the frequency, and the bottom panel are the estimates from the MLE. A is the abnormal state and N is the normal state.

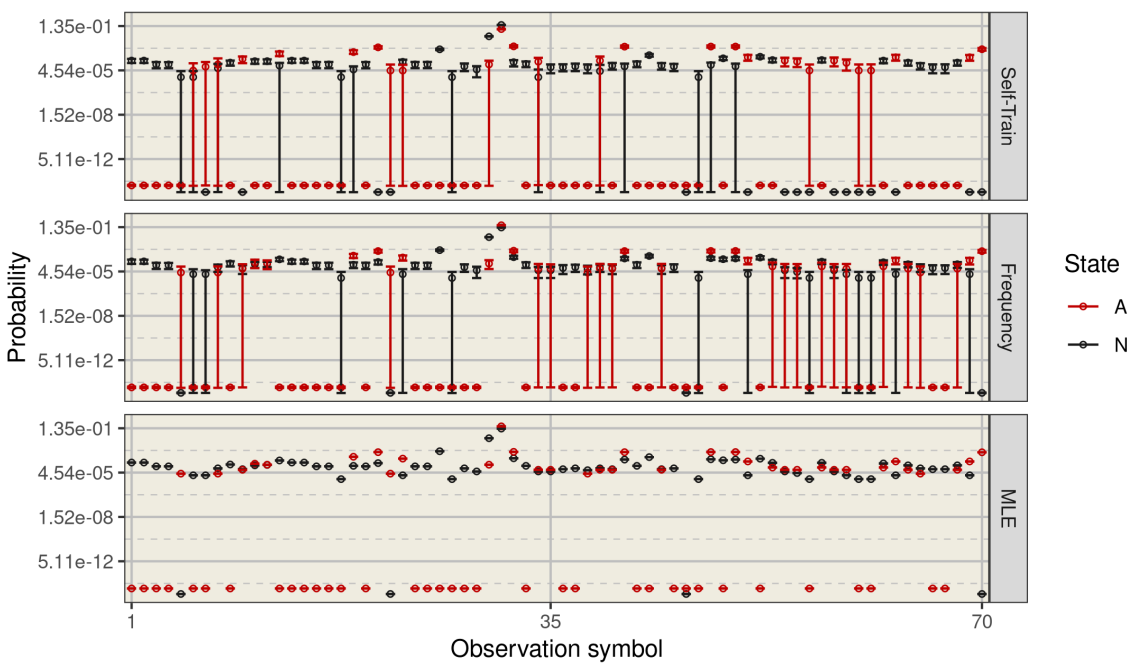

Figure 5.5: First 70 estimated two-state state-dependent probabilities with a 90% percentile interval. In the top panel are the estimates from the self-train procedure, in the middle panel are those from the frequency, and the bottom panel are the estimates from the MLE. A is the abnormal state and N is the normal state.

While Figure 5.4 may not be the clearest, looking at the first seventy two-state state-dependent probabilities plotted on a logarithmic scale, Figure 5.5, it is clear that there are a number of percentile intervals that have very small lower bound even though the estimate itself is orders of magnitude larger. These lower bounds are in fact an unintended consequence of creating percentile intervals for parameters estimated, in part or wholly, by using the frequency of appearance with a small offset added.

The 90% percentile intervals are constructed using the 50th and 950th elements from the ordered list of bootstrap parameter estimates, estimated using a bootstrap sequence (generated parametrically). If that bootstrapped sequence is one that does not feature a particular state-observation pair as often, this will result in an estimate (frequency estimate) of the state-dependent probability that is smaller. These smaller estimates give the lower bound of the interval, which are generally not as small as some of those seen in Figure 5.5. What causes these very small lower bounds (10^{-14}) is when a state-observation pair is not featured in the sequence at all, which results in the offset value (see Section 5.1) being divided by some number on the order of the length of the training sequence (10^5). If this happens at least 50 times, it results in these wider lower intervals.

While the percentile intervals are a good place to start when wanting to check

how well the parameter estimates capture the exact values, they are not infallible, and it is a good idea to check if the parameter and interval estimates are behaving appropriately. To that end, a simulation was performed using random known exact values to generate a sequence; that sequence was then used to check the behaviour of the parameter fitting and interval estimation.

The remaining parameter estimates are shown in the Appendix. The parameter estimates for the 3-state model are shown in Figures A.4 - A.9 and those for the 9-state model are shown in Figures A.10 - A.15.

5.1.2 Simulated parameter estimates

This section details the results of the parameter estimation, using a sequence simulated as detailed in Section 4.1.5.

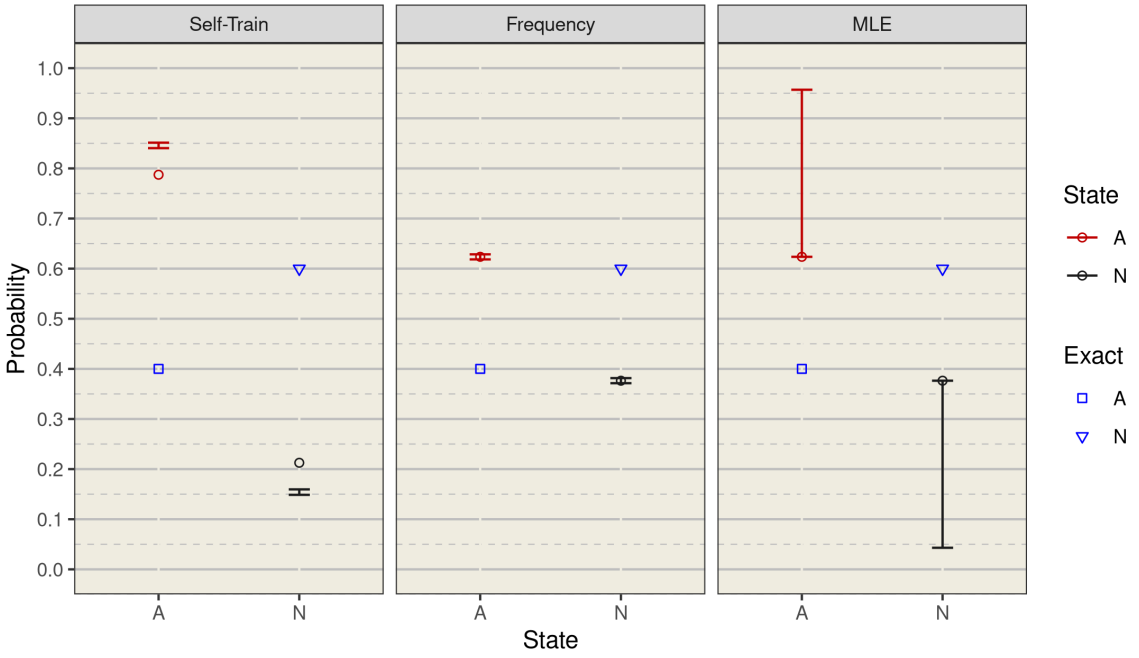

Figure 5.6: Simulation of the initial two-state probabilities with a 90% percentile interval. The left panel shows the self-train, the centre panel the frequency, and the right panel the MLE estimates. A is the abnormal state and N is the normal state.

The first thing that is apparent about the simulated parameter estimates in Figure 5.6, is that they are bad and the percentile intervals do not contain the exact value. This is actually unsurprising in this case, since the exact parameter values were chosen at random. This results in the initial state probabilities not agreeing with the transition probabilities.

What Figures 5.6 and 5.7 do suggest about the behaviour of the estimation procedures is — the self-training algorithm will tend to either over/underestimate the

exact parameter value and since the bootstrap essentially uses the parameter estimate as the 'exact' value, one has the same thing happening with the percentile intervals. This makes the behaviour in Figure 5.3 less surprising. Considering Figures 5.7 and 5.8 it is clear that the estimates made by the frequency and ML procedures agree well with the exact values in both figures. The self-training procedure at least shows better agreement with the exact values in Figure 5.8, than it has in Figures 5.6 and 5.7. This suggests that the estimation procedures may estimate the transition and state-dependent parameters more exactly. What is also visible in Figure 5.8 are the large lower bounds, that appear as a result of the offset added.

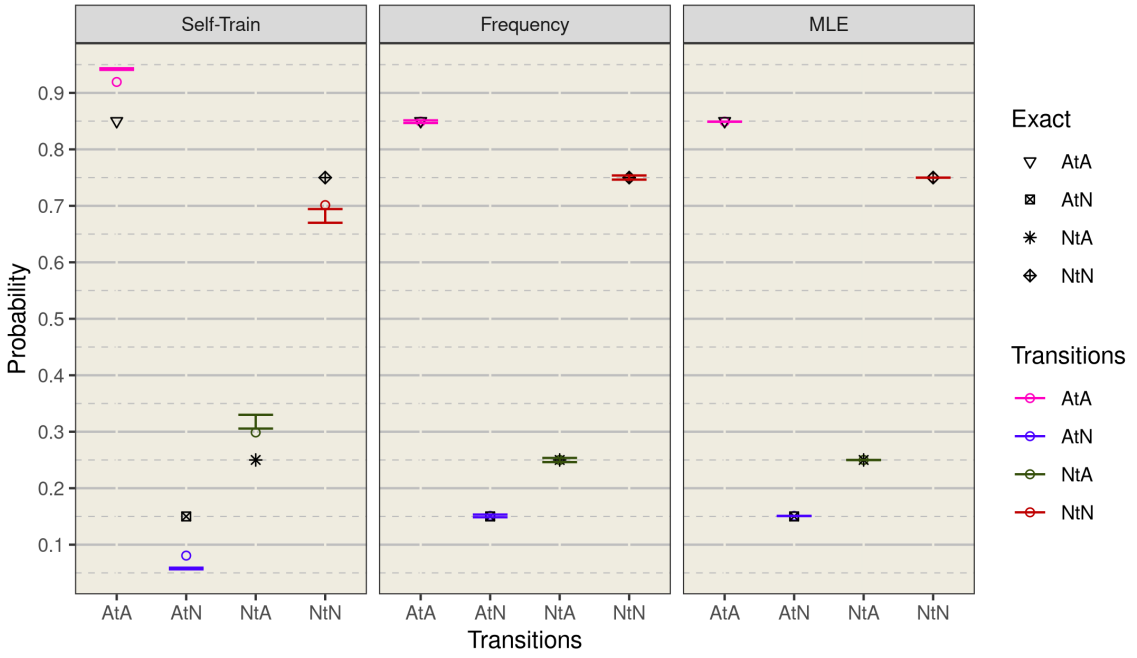

Figure 5.7: Simulation of the two-state transition probabilities with a 90% percentile interval. The left panel shows the self-train, the centre panel the frequency, and the right panel the MLE estimates. The transitions are AtA = abnormal to abnormal, AtN = abnormal to normal, NtA = normal to abnormal, and NtN = normal to normal respectively.

Figures 5.6 - 5.8 also seem to suggest that with a large enough labelled training set, performing a simple frequency estimate can produce estimates that are just as good as performing MLE. They also suggest that the percentile interval may not be a very good indicator of the accuracy of your estimate (close to the exact value), unless your estimate is fairly close to the exact value, but rather how similar your bootstrap estimates are to the value used to produce them.

5.1.3 Model selection criteria

As mentioned in Section 4.3.1 there needs to some criteria with which to determine the appropriate number of states the HMM should have, with the popular AIC and

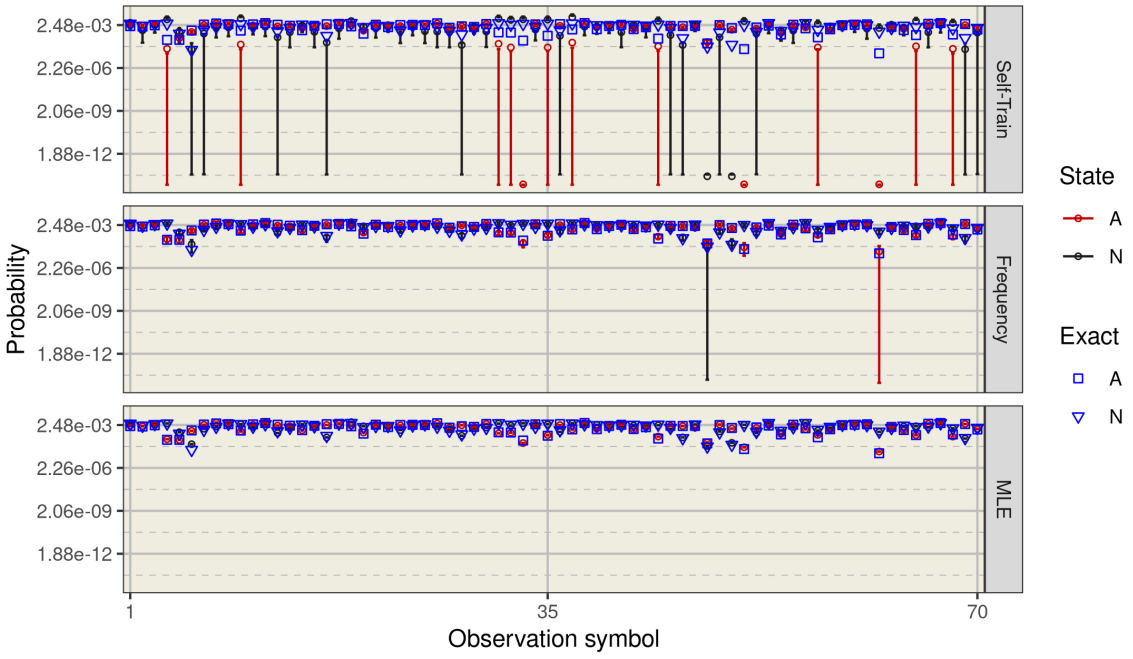

Figure 5.8: First 70 simulated two-state state-dependent probabilities with a 90% percentile interval. In the top panel are the estimates from the self-train procedure, in the middle panel are those from the frequency, and the bottom panel are the estimates from the MLE. A is the abnormal state and N is the normal state.

BIC criteria being chosen for that purpose. The lower the AIC/BIC value, the better.

Figure 5.9 shows values of both the AIC and BIC, plotted as a function of the number of states. It is important to remember when looking at the figure, that the criteria (AIC/BIC) are *relative* measures of fit, rather than absolute measures of fit. As such, one cannot say with absolute certainty that the nine-state model, with self-training parameter estimates, is a definitively better model than either of the other estimation procedure's nine-state models. They may all be equally poor at fitting the data, but the self-trained nine-state model may be the least poor.

Consider the discrepancy in the number of states when going from three states to nine states. While relative to the two and three-state models the nine-state model performs the best, however, there is no information about models with a number of states between three and nine. It could well be that a six or five-state model may outperform (i.e. have a lower criteria value) than the nine-state model.

Recalling that the value for either information criteria (Equations (4.4) or (4.5)) depends on the negative log-likelihood, this provides a reason as to why the AIC/BIC values are so large. The training set contains a total of 98119 observations, the longer the sequence is the smaller the likelihood will be and the larger the negative log-likelihood will be.

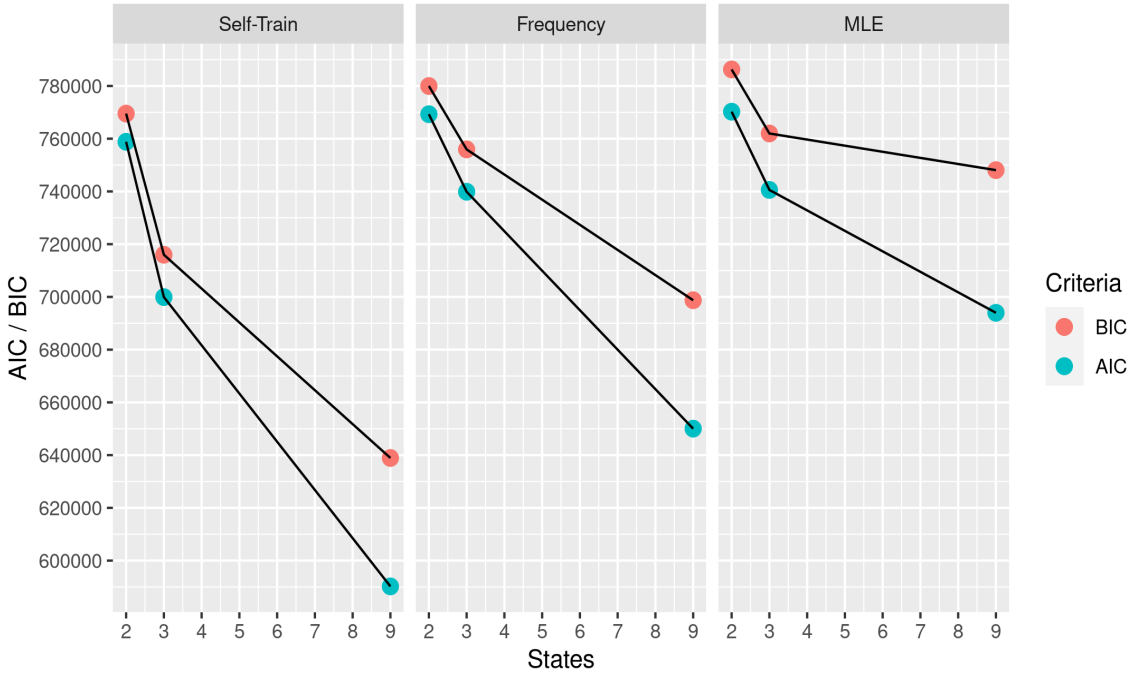

Figure 5.9: Comparison of the model selection criteria, the AIC/BIC values, calculated for each parameter estimation procedure and number of states. The left panel shows the self-train, the centre panel frequency, and the right panel the ML estimates of the AIC/BIC values respectively.

Figure 5.9 shows that the nine-state model, for each set of estimated parameters, may tentatively be labelled as the best choice of model — when compared to the other models using the same parameter estimation procedure. Hence when comparing all the nine-state models against each other, the nine-state self-training model appears to be the optimal choice overall.

However, one has to keep some considerations in mind:

- One must consider the number of parameters any of the procedures must estimate, relative to the number of observations in the training set. Each parameter estimation procedure needs to estimate 5111 parameters for the nine-state model, relative to the size of the training set this is a large number of parameters to estimate. This may lead to **overfitting** of the training set. Overfitting occurs when the model fits the training data too well and lacks the ability to generalise to other observing sequences (James *et al.* (2013)).

- Then pertaining specifically to the self-trained models. In the first part of the algorithm, a small portion of the training set is used to estimate initial parameter values (using frequency of appearance); then the algorithm labels the rest of the observations with a state using that initial estimate. This may misrepresent the prominence of certain states, state-transmissions, or state-

dependencies. For instance, the 'Observing' state (Table 4.2) may feature more prominently initially in the sequence. This will result in the initial and conditional probabilities of 'Observing' being relatively larger. In the calculation of the likelihood Equation (3.13) these relatively larger values will keep the likelihood larger, thus the negative log-likelihood and the AIC/BIC values will be smaller.

5.2 Anomaly detection

Due to the time sensitive nature (Bloemen *et al.*, 2016) of a number of the surveys MeerLICHT will be participating in, any down time the telescope experiences can represent a significant loss. Whether in terms of potential data collection, researcher time, or financial impact.

One method of keeping the telescope availability high (Sybilski *et al.*, 2014), that may be used in tandem with fault tolerant software or redundant hardware components, is to continuously monitor the 'health' (state) of the observatory and to counter any failures identified by the monitoring software.

The need to be able to monitor the state of the telescope is what forms the basis of the first research question developing a framework for fault detection within the MeerLICHT telescope. The intention is to make use of the telescope's abundant (and freely available) operational logs as the source of the raw data, as opposed to introducing monitoring tools for specific metrics (see Chapter 2, Section 2.2.1).

5.2.1 Anomaly detection within the context of HMMs

Observing changes in the entropy

Following the authors (Tan and Xi, 2008; Xie and Tang, 2012; Min and Shun-Zheng, 2006), the first attempt at using the information provided by the HMM to detect anomalies was to plot the average negative log-likelihood (entropy) value per time-step, observing the fluctuation in the value, and to see if that value exceeds a threshold value. This threshold value is calculated by taking sequence of observations (≈ 56000) that represented normal operation. An entropy value was then calculated, for each estimation procedure and number of states combination, with the average of estimation procedures for a particular number of states then being used.

The expectation in attempting to use the negative log-likelihood in this way, was that the negative entropy value would spike above the threshold when an anomaly state was included in the likelihood calculation and then decrease rapidly to a value

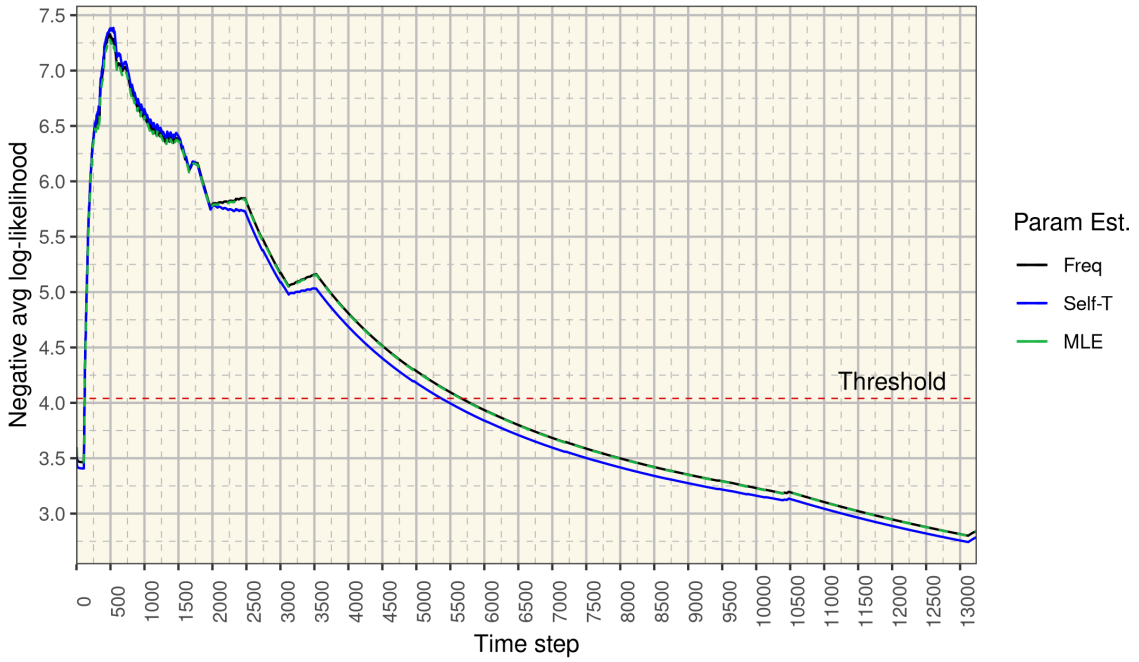

Figure 5.10: Comparison of the two-state model's negative entropy. The MLE entropy curve overlaps with the frequency entropy curve. The red threshold line indicates that if an entropy value is larger than it, the system should be in an anomalous state.

below the threshold. This is not the case. Figure 5.10 depicts that, before there have been 500 time steps, the negative entropy spikes over the threshold value, meaning the system is in an anomaly state, and then stays there until around step 5500.

Even when comparing what is depicted in Figure 5.10, with the actual distribution of states in the test set there is not that much agreement as to which states should be labelled as anomalies. One example is the time approximately between 3500 and 10500, in which there is a gradual decrease in the negative entropy until it is below the threshold that entire time the system is in the anomaly state. However, that is not clear using Figure 5.10 as using a static threshold appears to disadvantage a correct interpretation of what the current state is; this leads to a lack of correct understanding as to how the negative entropy would react to an anomalous state.

The remaining negative log-likelihoods are shown in the Appendix. The 3-state negative log-likelihood is shown in Figure A.16 and the 9-state negative log-likelihood is shown in Figure A.17.

Decoding the entire test sequence

Here the process of using decoding to perform anomaly detection is detailed.

The aim is to just decode the entire test sequence, compare the results of decoding to the actual test set states, and check how well the decoding performed by using the standard classification metrics discussed in Section 4.3.2.

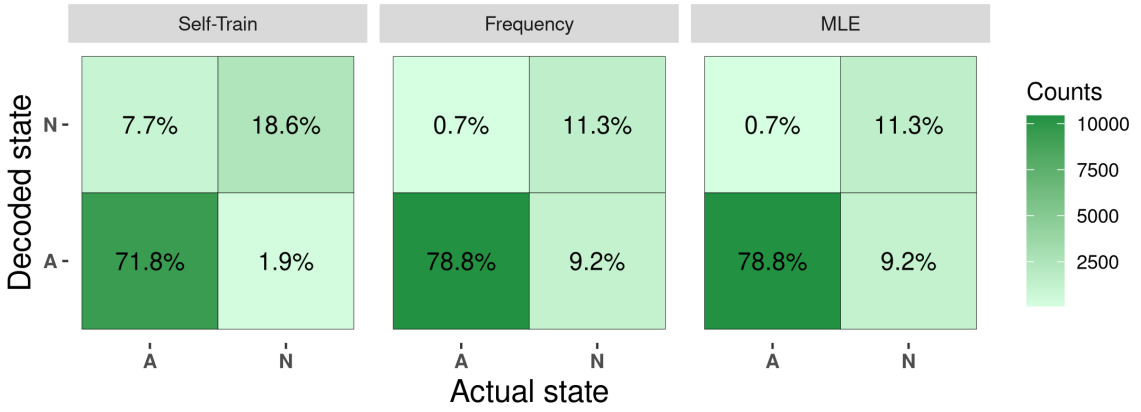

Figure 5.11: Two-state decoding confusion matrix. The true negative percentage is in the bottom left, with the true positive percentage being in the top right of each confusion matrix. The left panel is the self-train model, the centre is the frequency, and the right is the MLE.

Each model's decoding was assessed by comparing the test state sequence, to the individual state sequences the models decoded. The comparisons were summarised as confusion matrices, and the two state models are shown in Figure 5.11. In the context of the project it is not only important that anomalies be identified correctly (i.e. high % true negatives), but also for there to be a fewer anomalies labelled as normal (i.e. low % false positives).

In Figure 5.11, the percentages shown in each block refer to the classifications as a fraction of the total test sequence length. Visually inspecting the confusion matrices for the models, it is clear to see that all three models perform fairly well when it comes to correctly classifying true negatives. The frequency and MLE models performed slightly better (78.8%), than the self-train model which did worse (71.1%). While the self-train model did perform worse than the other two when classifying true negatives, it did much better with the true positives.

The remaining decoding confusion matrices are shown in the Appendix. The 3-state decoding confusion matrix is shown in Figure A.18 and the 9-state decoding confusion matrix is shown in Figure A.19.

For the different training regimes, Table 5.1 shows the $Precision_{\mu}$, $Recall_{\mu}$, and F1-score calculated for each of the models.

Table 5.1: Micro-averaged values for the decoded anomaly detection. The orange is frequency estimation, purple is MLE, and the green is self-train estimation.

	2-state			3-state			9-state		
Precision	0.94	0.94	0.71	0.90	0.88	0.87	0.86	0.86	0.70
Recall	0.55	0.55	0.91	0.90	0.88	0.87	0.86	0.86	0.70
F1-score	0.82	0.82	0.74	0.90	0.88	0.87	0.86	0.86	0.70

In the context of this work each model's precision is more important than the recall, and from the summary provided in Table 5.1, it is clear that for each choice of states the individual parameter estimation procedures perform fairly well. With the self-train model consistently having the lowest precision, across all three state choices, of the different estimation procedures.

Considering the F1-scores of each state model, the regimes that are able to decode the test sequences better are the frequency and ML regimes. This indicates that these estimation procedures may perform better when detecting anomalies. It is prudent at this time, however, to again consider Figure 5.9. This is because the results of Table 5.1 directly contradict the conclusion that may have been drawn when looking at Figure 5.9, that being that the best choice of model may have been the nine state self-train model. This is not surprising since it was noted that the self-training procedure used may have misrepresented its position as the prime candidate when considering the AIC/BIC model selection criteria alone.

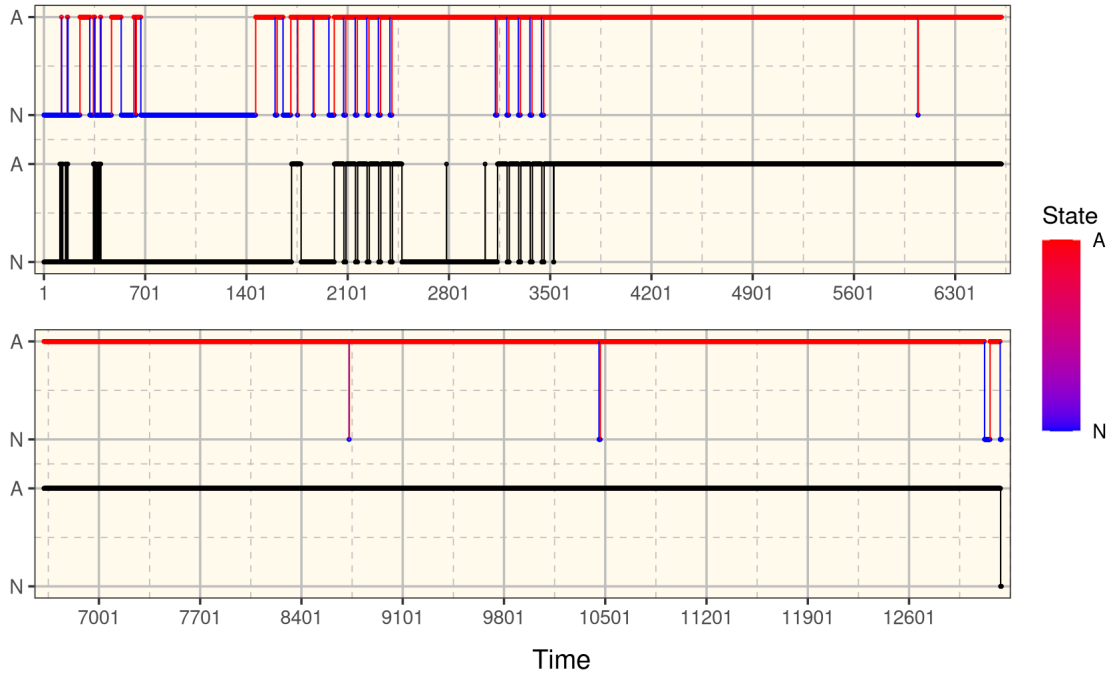

Figure 5.12: Two-state anomaly detection performed using decoding. The parameters used were the frequency estimates. A is the abnormal state and N is the normal state. The black plot represents the actual state sequence.

Figures 5.12 - 5.14 offer one possible suggestion for a dashboard, displaying the current state the system is in, that may be used for continuous monitoring. Looking at Figures 5.12 - 5.14 offers a quick visual confirmation of the precision and recall values in Table 5.1. For instance when looking at the frequency and MLE plots, it is clear that there are fewer instances when theses models labelled something as Normal (blue).

However, those that were labelled Normal were actually Normal. This implies that the precision of those models is high, but they mislabelled numerous instances when the state was Normal implying a low recall (agreeing with Table 5.1).

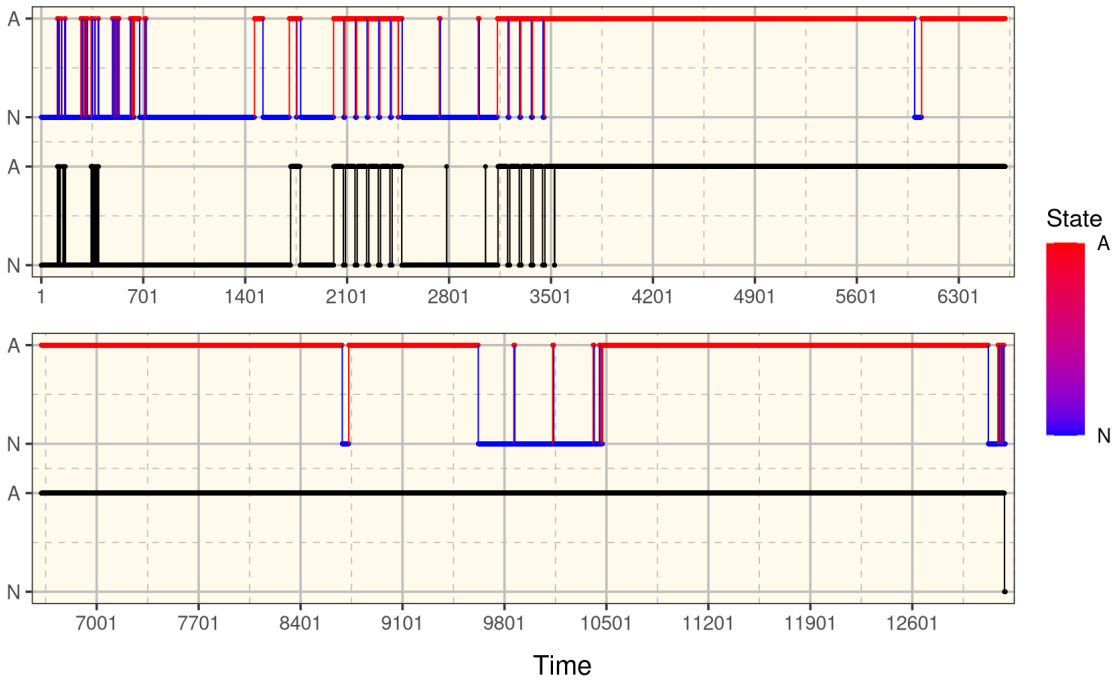

Figure 5.13: Two-state anomaly detection performed using decoding. The parameters used were the self-train estimates. A is the abnormal state and N is the normal state. The black plot represents the actual state sequence.

The remaining step-plots, showing the anomaly detection for decoding, are given in the Appendix. The step-plot for the 3-state anomaly detection via decoding is shown in Figures A.20 - A.22, and the step-plot for the 9-state anomaly detection via decoding is shown in Figures A.23 - A.25.

Predictions based on previous observations

Comparing Figure 5.11 to Figure 5.15, one can see that the former outperformed the latter when predicting true negatives (when considering the same model in each). While only the frequency and MLE models in Figure 5.15 appear to have outperformed their counterparts in Figure 5.11 when predicting true positives, as it appears the self-train model in Figure 5.15 was outperformed in both metrics.

Considering the false positives, which are more important in the context of this work than false negatives, all the models in Figure 5.15 had higher false positives than those in Figure 5.11. So when looking at the false positives, the 'predictions' from decoding appear as they may be better than those using Equation 4.11 to predict the states.

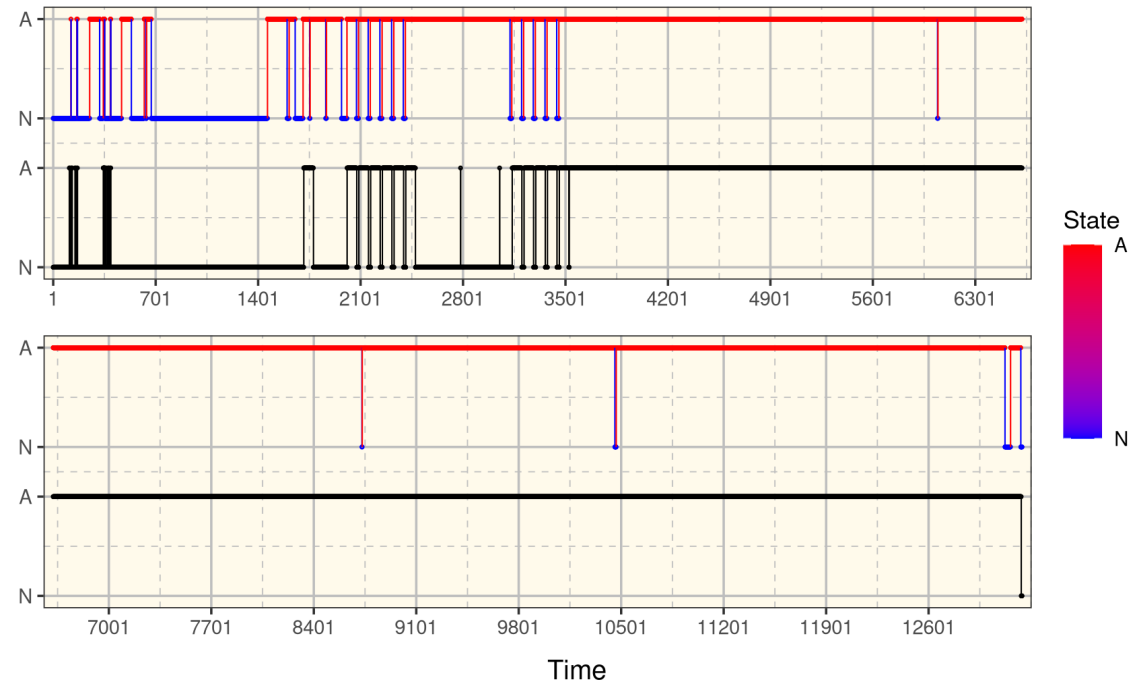

Figure 5.14: Two-state anomaly detection performed using decoding. The parameters used were the ML estimates. A is the abnormal state and N is the normal state. The black plot represents the actual state sequence.

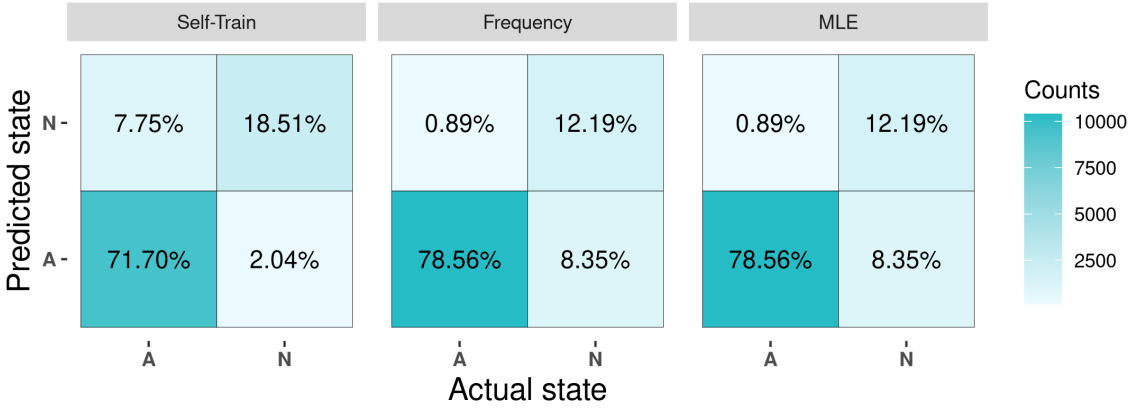

Figure 5.15: Two-state prediction confusion matrix. The true negative percentage is in the bottom left, with the true positive percentage being in the top right of each confusion matrix. The left panel is the self-train model, the centre is the frequency, and the right is the MLE.

The remaining one-step-ahead confusion matrices are shown in the Appendix. The 3-state one-step-ahead confusion matrix is shown in Figure A.26 and the 9-state one-step-ahead confusion matrix is shown in Figure A.27.

Looking at Table 5.2 one can see that the assumption that the former decoding model may perform better due to having fewer false positives was not wholly correct. Using Equation 4.11 appears to either produce an equal (or better) F1-score

Table 5.2: Micro-averaged values for the one-step ahead prediction. The orange is frequency estimation, purple is MLE, and the green is self-train estimation.

	2-state			3-state			9-state		
Precision	0.91	0.91	0.90	0.90	0.90	0.86	0.86	0.86	0.69
Recall	0.91	0.91	0.90	0.90	0.90	0.86	0.86	0.86	0.69
F1-score	0.91	0.91	0.90	0.90	0.90	0.86	0.86	0.86	0.69

for all the models using the frequency and ML parameter estimates. However, Equation 4.11 is out performed for the three and nine-state models using the self-train estimates.

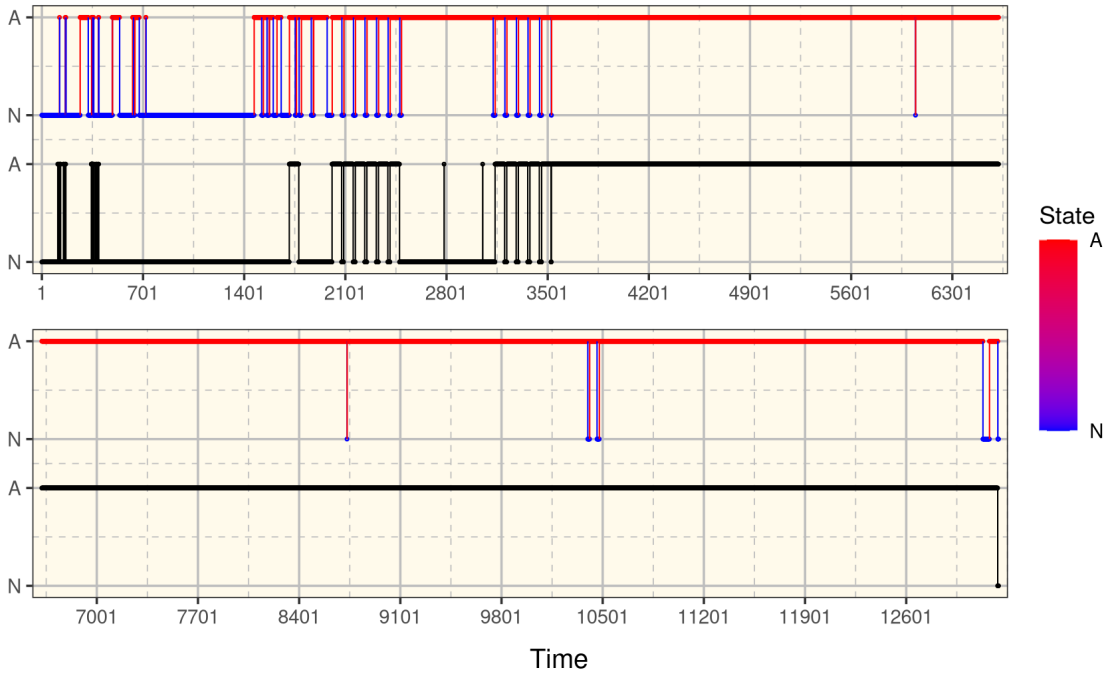

Figure 5.16: Two-state anomaly detection performed using the one-step-ahead prediction. The parameters used were the frequency estimates. A is the abnormal state and N is the normal state. The black plot represents the actual state sequence.

Considering Figure 5.16, it is clear that it is very similar to Figure 5.12. The decoding and one-step-ahead prediction have similar Precision values (decoding had a slightly higher value). However, their Recall values were vastly different. This becomes apparent when visually inspecting the time period between 350 and 1700 in both figures. In Figure 5.16 there were more times during that time period that were correctly classified as Normal, and this would increase the true positives and reduce the false negatives (as seen in Figure 5.15).

Table 5.2 indicates that the two-state self-train model has a better Precision when doing the one-step-ahead prediction, as opposed to the decoding.

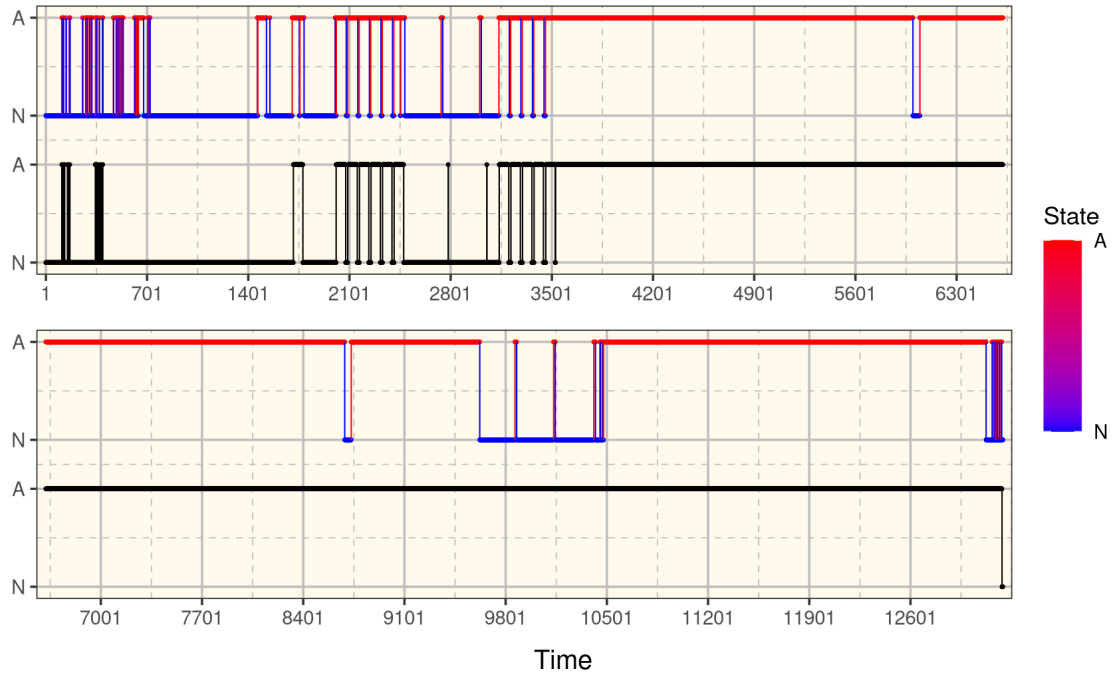

Figure 5.17: Two-state anomaly detection performed using the one-step-ahead prediction. The parameters used were the self-train estimates. A is the abnormal state and N is the normal state. The black plot represents the actual state sequence.

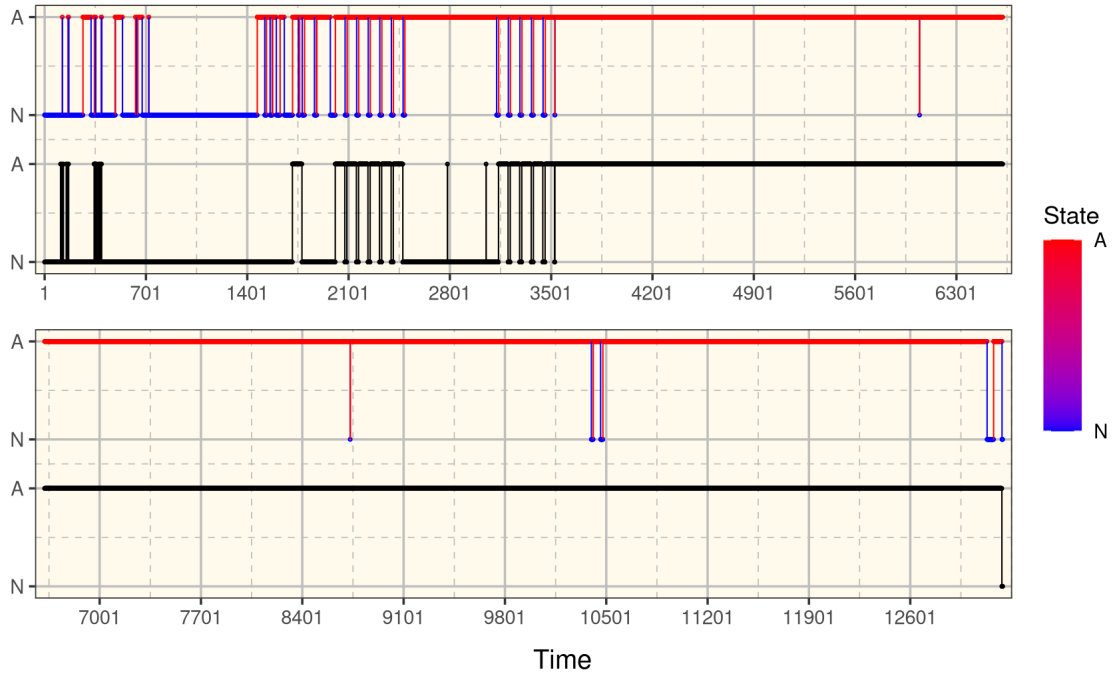

Figure 5.18: Two-state anomaly detection performed using the one-step-ahead prediction. The parameters used were the ML estimates. A is the abnormal state and N is the normal state. The black plot represents the actual state sequence.

However, it is not as simple as in the case of the frequency two-state model to indicate a region on Figure 5.17 that shows why the Precision is better than previously. The only region to indicate would be towards the end of the test sequence, between time-steps 12900 and 13239, Figure 5.17 has fewer false positives and this would contribute to a higher Precision.

Inspecting Figure 5.18, it too has a predicted Normal state close to 10500 similar to Figure 5.16. There are two clear regions in Figure 5.18 that would lead it to having a higher Recall that what was shown in Figure 5.14, both between 350 and 1700 time-steps. The first is around time-step 600, and the second around 1450, there are fewer false negatives. For the former this results in a narrower bar and the latter there are numerous up and downs between the two states.

The remaining step-plots, showing the anomaly detection for one-step-ahead prediction, are given in the Appendix. The step-plot for the 3-state anomaly detection via one-step-ahead prediction is shown in Figures A.28 - A.30, and the step-plot for the 9-state anomaly detection via one-step-ahead prediction is shown in Figures A.31 - A.33

Chapter 6

Conclusions

This work has presented the approach taken to address the problems of performing anomalous fault detection on the MeerLICHT telescope within an offline setting. One of MeerLICHT's aims is to participate in time-sensitive surveys (Bloemen *et al.*, 2016), as such the telescope is expected to have a high system availability and to be able to repeatedly take numerous exposures per night. This implies that any system downtime or delays experienced can lead to catastrophic loses for any research team making use of the telescope. To that end the telescope's log files, records of the telescope's operation, were investigated as source of insight into diagnosing (predicting) any faults the telescope is experiencing (may experience).

While making use of a telescope's log files to derive a model of the telescope's behaviour is not new (Pettinato *et al.*, 2019), this work made novel use of the Hidden Markov Model (HMM) to perform the task. The HMM's parameters were estimated using several methods (including performing a MLE) with varying success, often with the best estimations (narrowest confidence intervals) coming from a vanilla frequency estimate of the parameter values or the MLE of the parameters (often they were the same) (Section 5.1). The HMM was successfully used to perform anomaly detection (Section 5.2).

6.1 Findings

- After performing three separate parameter estimations, a frequency estimate; using the self-train algorithm; performing a maximum likelihood estimate (MLE); comparing the parameter estimates to a simulated data set it was found the frequency and ML estimates were the most precise (least variable).
- While the parameter estimates for the MLE and frequency methods were precise, there is no indication as to how closely they resemble the actual values

or not. However, since the F1-scores for all the most of the models are close to (or above) 70 it is reasonable to assume they are accurately resemble the actual values — although one cannot be certain.

- The reason that the frequency and MLE parameter estimates are similar is a result of the states being known. This results in the frequency estimates being the conditional MLE.
- The AIC/BIC scores were not an accurate choice for use as a model selector (combination of parameter estimation and number of states), as the model with the lowest AIC/BIC score self-train nine-state performed the worst (lowest F1-score) in both the decoding and prediction anomaly detection.
- Using the HMM's to predict the next state, given an observation, is better at performing anomaly detection than just performing a straight decoding of the state sequence.

6.2 Limitations

The biggest limitation in this work, was the fact it was done without knowing the full library of observable symbols i.e. all the different types of log messages. The only log messages that were considered were those in the dataset. However, it would be incorrect to assume that those that were used formed an exhaustive set of all the possible log messages the TCS could produce. As such, if any log messages are given to the framework developed in this work it would break. Another consequence of the symbol limitation, is that it limits this particular implementation to being offline only. Without knowledge of other unknown symbols, at the very least it makes using the logs to perform online continuous monitoring very difficult.

Apart from not knowing all the unique log messages, not all of the hub's log files were considered, only the Hardware Manager's logs, there could be information in the other logs that may be useful in improving the detection framework.

The dataset used in this work is from the early days of the telescope operation, and had more regularly appearing faults, however now the system is far more stable, with fewer errors.

6.3 Future work

In future it would be a good idea to try and include some of the industry standard logging infrastructure like ELK stack Elastic (2020), to make managing the logs and

including the log files from other domains, like the Sensor hub domain, easier.

Another interesting future investigation to be conducted is how HMM's could aid in the diagnosing of system delays. Any delay experienced by the telescope during operation can result in fewer observations being made in an evening, which represents an obvious loss of potential discoveries. HMM have an implied exponential state duration density (Rabiner, 1989), that can be calculated fairly easily. However, this may not accurately represent the duration spent in any particular state.

So it may also be advantageous to investigate an application of the Hidden Semi-Markov Model (HSMM) (Yu, 2010), which is an extension of the regular HMM that allows for any state duration density (not just an exponential distribution).

Some more investigation into the state-dependent distributions, and what observing symbols are more prevalent in what states could prove useful.

Appendix A

A.1 Types of variable phenomena

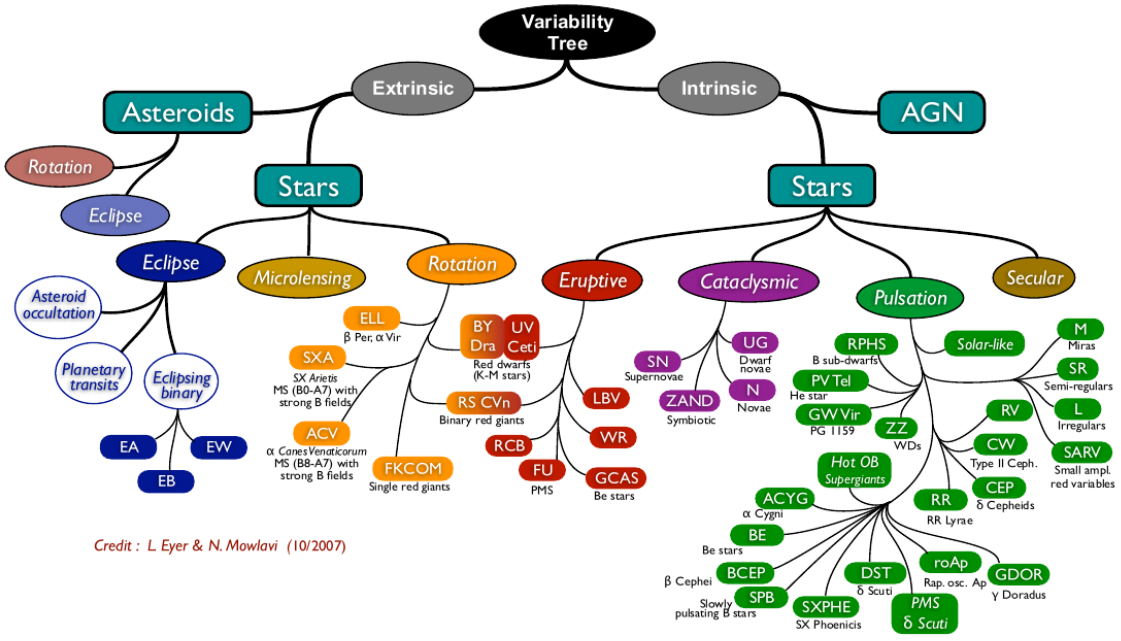

Figure A.1: The many different types of variable phenomena, grouped by the physical characteristics of the source (Ayer and Mowlavi, 2008).

A.2 Software Hierarchy

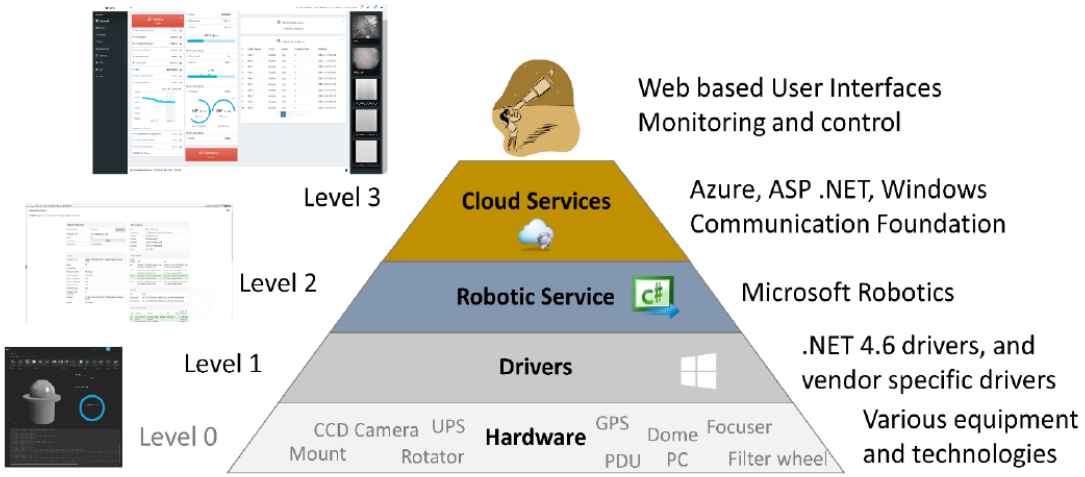

Figure A.2: Conceptual representation of the software that builds ABOT as a whole (Level 0 is typically independent of Sybilla Technologies). On the right are typical technologies used, and the left is a screenshot of typical user interface (Drzał *et al.*, 2016).

A.3 Proof of Equation 3.5

This entry into the appendix means to offer a formal proof of (3.6).

Proof. Let $t, u \in \mathbb{R}$. So,

$$\begin{aligned}
\vec{\delta}(t)\mathbf{\Gamma} &= \left(\Pr\left(C_t = 1\right), \ldots, \Pr\left(C_t = m\right)\right) \begin{pmatrix} \Pr(C_{t+1}=1|C_t=1) & \cdots & \Pr(C_{t+1}=m|C_t=1) \\ \vdots & \ddots & \vdots \\ \Pr(C_{t+1}=1|C_t=m) & \cdots & \Pr(C_{t+1}=1|C_t=m) \end{pmatrix} \\
&= \left(\sum_{i=1}^{m} \Pr\left(C_t = i\right) \Pr\left(C_{t+1} = 1 \mid C_t = i\right), \ldots\right) \\
&= \left(\sum_{i=1}^{m} \Pr\left(C_t = i \cap C_{t+1} = 1\right), \ldots\right) && \text{(Def. Conditional probability)} \\
&= \left(\Pr\left(C_{t+1} = 1\right), \ldots, \Pr(C_{t+1} = m)\right) && \text{(Def. Marginal probability)} \\
&= \vec{\delta}(t+1)
\end{aligned}$$

□

A.4 Additional parameter estimates

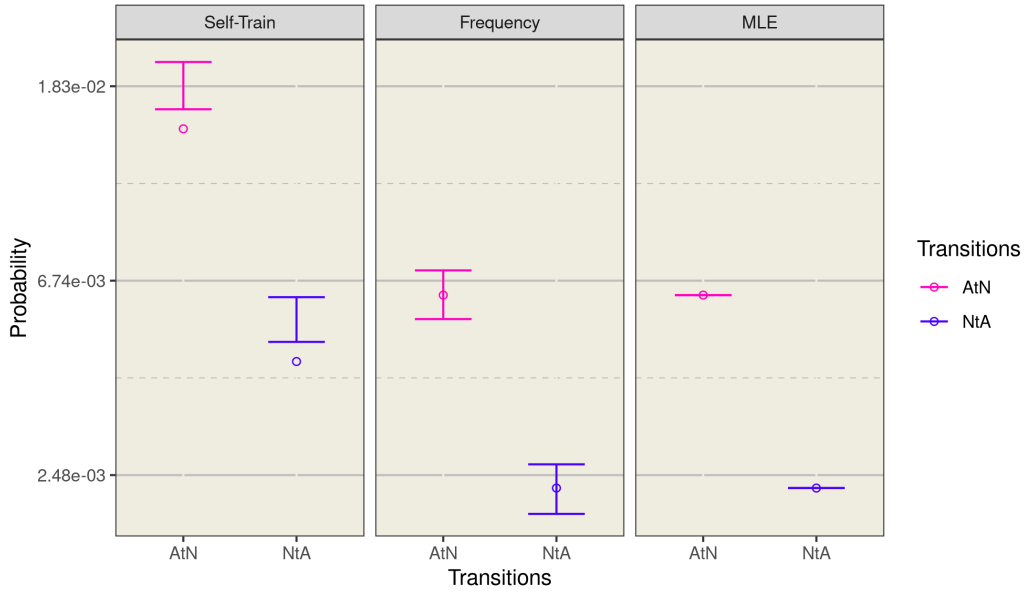

Figure A.3: Estimates of the two-state transition probabilities that are < 0.1. Each estimate is shown with a 90% percentile interval and plotted on a logarithmic scale. The transitions are NtA = normal to abnormal, and NtA = normal to abnormal respectively.

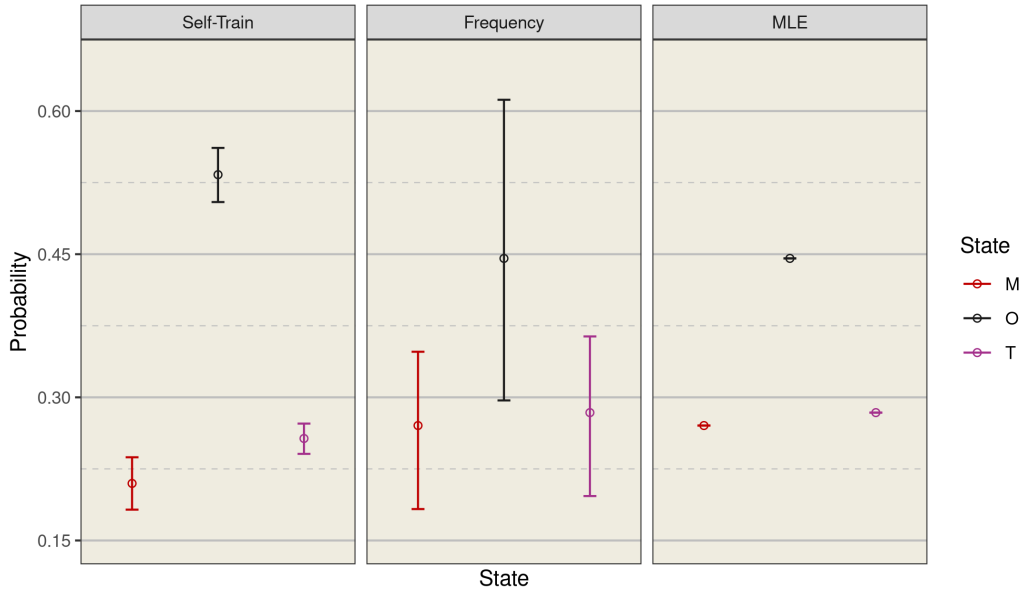

Figure A.4: Estimates of the initial three-state probabilities, with a 90% percentile interval. The left panel shows the self-train, the centre panel the frequency, and the right panel the MLE estimates. M is the malfunction state, O is the observing state, and T is the not-observing state.

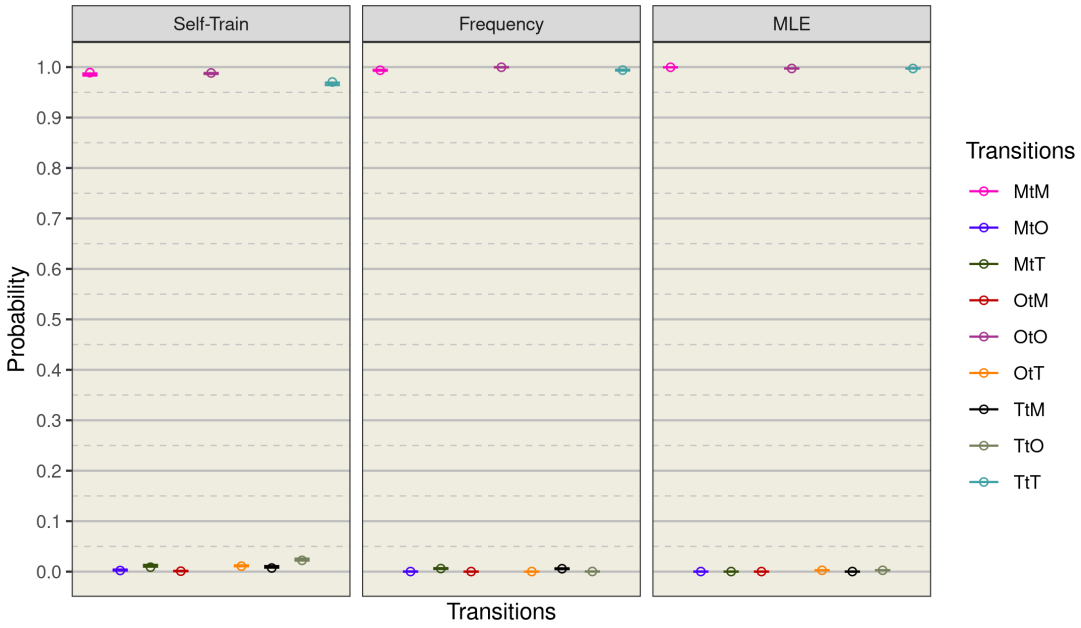

Figure A.5: All the estimated transition probabilities for the three-state models, with a 90% percentile interval. The left panel shows the self-train, the centre panel the frequency, and the right panel the MLE estimates. Some of the transitions are MtM = malfunction to malfunction, OtM = observing to malfunction, TtM = not observing to malfunction, etc.

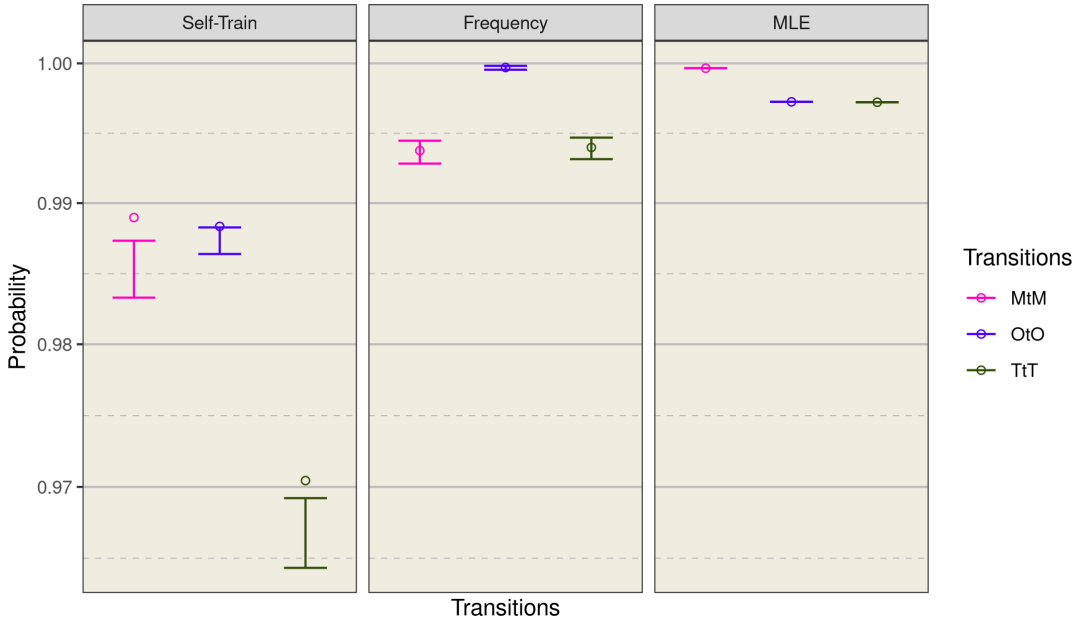

Figure A.6: Estimates of the three-state transition probabilities that are > 0.9. Each estimate is shown with a 90% percentile interval and plotted on a logarithmic scale. The transitions are MtM = malfunction to malfunction, OtO = observing to observing, and TtT = not observing to not observing respectively.

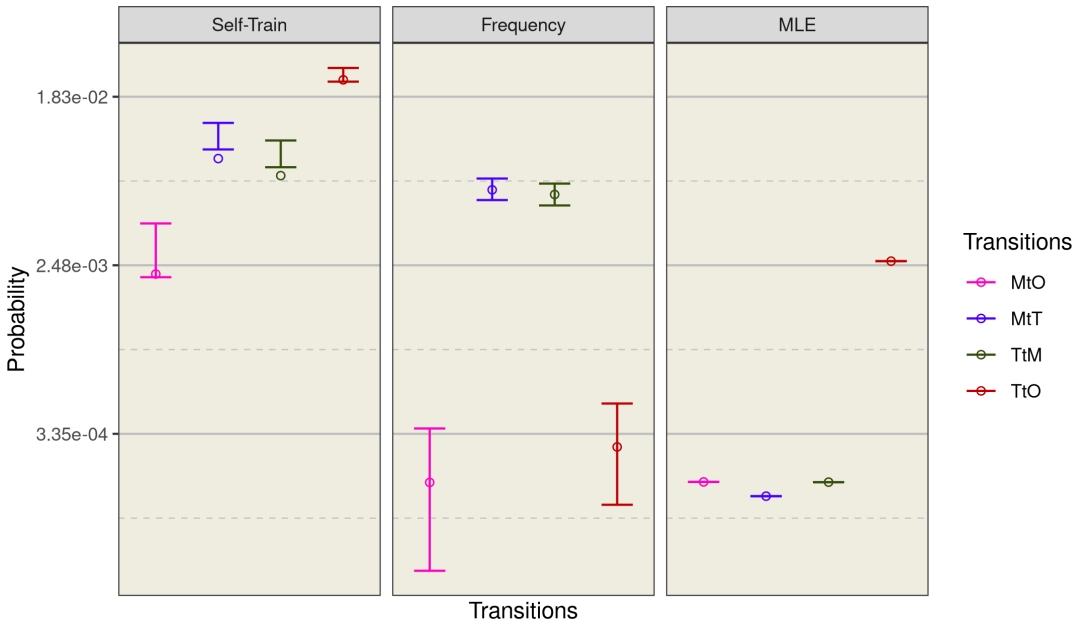

Figure A.7: Estimates of the three-state transition probabilities that are < 0.1. Each estimate is shown with a 90% percentile interval and plotted on a logarithmic scale. Some of the transitions are TtM = not observing to malfunction, OtM = observing to malfunction, etc.

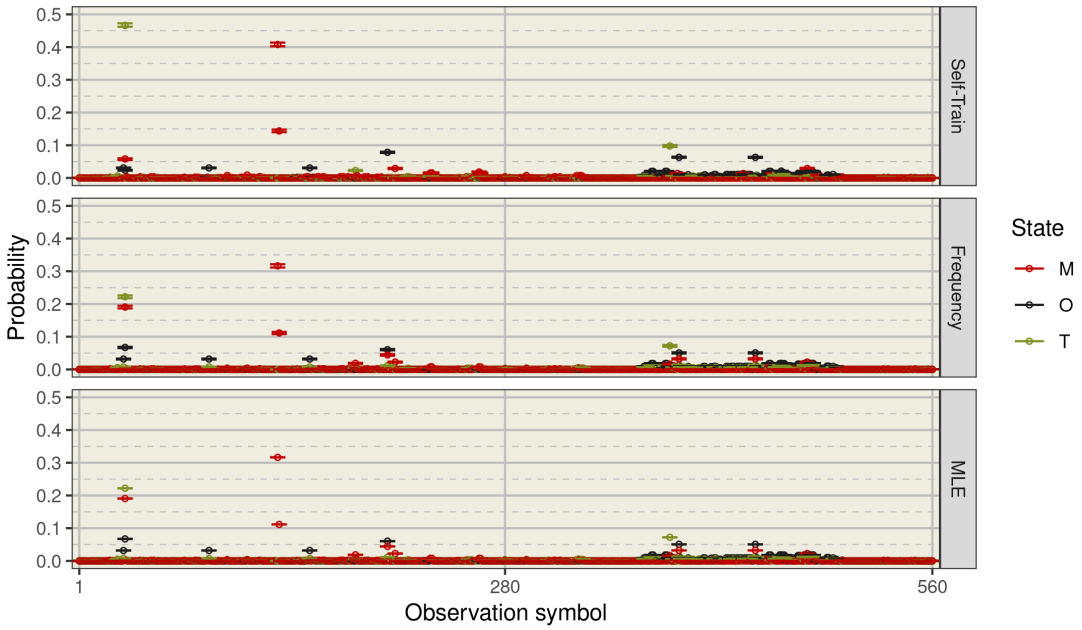

Figure A.8: Estimate of the three-state state-dependent probabilities with a 90% percentile interval. In the top panel are the estimates from the self-train procedure, in the middle panel are those from the frequency, and the bottom panel are the estimates from the MLE. M is the malfunction state, O is the observing state, and T is the not-observing state.

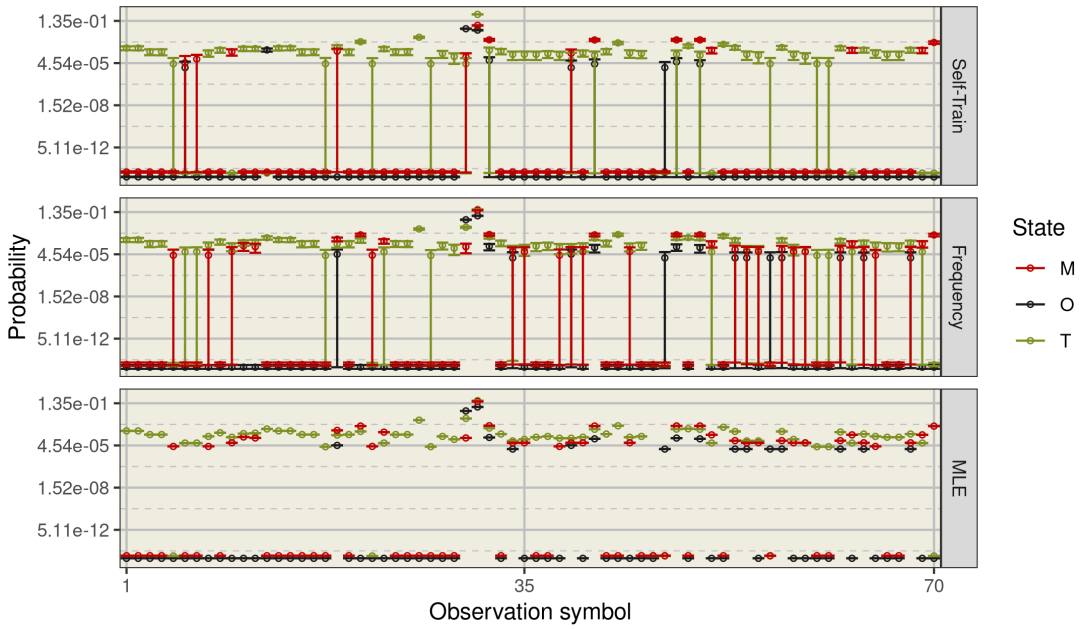

Figure A.9: Estimates of the first 70 three-state state-dependent probabilities, with a 90% percentile interval and plotted on a logarithmic scale. In the top panel are the estimates from the self-train procedure, in the middle panel are those from the frequency, and the bottom panel are the estimates from the MLE. M is the malfunction state, O is the observing state, and T is the not-observing state.

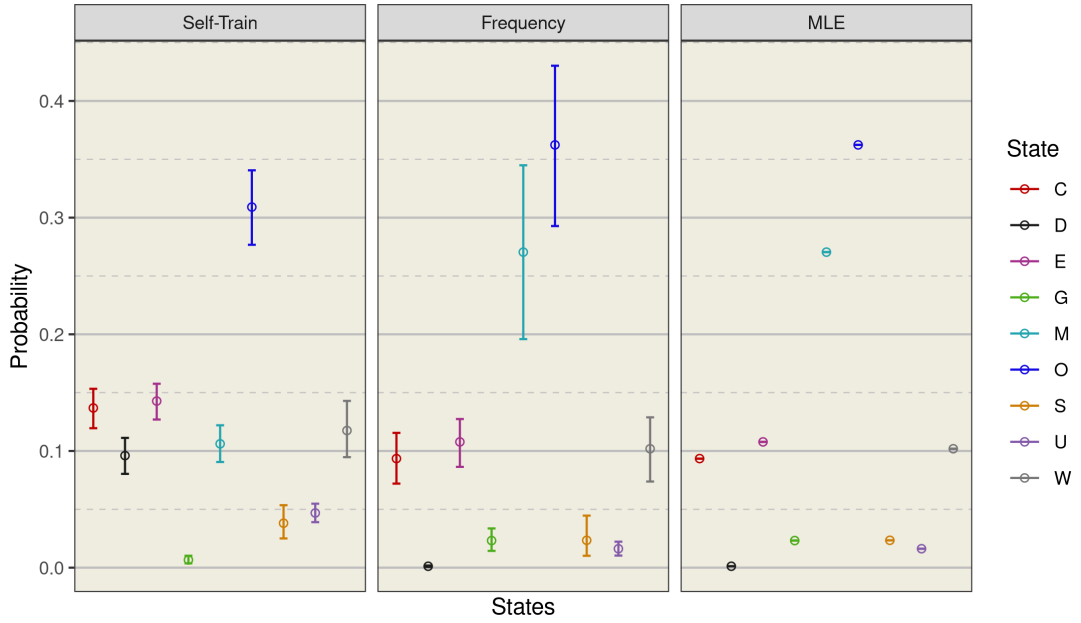

Figure A.10: Estimates of the initial nine-state probabilities, with a 90% percentile interval. The left panel shows the self-train, the centre panel the frequency, and the right panel the MLE estimates. The states are: M is the malfunction, O is the observing, E is the exposure, G is flatfielding, S is sleeping, W is waiting, U is startup, D is shutdown, and C is calibrating.

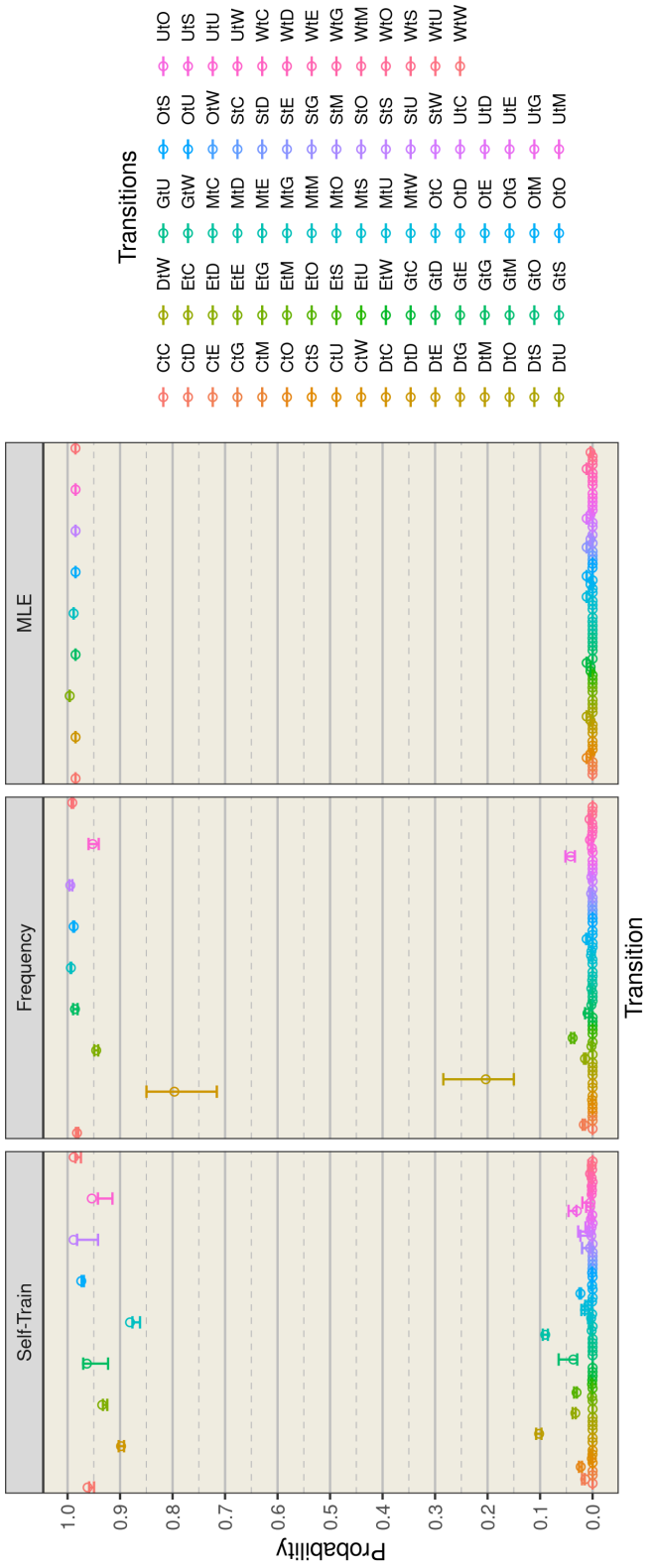

Figure A.11: All the estimated transition probabilities for the nine-state models, with a 90% percentile interval. The left panel shows the self-train, the centre panel the frequency, and the right panel the MLE estimates. Some of the transitions are MtM = malfunction to malfunction, CtE = calibration to exposure, WtS = waiting to sleeping, DtU = shutdown to startup, etc.

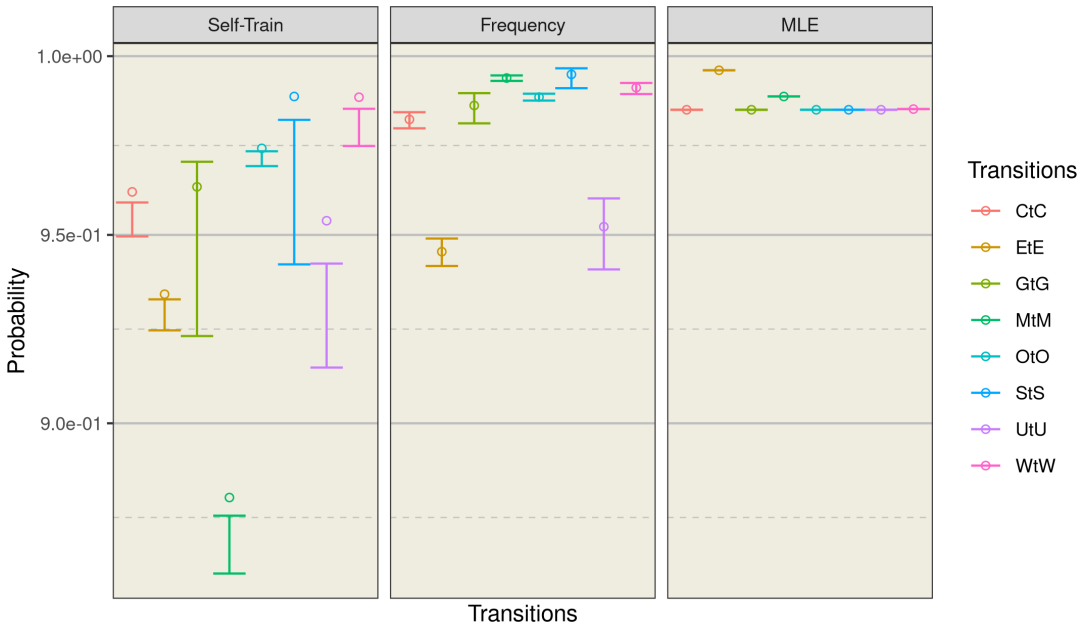

Figure A.12: Estimates of the nine-state transition probabilities that are > 0.7. Each estimate is shown with a 90% percentile interval and plotted on a logarithmic scale. Some of the transitions are CtC = calibration to calibration, MtM = malfunction to malfunction, WtW = waiting to waiting, etc.

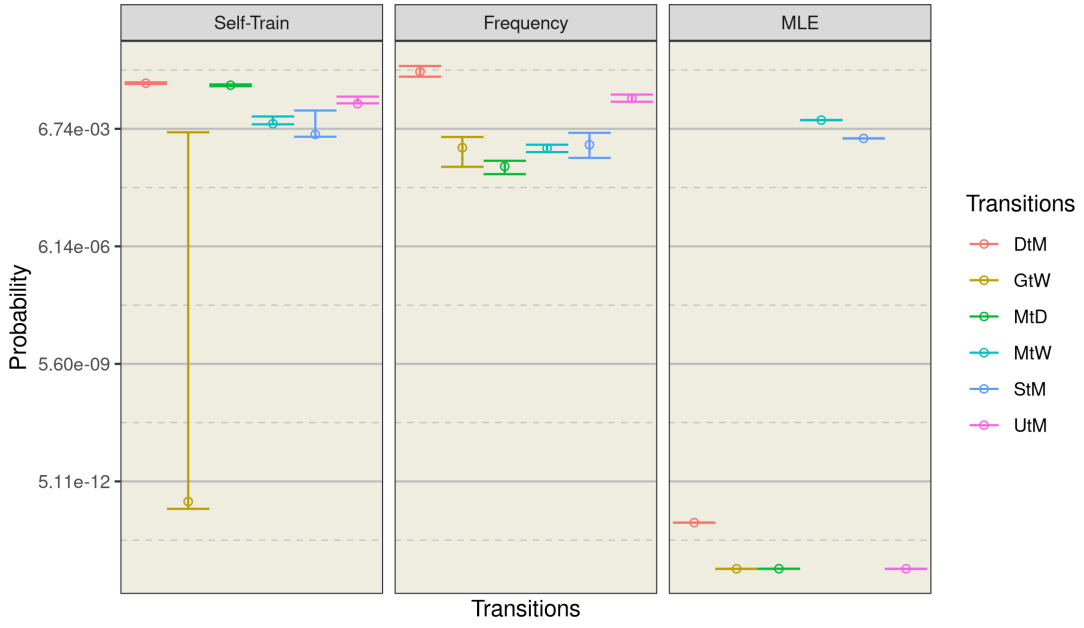

Figure A.13: Estimates of the nine-state transition probabilities that are < 0.15. Each estimate is shown with a 90% percentile interval and plotted on a logarithmic scale. Some of the transitions are StM = sleeping to malfunction, UtM = startup to malfunction, MtW = malfunction to waiting, etc.

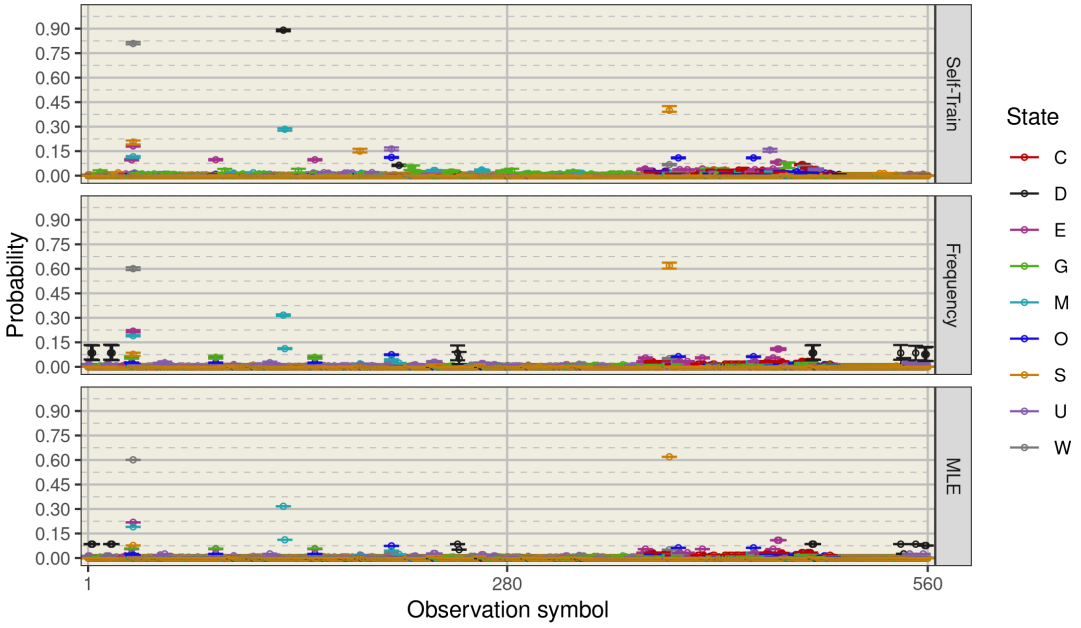

Figure A.14: Estimate of the nine-state state-dependent probabilities with a 90% percentile interval. In the top panel are the estimates from the self-train procedure, in the middle panel are those from the frequency, and the bottom panel are the estimates from the MLE. The states are: M is the malfunction, O is the observing, E is the exposure, G is flatfielding, S is sleeping, W is waiting, U is startup, D is shutdown, and C is calibrating.

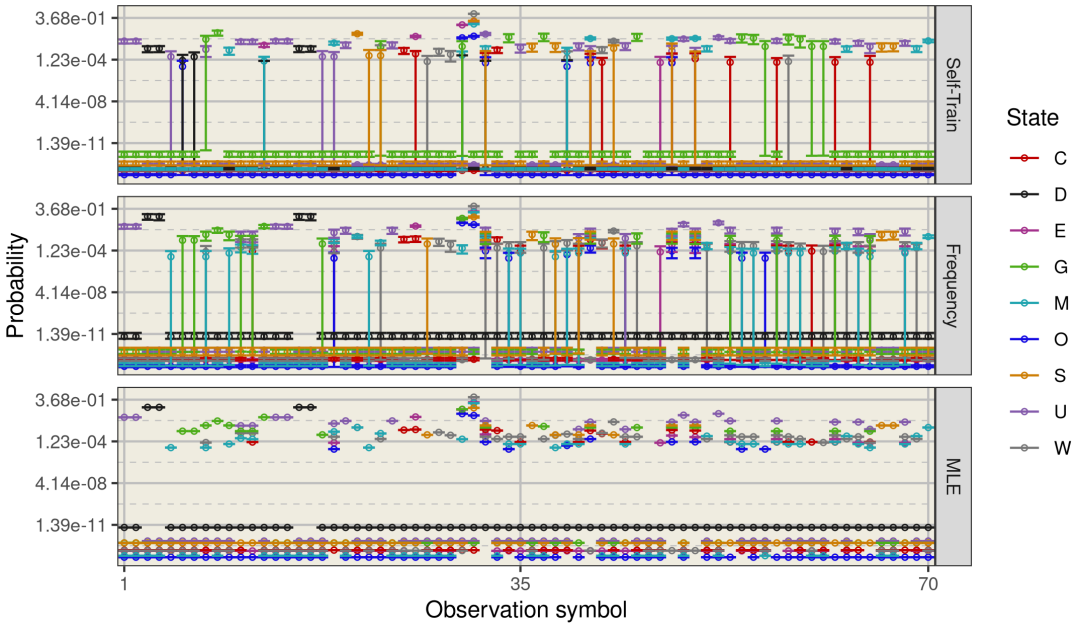

Figure A.15: Estimates of the first 70 nine-state state-dependent probabilities, with a 90% percentile interval and plotted on a logarithmic scale. In the top panel are the estimates from the self-train procedure, in the middle panel are those from the frequency, and the bottom panel are the estimates from the MLE. The states are: M is the malfunction, O is the observing, E is the exposure, G is flatfielding, S is sleeping, W is waiting, U is startup, D is shutdown, and C is calibrating.

A.5 Additional negative log-likelihood

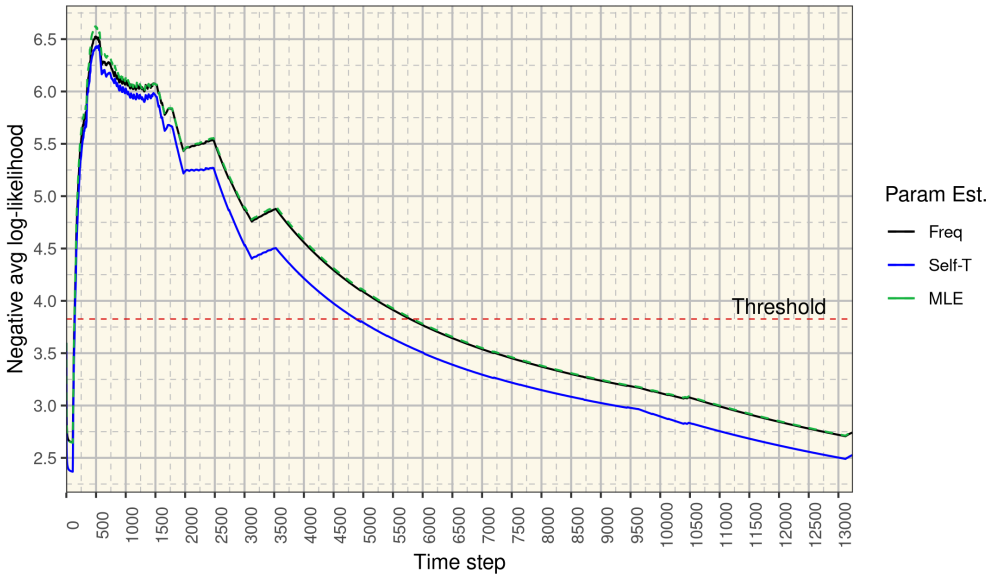

Figure A.16: Comparison of the three-state model's negative entropy. The frequency model's entropy is denoted by the black line, the self-train entropy by the blue line, and the MLE entropy by the dashed green line. The red threshold line indicates that if an entropy value is larger than it, the system should be in an anomalous state.

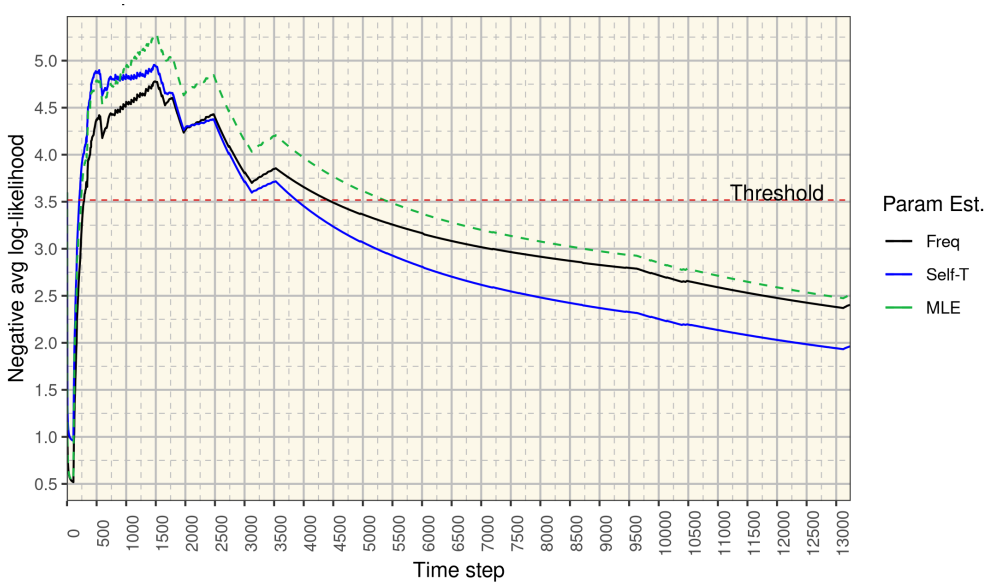

Figure A.17: Comparison of the nine-state model's negative entropy. The frequency model's entropy is denoted by the black line, the self-train entropy by the blue line, and the MLE entropy by the dashed green line. The red threshold line indicates that if an entropy value is larger than it, the system should be in an anomalous state.

A.6 Additional confusion matrices: decoding

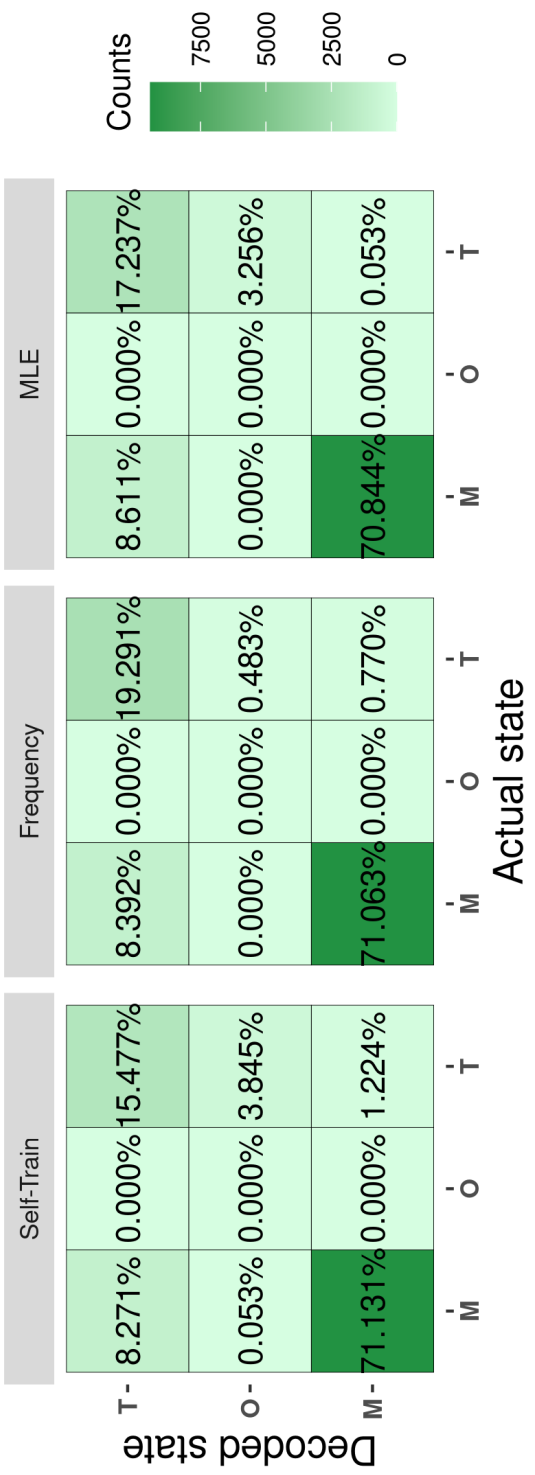

Figure A.18: Three-state decoding confusion matrix. The left panel is the self-train model, the centre is the frequency, and the right is the MLE.

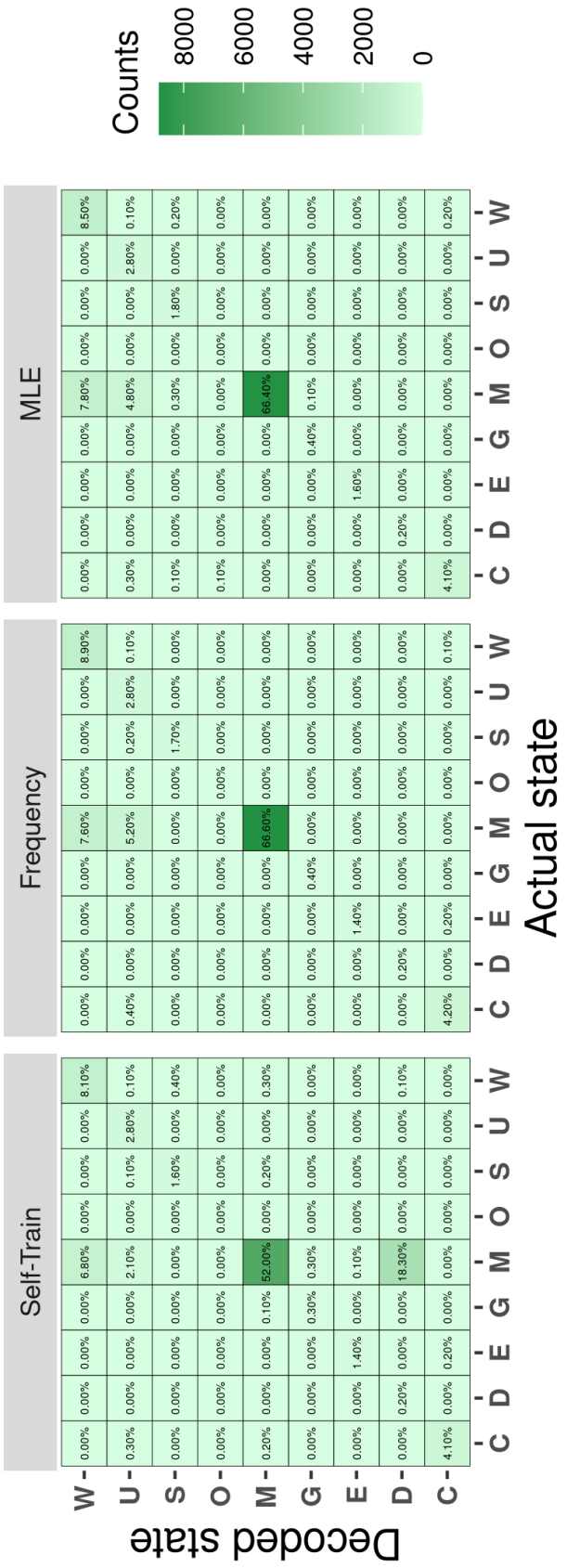

Figure A.19: Nine-state decoding confusion matrix. The left panel is the self-train model, the centre is the frequency, and the right is the MLE.

A.7 Additional decoded step plots

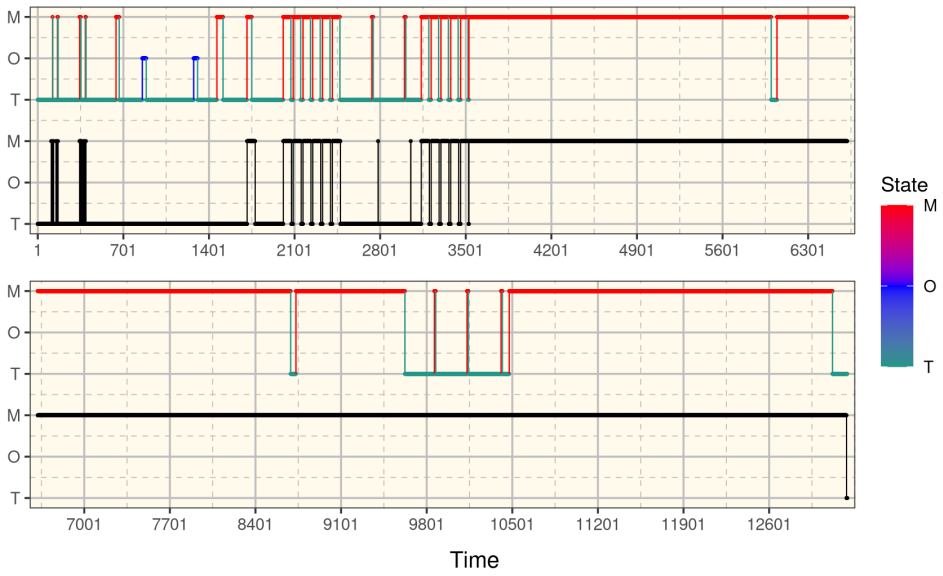

Figure A.20: Three-state anomaly detection performed using decoding, using parameters estimated via the frequency method. The decoded sequence is compared to the actual test sequence, here the black plot represents the actual state sequence. M is the malfunction state, O is the observing state, and T is the not-observing state.

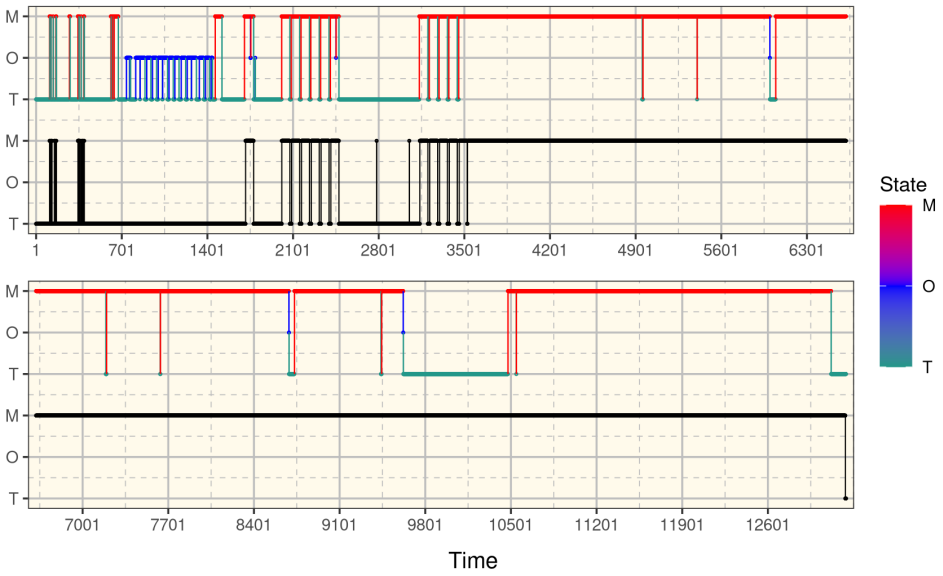

Figure A.21: Three-state anomaly detection performed using decoding, using parameters estimated via the self-train method. The decoded sequence is compared to the actual test sequence, here the black plot represents the actual state sequence. M is the malfunction state, O is the observing state, and T is the not-observing state.

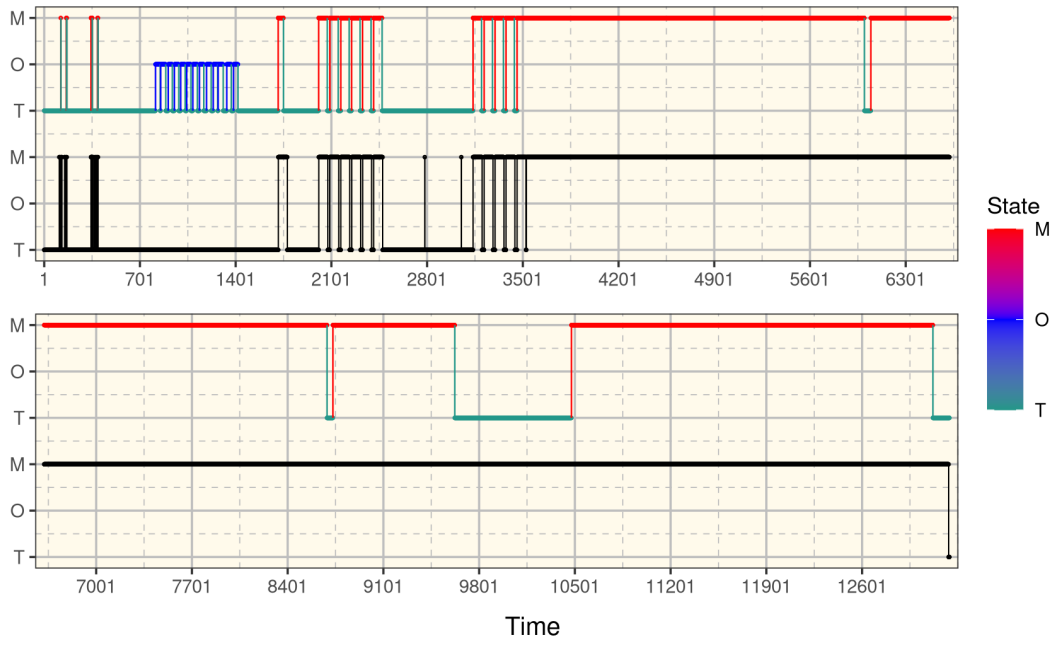

Figure A.22: Three-state anomaly detection performed using decoding, using parameters estimated via the MLE method. The decoded sequence is compared to the actual test sequence, here the black plot represents the actual state sequence. M is the malfunction state, O is the observing state, and T is the not-observing state.

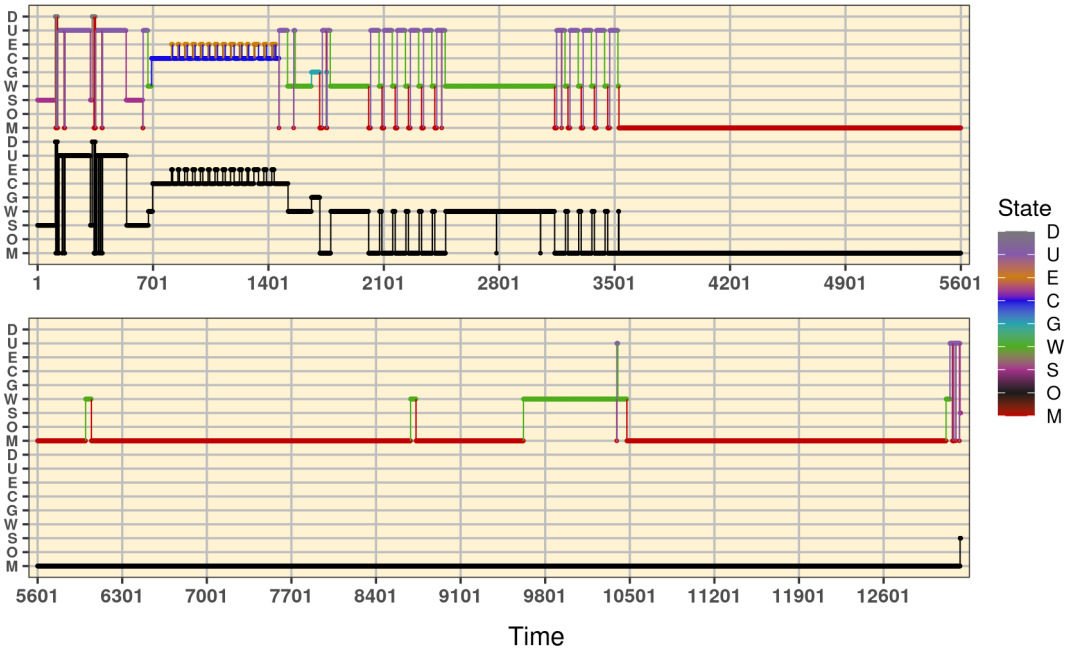

Figure A.23: Nine-state anomaly detection performed using decoding, using parameters estimated via the frequency method. The decoded sequence is compared to the actual test sequence, here the black plot represents the actual state sequence. The states are: M is the malfunction, O is the observing, E is the exposure, G is flatfielding, S is sleeping, W is waiting, U is startup, D is shutdown, and C is calibrating.

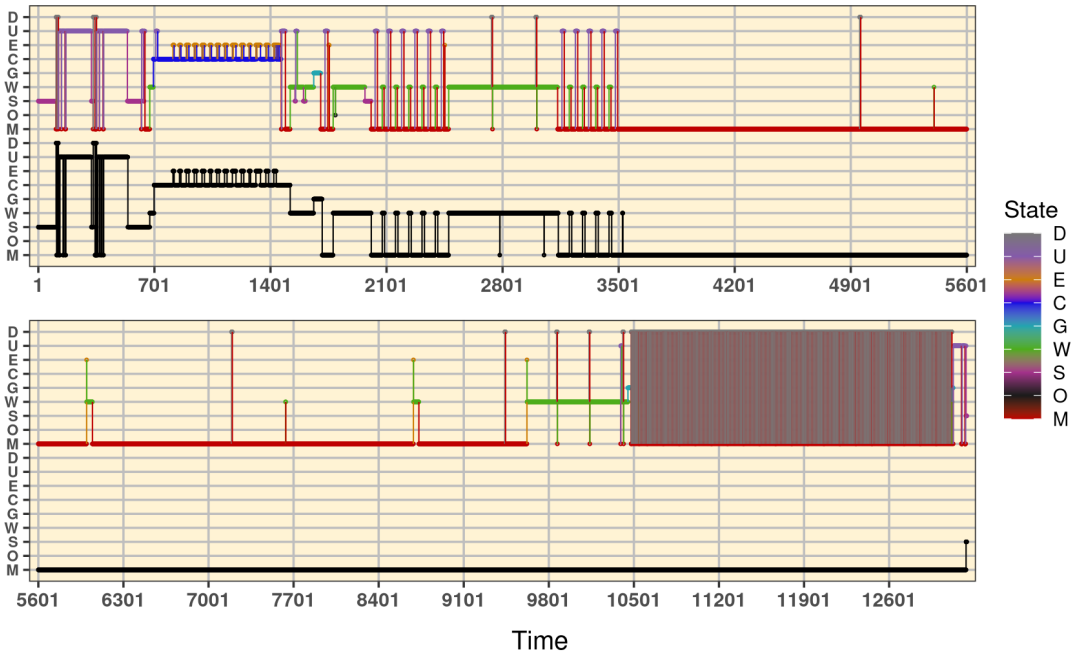

Figure A.24: Nine-state anomaly detection performed using decoding, using parameters estimated via the self-train method. The decoded sequence is compared to the actual test sequence, here the black plot represents the actual state sequence. The states are: M is the malfunction, O is the observing, E is the exposure, G is flatfielding, S is sleeping, W is waiting, U is startup, D is shutdown, and C is calibrating.

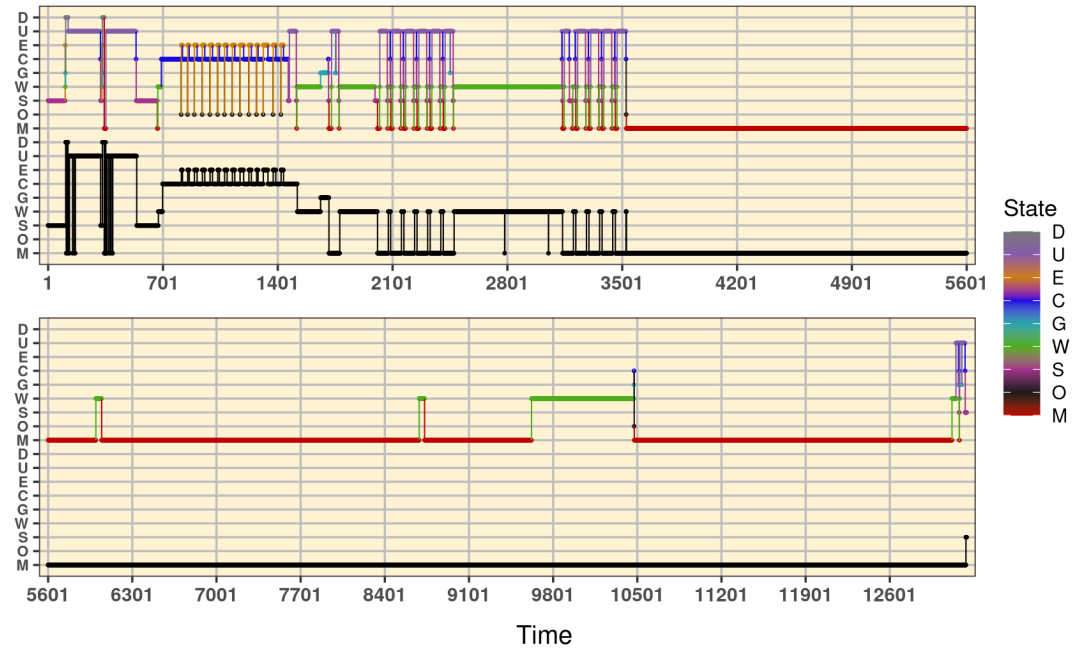

Figure A.25: Nine-state anomaly detection performed using decoding, using parameters estimated via the MLE method. The decoded sequence is compared to the actual test sequence, here the black plot represents the actual state sequence. The states are: M is the malfunction, O is the observing, E is the exposure, G is flatfielding, S is sleeping, W is waiting, U is startup, D is shutdown, and C is calibrating.

A.8 Proof of Equation 4.11

This entry into the appendix means to offer a formal proof of (4.11).

Proof. Let $\mathbf{A}$ denote $\mathbf{\Gamma}^h$ (h-step t.p.m) and $\vec{\mathrm{A}}(,i)$ the ith column of $\mathbf{A}$.

$$\Pr(C_{T+h} = i|\mathbf{X}^{(T)} = \mathbf{x}^{(T)}) = \frac{\Pr(C_{T+h} = i, \mathbf{X}^{(T)} = \mathbf{x}^{(T)})}{\Pr(\mathbf{X}^{(T)} = \mathbf{x}^{(T)})} \cdots\cdots (1)$$

Focusing on the numerator of (1), since the denominator is known to be L_T.

$$\begin{aligned}
&\Pr(C_{T+h} = i, \mathbf{X}^{(T)} = \mathbf{x}^{(T)}) \\
&= \sum_{c_1,\ldots,c_T}^{m} \Pr(C_{T+h} = i, \mathbf{X}^{(T)} = \mathbf{x}^{(T)}, \mathbf{C}^{(T)} = \mathbf{c}^{(T)}) \\
&= \sum_{c_1,\ldots,c_T}^{m} \delta_{c_1}(\gamma_{c_1,c_2}, \cdots, \gamma_{c_{T-1},c_T})(p_{c_1}(x_1), \cdots, p_{c_T}(x_T))a(c_T, i) \\
&= \sum_{c_T}^{m} \left(\sum_{c_1,\ldots,c_{T-1}}^{m} \delta_{c_1}(\gamma_{c_1,c_2}, \cdots, \gamma_{c_{T-1},c_T})(p_{c_1}(x_1), \cdots, p_{c_T}(x_T)) \right) a(c_T, i) \cdots\cdots (2)
\end{aligned}$$

Then using this result from page 66 of Zucchini *et al.* (2016):

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{m} \alpha_t(i)\gamma_{ij} \right) p_j(x_{t+1})$$

Which allows one to simplify (2):

$$\begin{aligned}
(2) &= \sum_{c_T}^{m} \alpha_T(c_T)a(c_T, i) \\
&= \vec{\alpha}_T \vec{\mathrm{A}}(,i) \\
&= \vec{\alpha}_T \mathbf{A} \mathbf{e}_i' \\
&= \vec{\alpha}_T \mathbf{\Gamma}^h \mathbf{e}_i' \cdots\cdots (3)
\end{aligned}$$

Combining (1) and (3) together gives one the desired result:

$$\Pr(C_{T+h} = i|\mathbf{X}^{(T)} = \mathbf{x}^{(T)}) = \vec{\alpha}_T \mathbf{\Gamma}^h \mathbf{e}_i' \qquad \square$$

A.9 Additional confusion matrices: prediction

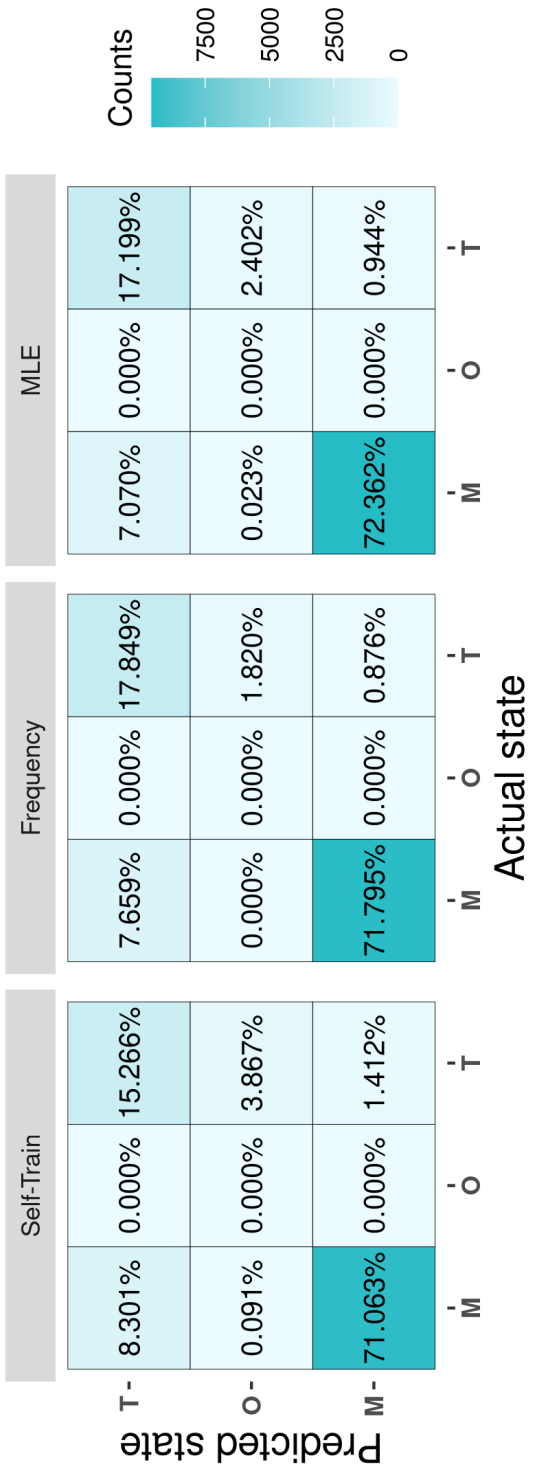

Figure A.26: Confusion matrix for the three-state predictions made using Equation 4.11. The left panel is the self-train model, the centre is the frequency, and the right is the MLE.

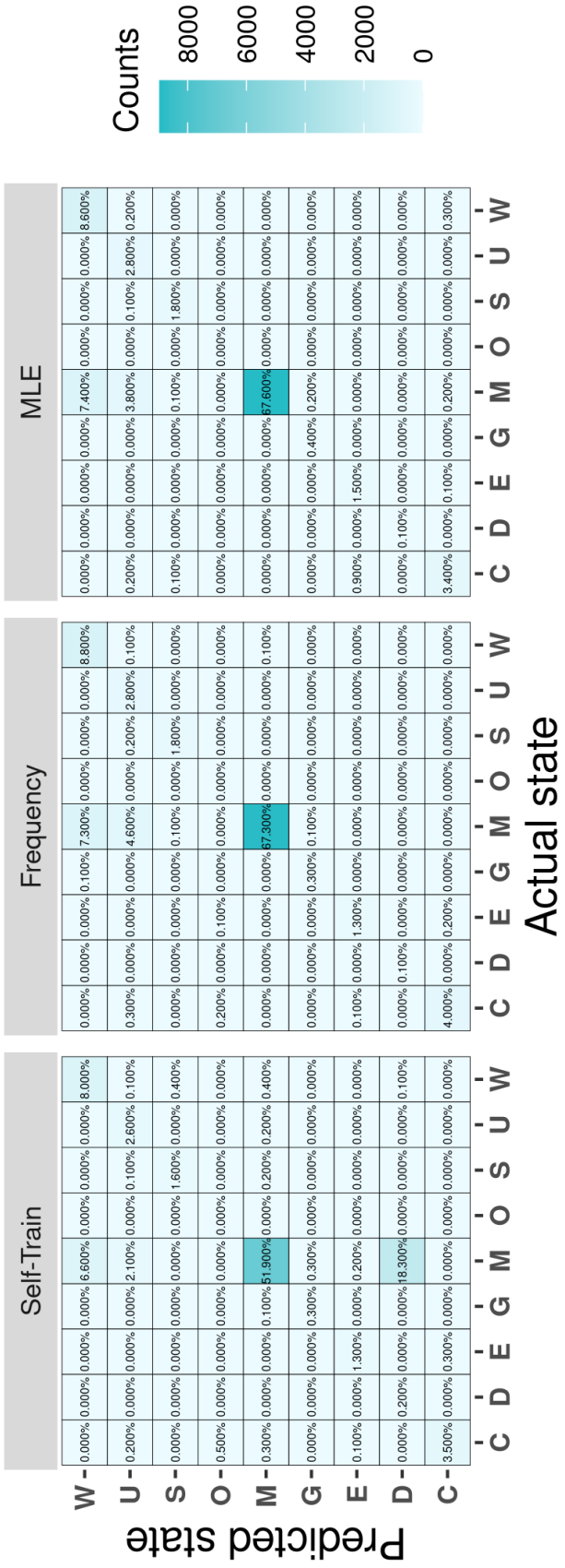

Figure A.27: Confusion matrix for the nine-state predictions made using Equation 4.11. The left panel is the self-train model, the centre is the frequency, and the right is the MLE.

A.10 Additional one-step-ahead step plots

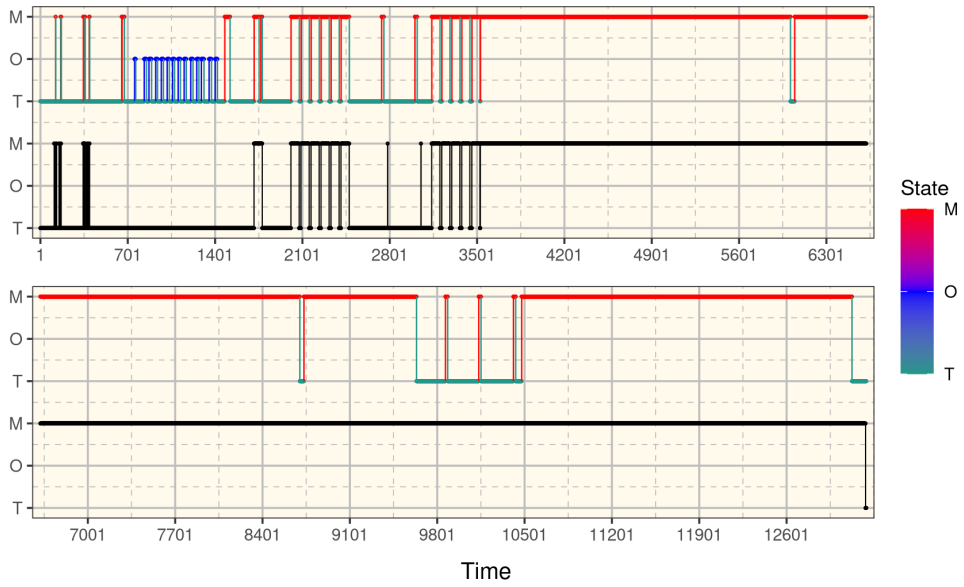

Figure A.28: Three-state anomaly detection performed using the one-step-ahead prediction, with parameters estimated via the frequency method. The predicted sequence is compared to the actual test sequence, the black plot represents the actual state sequence. M is the malfunction state, O is the observing state, and T is the not-observing state.

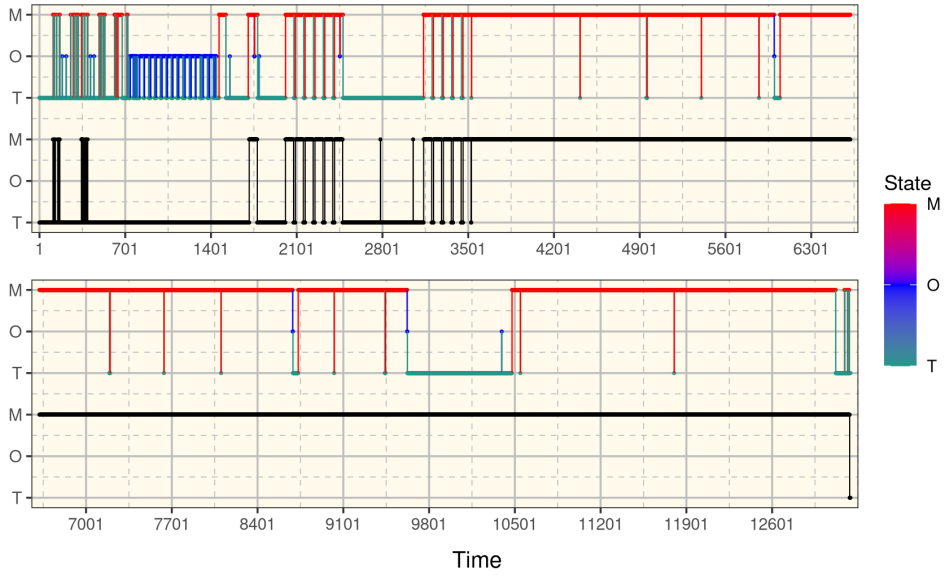

Figure A.29: Three-state anomaly detection performed using the one-step-ahead prediction, with parameters estimated via the self-train method. The predicted sequence is compared to the actual test sequence, the black plot represents the actual state sequence. M is the malfunction state, O is the observing state, and T is the not-observing state.

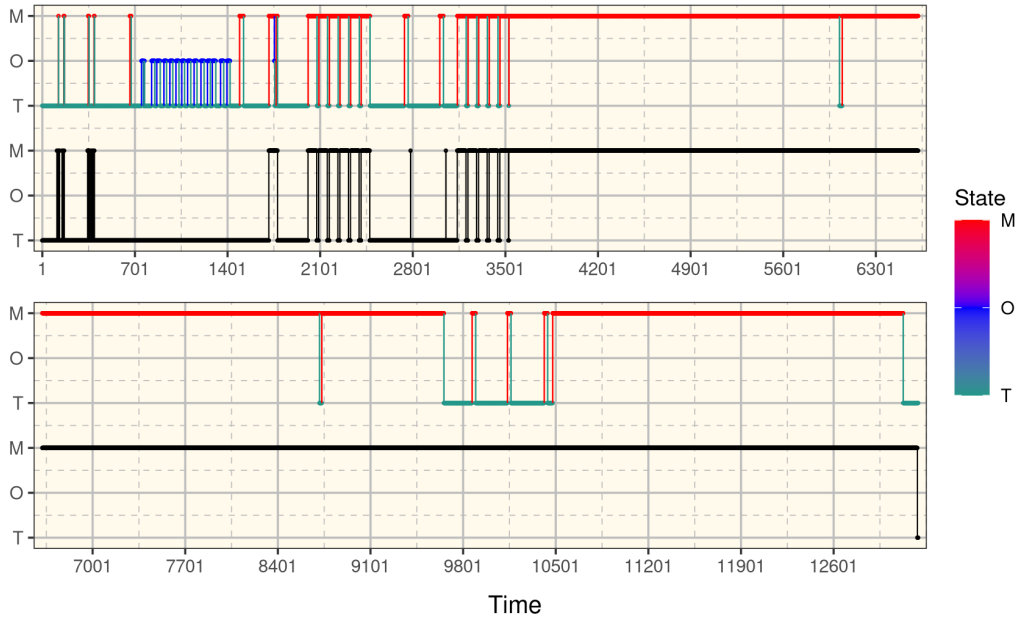

Figure A.30: Three-state anomaly detection performed using the one-step-ahead prediction, with parameters estimated via the MLE method. The predicted sequence is compared to the actual test sequence, the black plot represents the actual state sequence. M is the malfunction state, O is the observing state, and T is the not-observing state.

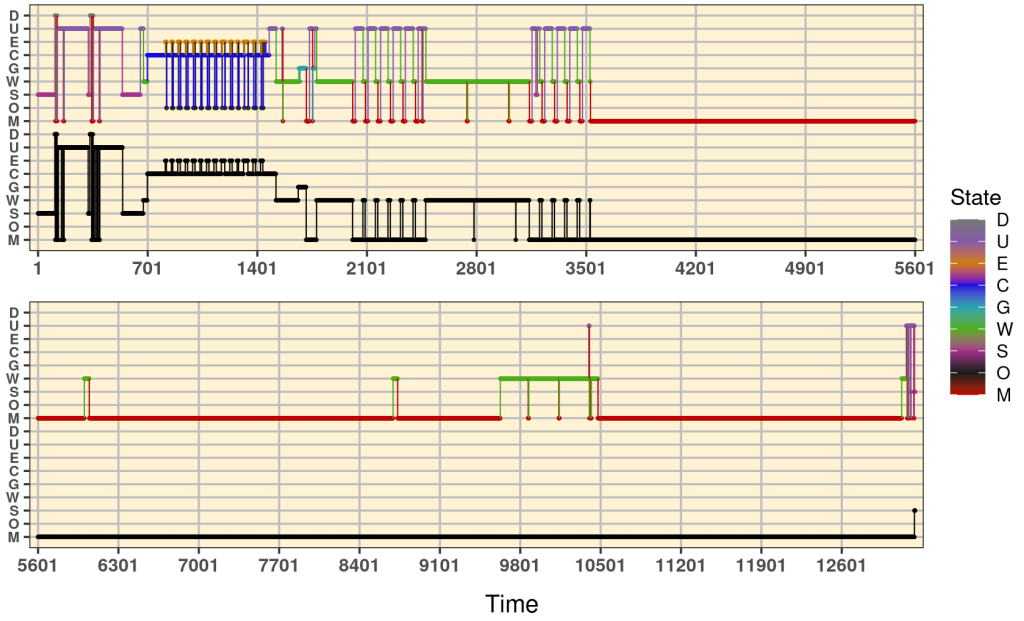

Figure A.31: Nine-state anomaly detection performed using the one-step-ahead prediction, with parameters estimated via the frequency method. The predicted sequence is compared to the actual test sequence, the black plot represents the actual state sequence. The states are: M is the malfunction, O is the observing, E is the exposure, G is flatfielding, S is sleeping, W is waiting, U is startup, D is shutdown, and C is calibrating.

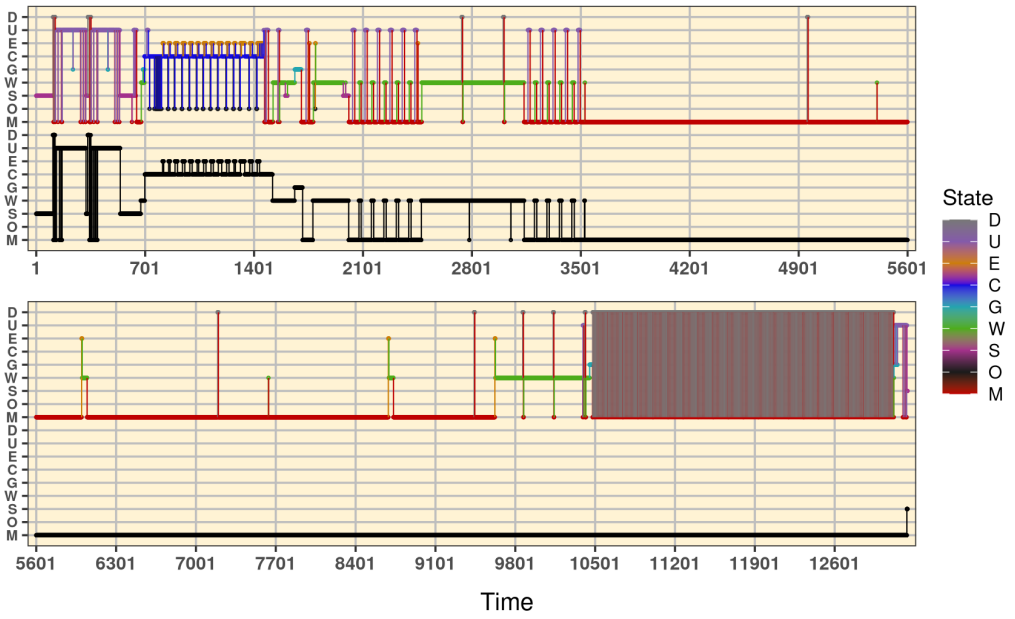

Figure A.32: Nine-state anomaly detection performed using the one-step-ahead prediction, with parameters estimated via the self-train method. The predicted sequence is compared to the actual test sequence, the black plot represents the actual state sequence. The states are: M is the malfunction, O is the observing, E is the exposure, G is flatfielding, S is sleeping, W is waiting, U is startup, D is shutdown, and C is calibrating.

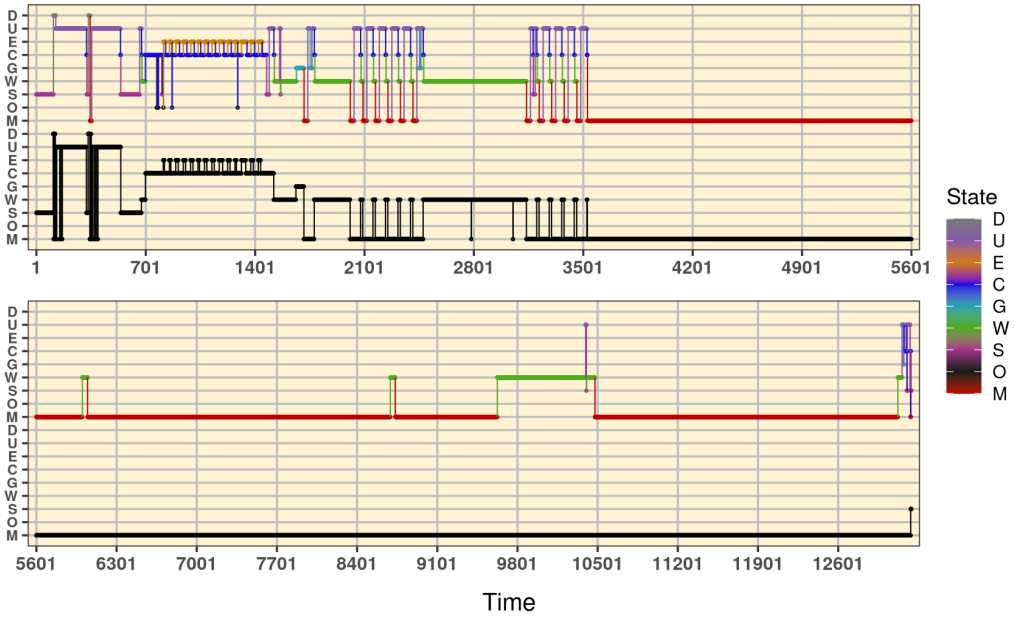

Figure A.33: Nine-state anomaly detection performed using the one-step-ahead prediction, with parameters estimated via the MLE method. The predicted sequence is compared to the actual test sequence, the black plot represents the actual state sequence. The states are: M is the malfunction, O is the observing, E is the exposure, G is flatfielding, S is sleeping, W is waiting, U is startup, D is shutdown, and C is calibrating.

List of References

Alspaugh, S. *et al.* (2014). Analyzing log analysis: An empirical study of user log mining. In: *28th Large Installation System Administration Conference (LISA14)*, pp. 62–77. USENIX Association. ISBN 978-1-931971-17-1.
Available at: https://www.usenix.org/conference/lisa14/conference-program/presentation/alspaugh

Ayer, L. and Mowlavi, N. (2008). Variable stars across the observational HR diagram. *Journal of Physics: Conference Series*, vol. 118, p. 012010.
DOI: 10.1088/1742-6596/118/1/012010.
Available at: https://doi.org/10.1088/1742-6596/118/1/012010

Bagnasco, S., Berzano, D., Guarise, A., Lusso, S., Masera, M. and Vallero, S. (2015). Towards monitoring-as-a-service for scientific computing cloud applications using the ElasticSearch ecosystem. *Journal of Physics: Conference Series*, vol. 664, no. 2, p. 022040.
DOI: 10.1088/1742-6596/664/2/022040.
Available at: https://doi.org/10.1088/1742-6596/664/2/022040

Baum, L.E. *et al.* (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, vol. 3, no. 1, pp. 1–8.

Bilmes, J.A. (2006). What hmms can do. *IEICE TRANSACTIONS on Information and Systems*, vol. 89, no. 3, pp. 869–891.
DOI: 10.1093/ietisy/e89–d.3.869.

BlackGEM (2022*a*). The BlackGEM local transient survey. [Online].
Accessed: 2022-02-15.
Available at: https://astro.ru.nl/blackgem/?page_id=302

BlackGEM (2022*b*). The BlackGEM qscan survey. [Online].
Accessed: 2022-02-15.
Available at: https://astro.ru.nl/blackgem/?page_id=290

Bloemen, S. *et al.* (2016). Meerlicht and blackgem: custom-built telescopes to detect faint optical transients. In: *Ground-based and Airborne Telescopes VI*, vol. 9906, pp. 2118 – 2126. International Society for Optics and Photonics, Proc. SPIE 9906.
DOI: 10.1117/12.2232522.
Available at: https://doi.org/10.1117/12.2232522

Booth, R.S. and Jonas, J.L. (2012). An overview of the MeerKAT project. *African Skies/Cieux Africains*, vol. 16, pp. 101 – 104. Provided by the SAO/NASA Astrophysics Data System.
Available at: https://ui.adsabs.harvard.edu/abs/2012AfrSk..16..101B

Boussemart, Y., Cummings, M.L., Fargeas, J.L. and Roy, N. (2011). Supervised vs. unsupervised learning for operator state modeling in unmanned vehicle settings. *Journal of Aerospace Computing, Information, and Communication*, vol. 8, no. 3, pp. 71–85.
DOI: 10.2514/1.46767.
Available at: https://doi.org/10.2514/1.46767

Chernick, M.R. (2008). *Bootstrap Methods: A Guide for Practitioners and Researchers*. 2nd edn. Wiley. ISBN 978-0-471-75621-7.

Chuvakin, A. (2007). Log mining: Beyond Log Analysis. [Online].
Accessed: 2021-04-08.
Available at: https://www.slideshare.net/anton_chuvakin/log-mining-beyond-log-analysis

Damevski, K., Chen, H., Shepherd, D. and Pollock, L. (2016). Interactive exploration of developer interaction traces using a hidden markov model. In: *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, p. 126–136. Association for Computing Machinery. ISBN 9781450341868.
DOI: 10.1145/2901739.2901741.
Available at: https://doi.org/10.1145/2901739.2901741

Drzał, M. *et al.* (2016). *BlackGEM software for automation*. Sybilla Technologies, Bydgoszcz, Poland. Technical Note Issue 1, Revision 3.

Du, M., Li, F., Zheng, G. and Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, p. 1285–1298. Association for Computing Machinery, New York, NY, USA. ISBN 9781450349468.
DOI: 10.1145/3133956.3134015.
Available at: https://doi.org/10.1145/3133956.3134015

Eaton, J.A., Henry, G.W. and Fekel, F.C. (2003). *Advantages of Automated Observing with Small Telescopes*, pp. 489–507. Springer Netherlands. ISBN 978-94-010-0253-0.

DOI: 10.1007/978-94-010-0253-0_38.
Available at: https://doi.org/10.1007/978-94-010-0253-0_38

Efron, B. and Tibshirani, R. (1986). Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, vol. 1, no. 1, pp. 54–75.
Available at: https://www.jstor.org/stable/2245500

Efron, B. and Tibshirani, R.J. (1993). *An Introduction to the Bootstrap*. 57, 1st edn. Chapman & Hall. ISBN 0-412-04231-2.

Elastic (2020). Elastic stack: Elasticsearch, Kibana, Beats & Logstash. [Online].
Accessed: 2020-03-15.
Available at: https://www.elastic.co/elastic-stack

Fu, Q. *et al.* (2009). Execution anomaly detection in distributed systems through unstructured log analysis. In: *2009 Ninth IEEE International Conference on Data Mining*, pp. 149–158. Institute of Electrical and Electronics Engineering.
DOI: 10.1109/ICDM.2009.60.
Available at: https://ieeexplore.ieee.org/abstract/document/5360240

Fujiwara, H. (2014). Exterior (southeast). [Online].
Accessed: 2022-01-27.
Available at: https://subarutelescope.org/en/gallery/spherical/1998/06/23/1934.html

Gadler, D., Mairegger, M., Janes, A. and Russo, B. (2017). Mining logs to model the use of a system. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 334–343.
DOI: 10.1109/ESEM.2017.47.
Available at: https://ieeexplore.ieee.org/abstract/document/8170120

Ganti, V. *et al.* (1999). CACTUS—clustering categorical data using summaries. In: *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 73–83.
DOI: 10.1145/312129.312201.
Available at: https://dl.acm.org/doi/pdf/10.1145/312129.312201

Gil, J.P., Reveco, J. and Shen, T.-C. (2016). Operational logs analysis at ALMA observatory based on ELK stack. In: *Software and Cyberinfrastructure for Astronomy IV*, vol. 9913, pp. 814 – 822. International Society for Optics and Photonics, SPIE.
Available at: https://doi.org/10.1117/12.2232258

Grandini, M., Bagli, E. and Visani, G. (2020). Metrics for multi-class classification: an overview. 2008.05756.

Guo, Z., Jiang, G., Chen, H. and Yoshihira, K. (2006). Tracking probabilistic correlation of monitoring data for fault detection in complex systems. In: *International Conference on Dependable Systems and Networks (DSN'06)*, pp. 259–268. Institute of Electrical and Electronics Engineering.
DOI: 10.1109/DSN.2006.70.
Available at: https://ieeexplore.ieee.org/abstract/document/1633515

He, P. *et al.* (2016). An evaluation study on log parsing and its use in log mining. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 654–661. Institute of Electrical and Electronics Engineering.
DOI: 10.1109/DSN.2016.66.
Available at: https://ieeexplore.ieee.org/abstract/document/7579781

Heck, L. and McClellan, J. (1991). Mechanical system monitoring using hidden markov models. In: *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, pp. 1697–1700. IEEE.
DOI: 10.1109/ICASSP.1991.150631.
Available at: https://ieeexplore.ieee.org/document/150631

Hovland, G.E. and McCarragher, B.J. (1998). Hidden markov models as a process monitor in robotic assembly. *The International Journal of Robotics Research*, vol. 17, no. 2, pp. 153–168.
DOI: 10.1177/027836499801700204.
Available at: https://doi.org/10.1177/027836499801700204

Hüdepohl, G. (2005). The observing platform as the crow flies. [Online].
Accessed: 2022-01-27.
Available at: https://www.eso.org/public/images/eso-paranal-33/

James, G., Witten, D., Hastie, T. and Tibshirani, R. (2013). *An introduction to statistical learning*, vol. 112. Springer. ISBN 978-1-0716-1417-4.
DOI: 10.1007/978-1-0716-1418-1.
Available at: https://doi.org/10.1007/978-1-0716-1418-1

Jayathilake, D. (2012). Towards structured log analysis. In: *2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE)*, pp. 259–264. Institute of Electrical and Electronics Engineering.
DOI: 10.1109/JCSSE.2012.6261962.
Available at: https://ieeexplore.ieee.org/abstract/document/6261962

Jiang, G., Chen, H. and Yoshihira, K. (2006). Discovering likely invariants of distributed transaction systems for autonomic system management. *Cluster Computing*, vol. 9, no. 4, pp. 385–399.

DOI: 10.1007/s10586-006-0008-1.
Available at: https://link.springer.com/content/pdf/10.1007/s10586-006-0008-1.pdf

Jordan, M.I. (2004). Graphical models. *Statistical Science*, vol. 19, no. 1, pp. 140–155.
DOI: 10.1214/088342304000000026.
Available at: https://doi.org/10.1214/088342304000000026

Jurafsky, D. and Martin, J.H. (2020). Speech and language processing. Taken from web exclusive Appendix A.
Available at: https://web.stanford.edu/~jurafsky/slp3/A.pdf

Lakhina, A., Crovella, M. and Diot, C. (2004). Diagnosing network-wide traffic anomalies. In: *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, p. 219–230. Association for Computing Machinery, New York, NY, USA. ISBN 1581138628.
DOI: 10.1145/1015467.1015492.
Available at: https://doi.org/10.1145/1015467.1015492

Langrock, R. and Zucchini, W. (2011). Hidden markov models with arbitrary state dwell-time distributions. *Computational Statistics & Data Analysis*, vol. 55, no. 1, pp. 715–724. ISSN 0167-9473.
DOI: 10.1016/j.csda.2010.06.015.
Available at: https://www.sciencedirect.com/science/article/pii/S0167947310002604

Marinkovic, A. (2013). Spectacular view of alma. [Online].
Accessed: 2022-01-27.
Available at: https://www.almaobservatory.org/en/image-gallery/?images-tags=antennas&pg=7

Michelot, T., Langrock, R. and Patterson, T.A. (2016). movehmm: an r package for the statistical modelling of animal movement data using hidden markov models. *Methods in Ecology and Evolution*, vol. 7, no. 11, pp. 1308–1315.
DOI: 10.1111/2041-210X.12578.
Available at: https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.12578

Min, L. and Shun-Zheng, Y. (2006). A network-wide traffic anomaly detection method based on hsmm. In: *2006 International Conference on Communications, Circuits and Systems*, vol. 3, pp. 1636–1640.
DOI: 10.1109/ICCCAS.2006.284987.

Nagappan, M. and Vouk, M.A. (2010). Abstracting log lines to log event types for mining software system logs. In: *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pp. 114–117. Institute of Electrical and Electronics Engineering.
DOI: 10.1109/MSR.2010.5463281.
Available at: https://ieeexplore.ieee.org/abstract/document/5463281

Nakamura, K. *et al.* (2020). Prototyping of log analysis infrastructure for the Subaru telescope based on the ALMA experience. In: *Observatory Operations: Strategies, Processes, and Systems VIII*, vol. 11449, pp. 524 – 531. International Society for Optics and Photonics, Proc. SPIE 11449.
DOI: 10.1117/12.2561190.
Available at: https://doi.org/10.1117/12.2561190

Nash, J.C. (2016). *optimr: A Replacement and Extension of the 'optim' Function.* R package version 2019-12.16.
Available at: https://CRAN.R-project.org/package=optimr

Oliner, A. *et al.* (2012). Advances and challenges in log analysis. *Communications of the ACM*, vol. 55, no. 2, pp. 55 – 61.
DOI: 10.1145/2076450.2076466.
Available at: https://dl.acm.org/doi/10.1145/2076450.2076466

Panuccio, A., Bicego, M. and Murino, V. (2002). A hidden markov model-based approach to sequential data clustering. In: *Structural, Syntactic, and Statistical Pattern Recognition*, pp. 734–743. Springer Berlin Heidelberg. ISBN 978-3-540-70659-5.
DOI: 10.1007/3-540-70659-3_77.
Available at: https://link.springer.com/chapter/10.1007/3-540-70659-3_77

Paterson, K. (2019). *Detecting optical transients and variables with MeerLICHT.* Ph.D. thesis, University of Cape Town, Cape Town, South Africa.

Pettinato, M., Gil, J.P., Galeas, P. and Russo, B. (2019). Log mining to re-construct system behavior: An exploratory study on a large telescope system. *Information and Software Technology*, vol. 114, pp. 121–136. ISSN 0950-5849.
DOI: 10.1016/j.infsof.2019.06.011.

Petty, M.D. (2012). Calculating and using confidence intervals for model validation. In: *Proceedings of the Fall 2012 Simulation Interoperability Workshop*, pp. 10–14.

Pieterse, D. (2019). *Telescope characterisation of the wide-field optical telescope MeerLICHT.* MSc Thesis, Radboud University, Nijmegen, Netherlands.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
Available at: https://www.R-project.org/

Rabiner, L.R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257 – 286.
DOI: 10.1109/5.18626.

SARAO (2016). MeerKAT joins the ranks of world's great scientific instruments through its first light image. [Online].
Accessed: 2021-03-11.
Available at: https://www.sarao.ac.za/media-releases/meerkat-joins-the-ranks-of-the-worlds-great-scientific-instruments-through-its-first-light-image/

Schipani, P. *et al.* (2020). Long term monitoring of the VST through telescope log data. In: *Observatory Operations: Strategies, Processes, and Systems VIII*, vol. 11449, pp. 566 – 575. International Society for Optics and Photonics, Proc. SPIE 11449.
DOI: 10.1117/12.2560495.
Available at: https://doi.org/10.1117/12.2560495

Smyth, P. (1994*a*). Hidden markov models for fault detection in dynamic systems. *Pattern Recognition*, vol. 27, no. 1, pp. 149–164. ISSN 0031-3203.
DOI: 10.1016/0031-3203(94)90024-8.
Available at: https://doi.org/10.1016/0031-3203(94)90024-8

Smyth, P. (1994*b*). Markov monitoring with unknown states. *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 9, pp. 1600–1612.
DOI: 10.1109/49.339929.
Available at: https://ieeexplore.ieee.org/10.1109/49.339929

Smyth, P. and Mellstrom, J. (1991). Fault diagnosis of antenna pointing systems using hybrid neural network and signal processing models. In: *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, p. 667–674. Morgan Kaufmann Publishers Inc. ISBN 1558602224.
DOI: 10.5555/2986916.2986998.
Available at: https://dl.acm.org/doi/abs/10.5555/2986916.2986998

Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, vol. 45, no. 4, pp. 427–437. ISSN 0306-4573.
DOI: 10.1016/j.ipm.2009.03.002.
Available at: https://www.sciencedirect.com/science/article/pii/S0306457309000259

Splunk (2020). The Data-to-Everything platform built for the cloud. [Online]. Accessed: 2020-03-15.
Available at: https://www.splunk.com/

Starovasnik, D.M. (2012). Know the automation situation. *Industrial Engineering*, vol. 44, no. 2, pp. 44–48.

Suh-Lee, C., Jo, J.-Y. and Kim, Y. (2016). Text mining for security threat detection discovering hidden information in unstructured log messages. In: *2016 IEEE Conference on Communications and Network Security (CNS)*, pp. 252–260.
DOI: 10.1109/CNS.2016.7860492.
Available at: https://ieeexplore.ieee.org/abstract/document/7860492

Sybilski, P.W. *et al.* (2014). Software for autonomous astronomical observatories: challenges and opportunities in the age of big data. In: *Software and Cyberinfrastructure for Astronomy III*, vol. 9152, pp. 473 – 486. International Society for Optics and Photonics, Proc. SPIE 9152.
DOI: 10.1117/12.2055836.
Available at: https://doi.org/10.1117/12.2055836

Tamposis, I.A., Tsirigos, K.D., Theodoropoulou, M.C., Kontou, P.I. and Bagos, P.G. (2018 11). Semi-supervised learning of Hidden Markov Models for biological sequence analysis. *Bioinformatics*, vol. 35, no. 13, pp. 2208–2215. ISSN 1367-4803.
DOI: 10.1093/bioinformatics/bty910, https://academic.oup.com/bioinformatics/article-pdf/35/13/2208/28878245/bty910.pdf.

Tan, X. and Xi, H. (2008). Hidden semi-markov model for anomaly detection. *Applied Mathematics and Computation*, vol. 205, no. 2, pp. 562–567. ISSN 0096-3003. Special Issue on Advanced Intelligent Computing Theory and Methodology in Applied Mathematics and Computation.
Available at: https://www.sciencedirect.com/science/article/pii/S0096300308003342

Tang, L. *et al.* (2011). Logsig: Generating system events from raw textual logs. In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, p. 785–794. Association for Computing Machinery. ISBN 9781450307178.
DOI: 10.1145/2063576.2063690.
Available at: https://doi.org/10.1145/2063576.2063690

Taylor, A.R. and Jarvis, M. (2017). MIGHTEE: The MeerKAT international GHz tiered extragalactic exploration. *IOP Conference Series: Materials Science and Engineering*, vol. 198, p. 012014.

DOI: 10.1088/1757-899x/198/1/012014.
Available at: https://doi.org/10.1088/1757-899x/198/1/012014

ter Horst, R., Kragt, J., Lesman, D. and Navarro, R. (2016). Novel and efficient ADC concept for BlackGEM telescope. In: *Advances in Optical and Mechanical Technologies for Telescopes and Instrumentation II*, vol. 9912, pp. 542 – 553. International Society for Optics and Photonics, Proc. SPIE 9912.
DOI: 10.1117/12.2232348.
Available at: https://doi.org/10.1117/12.2232348

Vaarandi, R. (2003). A data clustering algorithm for mining patterns from event logs. In: *Proceedings of the 3rd IEEE Workshop on IP Operations Management (IPOM 2003) (IEEE Cat. No.03EX764)*, pp. 119–126. Institute of Electrical and Electronics Engineering.
DOI: 10.1109/IPOM.2003.1251233.
Available at: https://ieeexplore.ieee.org/abstract/document/1251233

Xie, Y. and Tang, S. (2012). Online anomaly detection based on web usage mining. In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pp. 1177–1182.
DOI: 10.1109/IPDPSW.2012.143.

Xu, W. (2010). *System Problem Detection by Mining Console Logs*. Ph.D. thesis, University of California Berkeley, Berkely, California.

Xu, W. *et al.* (2009). Detecting large-scale system problems by mining console logs. In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, p. 117–132. Association for Computing Machinery. ISBN 9781605587523.
DOI: 10.1145/1629575.1629587.
Available at: https://doi.org/10.1145/1629575.1629587

Yamanishi, K. and Maruyama, Y. (2005). Dynamic syslog mining for network failure monitoring. In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, p. 499–508. Association for Computing Machinery, New York, NY, USA. ISBN 159593135X.
DOI: 10.1145/1081870.1081927.
Available at: https://doi.org/10.1145/1081870.1081927

Ying, J., Kirubarajan, T., Pattipati, K. and Patterson-Hine, A. (2000). A hidden markov model-based algorithm for fault diagnosis with partial and imperfect tests. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 463–473.
DOI: 10.1109/5326.897073.

Yu, S.-Z. (2010). Hidden semi-markov models. *Artificial Intelligence*, vol. 174, no. 2, pp. 215–243. ISSN 0004-3702.
Special Review Issue
DOI: 10.1016/j.artint.2009.11.011.
Available at: https://www.sciencedirect.com/science/article/pii/S0004370209001416

Zucchini, W., MacDonald, I.L. and Langrock, R. (2016). *Hidden Markov Models for Time Series: An Introduction Using R*. 150, 2nd edn. CRC Press. ISBN 978-1-4822-5384-9.