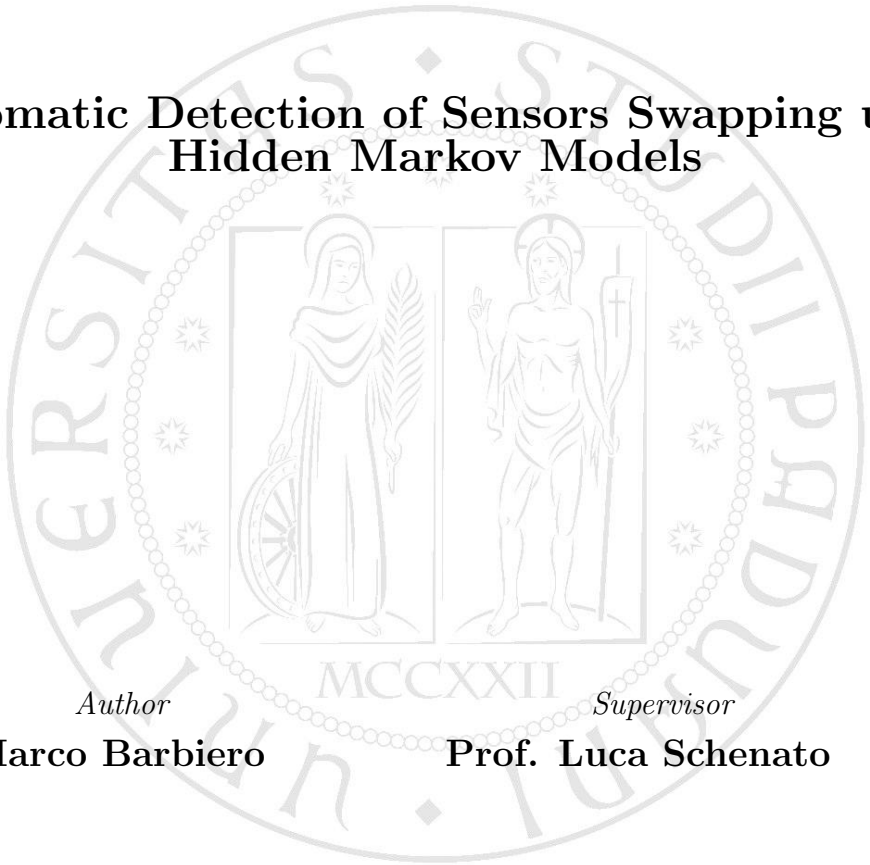
University of Padova

Department of Information Engineering

Master's degree in Automation Engineering

Automatic Detection of Sensors Swapping using Hidden Markov Models



Author
Marco Barbiero

Supervisor
Prof. Luca Schenato

September 11, 2018

Academic Year 2017/2018

Acknowledgements

The work presented in this thesis was carried out with the support of Professor Luca Schenato and Professor Ruggero Carli, Professors in the Department of Information Engineering. I am profoundly grateful to both for their help and concern, especially at times of difficulty when the problem seemed insurmountable and unsolvable, and the time was running out.

This work is also the concentration of knowledge and experience that I have gained over the last five years and more. For this reason, special deep thanks go to my parents, Mara and Tiziano, who have made this possible through their efforts, and to my brother Matteo, who has supported me in this long period of work. Furthermore, a very special gratitude also goes to my fellow classmates, Alessandro, Emanuele, and Maricarmen, who have proved through the last two years to be true friends.

I am thankful to all my friends too, that took care of me and supported me through all these years. In particular, heartfelt thanks go to Francesca, Irene, Dilan, and Matteo for providing me with unfailing support and continuous encouragement throughout these years. Finally, I am very grateful to all friends that are not mentioned here only for saving space: thank you for the time spent together. These moments have contributed to becoming the person I am, and for this, I am very grateful to you all.

Abstract

Nowadays, data covers an important part of our lives: a lot of studies are in progress and a lot of applications are being developed to improve its collection and management. This revolution touches all areas and forces them to revalue and redesign their functionalities: the lighting area is very active about this topic since lamps are very popular and diffused. Another big area of studies is energy saving: Europe 2020 program is near of its end but a lot of work has to be done to reach its objectives. A possible work field is the heating, ventilation, and air conditioning (HVAC) area: a lot of energy is utilised to air-condition building and houses but some of it is wasted. To solve this problem, an idea is to use sensors that monitor the presence of people within areas and connect these to the air conditioning systems. A smart way to include sensors in offices, places that are huge energy consumers, is to use smart lamps with a build-in proximity sensor and connectivity since they are widely diffused and usually positioned near the workers. Of course, the installation of the lamps is very important as well as the monitoring during time: indeed, these lamps must be installed and configured very careful not to make mistakes since lamps cannot be aware of their position and location naturally. This opus shows a possible way to implement an easy and efficient strategy to control the errors during the commission and monitor the lamps during the time. The office's workstations are modelled using hidden Markov Models whereas the Commission Errors Detection and the Failure Detection rely on custom algorithms developed taking inspiration from the Sequential Probability Ratio Test and the CUSUM algorithms. Finally, the two algorithms are tested in several different situations.

Sommario

Oggigiorno, i dati ricoprono un ruolo sempre più importante nella nostra vita: molti studi sono in corso e molte applicazioni sono in fase di sviluppo per migliorarne la raccolta e la gestione. Questa rivoluzione tocca tutti gli ambiti e li costringe a rivalutare e riprogettare le loro funzionalità: l'area dell'illuminazione è molto attiva su questo tema in quanto le lampade sono molto popolari e diffuse. Un'altra area di studi molto estesa riguarda il risparmio energetico: il programma Europa 2020 è prossimo alla conclusione ma c'è ancora molto da fare per raggiungere i suoi obiettivi. Un possibile ambito di lavoro è quello del riscaldamento, della ventilazione e della climatizzazione (HVAC): molta energia viene utilizzata per climatizzare edifici e case ma parte di essa viene sprecata. Per risolvere questo problema, un'idea è quella di utilizzare sensori che monitorano la presenza di persone all'interno delle aree e collegarli agli impianti di climatizzazione. Nel caso di uffici, uno degli ambienti più energivori, è sufficiente implementare dei sensori di prossimità all'interno delle lampade dal momento che quest'ultime sono molto diffuse. Le lampade devono, inoltre, fornire la connettività per lo scambio dei dati raccolti. Naturalmente, l'installazione delle lampade è molto importante così come il monitoraggio nel tempo: infatti, queste lampade devono essere installate e configurate molto attentamente per non commettere errori in quanto le lampade non sono naturalmente consapevoli della loro posizione. Questo lavoro riporta un modo possibile per implementare una strategia semplice ed efficiente per controllare gli errori durante la messa in servizio e monitorare le lampade durante il tempo. Le postazioni di lavoro dell'ufficio sono modellate utilizzando modelli di Markov nascosti (HMM), mentre il controllo di errori durante la messa in servizio (Commission Errors Detection) e il monitoraggio continuato nel tempo (Failure Detection) si basano su un algoritmo personalizzato sviluppato ispirandosi agli algoritmi Sequential Probability Ratio Test e CUSUM. Alla fine del lavoro, i due algoritmi vengono testati in diverse situazioni e i risultati riportati.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Nowadays, data covers an important part of our lives: a lot of studies are in progress and a lot of applications are being developed to improve its collection and management. This revolution touches all areas and forces them to revalue and redesign their functionalities: the lighting area is very active about this topic since lamps are very popular and diffused. Indeed, new light-systems are being developed that integrate sensing, control, and connectivity functions: more and more requests are being made about these features and the lighting area reacted developing smart lamps. These can be produced thanks to the Internet of Thing (IoT) revolution that brought sensoring and connectivity available to all since permitted to produce device with small dimensions and price. The Internet of Thing paradigm, as stated in [1], consists of including in a lot of things around us the capacity to communicate through Internet to provide information about us and them. Obliviously, this implies to overcome a lot of challenges since a huge number of different devices have to communicate.
Another big area of studies is energy saving: Europe 2020 program [2] is near of its end but a lot of work has to be done to reach its objectives. Indeed, in addition to economic manoeuvres and actions concerning school and research topics, this ambitious strategy also focuses on environmental and energy saving policies. The programme about these topics aims to achieve the 20-20-20 target, i.e. the greenhouse gas emissions 20% lower than 1990 levels, the 20% of energy coming from renewable sources, and a 20% increase in energy efficiency. The latter is the most difficult and quite far away: to achieve this, the EU must reduce primary energy consumption by a further 1.6 % over the six years from 2014 to 2020 [3]. A possible work field is the **heating, ventilation, and air conditioning (HVAC)** area: a lot of energy is utilised to air-condition building and houses but some of it is wasted [4]. For example, not all people that works in an office shuts off their conditioner when they leave or sometimes they cannot do even if they wanted to because the air-conditioning is centralised. To solve this problem, an idea is to use sensors that monitor the presence of people within areas and connect these to the air conditioning systems. A smart way to include sensors in offices is to use smart lamps with a build-in proximity sensor and connectivity since they are widely diffused and usually positioned near the workers.

1.1 Application

Implementing sensors in lamps is a good idea because lamps are widely spread in an office but, at the same time, they are positioned close to the workers, like the situation illustrated by Figure 1.1. It is essential for a successful data analysis

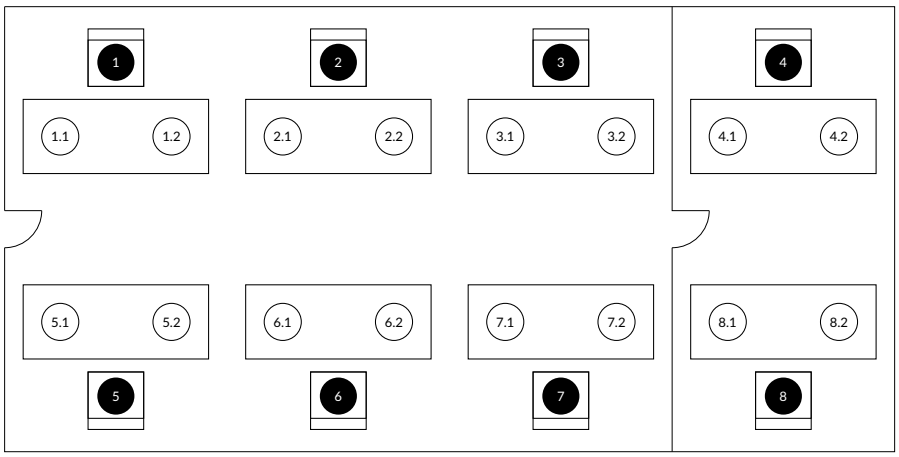


Figure 1.1: A stylised version of a office with twelve workstations. Each workstation is equipped with two smart lamps.

that the sensors are close to the people to be monitored. Indeed, the closer the sensors are to each other, the better performances can be achieved. That is why it is assumed to have two ceiling lamps hanging over each desk in order to monitor the person who is sit in front of the workstation. Having two lamps for desk is a good choice since it allows to have sensors redundancy to improve the quality of the measurement, and at same time, allows not to expend too much money on lamps and sensors like if buying three or more of them. Of course, the installation of the lamps is also of vital importance: indeed, these lamps must be installed and configured very careful not to make mistakes since lamps cannot be aware of their position and location naturally. A wrong installation could lead to several issues: the easiest to solve problem that can happen after a commissioning is that one of the sensors has not been connected. It is quite easy to solve since it is necessary only to test and monitor the sensors for a while and see if any of them do not send data. Silent problems, i.e. those that do not show malfunctions always but only in particular cases, are more difficult to detect and solve, instead. An example of silent malfunctions that often cannot be easily identified is a sensor positioned in a different position than that assigned. Indeed, this problem is difficult to find since it comes out not very often, no sensors are disconnected, and therefore it is difficult to find someone or something to blame: Let us think about a situation where two sensors should monitor a workstation in order to manage the air-conditioning but one is in a wrong place; finding that could be very tricky because, if it is assumed that, for example, the air-conditioning is activated when the two sensors see something within their surveillance area, the guy that sits in this workstation will never be able to identify the problem if the other is a very present worker. An even worse situation could occur in the case of worker monitoring without profiling, i.e. a people monitoring that is used only to obtain aggregated data and thus is

not linked to any specific worker. In this case, the data arrives at the analysis centre all together and it is not possible to trace which workstations produced them. Therefore, if there were an error in the commissioning of the sensors, no one would be able to detect it by looking only at the aggregated data, but it is still a big problem because it distorts the data. Let us think, for example, about the monitoring of presences in a library or store like an Apple Store in which products are seen and used over a desk: an error of that kind would lead to have irremediably compromised and useless monitoring data. In Figure1.2, an example of errors is reported.

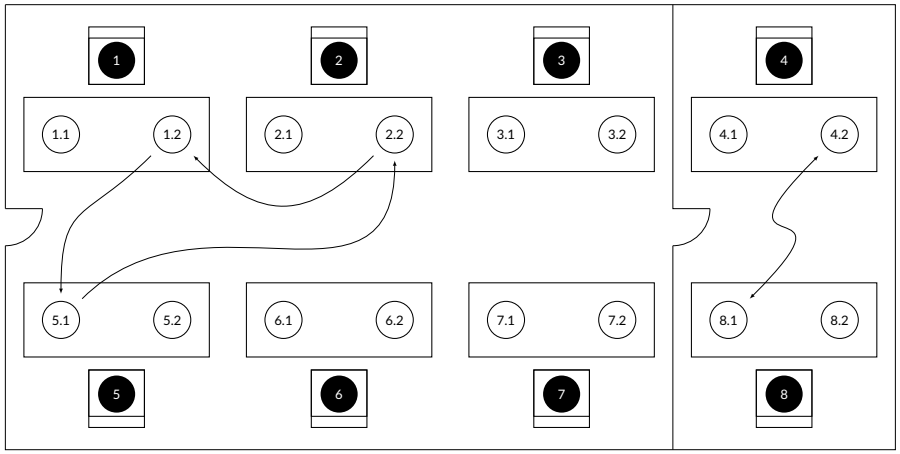


Figure 1.2: A stylised version of a office with twelve workstations. Each workstation is equipped with two smart lamps. Four mistakes are present.

From the examples above, it is clear that it is necessary to develop a suitable strategy to solve this issue, i.e. a plan of action to detect whether the commissioning was done in a good manner or not. The best idea could be to use an automatic identification tool to be installed in the lamp management system for a short period of time. This would be almost at no cost, it can be access to all data without leaks, and would not require new and expensive hours of work for the staff as in the case of rechecking all the lamps without any electronic system. In addition, with automatic systems it is possible to define an *a priori* accuracy of recognition, which is not possible with humans. This strategy can be called using a lot of different terms: in this opus, that branch is called **Commissioning Error Detection**. As it will be seen further on this opus, this strategy is based on checking which situation, correct or incorrect commissioning, is the most likely, i.e. has the highest probability. Of course, only the sensor data is utilised since they are the only available one.

Another similar problem is called **Failure Detection**. This consists of monitoring the sensors to see if they maintain performance over time or some difficulties arise. Indeed, offices and open to the public environments are very difficult to be controlled and therefore different things could happen to the sensors during time. Let us think for a moment about when refurbishment is done in a wide space: a lot of errors may occur when the lamps are refitted after being removed to allow paintwork of walls and ceilings or a space redesign. Another example could be tampering. Let us think, for example, of a laggard employee who does not want to

be monitored precisely or of a prankster worker who enjoys reversing the sensors of the lamps to make a mockery of the owner of the activity or of it boss. It is therefore necessary to provide for continuous monitoring and control of the sensors over time in order to be warned when any of these or similar situations occur. The strategy, as it can be clearly seen, is different from the problem above: in the other case, the goal of the Commissioning Error Detection is to give a report about which workstations work great and which ones have problems. In this case, the idea is to provide an alarm only when things start to go wrong instead of giving a report at almost regular intervals. Therefore, it is necessary to develop a quite different strategy to solve this even if the model to describe the two problems could be pretty similar. Like in the previous case, only sensor data can be used since they are the only ones available.

At the end of the day, the goal of this thesis is to develop some algorithms that allow to solve the two-previous illustrated problem: in other words, to check whenever sensors are installed in the wrong places or exchanged during time. As it was already shown, whether sensors that should look at same zones instead see different ones a lot of problem occurs. However, this is not an easy way and a lot of challenges have to be overcome, as it can be seen in the next section.

1.2 Challenges

A lot of challenges was encountered during the development of the solutions for the problems stated above: the main one was to develop a good model on workstation that allows to carry on the goal. Indeed, this is a not so widespread research topic and therefore, it is more difficult to find something ready to use. The situation can be described as a series of workstations where, in each one, there is a table and two ceiling lamps with an integrated occupancy sensor. Obviously, the presence of a person seated at the table can only be detected by analysing the outputs of the two sensors that look at the desk, since the actual presence is not directly accessible. To write down the idea used for this work in a more mathematical way, as can be seen later on, the **Hidden Markov Models** (HMM) tool was used. This is a very powerful mathematical tool that suits very well to this case. Indeed, it allows to model very well a workstation without neglecting the sensors and the fact that the position cannot be directly measured. The con is that this tool is quite difficult to understand, but, with the right tools, everything becomes much easier.

Other challenges to be faced are the ones about the sensors. Indeed, the modelling of an automation system depends a lot on the available sensors. Therefore, the latter are central to the creation of the model and must be analysed in depth in order not to create models that are too distant from reality. Obviously, the world is very complex and that is why it is necessary to put some stakes on how deep to go to describe the reality. Indeed, in our case different issues must be faced with regards to sensors: the first is that **sensors** are **noisy**, i.e. they are not perfect and sometimes give a different response from the correct one. Several causes can be found to explain this: the most important of these is the fact that the proximity sensors are based on electromagnetic waves and are therefore susceptible to thermal

noise and electromagnetic **interference**. These are widely present in an office due to PCs, printers, and all other electronic machinery, as well as the lamps themselves. Therefore, a strategy to face this must be developed: the faster is to use multiple sensors in order to reduce the noise. The second issue that it is present, is to have sometimes **cross-measurements**. Indeed, if two workstations are too close, it is possible that the sensor of one workstation measures the presence even in the adjacent one. This is a difficult problem to solve practically and for this reason it is better to model it as a generic noise and use some algorithms to mitigate it. The third issue is about the sensing area. Each sensor has a sensing area in which its performances are those described and guaranteed by the manufacturer. Outside that area, however, behaviour is not well defined, and may have different performance in different cases. In addition, the **sensing area** is not a discrete whole but a continuous one: that is, closer to the sensor, better performance, which decreases as you move away. Therefore, depending on the position in which the person is seated on the workstation, different performances can be achieved. This is very difficult to model and that is why in this work it was decided to assume that people sit perfectly in the middle of the desk, halfway between the two lamps. To reduce the load on this work, all these problems can be reduced to some probability of errors that will be included in the model parameters.

1.3 State of Art

A very little was done about the topic of detection commissioning errors or during time changes using sensor data in the lighting area as far as it is known. The main cause can be found in the fact that smart lamp area is a fairly new topic and that it has only begun to be developed in recent years. Moreover, installers are sometimes reluctant to introduce new installation methods and for this reason only recently, with the arrival of new genres of installers, the subject is beginning to be studied in depth. Most of the work in lighting commissioning area is about showing strategies to improve brightness or reduce energy wasting by better positioning of the lamps like in [5] and in [6]. In the first one a summary of energy management in lighting systems is done whereas in the second one proposes a method to estimate adjusted energy baseline using simulation models. Another useful article about this topic is [7]: in this paper, a study about the state of art is done along with a comparative between different approaches. The study was done very careful using very strict assessment criteria for the comparative. The ones that decide to investigate about using data in this field usually point to develop models based on statistic instead of using data coming from sensors. For example, in [8] a description about the light commissioning is done and some advice is shown to improve energy saving and comfort. Instead, in [9], an algorithm for the simulation of occupant presence is developed using inhomogeneous Markov chains. This paper is quite interesting since gives an idea on how to find parameters for modelling. About modelization of occupancy a lot work was done: for example, in [10] they proposed a model that can generate representative occupancy profiles in single office rooms using non-homogeneous Markov chains whereas in [11] authors summarised the current state of art about occupancy models. Furthermore, in [12], an idea about

how real time occupancy data from a wireless sensor network can be used to create occupancy models is shown. However, some works about using sensor data to model control lighting or HVAC systems are being done, like [13]. This article considers smart luminaries with an occupancy sensor and a lightness sensor, and tries to develop a strategy to achieve a designed illumination condition with low energy consumption. Another useful work is [14]: in this opus, the authors describe different lighting control strategies and their evolution with a focus on commercial office applications It is interesting also [15], which illustrates how fundamental are the occupancy sensors to obtain lasting results in the field of energy saving management from the point of view of air conditioning. Another interesting work is also [16] since it shows different types of sensors that can be used to detect people: they also develop a semi-Markov model to recognise people by pattern. The work presented here however assumes to receive binary occupancy data from sensors, but it is interesting to understand a possible way to implement it at low level.

All these works have been very useful to draw inspiration to solve the problems of Commissioning Error detection or Failure detection. Indeed, in many of these works, even if they do not deal directly with these problems, there are many ideas, sometimes just sketches, that have allowed to build something new. This work, therefore, takes inspiration from these ideas to develop a new vision.

1.4 Original Contribution

The final aim of this opus can be summarised in using two main targets: the first is to provide a model for the workstation behaviour to describe the commissioning error detection and the failure detection problems, but also to perform simulations. Indeed, simulations are the fastest way to test whether the algorithms work or not if a real office data is not available. The second target is to find some algorithms to solve the two issues in a fast and light way. Neither of the two targets had already been solved and for this reason it was necessary to create a large part of the opus from scratch.

The first original contribution brought by this opus is to use a hidden Markov model to create a model for the workstation behaviour: as stated in the state of art section, using this modelling idea is quite unusual in this field. However, it was decided to use hidden Markov models since they allow to provide a very tailored model of the problem: indeed, a stochastic model like this takes care about changing states as well as sensor behaviours using an all-in-one tool. Thanks to this tool, it was possible to create a fairly precise model of a workstation. As it will be seen later, after stating some assumptions, a workstation can be modelled with only two matrices and a vector if this type of model is used. The first matrix defines the behaviour of the states, i.e. how they change during the flow of time, whereas the second matrix defines the characteristics of the sensors using their probabilities of miss detection and false alarm. The vector instead encloses the initial state probabilities of the workstation. These elements can be easily built, either by using manufacturer's data for sensors and estimates for status change, or by using more complex algorithms, which will be explained in the further chapters. Indeed, hidden Markov models are widely studied and thus a lot of tools are available to analyse

them and to build something new. In fact, in this work, there are two models built for a workstation: as already defined above, the two possible behaviours for a workstation are correct and incorrect. The first occurs when the lamps are correctly associated with the table whereas the second occurs if at least one of the two lamps is not in the assigned place. The purpose of the work then becomes to find some tools that allow to define which of the two models, the correct one or the incorrect one, is currently the most likely in the workstation. It is necessary to highlight that there are two situations in which this test must be carried out: immediately after a commissioning or continued over time. In the first case, the problem is called commissioning error detection, while in the second, failure detection is mentioned.

The main original contribution made in this opus is to apply the sequential hypothesis testing to a hidden Markov models for both commissioning error detection and failure detection situations. This required a lot of study since there was nothing ready to use. Indeed, it was necessary to study the different methods of application of the sequential hypothesis testing and find those best suited to the situation: for the commissioning error detection the **Sequential Probability Ratio Test** (SPRT) was used, while for the failure detection problem the **CUmulative SUM** (CUSUM) was adopted. The main difference between the two methods is the amount of information available and therefore used. In the first case, the commissioning error detection, no information is available about the status of the workstation since the purpose is to determine if commissioning has been completed correctly or if any errors have been made. In the second case, instead, information about the correct commissioning is available and the purpose is to monitor the workstation to see if new problems can occur. For this reason, it was necessary to adopt two different algorithms to carry out the work. The biggest problem, however, was still open: both methods require a last-outcome dependent probability function, but the latter was not available by now. Indeed, it is quite complicated to derive a probability function from a hidden Markov model since the probability of states and observations varies according to the past. It was therefore necessary to build a function that simulates the behaviour of a probability distribution dynamically, and since it was not available, it was necessary to build it from scratch on the basis of the measurable and available values: it must be remembered that, in this work, only data coming from the sensors are available and, for this reason, all the functions based on the status value had to be discarded. The problem is more complex than it can be thought since the probabilistic description of the hidden Markov models makes wide use of the previous state, which is not directly available. It was therefore necessary to rewrite part of this description, removing as much as possible the dependency on the states and use estimators to make up for this. The mission has been almost completely reached but a small constraint has been necessary to introduce: the probability function used in the sequential probability ratio test and in CUSUM depends, in fact, not only on the last available measure, but also on all the previous ones, making the analysis of the algorithm performances a little more complex. However, it was possible to make both algorithms in-place, thus making a future implementation in commercial products possible. Indeed, the main purpose of this work is to provide some tools that can be easily implemented in commercial products: to do this, it is necessary that the proposed algorithms are fast and need a little amount of memory, since the devices that will have to accommodate these

algorithms are cheap and do not have large computational capabilities. However, the algorithms are also suitable for centralised use, where an external machine performs the algorithms instead of the individual workstations. Hence, even though the in-place implementation needs to be improved, the use of the two methods and the centralised approach allows to this opus to be an interesting innovation for the problem and to be adopted in the industrial environment.

1.5 Outline

This work can be divided in six parts: after a brief introduction, the formulation of problem is shown in Chapter 2 along with a possible modelization using sequential hypothesis testing. This part tries to expand the ideas presented in this introduction. The third part gives some statistic and mathematical complements to understand better the mathematical tools behind the idea: the hidden Markov models and the tree possible issues given sequences of data is illustrated. Furthermore, some useful algorithms that can be used to solve these types of problem are described. The Chapter 4 is the core of the work: in this part algorithms like the sequential probability ratio test and CUSUM are described along with their usage to implement this work. Furthermore, in this chapter, some problems encountered and their discovered solutions are described. Chapter 5 shows simulations performed to check the quality of the model and algorithms used in this work. Since algorithms have some tuning parameters called thresholds, comparisons between different values are shown to find the best ones. Furthermore, to facilitate the reader, different and innovative methods for showing the results are used. Last chapter summarises all and reports future improvements that can be implemented.

Chapter 2

Problem Formulation

In this chapter, the main problems and the ideas it was decided to adopt for the workstation models are illustrated. To be more precise, initially a detailed description of the problems will be made: after that, a brief introduction about occupancy sensor is made while, afterwards, the assumptions used will be shown along with the notation adopted. Eventually, the idea that will be pursued is shown.

2.1 The Problems

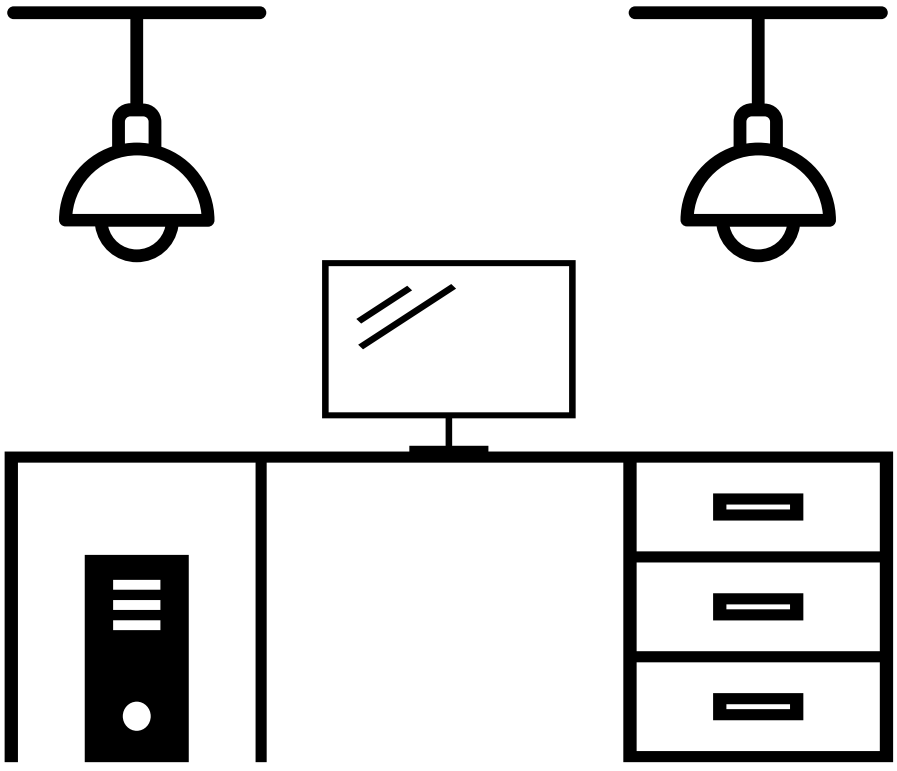


Figure 2.1: The situation illustrated. Two smart lamps watch the same workstation. Front view.

There are two main problems that this work is aimed to solve with: the detection

of commissioning errors and the identification of failure during time. These two problems, however, are very much related to each other since they both occur in the same environment, namely a workstation. As it can get guessed, the work is primarily aimed to be implemented in work environments such as offices albeit it can be adapted to any environments where people sit in front of a table, such as libraries or even restaurants. Even modern stores like the Apple Stores, where products are placed on tables to be tested, can adopt this idea for monitoring customers and understanding which products are more desired than others. Although this is true and start to spread, in this work the whole of the desk and the lights connected to it will still be called workstations, even if, as already mentioned, the idea can also be exported outside the strictly working environment.

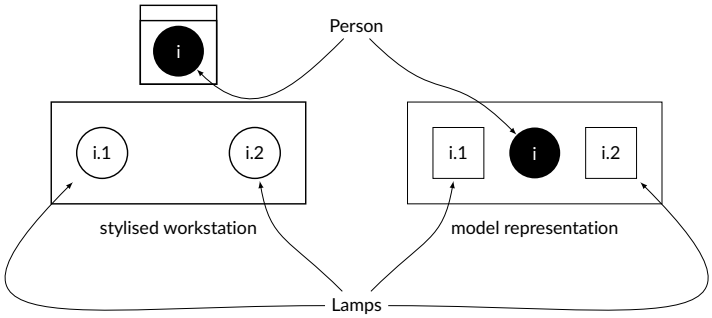


Figure 2.2: The situation illustrated. Two smart lamps watch the same workstation. Bird's-eye view.

Figure 2.2 shows a currently situation in a standard office from a bird-eye view: every workstation is composed of a pair of ceiling lamps that illuminate each desk. To make the design more agile and clear, from now on the model will be represented using the right-hand model in that figure. Usually, nobody cares too much about lamps except when they break, and people are shrouded in darkness. In this work it was decided to concentrate the focus on them and adding to them something new: it is assumed these lamps are smart type ones, i.e. they have a built-in proximity sensor that allows them to identify whether a person is sit at the desk. This is not a strange request since nowadays several lighting producers provide lamps like the ones are required for this project. Indeed, some new regulations require that the lighting level must be controlled to always be above of a minimum brightness threshold, so as to provide an adequate level of comfort to the worker. Furthermore, to improve energy saving and comfort of workers, some companies start to monitor workers' behaviour to switch off lights and air-conditioning when they leave. Therefore, these smart lamps are the answer and they start to spread across offices.

The problems that this opus sets out to solve, i.e. identify whether there were any errors in the commissioning of the workstations or whether they become unreliable over time, have a common work base and they are equally important. Indeed, the data collection is meaningless if data gathered are not reliable and therefore is very useful to have an alarm about this. The target of this work is to provide a tool that shows if data is reliable and, if it is not, when the issue started. However,

as these are two problems, it will necessary to provide two different algorithms. Nevertheless, it is important to find similarities between the two problems in order to reduce the workload. The main thing in common between the two issues is the environment to be monitored. Indeed, in both, there are workstations like those illustrated above.

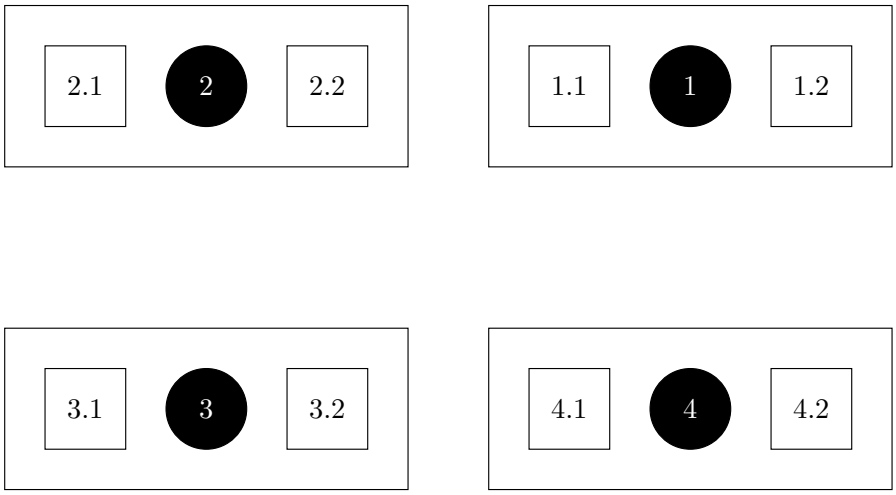


Figure 2.3: A common office with four workstations: Bird's-eye view.

In Figure 2.3 is illustrated a common situation in which this opus can be applied: an office where there are a lot of desks like the ones that have been shown in Figure 2.1. Each desk has two smart lamps over it and a number; each lamp is called using the number of the desk along with one or two: sitting in front of the desk, the utilised convention is to use number one for the left-hand-side lamp and the number two for the right-hand-side one. At the end of the day, each desk numbered i has two lamps called $i.1$ and $i.2$. In the case illustrated, two desks are shown with numbers one and two: therefore, desk one has lamps 1.1 and 1.2 whereas desk two has lamps 2.1 and 2.2. These numbers are assigned during the commissioning of the office, when people assembles desks, connects PCs, and installs the lighting. The issue occurs when two sensors that belong to different desks are put together over the same desk, and the aim of this work is to develop a solution for recognising this. Several reasons can be found to explain that: for example, a worker could swap a pair of lamps because one of his is broken or for luddism. Another reason can be mistaken during installation after an office renovation. Another one may be a distraction error committed by the installer during commissioning. From these examples, it is clear to see that there are two big situations where that could happen: indeed, as already stated before, errors can be committed during commissioning or later during time. The main difference between these two cases is that, in the first,*a priori* information is not available whereas, in the second, it is aware that commissioning has been done in a comprehensive manner, but it is required to monitor if some changes occur. Before to illustrate these two approaches, it is better to fix some assumptions to reduce the magnitude of the problem.

2.2 Modelization and Assumptions

As it can be seen, the problem is quite wide since a lot of different variables are present in a workstation. The main ones are about sensors and people: indeed, a lot of different sensors can be used as well as the person behaviour can be very variable from person to person. Therefore, it is essential to rejuvenate the working field and thus make assumptions. But, before to do that, it is better to illustrate what is an **occupancy sensor** and a **proximity sensor**.

2.2.1 Occupancy & Proximity Sensors

An occupancy sensor is a device that is used to detect the presence of a person in a defined environment, usually a room. The target of this type of devices is to determine whether electrically powered load should be turn on or not. These are very useful in public environments where visitors do not take responsibility for their energy consumption, and where they often do not take great care to be diligent in switching off electronic equipment when not in use. Therefore, the final target of occupancy sensor installation is to improve the energy saving, and this has led some countries to oblige to install them in public spaces.

Two are the most prevalent types of occupancy sensors [17]: the **passive infrared** (PIR) and the **active ultrasonic** type. A passive infrared sensor will turn on, i.e. a high output is provided, whenever it detects a heat source that is moving or is barely visible, whereas an active ultrasonic sensor exploits the Doppler effect: it emits ultrasonic waves at frequencies of 25 kHz or higher, in order not to be heard by human beings, and it listens to the return echoes. If it detects a significant Doppler shift, it is expected that a moving body is present, and the sensor will turn on. In both of cases, a motion sensitivity threshold can be set by the user as well as the duration of the high signal provided by the sensor for each detection.

Both of the types of sensors have cons and pros: the passive infrared one may not be able to detect people behind barriers like resting-room modesty panel, and they maybe have some **dead spot**, i.e. some areas in which they are less sensitive to heat sources than the whole of the environment. The active ultrasonic sensors are resistant to these disturbances since the sonic waves emitted will permeate all the environment. However, this type of sensors has cons too. Indeed, they are susceptible to false triggers from a variety of sources: if the sensor is placed on a wall bordering another room, some of the emitted waves may go to the other room and give false alarms in someone there is moving. Furthermore, wind and air-conditioning can be detected as moving people if the sensibility is set too high. Usually, in both of the types of sensors, some electronics is put in support of these sensors to expand signals and clean up noises as well as to manage sampling time. In recent years, to improve the capabilities of occupancy sensors, both of types are often used in pairs in order to exploit the best features of them.

Similar to them, but with a shorter working range, are the proximity sensors. The operating principle is very similar in the case of those used to identify people, but usually, they have lower power due to low working range. However, in common language, the two terms are often used to mean the same object. Eventually, in

this opus, a generic occupancy sensor that merges the two types is used since it allows to simplify the description without loss of generality.

2.2.2 Assumptions & Notation Adopted

Several assumptions were done to simplify the work, but it has always tried to make them as general as possible in order to make this work reusable. Some of them are used in order to circumvent technical problems, such as not having smart lamps available. These types of assumptions can be easily dropped if some tests on the smarts lamps are done, and, thus, a data description of a smart lamp is available. Part of these assumptions were also done to simplify the work description, but they can be freely dropped at the cost of greater notation complexity. The other ones, instead, are the result of the nature of the problem, and, therefore, it must be dealt with, like the ones about sensors. The main of this work are the following. The first one is about the state of a workstation: it is defined as the state of the worker assigned to it and:

Assumption 1. *Workstation has two possible states: worker is absent, or worker is present.*

The state is binary and then, the most natural way to represent it is using $\{0, 1\}$ digits. In this work, to indicate the state of a workstation, it is used the notation $x_i(t)$: this refers to the state of workstation i at time t. As it can easily be guessed, if $x_i(t) = 0$, it means the worker is absent whereas, if $x_i(t) = 1$, the worker is present ahead of the workstation. Another assumption about workstations is:

Assumption 2. *Workstations are slow dynamics systems, i.e. changes of states are slow.*

This is a natural assumption that derives from the nature of the problem since the work is related to people: indeed, in offices, workers usually change their state, i.e. they sit on their workstation or they have left, very rarely for a machine's point of view. Then, it is expected that probability to change states will be low whereas the probability to stay in the same state will be high. Furthermore:

Assumption 3. *Workstations are time-invariant systems, i.e. probability of change states does not vary during time.*

This assumption is quite feasible since people are fairly regular and predictable beings. However, this assumption simplifies a lot the work since allows to use constant values with regard to time. Another simplifying assumption is:

Assumption 4. *Workstations presences or absences are independent between workers.*

This is a very strong assumption and it could be not always verified in all offices because of arriving, leaving and break times are usually the same for all the workers. However, it should be remembered that, in modern offices, people often get up, go to the bathroom, welcome a customer, or retrieve old documents, i.e. they often leave their workplaces and then return: so, this hypothesis is not completely

strange, and it could be used without reducing the possible applications of this work. At a closer look this idea would seem to clash with the Assumption 2 about slow dynamic but these movements are seen as very rare by the sensors because of their very small sampling time. Another assumption about the operation of a workstation is:

Assumption 5. *Workstation state is not directly available, but it is measured by a pair of sensors.*

This is a natural assumption since the only way to identify the workstation state is to use data provided by the sensors that are built-in into the lamps watching the desk. Sensors are not 100 percent reliable because of disturbances and noise: this leads to modelize the problem as a hidden state problem. It is useful to define a notation for sensor's outcomes. A good way is to use $y_j^{(i)}(t)$ for sensor $i.j$ at time t. Like for the state case, sensors' outcomes are binary: 0 means sensor has not detected any presence whereas 1 means sensor has detected a person in its detection area. It is also useful to provide a compact notation to return the two outcomes of the lamps belonging to the same table: in this case, $y^{(i)}(t)$ is adopted. Since $y^{(i)}(t)$ encloses measurements coming from two different sensors, it can assume four values: to simplify the implementation, it was decided to use the binary code to represent them, convert it to decimal one and then add to it one to work with non-zero values. The first digit contains the value from sensors $i.1$ whereas the second stores measurement coming from sensor $i.2$. To be clearer, an example is provided: let $y_1^{(i)}(1) = 1$ and $y_2^{(i)}(1) = 0$ be; in this case $y^{(i)}(1) \rightarrow 10_2 \rightarrow 2 \rightarrow +1 = 3$. In Table 2.1 all possible examples are reported.

Table 2.1: Notation examples about $y^{(i)}(t)$ with regards to sensors $i.1$ and $i.2$

$y^{(i)}(t)$	$y_1^{(i)}(t) = 0$	$y_1^{(i)}(t) = 1$
$y_2^{(i)}(t) = 0$	1	2
$y_2^{(i)}(t) = 1$	3	4

Other assumptions are needed to define the occupancy sensors. The most important can be stated as:

Assumption 6. *States and measurements are discrete-time values.*

This is a forced technical assumption: indeed, sensors have a sampling time and then measurements are discrete-time values. About states, the issue is different: states are continuous-time variables since they can be defined for all possible instants of time. To simplify the work, it is also assumed that states are discrete-time variables and that they have the same sampling time as sensors' measurements: this sampling time is called T_s. Using discrete-time variables for states, it is not a problem if T_s is small enough: indeed, a continuous-time function can be approximated as well as desired using a discrete time series with adequate sampling time. A useful assumption to better outline the model is:

Assumption 7. *An occupancy sensor in a workstation is completely described by probabilities of miss detection p_m^s and of false trigger p_f^s.*

The aim of this assumption is to reduce the complexity of sensors: as already stated, occupancy sensors have some problems, like false triggers or dead spot. Therefore, it is useful to describe all of these problems using only two values: the first, p_m^s, is the probability of miss detection, i.e. the probability of the sensor to give a low output even if a person is present. The second, p_f^s, is the probability of false trigger or false alarm, i.e. the probability of the sensor to give a high value as output even if nobody is in there. Using these two probabilities, it is possible to create an observations matrix for the pair of sensors in a workstation without too much effort as can be seen in the Chapter 3. Indeed, in a more mathematical way, the two values represent:

$$p_m^s = \mathbb{P}\left[y_s(t) = 0 \mid x(t) = 1\right] \qquad \text{and} \qquad p_f^s = \mathbb{P}\left[y_s(t) = 1 \mid x(t) = 0\right]$$

and, as it will be seen, they are the building blocks to build the observation matrix. Thanks to these assumption, now it is possible to model better the problem.

2.2.3 Problems Modelization

In this part, it is possible to forget about the two small problems previously stated, i.e. the commissioning error detection and the failure detection: both of them share the same model and the possible situations because their only difference is the time in which the analysis is done and what data is available. Indeed, the first is done without any information whereas the second is performed using already available data. Therefore, for building a model of the situations, the type of analysis it is wanted to be done is not important.

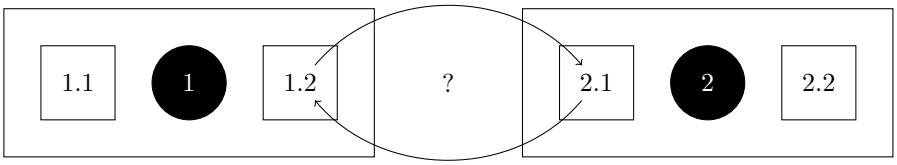


Figure 2.4: The fundamental problem illustrated. A pair of sensors is divided and one of them is taken to another place.

Furthermore, to simplify the description, it is decided to focus on workstation 1 and 2, i.e. it is assumed that an office contains only two workstations. Since the problem and the algorithms that will be proposed are distributed, the solution can be extended for each desk available without any effort. At the same manner, it is also decided to have lamps 1.1, 1.2, 2.2 and 2.1: the first two are normally present on the desk 1 whereas the other ones are usually placed over desk 2. To reduce the number of possible swaps, it is decided to focus on the swap between lamp 1.2 and lamp 2.1. This is not a limitation because, as it can be seen later, the algorithms presented are independent with regards to which lamps have been swapped.

In light of what it is stated above, two possible cases are available in this situation with regards to desk 1:

$$\begin{cases} \theta_0: & \text{Lamps 1.1 and 1.2 are present over desk 1} \\ \theta_1: & \text{Lamps 1.1 and 2.1 are present over desk 1} \end{cases}$$

It is worth reminding that, for desk 1, the only available measurements come from sensors in lamps 1.1 and 1.2. So, for case θ_1, only one measurement regards the current workstation whereas the other considers an unknown desk. In Figure 2.5

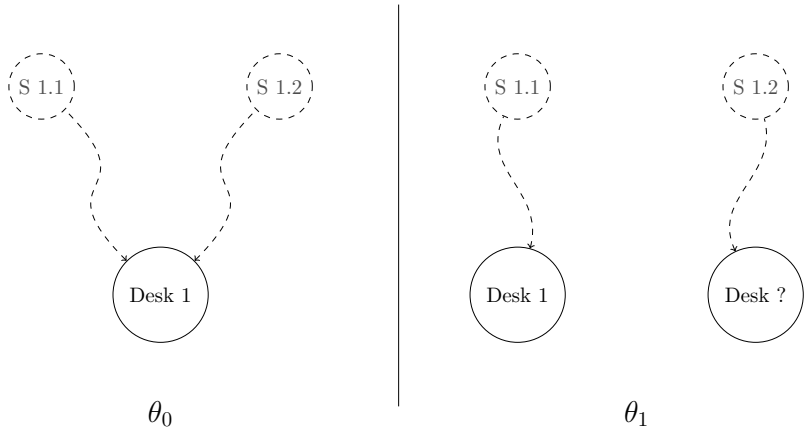


Figure 2.5: The two possible situations illustrated.

are represented the two situations in a fancier way: θ_0 (left-hand side) and θ_1 (right-hand side). In that picture, smart lamps are identified using S $i.j$ following the notation defined above. The issue now is how to express this problem in a more mathematical and statistical way exploiting the two different cases.

Taking inspiration from Figure 2.5, the problem can be rewritten focusing on the behaviour of the two observations, S 1.1 and S 1.2, and the links between them in the two different hypotheses. In the first case, θ_0, it can be seen the two measurements come out from the same desk, i.e. they are correlated with each other. In the other case, θ_1, the two measurements come out from two different desks. From Assumption 4, the two desks are independent and then it is possible to define that, in this case, the two measurements are independent from each other. It is worth reminding that, since they are independent by construction, they maintain these proprieties regardless of the temporal instant. Therefore, the two situations can be rewritten, in a more mathematical way, as follows:

$$\begin{cases} \theta_0: & \text{Measurement from 1.1 and 1.2 are correlated} \\ \theta_1: & \text{Measurements from 1.1 and 1.2 are independent from each other} \end{cases}$$

Now the issue is about how-to condensate that information in a tool that can analyse and give a response about which hypothesis is true. It is worth highlighting that this tool can only access observation data whereas it cannot check the workstation state. Furthermore, this operation must be done for the two different detection problems, even if a huge part of the work can be shared among them. Before trying to find a tool that allows to perform this, it is better to take some time to create and describe a model for the desk system. It is necessary to develop a mathematical model of a workstation which takes into account the presence of

inaccessible conditions and noise-dirty observations. The most indicated tool for this work are the Markov chains and the hidden Markov models, as it can be seen in the following Chapter 3.

Chapter 3

HMM and the Model

In this chapter, a brief description about Markov Chains and Hidden Markov Models is done. After that, the three problems that can be solved using the Hidden Markov Models are shown along with some useful algorithms that can be used to resolve these issues. Thus, using that knowledge, the model adopted for this work is derived and described.

3.1 Markov Chains

A Markov Chain is a particular type of discrete dynamic system firstly developed by Andrey Markov in the early 20th century. It is a system with a finite[1] number of states and where the transition between different states is driven by probabilities instead of relying on deterministic functions. To be more precise, a Markov chain $\mathcal{Q}$ gives a theoretical model to describe the behaviour of a discrete-time system, i.e. in which state the chain is. In this work, a model with two states is sufficient since only two possible position of worker are available, as stated in Assumption 1. Thus, it can be defined the set of the $N = 2$ available states, $\mathcal{S} = \{s_1, s_2\}$. Since, in this opus, states are absent or present, the most intuitive idea is to use binary values to express the elements in $\mathcal{S}$. Thus:

$$\mathcal{S} = \{s_1, s_2\} = \{0, 1\}$$

As already stated above, to describe the jumps between states, a probability values are required. Different ideas could be used to define them; to reduce this uncertainty, the main hypothesis of the Markov chains must be disclosed:

Assumption 8. *The chain state at time $t + 1$, $x(t + 1) = x_{t+1} \in \mathcal{S}$, depends only on the system state at time t, $x(t) = x_t \in \mathcal{S}$, and on the transition probabilities.*

That assumption, called **Time-homogeneous property**, states that it is needed to provide only probabilities of transition between states regardless what happened before. In other words, the system relies only on the current state to choose the next step and can forget what happens before. To simplify the notation,

[1]Markov chain theory also works with a numerable infinity of states, but it can be freely omitted as only a finite number of states are required for this work.

time-invariant probabilities are assumed: this assumption could be not required to perform the algorithm. In a more mathematical way, the Assumption 8 can be rewritten as:

$$\mathbb{P}[x(t+1) = s_i \mid x(t), x(t-1), \ldots, x(0)] = \mathbb{P}[x(t+1) = s_i \mid x(t)] \quad \forall\, t > 0$$

It is clear that, for this work, only four probabilities of transition must be defined since only two states are available: the notation for the probability of transition from state s_i to s_j adopted for this opus is:

$$a_{ij} = a_i(j) = \mathbb{P}[x(t+1) = s_j \mid x(t) = s_i]$$

For speed up the reckons, it is useful to write these probabilities in a matrix: using i to identify the row and j for the column, a transition matrix $A \in \mathbb{R}^{N \times N}$ can be obtained from these probabilities. It is worth noticing that this matrix is a row-stochastic matrix since the sum by row is equal to one for each matrix row: this is obviously since the sum of complementary[2] events probabilities is equal to one. A row vector like this is called **stochastic vector** and it is used to express the probabilities about the chain to be in a state for all possible states at time t. For example,

$$\pi(t) = [\pi_1(t)\, \pi_2(t)]$$

express that, at time t, the probability to be in the first state is $\pi_1(t)$ where as to be in the second is $\pi_2(t)$. Obviously, this works for all possible N, but it is decided to focus only on the binary case, since it is the one interesting for this opus.

Let $\pi_0 \in \mathbb{R}^{N \times 1}$ be the initial probabilities: if it is assumed that the initial conditions are known, then $\pi_0 = e_i^\intercal$ for a particular i. Remember that e_i identifies the all-zero column vector except from position i where there is a one. Furthermore, $\pi(1)$ can be defined as a row vector that contains the i-th row of matrix A: that row contains the transition probabilities given the fact that the current state is s_i. In matrix form:

$$\pi(1) = \pi_0 A$$

At the same manner, $\pi(t)$ can be calculated for all t higher than zero, i.e.:

$$\pi(t+1) = \pi(t)A \tag{3.1}$$

this is the **probability update rule** for the Markov chain. A different way to express the behaviour of the chain stated in 3.1 is:

$$\pi(t+1) = \pi(t)A = [\pi(t-1)A]A = \cdots = \pi_0 A^{t+1}$$

This formula prompts to investigate about what happens when t is closer to infinity: before to find that, more information about stochastic matrices are required. This formula also suggests that the only two things that identify a Markov chain given a set of state is the initial probability π_0 and the transition matrix A. Thus, a compact way to describe a Markov chain is

$$\mathcal{Q} = (A, \pi_0)$$

[2]Two events are said to be complementary when the occurrence of one excludes the occurrence of the other but one of the two will certainly occur.

Row-stochastic matrices, also called right-stochastic matrices, have several properties [18, ch. 3] that could be very useful: the first is, since in this work $N = 2$, A can be written as:

$$A = \begin{bmatrix} a_{11} & 1 - a_{11} \\ 1 - a_{22} & a_{22} \end{bmatrix}$$

i.e. there is only two parameters (a_{11} and a_{22}) available to define this matrix and they must be positive values lower than 1. It is decided to drop null probabilities because in this opus are not required and this operation allows to work with positive matrices and to use some useful properties. Furthermore, this allows to call the chain **regular** without doing some complex reckons: regular chains require that A matrix is primitive but, since positivity is a stricter feature, the whole thing can be solved much faster. The second property is that a row-stochastic matrix has a got a right eigenvector equal to a column of ones and the respectively eigenvalue is 1. Indeed, right multiplying a matrix by a column of ones means to sum by row. Furthermore, it can be proven [18, Lemma 4 p. 120] that all other eigenvalues have magnitude lower than 1. In this work, it is used row vectors to express probabilities. Thus, it is useful to have some properties about left eigenvectors. As stated before, A is a positive matrix: this means that it is primitive, i.e. $\exists k : A^k > 0$, and irreducible, i.e. $\sum_{k=0}^{N-1} A^k > 0$. Therefore, the **Perron-Frobenius** theorem can be used: this theorem states that, in this case, there is only one dominant eigenvalue, i.e. only one with maximum magnitude, and eigenvectors are unique and positive. The most interesting eigenvector is the one associated with the maximum magnitude eigenvalue that, from what it was written above, is equal to one. Indeed, it can be proven [19, p. 456] that, if w_1 is the left eigenvector associated with eigenvalue 1:

$$\lim_{t\to\infty} \pi(t+1) = \lim_{t\to\infty} \pi_0 A^{t+1} = w_1 \tag{3.2}$$

This is the **asymptotic probability distribution** of the Markov chain associated to the matrix A and it allows to describe the behaviour of the chain after the transitional period has elapsed.

There is another way, more visual, to describe Markov chains and it is to use graphs. An example can be seen in Figure 3.1.

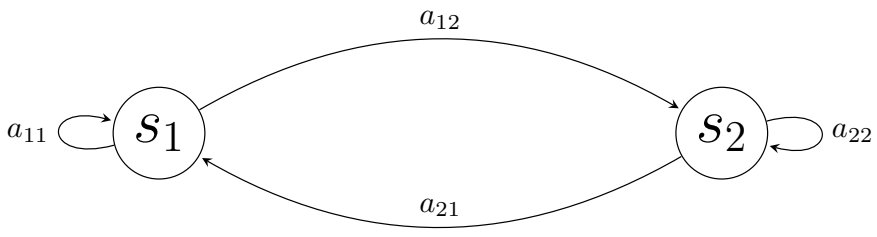


Figure 3.1: The graph associated to a binary Markov chain.

A graph is a pair composed by a set of nodes, in this case $\mathcal{S}$, and a set of vertices that link nodes. To be more precise, the Markov chain above described can be represented using an oriented connected weighted graph: indeed, it is connected

since exists a path that connects every node, it is oriented because vertices have a starting and an ending node, and it is weighted since all of the vertices have a weight. The weight represents the probability that a transition can occur if the current state is the starting node and the future state is the other where the edge lays. If, as a node set, the state set of the Markov chain is used, the graph obtained contains all the information that contains the pair $(\mathcal{S}, A)$. This type of representation is widely used because it allows to understand better the behaviour of the chain, which can be difficult to clue at with a glance when using only the transition matrix.

The theory described above would be perfect if it were possible to have a direct access to the state of the chain, i.e. if the sensors could not make any mistakes. This cannot be achieved, however, and it is therefore necessary to extend this theory slightly to include observations. To resolve this issue, statisticians and mathematicians developed an extension of Markov Chains, called **Hidden Markov Models**.

3.2 Hidden Markov Models

A hidden Markov model, in a nutshell, is a Markov chain where states cannot be directly observed: in this case, only events about states can be observed. Indeed, in the hidden Markov model theory, the chain is modelled, as a common Markov chain but every state, when it is reached, generate an event that depends only on the probability distribution and the state. In Figure 3.2 a graphical representation of a binary hidden Markov model with four observable events is shown.

In this Figure, events are represented as circles filled with dots and states are to them using dotted arrows: the weight of those arrows represents the probability that the event happens given the current state.

From a more mathematical point of view, the additions with respect to the standard chains can be summarised as follows. Let $\mathcal{O} = \{w_1, w_2, w_3, \dots, w_M\}$ be the set of the M observable events: the probability that an event is observed given the current state can be expressed as:

$$c_{ij} = c_i(j) = \mathbb{P}[y(t+1) = w_j \mid x(t) = s_i]$$

where $y(t)$ is the event observed at time t. It can be noticed that the probability of an observation depends only on the current state and it is independent of previous events observed or states. This concept can be summarised in the following assumption:

Assumption 9. *Given the state $x(t)$, the observation $y(t) = y_t \in \mathcal{O}$ is independent of other observations and states and depends only on current state $x(t)$.*

Similar to what was done above, event probabilities are assumed time-invariant and they can be written in a more compact way to build the **observation matrix** C. Obviously, the C matrix is row-stochastic and non-negative. Using that one, a simple rule to express probabilities of events is:

$$\sigma(t+1) = \sigma(t)C \tag{3.3}$$

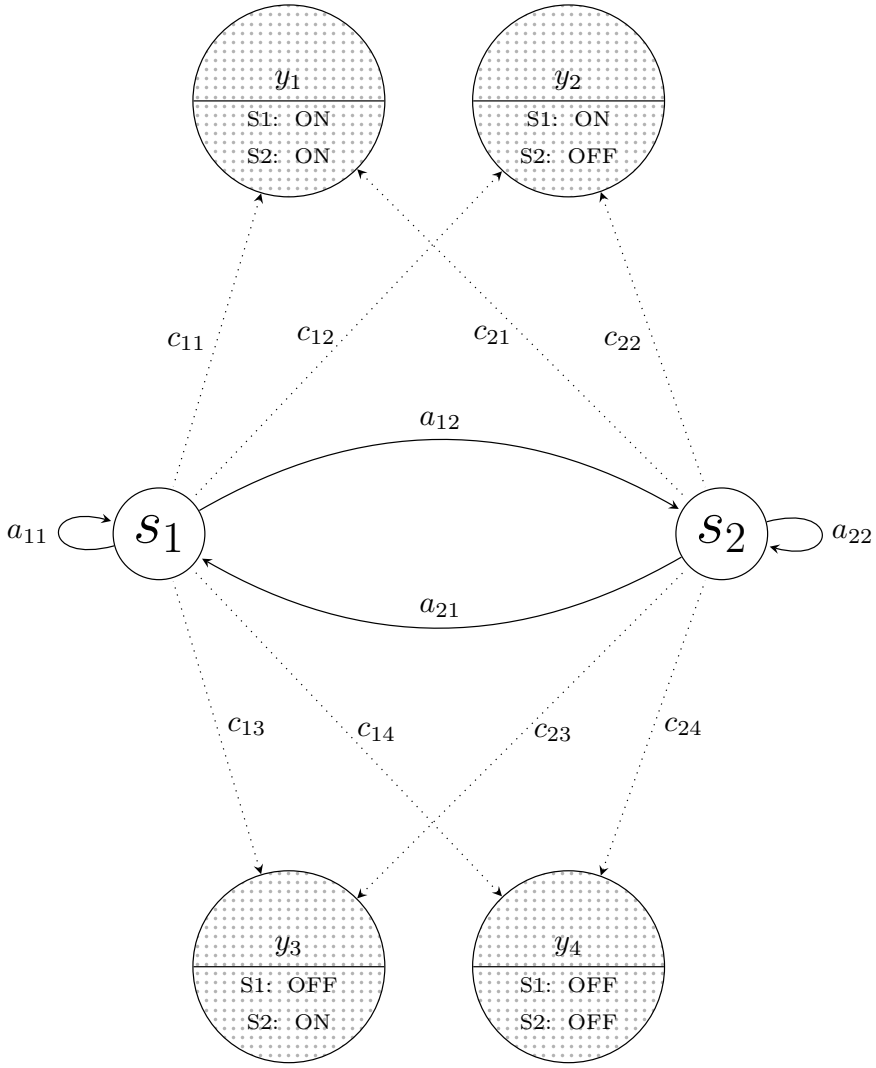


Figure 3.2: The graph associated to a binary Markov chain with four observable events detected by two sensors.

where $\sigma(t) \in \mathbb{R}^{M\times 1}$ is a row vector that contains the probabilities of all events at time t, i.e. $\sigma_i(t) = \mathbb{P}[y_t = w_i]$.

At the end of the day, a hidden Markov model can be summarised as a stochastic system described by a transition matrix and an observation matrix where their states and observations probabilities evolve following formulae 3.1 and 3.3. Similar to what was done in the Markov chain section, a hidden Markov model given the states and observations sets can be represented as:

$$\mathcal{Q}_h = (A, C, \pi_0)$$

Even if it is quite simple to be described, this theory allows to solve a lot of issues: the three fundamental ones are the **three HMM problems**.

3.3 A different approach to describe the probability behaviour of a HMM

To simplify the description of the three problems, a good idea is to do some cosmetics improvements to the formulae 3.1 and 3.3. The final aim of this subsection is to describe the probability behaviour of a hidden Markov model using analogies with the **Kalman filter**. This filter is widely diffused to estimate unknown variables in the continuous-space and then making comparative with it allows to facilitate the comprehension[3]. The main concepts that can be copied are the **prediction step** and the **update step**. Indeed, a simple description for the probability of states is required and a working idea can be to copy the concept of prediction-update steps. For example, in the Kalman filter, for each sampling time two operations are performed: the first, called prediction step, tries to predict the state using the previous estimation whereas, the second, tries to estimate the state comparing the prediction performed in the previous step and the measurements that are collected in that time interval. It would be nice if it were possible to reduce the behaviour of state probabilities in a hidden Markov model to such a simple system. As it will be seen, this can be done in hidden Markov models too. But, before to show the algorithms, it is better to illustrate some mathematical complements that will facilitate the comprehension.

3.3.1 Mathematical Complements

First of all, it is worth reminding that $a_{ij} = a_i(j) = \mathbb{P}[x_{t+1} = s_j \mid x_t = s_i]$ whereas $c_{ij} = c_i(j) = \mathbb{P}[y_t = w_j \mid x_t = s_i]$. Another two things that will be useful are the **Bayes' rule**:

$$\mathbb{P}[x, y] = \mathbb{P}[x \mid y]\mathbb{P}[y] = \mathbb{P}[y \mid x]\,\mathbb{P}[x] \tag{3.4}$$

and the **marginalization rule**:

$$\mathbb{P}[x] = \sum_{y \in \mathcal{O}} \mathbb{P}[x, y] = \sum_{y \in \mathcal{O}} \mathbb{P}[x \mid y]\,\mathbb{P}[y] \tag{3.5}$$

Before starting with the maths, a good idea is to define some notations that can be useful. As a convention, it was decided that:

- T means the last time in which an observation was obtained, i.e. all data available are $Y_T = \{y(1), \ldots, y(T)\}$. In this work, the first sample is obtained in $t = 1$; this choice was done to maintain compatibility with the MATLAB toolbox that indexes arrays and vectors from 1.
- All vectors are row vectors if not differently stated. This is a common choice in statistics but a quite strange one in Engineering. Anyway, this means that left matrix multiplications are performed instead of the common right ones. To reduce the possibility of committing misunderstandings, the time-depending values are stored by columns. It could be seeming topsy-turvy,

[3]The Kalman filter knowledge is not required to understand the following parts.

but this convention allows to reduce mistakes when these algorithms will be implemented in a computer. Indeed, a real variable is stored in a column, and then each row represents a sampled-time interval, whereas a vector variable is stored in a matrix: in this case, the i-th row contains the vector at the i-th sampled-time interval.

- In this opus, the **element-wise multiplication** is indicated with $\odot$ and follows the same behaviour of the `times` MATLAB's function[4]. The most used feature is vector by vector element-wise product: this allows to compact the following:
$$a(i) = b(i)\, c(i)\, \forall i \in [1 : I]$$
into
$$a = b \odot c, \quad a, b, c \in \mathbb{R}^{1\times I}$$
The latter operation is much faster than the for-loop in MATLAB.

- The standard notation adopted henceforth for a hidden Markov model is $\mathcal{Q} = (A, C, \pi_0)$. The subscript $_h$ has been removed to facilitate notation and to speed up the writing. No misunderstanding with Markov chain can occur since they are no longer used in this opus.

- $\hat{P}_{\tau|T}(i)$ is the probability that the chain was, is or will be in the state s_i at time $\tau > 0$ given the observations from 1 to $T > 0$, i.e. $\hat{P}_{\tau|T}(i) = \mathbb{P}[x(\tau) = s_i \mid Y_T]$.

3.3.2 The Filter

With the previous assumptions clear in the mind, it is possible to describe well the various steps of the filter.

Prediction Step

In the so-called prediction step, previous estimated values are used to obtain the new ones. Using the notation defined above:

$$\begin{aligned}
\hat{P}_{t+1|t}(i) &= \mathbb{P}[x_{t+1} = s_i \mid Y_t] \qquad (3.6)\\
&= \sum_{s_j\in\mathcal{S}} \mathbb{P}[x_{t+1} = s_i, x_t = s_j \mid Y_t]\\
&\overset{\text{Bayes}}{=} \sum_{s_j\in\mathcal{S}} \mathbb{P}[x_{t+1} = s_i \mid x_t = s_j, Y_t]\, \mathbb{P}[x_t = s_j \mid Y_t]\\
&\overset{\text{Assum. 8}}{=} \sum_{s_j\in\mathcal{S}} \mathbb{P}[x_{t+1} = s_i \mid x_t = s_j]\, \mathbb{P}[x_t = s_j \mid Y_t]\\
&= \sum_{s_j\in\mathcal{S}} a_{ji}\, \mathbb{P}[x_t = s_j \mid Y_t]\\
&= \sum_{s_j\in\mathcal{S}} a_{ji}\, \hat{P}_{t|t}(j) \qquad (3.7)
\end{aligned}$$

[4] `https://it.mathworks.com/help/matlab/ref/times.html`

This shows how to link the expected states probabilities with the previous states probabilities. It is worth highlighting that, in the second step, the Bayes's rule is used whilst, in the third, the Assumption 8 has been exploited, i.e. future state depends on the previous state only.

Update Step

In the so-called update step, previous predicted values are used to obtain the estimation of them, i.e. to calculate $\hat{P}_{t|t}(i)$ using only $\hat{P}_{t|t-1}(i)$ and the current observation y_t. In a mathematical way:

$$\begin{aligned}
\hat{P}_{t|t}(i) &= \mathbb{P}[x_t = s_i \mid Y_t] \\
&\overset{\text{Bayes}}{=} \frac{\mathbb{P}[Y_t \mid x_t = s_i]\,\mathbb{P}[x_t = s_i]}{\mathbb{P}[Y_t]} \\
&\overset{\text{Split past/present}}{=} \frac{\mathbb{P}[Y_{0:t-1} \mid x_t = s_i]\,\mathbb{P}[y_t \mid x_t = s_i]\,\mathbb{P}[x_t = s_i]}{\mathbb{P}[Y_t]} \\
&= \frac{\mathbb{P}[x_t = s_i \mid Y_{0:t-1}]\,\mathbb{P}[Y_{0:t-1}]\,\mathbb{P}[y_t \mid x_t = s_i]}{\mathbb{P}[Y_t]} \\
&\overset{f = \frac{\mathbb{P}[Y_{0:t-1}]}{\mathbb{P}[Y_t]}}{=} \hat{P}_{t|t-1}(i)\, f\, c_i(y_t)
\end{aligned}$$

In the last step a definition was made: that can be also written as $f = \dfrac{1}{\sum\limits_{s_i \in \mathcal{S}} \hat{P}_{t|t-1}(i)\, c_i(y_t)}$ since $\sum\limits_{s_i \in \mathcal{S}} \hat{P}_{t|t}(i) = 1$ and f is constant with regards to the time. Summarizing, the update rule is:

$$\hat{P}_{t|t}(i) = \frac{\hat{P}_{t|t-1}(i)\, c_i(y_t)}{\sum\limits_{s_j \in \mathcal{S}} \hat{P}_{t|t-1}(j)\, c_j(y_t)} \tag{3.8}$$

Initialization step

It is clear that an initialization step is required for this system in order to work:

$$\begin{aligned}
\hat{P}_{0|0}(i) &= \mathbb{P}[x_t = s_i \mid y_0] \\
&= \frac{\hat{P}_{0|-1}(i)\, c_i(y_0)}{\sum\limits_{s_j \in \mathcal{S}} \hat{P}_{0|-1}(j)\, c_j(y_0)} \\
&\overset{\hat{P}_{0|-1}(i) = \pi_0(i)}{=} \frac{\pi_0(i)\, c_i(y_0)}{\sum\limits_{s_j \in \mathcal{S}} \pi_0(j)\, c_j(y_0)}
\end{aligned}$$

It is decided to use $\hat{P}_{0|-1}(i) = \pi_0(i)$ because no further information is available.

The filter

At this stage, we need to put all together to build a filter like the Kalman one. The target of this filter is finding the probability of all possible reachable states at

time t using the observations collected from 1 to T. Therefore:

$$\begin{aligned}
\hat{P}_{t|T}(i) &= \mathbb{P}[x_t = s_i \mid Y_T] = \\
&\underset{\text{Bayes}}{=} \frac{\mathbb{P}[Y_t \mid x_t = s_i]\,\mathbb{P}[Y_{t+1:T} \mid x_t = s_i]\,\mathbb{P}[x_t = s_i]}{\mathbb{P}[Y_t]} \\
&\underset{\text{Bayes Reverse}}{=} \frac{\overbrace{\mathbb{P}[Y_t, x_t = s_i]}^{\alpha_{t|t}(i)}\,\overbrace{\mathbb{P}[Y_{t+1:T} \mid x_t = s_i]}^{\beta_{t|t+1}(i)}}{\mathbb{P}[Y_t]} \\
&\underset{\text{rewrite t}}{=} \frac{\overbrace{\mathbb{P}[Y_{0:t-1}, x_t = s_i]}^{\alpha_{t|t-1}(i)}\,\overbrace{\mathbb{P}[Y_{t:T} \mid x_t = s_i]}^{\beta_{t|t}(i)}}{\mathbb{P}[Y_t]}
\end{aligned} \tag{3.9}$$

where

$$Y_{a:b} = \{y_a, y_{a+1}, \dots y_b\},\ a < b \in \mathbb{N}$$

Defining:

$$\alpha_{t|h}(i) = \mathbb{P}[Y_{0:h}, x_t = s_i] \tag{3.10}$$

$$\beta_{t|h}(i) = \mathbb{P}[Y_{h:T} \mid x_t = s_i] \tag{3.11}$$

for $h \in [1:T]$ and remembering probabilities are normalised, and therefore

$$\mathbb{P}[Y_t] = \sum_{s_j \in \mathcal{S}} \alpha_{t|t-1}(j)\,\beta_{t|t}(j)$$

the probability to be in state s_i at time t based on observations from 1 to T is:

$$\hat{P}_{t|T}(i) = \mathbb{P}[x_t = s_i \mid Y_T] = \frac{\alpha_{t|t-1}(i)\,\beta_{t|t}(i)}{\sum\limits_{s_j \in \mathcal{S}} \alpha_{t|t-1}(j)\,\beta_{t|t}(j)} \tag{3.12}$$

A new issue comes out: indeed, the new $\alpha_{t|h}$ and $\beta_{t|h}(i)$ was defined and then new prediction and update rules must be specified. These are quite simple to define for the α case. The prediction rule is:

$$\begin{aligned}
\alpha_{t+1|t}(i) &= \mathbb{P}[Y_t, x_{t+1} = s_i] \\
&\underset{@\ (3.4)}{=} \mathbb{P}[x_{t+1} = s_i \mid Y_t]\,\mathbb{P}[Y_t] \\
&\underset{@\ (3.5)}{=} \hat{P}_{t+1|t}(i)\,\mathbb{P}[Y_t] \\
&\underset{@\ (3.7)}{=} \sum_{s_j \in \mathcal{S}} a_{ji}\,\hat{P}_{t|t}(j)\,\mathbb{P}[Y_t] \\
&\underset{@\ (3.33)}{=} \sum_{s_j \in \mathcal{S}} a_{ji}\,\frac{\alpha_{t|t}(j)}{\mathbb{P}[Y_t]}\,\mathbb{P}[Y_t] \\
&= \sum_{s_j \in \mathcal{S}} a_{ji}\,\alpha_{t|t}(j)
\end{aligned} \tag{3.13}$$

whilst the update rule is:

$$\begin{aligned}
\alpha_{t|t}(i) &= \mathbb{P}[Y_t, x_t = s_i] && (3.14)\\
\overset{@\,(3.4)}{} &= \mathbb{P}[Y_t \mid x_t = s_i]\,\mathbb{P}[x_t = s_i]\\
\overset{\perp}{} &= \mathbb{P}[y_t \mid x_t = s_i]\,\mathbb{P}[Y_{t-1} \mid x_t = s_i]\,\mathbb{P}[x_t = s_i]\\
\overset{@\,(3.4)}{} &= c_i(y_t)\,\mathbb{P}[Y_{t-1}, x_t = s_i]\\
&= c_i(y_t)\,\alpha_{t|t-1}(i) && (3.15)
\end{aligned}$$

The initialisation step is:

$$\alpha_{0|-1}(i) = \mathbb{P}[x_0 = i] = \pi_0(i);$$

For β, instead, the formulae are slightly different since β is a backward filter. In this case, future values are required to calculate the one for the present. The prediction rule is:

$$\begin{aligned}
\beta_{t-1|t}(i) &= \mathbb{P}[Y_{t:T} \mid x_{t-1} = s_i]\\
\overset{\text{marginalize}}{} &= \sum_{s_j \in \mathcal{S}} \mathbb{P}[Y_{t:T}, x_t = j \mid x_{t-1} = s_i]\\
\overset{@\,(3.4)}{} &= \sum_{s_j \in \mathcal{S}} \mathbb{P}[Y_{t:T} \mid \underbrace{x_t = j, x_{t-1} = s_i}_{x_{t-1}=s_i \text{ is redundant}}]\,\mathbb{P}[x_t = j \mid x_{t-1} = s_i]\\
&= \sum_{s_j \in \mathcal{S}} \mathbb{P}[Y_{t:T} \mid x_t = j, x_{t-1}]\,\mathbb{P}[x_t = j \mid x_{t-1} = s_i]\\
&= \sum_{s_j \in \mathcal{S}} \beta_{t|t}(j)\,a_{ij} && (3.16)
\end{aligned}$$

whereas the update one is:

$$\begin{aligned}
\beta_{t|t}(i) &= \mathbb{P}[Y_{t:T} \mid x_t = s_i]\\
&= \mathbb{P}[y_t \mid x_t = s_i]\,\mathbb{P}[Y_{t+1:T} \mid x_t = s_i]\\
&= c_i(y_t)\,\beta_{t|t+1}(i); && (3.17)
\end{aligned}$$

and the initialisation step is:

$$\beta_{T|T}(i) = \mathbb{P}[y_T \mid x_T = s_i] = c_i(y_T)$$

All these formulae can be defined for each state in $\mathcal{S}$. To speed up the process, a smart idea is to use matrices, vectors, and element-wise product. Let $\alpha_{t|h}$ and $\beta_{t|h}$ row vectors with N columns, then the formulae (3.13) and (3.17) can be written in a more compact way to exploit them. For the α case:

$$\begin{aligned}
\alpha_{t|t} &= \alpha_{t|t-1} \odot C(:, y_t)^\intercal && (3.18)\\
\alpha_{t+1|t} &= \alpha_{t|t}\,A && (3.19)\\
\alpha_{0|-1} &= \pi_0 && (3.20)\\
\alpha_{t+1|t+1} &= \alpha_{t+1|t} \odot C(:, y_{t+1})^\intercal\\
&= (\alpha_{t|t}\,A) \odot C(:, y_{t+1})^\intercal && (3.21)\\
\alpha_{t+1|t} &= \alpha_{t|t}\,A\\
&= (\alpha_{t|t-1} \odot C(:, y_{t+1})^\intercal)\,A && (3.22)
\end{aligned}$$

where $C(:, i)$ is the i-th column of matrix C. At the same manner, for β:

$$\beta_{t|t} = \beta_{t|t+1} \odot C(:, y_t)^\intercal \tag{3.23}$$
$$\beta_{t|t+1} = \beta_{t+1|t+1}\, A^\intercal \tag{3.24}$$

since:

$$\beta_{t-1|t}(i) = \sum_{s_j \in \mathcal{S}} \beta_{t|t}(j)\, a_{ij} = \beta_{t|t}\, A(i, :)^\intercal$$

where $A(i, :)$ is the i-th row of A, i.e. the i-th column of $A^\intercal$.

These formulae allow to write in an elegant way the update and prediction steps for the filter that describes the model probabilities behaviour. It can be seen their powerful in the following part, the one that illustrates the three problems.

3.4 The Three Problems

Hidden Markov models are used for solving a wide range of issues, from speech recognition to genomics sequencing, but these can be collected in three different set of problems:

- Evaluation problems
- Decoding problems
- Learning problems

Several algorithms were developed to solve these issues, as can be seen in detail as follows.

3.4.1 Evaluation Problem

Let $\mathcal{Q} = (A, C, \pi_0)$ be a hidden Markov model with set of states $\mathcal{S}$ and observation set $\mathcal{O}$. The evaluation problem consists of calculating the probability of a given sequence of observation, $Y_T = [y_1\, y_2\, \dots\, y_T]^\intercal$, i.e. find $\mathbb{P}[Y_t \mid \mathcal{Q}]$. This is not a difficult problem to solve since, using simple probability rules stated above:

$$\mathbb{P}[Y_T] = \sum_{X_T \in \mathcal{S}^T} \mathbb{P}[Y_T \mid X_T]\, \mathbb{P}[X_T] = \sum_{X_T \in \mathcal{S}^T} \left(\prod_{k=1}^{T} c_{x_k}(y_k) \prod_{l=1}^{T-1} a_{x_l}(x_{l+1}) \right)$$

where

$$\mathcal{S}^T = \underbrace{\mathcal{S} \times \mathcal{S} \times \cdots \times \mathcal{S}}_{T}$$

is the T-ary Cartesian power of the states set. The formula stated above is quite simple to write but it has a huge computational cost: indeed, there is a summation with N^T elements and, for each element, $(T-1)T$ products have to be performed. Therefore, this algorithm has a complexity of $O(N^T)$ that is too high to be used

normally. For example, given a sequence of 100 observations coming from a two-state hidden Markov model, that operation will took about 4×10^{15} years on a High-end CPU[5], about three hundred thousand times the age of the universe. To solve in a smarter way this issue, the **forward algorithm** and the **backward algorithm** were developed. These algorithms allow to reduce the complexity to $O(T\,N^2)$: in the previous example, using this algorithm, the reckon would only take four nanoseconds.

Forward Algorithm

Let $\alpha_{t|t}(i)$ be defined following (3.10), i.e. $\alpha_{t|t}(i) = \mathbb{P}[Y_{0:t}, x_t = s_i]$. The forward algorithm allows to solve issue of finding the probability of an observed event sequence: it exploits the marginalisation of the last sample of alpha to provide a faster way than the previous algorithm to perform this. Indeed, using (3.5) and the definition above:

$$\mathbb{P}[Y_T] = \sum_{s_i \in \mathcal{S}} \mathbb{P}[Y_T, x_T = s_i] = \sum_{s_i \in \mathcal{S}} \alpha_{T|T}(i)$$

Furthermore, using (3.4) and the fact that the alpha is a stochastic vector, i.e. the sum of its components is equal to one:

$$\mathbb{P}\left[x_T = s_i \mid Y_T\right] = \frac{\mathbb{P}\left[Y_{0:T}, x_T = s_i\right]}{\mathbb{P}\left[Y_T\right]} = \frac{\alpha_{T|T}(i)}{\mathbb{P}\left[Y_T\right]} = \frac{\alpha_{T|T}(i)}{\sum\limits_{s_j \in \mathcal{S}} \alpha_{T|T}(j)} \tag{3.25}$$

Therefore, a new to obtain

$$\mathbb{P}\left[Y_T\right] = \sum_{s_j \in \mathcal{S}} \alpha_{T|T}(j) \tag{3.26}$$

can be found. To perform that reckon, N sums are needed. Furthermore, T alphas are required to be calculated and each one takes $N^2 + N$ operations. Summing all together, this algorithm has a $O(T \cdot N^2)$ complexity.

Backward Algorithm

The issue of find the probability of an observed event sequence can be also solved marginalising with regard to the first state. Mathematically, using (3.5) and (3.4), the problem becomes:

$$\mathbb{P}\left[Y_T\right] = \sum_{s_i \in \mathcal{S}} \mathbb{P}\left[Y_T | x_1 = s_i\right] \mathbb{P}\left[x_1 = s_i\right] = \sum_{s_i \in \mathcal{S}} \beta_{1|1}(i)\pi_i(0)$$

Using the beta definition (3.11), this problem can be solved calculating the betas from T to 1 with (3.23) and (3.24). In a more compact way, the solution can be written as

$$\mathbb{P}\left[Y_T\right] = \beta_{1|1}\pi_0^\intercal \tag{3.27}$$

[5]For this reckon, a Intel i5-3550 with 100 GFLOPS was taken as example, assuming that it can be performed 10^{10} operation for second. This is a optimistic estimation because it not include the memory access time and latency.

Like the other algorithm, T betas are required to be calculated and each one takes $N^2 + N$ operations. Summing all together, also this algorithm has a $O(T \cdot N^2)$ complexity. It is worth highlighting that also in the other algorithm, it is necessary the knowledge of π_0 since it is the initial condition for the alphas.

Forward-Backward Algorithm

Since modern CPU are no more single thread but they can perform several reckons at same time, it is useful to exploit this to speed up the search for $\mathbb{P}[Y_T]$. An idea is to combine forward and backward algorithms: indeed,

$$\begin{aligned}
\mathbb{P}[Y_T] &= \sum_{s_i \in \mathcal{S}} \mathbb{P}[Y_T, x_t = s_i] = \sum_{s_i \in \mathcal{S}} \mathbb{P}[Y_{1:t-1}, Y_{t:T}, x_t = s_i] \\
&= \sum_{s_i \in \mathcal{S}} \mathbb{P}[Y_{t:T} \mid Y_{1:t-1}, x_t = s_i] \, \mathbb{P}[Y_{1:t-1}, x_t = s_i] \qquad (3.28) \\
&= \sum_{s_i \in \mathcal{S}} \mathbb{P}[Y_{t:T} \mid x_t = s_i] \, \mathbb{P}[Y_{1:t-1}, x_t = s_i] \qquad (3.29) \\
&= \sum_{s_i \in \mathcal{S}} \beta_{t|t}(i) \, \alpha_{t|t-1}(i) \qquad (3.30)
\end{aligned}$$

where in step (3.28) Bayes's rule (3.4) was used whereas in (3.29) the Assumption 9 was exploited. Using $t = \lceil \frac{T}{2} \rceil$, the algorithm is split into two parts that can live independently and therefore they can be performed simultaneously. This allows to halve the time if this code is executed on a modern CPU.

3.4.2 Decoding Problem

The decoding problem contains all the issue about to estimate states given observations. The main issues that are included in this are the **estimation of a state** and the **estimation of the sequence of states**. Even if these two problems are quite similar, two different algorithms were developed.

State Estimation

The state estimation problem consists of finding the most likely state given the observations, i.e. $\tilde{x}_{t|T}$. To find that it is necessary to calculate the probability of a state given a sequence of observed events, i.e. $\mathbb{P}[x_t = s_i \mid Y_T]$. It is worth reminding that has already been defined:

$$\mathbb{P}[x_t = s_i \mid Y_T] = \hat{P}_{t|T}(i)$$

and the solution is (3.12), i.e.:

$$\hat{P}_{t|T}(i) = \frac{\alpha_{t|t-1}(i) \, \beta_{t|t}(i)}{\sum_{s_j \in \mathcal{S}} \alpha_{t|t-1}(j) \, \beta_{t|t}(j)} \qquad (3.31)$$

Also, in this case the algorithm can be paralleled to calculate alphas and betas. Remind that, if $t = T$, i.e. only observations until t are available, no betas are required to be calculated. Indeed, if $\hat{P}_{t|t}(i)$ is needed:

$$\hat{P}_{t|t}(i) = \mathbb{P}[x_t = s_i \mid Y_t] = \frac{\alpha_{t|t}(i)}{\sum_{s_j \in \mathcal{S}} \alpha_{t|t}(j)} \tag{3.32}$$

$$= \frac{\mathbb{P}[x_t = s_i, Y_t]}{\mathbb{P}[Y_t]} = \frac{\alpha_{t|t}(i)}{\mathbb{P}[Y_t]} \tag{3.33}$$

The state estimation problem required to find a state that is the most likely, i.e. it has the higher probability value. To solve this, an idea is to use the argmax function. Indeed, using the argmax properties:

$$\tilde{x}_{t|T} = \arg\max_{s_i \in \mathcal{S}} \left[\hat{P}_{t|T}(i)\right] = \arg\max_{s_i \in \mathcal{S}} \left[\frac{\alpha_{t|t-1}(i)\,\beta_{t|t}(i)}{\sum_{s_j \in \mathcal{S}} \alpha_{t|t-1}(j)\,\beta_{t|t}(j)}\right]$$

$$= \arg\max_{s_i \in \mathcal{S}} \left[\alpha_{t|t-1}(i)\,\beta_{t|t}(i)\right] \tag{3.34}$$

Like in the previous case, if $T = t$, the betas part can be dropped.

The solution above suggests a possible way to resolve the other issue, the state sequence estimation. Indeed, if the (3.34) is used for each t from 1 to T, a sequence of most likely state is built, i.e. $\tilde{X}_T = \{\tilde{x}_{1|T}, \ldots, \tilde{x}_{T|T}\}$. A possible implementation can be seen in Algorithm 1 using the formulae illustrated in Section 3.3.2. This

Algorithm 1 Likeliest states sequence algorithm

$\alpha_{0|-1} = \pi_0$ ▷ Initialisation
$\beta_{T|T}(i) = c_i(y_T)$
for $t = 1,\ t < T,\ t_{++}$ **do** ▷ Compute α
  $\alpha_{t+1|t} = (\alpha_{t|t-1} \odot C(:, y_{t+1})^\intercal)\, A$
end for
for $t = T,\ t > 0,\ t_{--}$ **do** ▷ Compute β
  $\beta_{t|t} = (\beta_{t+1|t+1}\, A^\intercal) \odot C(:, y_t)^\intercal$
end for
for $t = 1,\ t < T,\ t_{++}$ **do** ▷ Argmax step
  $\tilde{x}_{t|T} = \arg\max_{s_i \in \mathcal{S}} \left[\alpha_{t|t-1}(i)\,\beta_{t|t}(i)\right]$
end for

algorithm is quite good, but it is not the best can be done because it does not take into account the links between consecutive states in the chain. Therefore, a new algorithm was developed to reach better results.

State Sequence Estimation

The problem, which is slightly different from the previous case, consists to find the most probability sequence of states given the observations Y_T , i.e.

$$\hat{X}_T = \arg\max_{X_T \in \mathcal{S}^T} \left[\mathbb{P}\left[X_T \mid Y_T\right]\right] \tag{3.35}$$

This can also be rewritten as:

$$\hat{X}_T = \arg\max_{X_T \in \mathcal{S}^T} [\mathbb{P}[X_T \mid Y_T]] = \arg\max_{X_T \in \mathcal{S}^T} \left[\frac{\mathbb{P}[X_T, Y_T]}{\mathbb{P}[Y_T]} \right] = \arg\max_{X_T \in \mathcal{S}^T} [\mathbb{P}[X_T, Y_T]] \tag{3.36}$$

An iterative solution for this problem was developed by Andrew James Viterbi[6] in 1967 and published on [20]. This algorithm is widely used in the telecommunication field since the receiving of a bit can be modelled effectively using hidden Markov models and therefore, to retrieve the sequence of bits sent, i.e. the message, a state sequence estimation must be done.

The Viterbi's algorithm

The Viterbi's algorithm was developed to resolve in a better way the following problem:

$$\hat{X}_T = \arg\max_{X_T \in \mathcal{S}^T} [P[X_T | Y_T]]$$

i.e. the most likely assignment of state sequence. Therefore, this algorithm focuses on finding the best path of states. The its derivation is proposed below: it is quite simple since it uses the properties of the argmax function in addition to some probability rules. The idea is to use a two-pass filter: the first pass is done to find the best final state using the previous algorithm whereas the second one allows to find the best path using the information given by the outcomes and the first step.

To speed up the proof, a definition has to be outlined. The one is:

$$\delta_{t|h}(i) = \max_{x(\tau) \in \mathcal{S}, \tau \in 0:t-1} [P[X_{1:t-1}, x_t = i, Y_{1:h}]] \tag{3.37}$$

i.e. the maximum over the first $t-1$ possible states of the joint probability of observations (from 1 to h) and states given a particular final state s_i. To simplify the notation, $\delta_t(i) = \delta_{t|t}(i)$; furthermore, like it was done above, deltas can be written in vectorial form, i.e. $\delta_{t|h} \in \mathbb{R}^{1 \times N}$. Therefore, the state estimation problem can be written as:

$$\begin{aligned} \hat{x}_{t|t} = \arg\max_{s_i \in \mathcal{S}} [\delta_t(i)] &= \arg\max_{s_i \in \mathcal{S}} \left[\max_{X_{t-1} \in \mathcal{S}^{t-1}} [P[X_{t-1}, x_t = i, Y_t]] \right] \\ &= \arg\max_{s_i \in \mathcal{S}} \left[\max_{X_{t-1} \in \mathcal{S}^{t-1}} [P[X_{t-1}, x_t = i \mid Y_t] \, P[Y_t]] \right] \\ &= \arg\max_{s_i \in \mathcal{S}} \left[\max_{X_{t-1} \in \mathcal{S}^{t-1}} [P[X_{t-1}, x_t = i \mid Y_t]] \right] \end{aligned}$$

using Bayes' rule. Therefore, for the last state:

$$\hat{x}_T = \arg\max_{s_i \in \mathcal{S}} [\delta_T(i)] \tag{3.38}$$

[6] Also known as Andrea Giacomo Viterbi, he was born in Bergamo in 1935 and he is one of the co-founders of Qualcomm Inc., a big semiconductor and telecommunications equipment company.

Like in the state estimation, it is useful to have a recursion algorithm for calculating δ_t. A possible solution is:

$$
\begin{aligned}
\delta_{t+1}(i) &= \max_{X_t \in \mathcal{S}^t} \left[P[X_t, x_{t+1} = i, Y_{t+1}]\right] \\
&= \max_{X_{t-1} \in \mathcal{S}^{t-1}} \left[\max_{s_j \in \mathcal{S}} \left[P[X_{t-1}, x_t = s_j, x_{t+1} = i, Y_{t+1}]\right]\right] \\
&= \max_{s_j \in \mathcal{S}} \left[\max_{X_{t-1} \in \mathcal{S}^{t-1}} \left[P[X_{t-1}, x_t = s_j, x_{t+1} = i, Y_{t+1}]\right]\right] \\
\text{@ 3.4} &= \max_{s_j \in \mathcal{S}} \left[\max_{X_{t-1} \in \mathcal{S}^{t-1}} \left[P[X_{t-1}, Y_{t+1} \mid x_t = s_j, x_{t+1} = i]\, P[x_t = s_j, x_{t+1} = i]\right]\right] \\
\text{split Y} &= \max_{s_j \in \mathcal{S}} \left[\max_{X_{t-1} \in \mathcal{S}^{t-1}} \left[\begin{array}{r} P[X_{t-1}, Y_t \mid x_t = s_j, x_{t+1} = i]\, P[x_t = s_j, x_{t+1} = i]\cdot \\ P[y_{t+1} \mid x_t = s_j, x_{t+1} = i] \end{array}\right]\right] \\
\text{@ 3.4} &= \max_{s_j \in \mathcal{S}} \left[\max_{X_{t-1} \in \mathcal{S}^{t-1}} \left[\begin{array}{r} P[X_{t-1}, Y_t \mid x_t = s_j, x_{t+1} = i]\cdot \\ P[y_{t+1} \mid x_t = s_j, x_{t+1} = i]\cdot \\ P[x_{t+1} = i \mid x_t = s_j]\, P[x_t = s_j] \end{array}\right]\right] \\
\text{redundancy} &= \max_{s_j \in \mathcal{S}} \left[\max_{X_{t-1} \in \mathcal{S}^{t-1}} \left[\begin{array}{r} P[X_{t-1}, Y_t \mid x_t = s_j]\, P[y_{t+1} \mid x_{t+1} = i]\cdot \\ P[x_{t+1} = i \mid x_t = s_j]\, P[x_t = s_j] \end{array}\right]\right] \\
\text{def and order} &= \max_{s_j \in \mathcal{S}} \left[\max_{X_{t-1} \in \mathcal{S}^{t-1}} \left[P[X_{t-1}, Y_t \mid x_t = s_j]\, P[x_t = s_j]\, c_i(y_{t+1})\, a_{ji}\right]\right] \\
\text{@ 3.4 rev} &= \max_{s_j \in \mathcal{S}} \left[\underbrace{\max_{X_{t-1} \in \mathcal{S}^{t-1}} \left[P[X_{t-1}, Y_t, x_t = s_j]\right]}_{\delta_t(j)}\, c_i(y_{t+1})\, a_{ji}\right] \\
&= \max_{s_j \in \mathcal{S}} \left[\delta_t(j)\, c_i(y_{t+1})\, a_{ji}\right] \\
&= \max_{s_j \in \mathcal{S}} \left[\delta_t(j)\, a_{ji}\right]\ c_i(y_{t+1}) = \delta_{t+1|t+1}(i)
\end{aligned}
$$

This solution exploits the power of Bayes' rule and allows to write a prediction plus update steps algorithms: indeed, if it is called $\delta_{t+1|t}(i)$ the prediction step whilst $\delta_{t|t}(i)$ the update one:

$$\delta_{t+1|t}(i) = \max_{s_j \in \mathcal{S}} \left[\delta_t(j)\, a_{ji}\right] \tag{3.39}$$

$$\delta_{t+1|t+1}(i) = \delta_{t+1|t}(i)\, c_i(y_{t+1}) \tag{3.40}$$

$$\delta_{0|0}(i) = c_i(y_0)\, \pi_0(i)$$

It is worth remembering that a_{ij} is the element of the transition matrix A in position ij and c_ij is the element of the observation matrix C in position ij whilst π_0 represents the initial probability of the chain. Now it is possible to find $\hat{X}_T$. Indeed, the last state is $\hat{x}_T = \arg\max_{s_i \in \mathcal{S}} \left[\delta_T(i)\right]$ and it can be calculated with (3.39) and (3.40). Using that information and the property that joint and conditioned probabilities are equal, save for a value that is constant with regards to X_T, i.e.

$$\hat{X}_T = \arg\max_{X_T \in \mathcal{S}^T} \left[P[X_T \mid Y_T]\right] = \arg\max_{X_T \in \mathcal{S}^T} \left[P[X_T, Y_T]\right]$$

Therefore, it is possible to find a recursive method:

$$
\begin{aligned}
&\max_{X_{T-1}\in\mathcal{S}^{T-1}} [P[X_{T-1}, x_T = \hat{x}_T, Y_T]] = \\
\overset{@\ 3.4}{=}\ &\max_{X_{T-1}\in\mathcal{S}^{T-1}} [P[X_{T-1}, Y_T \mid x_T = \hat{x}_T]\, P[x_T = \hat{x}_T]] \\
\overset{\perp}{=}\ &\max_{X_{T-1}\in\mathcal{S}^{T-1}} \left[P[X_{T-1}, Y_{T-1} \mid x_T = \hat{x}_T]\ \underbrace{P[y_t \mid x_T = \hat{x}_T]}_{\perp\ \mathrm{x}}\ P[x_T = \hat{x}_T] \right] \\
=\ &\max_{X_{T-1}\in\mathcal{S}^{T-1}} [P[X_{T-1}, Y_{T-1} \mid x_T = \hat{x}_T]\, P[x_T = \hat{x}_T]] \\
\overset{@\ 3.4\ \mathrm{rev}}{=}\ &\max_{X_{T-1}\in\mathcal{S}^{T-1}} [P[X_{T-1}, Y_{T-1}, x_T = \hat{x}_T]] \\
=\ &\max_{X_{T-2}\in\mathcal{S}^{T-2}} \left[\max_{s_j\in\mathcal{S}} [P[X_{T-2}, Y_{T-1}, x_T = \hat{x}_T, x_{T-1} = s_j]] \right] \\
\overset{@\ 3.4}{=}\ &\max_{X_{T-2}\in\mathcal{S}^{T-2}} \left[\max_{s_j\in\mathcal{S}} \left[\begin{array}{r} P[X_{T-2}, Y_{T-1} \mid x_T = \hat{x}_T, x_{T-1} = s_j]\cdot \\ P[x_T = \hat{x}_T, x_{T-1} = s_j] \end{array} \right] \right] \\
\overset{@\ 3.4}{=}\ &\max_{X_{T-2}\in\mathcal{S}^{T-2}} \left[\max_{s_j\in\mathcal{S}} \left[\begin{array}{r} P[X_{T-2}, Y_{T-1} \mid x_{T-1} = s_j]\, P[x_T = \hat{x}_T \mid x_{T-1} = s_j]\cdot \\ P[x_{T-1} = s_j] \end{array} \right] \right] \\
=\ &\max_{X_{T-2}\in\mathcal{S}^{T-2}} \left[\max_{s_j\in\mathcal{S}} \left[\begin{array}{r} P[X_{T-2}, Y_{T-1} \mid x_{T-1} = s_j]\, P[x_{T-1} = s_j]\cdot \\ \underbrace{P[x_T = \hat{x}_T \mid x_{T-1} = s_j]}_{a_j(\hat{x}_T)} \end{array} \right] \right] \\
\overset{@\ 3.4\ \mathrm{rev}}{=}\ &\max_{X_{T-2}\in\mathcal{S}^{T-2}} \left[\underbrace{\max_{s_j\in\mathcal{S}} [P[X_{T-2}, Y_{T-1}, x_{T-1} = s_j]]}_{\delta_{T-1|T-1}(j)}\ a_j(\hat{x}_T) \right] \\
=\ &\max_{s_j\in\mathcal{S}} [\delta_{T-1}(j)\, a_j(\hat{x}_T)]
\end{aligned}
\tag{3.41}
$$

Even if they seem a lot of difficult reckons, they are quite simple. The first step exploits the Bayes' rule whereas the second one the rule about probability of independent events. In the third, elements that do not depend on X_T are removed whilst in the following the Bayes' rule is exploited in reverse, i.e. to compact the equation. The fifth step splits the maximum function into two parts whilst the sixth and the seventh exploit the Bayes' rule to write in a more useful way (step 8) the formula. After that, a swap is done, and it can be noticed that the elements in the formula are already known. Obviously, the state at time $T-1$ which allows to have the maximum probability of the sequence is the argmax of the last step. Therefore $\hat{X}_T = [\hat{x}_1, \ldots, \hat{x}_T]$ where

$$
\hat{x}_{t-1} = \arg\max_{s_j\in\mathcal{S}} [\delta_{t-1}(j)\, a_j(\hat{x}_t)] \tag{3.42}
$$

Summarizing, the algorithm needs four steps:

1. Initialization:

$$
\delta_1(i) = c_i(y_1)\, \pi_0(i)
$$

2. Forward pass:
$$\delta_t(j) = \max_{k \in \mathcal{S}} \left[\delta_{t-1}(k)\, a_{kj}\, c_j(y_t) \right]$$

3. Final state finding:
$$\hat{x}_T = \arg\max_{k \in \mathcal{S}} \left[\delta_T(k) \right]$$

4. Backward pass:
$$\hat{x}_{t-1} = \arg\max_{s_j \in \mathcal{S}} \left[\delta_{t-1}(j)\, a_j(\hat{x}_t) \right]$$

some optimizations can be applied to speed up the algorithm. A possible implementable solution is reported in as Algorithm 2. As can be seen, it is a very efficient

Algorithm 2 Viterbi's algorithm

for all $s_i \in \mathcal{S}$ **do** ▷ Initialisation
  $\delta_1(i) = c_i(y_1)\, \pi_0(i)$
  $z_1(i) = 0$
end for
for $t = 2$, $t \leq T$, t_{++} **do** ▷ Forward pass
  for all $s_j \in \mathcal{S}$ **do**
    $\delta_t(j) = \max_{k \in \mathcal{S}} \left[\delta_{t-1}(k)\, a_{kj}\, c_j(y_t) \right]$
    $z_t(j) = \arg\max_{k \in \mathcal{S}} \left[\delta_{t-1}(k)\, a_{kj}\, c_j(y_t) \right]$ ▷ most probable previous state
  end for
end for
$\hat{x}_T = \arg\max_{k \in \mathcal{S}} \left[\delta_T(k) \right]$ ▷ Compute last element
for $t = T$, $t \geq 2$, t_{--} **do** ▷ Backward pass
  $\hat{x}_{t-1} = z_t(x_t)$
end for

algorithm: its complexity is $O(N^2\, T)$ instead of $O(N^T)$ that is the complexity of a brute-force argmax.

3.4.3 Learning Problem

The last problem is the learning problem, i.e. how to find the initial probabilities, the transition matrix A, and the observations matrix C given a sequence of observations Y_T. The idea is to maximise the likelihood, i.e.:

$$\{\pi_0, A, C\} = \theta = \arg\max_{\theta} \left[\mathbb{P} \left[Y_T \mid \theta \right] \right]$$

The most used formula is the **Baum-Welch algorithm** [21] that exploits the **Expectation–maximization** approach and the forward-backward algorithm to find a quasi-optimal solution, i.e. the local maximum probability given the initial condition. This approach relies on doing these the EM two steps until convergence: the first step calculates the probability of the current model whereas the other performs the maximisation to find the variables.

Baum-Welch algorithm

This algorithm was developed in order to solve the third problem with a quasi-optimal solution. Indeed, no analytic way is known to solve it. Therefore, an iterative procedure must be used: the full proof about this algorithm can be found in [22] whereas, here, only the most useful formulae will be reported. This algorithm, to work, takes as inputs random parameters. The **first step** is the expectation one: in this one, the probability of the path to be at state s_i at time t making a transition to state s_j given the observations and the current parameters θ is calculated, i.e. $\xi_t(i,j)$. Before doing that, a slightly more general version of (3.9) should be define:

$$\hat{P}_{t|T,\theta}(i) = \mathbb{P}[x_t = s_i \mid Y_T, \theta] \tag{3.43}$$

exploiting (3.31), it can be rewritten as:

$$\begin{aligned}\hat{P}_{t|T,\theta}(i) &= \frac{\alpha_{t|t-1,\theta}(i)\,\beta_{t|t,\theta}(i)}{\sum\limits_{s_j \in \mathcal{S}} \alpha_{t|t-1,\theta}(j)\,\beta_{t|t,\theta}(j)} \\ &= \frac{\alpha_{t|t,\theta}(i)\,\beta_{t|t+1,\theta}(i)}{\sum\limits_{s_j \in \mathcal{S}} \alpha_{t|t,\theta}(j)\,\beta_{t|t+1,\theta}(j)}\end{aligned} \tag{3.44}$$

After that, $\xi_t(i,j)$ can be defined[7] as:

$$\begin{aligned}\xi_t(i,j) &= \mathbb{P}\left[x_t = s_i, x_{t+1} = s_j \mid Y_T, \theta\right] \\ &= \mathbb{P}\left[x_{t+1} = s_j \mid x_t = s_i, Y_T, \theta\right] \mathbb{P}\left[x_t = s_i \mid Y_T, \theta\right] \\ &= \mathbb{P}\left[x_{t+1} = s_j \mid x_t = s_i, \theta\right] \frac{\mathbb{P}\left[x_t = s_i, Y_T \mid \theta\right]}{\mathbb{P}\left[Y_T \mid \theta\right]} \\ &\overset{(3.9)}{=} \mathbb{P}\left[x_{t+1} = s_j \mid x_t = s_i, \theta\right] \frac{\mathbb{P}\left[x_t = s_i, Y_t \mid \theta\right] \mathbb{P}\left[x_t = s_i, Y_{t+1:T} \mid \theta\right]}{\mathbb{P}\left[Y_T \mid \theta\right]} \\ &= \frac{\alpha_{t|t}(i)\,a_{ij}\,\beta_{t+1|t+1}(j)}{\mathbb{P}\left[Y_T \mid \theta\right]}\end{aligned} \tag{3.45}$$

$$= \frac{\alpha_{t|t}(i)\,a_{ij}\,c_i(y_{t+1})\,\beta_{t+1|t+2}(j)}{\mathbb{P}\left[Y_T \mid \theta\right]} \tag{3.46}$$

As it can be seen, alphas and betas have to be calculated in this algorithm too.

The **second step** is the maximisation one: in this one, the θ's parameters are re-estimated with regards to maximize the likelihood. The re-estimation formulae for $\theta = \pi_0, A, C$ are:

$$\pi_0 = \hat{P}_{0|T,\theta} \in \mathbb{R}^{1\times N} \tag{3.47}$$

$$a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \hat{P}_{t|T,\bar{\theta}}(i)} \tag{3.48}$$

$$c_{ij} = \frac{\sum_{t=1}^{T-1} \delta_{y_t}(j)\,\xi_t(i,j)}{\sum_{t=1}^{T} \hat{P}_{t|T,\bar{\theta}}(i)} \tag{3.49}$$

[7] In [21], they defined $\beta_t = \mathbb{P}[Y_{t+1:T} \mid x_t = s_i]$, i.e. $\beta_t = \beta_{t|t+1}$.

where $\bar{\theta}$ are the parameters estimated during the previous iteration of the algorithm whereas:

$$\delta_{y_t}(j) = \begin{cases} 1 & \text{if } y_t = w_j \\ 0 & \text{if } y_t \neq w_j \end{cases} \tag{3.50}$$

Some of the formulae showed above are quite tricky: the first one represents the probability of the first step to be in one of the states given the observations and therefore are quite plain to use (3.43). Instead, for the other ones are useful to define better the summations:

$\sum_{t=1}^{T-1} \xi_t(i,j)$ is the number of expected transitions from state s_i to s_j;
$\sum_{t=1}^{T-1} \hat{P}_{t|T,\bar{\theta}}(i)$ is the number of expected transitions from state s_i;
$\sum_{t=1}^{T} \hat{P}_{t|T,\bar{\theta}}(i)$ is the number of expected times to be in state s_i.

It is useful to highlight that the last two summations differ only by $P_{T|T,\bar{\theta}}(i)$. The transition matrix estimation formula, therefore, represents the ratio between the expected number of changes from state s_i to s_j and the number of expected transition from state s_i. Instead, the observation matrix estimation formula is the ratio of the expected number of time of being in state s_i and observing the event w_j over the expected number of time to be in state s_i.

These two steps must be performed until convergence: since it is an iterative method, a good idea is to execute this algorithm with several initial conditions and then take the solution with the highest probability $\mathbb{P}[Y_T \mid \theta]$.

The Markov Chain case

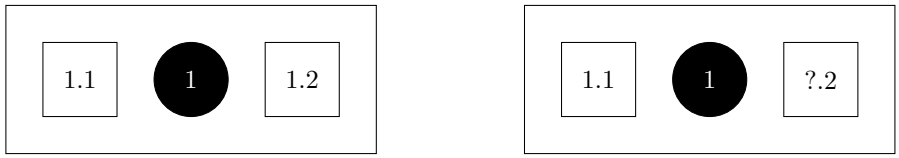
In the case of a directly observable chain, the algorithm to estimate π_0 and A is faster and simpler than the Baum-Welch one. Indeed, counting the number of switch is sufficient:

$$a_{ij} = \frac{1}{T-1} \sum_{t=1}^{T} \delta_{x_t}(i)\delta_{x_{t+1}}(j)$$

A possible estimation of π_0 is $a(x_1)^\intercal$, i.e. the x_1-th column of the estimate transition matrix. Indeed, if no initial condition is accessible, the only useful available information is the first state and the estimated transition matrix. The latter allows to know the probability to go in state s_j: therefore, if the column of A regarding the future state x_1 is considered, the probabilities of their possible parent states are selected, i.e. the estimate that it has been looking for.

3.5 The Model

After this brief summary about Markov chains and hidden Markov models, all ingredients to develop a model for the problem are ready. As stated before in this Chapter, the problem to be solved in this section consists to model a workstation



θ_0 θ_1

Figure 3.3: A stylised version of the workstation with two sensors. In the left-hand side there is the working one whereas in the right-hand side a damaged workstation is drawn.

like the ones in Figure 3.3. Remember that two models are needed: one for the working workstation and one for the topsy-turvy workstation. To simplify the work, as already stated, all the workstations have the same behaviour, i.e. they have the same transition matrix A. Furthermore, all sensors are equally powerful and thus they have the same observation matrix C.

3.5.1 The first Model

As it can be seen, to model a working workstation a hidden Markov model with two states and four possible observable events can be used. Indeed, as stated in Chapter 2, the worker states are two, i.e. absent or present, and the four possible observations are:

1. The two sensors reported an absence.
2. The left-hand side sensor reported an absence whereas the other reported a presence.
3. The left-hand side sensor reported a presence whereas the other reported an absence.
4. The two sensors reported a presence.

In a more mathematical way, this model can be written as:

$$\mathcal{Q}_{\theta_0} = (A^{\theta_0}, C^{\theta_0}, \pi_0^{\theta_0}) \tag{3.51}$$

where $A^{\theta_0} \in \mathbb{R}^{2\times 2}$, $C^{\theta_0} \in \mathbb{R}^{2\times 4}$, and $\pi_0^{\theta_0} \in \mathbb{R}^{1\times 2}$ since states are $\mathcal{S} = \{s_1, s_2\} = \{0, 1\}$ whilst the observable events are $\mathcal{O} = \{w_1, w_2, w_3, w_4\} = \{1, 2, 3, 4\}$. An illustration of this chain can be seen in Figure 3.4. The matrices used for this chain can be estimated using the algorithm described in subsection 3.4.3. Even if, for this work, it is assumed that all of these values are known, it is useful to show how to build the overall observation matrix C if only the singular sensors matrices are available. Remember that the observations in reality are two and they are merged following Table 2.1. Let $C^{(1)}, C^{(2)} \in \mathbb{R}^{2\times 4}$ be the two observation matrices of the two sensors defined as:

$$c_{ij}^{(k)} = \mathbb{P}\left[y_k(t) = w_j^s \mid x(t) = s_i\right]$$

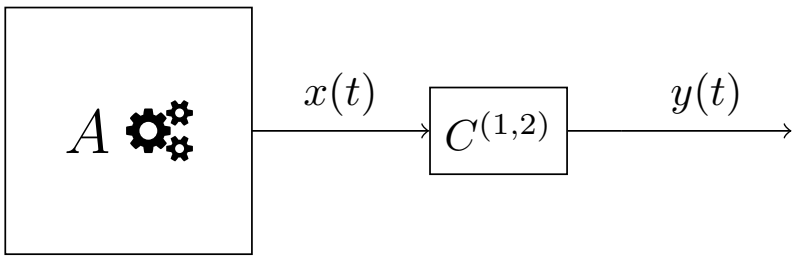


Figure 3.4: The modelled system behind a working workspace.

where $w_j^s \in \{w_1^s, w_2^s\} = \{0, 1\}$ is the event observed by the sensor. Then, since C^{θ_0} represent the probability of an event given the current state, i.e.:

$$c_{ij}^{\theta_0} = \mathbb{P}\left[y(t) = w_j \mid x(t) = s_i\right] = \mathbb{P}\left[y_1(t) = w_l^s \cap y_2(t) = w_m^s \mid x(t) = s_i\right]$$

using l and m s.t.:

j	l	m
1	1	1
2	2	1
2	1	2
4	2	2

For Assumption 9, the observations are independent each other, therefore:

$$c_{ij}^{\theta_0} = \mathbb{P}\left[y_1(t) = w_l^s \mid x(t) = s_i\right] \mathbb{P}\left[y_2(t) = w_m^s \mid x(t) = s_i\right] = c_{il}^{(1)}\, c_{im}^{(2)}$$

In a more compact way:

$$C^{\theta_0} = \begin{bmatrix} \mathrm{Vect}(C^{(1)}(:,1) \odot C^{(2)}(:,1)) \\ \mathrm{Vect}(C^{(1)}(:,2) \odot C^{(2)}(:,2)) \end{bmatrix} \tag{3.52}$$

where the $\mathrm{Vect}(\cdot)$ function is defined as:

$$\mathrm{Vect}(\cdot)\colon \mathbb{R}^{M \times N} \to \mathbb{R}^{1 \times M\dot{N}}, \quad B \mapsto \begin{bmatrix} B(:,1)^\intercal & \dots & B(:,N)^\intercal \end{bmatrix} \tag{3.53}$$

3.5.2 The second Model

The other model can be written as a pair of hidden Markov models with only two observable events for each one. Indeed, in this case the two sensors are looking at two different desks, i.e. they are observations of two distinct chains. An illustration of this chain can be seen in Figure 3.5 whereas a mathematical expression for this is:

$$\mathcal{Q}_{a\theta_1} = \{(A^{(1)}, C^{(1)}, \pi_0^{(1)}), (A^{(2)}, C^{(2)}, \pi_0^{(2)})\} \tag{3.54}$$

Like in the previous case, $A^{(1)}, A^{(2)} \in \mathbb{R}^{2\times 2}$ and $\pi_0^{(1)}, \pi_0^{(2)} \in \mathbb{R}^{1\times 2}$ but, now, $C^{(1)}, C^{(2)} \in \mathbb{R}^{2\times 2}$, i.e. each sensor has its observation matrix. These matrices are defined as:

$$c_{ij}^{(k)} = \mathbb{P}\left[y_k(t) = w_j^s \mid x_k(t) = s_i\right]$$

Furthermore, the states are $\mathcal{S} = \{s_1, s_2\} = \{0, 1\}$ and the observable events, for each chain, are $\mathcal{O}^s = \{w_1^s, w_2^s\} = \{0, 1\}$. Anyway, the two observations are merged into one following Table 2.1: therefore, the real set is $\mathcal{O} = \{w_1, w_2, w_3, w_4\} = \{1, 2, 3, 4\}$. To compare the results with the other chain in a faster way, it would be better to

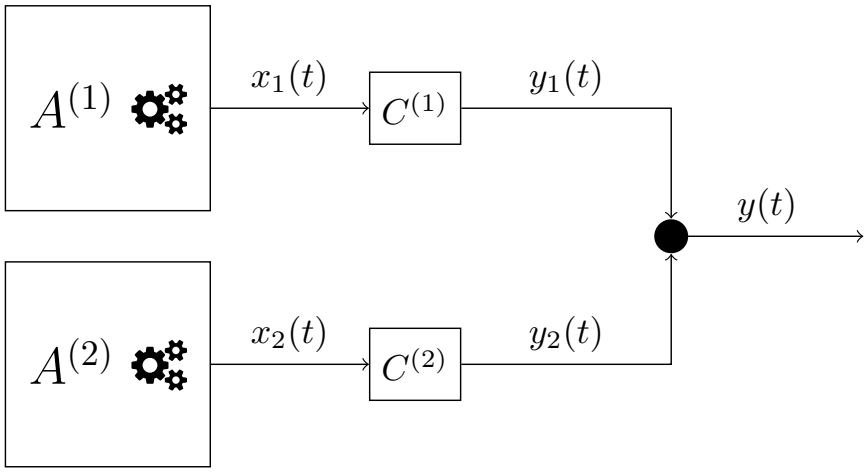


Figure 3.5: The modelled system behind a no-working workstation.

merge the pair of chains into a singular one. It is worth highlighting that the two chains are independent each other: therefore, it is not so difficult to merge them. The new merged model is:

$$\mathcal{Q}_{\theta_1} = (A^{\theta_1}, C^{\theta_1}, \pi_0^{\theta_1}) \tag{3.55}$$

where set of available state of this model are:

$$\begin{aligned}\mathcal{S}_m &= \{s_1^{(1)} \cap s_1^{(2)}, s_2^{(1)} \cap s_1^{(2)}, s_1^{(1)} \cap s_2^{(2)}, s_2^{(1)} \cap s_2^{(2)}\} \\ &= \{s_{11}^m, s_{21}^m, s_{12}^m, s_{22}^m\} = \{s_1^m, s_2^m, s_3^m, s_4^m\}\end{aligned}$$

whilst the set of the possible observations does not change. The state set is built taking into account all possible combinations of states that the two chains can provided: since each chain has two states and it is independent with regard to the other one, the number of possible states of the new model is four. This fact brings to have a transition matrix that is four-by-four: indeed, the transition matrix A^{θ_1} is built following this schema:

	$x(t+1) = s_{11}^m$	$x(t+1) = s_{21}^m$	$x(t+1) = s_{12}^m$	$x(t+1) = s_{22}^m$
$x(t) = s_{11}^m$	$a_{11}^{\theta_1}$	$a_{12}^{\theta_1}$	$a_{13}^{\theta_1}$	$a_{14}^{\theta_1}$
$x(t) = s_{21}^m$	$a_{21}^{\theta_1}$	$a_{22}^{\theta_1}$	$a_{23}^{\theta_1}$	$a_{24}^{\theta_1}$
$x(t) = s_{12}^m$	$a_{31}^{\theta_1}$	$a_{32}^{\theta_1}$	$a_{33}^{\theta_1}$	$a_{34}^{\theta_1}$
$x(t) = s_{22}^m$	$a_{41}^{\theta_1}$	$a_{42}^{\theta_1}$	$a_{43}^{\theta_1}$	$a_{44}^{\theta_1}$

Remember that $a_{ij}^{\theta_1} = \mathbb{P}\left[x(t+1) = s_j^m \mid x(t) = s_i^m\right]$. Therefore, the transition matrix of the merged model can be built as follows exploiting the independence:

$$\begin{aligned} a_{ij}^{\theta_1} &= \mathbb{P}\left[x(t+1) = s_j^m \mid x(t) = s_i^m\right] \\ &= \mathbb{P}\left[x_1(t+1) = s_{k_1} \cap x_2(t+1) = s_{k_2} \mid x_1(t) = s_{l_1} \cap x_2(t) = s_{l_2}\right] \end{aligned}$$

using k and l s.t.:

j	k_1	k_2
1	1	1
2	2	1
2	1	2
4	2	2

i	l_1	l_2
1	1	1
2	2	1
2	1	2
4	2	2

Since the two chains do not communicate with each other, the states of the chains are independent each other, therefore:

$$a_{ij}^{\theta_1} = \mathbb{P}\left[x_1(t+1) = s_{k_1} \mid x_1(t) = s_{l_1}\right] \mathbb{P}\left[x_2(t+1) = s_{k_2} \mid x_2(t) = s_{l_2}\right] = a_{l_1 k_1}^{(1)} \, a_{l_2 k_2}^{(2)}$$

Using $A^{(1)}$ and $A^{(2)}$ as transition matrices for the model with two chains, A^{θ_1} can be calculated in a more compact way as:

$$A^{\theta_1} = \begin{bmatrix} \text{Vect}(A^{(1)}(:,1) \odot A^{(2)}(:,1)) \\ \text{Vect}(A^{(1)}(:,2) \odot A^{(2)}(:,1)) \\ \text{Vect}(A^{(1)}(:,1) \odot A^{(2)}(:,2)) \\ \text{Vect}(A^{(1)}(:,2) \odot A^{(2)}(:,2)) \end{bmatrix} \tag{3.56}$$

where the Vect$(\cdot)$ function is defined in (3.53).

Similar to A^{θ_1}, C^{θ_1} is built following:

	$y(t) = 00 = w_1$	$y(t) = 01 = w_2$	$y(t) = 10 = w_3$	$y(t) = 11 = w_4$
$x(t) = s_{11}^m$	$c_{11}^{\theta_1}$	$c_{12}^{\theta_1}$	$c_{13}^{\theta_1}$	$c_{14}^{\theta_1}$
$x(t) = s_{21}^m$	$c_{21}^{\theta_1}$	$c_{22}^{\theta_1}$	$c_{23}^{\theta_1}$	$c_{24}^{\theta_1}$
$x(t) = s_{12}^m$	$c_{31}^{\theta_1}$	$c_{32}^{\theta_1}$	$c_{33}^{\theta_1}$	$c_{34}^{\theta_1}$
$x(t) = s_{22}^m$	$c_{41}^{\theta_1}$	$c_{42}^{\theta_1}$	$c_{43}^{\theta_1}$	$c_{44}^{\theta_1}$

Remembering that $c_{ij}^{\theta_1} = \mathbb{P}\left[y(t) = w_j \mid x(t) = s_i^m\right]$, the observation matrix of the merged chain C^{θ_1} can be calculated as:

$$\begin{aligned} c_{ij}^{\theta_1} =& \mathbb{P}\left[y(t) = w_j \mid x(t) = s_i^m\right] \\ =& \mathbb{P}\left[y_1(t) = w_{k_1}^s \mid x(t)_1 = s_{l_1}\right] \mathbb{P}\left[y_2(t) = w_{k_2}^s \mid x_2(t) = s_{l_2}\right] = c_{l_1 k_1}^{(1)} \, c_{l_2 k_2}^{(2)} \end{aligned}$$

using k and l s.t.:

j	k_1	k_2
1	1	1
2	2	1
2	1	2
4	2	2

i	l_1	l_2
1	1	1
2	2	1
2	1	2
4	2	2

and exploiting the fact that the observation and the states are independent between different chains. Using $C^{(1)}$ and $C^{(2)}$ as observation matrices for each sensor of the model with two chains, C^{θ_1} can be also calculated as:

$$C^{\theta_1} = \begin{bmatrix} \text{Vect}(C^{(1)}(:,1) \odot C^{(2)}(:,1)) \\ \text{Vect}(C^{(1)}(:,1) \odot C^{(2)}(:,2)) \\ \text{Vect}(C^{(1)}(:,2) \odot C^{(2)}(:,1)) \\ \text{Vect}(C^{(1)}(:,2) \odot C^{(2)}(:,2)) \end{bmatrix} \tag{3.57}$$

where the Vect$(\cdot)$ function is defined in (3.53).

3.5.3 Further Steps

The target of this opus is to develop an algorithm that allows to find what is the current model for each desk, i.e. to choose between case θ_0 and θ_1. In the first case, the model is Q_{θ_0}, i.e. the parameters of the model are $\theta = \theta_0$, whereas in the second one the parameters of the model are $\theta = \theta_1$, i.e. the model is Q_{θ_1}. Two different scenarios are available: the first, called **commissioning error detection**, is the one in which it is wanted to know if, during the commissioning, some errors were done. In this case, no assumptions about the situation are done. The other, instead, is called **failure detection**: in this case, it is assumed that the workstation is in the θ_0 case and it is wanted to detect whether the workstation has a failure, i.e. its behaviour becomes θ_1. The most immediate idea to solve the problem in these two scenarios is to use the probability of an observed event sequence in the two cases to check which is the most likely model. In a more mathematical way:

$$\theta = \underset{\theta=\{\theta_0,\theta_1\}}{\arg\max} \left[\mathbb{P}\left[Y_T \mid \theta\right]\right]$$

The pro about this idea is its simplicity: indeed, all it is computable using the Forward-Backward Algorithm (3.30). But there are some cons: no flexibility about starting and ending time and numerical problems caused by small probabilities. The first negative aspect can be solved if a window of observations is used instead of using all of them. The second can be alleviated using the ratio between the two probabilities and, therefore, making some simplifications. But, there is a smarter way to solve these problems as can be seen in the following chapter that introduces the sequential hypothesis tests.

Chapter 4

Event Detection with Time

This Chapter focuses on the decision-making process for choosing between the hypotheses presented in the last Chapter. In this one, a brief introduction about hypothesis tests, ratio tests, and sequential tests is shown: after that, the algorithm developed is presented. Finally, a discussion about thresholds and some numerical problems encountered takes place.

4.1 Available Tools

As stated in the previous Chapter, more powerful tools are needed to solve the problem: one of them is the **statistical hypothesis test**. Statistical hypothesis test is a powerful tool of statistical inference that allows to choose between two hypotheses using a data-set. The first hypothesis is called **null hypothesis** since usually is the one that it is believed it is true or the one that it is expected to be true, whilst the other one is called **alternative hypothesis** and represents the mutually exclusive option. In this work, $\mathcal{H}_0$ model presented in Section 3.5 is the null hypothesis whereas $\mathcal{H}_1$ model is the alternative one. To perform the decision, a parameter is needed for the test: it is called **significance level** and it is indicated by α. This value defines the probability of rejecting the null hypothesis when it is true, i.e. to commit the Type I error. Usually, this value is set to 1% or 5%. Table 4.1 summarises all possible errors that can be committed in during hypothesis testing. It is worth reminding that type II error cannot be set *a priori*, but it can be only calculated after setting the type I error value.

Another requirement is to define the data distribution: in this work, models have already been defined in chapter 3, but it is very difficult to define from them some probability distributions that can fit without issues. Indeed, as already seen, the simplest probability available is the one of an observed event sequence and is not easy to extract a probability distribution from that. Therefore, it is better to use a slightly different version of this tool: the **likelihood-ratio test**.

The Likelihood-ratio test is a statistical test that has the same target of the statistical hypothesis one. But, instead of relying upon a test statistic distribution of data, it is based on the likelihood ratio of the two hypotheses: usually, the log of the likelihood is used in order to avoid numerical problems and be able to use theorems for a better determination of acceptance zones. In a more mathematical

Table 4.1: Possible hypothesis test errors

	$\mathcal{H}_0$ true	$\mathcal{H}_0$ false
reject $\mathcal{H}_0$	type I error (false positive) $P = \alpha$	correct inference (true positive) $P = 1 - \beta$
accept $\mathcal{H}_0$	correct inference (true negative) $P = 1 - \alpha$	type II error (false negative) $P = \beta$

way, suppose to have a random variable X and outcomes x. The ingredients needed to use this tool are:

Total Possible Parameter Space Ω contains all possible parameters θ that can be used in the model;

Null Hypothesis $\mathcal{H}_0$**:** this hypothesis states $\theta \in \omega$ where $\omega \subset \Omega$;

Alternative Hypothesis $\mathcal{H}_1$**:** this hypothesis states $\theta \in \omega'$, ω' s.t. $\omega' \subset \Omega \quad \wedge \quad \omega \cup \omega' = \Omega \quad \wedge \quad \omega \cap \omega' = \emptyset$;

Probability Density Function $f(y|\theta)$ is the probability density function of the model when the parameter is equal to θ. The respective likelihood function is $\mathcal{L}(\theta|x)$;

The most indicated probability to be used in likelihood function is the one of an observed event sequence given the system parameters, i.e.:

$$\mathcal{L}(\theta|x) = \log \mathbb{P}\left[Y_T = x \mid \theta\right]$$

since it is easy to calculate and contains all of the information available. Remember that, the only accessible data is the one regarding the sensors and no information about state can be retrieved. Furthermore, it can be clearly seen that no one of these ingredients are function of time: this means that, using this tool, the information carried by the time flowing is not taken into account. This is not a smart idea since the system considered is a dynamic one: in fact, data is provided for every sampling time and results could be different if it is considered one moment or another. But it is not all to throw away: the basic idea is great, and all those ingredients can be got without too much effort as it was seen in Chapter 3. It would only take a small change to include the time: the solution is to use the **sequential probability ratio test** [23] to take into account the flowing time.

Before talking about sequential probability ratio test, it is useful to describe its ancestor, the **sequential hypothesis test**. Using the words "sequential hypothesis test", it is meant a sequential test of a statistical hypothesis, i.e. a procedure which gives, at every time of sampling, a specific rule about the test in order to make one of the three possible decisions. These are the following:

a) to accept the null hypothesis;

b) to accept the alternative hypothesis;

c) to continue the experiment to make additional observation since none of the formerly option can be chosen at the time.

Obviously, this test procedure is performed sequentially and ends if decision a) or b) is made.

The main difference between this approach and the previous one is that it is not known *a priori* how much time is needed to perform the decision. In this work, that value is called **Alarm Time** (t_a). Several attempts are made to provide an estimation about this issue analytically: in this work, it was decided to use a numerical approach to estimate the alarm time, i.e. performing several simulations about the algorithm.

4.2 SPRT

The Sequential Probability Ratio Test is a classic decision maker among two possible choices. It has been developed in 1943 by Abraham Wald and presented in [23]: its target was to provide a powerful algorithm that combines the effectiveness of a likelihood ratio test with the flexibility of the sequential test. Another useful characteristic is that it allows to save about 50% of observation comparing with other classic test: but the best feature of this algorithm is that it does not need to determine any probability distribution of data since only algebraic operations are needed for working.

Mathematically speaking, suppose to have a random variable $\mathcal{Y}$ with probability distribution $f(y|\theta)$, even unknown, where θ are the parameters of the assumed model, and observations $Y_t = [y(1), \dots, y(t)]$ taken one a time until t, the time when decision will be made. These observations are orthogonal each other, i.e.

$$\mathbb{E}_\theta[y(i)y(j)] = \mathbb{E}_\theta[y(i)]\mathbb{E}_\theta[y(j)] \qquad \forall\, i \neq j \tag{4.1}$$

This means that every fresh observation brings new information to the algorithm. It is wanted to test the null hypothesis $\mathcal{H}_0 : \theta = \theta_0$ against the alternative one $\mathcal{H}_1 : \theta = \theta_1$. Like in the sequential hypothesis test, three possible decisions can be made:

a) to accept the null hypothesis $\mathcal{H}_0$;

b) to accept the alternative hypothesis $\mathcal{H}_1$;

c) to continue the experiment to make additional observation since none of the formerly option can be chosen at the time $(t \to t + 1)$.

To perform the algorithm, the likelihood ratio associated to the k-th observation $y(k)$ must be defined, i.e.:

$$\lambda(k) = \log \frac{f(y(k), \theta_1)}{f(y(k), \theta_0)}$$

Furthermore, the sum of these values up to t is denoted with:

$$\Lambda(t) = \sum_{k=1}^{t} \lambda(k) = \log \prod_{k=1}^{t} \frac{f(y(k), \theta_1)}{f(y(k), \theta_0)}$$
$$= \log \frac{\prod_{k=1}^{t} f(y(k), \theta_1)}{\prod_{k=1}^{t} f(y(k), \theta_0)} = \log \frac{f(Y_t, \theta_1)}{f(Y_t, \theta_0)}$$

and represents the log-likelihood ratio for the two hypotheses until t. The algorithm is very simple: whilst $\Lambda(t)$ is in the thresholds area, a new sample is needed. If $\Lambda(t)$ is lower than the first threshold, the decision a) is made whereas if $\Lambda(t)$ is higher than the second threshold, the decision b) is made. This algorithm is reported in a more elegant way in Algorithm 3. The problem now is to define these thresholds, η_0

Algorithm 3 SPRT algorithm

```
t=1
Λ = 0
while η1 > Λ > η0 do
    get new observation y(t)
    Λ = Λ + log f(y(t), θ1) / f(y(t), θ0)
    t++
end while
if η1 > Λ then
    return H0 refused
else
    return H0 accepted
end if
```

and η_1. The best idea is to link them to misdetection and false alarm probabilities. The first is also known as Type I error probability, i.e.

$$p_m = \mathbb{P}\left[\texttt{say } \mathcal{H}_0 \mid \mathcal{H}_1 \texttt{ is true}\right]$$

whereas the second is the Type II error probability, i.e.

$$p_f = \mathbb{P}\left[\texttt{say } \mathcal{H}_1 \mid \mathcal{H}_0 \texttt{ is true}\right]$$

Using these values, Wald proved that the thresholds must satisfy the following inequalities:

$$\eta_0 \geq \log \frac{p_m}{1 - p_f} \quad \text{and} \quad \eta_1 \leq \log \frac{1 - p_m}{p_f} \tag{4.2}$$

to have:

$$p_m \leq \mathbb{P}\left[\texttt{say } \mathcal{H}_0 \mid \mathcal{H}_1 \texttt{ is true}\right] \quad \text{and} \quad p_f \leq \mathbb{P}\left[\texttt{say } \mathcal{H}_1 \mid \mathcal{H}_0 \texttt{ is true}\right]$$

Therefore, the best idea for the thresholds is to use:

$$\eta_0 = \log \frac{p_m}{1 - p_f} \quad \text{and} \quad \eta_1 = \log \frac{1 - p_m}{p_f} \tag{4.3}$$

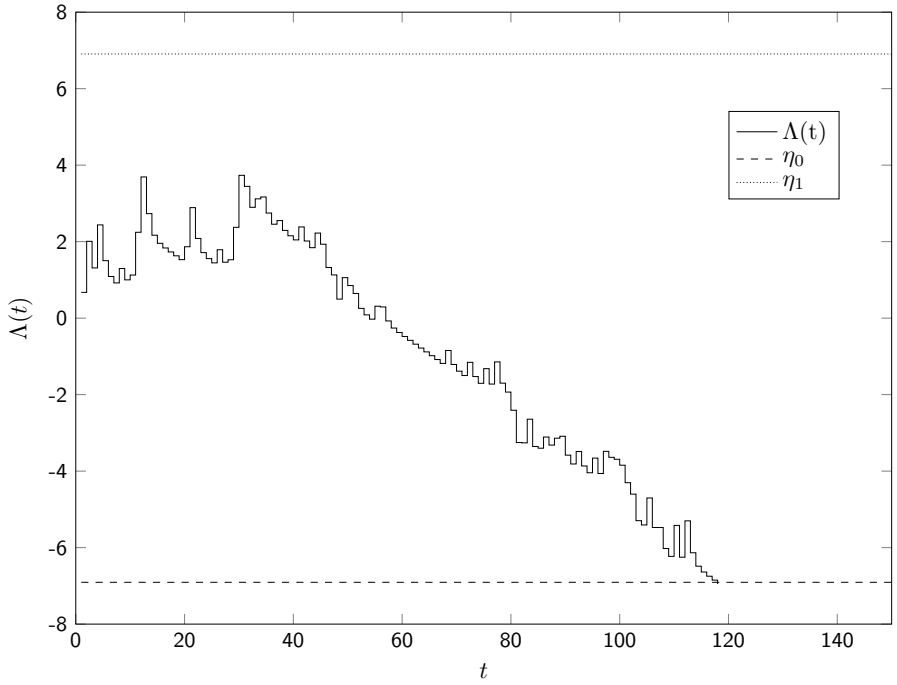


Figure 4.1: Example of SPRT $\Lambda(t)$ behaviour when the null hypothesis is true using $p_m = p_f = 1\%$.

since this choice allows to satisfy the inequalities using the minimum no-threshold area.

In Figure 4.1 it is shown the behaviour of $\Lambda(t)$ when the true model is the one in $\mathcal{H}_0$. Instead, in Figure 4.2 the behaviour of $\Lambda(t)$ when the true model is the one in $\mathcal{H}_1$ is shown. In both of the cases, it can be seen that the algorithm stops to calculate $\Lambda(t)$ when one of the threshold is reached.

Watching the pictures, this algorithm seems perfect for solving the first problem, i.e. how to check if the commissioning of the lamps has been done correctly. Indeed, in this case nobody knows which the correct hypothesis is and therefore both of them must be taken into account. It remains open the issue about what to use as function f in the log ratio. But before to explain that, it is better to think to a way for solving the other problem, i.e. how to check if lamps have been moved.

4.3 CUSUM

Using the sequential probability ratio test for checking if lamps have been moved is not a so smart idea. Usually lamps are not moved or, if moved, nobody knows whenever it happened. Therefore, using this algorithm, a very poor result will be obtained since it stops when it finds which the hypothesis is true, and it does not monitor model changes. A new algorithm is needed: the main feature requested is, given an assumed model, to identify whenever a change happens. It is a slightly difference between the previous seen algorithm: this algorithm can stop only when a change is detected, not when it is sure about which model observations come from. To solve this, E.S. Page developed and reported in [24] the **CUmulative**

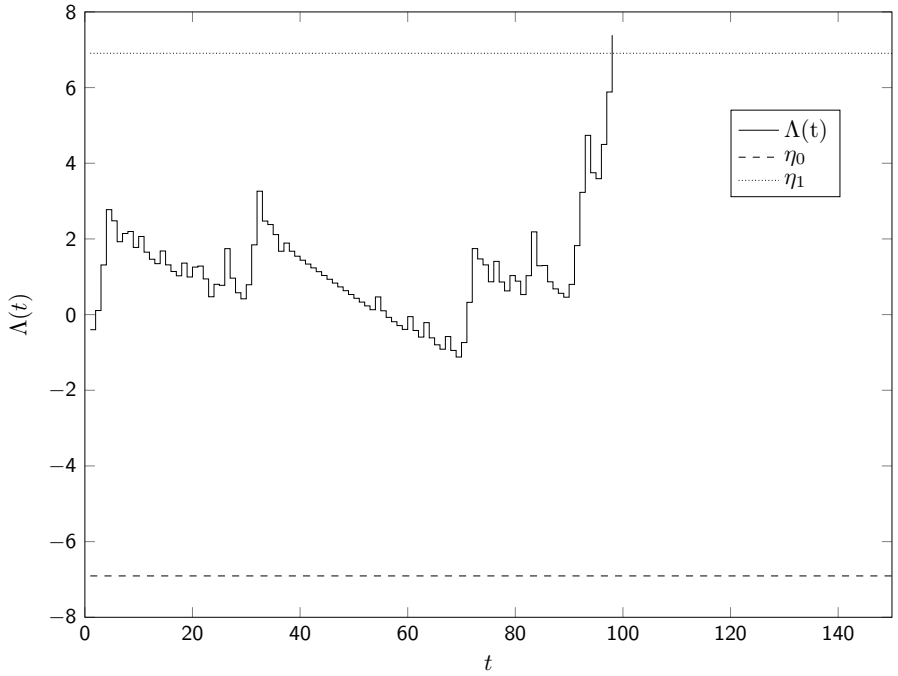


Figure 4.2: Example of SPRT $\Lambda(t)$ behaviour when the null hypothesis is false using $p_m = p_f = 1\%$.

SUM algorithm.

Similar to the sequential probability ratio test, for the CUSUM Algorithm [25] the ingredients are a random variable $\mathcal{Y}$ with independent and identically distributed samples $y(k)$ for $k < t$. Each of this sample follows the probability distribution $f(y|\theta)$, even unknown, where θ are the parameters of the assumed model. It is assumed that there are two possible models and a time T_c in which the used model is changed, i.e. samples from 1 to $T_c - 1$ come from a model with parameters $\theta = \theta_0$ whereas the samples from T_c to t come from a model with parameters $\theta = \theta_1$. This value, T_c, is called **change time**. Only θ_0 and θ_1 are available whilst T is unknown, and it has to be found. Therefore, there are two possible hypotheses in time k:

- $\mathcal{H}_0$: no-change hypothesis. In this case, the probability density function is:

$$f(Y_t|\mathcal{H}_0) = \prod_{i=1}^{t} f(y(i), \theta_0)$$

- $\mathcal{H}_1$: one-change hypothesis. In this, instead, the probability density function becomes:

$$f(Y_t|\mathcal{H}_1) = \prod_{i=1}^{T_c-1} f(y(i), \theta_0) \prod_{i=T_c}^{t} f(y(i), \theta_1)$$

As already stated, the main target of the CUSUM is to decide whether hypothesis is true. If $\mathcal{H}_0$ is the one true, then the algorithm should continue. Otherwise, the algorithm should stop and provide an estimation about T_c. It can be seen there are two operations to be done in this algorithm: the first is to **detect** which hypothesis

is true whereas the second is to **estimate** the change time. Obviously, this step should be done only when the first step returns a change.

The first step, also known as **detection step**, seeks to decide which hypothesis is true: in other words, it tries to detect if the model has changed. It based on the fact that the log-likelihood ratio

$$\lambda(t) = \log \frac{f(y(t), \theta_1)}{f(y(t), \theta_0)}$$

changes sign when a parameters modification is done, i.e. $\Lambda(t)$ changes slope sign: this can be seen in Figure 4.3. Indeed, there it can be seen that, before the change,

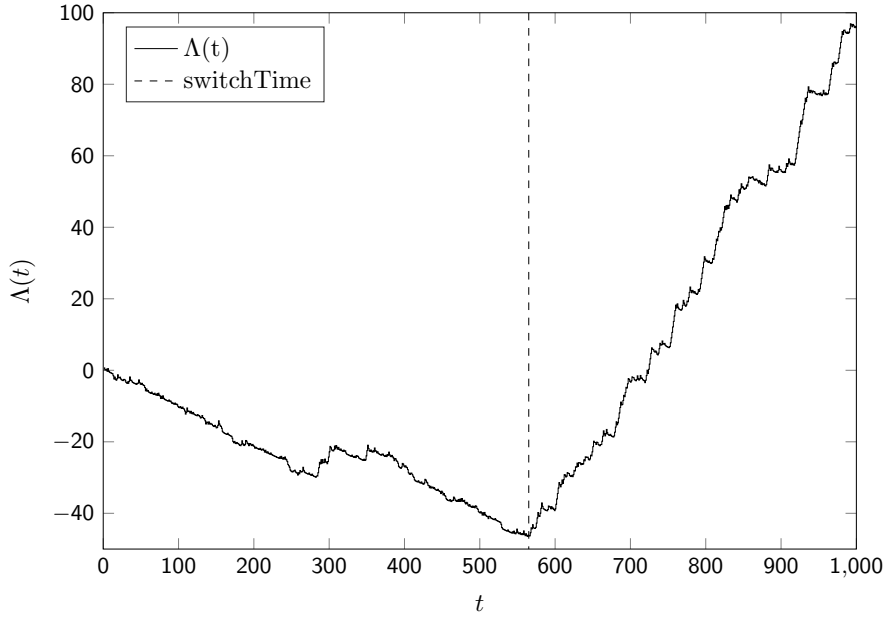


Figure 4.3: Example of SPRT $\Lambda(t)$ behaviour: the change of parameters is performed at $t = 565$.

the slope of $\Lambda(t)$ is positive: after the change, instead, the slope becomes negative. The idea is to exploit that using some moving thresholds instead of measuring the slope to have a more robust algorithm. A further footstep about this idea is to set to zero negative values of $\Lambda(t)$: indeed, the knowledge about that $\mathcal{H}_0$ is true is assumed and therefore it is needed to check only whether the slope becomes negative. If this is done, it is possible to recycle the SPRT algorithm: indeed, if the first hypothesis is true, it is like that the algorithm has never been started since $\Lambda(t)$ is zero or very close to. On the other side, if the null hypothesis becomes false, the algorithm works like SPRT and therefore it can use the same thresholds.

The second step is called **estimation step** and tries to estimate the value of T_c. Exploiting the previous idea, the best estimation is to use the last time in which $\Lambda(k)$ changes slope before the detection step claims that there was a change, i.e., if all negative values of $\Lambda(k)$ are set to zero, the last time when $\Lambda(k)$ was zero.

These ideas can be written in a more mathematical way [25] in order to prove their correctness. First of all, $\lambda(k)$ can be written, exploiting some properties of

the summations to show the recursive property, as:

$$\Lambda(k) = \lambda(k) + \Lambda(k-1)$$

Exploiting the sequential probability ratio test algorithm for the two hypotheses and their definitions, the best way to write down $\Lambda(t)$ is:

$$\begin{aligned}
\Lambda(t) &= \sum_{i=1}^{t} \log \frac{f(y(i), \mathcal{H}_1)}{f(y(i), \mathcal{H}_0)} \\
&= \log \prod_{i=1}^{t} \frac{f(y(i), \mathcal{H}_1)}{f(y(i), \mathcal{H}_0)} \\
&= \log \frac{\prod_{i=1}^{t} f(y(i), \mathcal{H}_1)}{\prod_{i=1}^{t} f(y(i), \mathcal{H}_0)} \\
&= \log \frac{f(Y_t \mid \mathcal{H}_0)}{f(Y_t \mid \mathcal{H}_1)} \\
&= \log \frac{\prod_{i=1}^{T_c-1} f(y(i), \theta_0) \prod_{i=T_c}^{t} f(y(i), \theta_1)}{\prod_{i=1}^{t} f(y(i), \theta_0)} \\
&= \log \frac{\prod_{i=T_c}^{t} f(y(i), \theta_1)}{\prod_{i=T_c}^{t} f(y(i), \theta_0)} \\
&= \log \prod_{i=T_c}^{t} \frac{f(y(i), \theta_1)}{f(y(i), \theta_0)} \\
&= \sum_{i=T_c}^{t} \log \frac{f(y(i), \theta_1)}{f(y(i), \theta_0)} \\
&= \sum_{i=T_c}^{t} \log \frac{f(y(i), \theta_1)}{f(y(i), \theta_0)} \\
&= \sum_{i=T_c}^{t} \lambda(i)
\end{aligned}$$

It is easy to realise that this is the formula used in a sequential probability ratio test with starting time $t = T_c$ instead of the common one $t = 0$. The problem is that nobody known when the change is made, i.e. when start to sum and compare with the thresholds. An idea is to use a generalised log-likelihood, as suggested by [25], i.e.

$$\Lambda_G(t) = \max_{T_c \in [1,\dots,t]} \left[\sum_{i=T_c}^{t} \lambda(i) \right]$$

In this case, only $\mathcal{H}_1$ is checked: in other words, only η_1 threshold is utilised. Obviously, the best estimation for T_c is the argmax of that expression. As it can be found in [26, Chap. 2], instead of doing the max/argmax at each time t to found if a change was made and estimate when, a recursive algorithm can be used, like the one shown in Algorithm 4. Indeed, it can be proven that the $\Lambda_G(t)$ can be also

written as:

$$\Lambda_G(t) = \begin{cases} 0 & \text{if } \Lambda_G(t-1) + \lambda(t) < 0 \\ \Lambda_G(t-1) + \lambda(t) < 0 & \text{otherwise} \end{cases}$$

and, if at time $t_a = t$ the hypothesis $\mathcal{H}_1$ is accepted,

$$T_c = t_a - N_{t_a} + 1$$

where N_{t_a} is the last time when $\Lambda_G(t) = 0$. This improved version of the algorithm

Algorithm 4 CUSUM algorithm

```
t=1
N_{t_a} = 0
Λ_G = 0
while Λ_G < η_1 do
    get new observation y(t)
    Λ_G = Λ_G + log f(y(t), θ_1) / f(y(t), θ_0)
    if Λ_G < 0 then
        Λ_G = 0
        N_{t_a} = 0
    else
        N_{t_a}++
    end if
    t++
end while
return H_1 accepted, T_c = t − N_{t_a}
```

allows to be used on-line, that is necessary in this opus. A possible behaviour of $\Lambda_G(t)$ can be found in Figure 4.4. There, it can be seen that, in the first part of the graph, Λ_G is zero or very close to it. The main cause of this behaviour can be found in the high noisy measurements. After the change, instead, Λ_G starts to grow until it reaches the threshold η_1 and thus the algorithm stops. A good idea for η_1 is to use

$$\eta_1 = \log \frac{1 - p_m}{p_f}$$

since, as already proved before, the CUSUM is a sequential probability ratio test with a moving start. Obviously, since the test is repeated until a change is detected, the value of p_m does not represent the probability of a false alarm, but their frequency.

4.4 SPRT and CUSUM for HMM

In the previous two sections, two methods for hypothesis checking were shown. The first one, the sequential probability ratio test, is useful for testing which model is currently working whereas the second, the CUSUM algorithm, is preferable for

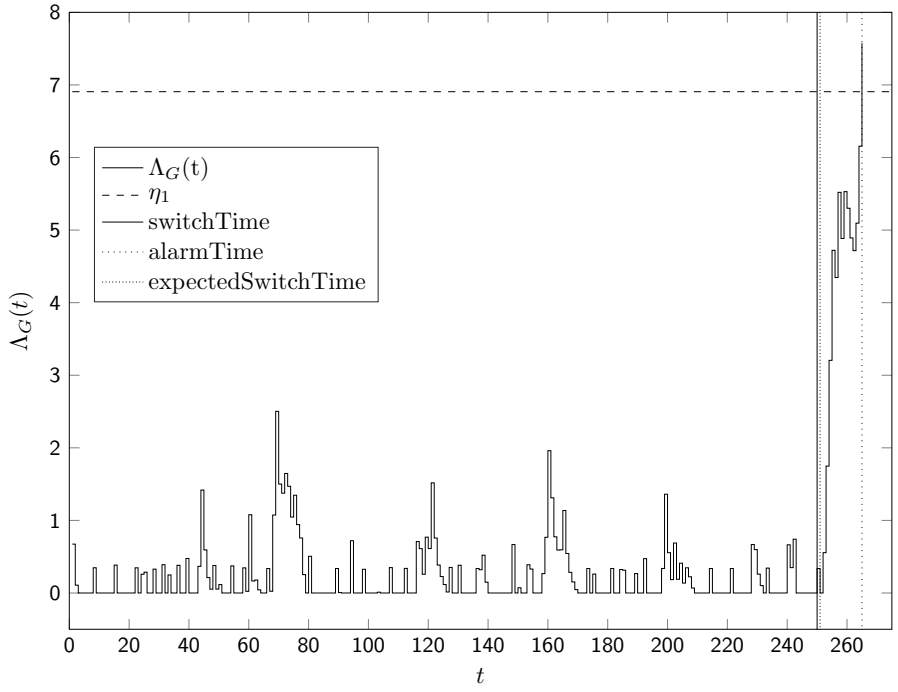


Figure 4.4: Example of CUSUM $\Lambda_G(t)$ behaviour: the change of parameters is performed at $Ts = 250$ (switch time) whereas the algorithm returns $T_c = 251$ (expected switch time) at $T_a = 267$ (alarm time).

checking whether a model changes and finding the moment when it happens. The problem now is how to use these tools on hidden Markov model, i.e. finding what function $f(y(k), \theta)$ should be used in the two algorithms. The idea is to use the probability of $y(k)$ given the system: this is an almost obligatory choice since $y(k)$ is the only thing that it can be measured in our system. Before doing the maths, a reminder should be made: from Chapter 3, all systems used in this opus can be written as:

$$\mathcal{Q}_\theta = (A_\theta, C_\theta, \pi_{0_\theta})$$

where A_θ is the transition matrix, i.e. $a_{ij} = a_i(j) = \mathbb{P}[x_{t+1} = s_j \mid x_t = s_i]$ whereas C_θ the observation one, i.e. $c_{ij} = c_i(j) = \mathbb{P}[y_t = w_j \mid x_t = s_i]$. The θ subscript is utilised to remember that there are two models that differ for the matrices A_θ and C_θ as already stated in Section 3.5.

The probability $y(k)$ given the system cannot be directly calculated since the only information available, i.e. A and C, is function of the current state, that it is not freely measurable. Therefore, some manipulations are required to make explicit the dependence from the states:

$$\begin{aligned} f(y(k), \theta) &= P[y(t) \mid \theta] \\ &= \sum_{x_j \in \mathcal{S}} P[y(k) \mid x_j, \theta] P[x_j \mid \theta] = \sum_{x_j \in \mathcal{S}} C(x_j, y(k)) P[x_j \mid \theta] \end{aligned}$$

There was used the marginalisation probability property (3.5). As it can be seen, the summation requires two elements: the first is quite easy to be accessed whereas the

problem is to calculate $P[x_j \mid \theta]$ since in hidden Markov models a state depends only on previous state and it is not directly measurable. Therefore, there is not a direct way to measure the probability of a state without using complex algorithms like the ones shown in Chapter 3. A possible workaround is to use previous observations to develop a fast-calculable formula:

$$\begin{aligned} f(y(k), Y_{0:k-1}, \theta) &= P[y(t) \mid Y_{0:k-1}, \theta] \\ &= \sum_{x_j \in \mathcal{S}} P[y(k) \mid x_j, \theta] P[x_j \mid Y_{0:k-1}, \theta] \\ &= \sum_{x_j \in \mathcal{S}} C_\theta(x_j, y(k)) P[x_j \mid Y_{0:k-1}, \theta] \end{aligned}$$

It is worth noting that $P[y(k) \mid x_j, Y_{0:k-1}, \theta] = P[y(k) \mid x_j, \theta]$: this derives from the link between states and observations that appears using hidden Markov models. Indeed, an observation depends only on the current state of the chain and indirectly on previous observations since they are outcomes of past states that influence the current one. This idea has one negative aspect: the independence between measurements is not still valid since instead of using $f(y(k), \theta)$, $f(y(k), Y_{0:k-1}, \theta)$ is used. In other words, the measurement used is $(y(k) \mid Y_{0:k-1})$ instead of $y(k)$. If (4.1), that describes independence between measurements, is used, it is easy to prove that, if $(y(k), Y_{0:k-1})$ is used instead of $y(k)$, the independence is not present. However, this does not preclude the use of the previous algorithms and the formulae to calculate the thresholds since Wald in [23, p. 130] wrote that the independence is useful only to prove faster (4.2) but these are still valid even if independence of measurements is dropped. Therefore, a remark should be done:

Remark. *The independence of the observations is not necessary for the validity of* (4.2).

Obviously, this applies on CUSUM too, since CUSUM algorithm can be seen as an SPRT algorithm with different starting point. Therefore, it is possible to pursuing that idea: from Chapter 3,

$$P[x_t = x_i \mid Y_{0:t-1}] = \frac{\alpha_{t|t-1}(i)}{\sum_{j \in \mathcal{S}} \alpha_{t|t-1}(j)}$$

So, putting all together:

$$f(y(t), Y_{0:t-1}, \theta) = \sum_{x_j \in \mathcal{S}} C_\theta(x_j, y(t)) \frac{\alpha_{t|t-1}(j)}{\sum_{x_l \in \mathcal{S}} \alpha_{t|t-1}(l)} \tag{4.4}$$

The previous formula can be written in a more compact way:

$$f(y(t), Y_{0:t-1}, \theta) = \frac{\alpha_{t|t-1} C_\theta(:, y(t))}{\sum_{x_l \in \mathcal{S}} \alpha_{t|t-1}(l)} \tag{4.5}$$

where, to speed up the reckon in MATLAB,

$$\alpha_{t|t-1} = \alpha_{t-1|t-1} A_\theta = \left(\alpha_{t-1|t-2} \odot C_\theta(:, y_{t-1})^\intercal\right) A_\theta$$

This formula can be rewritten as future iteration form:

$$\alpha_{t+1|t} = \left(\alpha_{t|t-1} \odot C_\theta(:, y_t)^\intercal\right) A_\theta$$

to allow having in-place algorithms as it can be seen later in this Chapter. At this stage, with these information, it is possible to develop and write down the two algorithms to solve the two problems.

4.4.1 Commissioning Error Detection

The commissioning error detection is the first problem analysed: the target is to check whether some errors are made during commissioning. In other words, this problem aims to find if exists any pair of lamps that is not over the same workstation. Obviously, there are only two possible situations in each workstation:

a) The commissioning was done correctly;

b) Some errors were made during commissioning.

The issue is that none of them are known true*a priori*. Therefore, the best idea is to use the sequential probability ratio test with $f(y(t), Y_{0:t-1}, \theta)$ for each workstation in the area. A possible implementation for a workstation is shown in Algorithm 5.

Algorithm 5 Commissioning Error Detection algorithm (SPRT)

`t=1`
$\alpha_{1|0}^{\theta_1} = \pi_0^{\theta_1}$
$\alpha_{1|0}^{\theta_0} = \pi_0^{\theta_0}$
$\Lambda = 0$
while $\eta_1 > \Lambda > \eta_0$ **do**
`get new observations` $y_1(t)$ `and` $y_2(t)$ `and merge into` $y(t)$
$\Lambda = \Lambda + \log \dfrac{\alpha_{t|t-1}^{\theta_1} C_{\theta_1}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha_{t|t-1}^{\theta_1}(l)} - \log \dfrac{\alpha_{t|t-1}^{\theta_0} C_{\theta_0}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha_{t|t-1}^{\theta_0}(l)}$
$\alpha_{t+1|t}^{\theta_1} = \left(\alpha_{t|t-1}^{\theta_1} \odot C_{\theta_1}(:, y_t)^\intercal\right) A_{\theta_1}$
$\alpha_{t+1|t}^{\theta_0} = \left(\alpha_{t|t-1}^{\theta_0} \odot C_{\theta_0}(:, y_t)^\intercal\right) A_{\theta_0}$
`t++`
end while
if $\eta_1 > \Lambda$ **then**
return $\mathcal{H}_0$ `refused (commissioning: error)`
else
return $\mathcal{H}_0$ `accepted (commissioning: OK)`
end if

The input data is the measurements taken from the lamps that are believed to be a pair, i.e. $y_1(t)$ and $y_2(t)$. To be used, these two measurements are merged following Table 4.1. As thresholds η_0 and η_1, the formulae (4.3) provided by Wald can be exploited or they can be defined using a trial-and-error approach. Algorithm

5 is an in-place algorithm, i.e. the memory space required does not change during time, since only Λ and α values must be stored. Therefore, the space complexity of this algorithm is $O(1)$. From a computational complexity point of view, instead, it is more difficult to say anything about. However, it can be seen that the computational complexity does not change during time. More useful is to analyse the behaviour of the algorithm when multiple workstations are monitored. Indeed, this algorithm can be implemented in a distributed way, i.e. in each workstation in the office space, or it can be implemented in a machine that performs Algorithm 5 for all the workstations, using a centralised solution. For the latter case, it is sufficient to add an external for-loop for each workstation. However, this is not a huge problem since this is a *una tantum* situation. Anyway, since a for-loop is enough to monitor an entire office, the computational complexity for a centralised monitoring solution is $O(n)$, where n is the number of monitored workstations. Performances of this algorithm can be found in Chapter 5.

4.4.2 Failure Detection

The other problem analysed in this opus is to check if, after a correct commissioning, any of the lamps changes position, i.e. a pair of lamps is split. In this issue, for each workstation, there are two situations:

a) The pair is not split;

b) The pair was coupled but now is split.

The best idea to solve this is to use the CUSUM algorithm with $f(y(t), Y_{0:t-1}, \theta)$. Indeed, a) is the $\mathcal{H}_0$ of a CUSUM test whereas b) is the $\mathcal{H}_1$ hypothesis. A possible implementation for a workstation is shown in Algorithm 6. As threshold η_1, the formula present in (4.3) provided by Wald can be used or it can be defined using a trial-and-error approach.

Like for the previous one, the input data for this algorithm are the measurements taken from the lamps that are believed to be a pair, i.e. $y_1(t)$ and $y_2(t)$. To be used, these two measurements are merged following Table 4.1. Similar to the previous one algorithm, this one is an in-place algorithm, since only Λ and α values must be stored. Therefore, the space complexity of this algorithm is $O(1)$. From a computational complexity point of view, instead, it is difficult to say nothing but the complexity does not grow during time: since it must be performed until a change is detected but there is no certainty that this will happen, a computational complexity index has very little utility. More useful is to find how it scales monitoring more workstations in order to know how fast the CPU has to be. Indeed, like for the previous one, this algorithm can be implemented or in a distributed way, i.e. in each workstation in the office space, or it can be implemented using a centralised solution on a machine that performs Algorithm 6 for all the workstations using a for-each loop. Therefore, the centralised solution has a linear increase in complexity compared to the number of workstations, i.e. computational complexity is $O(n)$ where n is the number of monitored workstations. This time, instead, concurrent execution of this algorithm is crucial because it must be performed 24/7 for each workstation to check whether failures occur. Obviously, some optimisation can be

Algorithm 6 Failure Detection algorithm (CUSUM)

`t=1`
$N_{t_a} = 0$
$\alpha^{\theta_1}_{1|0} = \pi^{\theta_1}_0$
$\alpha^{\theta_0}_{1|0} = \pi^{\theta_0}_0$
$\Lambda_G = 0$
while $\Lambda_G < \eta_1$ **do**
  `get new observations` $y_1(t)$ `and` $y_2(t)$ `and merge into` $y(t)$
  $\Lambda = \Lambda + \log \dfrac{\alpha^{\theta_1}_{t|t-1} C_{\theta_1}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha^{\theta_1}_{t|t-1}(l)} - \log \dfrac{\alpha^{\theta_0}_{t|t-1} C_{\theta_0}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha^{\theta_0}_{t|t-1}(l)}$
  $\alpha^{\theta_1}_{t+1|t} = \left(\alpha^{\theta_1}_{t|t-1} \odot C_{\theta_1}(:, y_t)^\intercal\right) A_{\theta_1}$
  $\alpha^{\theta_0}_{t+1|t} = \left(\alpha^{\theta_0}_{t|t-1} \odot C_{\theta_0}(:, y_t)^\intercal\right) A_{\theta_0}$
  if $\Lambda_G < 0$ **then**
    $\Lambda_G = 0$
    $N_{t_a} = 0$
  else
    N_{t_a}`++`
  end if
  `t++`
end while
return $\mathcal{H}_1$ `accepted - Failure detected on,` $T_c = t - N_{t_a}$

done according to requirements, e.g. reduce the sampling time, even if they could be unnecessary since the algorithm is quite lightweight. For further information, check Chapter 5.

4.5 Numerical Problems

The Algorithms 5 and 6 are quite powerful as it can be seen in Chapter 5 but they are not so numerically stable. Indeed, the calculation of $\alpha_{t|t-1}$ is nothing but a series of summations and multiplications with decimal numbers, and this leads to have very small values. These values, if the monitoring time is quite long, can be close to machine epsilon and thus gives numerical problems with divisions or logarithms. A solution can be found analysing in particular the $f(y(t), Y_{0:t-1}, \theta)$ formula. Remember that

$$\begin{aligned} f(y(t), Y_{0:t-1}, \theta) &= \frac{\alpha_{t|t-1} C_\theta(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha_{t|t-1}(l)} \\ &= \frac{\alpha_{t|t-1}}{\sum\limits_{x_l \in \mathcal{S}} \alpha_{t|t-1}(l)} C_\theta(:, y(t)) \end{aligned}$$

There it can be seen that a normalisation of $\alpha_{t|t-1}$ is done: therefore, if that normalisation is performed directly during the calculation of $\alpha_{t|t-1}$ given $\alpha_{t-1|t-2}$,

the numerical problem can be attenuated. Thus, in Algorithms 5 and 6, it is better to use:

$$\bar{\alpha}_{t+1|t} = \frac{\alpha_{t+1|t}}{\sum\limits_{x_l \in \mathcal{S}} \alpha_{t+1|t}(l)}$$

along with

$$f(y(t), Y_{0:t-1}, \theta) = \bar{\alpha}_{t|t-1} C_\theta(:, y(t))$$

It is worth highlighting that, without these optimisation, the algorithms become worthless: even if MATLAB[1] has a default machine epsilon $\epsilon = 4.9407 \times 10^{-324}$ that it seems big, often it is not enough. Indeed, the operations done over α are:

$$\alpha_{t+1|t} = \left(\alpha_{t|t-1} \odot C_\theta(:, y_t)^\intercal\right) A_\theta$$

i.e.

$$\alpha_{t+1|t}(i) = \sum_{s_j \in \mathcal{S}} a_{ji}\, \alpha_{t|t}(j) C_\theta(i, y_t)$$

where each element of A_θ and C_θ is smaller than one. Since these are row-stochastic matrices, in each row there is one element lower or equal to 0.5. Let us assume, to have easier reckons, that A_θ and C_θ are square matrices 2×2 with all elements equal to 0.5, i.e. use a hidden Markov model with 2 states and 2 possible events. Therefore, the previous formula becomes:

$$\alpha_{t+1|t}(i) = 0.25 \sum_{s_j \in \mathcal{S}} \alpha_{t|t}(j)$$

Since $\alpha_{1|0} = \pi_0$, the sum of its element is equal to one. Let us assume that $\pi_0 = [0.5\, 0.5]$, a common choice if no data is available. The formula therefore becomes:

$$\alpha_{t+1|t}(i) = \alpha_{t+1|t}(i) = 0.25^{t-1}$$

that is lower than the machine epsilon for

$$0.25^{t-1} = \epsilon \quad \rightarrow \quad t = \log_{0.25} \epsilon + 1 = 537 + 1 = 538$$

i.e. after 538 samples the value of α becomes too small to be distinguished from zero. This is not a high value, especially in the second algorithm that it must be executed until a fault is detected. Indeed, using a one-hour sampling time, which could be too much slow, 538 samples represent only three weeks. Furthermore, if a one-minute sampling time is used, that is a common value, only up to 9 hours can be monitored before to have issues. This proves that the normalisation introduced in the first part of this section is fundamental for having working algorithms.

[1] The machine epsilon in MATLAB for quasi-zero values can be found using `eps(0)`.

Chapter 5

Simulations

In Chapter 4, the algorithms to solve the commissioning error detection and failure detection problems were shown. In this chapter, these algorithms will be implemented in MATLAB and some simulations will be shown paying close attention to the algorithms behaviour using different of model parameters and thresholds. However, before doing that, a brief introduction about how to implement a hidden Markov model in MATLAB will be done.

5.1 Implementation

For this opus, MATLAB tool-chain was chosen since it allows to implement hidden Markov models and the developed algorithm in an easier way than using more powerful coding languages like C or Java. Furthermore, it includes a lot of tools for plotting and analysing the obtained results. To speed up the coding and producing a more robust code, it was decided to use an object-oriented approach for building the model along with a more functional approach for implementing the algorithms.

Three classes and several functions were developed to implement the algorithms and the model. The developed classes are:

Workstation this class defines the model of a workstation, with sensors and people.

SPRT this class defines the needed parameters and functions to implement a sequential probability ratio test.

CUSUM this class defines the needed parameters and functions to implement the CUSUM algorithm.

In the following subsections they will be analysed in particular to better investigate their behaviour.

5.1.1 Workstation class

Figure 5.1 shows the UML diagram of the Workstation class.

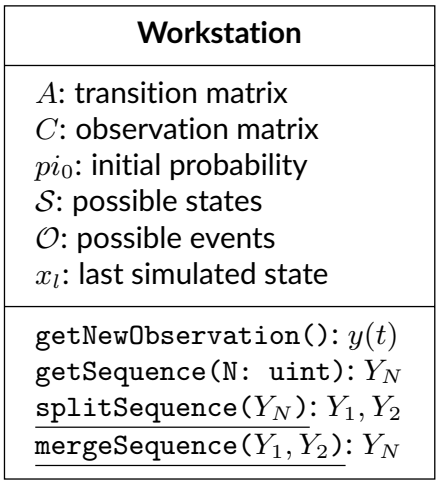


Figure 5.1: UML Diagram of class Workstation.

From the Figure, it can be seen that this class implements several tasks: defining and representing a workstation as well as providing a simulation of it. To do this, several parameters and variables must be defined. The first ones are the parameters of the hidden Markov model to be simulated, that, as seen in Chapter 3, are the transition matrix A, the observation matrix C, the initial states probabilities π_0, and the states and observable events sets $\mathcal{S}$ and $\mathcal{O}$. Since the workstation, in this opus, is modelled as a hidden Markov model with two states and four possible observations (see Chapters 2 and 3), the previous stated parameters can be represented in MATLAB using well defined matrices and vectors. Indeed, A is a $\mathbb{R}^{2\times2}$ matrix, C is a $\mathbb{R}^{2\times4}$ matrix, π_0 is a two columns row vector, whereas the two sets are represented using two row vectors that contain all the possible values. Another value is necessary to include in this class: the last state field. Indeed, if a single step simulation is done, i.e. to provide a new sample every time the function is called, a record about the last state is necessary. Furthermore, it is also sufficient since states and observations of a hidden Markov model depends only on the previous state. In order to follow the information hiding principle, which allows to reduce bugs and issues in coding, all of these parameters have a private scope, i.e. the cannot be accessed outside of the class.

Along with variables, methods and functions are necessary for building a class too. The first method is the constructor, i.e. the function that allows to initialise the previously described parameters. All of them are directly initialised by the constructor except for the last state parameter: indeed, this one is initialised using a random function based on π_0 to represent the initial state. In MATLAB this can be done using the function `randsrc(m,n,[S; prob])`: it takes as input two integer values, `m` and `n`, whereas a matrix composed by two rows in which the first is an alphabet `S`, and the second their probabilities `prob`, and returns a matrix `m` times `n` which entry independently is chosen from entries in alphabet taking into account their probabilities. For this work, it is used:

```
x_l = randsrc(1,1,[S; pi0])
```

since only one sample is required with possible states in $\mathcal{S}$ with probabilities π_0.

The second included method in this class is the `getNewObservation` method.

As the name suggests, this function simulates a new observation by moving the chain one step further. Obviously, the new status will be saved in x_l. The algorithm behind this function is reported in Algorithm 7. The first operation is to extract

Algorithm 7 getNewObservation

```
probState1 = A(x_l,1)
if rand()>probState1 then
    x_l = S(2)
else
    x_l = S(1)
end if
probObserv = C(x_l,:)
y_l = randsrc(1,1,[0; probObserv])
return y_l
```

the probability of going to state s_1 given the state x_l. This information is available and can be found in the A matrix. Since only two states are available, there is no need to check the other probability value. The second step is to generate a random value that belongs to the uniform distribution between zero and one. If this value is higher than the probability found in the previous step, then the new state will be the s_2, otherwise the new state will be s_1. The further step is to provide the new observation: therefore, the probability of all possible events given the current state is extracted from C matrix and, using the `randsrc` MATLAB function, a new observation is simulated and returned. For debugging purposes and to analyse the algorithms performances, the simulated state is returned among with the observation.

Sometimes is also useful to simulate an entire sequence: for this cause, the `getSequence` method was developed. This function does nothing but implement a for cycle to run `N` times the Algorithm 7. It returns the sequence of the simulated observations along with the sequence of the simulated states.

The Workstation class implements two more algorithms: `mergeSequence` and `splitSequence`. These two methods were developed for allowing to split a workstation, and merge the outcomes of two different simulated workstation. Indeed, the `getNewObservation` method uses the compact notation for observed events. The `splitSequence` method requires as input a sequence of observations created using `getNewObservation` or `getSequence` methods: even a single observation can be used since it is a sequence with length equal to one. It returns the two sensors corresponding sequences implementing Table 2.1. On other hand, `mergeSequence` implements the opposite function: it required two same-length sensor-observation sequences created using `splitSequence`, even from different iterations, and it returns the corresponding merged sequences following Table 2.1. These two functions are used together for designing the simulation workbench.

5.1.2 SPRT and CUSUM classes

Before illustrating the simulation workbench, it is useful to show how to implement the two algorithms developed for this work, i.e. the Commissioning Error

Detection algorithm (Algorithm 5) and the Failure Detection algorithm (Algorithm 4). In Figure 5.2 the UML diagrams for the Commissioning Error Detection and the Failure Detection are shown.

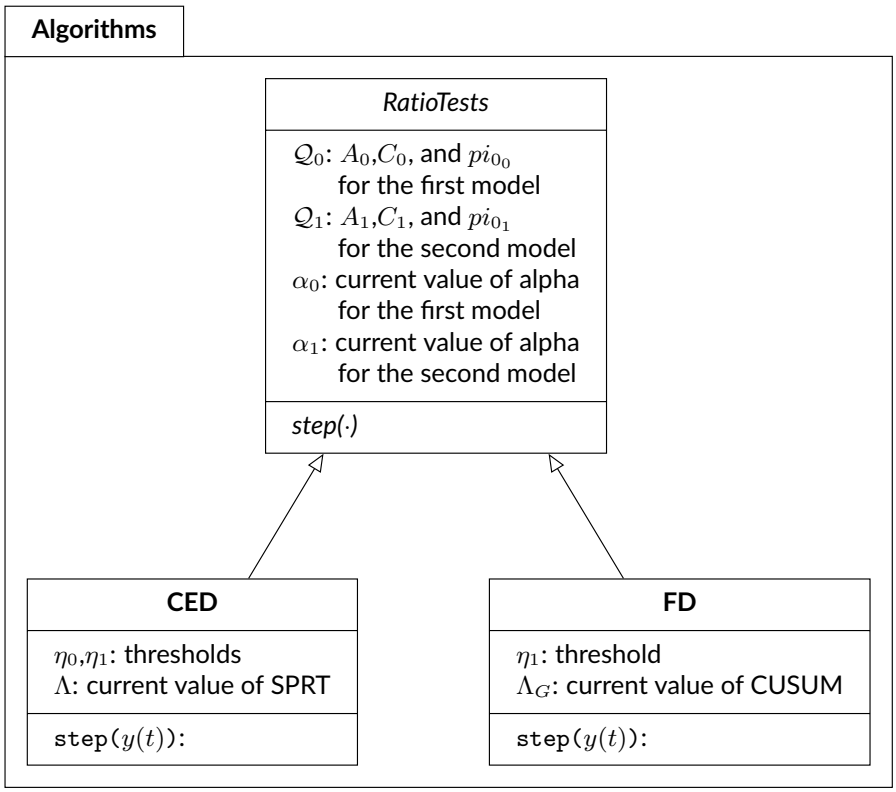


Figure 5.2: UML Diagram of classes CED, FD, and their parent `RatioTests`.

Furthermore, in that Figure, another class is also shown: the `RatioTests` class. Since the two algorithms are quite similar, especially as regards the input parameters for the hypotheses, a good idea is to recycle some code. Therefore, an abstract class was developed along with the two sub-classes. The super-class contains several parameters that will be useful for implementing the algorithms: the two first fields are about the parameters of the chain in the two possible hypotheses. Indeed, as stated in Chapter 4, for both of the algorithms it is necessary to define a chain for the no error chain along with one for the error case. The second parameter field contains the α values that must be stored in order to run the algorithms. A constructor is also developed to initialise these fields in an automatic-way: indeed, it is sufficient to provide only the A, C, and π_0 matrices for a chain, and it builds the two chains. The α values are initialised according to π_0. It is worth noticing that an abstract method is present in this class: `step`. This method requires an observation as input and returns a structure that contains the response of the algorithm, i.e. if a choice is made, the choice along with the achieved performances is returned, whereas, if no choice can be made, a continue-flag is returned. Since it is an abstract method, it must be implemented in the sub-classes, i.e. in `CED` and `FD`. The first sub-class implements the Commissioning Error Detection algorithm (Algorithm 5): three new parameters are required for this algorithm. The first is Λ, that is used to save the current value of the summation, whereas the other

two are the desired thresholds. These two values are initialised by the user using the constructor whereas the other is set to zero. The constructor also calls the super-class constructor to initialise the other parameters. `FD` class is designed very similar to this except for the different number of thresholds, which is one in this case. In both of classes, the `step` function implements respectively the central steps of the two algorithms in Chapter 4, i.e. the part of the algorithm that is inside the while-loop. For the first one, this method returns:

0 if no choice can be made

1 if the null hypothesis is true

-1 if the null hypothesis is false

whilst, in the second class, the possible `step` function outputs are:

0 if no issues are detected

T_c if an issue is detected, where T_c is the expected time in which change was made.

With these ingredients, a simulation workbench can be finally built.

5.1.3 The Simulation Implementation

To perform tests and analysis for these algorithms, it is necessary to develop a simulation workbench. Similar to the real ones that facilitate to create or modify objects because they make the necessary tools easily available, a simulation workbench is a collection of classes, scripts and functions that allows to perform tests in an all-in-one environment. For this opus, the workbench is composed by the previous illustrated classes and functions, and three scripts that allow to carry on the tests. The first script is called `RunSimulation1` and it performs the following operations:

1. defining parameters for the `CED` class and initialising them
2. running the simulation of faults and performing the `CED` algorithm
3. calculating the accuracy and the performances of the used algorithms
4. plotting the results

The running of the simulation is not trivial, and it is summarised in Figure 5.3.

The first operation to be done is to insert the parameters of the chains that will be used in the `CED` class, i.e. the two thresholds. The second step is to insert the parameters of the two workstations, i.e. A, C, and π_0. These values will be given to the two class constructors to build the respectively objects. Indeed, the `CED` class constructor automatically builds the two chains using only that parameters. The third operation is to select the flag of the script in order to select whether or not a fault has to be simulated. After initialising some variables, like the time counter, the constructors are called to create two `Workstation` objects and a `CED` object.

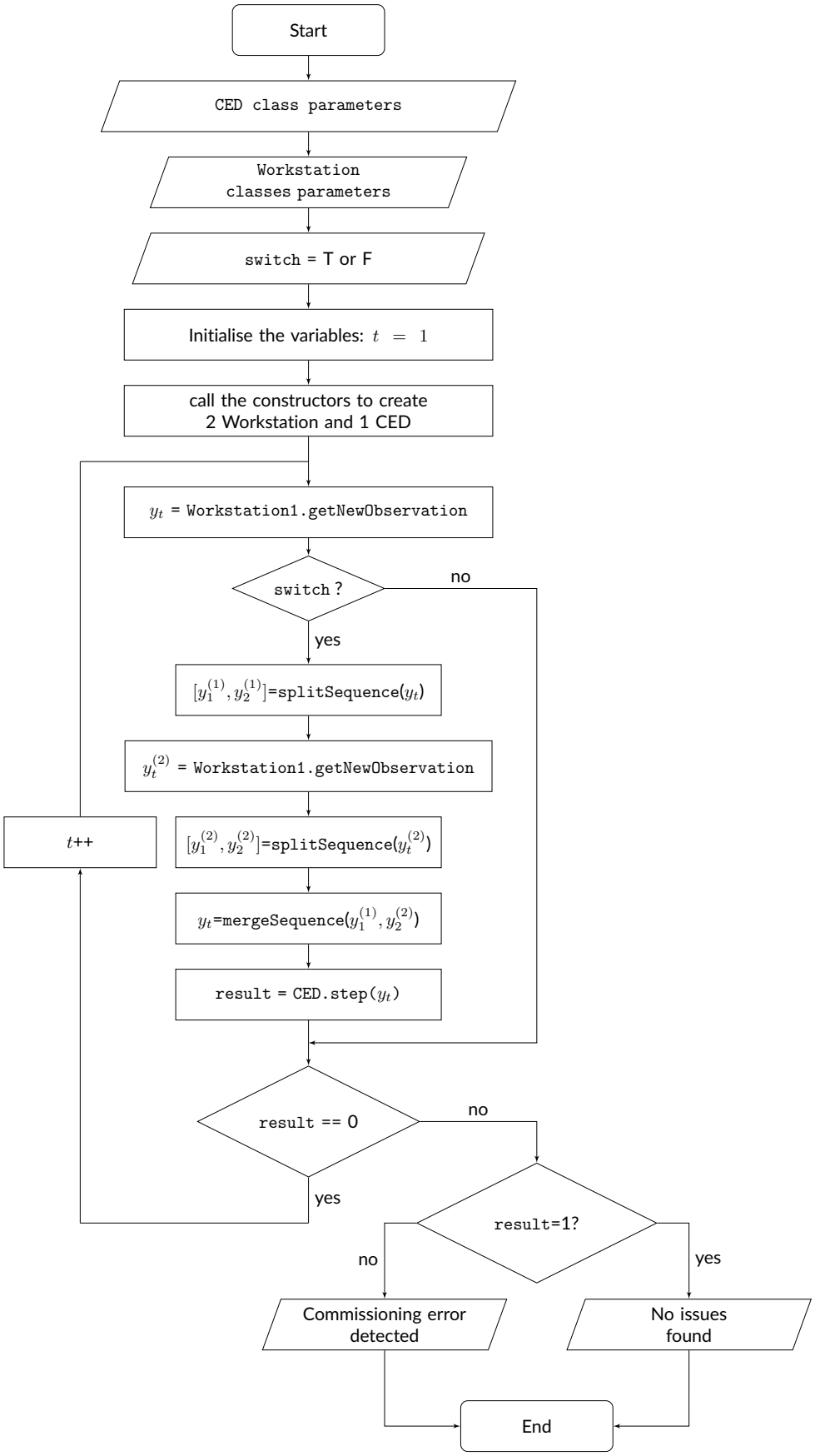


Figure 5.3: Flow chart of `RunSimulation1`.

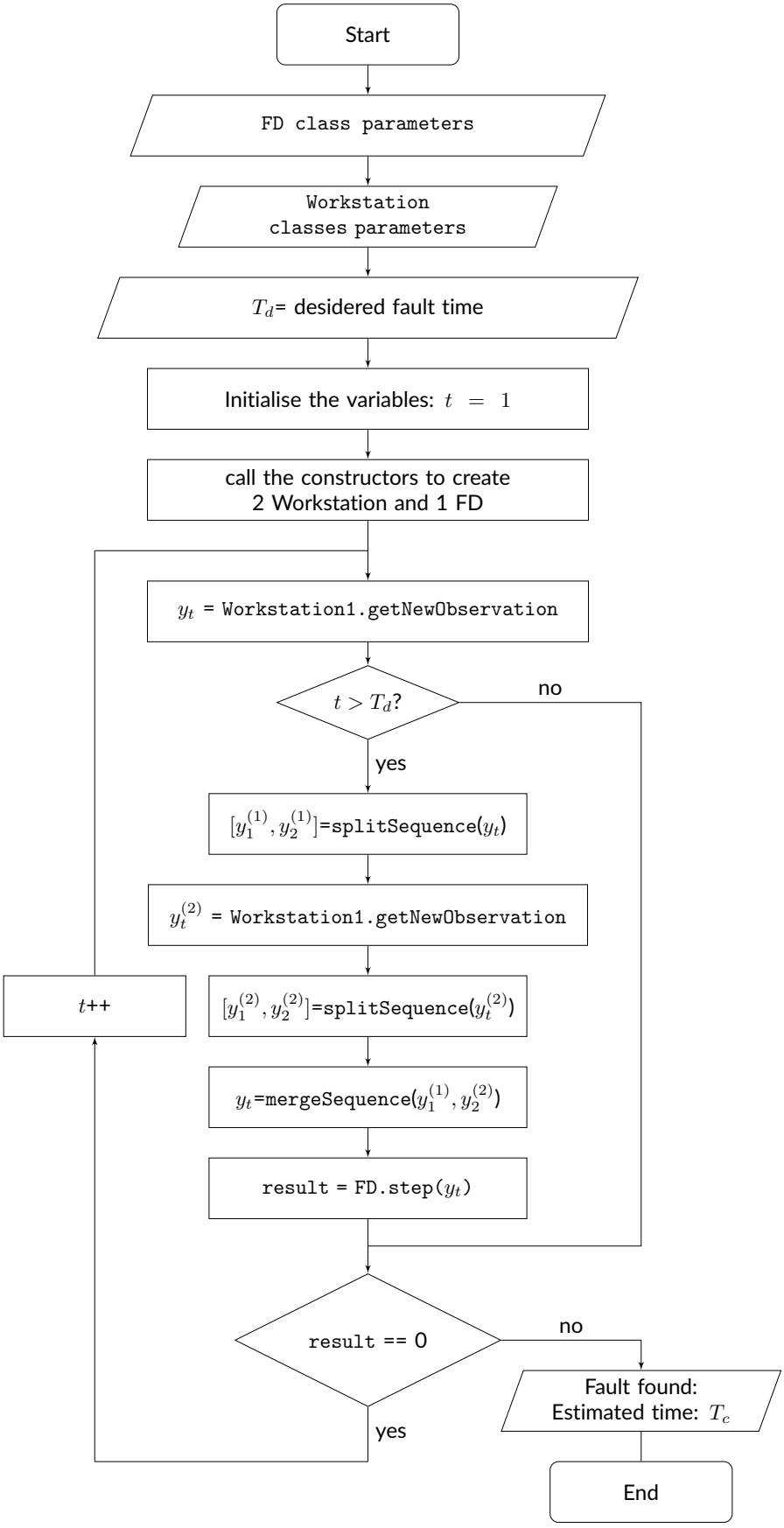


Figure 5.4: Flow chart of `RunSimulation2`.

Now the while-loop starts: the first step is to simulate a new observation using the first `Workstation` object. If a switch is desired to be simulated, the observation just obtained is split using `splitSequence`, a new observation is simulated using the other `Workstation` object and split: after that, the value of sensor one of the first observation is merged together with the value of second sensor of the second observation using `mergeSequence`. If no switch is desired, these steps are ignored. The further step is to call the `step` function of `CED` class: as stated before, giving an observation (directly or after a split-merge) as input, it returns whether hypothesis is true or if more data is necessary. If a choice is not made, the while-loop restarts after increasing the time counter. Otherwise, an output is returned according to the function result and the script stops. The other script, called `RunSimulation2`, is quite similar to `RunSimulation1` and it is used to simulate a failure in a workstation, i.e. a smart lamp has been moved after the commissioning was completed and tested correctly. In Figure 5.3, this new script is shown.

The main differences between the two scripts are concentrated in the different initialisation and in the different ending. Instead, the central part is the same for the two scripts. The initialisation of the `FD` class requires only one threshold whereas the workstations parameters are used at the same manner of the other script: indeed, since `FD` class inherits the properties from the super-class that is shared with the `CED` class, the two constructors work similar. The further step, that it is different with respect to the other script, is to decide the desired fault time, i.e. the time in which the switch is made. Until the end, this script behaves like the other. The end is different this time: the returned values is, this time, the estimated switch time T_c.

Thanks to these scripts, it is possible to carry on the simulations in an easy way.

5.2 Simulations introduction

The best way to test the algorithms presented in this paper is through simulations. Indeed, since workstation models are based on probability, it is difficult to analyse the algorithm performances using only deterministic methods. Furthermore, to make the results comparable, it is also necessary to introduce some limitations about the model parameters: the possible values of the parameters are infinite and therefore it would be impossible to simulate them all. For these reasons, it is better to state some assumptions about the carried simulation.

5.2.1 Assumptions

The main parameter limitations are introduced to favourite the thresholds comparison. Indeed, even if a formula was stated to calculate them, it is useful to check them on field. The first is the following:

Simulation Assumption 1. A and C matrices of a workstation depend respectively only on γ and δ, i.e.:

$$A = \begin{bmatrix} \gamma & 1-\gamma \\ 1-\gamma & \gamma \end{bmatrix} \qquad C = \begin{bmatrix} \delta & 1-\delta \\ 1-\delta & \delta \end{bmatrix}$$

This assumption was done to reduce the complexity of the matrices that could lead to have incomparable results. However, this assumption is not strange since it states that probabilities of state change are the same, no matter the previous state, and that the workstation smart lamp sensors have all the same performances. Another assumption about parameters is:

Simulation Assumption 2. γ and δ are the same for all workstations in an office.

This assumption was made to reduce the number of possible combinations in order to allow a threshold comparison and a shorter execution time of the overall simulation. In view of these assumptions, it is possible to state the used parameter for simulations. For δ and γ, the used values are:

$$\delta, \gamma \in \{0.65, 0.70, \ldots, 0.95\}$$

This allows to exclude uninteresting values, such as those close to 0.5 in which the chain is highly noisy, or unreal values, such as the value 1 which indicates a stationary chain in the case of γ or noise free sensors in the case of δ, which violates the assumptions. Furthermore, six different pairs of thresholds (η_0, η_1) were used in the first test whereas in the second the η_0 thresholds were dropped: to define them, six probabilities, i.e.

$$p_m = p_f \in \{2 \times 10^{-1}, 10^{-1}, 5 \times 10^{-2}, 10^{-2}, 10^{-3}, 10^{-4}\}$$

were stated and then the formulae 4.3 were applied to them. These probabilities have been chosen to simulate real values that can be used daily. In MATLAB, the threshold values can be calculated in only two rows using:

```
n0 = log (p_m./(1-p_f));
n1 = log ((1-p_m)./p_f);
```

These values are reported in Table 5.1.

$p_m = p_f$	20%	10%	5%	1%	0.1%	0.01%
η_0	−1.3863	−2.1972	−2.9444	−4.5951	−6.9068	−9.2102
η_1	1.3863	2.1972	2.9444	4.5951	6.9068	9.2102

Table 5.1: Thresholds used during simulations.

The last thing to highlight is the initial condition used for the chain set-up: for all tests, $\pi_0 = [0.5\, 0.5]$ was used.

5.2.2 Performed Tests

It is also important to describe the tests that will be performed and analysed: it easy to see that different analysis must be done for the Commissioning Error Detection and the Failure Detection. The tests carried out to check the commissioning error detection algorithm are as follows:

1. percentage value of miss detection (CED-MD)
2. percentage value of false alarm (CED-FA)
3. percentage value of overall errors (CED-OE)
4. time to detect (CED-T)

These tests are quite common for analysing this type of algorithms. The first consists of simulating several correct commissioned workstations, running the Algorithm 5 for each one, and counting how many errors were made, i.e. how many times the algorithm states that there was an error during commissioning. For this test, two hundred and fifty workstations were simulated. The second one is quite similar: two hundred and fifty faulty workstations were simulated, ran the Algorithm 5 for each one, and counted how many errors were made, i.e. how many times the algorithm states that a workstation received a good commissioning. The third test merges the two previously stated tests to show how many errors the algorithm commits: for this test, two hundred and fifty faulty workstations and two hundred and fifty hundred good ones were simulated. The last test consists of recording how many samples are necessary for the algorithm to make a choice: for this test too, five hundred workstations, half in good health, half faulty, were simulated. To reduce the amount of computation time, all these tests are carried concurrently.

For the Failure Detection analysis, the performed tests are:

1. mean time between two false alarms (FD-FA)
2. time to detect (FD-TA)
3. expected switch-time error (FD-TE)

The first test is about to simulate a healthy workstation and count the number of false alarms reported by the Algorithm 6 within a time window. Every time a false alarm is detected, the summation was reset to zero. In this work, the window was chosen of ten thousand samples and the test was repeated a hundred times for each model and threshold. Obviously, to find the average time between two false alarms, simply multiply the length of used window by the number of performed runs and divide the whole by the number of false alarms found. The other two tests, instead, consist of simulating one thousand workstations that will have an issue between the twentieth sample and the fiftieth sample, and performing Algorithm 6. For consistent data, false alarms were ignored. Similar to above, the time to detect test consists of finding how many samples are required for the algorithm to detect the fault after it happened whereas, the third, analysed how much error is made when the switch-time estimation is performed. To reduce the amount of computation time, these two tests were carried concurrently.

Some of the tests performed on Algorithm 6 are different from those performed on the other algorithm, since the different behaviour makes some of the previous ones useless. Indeed, miss detection errors cannot be made as the algorithm runs until it returns that the workstation has become faulty. Not even counting the number of false alarms makes sense, since if the workstation is never modified,

the algorithm will run for an infinite amount of time. For this, it was decided to monitor only the time between consecutive false alarms.

It is worth remembering that all tests for Commissioning Error Detection and Failure Detection are performed using six different thresholds as well as seven different values for γ and δ. The utilised thresholds are the ones reported in Table 5.1, and they are stated in tables and graphs by the respective values of p_f and p_m. The results of these tests are illustrated in the following section and can be found in the Appendix.

5.3 Simulation Tests Results & Plots - Commissioning Error Detection

In this section, the results of the above described tests about commissioning error detection (CED) are shown. The main tools utilised to illustrate them are plots and tables. Two different approaches are used to show the results for each performed test: the first one is to group the results by thresholds, no matter the γ or δ values used, to make a glance comparison (Subsection 5.3.1). The other idea is to group the results for the threshold and show only the ones that have the same γ value, i.e. simulations about same test that share the same A matrix are reported in the same table or plot (Subsection 5.3.2), to encourage a more in particular comparison: however, sometimes different approaches can be followed, like aggregating the results by δ value.

5.3.1 Commissioning Error Detection - overview

In this subsection, the grouped results of the commissioning error detection tests are reported. Remember that four tests were performed using six thresholds and forty-nine different chains. The full results can be found in Appendix A.

Commissioning Error Detection - Miss Detection

In Appendix Table A.1 the percentage values of the miss detection test are reported. Remember that this test counts how many times the algorithm states the commissioning was done correctly even if there is a commissioning error. Forty-nine different parameters were simulated using for each one two hundred and fifty chains: therefore, 12250 Montecarlo runs were simulated and thus the sensibility is around 0.01%. In Table 5.2 and Figure 5.5 these results are summarised, normalised, and grouped by the used threshold.

p_m	20%	10%	5%	1%	0.1%	0.01%
% mean	22.24	10.66	5.27	1.14	0.09	0.02

Table 5.2: Commissioning error detection - miss detection test: percentage value of situations in which the algorithm gives a wrong result. 12250 Montecarlo runs simulated.

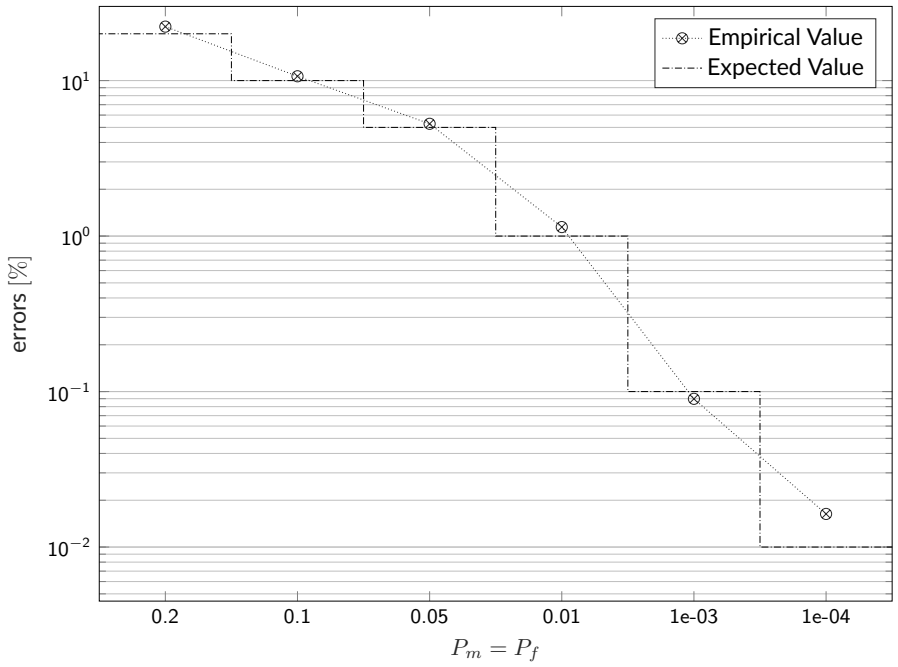


Figure 5.5: Commissioning error detection - miss detection test: percentage value of situations in which the algorithm gives a wrong result. 12250 Montecarlo runs simulated.

As it can be seen, most results meet expected values, i.e. the number of miss detection is within the assigned probability value after taking into account test sensitivity, or they are very close to it. Only using the first threshold, as it is clearly illustrated in the graph, the committed errors are appreciably higher than the expected ones: a possible fault can be found in not having used enough Montecarlo runs for each model for the test, and, therefore, the variance of the test is still significant. However, this means that the Wald's formulae for thresholds work quite well, they are reliable, and they can be used in daily situations. To analyse better this idea, grouped by γ and δ results are shown in the Section 5.3.2.

Commissioning Error Detection - False Alarm

In Appendix Table A.2 the percentage values of the false alarm test are reported. Remember that this test counts how many times the algorithm states the commissioning was done incorrectly even if there is not a commissioning error. Also, for this test, forty-nine different parameters were simulated using for each one two hundred and fifty chains: therefore, 12250 Montecarlo runs were simulated and thus the sensibility is around 0.01%. These results are summarised, normalised, and grouped by the used threshold in Table 5.3 and Figure 5.5.

As it can be seen, all the results are appreciably lower than the expected values: using the larger threshold, no error were committed. The main difference with the previous test is that, in this test, the thresholds are quite conservative, and, therefore, they can be set lower than the values that Wald suggests. Indeed, results are about 35 % lower than expected values. Furthermore, it seems that the number

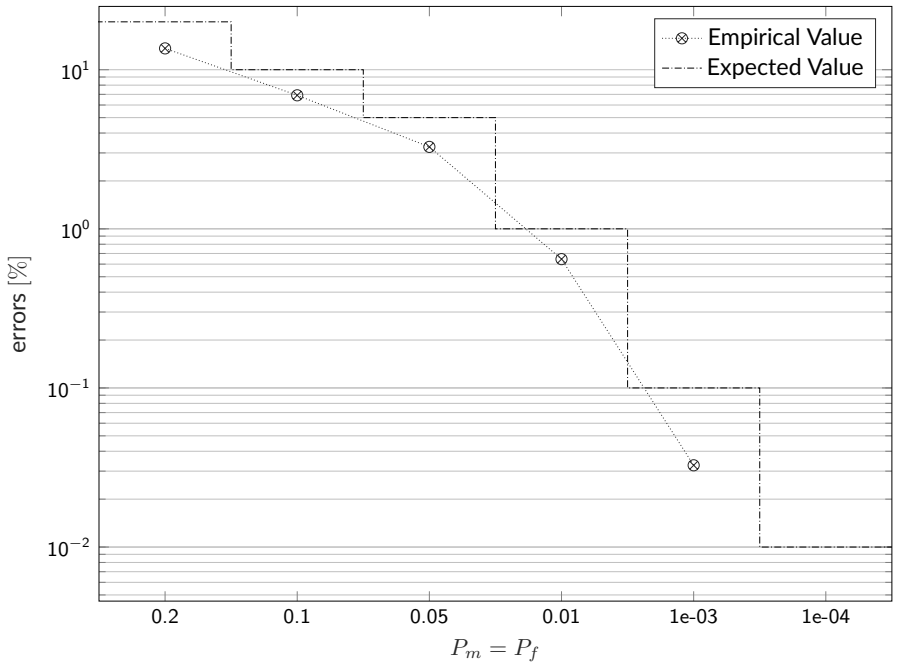


Figure 5.6: Commissioning error detection - false alarm test: percentage value of situations in which the algorithm gives a wrong result. 12250 Montecarlo runs simulated.

p_f	20%	10%	5%	1%	0.1%	0.01%
% mean	13.62	6.91	3.27	0.64	0.03	0.00

Table 5.3: Commissioning error detection - false alarm test: percentage value of situations in which the algorithm gives a wrong result. 12250 Montecarlo runs simulated.

of Montecarlo runs are sufficient since test variances are negligible. Therefore, also in this case, it can be said that Wald's formulae for thresholds work well: indeed, using the stated probabilities, the thresholds are sufficient to take into account test variances. This concept is stated better in the Section 5.3.2, in which the results are illustrated grouped by γ and δ.

Commissioning Error Detection - overall number of errors

In Appendix Table A.3 the percentage values of the number of test that have failed the miss detection or the false alarm tests are reported. Remember that this test merges the previous twos and then five hundred situations were simulated for each chain pair of parameters. Therefore, forty-nine different parameters were simulated using for each one five hundred chains: 24500 Montecarlo runs were simulated and thus the sensibility is close to 0.004%. Like in the previous cases, the results are normalised, summarised, and grouped by the used threshold in Table 5.4 and Figure 5.7.

The obtained values are lower than the ones in Table 5.2 but higher than the

$p_m = p_f$	20%	10%	5%	1%	0.1%	0.01%
% mean	17.927	8.784	4.273	0.894	0.061	0.008

Table 5.4: Commissioning error detection - miss detection and false alarm test: percentage value of situations in which the algorithm gives a wrong result. 24500 Montecarlo runs simulated.

ones in Table 5.3. This can be explained by the test procedure: for this test, the previous two tests are merged, and, obviously, a sort of mean is done. However, all of them are lower than the expected values utilised to build the thresholds. However, all the results are within the expected area, and they are about 10 % lower than the respectively thresholds.

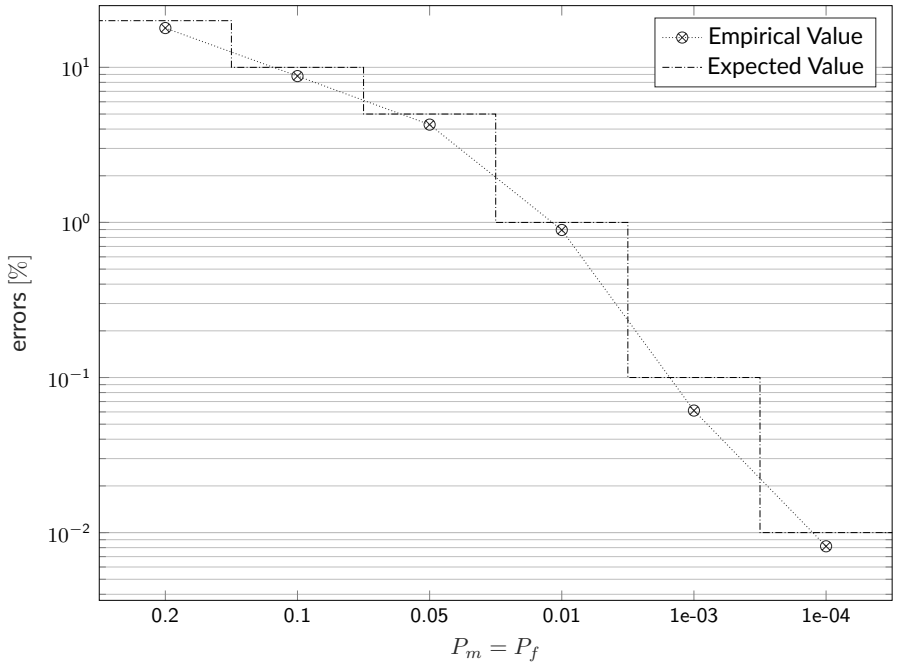


Figure 5.7: Commissioning error detection - miss detection and false alarm test: percentage value of situations in which the algorithm gives a wrong result. 24500 Montecarlo runs simulated.

Therefore, this test confirms that the Wald formulae suit well for this algorithm, and that produce good results. This can be also seen, more in detail, in the Section 5.3.2 in which the results are illustrated grouped by γ and δ.

Commissioning Error Detection - samples to make a choice

The last committed test regards the algorithm run time, i.e. how many samples are required by the algorithm to make a choice. In Table 5.5 the means and the standard deviations of the required samples for each threshold are reported along with their ranges.

$p_m = p_f$	20%	10%	5%	1%	0.1%	0.01%
mean [samples]	34.96	68.30	99.95	163.77	247.60	327.76
dev.std [samples]	45.19	94.17	140.69	235.07	361.47	479.59
range [samples]	181.34	393.08	580.74	948.73	1468.17	1938.70

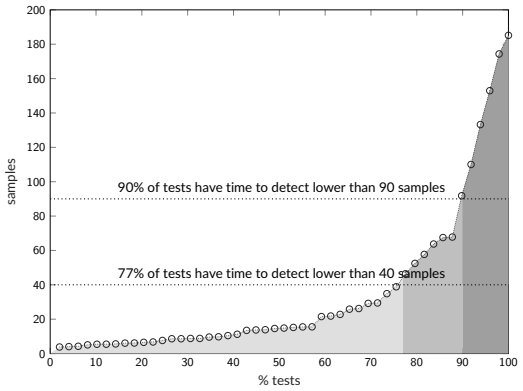
Table 5.5: Commissioning error detection - computation time test: means and standard deviations of samples required by the algorithm to make a choice using different thresholds. 24500 Montecarlo runs simulated.

For this test it was decided to show the model standard deviations: this index is useful for controlling whether using only mean is reliable to make considerations or not. Indeed, a high variance means that different outcomes behave in very different manners whereas a low variance describes the data as compact and close to mean. To calculate this index, the standard deviation of each column of Table 5.5 was performed using the `std` command in MATLAB. In the previous test, the variance values were very low: the maximum test standard deviation found was 5% but, usually, the values were around 1% or less. Instead, in this case, the variance is quite high and therefore it seems no conclusions can be drawn. In these situations, a good idea is to calculate the range. i.e. the difference between the maximum and the minimum values in a data-set. As it can be seen in Table 5.5, the range value is very high and, therefore, it means that the data is very sparse. In other words, there are cases in which the algorithm is very fast and cases where it is very slow, no matter the threshold chosen. However, a partial conclusion can be drawn: the lower the threshold, the faster the algorithm, even if it seems there are some models that are faster than other also using a larger threshold. To provide a better explanation, a different method of aggregate reporting must be used.

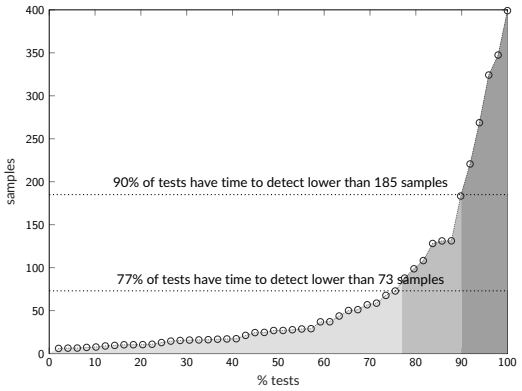
In Figure 5.8, percentile plots for all used thresholds are reported since this type of plots allows to classify with a glance the distribution of the data. Before defining what a percentile plot [27, Sect. 2.2] is, it is better to state what a percentile and a quantile are. The i percentile, $i \in \{1, 2, \ldots, 99\}$, is a number on the scale of the data that divides the data into two groups, so that the i % of the observations fall below and the other ones fall above. The $0.i$ quantile works in the same way, only using fractions instead of percentage values. Therefore, a percentile graph is the plot of the percentile "function": the faster way to make it is to sort y-data values by ascending order and plot them over a normalised x-axis.

As it can be seen there, for all the different utilised thresholds, the plots are pretty similar if, for a moment, the adopted scale factor is ignored: in all cases, about 77% of the tests are completed within a reasonable time. Instead, to have that 90% of the tests are completed, it is necessary to wait a time twice as long as the one necessary for the first 77%. Furthermore, for everyone to finish, it is necessary to wait up to five times the time that for the 77% of tests is more than enough. These plots clearly show that this behaviour is influenced by the model utilised, i.e. the γ and δ values, since the threshold does not change within a single plot. To better highlight this, a new display method should be used.

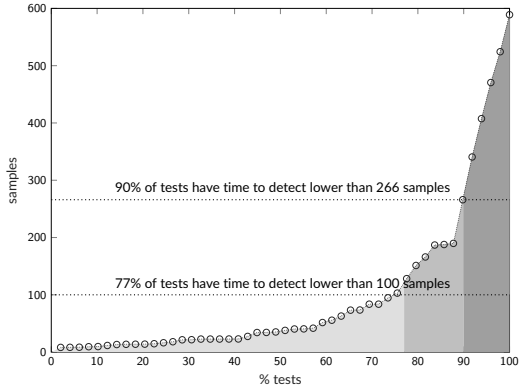
In Figure 5.9, the same information of 5.8 is displayed using **Box Plots**. A



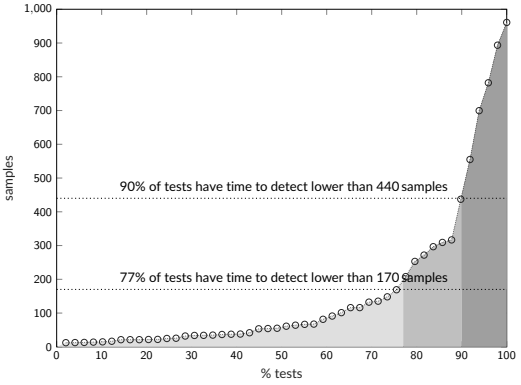
(a) $p_f = p_m = 2 \times 10^{-1}$



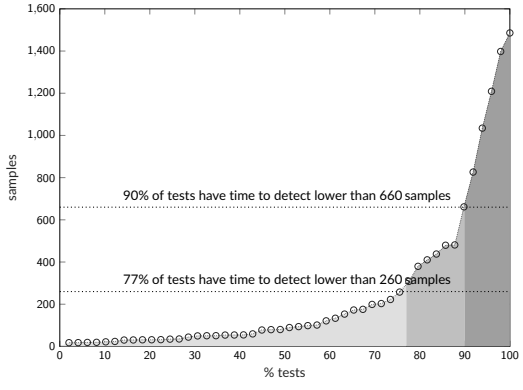
(b) $p_f = p_m = 10^{-1}$



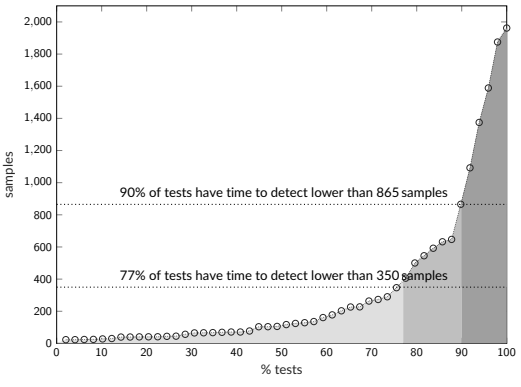
(c) $p_f = p_m = 5 \times 10^{-2}$



(d) $p_f = p_m = 10^{-2}$

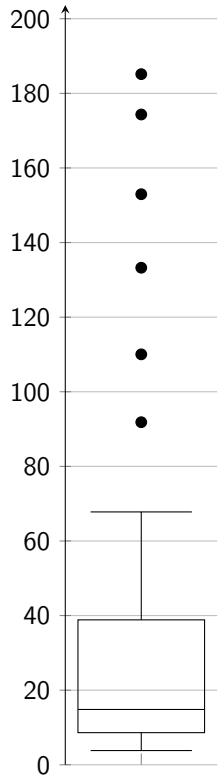


(e) $p_f = p_m = 10^{-3}$

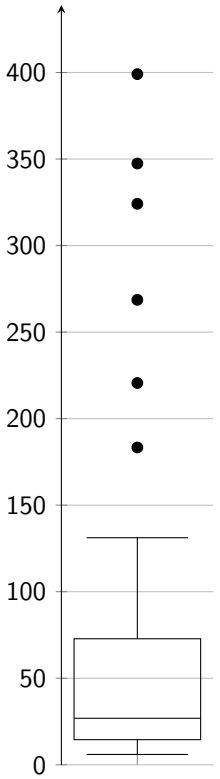


(f) $p_f = p_m = 10^{-4}$

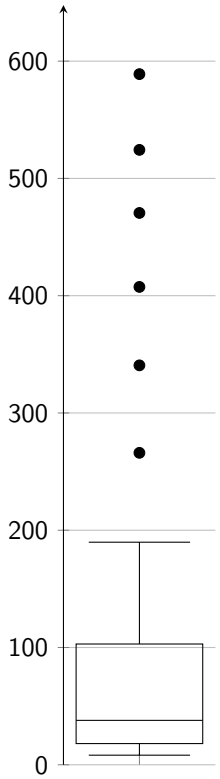
Figure 5.8: Commissioning error detection - computation time test: percentile plots of samples required by the algorithm to make a choice using different thresholds. 24500 Montecarlo runs simulated.



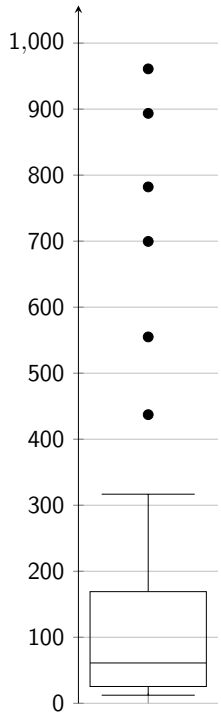
(a) $p_f = p_m = 2 \times 10^{-1}$



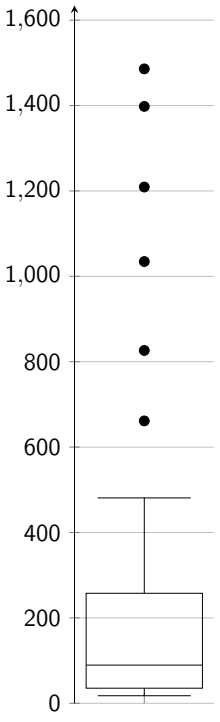
(b) $p_f = p_m = 10^{-1}$



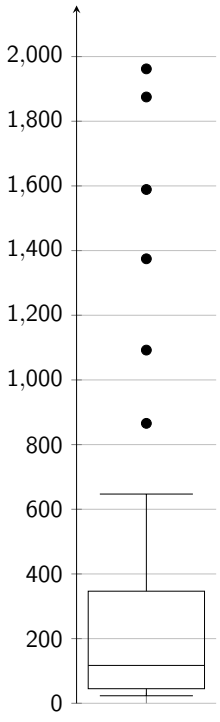
(c) $p_f = p_m = 5 \times 10^{-2}$



(d) $p_f = p_m = 10^{-2}$



(e) $p_f = p_m = 10^{-3}$



(f) $p_f = p_m = 10^{-4}$

Figure 5.9: Commissioning error detection - computation time test: box plots of samples required by the algorithm to make a choice using different thresholds. 24500 Montecarlo runs simulated.

box plot [28], also known as **box-and-whisker plot**, is a powerful tool to rapidly summarise and interpret data coming from experiments: it allows, with only a glance, to find the main properties of a data-set. The diagram [27, Sect. 2.5] was developed by J. Tukey, and it is composed by a rectangle (the box), a horizontal line inside the rectangle, two horizontal lines (the whiskers) outside of the rectangle linked to it by a vertical line, and some points above them. The lower horizontal side of the rectangle represents the first quartile, i.e. the middle value between the smallest number and the median of the data-set, whereas the higher horizontal side represents the third quartile, i.e. the middle value between the maximum number of the data-set and the median. The horizontal line inside the rectangle is the data-set median, also known as second quartile or 50-th percentile. The whiskers are a little more difficult to be drawn: the upper one has an y-value equal to $+1.5\times$IQR whereas the lower one is equal to $-1.5\times$IQR, if IQR is defined as the interquartile range, i.e. the difference between the third and the first quartile. The lengths of the vertical lines that connect the whiskers to the rectangle show how stretched the tails of the data distribution are. The points drawn beyond the whiskers are the data points that are not included in the range $[-1.5, +1.5]\times$ IQR, and, for this, they are often defined outliers even if their mere presence in this area is not enough to classify them in this way.

γ	δ
0.65	0.65
0.70	0.65
0.75	0.65
0.80	0.65
0.85	0.65
0.90	0.65

Table 5.6: Commissioning error detection - computation time test: model parameters that produce outliers in Figure 5.9. The first is furthest from the upper whisker, and so on.

The box plots in Figure 5.9 seem very similar to each other at first glance: if the scale factor is ignored, the diagram is almost similar for each graph: in each box plot the lower whisker is very close to the rectangle whereas the other whisker is very far apart. Furthermore, the horizontal line that represents the median is not in the centre of the rectangle, but it is closer to the lower side of the rectangle than to the higher one. This means that the distribution of data is not symmetric, and that the right side of the distribution is very long. This leads to think that there are some very fast models along with very slow ones, even if, since the median is closer to the lower whisker than the upper one, there are faster models than slow ones. Another interesting thing that these plots show is that there are six points above the last whisker in each plot. This means there are six models that provide values higher than one and half times of the interquartile range, and they are probably outliers. In other words, there are six models whose behaviour is very different from

the other ones, and, therefore, it is not possible to perform any overall analysis about algorithm computation time. Since the models differ from each other only by γ and δ, some values of these two parameters must be blamed for this. Indeed, the model parameters that produce outlier data is only six and are the same for all the used thresholds: they are reported in Table 5.6.

From these data, it seems that a low value of δ in the model leads to have a slow algorithm: this seems a likely idea since a low value of δ means that the sensors are very noisy and thus the measurements are not very reliable. This will be investigated deeper in the following section where more in particular γ and δ analyses will be performed, and thus an identification of these odd models can be performed.

Commissioning Error Detection - overall aggregated analysis

After a first summary analysis of the results, it appears that in the first three tests of commissioning error detection, which consist of counting the number of miss detection, false alarms and their sum, Wald's formula works quite well. Only in the first test, some of the results do not respect the expected values, even for just a little. Moreover, especially in the case of the false alarms test, the values obtained are appreciably lower than those imposed: this means that it is possible to reduce the thresholds a little more compared to Wald's formula, which therefore seems a little conservative for daily use.

Different conclusions were drawn from the fourth test, the one regarding the algorithm time performances. As it can be seen, using the aggregated data, it was not possible to obtain anything else that, on average, a higher threshold algorithm takes longer to finish than a lower threshold one. For this reason, it was necessary to introduce two different methods of aggregate reporting, the percentile plot, and the box plot. Using these tools, it was found that there are six models that have a very different behaviour with regards to the other ones. All of these ones have $\delta = 0.65$, the lowest value available for the sensor sensibility. To find the mathematical motivation for this behaviour, more in particular analysis must be performed.

5.3.2 Commissioning Error Detection - γ and δ analysis

To better appreciate the results that can be found in Appendix A, it is useful to plot the test values using a fixed variable, γ or δ, and compare the behaviour of the others as it changes. Remember that, models are defined as:

$$A = \begin{bmatrix} \gamma & 1-\gamma \\ 1-\gamma & \gamma \end{bmatrix} \qquad C = \begin{bmatrix} \delta & 1-\delta \\ 1-\delta & \delta \end{bmatrix}$$

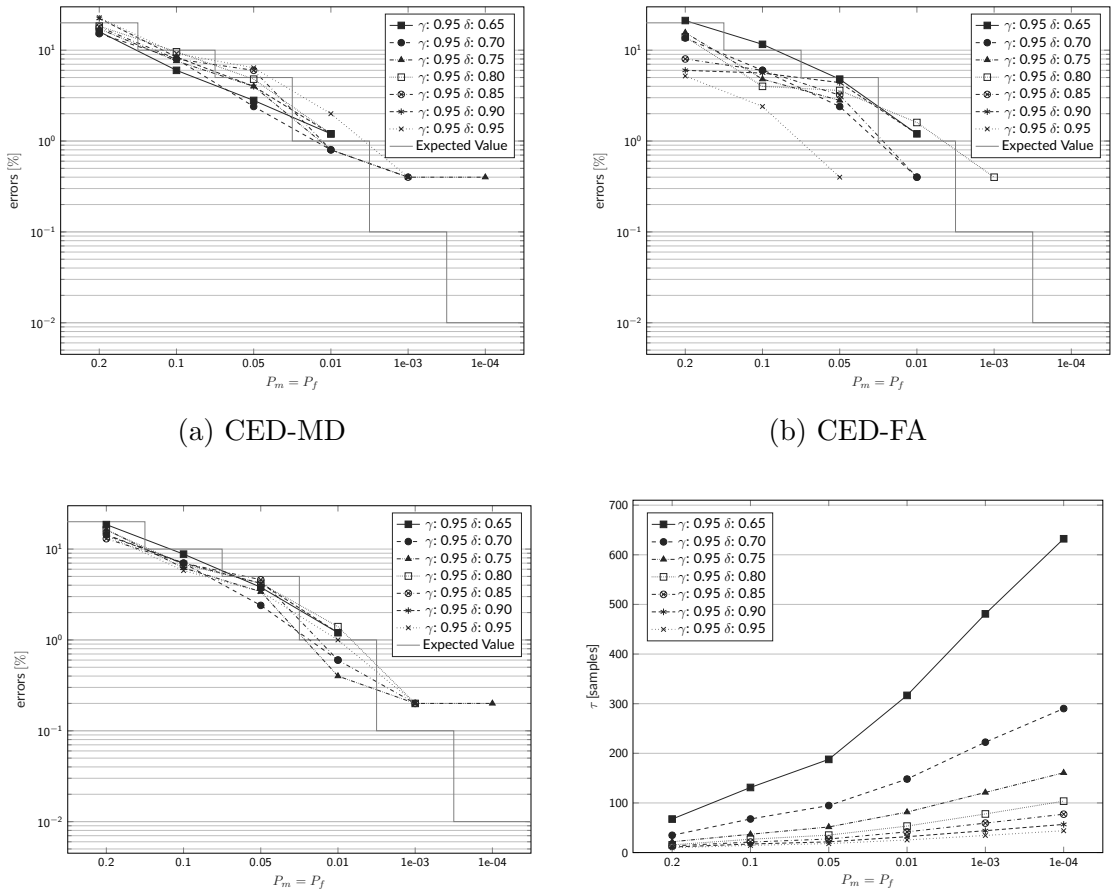
A lot of comparisons can be done using the simulated data: in this paper, only the most relevant ones are reported. Indeed, forty-nine different models were simulated five hundred times, but, therefore, only some of them can be shown.

It is worth highlighting the different sensibility of the tests performed to better understand the results. The first two tests are carried on over two hundred and fifty simulations: this means that the sensibility of the results is $\frac{1}{250} = 0.004 =$

$4 \times 10^{-3} = 4 \times 10^{-1}\%$. In other words, the points reported in plots for tests (CED-FA) and (CED-MD) that are close to $4 \times 10^{-1}\%$ are not sufficient precise and are within a hypothetical confidence interval of zero. At the same manner, the points in (CED-OE) test that are close to $2 \times 10^{-1}\%$ should be treated as zero values.

Commissioning Error Detection - $\gamma = 0.95$ analysis

The first comparison was done using a fixed high γ, i.e. $\gamma = 0.95$, and changing the δ value. This means that models with rare state changes were utilised. The tests plots can be found in Figure 5.10.



(a) CED-MD

(b) CED-FA

(c) CED-OE

(d) CED-T

Figure 5.10: Commissioning error detection tests with fixed $\gamma = 0.95$. In a) and b) 250 Montecarlo runs were performed whereas in c) and d) 500 Montecarlo runs were executed. Lower is better.

In this figure, it can be seen that the results of the first test do not show a very clear dependence on δ values but it seems that low values of δ provide more stable results than high ones. Indeed, high δ values do not ensure a lower number of errors but, sometimes, they do not respect the *a priori* miss detection probability. However, to be sure of this and not be deceived by the variance of the tests, a similar analysis with other values of α must be performed.

In the false alarm test, no correlation between algorithm results and the δ values used for the models can be clearly seen: indeed, an order is very difficult to be found. In this case, all cases except two respect the probability values defined: the model with issues is the one with the lowest δ. It seems that a high value of δ gives better results in false alarm test, but is more susceptible to performing miss detections, whereas a low value of δ does the opposite. Furthermore, it seems that false alarms are rarer than miss detections for workstations with high δ values: using the largest threshold, no false alarms were reporting during the simulations whereas with the second largest only one false alarm occurred. This is not always desired since, in the case of commissioning, it is better to have a false alarm with respect to an undetected error.

Obviously, since the third test (CED-OE) merges the two previous ones, nothing more can be said since all the results are within the expected values after considering the test sensitivity, but an order cannot be found. However, a better situation with respect to the miss detection analysis can be found here, since all parameter choices follow the expected threshold values. This means that the Wald formulae are perfect for $\gamma = 0.95$.

The fourth test, the one that represents how much time the algorithm takes to make a choice, is much more clarifying and lends itself to many considerations: as it can be clearly seen, as the required probability increases, the number of required samples to make a decision increases. This was expected since the higher the probability, the higher the thresholds must be used. Another expected fact is that, using the same required probability, with the increase of the model δ the number of samples used increases. This can be explained if it is considered the fact that the value of δ represents the probability of correctly measuring the status, i.e. the accuracy of the sensors. As a result, noisier measurements lead to a lower probability of a correct measurement, and consequently the sum, Λ, grows more slowly and takes longer to reach the set thresholds.

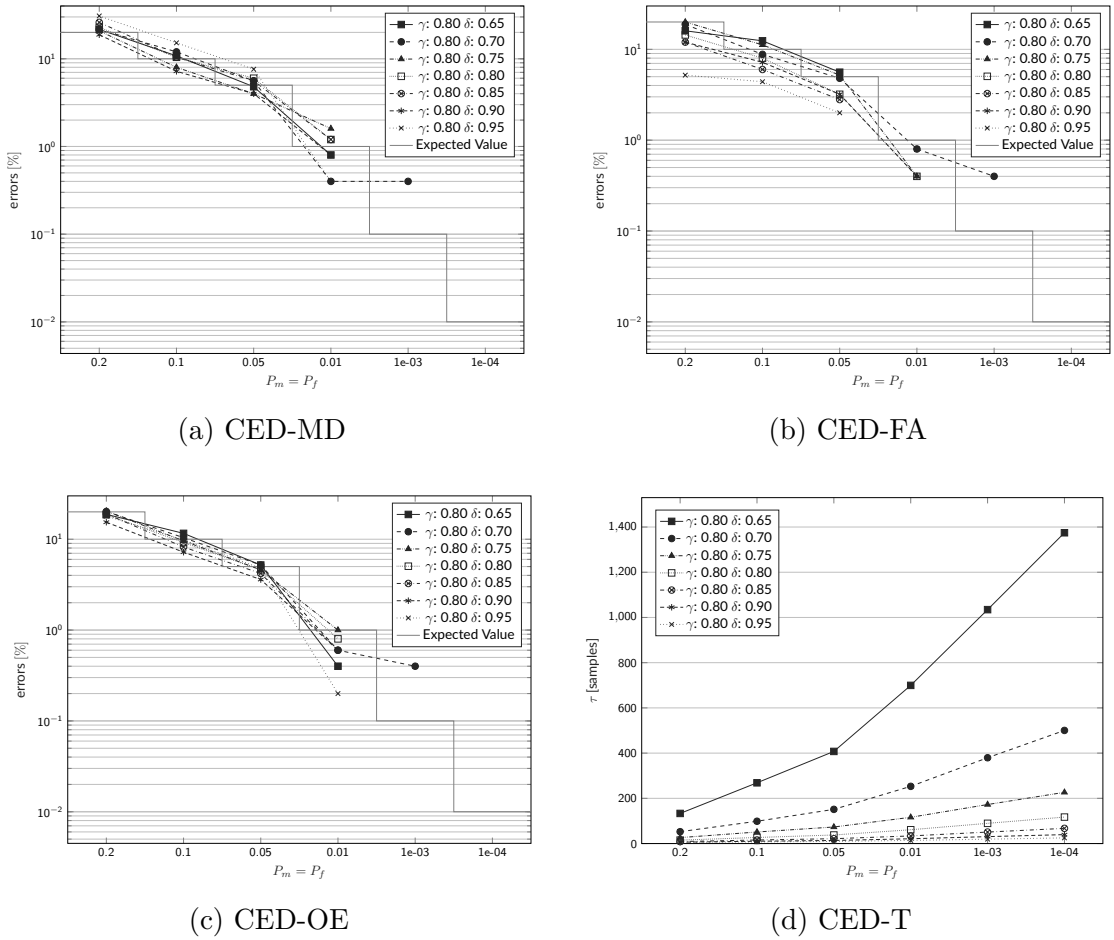
Commissioning Error Detection - $\gamma = 0.80$ analysis

The second comparison was done using a fixed medium γ, i.e. $\gamma = 0.80$, and changing the δ value. This means that models with not so rare state changes were simulated. The tests plots can be found in Figure 5.11.

In the first test, the miss detection one, the majority of the simulated models are within the expected values, but there are more than ten situations that not respect the defined probability. Interestingly, high and low values of γ produce unexpected results, and the models that leads the algorithm to do not respect the limits are often different by using one threshold or another. However, it seems that high values of γ lead the algorithm to be more susceptible to make errors.

A better situation can be found in the false alarm test: indeed, in this test, a smaller number of test fail to reach the desired error thresholds and the ones are the models that have a low value of γ. In this case too, it is difficult to find a consistent order for the models.

The third test provides a little more information than the two previous tests: it seems that, using higher values of δ, the algorithm performs better than using low ones, even if an order is not so clearly defined. However, also in this test, there are



(a) CED-MD

(b) CED-FA

(c) CED-OE

(d) CED-T

Figure 5.11: Commissioning error detection tests with fixed $\gamma = 0.80$. In a) and b) 250 Montecarlo runs were performed whereas in c) and d) 500 Montecarlo runs were executed. Lower is better.

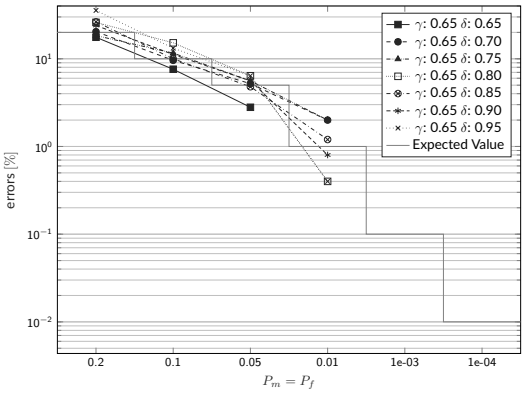
some models that do not allow the algorithm to reach the expected error thresholds: it seems that, using models with a medium value of γ, the performances of Wald formulae start to deteriorate.

The fourth test results, instead, are very clear and reaffirm that the lower the model δ value, the faster the algorithm: furthermore, like in the previous test, low δ value models with low thresholds take more time to finish than high δ value models with high thresholds.

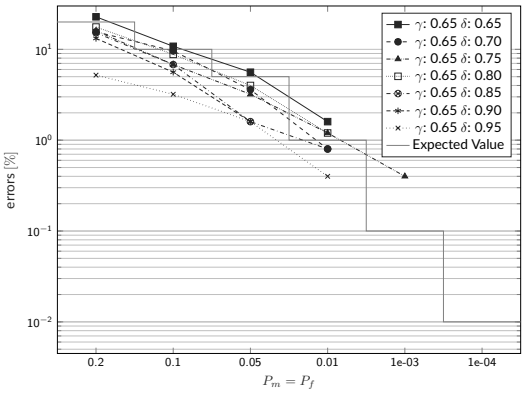
Commissioning Error Detection - $\gamma = 0.65$ analysis

The last comparison was done using models with a fixed low γ, i.e. $\gamma = 0.65$, and changing the δ value. This means that simulated models change frequently the states. The tests plots can be found in Figure 5.12.

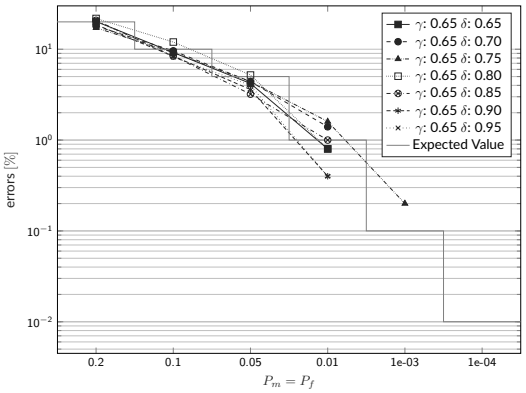
In the first test, the miss detection test, a huge number of the simulated models are outside the expected values ranges: there more than fifteen situations that do not respect the defined probability. Interestingly, high and low values of γ produce unexpected results, but it seems that high values of γ are more susceptible to make



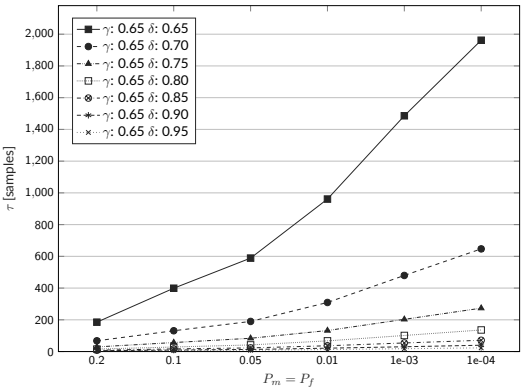
(a) CED-MD



(b) CED-FA



(c) CED-OE



(d) CED-T

Figure 5.12: Commissioning error detection tests with fixed $\gamma = 0.65$. In a) and b) 250 Montecarlo runs were performed whereas in c) and d) 500 Montecarlo runs were executed. Lower is better.

errors. After the third comparison, it is very difficult to blame the test variances for this, and likely a cause might be found.

Like in the previous comparison, better situation can be found in the false alarm test: in this test, a smaller number of test fail to reach the desired error thresholds and the one is the model that has the lowest value of γ. Instead, in this case, a consistent order for the models can be glimpsed. It seems that the higher the δ value, the lower the errors made.

The third test does not provide more information than the two previous tests, but it quite confirms that an overall order for the models is difficult to be found. Moreover, it shows that the worst model is the one with $\gamma = 0.80$: the cause can be found in that it excels neither in the first test nor in the false alarm test, while all others have a test that is more favourable to them.

The last test confirms the ones shown in the previous comparison, but this time the algorithm takes a very long time to be executed with respect to the other cases. It seems that also the value of γ should be blamed for the computation time, as it can be seen later on.

Commissioning Error Detection - overall δ analysis

In Table 5.7, the previous results are summarised to allow a fast availability.

It can be clearly seen that the pattern is the quite same for each different workstation γ value: indeed, except for the first case, the miss detection tests are often failed whereas the false alarm ones are usually passed. Furthermore, the third test is usually passed too, whereas the last test states that the higher values of δ are used, the faster is the algorithm.

γ	CED-MD	CED-FA	CED-OE	CED-T
0.95	RT, LiB	RT, HiB	RT	HiB
0.80	NRT, LiB	RT, HiB	RT	HiB
0.65	NRT, LiB	RT, HiB	RT	HiB

Table 5.7: Recap of the results in the different tests: δ analysis. LiB = lower δ values is better, HiB = higher δ values is better, RT = respected thresholds, NRT = not respected thresholds, NA = no choice made.

Motivations Several causes can be found to explain the behaviour of the test results. But the common one can be found in the way the algorithm was built. It is worth reminding that the main step of Algorithm 5 is:

$$\Lambda(t+1) = \Lambda(t) + \log \frac{\alpha^{\theta_1}_{t|t-1} C_{\theta_1}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha^{\theta_1}_{t|t-1}(l)} - \log \frac{\alpha^{\theta_0}_{t|t-1} C_{\theta_0}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha^{\theta_0}_{t|t-1}(l)}$$

where, the update rule for α values are:

$$\alpha^{\theta_1}_{t+1|t} = \left(\alpha^{\theta_1}_{t|t-1} \odot C_{\theta_1}(:, y_t)^\intercal\right) A_{\theta_1}$$
$$\alpha^{\theta_0}_{t+1|t} = \left(\alpha^{\theta_0}_{t|t-1} \odot C_{\theta_0}(:, y_t)^\intercal\right) A_{\theta_0}$$

and the C matrices are built following (3.52) and (3.56), i.e.:

$$C^{\theta_0} = \begin{bmatrix} \delta^2 & \delta(1-\delta) & \delta(1-\delta) & (1-\delta)^2 \\ (1-\delta)^2 & \delta(1-\delta) & \delta(1-\delta) & \delta^2 \end{bmatrix}$$

$$C^{\theta_1} = \begin{bmatrix} \delta^2 & \delta(1-\delta) & \delta(1-\delta) & (1-\delta)^2 \\ \delta(1-\delta) & (1-\delta)^2 & \delta^2 & \delta(1-\delta) \\ \delta(1-\delta) & \delta^2 & (1-\delta)^2 & \delta(1-\delta) \\ (1-\delta)^2 & \delta(1-\delta) & \delta(1-\delta) & \delta^2 \end{bmatrix}$$

It is also worth noticing that, if δ is close to 0.5, all the elements matrices become equal and close to 0.25. On other hand, if δ is close to 1, the matrices become very

sparse and all the no-zero entries are close to 1. The last thing to remember is that δ represents the sensibility of the sensors utilised, i.e. how much the algorithm can trust the outcomes.

Putting the whole together, some conclusions can be drawn. The first is that, for the same γ used, a high value of δ leads to having less uniform C matrices and that favours, during the α update, the most probable values: therefore, the α vectors will have very similar elements if δ is close to 0.5 whereas very different if δ is close to one. Since the value of α is normalised in the algorithm step as shown above, a high δ leads to having the likely part of the α vector very high whilst using a low δ leads to having them low. Another thing to take care about is the Λ value behaviour since a difference between logarithms is also made in the algorithm step. To better understand, this step should be made more explicit:

$$\log \frac{\alpha^{\theta_1}_{t|t-1} C_{\theta_1}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha^{\theta_1}_{t|t-1}(l)} - \log \frac{\alpha^{\theta_0}_{t|t-1} C_{\theta_0}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha^{\theta_0}_{t|t-1}(l)} = \log \frac{\dfrac{\alpha^{\theta_1}_{t|t-1} C_{\theta_1}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha^{\theta_1}_{t|t-1}(l)}}{\dfrac{\alpha^{\theta_0}_{t|t-1} C_{\theta_0}(:, y(t))}{\sum\limits_{x_l \in \mathcal{S}} \alpha^{\theta_0}_{t|t-1}(l)}}$$

A further useful step is to make this formula explicit in the case of the two extreme values of δ. For δ close to 0.5, the formula can be written as:

$$l_l = \log \frac{K\,\pi_0}{K\,\pi_0} = \log 1 = 0, \qquad K \in \mathbb{R}$$

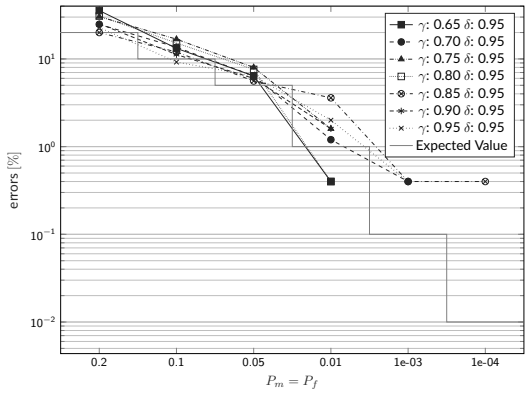
since the C matrices are very similar to a constant value times an all-ones matrix, the α values are constant from the beginning, and this leads to have logarithm of a division of same things, that is equal to zero. Even from an intuitive point of view, it seems to work: indeed, if δ is close to 0.5, it means that the measurements are not reliable and consequently the algorithm does not take them into account. Therefore, since the algorithm cannot distinguish between the two cases, since the C matrix is the same for all measurements, it gives the same weight to both the cases: this leads the ratio to be equal to one. Obviously, the logarithm of one is zero and this leads to having a constant Λ equal to 0 since the sum never grows: thus, the algorithm never stops. For δ value close to 1, the C matrices are very sparse, and have a lot of zeros in the cells that represent the miss detection or false alarm of the sensor. It is worth noticing that the matrix C^{θ_1} becomes similar to an identity matrix whereas the other becomes a null matrix except for the first and the last entries that are equal to one. This means that the elements of the vector α can only be zero or one, and this only depends on the value of the state of the chain, since the sensor is infallible. This leads to a ratio of zero or infinite depending on the situation already at the first step. By doing the logarithm, this becomes a positive or negative infinity based on which of the two hypotheses is true. Then the algorithm ends in one step since $l_h = \pm\infty$.

Summarising, from what has been written above, the most important conclusion that can be drawn is that, as expected, a high value of δ allows the algorithm to take less time than a low one as Λ grows much faster in this case. Indeed, high

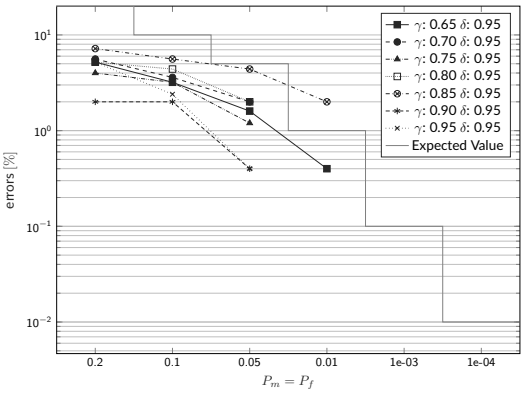
values of δ lead the algorithm to trust a lot the measurements, and thus to assign to them a high probability values. This causes to have an appreciable difference of logarithms that makes Λ to reach the threshold very fast. However, it is also interesting to check out the behaviour of the algorithm when different values of γ are utilised while using constant δ values. This is done in the following subsections.

Commissioning Error Detection - $\delta = 0.95$ analysis

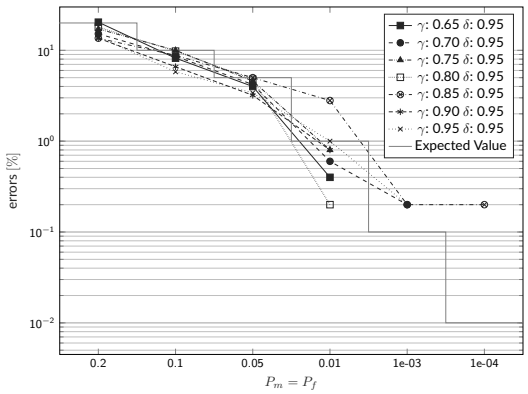
Similar to what was done for the test results with γ fixed, also for the tests using models with δ fixed the comparison will be performed using three values of δ. The first comparison is done using $\delta = 0.95$, i.e. the workstation has two quite perfect sensors since their sensibility is very high. In Figure 5.13 the plots of the tests that utilised $\delta = 0.95$ are shown.



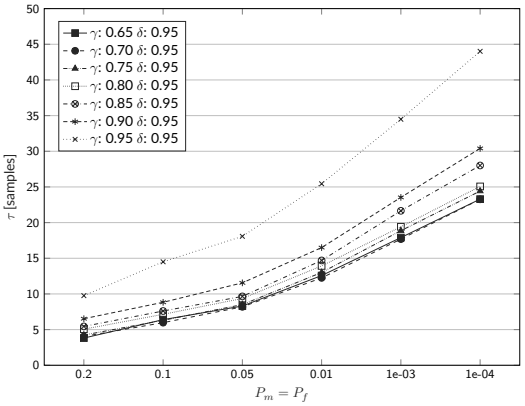
(a) CED-MD



(b) CED-FA



(c) CED-OE



(d) CED-T

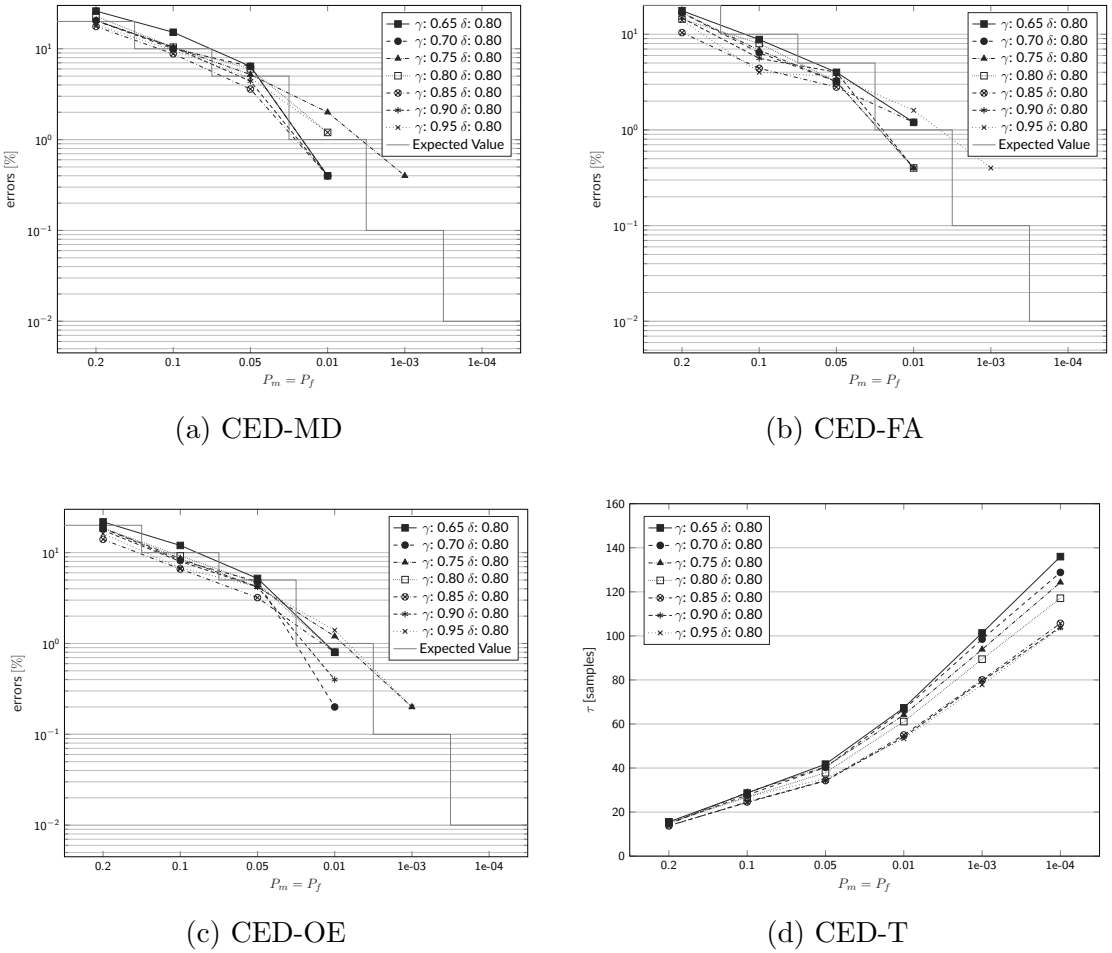
Figure 5.13: Commissioning error detection tests with fixed $\delta = 0.95$. In a) and b) 250 Montecarlo runs were performed whereas in c) and d) 500 Montecarlo runs were executed. Lower is better.

The miss detection test clearly shows that, using such a high value of δ, it is difficult to find an order that clearly illustrates the situation. Another information provided by the first graph is that very few cases meet the defined probabilities of miss detection. This is the worst situation so far. The situation changes considerably

if the second graph is looked at: indeed, in that plot, almost all cases perform much better than the assigned probabilities, especially in the cases of high error probability. Not even here a clear order between the models can be found but it seems that a higher γ will produce better results. The third plot merges the two previous tests: in this one, almost all tests are within the expected probabilities. In this situation too, a clear order between models cannot be found, but it seems that all the model behaviours are pretty close to each other. The last test, the one that measures how long the algorithm takes to finish, is the clearest one. Indeed, a clear order can be found: the higher the γ value, the slower the algorithm. This is an interesting result since, using a δ fixed values, the higher the δ value, the faster the algorithm. To be sure about this, it is better to analyse some more different values.

Commissioning Error Detection - $\delta = 0.80$ analysis

In Figure 5.14, the plots of the results of the tests performed with $\delta = 0.80$ are shown. A value of 0.80 for δ is a medium value and represents a sensor that is not perfect, but it is sufficient precise for daily use.



(a) CED-MD

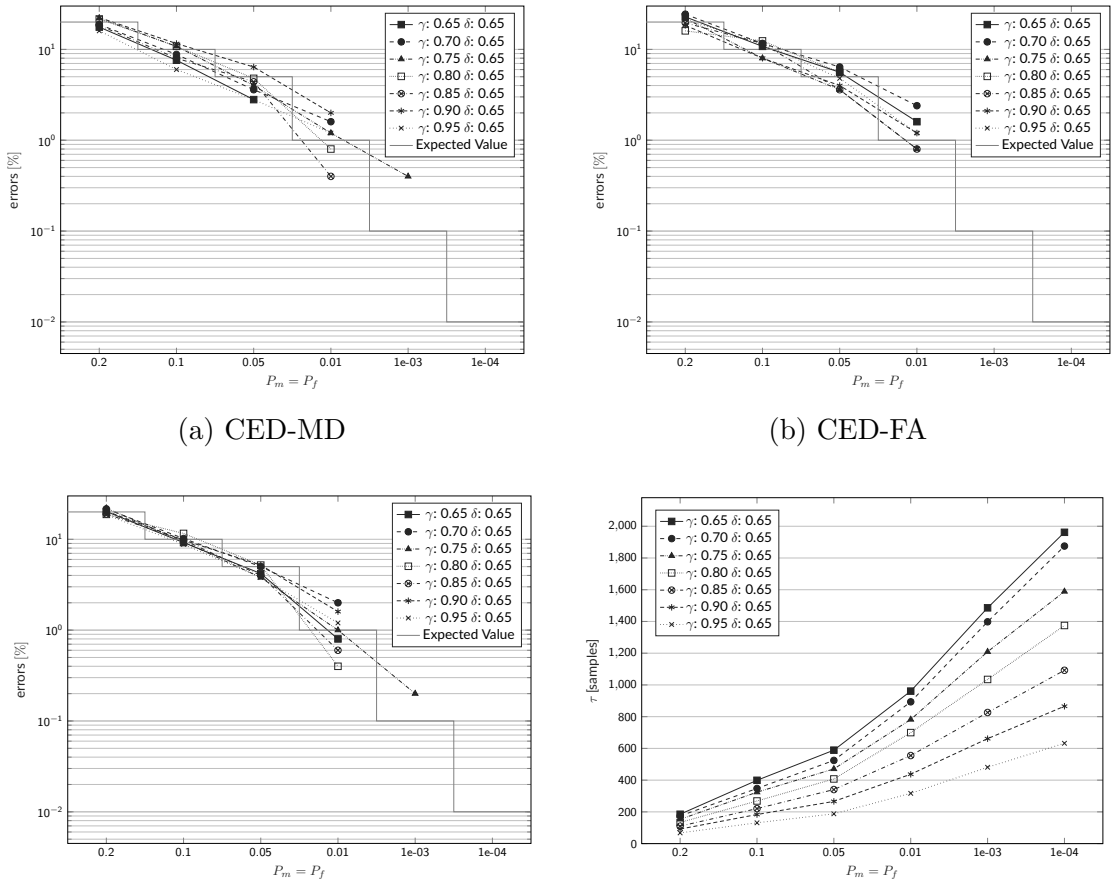
(b) CED-FA

(c) CED-OE

(d) CED-T

Figure 5.14: Commissioning error detection tests with fixed $\delta = 0.80$. In a) and b) 250 Montecarlo runs were performed whereas in c) and d) 500 Montecarlo runs were executed. Lower is better.

The situation using $\delta = 0.80$ is smoother than the previous one. In the first plot, the one regarding the miss detection test, it can be seen that the models respect more the thresholds than in the case of using $\delta = 0.95$. Vice-versa, in the second plot, it can be seen that the results are worse than the ones in 5.13b even if almost all models respect the pre-set thresholds. Furthermore, the third test is pretty similar to the one performed with $\delta = 0.95$ in which almost all models behave correctly. Therefore, it seems that using a medium δ value allows to have better results in miss detection test than using a high δ value. This, however, is paid with a worse performance in the test of false alarms. The third test, instead, does not seem to be affected of the δ value but only by γ values. Also, in this case the fourth test is the clearest but it gives very different results from the previous one: indeed, even in this case an order is clearly visible, but it is the opposite of the previous case. Here, the lower γ value, the slower the algorithm. However, unlike the tests carried out with the value of γ fixed, the distances between the models are quite small: so, the threshold choice affects much more the duration of the calculation than the γ value utilised for the model. This issue needs further investigation, which will be carried out later with new δ value tests.



(a) CED-MD

(b) CED-FA

(c) CED-OE

(d) CED-T

Figure 5.15: Commissioning error detection tests with fixed $\delta = 0.65$. In a) and b) 250 Montecarlo runs were performed whereas in c) and d) 500 Montecarlo runs were executed. Lower is better.

Commissioning Error Detection - $\delta = 0.65$ analysis

It is useful to check out what happens if a low value of δ is employed for the tests, i.e. if a very poor performance sensor is utilised. The results of these tests are reported in 5.15

Following the trend of the two previous cases, the first graph shows that the number of miss detection errors in the tests is slightly lower than before, whilst the second graph shows that the false alarms are slightly increasing using models with noisier sensors. With this choice of δ, the number of false alarms almost often exceeds the pre-set values, even if, using the two largest thresholds, no false alarms were reported. The third test shows that the average errors are usually within the thresholds, as expected. Similar to the previous choice of δ, also in this case the fourth test behaves as expected, i.e. models with high values of γ lead the algorithm to be faster than using small values. However, unlike the previous case, the distances between the different models have grown a lot: this means that if the value of δ is low, the value of γ is crucial to determine the computation time of the algorithm. After all, it is clear that a different way of illustrating this comparison is needed.

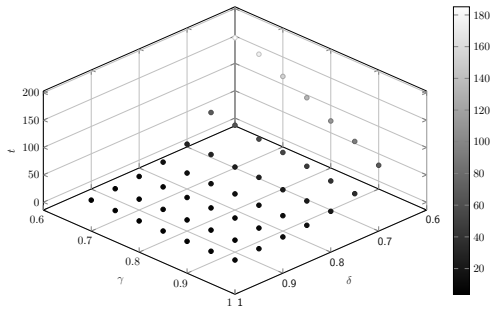
5.3.3 Computation time Analysis

The most interesting question that has remained unsolved by the previous tests is the one concerning the computation time of the algorithm. To try to solve this issue, some 3D plots are quite useful, as can be seen in Figure 5.16.

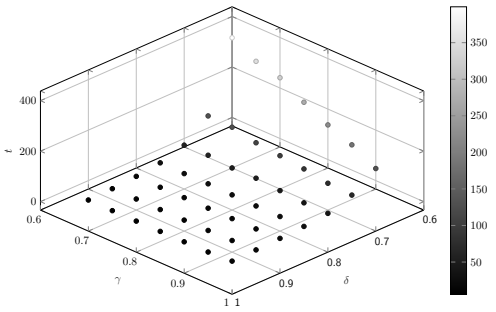
In this Figure, 3D plots are used to display how much time the algorithm requires to analyse the model and make a choice: in the x-axis the γ values are plotted whereas the y-axis reports the δ values. The z-axis, instead, represents the amount of time required: this value can be also read using the black intensity of the plot marks. As it is stated by the colour-bar next to each plot, the darker the mark, the faster the algorithm.

Looking to each plot, it seems that a pattern can be found: each plot is quite flat for δ from 0.7 to 1, but it starts to grow and becomes like a series of hairpins as the δ value decreases and the γ value approaches 0.65. So, it becomes clear that the main responsible for the duration of the algorithm is the value of δ and only secondarily the α value can be blamed. Another interesting thing, which is much more evident here than in the single value charts, is that the computation time trend compared to δ value that drops is almost exponential while, in the case of γ value decreasing, this is almost linear. This is more evident using the orthogonal views, like the ones shows in Figure 5.17. This means that, if it is possible to do, it is better to invest a little more money to have good sensors than into a faster machine.

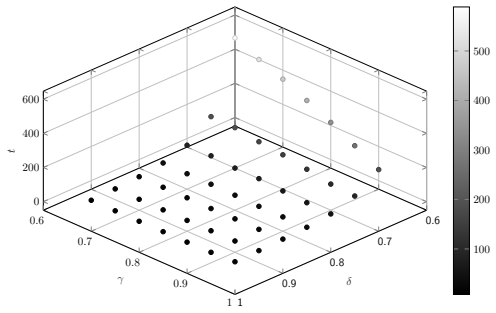
This intuition is also confirmed by the mathematics: it is worth remembering that the main steps of Algorithm 5 are the α update and the log-ratio. These can



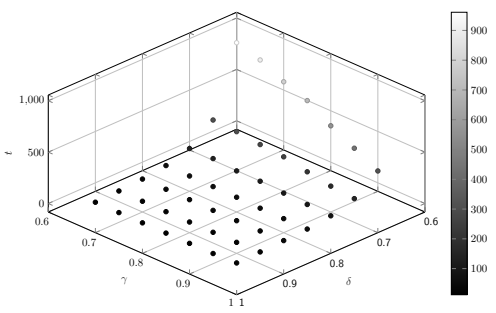
(a) $p_f = p_m = 2 \times 10^{-1}$



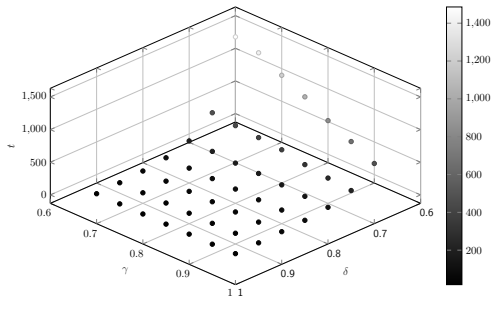
(b) $p_f = p_m = 10^{-1}$



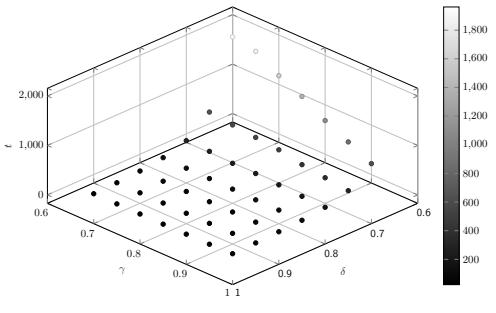
(c) $p_f = p_m = 5 \times 10^{-2}$



(d) $p_f = p_m = 10^{-2}$

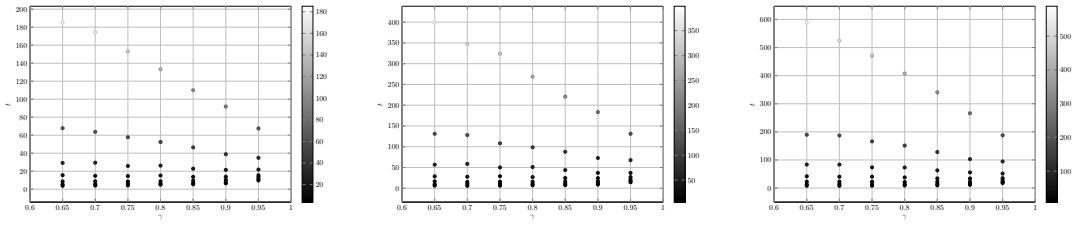


(e) $p_f = p_m = 10^{-3}$

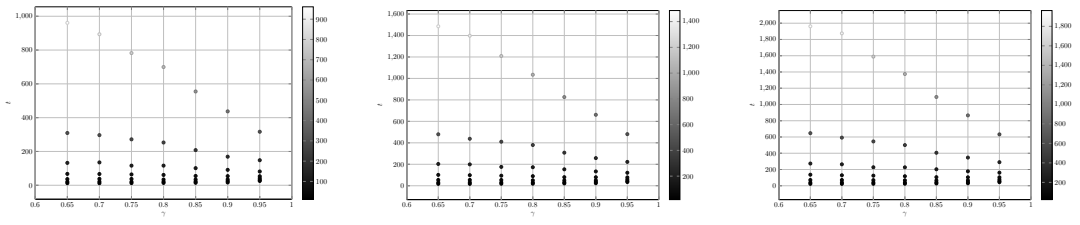


(f) $p_f = p_m = 10^{-4}$

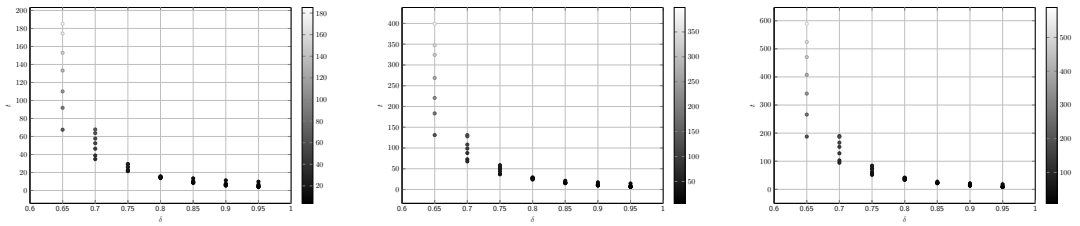
Figure 5.16: Commissioning error detection - computation time test: 3D plots of the number of samples required by the algorithm to make a choice using different thresholds. 24500 Montecarlo runs simulated.



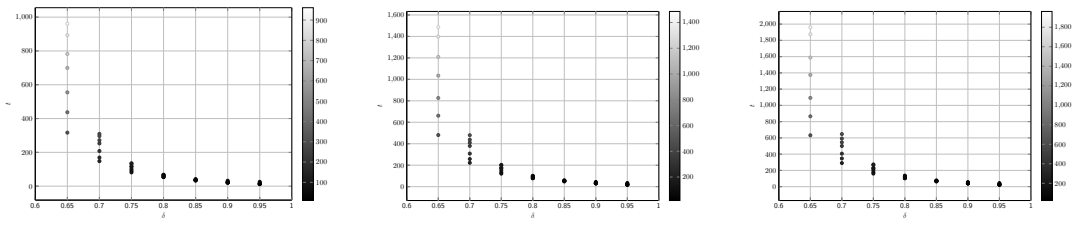
(a) $p_f = p_m = 2 \times 10^{-1}$ (b) $p_f = p_m = 10^{-1}$ (c) $p_f = p_m = 5 \times 10^{-2}$



(d) $p_f = p_m = 10^{-2}$ (e) $p_f = p_m = 10^{-3}$ (f) $p_f = p_m = 10^{-4}$



(g) $p_f = p_m = 2 \times 10^{-1}$ (h) $p_f = p_m = 10^{-1}$ (i) $p_f = p_m = 5 \times 10^{-2}$



(j) $p_f = p_m = 10^{-2}$ (k) $p_f = p_m = 10^{-3}$ (l) $p_f = p_m = 10^{-4}$

Figure 5.17: Commissioning error detection - computation time test: 2D views of 3D plots about samples required by the algorithm to make a choice using different thresholds. Views from a) to f) are about δ values whereas views from g) to l) are about γ values. 24500 Montecarlo runs simulated.

be easily summarised in:

$$\Lambda(t+1) = \Lambda(t) + \log \frac{\left(\left(\alpha^{\theta_1}_{t-1|t-2} \odot C_{\theta_1}(:, y_{t-1})^\intercal\right) A_{\theta_1}\right) C_{\theta_1}(:, y_t)}{\sum_{x_l \in \mathcal{S}} \left(\left(\alpha^{\theta_1}_{t-1|t-2} \odot C_{\theta_1}(:, y_t)^\intercal\right) A_{\theta_1}\right)(l)} - \log \frac{\left(\left(\alpha^{\theta_0}_{t-1|t-2} \odot C_{\theta_0}(:, y_{t-1})^\intercal\right) A_{\theta_0}\right) C_{\theta_0}(:, y_t)}{\sum_{x_l \in \mathcal{S}} \left(\left(\alpha^{\theta_0}_{t-1|t-2} \odot C_{\theta_0}(:, y_t)^\intercal\right) A_{\theta_0}\right)(l)}$$

From this mathematical step, which compacts the two operations performed at each iteration of the algorithm, it is very evident that, for calculating the input for the logarithm function, two multiplications for the matrix C are performed whereas only one A multiplication is required. This means that a change in value of δ has twice the weight of changing the value of γ by the same amount. However, since normalisation is performed, the γ value and the first multiplication using δ can be ignored. There remains a multiplication for δ that is not absorbed, and therefore remains at the numerator: if δ has a great value, the logarithm will be high too. Vice-versa, if δ has a value close to one half, as seen in the overall analysis, the logarithm value is low. Remembering that the logarithm is a function that spreads the values smaller than one, the expected behaviour is immediately found. About α, it would seem that the algorithm should be independent of the A matrix used. In fact, the problem lies not in the algorithm but in the data it receives. If the γ is very high, it means that state changes are very rare, and consequently in the α vector only one particular component grows while all the others decrease. Please note that the entry i-th of the carrier α is the probability of being in the state i and observing a particular sequence of observations. On the other hand, if the value of γ is close to 0.5, it means that state changes are frequent, and therefore the vector α has entries very similar among them. To clarify the idea, it is better to use a situation in which α has two components: obviously, the reasoning works also for larger dimensions. Exploiting the following property:

$$(a\gamma_1 + b(1-\gamma_1)) > (c\gamma_2 + d(1-\gamma_2))$$

for $a + b = c + d = 1$, $a > c$, $b < d$, and $\gamma_1 > \gamma_2 > 0.5$, it becomes clear that the contribution to the Λ adder is higher in the case of γ large than in the one with γ small, and for this reason it reaches the threshold in less time, allowing the algorithm to finish using fewer samples. As mentioned above, this has a marginal impact if the value of δ is low. Indeed, using a low δ, the C matrix no longer favours some values at the expense of others and so the components of the α vector all become equal to each other, causing the property defined above to lose substance. For this reason, a high value of δ should be always used, i.e. sufficient precise sensors shall be implemented.

In summary, it was found that algorithm has better performance analysing models with values close to 1 for γ and δ than models with values close to 0.5. In addition, it has been seen that the value of δ weighs much more than the γ value

for the algorithm duration. The last remark to make is about the large distance between points among same γ value if δ begins to be small. The fault is to be attributed to the logarithm, which spreads very small values, made even smaller by a low value of δ. As it will be seen in the following Sections, this behaviour can be also found in the Failure Detection Algorithm.

5.4 Simulation Tests Results & Plots - Failure Detection

In this Sections, the results of the tests about failure detection are shown. The three metrics used here are the mean time between two false alarms, the time taken to execute the algorithm, and the difference between the expected switch time and real one. For the first test, a chain that represents a healthy workstation was built, simulated for ten thousand samples, and reported the number of false alarms. This operation was performed one hundred times to provide stable results. For the latter two tests, the *modus operandi* was the following for each Montecarlo run performed: two chains were built, one to represent the working situation and one for the failure situation. Furthermore, a random number in range from 20 to 50 was simulated to indicate the switching time. With regard to the algorithm, it received the observations from the first chain or from the second depending on elapsed time. The switching time range was defined in order to have a chain sufficiently heated but not too long to simulate. Simulations that reported false alarm were dropped. A more in particular description of the tests can be found in Subsection 5.2.2. However, it is worth highlighting that in the first a higher value means a better model for the algorithm whereas the other two are lower is better tests. Similar to what was done in the case of commissioning error detection, the main tools utilised to illustrate them are plots and tables, and two different approaches are used to show the results: initially, an overview will be given, but later on, for the most interesting cases, a more in particular analysis will be carried on.

5.4.1 Failure Detection - overview

In this part, a test overview will be presented. As in the test of commissioning error detection, forty-nine different models were simulated: each one differs from the others by the value of γ or δ. For the first test, one hundred simulations with a length of ten thousand samples have been considered. Therefore, to find the mean time between two false alarms, the reckon to be performed is:

$$\tau_{FA} = \frac{10^6}{\#\text{FA}} \tag{5.1}$$

For the other two tests, for each model, one thousand Montecarlo runs were performed: thus, a total of 49000 simulations were executed. In this case too, in all tests, six different thresholds were used, and they have the same value of η_1 utilised in commissioning error detection test.

Failure Detection - False Alarms

In Appendix Table A.5 the mean time values between two false alarms, grouped by the model utilised, are reported. For this test, a false alarm is defined as the case in which the algorithm communicates to have detect a fault even if it has not happened yet. Every time a false alarm is reported, the value of Λ_G is reset to zero. The mean of the results obtained using (5.1) is plotted in Figure 5.18 and reported in Table 5.8. Furthermore, in the Table the median of the data can be also found.

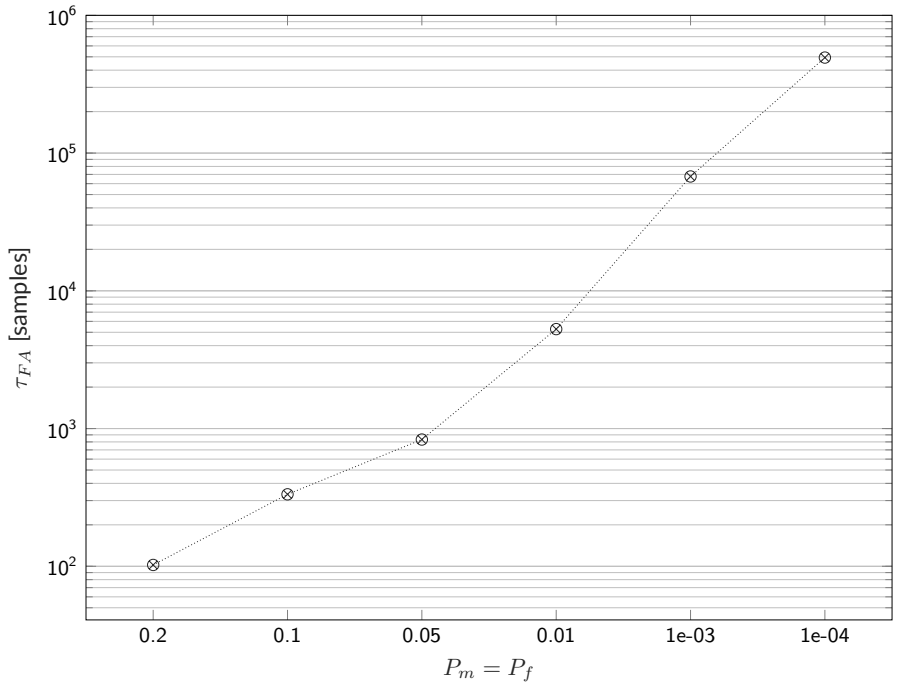


Figure 5.18: Failure Detection - mean time between two false alarms test. 100 Montecarlo runs. 10000 samples for each run. Higher is better.

As it can be seen, the situation is clear: the lower the threshold, the lower the time between two false alarms. This can be rewritten in a more familiar way as the lower the threshold, the higher the number of false alarms within a time window. This is not strange and retraces the same results obtained in commissioning error detection test.

$p_m = p_f$	20%	10%	5%	1%	0.1%	0.01%
mean [samples]	102.27	333.11	832.26	5.2×10^3	6.7×10^4	4.9×10^5
median [samples]	64.65	191.53	511.77	2.9×10^3	2.4×10^4	3.3×10^5

Table 5.8: Failure Detection - mean time between two false alarms test. 100 Montecarlo runs. 10000 samples for each run. Higher is better.

The results obtained, especially those with high threshold values, are very interesting and show that this method of failure detection can also be applied in the real world. Indeed, in the last two cases, the average between two false alarms

is one hundred thousand samples and this corresponds, using a sampling time of one minute, to more than two months. However, similar to what happened in the commissioning error detection test, a differentiated analysis for the different γ and δ values is required: a median quite far from the average indicates that there is some variability between different cases, and this is the fastest method to find the cause.

Failure Detection - Time to Detect & Expected Switch-time Error

In Appendix Table A.6 and A.7 the results of the time to detect and the expected switch-time error tests are reported. It was decided to perform a joint treatment for these tests since the twos are very closely related. In Table 5.9 and Figure 5.19, the results of these tests are reported.

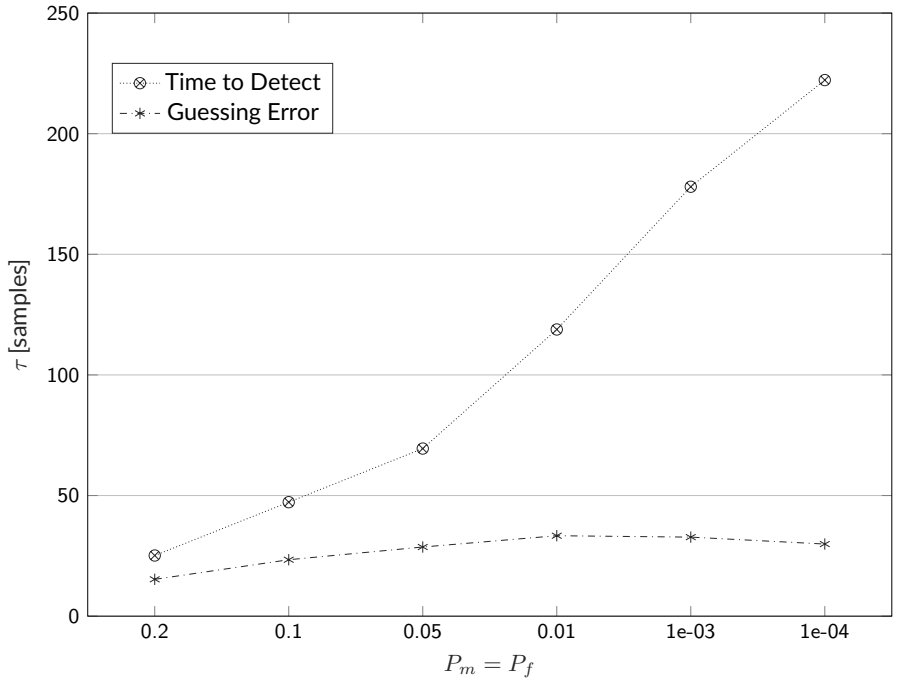


Figure 5.19: Failure Detection - time to detect and expected switch-time error tests. 49000 Montecarlo runs. No false alarms. Lower is better.

Results are quite interesting since the two tests provide very different results. The first, the time to detect (TA) test, shows that the higher the threshold, the higher the time to give a response. Furthermore, simulations performed using different p_m values provide very different results: the simulations using the largest value of p_m are up to ten times faster to detect then the smallest one. The second test, instead, shows that there is not so much difference between thresholds when estimation of the switching time is performed. This can be seen better looking at the median of the estimation test results. This index describes the behaviour of the fifty percent of the tests and shows that the values are pretty close to each other, especially when high thresholds are adopted. Indeed, the difference between the simulations using the three largest thresholds is lower than one sample.

$p_m = p_f$	20%	10%	5%	1%	0.1%	0.01%
FD-TA - mean [samples]	25.14	47.25	69.48	118.84	178.01	222.22
FD-TA - median [samples]	10.79	17.14	23.45	38.71	61.43	84.75
FD-TE - mean [samples]	15.27	23.39	28.66	33.35	32.76	29.89
FD-TE - median [samples]	7.54	10.53	12.45	14.72	15.26	15.35

Table 5.9: Failure Detection - time to detect (TA) and expected switch-time error (TE) tests. 49000 Montecarlo runs. No false alarms. Lower is better.

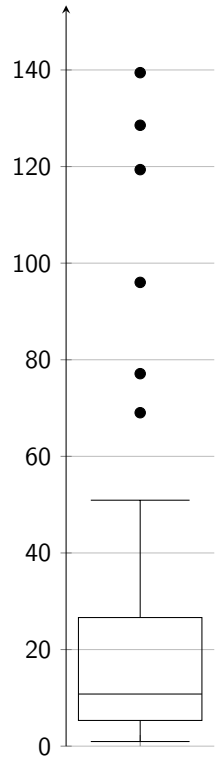
However, the algorithm performances are better using model with the lowest values than with the highest ones. Intuitively it might seem strange since, usually, a low threshold means a not full reliable result, but in reality this fact is fully predictable: the used estimation method is based on taking note of the last time the sum was reset to zero. If the threshold is low, the algorithm takes only few moments between the reset and the transmission notification as the test is very sensitive. Vice-versa, if the threshold is very high, it takes a long time between the detection and the notification since the Λ_G value has to become large to trig the threshold, as observed in the previous test. As a result, it is more likely than the previous case to have a series of noisy measures that reset the counter even if the switch has already been made. Therefore, a substantial error is likely to occur.

Anyway, these results cannot be fully trusted since it seems that some variance is present: the mean and the median differ a lot in both tests. Similarly to what was done for the algorithm in the commissioning error detection Section, also in this case it is useful to show the relative box plots. These are reported in Figure 5.20 and Figure 5.21 respectively. There, it can be seen that, in both tests, there are six models that do not behave like the others. They are the same ones that were found in the commissioning error detection test, i.e. the ones that have $\delta = 0.65$. Hence, the results of the first test provide an evidence that CUSUM is nothing more than a SPRT where the counter is repeatedly reset: this is the motivation why, for the time to detect tests, the both algorithms have very close results.

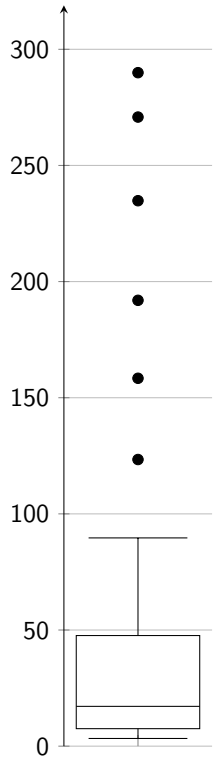
In the box plots it can be also seen that, in addition to the median of the tests, the third quartile is also very low compared to the maximum value reached. This means that the algorithm works well in almost the situations, since in the seventy percent of the simulated models lead to good performances, but there are some cases make it particularly slow and inaccurate. However, also in this case a more detailed analysis is required and will be carried on in the following Subsection.

5.4.2 Failure Detection - γ and δ analysis

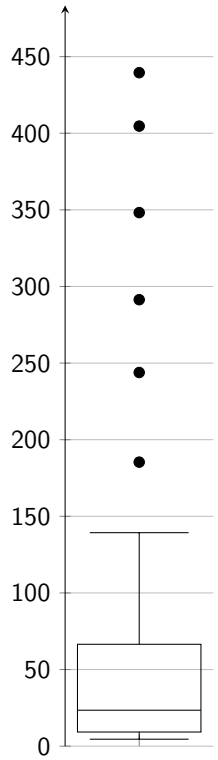
In this part, the most interesting values found, grouped according to the used model, are reported. It is worth remembering that γ describes the transition matrix whereas δ defines the observation matrix of the model. Obviously, the time to detect and the estimation switching time tests were carried out by removing the simulations that reported a false alarm.



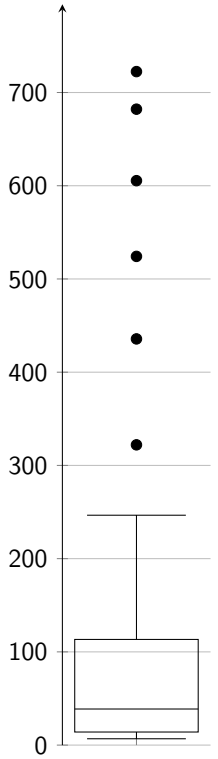
(a) $p_f = p_m = 2 \times 10^{-1}$



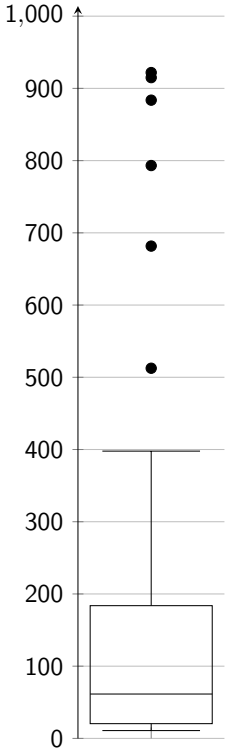
(b) $p_f = p_m = 10^{-1}$



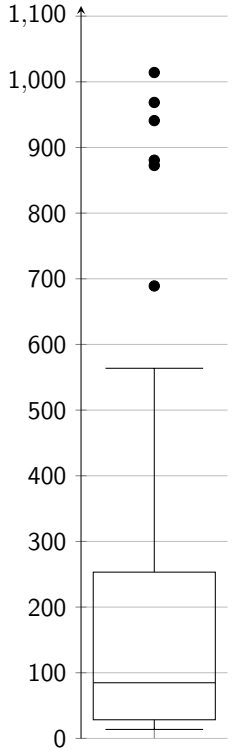
(c) $p_f = p_m = 5 \times 10^{-2}$



(d) $p_f = p_m = 10^{-2}$

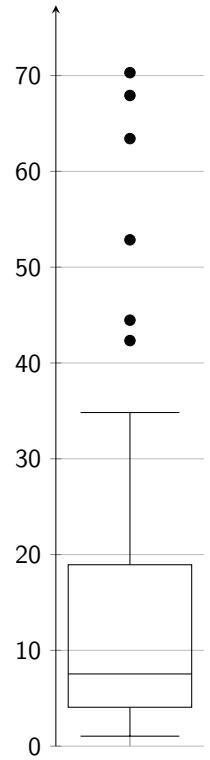


(e) $p_f = p_m = 10^{-3}$

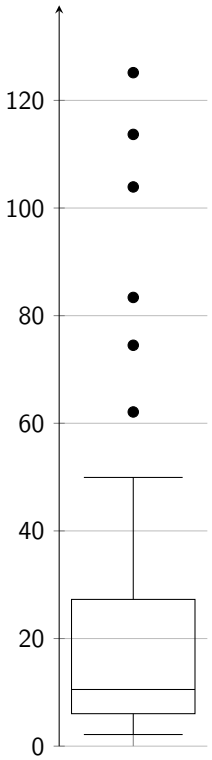


(f) $p_f = p_m = 10^{-4}$

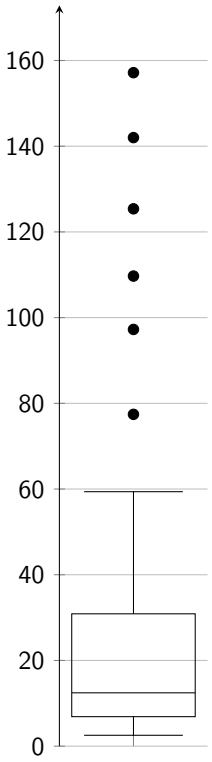
Figure 5.20: Failure Detection - box plots for time to detect test. 49000 Montecarlo runs. No false alarms. Lower is better.



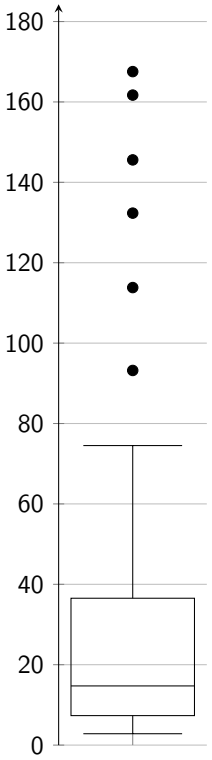
(a) $p_f = p_m = 2 \times 10^{-1}$



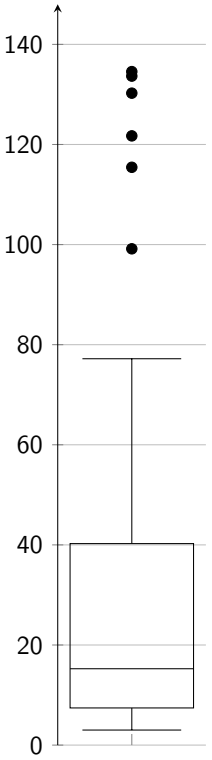
(b) $p_f = p_m = 10^{-1}$



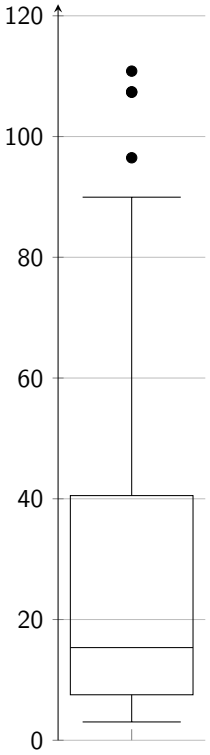
(c) $p_f = p_m = 5 \times 10^{-2}$



(d) $p_f = p_m = 10^{-2}$



(e) $p_f = p_m = 10^{-3}$

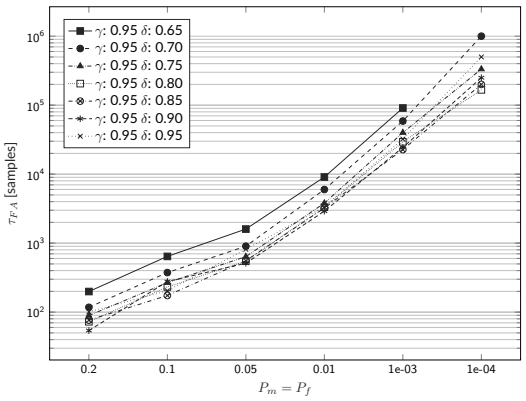


(f) $p_f = p_m = 10^{-4}$

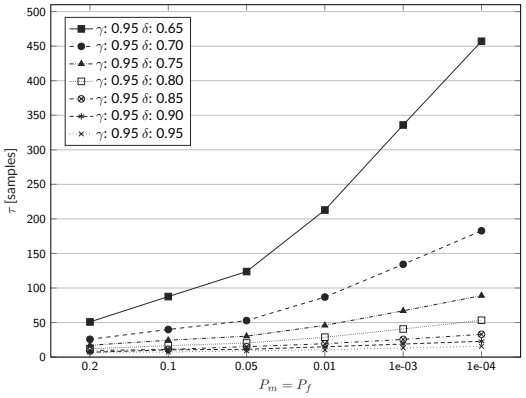
Figure 5.21: Failure Detection - box plots for switch-time estimation error. 49000 Montecarlo runs. No false alarms. Lower is better.

Failure Detection - $\gamma = 0.95$ analysis

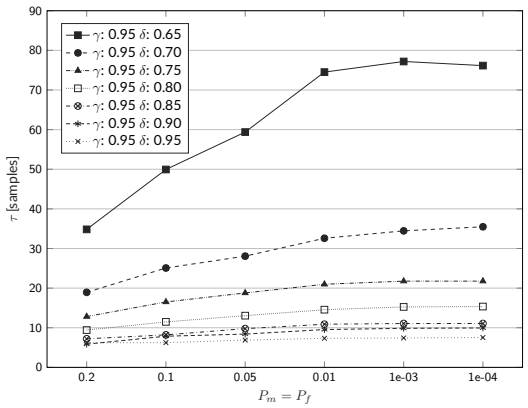
In Figure 5.22 the results of tests performed using $\gamma = 0.95$ are reported. In the first test, it turns out that the lower the δ, the better the algorithm's false alarm performances. Indeed, for the last used threshold, the plot mark for the model with $\delta = 0.65$ is missing, since no false alarm were found in the time window of each one of the one hundred simulations carried on. Since the time window is set to ten thousand samples, this means that the algorithm is very judicious with regard to false alarms, especially for models with highly noisy sensors. In other cases, however, the situation is not tragic as they are all in the same order of magnitude.



(a) FD-FA



(b) FD-TA



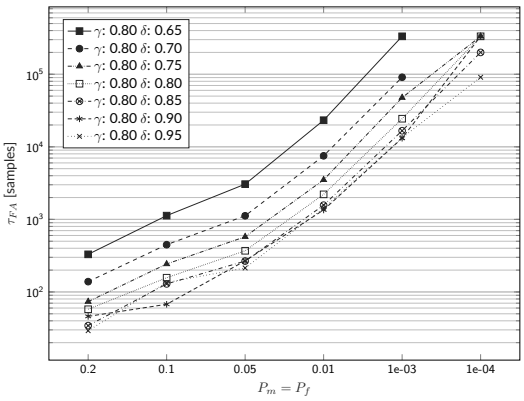
(c) FD-TE

Figure 5.22: Failure detection tests with fixed $\gamma = 0.95$. FA: 100 Montecarlo runs executed, 10000 samples for each run, higher is better. TA/TE: 1000 Montecarlo runs executed, no false alarm, lower is better.

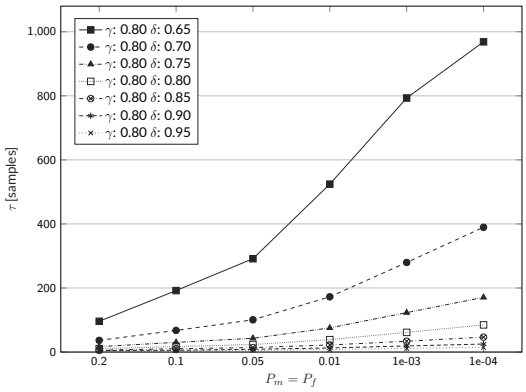
In the other two tests, the results reflect what is expected since an order between the different models can be easily found: the higher the δ value, the faster and more precise the algorithm. Furthermore, similarly to what happened in commissioning error detection, the higher the threshold value, the longer the time taken by the algorithm to make a choice if the same model is analysed.

Failure Detection - $\gamma = 0.80$ analysis

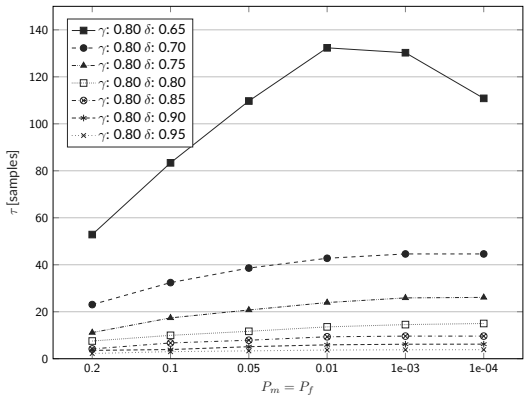
As previously performed, also in this case the results obtained using an intermediate γ value, i.e. $\gamma = 0.80$, are reported. They can be seen in Figure 5.23.



(a) FD-FA



(b) FD-TA



(c) FD-TE

Figure 5.23: Failure detection tests with fixed $\gamma = 0.80$. FA: 100 Montecarlo runs executed, 10000 samples for each run, higher is better. TA/TE: 1000 Montecarlo runs executed, no false alarm, lower is better.

In the first test, the situation is pretty similar to the one found with the fixed $\gamma = 0.95$. The main differences can be found in that the distances between the different models are slightly increasing, and that there are two models that, using the last threshold, have not recorded false alarms. Same as before, higher δ values lead the algorithm to commit errors with a higher frequency, i.e. the time between false alarms decreases.

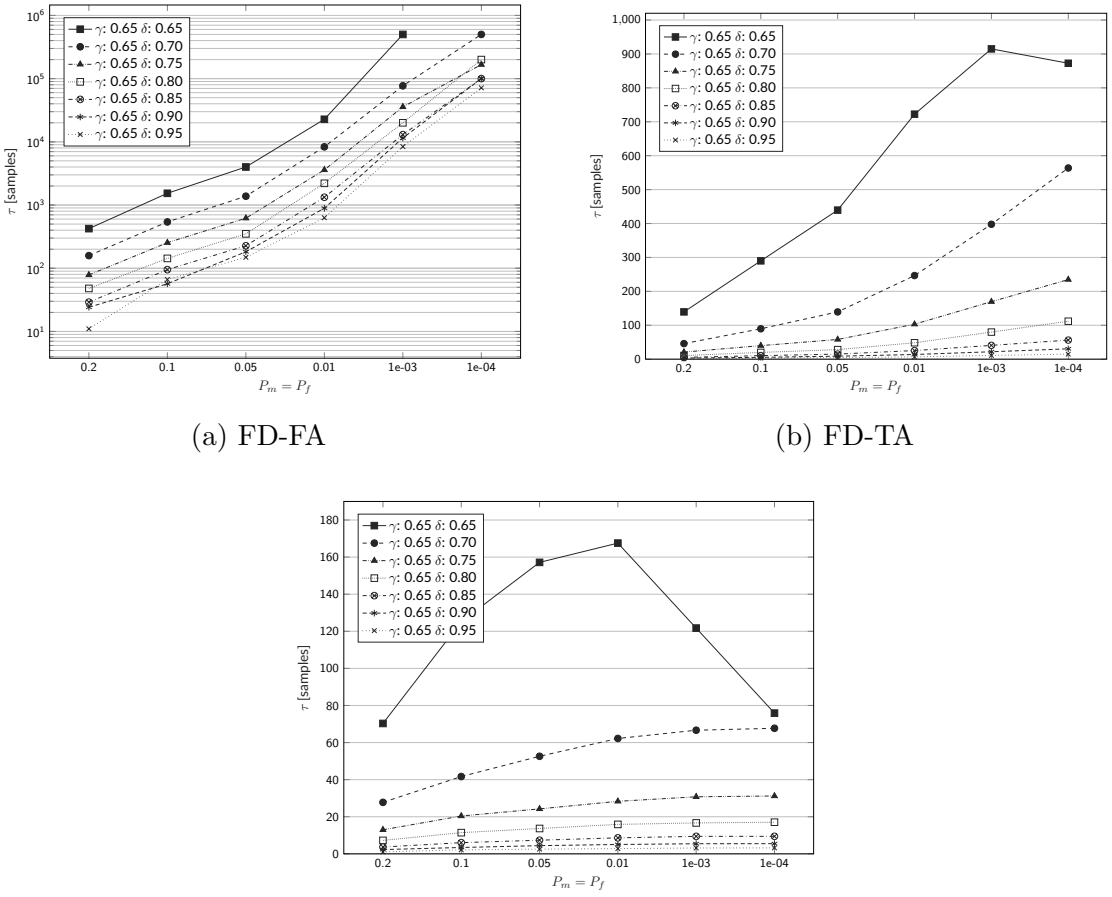
The second test plot has a shape very close to the one reported for the test using $\gamma = 0.95$: the lower the model δ value, the faster the algorithm. In this case, however, the scale is two times larger than before, thus highlighting that the overall performances are worse. However, also in this case, the worst performances are obtained by the model which has the lowest δ value: it seems that, using the latter value, the algorithm is very careful about making decision, and therefore is slower

but less prone to giving false alarms.

The third test, instead, is more interesting: as it can be seen, the shapes of the plots of all the models are the same of the previous tests except for the one with the lowest δ. For this model, the estimation error is not growing monotonous but, instead, decreases. This is quite strange and should be analysed using a different value of γ.

Failure Detection - $\gamma = 0.65$ analysis

The last comparative for a fixed γ value is performed for $\gamma = 0.65$, a low value. This means that the simulated models have quite frequent state changes. The results are reported in Figure 5.24.



(a) FD-FA

(b) FD-TA

(c) FD-TE

Figure 5.24: Failure detection tests with fixed $\gamma = 0.65$. FA: 100 Montecarlo runs executed, 10000 samples for each run, higher is better. TA/TE: 1000 Montecarlo runs executed, no false alarm, lower is better.

The first test results are pretty similar to the ones found for the other γ values. Therefore, the lower the δ value, the better. In this case too, the algorithm makes no errors for the model with $\delta = 0.65$.

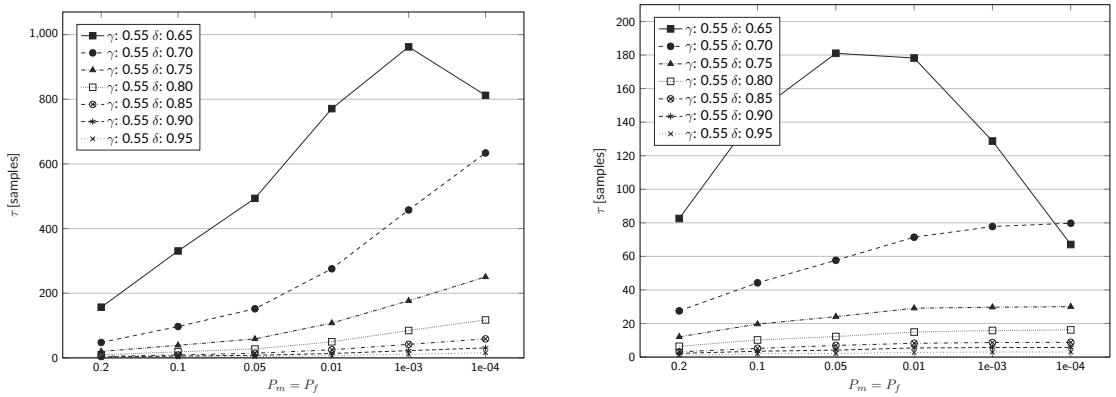
The other two tests, instead are very interesting: in these ones, the shapes of the results about the smallest value of δ are very different than before. In the

second test, only for the last threshold a strange value is reached whereas, in the third test, the curve loses its monotonicity starting from the fourth threshold: for the higher threshold, the performances for models with $\delta = 0.65$ or $\delta = 0.70$ are pretty the similar. However, evidence of this behaviour had already been found in the previous analysis: to find more information about this, an analysis with a fixed δ is required, like the one is carried out in the following subsections.

Failure Detection - overall fixed γ analysis

Summing up, a leitmotiv can easily be found: low δ values models allow the algorithm to reach good results in false alarm tests, but, at same time, lead to poor performances in the time to detect test and in the estimation error test. The models with high δ values lead the algorithm to have the opposite behaviour: it has good performances in the second and third tests whereas in the first the performances are appreciably lower than the ones obtained in the other models.

In the analyses carried out, a strangeness had emerged: for the lowest values of δ, the time to detect test curve and one about the estimation error test were very different from those obtained with the other values of δ. In addition, it had been seen that as the γ value fells, the curve changed more and more. To bring further information, in Figure 5.25 the situation using $\gamma = 0.55$ is reported.



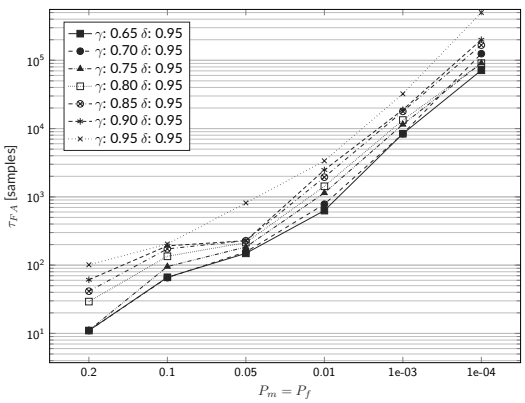
(a) FD-TA (b) FD-TE

Figure 5.25: Failure detection tests with fixed $\delta = 0.95$. FA: 100 Montecarlo runs executed, 10000 samples for each run, higher is better. TA/TE: 1000 Montecarlo runs executed, no false alarm, lower is better.

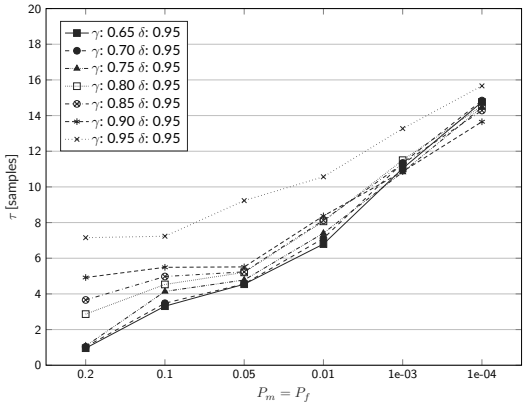
This Figure proves that the algorithm behaviour with the $\delta = 0.65$ models in these two tests is not random but has a precise motivation: however, a in detail analysis with δ fixed is required before a verdict can be issued.

Failure Detection - $\delta = 0.95$ analysis

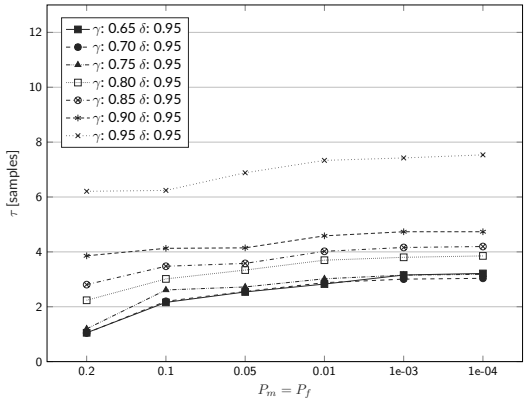
The first analysis using a fixed δ value models is the one using $\delta = 0.95$. This means that the workstations with powerful sensors are simulated: the results are plotted in Figure 5.26.



(a) FD-FA



(b) FD-TA



(c) FD-TE

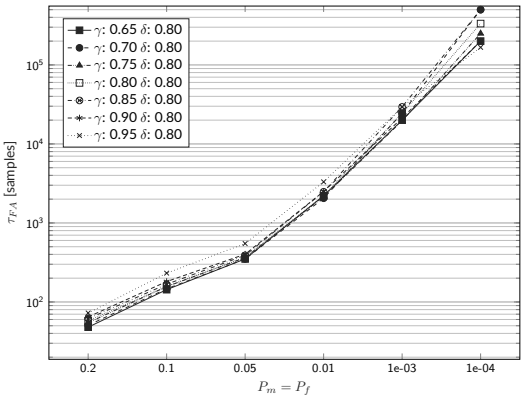
Figure 5.26: Failure detection tests with fixed $\delta = 0.95$. FA: 100 Montecarlo runs executed, 10000 samples for each run, higher is better. TA/TE: 1000 Montecarlo runs executed, no false alarm, lower is better.

Similar to what happens in simulations with a fixed γ value, no model allows the algorithm to achieve the best performance in all three tests: indeed, the best case in the first test become the worst in the other two tests and vice-versa. In this analysis, however, the difference of achieved performances using the different models is really small (in the last two tests is less than four samples), and therefore, the following conclusion can be drawn: the performance of the algorithm does not depend on the value of γ if that of δ is high.

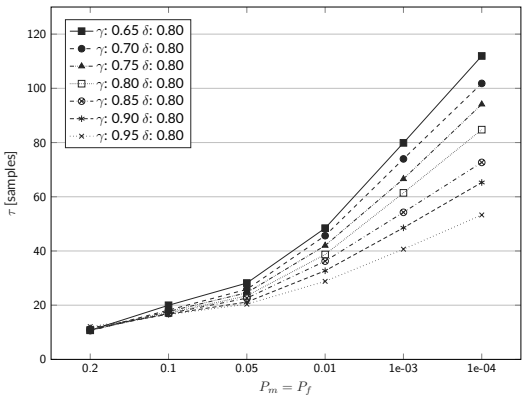
Failure Detection - $\delta = 0.80$ analysis

The results of the simulations carried on using models with $\delta = 0.8$ are reported in Figure 5.27.

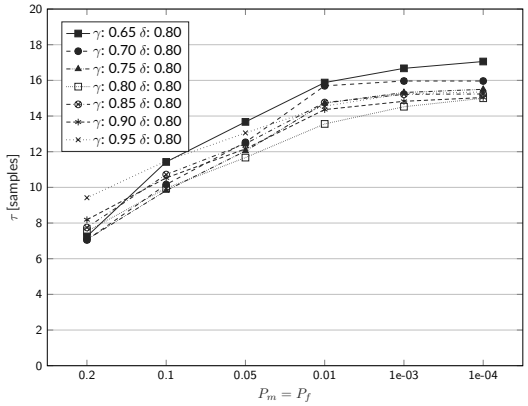
These simulations provide quite different results than before: in the first tests, the algorithm performs well in all the situations and the δ value does not matter so much. In the second test and in the third test the results using different models are very similar to each other although it seems that, unlike the previous case, the models with higher δ values allow the algorithm to have better performances than



(a) FD-FA



(b) FD-TA



(c) FD-TE

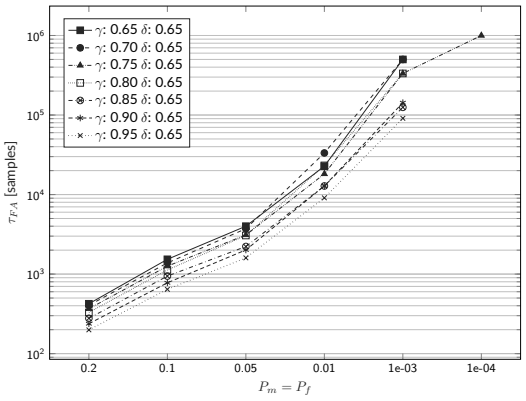
Figure 5.27: Failure detection tests with fixed $\delta = 0.80$. FA: 100 Montecarlo runs executed, 10000 samples for each run, higher is better. TA/TE: 1000 Montecarlo runs executed, no false alarm, lower is better.

using models with noisy sensors. However, the difference of performances using different models is quite negligible also in this case.

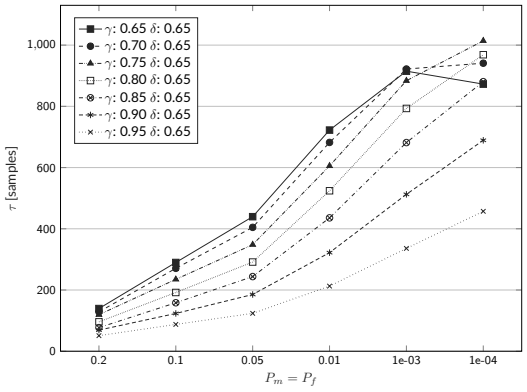
Failure Detection - $\delta = 0.65$ analysis

The last analysis is the most interesting one since it is performed using $\delta = 0.65$ models, the value that in the fixed γ analysis led the algorithm to an unusual behaviour. The results are reported in Figure 5.28. It can be clearly seen that the difference of performances caused by models are more evident than before, especially in the second and third tests.

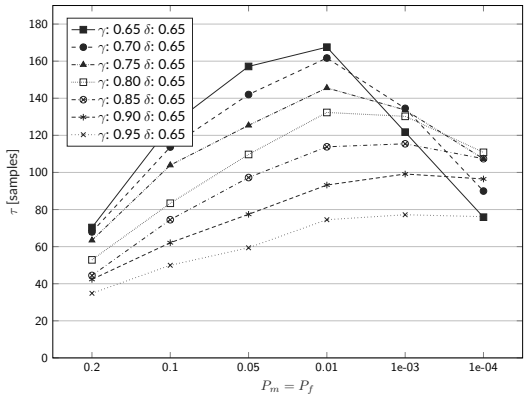
The first test shows that, with a low value of δ, the algorithm is very reliable as, using the larger threshold, only one false alarm is reported on the 600 000 simulated samples. The second, instead, shows that the performances drop a lot than before: however, it is worth highlighting that the unusual algorithm's behaviour appears for the models with γ equal to 0.65 and 0.70. In these cases, the performances of the algorithm in the second test are better using the largest threshold than using the second largest. This is counter-intuitive as the larger the threshold, the slower



(a) FD-FA



(b) FD-TA



(c) FD-TE

Figure 5.28: Failure detection tests with fixed $\delta = 0.65$. FA: 100 Montecarlo runs executed, 10000 samples for each run, higher is better. TA/TE: 1000 Montecarlo runs executed, no false alarm, lower is better.

the algorithm, as seen before. The third test highlights a time more this fact: the model with the smallest value of γ is the worst in this test using the first four threshold whereas is the best using the largest threshold. This behaviour is fairly faithfully reproduced by the model with $\gamma = 0.7$.

Failure Detection - overall fixed δ analysis

The last three analyses showed that the γ value of the model is not so decisive for the algorithm performances, although for models with a low δ value the performances vary quite a bit. Furthermore, it was shown that the algorithm has very good performances also in models with low δ and γ values, even if a high threshold shall be used.

Analysing these results, the most interesting thing discovered is that, using high thresholds, the Algorithm performs better using noisy sensors than using good ones if the model has frequent changes of states. This is quite strange since it is expected that the noisier the sensors, the worse the performances of the algorithm. However, this is a situation out of the assumptions since it is assumed that the

workstations have state changes with low frequency: for this reason, this situation will be analysed in a future work.

Chapter 6

Conclusions & Future Works

In this opus, a new way of modelling sensors and workstations has been presented along with some algorithms for fault detection. In the light of the birth of smart lamps with built-in sensors, it is necessary to develop new methods to describe their behaviour and monitor them. The choice to describe them fell on the hidden Markov models: using them, it was possible to obtain a powerful description that allows to solve the main problems that smart lamps brought to light. Furthermore, this type of description allows to reuse parts of some available algorithms. The problems are the commissioning error detection, i.e. to detect if some mistakes were committed by the installer during the commissioning, and the failure detection, i.e. to detect if during the life of the workstation some changes to lamp position were performed.

In this paper, two methods were presented to solve the problems of commissioning error detection and failure detection respectively. The first was solved developing an algorithm based on the sequential probability ratio test whereas the second issue was solved thanks to a custom version of the cumulative summation algorithm. The second algorithm allows to estimate the fault time, too.

The main problem encountered during the development of both algorithms was how to calculate the probability of an observation given only the previous ones. The solution was tough to be found but the result repaid all the efforts. Indeed, the solution found is not restricted to this work, but it can be used in all situations where there are hidden Markov models.

To validate the two developed algorithms, several tests were carried out using simulated models. With regards to the first algorithm, tests about miss detection, false alarms, and time to detect were carried on using forty-nine different models. The results obtained were very encouraging, especially in the case of models with slow dynamics and precise sensors, the most common ones. At the same manner, the second algorithm was checked: the time to detect, the mean time between false alarms and the fault time estimation error tests were performed using forty-nine different workstation models. In this case too, the results were good, especially for the common models: the mean time between false alarms is higher than one million samples whereas the estimation error is lower than ten samples.

The tests highlight that the modelling is suitable, and that these algorithms are quite ready to be implemented in every day office workstations, even if some more work shall be accomplished. Indeed, some aspects of the algorithms are

not unravelled yet: for example, the way the algorithm performances depend on workstation parameters is not fully understood yet, but only outlined. Moreover, it is interesting to perform an analysis on the sensitivity of the algorithms, i.e. how their performances change if the algorithms do not know the real parameters of the workstations. A comparative with other algorithms developed, especially the ones based on machine learning, is interesting too: this can be performed in a follow-up paper along with an on-field test in an office; the latter shall be performed before real implementation to be sure that simulations match the real environment.

Hence, this opus represents a turning point for the development of the topic treated. In particular, if the aspects that have not been taken into account in this analysis will be deepened, this work could be employed in a real environment in order to show its potential in practice.

Appendix A

Simulations Full Results

In this appendix, the results obtained thanks to the simulations are reported. It is worth noticing that, in the tables, to highlight the values that do not respect the thresholds a bold font is used.

A.1 Commissioning Error Detection

A.1.1 CED-MD test

γ	δ	20%	10%	5%	1%	0.1%	0.01%
0.65	0.65	17.6	7.6	2.8	0.0	0.0	0.0
0.65	0.70	20.4	9.6	5.2	2.0	0.0	0.0
0.65	0.75	18.4	**11.6**	5.6	2.0	0.0	0.0
0.65	0.80	**26.0**	**15.2**	**6.4**	0.4	0.0	0.0
0.65	0.85	**26.4**	10.0	4.8	1.2	0.0	0.0
0.65	0.90	**24.0**	**11.2**	5.6	0.8	0.0	0.0
0.65	0.95	**35.6**	**13.2**	**6.4**	0.4	0.0	0.0
0.70	0.65	18.8	8.8	3.6	1.6	0.0	0.0
0.70	0.70	**23.2**	**11.6**	5.6	1.2	0.0	0.0
0.70	0.75	**25.2**	10.8	6.0	0.0	0.0	0.0
0.70	0.80	20.4	10.0	**6.4**	0.4	0.0	0.0
0.70	0.85	**26.4**	**15.2**	**7.6**	0.8	0.0	0.0
0.70	0.90	**37.6**	**16.4**	**8.0**	0.4	0.4	0.0
0.70	0.95	**24.8**	**13.6**	**6.4**	1.2	0.4	0.0
0.75	0.65	**22.0**	10.8	4.0	1.2	0.4	0.0
0.75	0.70	17.2	10.4	5.6	1.6	0.0	0.0
0.75	0.75	19.2	9.2	4.8	0.4	0.0	0.0

0.75	0.80	20.4	10.4	5.2	2.0	0.4	0.0
0.75	0.85	**26.8**	**11.6**	4.8	0.4	0.0	0.0
0.75	0.90	**26.4**	**16.0**	**8.0**	**2.4**	0.0	0.0
0.75	0.95	**30.4**	**16.8**	**8.0**	1.6	0.0	0.0
0.80	0.65	**21.6**	10.8	4.8	0.8	0.0	0.0
0.80	0.70	**21.6**	**12.0**	5.6	0.4	0.4	0.0
0.80	0.75	20.8	8.0	4.0	1.6	0.0	0.0
0.80	0.80	**22.8**	10.4	6.0	1.2	0.0	0.0
0.80	0.85	**26.0**	10.4	5.6	1.2	0.0	0.0
0.80	0.90	18.8	7.2	4.0	0.8	0.0	0.0
0.80	0.95	**30.8**	**15.2**	**7.6**	0.4	0.0	0.0
0.85	0.65	18.4	8.0	4.4	0.4	0.0	0.0
0.85	0.70	20.8	**12.0**	5.2	0.8	0.0	0.0
0.85	0.75	**25.2**	**12.4**	6.0	1.6	0.0	0.0
0.85	0.80	17.6	8.8	3.6	0.4	0.0	0.0
0.85	0.85	20.8	8.0	4.4	**2.4**	0.0	0.0
0.85	0.90	**27.2**	**12.0**	4.8	1.6	0.8	0.0
0.85	0.95	20.0	**12.0**	5.6	**3.6**	0.4	0.4
0.90	0.65	**22.4**	**11.6**	**6.4**	2.0	0.0	0.0
0.90	0.70	17.2	5.6	4.0	2.0	0.0	0.0
0.90	0.75	19.2	**12.4**	5.6	0.0	0.0	0.0
0.90	0.80	20.4	10.4	4.4	0.4	0.0	0.0
0.90	0.85	18.8	10.4	3.6	0.8	0.0	0.0
0.90	0.90	16.8	6.8	5.6	2.0	0.0	0.0
0.90	0.95	**25.2**	**11.2**	6.0	1.6	0.0	0.0
0.95	0.65	16.0	6.0	2.8	1.2	0.0	0.0
0.95	0.70	15.2	8.0	2.4	0.8	0.0	0.0
0.95	0.75	17.2	7.6	4.0	0.8	0.4	0.4
0.95	0.80	18.4	9.6	4.8	1.2	0.0	0.0
0.95	0.85	18.0	8.0	6.0	0.8	0.4	0.0
0.95	0.90	**22.4**	8.4	4.0	1.2	0.0	0.0
0.95	0.95	**22.8**	9.2	**6.4**	2.0	0.4	0.0

Table A.1: CED-MD test: percent value of miss detection. 250 Montecarlo runs. Lower is better.

A.1.2 CED-FA test

γ	δ	20%	10%	5%	1%	0.1%	0.01%
0.65	0.65	**22.8**	**10.8**	**5.6**	**1.6**	0.0	0.0
0.65	0.70	15.6	9.6	3.6	0.8	0.0	0.0
0.65	0.75	16.0	6.8	3.2	**1.2**	**0.4**	0.0
0.65	0.80	17.6	8.8	4.0	**1.2**	0.0	0.0
0.65	0.85	15.2	6.8	1.6	0.8	0.0	0.0
0.65	0.90	13.2	5.6	1.6	0.0	0.0	0.0
0.65	0.95	5.2	3.2	1.6	0.4	0.0	0.0
0.70	0.65	**24.4**	**11.6**	**6.4**	**2.4**	0.0	0.0
0.70	0.70	18.0	7.6	2.4	0.0	0.0	0.0
0.70	0.75	13.6	6.0	3.2	0.0	0.0	0.0
0.70	0.80	16.8	6.4	3.2	0.0	0.0	0.0
0.70	0.85	12.8	8.0	4.8	0.8	0.0	0.0
0.70	0.90	14.0	8.4	**5.6**	0.4	0.0	0.0
0.70	0.95	5.6	3.6	2.0	0.0	0.0	0.0
0.75	0.65	**22.0**	8.0	3.6	0.8	0.0	0.0
0.75	0.70	16.4	7.6	2.4	0.8	0.0	0.0
0.75	0.75	12.4	8.0	3.2	0.4	0.0	0.0
0.75	0.80	16.8	6.8	3.2	0.4	0.0	0.0
0.75	0.85	11.2	4.8	4.0	0.4	0.0	0.0
0.75	0.90	11.2	3.2	0.8	0.4	0.0	0.0
0.75	0.95	4.0	3.2	1.2	0.0	0.0	0.0
0.80	0.65	16.0	**12.4**	**5.6**	0.0	0.0	0.0
0.80	0.70	18.8	8.8	4.8	0.8	**0.4**	0.0
0.80	0.75	20.0	**11.2**	**5.2**	0.4	0.0	0.0
0.80	0.80	14.4	8.0	3.2	0.4	0.0	0.0
0.80	0.85	12.0	6.0	2.8	0.0	0.0	0.0
0.80	0.90	12.0	7.2	3.2	0.4	0.0	0.0

0.80	0.95	5.2	4.4	2.0	0.0	0.0	0.0
0.85	0.65	19.6	**11.6**	3.6	0.8	0.0	0.0
0.85	0.70	17.6	8.8	**5.6**	**1.2**	0.0	0.0
0.85	0.75	16.0	9.2	4.0	0.4	0.0	0.0
0.85	0.80	10.4	4.4	2.8	**1.2**	0.0	0.0
0.85	0.85	10.4	5.6	1.6	0.8	0.0	0.0
0.85	0.90	11.2	5.6	2.8	**1.2**	0.0	0.0
0.85	0.95	7.2	5.6	4.4	**2.0**	0.0	0.0
0.90	0.65	18.4	8.0	4.0	**1.2**	0.0	0.0
0.90	0.70	12.8	7.2	2.8	0.4	0.0	0.0
0.90	0.75	16.8	9.2	3.2	0.4	0.0	0.0
0.90	0.80	14.8	5.6	4.0	0.4	0.0	0.0
0.90	0.85	13.2	7.2	2.4	**1.2**	**0.4**	0.0
0.90	0.90	10.0	5.2	3.2	0.8	0.0	0.0
0.90	0.95	2.0	2.0	0.4	0.0	0.0	0.0
0.95	0.65	**21.2**	**11.6**	4.8	**1.2**	0.0	0.0
0.95	0.70	13.6	6.0	2.4	0.4	0.0	0.0
0.95	0.75	15.6	4.8	2.8	0.0	0.0	0.0
0.95	0.80	14.0	4.0	3.6	**1.6**	**0.4**	0.0
0.95	0.85	8.0	6.0	3.2	0.4	0.0	0.0
0.95	0.90	6.0	5.6	4.4	**1.2**	0.0	0.0
0.95	0.95	5.2	2.4	0.4	0.0	0.0	0.0

Table A.2: CED-FA test: percent value of false alarm. 250 Montecarlo runs. Lower is better.

A.1.3 CED-OE test

γ	δ	20%	10%	5%	1%	0.1%	0.01%
0.65	0.65	**20.2**	9.2	4.2	0.8	0.0	0.0
0.65	0.70	18.0	9.6	4.4	**1.4**	0.0	0.0
0.65	0.75	17.2	9.2	4.4	**1.6**	**0.2**	0.0
0.65	0.80	**21.8**	**12.0**	**5.2**	0.8	0.0	0.0
0.65	0.85	**20.8**	8.4	3.2	1.0	0.0	0.0

0.65	0.90	18.6	8.4	3.6	0.4	0.0	0.0
0.65	0.95	**20.4**	8.2	4.0	0.4	0.0	0.0
0.70	0.65	**21.6**	**10.2**	5.0	**2.0**	0.0	0.0
0.70	0.70	**20.6**	9.6	4.0	0.6	0.0	0.0
0.70	0.75	19.4	8.4	4.6	0.0	0.0	0.0
0.70	0.80	18.6	8.2	4.8	0.2	0.0	0.0
0.70	0.85	19.6	**11.6**	**6.2**	0.8	0.0	0.0
0.70	0.90	**25.8**	**12.4**	**6.8**	0.4	**0.2**	0.0
0.70	0.95	15.2	8.6	4.2	0.6	**0.2**	0.0
0.75	0.65	**22.0**	9.4	3.8	1.0	**0.2**	0.0
0.75	0.70	16.8	9.0	4.0	**1.2**	0.0	0.0
0.75	0.75	15.8	8.6	4.0	0.4	0.0	0.0
0.75	0.80	18.6	8.6	4.2	**1.2**	**0.2**	0.0
0.75	0.85	19.0	8.2	4.4	0.4	0.0	0.0
0.75	0.90	18.8	9.6	4.4	**1.4**	0.0	0.0
0.75	0.95	17.2	10.0	4.6	0.8	0.0	0.0
0.80	0.65	18.8	**11.6**	**5.2**	0.4	0.0	0.0
0.80	0.70	**20.2**	**10.4**	**5.2**	0.6	**0.4**	0.0
0.80	0.75	**20.4**	9.6	4.6	1.0	0.0	0.0
0.80	0.80	18.6	9.2	4.6	0.8	0.0	0.0
0.80	0.85	19.0	8.2	4.2	0.6	0.0	0.0
0.80	0.90	15.4	7.2	3.6	0.6	0.0	0.0
0.80	0.95	18.0	9.8	4.8	0.2	0.0	0.0
0.85	0.65	19.0	9.8	4.0	0.6	0.0	0.0
0.85	0.70	19.2	**10.4**	**5.4**	1.0	0.0	0.0
0.85	0.75	**20.6**	**10.8**	5.0	1.0	0.0	0.0
0.85	0.80	14.0	6.6	3.2	0.8	0.0	0.0
0.85	0.85	15.6	6.8	3.0	**1.6**	0.0	0.0
0.85	0.90	19.2	8.8	3.8	**1.4**	**0.4**	0.0
0.85	0.95	13.6	8.8	5.0	**2.8**	**0.2**	**0.2**
0.90	0.65	**20.4**	9.8	**5.2**	**1.6**	0.0	0.0
0.90	0.70	15.0	6.4	3.4	**1.2**	0.0	0.0
0.90	0.75	18.0	**10.8**	4.4	0.2	0.0	0.0

0.90	0.80	17.6	8.0	4.2	0.4	0.0	0.0
0.90	0.85	16.0	8.8	3.0	1.0	**0.2**	0.0
0.90	0.90	13.4	6.0	4.4	**1.4**	0.0	0.0
0.90	0.95	13.6	6.6	3.2	0.8	0.0	0.0
0.95	0.65	18.6	8.8	3.8	**1.2**	0.0	0.0
0.95	0.70	14.4	7.0	2.4	0.6	0.0	0.0
0.95	0.75	16.4	6.2	3.4	0.4	**0.2**	**0.2**
0.95	0.80	16.2	6.8	4.2	**1.4**	**0.2**	0.0
0.95	0.85	13.0	7.0	4.6	0.6	**0.2**	0.0
0.95	0.90	14.2	7.0	4.2	**1.2**	0.0	0.0
0.95	0.95	14.0	5.8	3.4	1.0	**0.2**	0.0

Table A.3: CED-OE test: percent value of miss detection or false alarm. 500 Montecarlo runs. Lower is better.

A.1.4 CED-T test

γ	δ	20%	10%	5%	1%	0.1%	0.01%
0.65	0.65	185.2	399.1	588.9	961.0	1485.9	1962.0
0.65	0.70	67.8	131.1	189.8	309.3	479.5	647.0
0.65	0.75	29.2	56.8	83.7	132.4	203.1	273.3
0.65	0.80	15.6	28.6	41.8	67.3	101.3	136.0
0.65	0.85	8.6	16.7	22.9	36.5	54.1	71.0
0.65	0.90	5.4	9.5	13.2	21.2	30.2	39.8
0.65	0.95	3.8	6.4	8.3	12.6	17.9	23.3
0.70	0.65	174.3	347.4	524.3	893.6	1398.0	1875.0
0.70	0.70	63.7	128.1	186.9	296.6	437.9	592.1
0.70	0.75	29.4	58.7	83.7	135.2	199.1	263.6
0.70	0.80	14.8	27.7	40.3	66.6	98.5	128.8
0.70	0.85	8.8	15.2	22.9	37.9	54.6	71.0
0.70	0.90	5.6	10.4	13.9	22.0	32.1	41.4
0.70	0.95	4.0	6.0	8.2	12.3	17.7	23.3
0.75	0.65	153.0	324.1	470.6	782.2	1209.1	1588.9
0.75	0.70	57.7	108.2	166.1	271.8	410.3	545.4

0.75	0.75	25.8	50.1	73.5	116.1	175.7	227.4
0.75	0.80	14.6	28.9	40.5	64.1	93.9	124.3
0.75	0.85	8.6	15.9	22.7	37.3	53.8	69.1
0.75	0.90	6.1	10.3	13.8	21.1	31.4	40.7
0.75	0.95	4.2	6.3	8.5	13.0	18.9	24.4
0.80	0.65	133.3	268.6	407.5	699.6	1034.6	1374.6
0.80	0.70	52.5	98.6	151.2	253.0	379.5	500.3
0.80	0.75	26.2	51.1	73.2	116.2	172.6	226.6
0.80	0.80	15.2	26.8	37.9	61.1	89.4	117.1
0.80	0.85	8.8	15.7	21.5	33.7	50.8	67.1
0.80	0.90	6.1	10.2	13.6	21.2	30.6	39.5
0.80	0.95	5.1	7.1	9.4	13.9	19.4	25.1
0.85	0.65	110.0	220.6	340.6	554.9	826.4	1092.2
0.85	0.70	46.4	87.8	128.2	208.0	308.2	405.9
0.85	0.75	22.8	43.7	62.9	101.2	153.2	202.9
0.85	0.80	13.7	24.6	34.3	55.0	80.0	105.7
0.85	0.85	9.6	16.1	22.8	34.0	49.8	65.4
0.85	0.90	6.7	10.8	14.6	21.7	31.5	41.1
0.85	0.95	5.4	7.6	9.7	14.7	21.7	28.0
0.90	0.65	91.8	183.3	266.0	437.2	661.4	865.7
0.90	0.70	38.9	72.8	103.0	169.1	257.6	346.7
0.90	0.75	21.5	37.0	55.7	91.2	133.5	177.6
0.90	0.80	13.9	24.4	34.2	54.1	79.5	104.0
0.90	0.85	10.4	17.0	22.8	34.7	50.6	66.2
0.90	0.90	7.6	12.8	16.3	24.7	35.3	45.1
0.90	0.95	6.5	8.8	11.6	16.5	23.5	30.4
0.95	0.65	67.5	131.2	187.8	316.7	481.0	632.3
0.95	0.70	34.8	67.7	94.7	148.1	222.4	290.2
0.95	0.75	21.8	36.9	51.6	81.5	121.2	160.8
0.95	0.80	15.5	26.9	35.2	53.4	77.8	103.7
0.95	0.85	13.5	21.2	27.4	41.7	59.4	77.3
0.95	0.90	11.2	17.3	21.6	32.0	44.1	56.7
0.95	0.95	9.8	14.5	18.1	25.4	34.5	44.0

Table A.4: CED-T test: samples to make to a choice. 500 Montecarlo runs. Lower is better.

A.2 Failure Detection

A.2.1 FD-FA test

γ	δ	20	10	5	1	0.1	0.01
0.65	0.65	423.37	1533.74	4000.00	2.3×10^4	5.0×10^5	$> 10^6$
0.65	0.70	158.40	540.25	1379.31	8.3×10^3	7.7×10^4	5.0×10^5
0.65	0.75	78.44	253.74	619.20	3.6×10^3	3.6×10^4	1.7×10^5
0.65	0.80	47.95	143.12	350.39	2.2×10^3	2.0×10^4	2.0×10^5
0.65	0.85	29.10	95.08	226.91	1.3×10^3	1.3×10^4	1.0×10^5
0.65	0.90	24.19	56.76	182.42	8.9×10^2	1.1×10^4	1.0×10^5
0.65	0.95	11.02	66.39	148.83	6.3×10^2	8.4×10^3	7.1×10^4
0.70	0.65	404.86	1381.22	3717.47	3.3×10^4	5.0×10^5	$> 10^6$
0.70	0.70	152.28	508.39	1282.05	8.0×10^3	7.1×10^4	1.0×10^6
0.70	0.75	79.49	246.61	619.20	3.7×10^3	4.8×10^4	1.0×10^6
0.70	0.80	50.71	146.67	350.02	2.1×10^3	2.4×10^4	5.0×10^5
0.70	0.85	31.55	105.05	232.99	1.3×10^3	1.2×10^4	3.3×10^5
0.70	0.90	27.26	60.93	188.57	9.9×10^2	9.3×10^3	7.1×10^4
0.70	0.95	11.02	65.58	156.10	7.8×10^2	8.4×10^3	1.3×10^5
0.75	0.65	366.84	1257.86	3105.59	1.8×10^4	3.3×10^5	1.0×10^6
0.75	0.70	147.45	479.85	1212.12	7.0×10^3	7.7×10^4	1.0×10^6
0.75	0.75	78.08	253.61	612.37	3.6×10^3	3.2×10^4	1.0×10^6
0.75	0.80	54.56	156.42	364.30	2.1×10^3	2.0×10^4	2.5×10^5
0.75	0.85	33.27	118.22	252.84	1.5×10^3	1.4×10^4	1.3×10^5
0.75	0.90	39.05	65.90	221.29	1.2×10^3	1.2×10^4	9.1×10^4
0.75	0.95	11.07	95.40	184.30	1.1×10^3	1.1×10^4	9.1×10^4
0.80	0.65	329.82	1126.13	3058.10	2.3×10^4	3.3×10^5	$> 10^6$
0.80	0.70	138.66	447.83	1121.08	7.5×10^3	9.1×10^4	$> 10^6$
0.80	0.75	73.55	242.78	576.37	3.5×10^3	4.8×10^4	3.3×10^5
0.80	0.80	57.73	156.99	369.28	2.2×10^3	2.4×10^4	3.3×10^5

0.80	0.85	34.34	129.05	266.24	1.6×10^3	1.7×10^4	2.0×10^5
0.80	0.90	46.08	67.56	268.31	1.3×10^3	1.3×10^4	3.3×10^5
0.80	0.95	29.26	134.83	213.95	1.4×10^3	1.3×10^4	9.1×10^4
0.85	0.65	277.85	943.40	2222.22	1.3×10^4	1.3×10^5	$> 10^6$
0.85	0.70	127.16	410.00	1000.00	6.6×10^3	7.7×10^4	$> 10^6$
0.85	0.75	73.77	234.58	572.41	3.5×10^3	4.3×10^4	$> 10^6$
0.85	0.80	61.47	167.53	389.11	2.5×10^3	2.9×10^4	5.0×10^5
0.85	0.85	33.62	140.81	303.86	1.8×10^3	1.6×10^4	1.3×10^5
0.85	0.90	50.96	68.20	312.89	1.5×10^3	1.8×10^4	1.0×10^6
0.85	0.95	41.53	173.01	228.26	1.9×10^3	1.8×10^4	1.7×10^5
0.90	0.65	237.53	777.00	2012.07	1.3×10^4	1.4×10^5	$> 10^6$
0.90	0.70	120.86	379.22	947.87	5.7×10^3	5.9×10^4	1.0×10^6
0.90	0.75	77.75	237.76	562.75	3.2×10^3	3.0×10^4	2.0×10^5
0.90	0.80	64.65	181.23	402.09	2.5×10^3	2.1×10^4	2.0×10^5
0.90	0.85	50.82	154.92	367.24	2.0×10^3	2.0×10^4	1.7×10^5
0.90	0.90	52.71	154.80	378.50	1.8×10^3	2.0×10^4	1.7×10^5
0.90	0.95	61.06	191.53	228.26	2.5×10^3	1.9×10^4	2.0×10^5
0.95	0.65	198.97	642.67	1600.00	9.1×10^3	9.1×10^4	$> 10^6$
0.95	0.70	117.47	374.67	904.16	6.0×10^3	5.9×10^4	1.0×10^6
0.95	0.75	89.40	267.45	637.76	3.8×10^3	4.0×10^4	3.3×10^5
0.95	0.80	72.53	231.59	548.85	3.3×10^3	2.9×10^4	1.7×10^5
0.95	0.85	76.54	174.22	556.17	3.2×10^3	2.3×10^4	2.0×10^5
0.95	0.90	54.10	277.93	511.77	2.9×10^3	2.4×10^4	2.5×10^5
0.95	0.95	101.22	203.96	815.00	3.4×10^3	3.2×10^4	5.0×10^5

Table A.5: FD-FA test: mean time between two false alarms. 100 Montecarlo runs. 10000 samples for each run. Higher is better.

A.2.2 FD-TA test

γ	δ	20	10	5	1	0.1	0.01
0.65	0.65	139.44	289.94	439.53	722.43	914.91	872.60
0.65	0.70	45.93	89.65	139.32	246.58	397.76	563.71
0.65	0.75	20.89	39.92	58.14	103.15	169.53	234.72
0.65	0.80	10.73	19.97	28.17	48.42	79.88	111.94

0.65	0.85	5.32	10.38	15.01	25.38	40.68	56.23
0.65	0.90	3.32	5.42	8.80	13.80	21.90	30.55
0.65	0.95	0.96	3.31	4.54	6.80	11.04	14.76
0.70	0.65	128.55	270.83	404.66	682.13	921.90	940.91
0.70	0.70	43.08	83.26	130.58	232.89	374.24	520.25
0.70	0.75	19.42	36.73	54.95	96.94	154.46	212.97
0.70	0.80	10.62	18.19	25.84	45.64	73.97	101.82
0.70	0.85	5.73	10.86	14.72	24.17	38.67	53.23
0.70	0.90	3.79	5.49	8.30	13.00	20.40	28.37
0.70	0.95	1.03	3.48	4.56	7.11	11.34	14.84
0.75	0.65	119.36	234.82	348.21	605.43	883.70	1014.17
0.75	0.70	39.59	74.62	113.08	205.86	338.10	465.84
0.75	0.75	17.56	33.54	47.78	85.86	140.83	196.28
0.75	0.80	10.60	17.67	24.60	41.99	66.58	94.12
0.75	0.85	5.82	10.17	13.88	23.28	36.50	49.54
0.75	0.90	4.33	5.27	8.89	14.02	20.41	27.39
0.75	0.95	1.08	4.14	4.78	7.40	10.83	14.46
0.80	0.65	96.02	191.93	291.37	524.21	793.17	968.57
0.80	0.70	36.48	67.52	100.63	172.44	279.64	389.55
0.80	0.75	16.88	30.29	42.97	75.42	123.13	170.73
0.80	0.80	10.79	17.14	23.45	38.71	61.43	84.75
0.80	0.85	5.62	10.50	14.01	22.14	33.82	46.59
0.80	0.90	4.97	5.64	8.67	12.81	19.03	25.01
0.80	0.95	2.87	4.53	5.21	8.07	11.50	14.54
0.85	0.65	77.12	158.37	243.86	435.70	681.64	880.62
0.85	0.70	31.48	60.67	86.71	149.12	242.68	332.93
0.85	0.75	17.13	29.12	40.50	69.06	109.85	149.45
0.85	0.80	11.03	16.94	22.59	36.25	54.26	72.68
0.85	0.85	5.77	10.23	13.35	20.56	30.75	42.05
0.85	0.90	4.97	5.50	8.94	12.71	18.47	24.01
0.85	0.95	3.66	4.98	5.24	8.13	11.30	14.29
0.90	0.65	69.03	123.40	185.37	322.07	512.52	689.08
0.90	0.70	26.63	47.61	66.48	113.35	183.78	253.21

0.90	0.75	15.37	25.36	34.57	57.84	87.35	120.14
0.90	0.80	11.59	16.71	21.17	32.70	48.55	65.27
0.90	0.85	7.71	10.66	13.46	19.32	27.82	36.77
0.90	0.90	5.29	7.53	9.10	12.64	17.62	22.58
0.90	0.95	4.92	5.49	5.51	8.37	10.86	13.65
0.95	0.65	50.92	87.59	123.77	212.82	335.92	457.06
0.95	0.70	25.76	39.96	52.89	86.87	134.25	182.80
0.95	0.75	17.14	24.39	30.26	45.92	66.97	89.00
0.95	0.80	12.09	16.53	20.38	28.79	40.66	53.28
0.95	0.85	9.52	11.14	15.10	19.34	25.57	32.93
0.95	0.90	7.01	10.48	11.28	14.93	18.86	22.87
0.95	0.95	7.15	7.23	9.23	10.57	13.27	15.66

Table A.6: FD-TA test: samples to make to a choice. 1000 Montecarlo runs. Lower is better.

A.2.3 FD-TE test

γ	δ	20	10	5	1	0.1	0.01
0.65	0.65	70.30	125.15	157.16	167.54	121.72	75.88
0.65	0.70	27.76	41.69	52.60	62.21	66.65	67.70
0.65	0.75	12.96	20.41	24.24	28.33	30.76	31.20
0.65	0.80	7.25	11.43	13.67	15.88	16.68	17.06
0.65	0.85	3.68	6.03	7.36	8.64	9.43	9.44
0.65	0.90	2.30	3.43	4.42	5.04	5.45	5.49
0.65	0.95	1.05	2.16	2.54	2.83	3.16	3.21
0.70	0.65	67.92	113.68	142.01	161.70	134.56	89.95
0.70	0.70	25.79	38.83	49.38	58.82	62.97	63.97
0.70	0.75	12.02	18.50	23.54	29.20	30.76	31.11
0.70	0.80	7.05	10.16	12.53	15.69	15.96	15.96
0.70	0.85	4.13	6.58	7.64	8.93	9.62	9.62
0.70	0.90	2.53	3.54	4.23	4.95	5.29	5.30
0.70	0.95	1.04	2.20	2.56	2.88	3.01	3.04
0.75	0.65	63.42	103.92	125.39	145.58	133.66	107.33

0.75	0.70	24.11	35.61	44.47	55.07	61.74	62.08
0.75	0.75	11.48	18.55	22.53	26.58	28.87	29.18
0.75	0.80	7.08	9.82	12.04	14.72	15.32	15.50
0.75	0.85	4.32	6.38	7.61	9.04	9.68	9.68
0.75	0.90	2.97	3.53	4.68	5.55	5.89	5.92
0.75	0.95	1.19	2.61	2.72	3.02	3.15	3.18
0.80	0.65	52.85	83.38	109.70	132.34	130.26	110.83
0.80	0.70	23.04	32.40	38.59	42.76	44.60	44.60
0.80	0.75	11.14	17.38	20.75	23.92	25.89	26.11
0.80	0.80	7.54	9.96	11.67	13.56	14.52	15.00
0.80	0.85	4.18	6.71	7.87	9.35	9.62	9.62
0.80	0.90	3.56	3.91	5.12	5.91	6.17	6.20
0.80	0.95	2.24	3.01	3.34	3.70	3.80	3.85
0.85	0.65	44.47	74.50	97.24	113.82	115.44	107.41
0.85	0.70	20.27	32.10	36.79	43.72	46.56	46.95
0.85	0.75	11.79	17.33	20.25	23.83	25.04	25.04
0.85	0.80	7.75	10.72	12.45	14.76	15.22	15.24
0.85	0.85	4.43	6.27	7.71	8.94	9.70	9.70
0.85	0.90	3.79	4.01	5.25	5.86	6.19	6.20
0.85	0.95	2.81	3.48	3.58	4.02	4.16	4.20
0.90	0.65	42.34	62.10	77.43	93.20	99.16	96.48
0.90	0.70	18.22	27.27	30.88	36.54	40.26	40.53
0.90	0.75	10.47	15.42	18.25	20.82	21.80	21.80
0.90	0.80	8.19	10.53	12.16	14.36	14.83	15.04
0.90	0.85	5.45	7.05	8.02	9.05	9.52	9.52
0.90	0.90	4.06	5.02	5.69	6.32	6.49	6.53
0.90	0.95	3.86	4.13	4.15	4.58	4.73	4.73
0.95	0.65	34.83	49.94	59.38	74.51	77.19	76.15
0.95	0.70	18.94	25.08	28.07	32.58	34.45	35.48
0.95	0.75	12.82	16.51	18.78	20.98	21.76	21.76
0.95	0.80	9.42	11.48	13.05	14.56	15.26	15.35
0.95	0.85	7.21	8.21	9.77	10.89	11.07	11.07
0.95	0.90	5.86	7.87	8.40	9.54	9.86	9.95

0.95	0.95	6.21	6.24	6.88	7.33	7.42	7.54

Table A.7: FD-TE test: difference between the expected switching time and the sample in which the switching takes place. 1000 Montecarlo runs. No false alarms. Lower is better.

Bibliography

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," **Computer networks**, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] "Council Recommendation (EU) 2015/1184 of 14 July 2015 on broad guidelines for the economic policies of the Member States and of the European Union," Jul. 2015. [Online]. Available: http://data.europa.eu/eli/reco/2015/1184/oj

[3] Eurostat, "Smarter, greener, more inclusive? INDICATORS TO SUPPORT THE EUROPE 2020 STRATEGY," 2016. [Online]. Available: http://ec.europa.eu/eurostat/documents/3217494/7566774/KS-EZ-16-001-EN-N.pdf

[4] D. Yan, J. Xia, W. Tang, F. Song, X. Zhang, and Y. Jiang, "DeST — An integrated building simulation toolkit Part I: Fundamentals," **Building Simulation**, vol. 1, no. 2, pp. 95–110, jun 2008.

[5] T. D. Baenziger, "Energy management in lighting systems," in **Industry Applications Conference, 2000. Conference Record of the 2000 IEEE**, vol. 5. IEEE, 2000, pp. 3452–3459.

[6] M. Miyata, H. Yoshida, M. Asada, T. Iwata, Y. Tanabe, and T. Yanagisawa, "Estimation of energy baseline by simulation for on-going commissioning and energy saving retrofit," 2006.

[7] C. de Bakker, M. Aries, H. Kort, and A. Rosemann, "Occupancy-based lighting control in open-plan office spaces: A state-of-the-art review," **Building and Environment**, vol. 112, pp. 308–321, 2017.

[8] F. Rubinstein, D. Avery, J. Jennings, and S. Blanc, "On the calibration and commissioning of lighting controls," in **Proceedings of the Right Light**, vol. 50, 1997, p. 207.

[9] J. Page, D. Robinson, N. Morel, and J.-L. Scartezzini, "A generalised stochastic model for the simulation of occupant presence," **Energy and Buildings**, vol. 40, no. 2, pp. 83–98, jan 2008.

[10] C. Sandels, J. Widén, and L. Nordström, "Simulating occupancy in office buildings with non-homogeneous Markov chains for demand response analysis," in **Power & Energy Society General Meeting, 2015 IEEE**. IEEE, 2015, pp. 1–5.

[11] D. Yan, W. O'Brien, T. Hong, X. Feng, H. B. Gunay, F. Tahmasebi, and A. Mahdavi, "Occupant behavior modeling for building performance simulation: Current state and future challenges," **Energy and Buildings**, vol. 107, pp. 264–278, 2015.

[12] V. L. Erickson, M. Á. Carreira-Perpiñán, and A. E. Cerpa, "Occupancy modeling and prediction for building energy management," **ACM Transactions on Sensor Networks (TOSN)**, vol. 10, no. 3, p. 42, 2014.

[13] A. Pandharipande and D. Caicedo, "Smart indoor lighting systems with luminaire-based sensing: A review of lighting control approaches," **Energy and Buildings**, vol. 104, pp. 369–377, 2015.

[14] A. Pandharipande and G. R. Newsham, "Lighting controls: Evolution and revolution," **Lighting Research & Technology**, vol. 50, no. 1, pp. 115–128, jan 2018.

[15] S. Nagarathinam, S. R. Iyer, A. Vasan, V. Sarangan, A. Sivasubramaniam **et al.**, "On the utility of occupancy sensing for managing hvac energy in large zones," in **Proceedings of the 2015 ACM Sixth International Conference on Future Energy Systems**. ACM, 2015, pp. 219–220.

[16] B. Dong and B. Andrews, "Sensor-based occupancy behavioral pattern recognition for energy and comfort management in intelligent buildings," in **Proceedings of building simulation**, 2009, pp. 1444–1451.

[17] D. D. Myron, E. R. Williams, C. C. Hardin, T. W. Woytek, and M. A. Stephens, "Occupancy sensor and method of operating same," United States of America patentus US5 640 143A, 1997. [Online]. Available: https://patents.google.com/patent/US5640143A

[18] A. Cenedese, M. Fabris, G. Michieletto, R. Antonello, and the CSD class of 2016-17, "Control System Design Lecture notes," Dec. 2017.

[19] E. Fornasini, "Catene di Markov," in **Appunti di teoria dei sistemi**. Padova: Edizioni Libreria Progetto, 2015, ch. 13. [Online]. Available: http://www.dei.unipd.it/~fornasini/13_2011_Catene%20di%20Markov13.pdf

[20] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," **IEEE transactions on Information Theory**, vol. 13, no. 2, pp. 260–269, 1967.

[21] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," **Proceedings of the IEEE**, vol. 77, no. 2, pp. 257–286, 1989.

[22] S. Tu, "Derivation of baum-welch algorithm for hidden markov models." [Online]. Available: http://people.csail.mit.edu/stephentu/writeups/hmm-baum-welch-derivation.pdf

[23] A. Wald, "Sequential tests of statistical hypotheses," **The annals of mathematical statistics**, vol. 16, no. 2, pp. 117–186, 1945.

[24] E. S. Page, "Continuous inspection schemes," **Biometrika**, vol. 41, no. 1/2, pp. 100–115, 1954.

[25] P. Granjon, "The CuSum algorithm-a small review," 2013.

[26] M. Basseville, I. V. Nikiforov **et al.**, **Detection of abrupt changes: theory and application**. Prentice Hall Englewood Cliffs, 1993, vol. 104.

[27] J. M. Chambers, W. S. Cleveland, P. A. Tukey, and B. Kleiner, **Graphical Methods for Data Analysis (Wadsworth & Brooks/Cole Statistics/Probability Series)**. Duxbury Press, 1983. [Online]. Available: https://www.amazon.com/Graphical-Analysis-Wadsworth-Statistics-Probability/dp/053498052X

[28] D. F. Williamson, R. A. Parker, and J. S. Kendrick, "The box plot: a simple visual method to interpret data," **Annals of internal medicine**, vol. 110, no. 11, pp. 916–921, 1989.

[29] L. R. Rabiner and B.-H. Juang, "An introduction to hidden Markov models," **IEEE assp magazine**, vol. 3, no. 1, pp. 4–16, 1986.

[30] S. H. Dandach, R. Carli, and F. Bullo, "Accuracy and decision time for decentralized implementations of the sequential probability ratio test," in **American Control Conference (ACC), 2010**. IEEE, 2010, pp. 2390–2395.

[31] C.-D. Fuh, "SPRT and CUSUM in hidden Markov models," **The Annals of Statistics**, vol. 31, no. 3, pp. 942–977, jun 2003.