Diploma Thesis

Analysing and Extending Model-free condition monitoring with confidence

submitted in satisfaction of the requirements for the degree of

Diplom-Ingenieur

of the TU Wien, Faculty of Electrical Engineering Engineering and Information Technology

---

Diplomarbeit

Analyse und Erweiterung des Model-free condition monitoring with confidence

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines

Diplom-Ingenieurs

eingereicht an der Technischen Universität Wien, Fakultät für Elektrotechnik und Informationstechnik

von

Daniel Schnöll, BSc

Matr.Nr.: 01525039

unter der Anleitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. **Thilo Sauter**

Institute of Computer Technology
Technische Universität Wien
Karlsplatz 13, 1040 Wien, Österreich

Wien, am 29.09.2021

Abstract

Monitoring systems can be used to check the condition of a system and determine if it needs maintenance. Due to complexity, many systems do not have models, which are usually required for monitoring. However, it is desirable to monitor them to maintain quality and improve lifetime. It is known that monitoring is possible with simple assumptions and minimal system knowledge, but without a system specific model. These assumptions rely on signal features. Improving feature extraction should make the process in general more reliable. This thesis focuses on feature extraction and interpretation. The signal feature considered here is a jump, and it should be possible to detect a non-instantaneous jump also in presence of a signal drift. The difference between a drift and a non-instantaneous jump is purely system dependent and cannot be defined a priori. Therefore, the number of samples representing the minimum duration of a signal state is defined. Strong changes that take less than half of this duration cause the state to change, which defines a jump. This thesis is based on a part of CCAM. CCAM is a context-aware monitoring system with confidences. It uses these confidences for a local clustering algorithm to define a system state. It was stripped of assumptions that would limit the usability, then analysed and extended upon, based on the analysis. The extensions remove drift in the immediate history of a state, increase the probability that a state is formed, and extract parameters at runtime. Tests have been performed and the removal of the limiting assumptions has significantly improved the behaviour for general signals. The extensions improve this even further. This was to be expected for signals using the noise distribution the extensions and the analysis were based on. However, as soon as more realistic signals are investigated, the performance degrades. This stems from parts of CCAM that are not modified in the scope of this thesis, and it is evident that further improvements are necessary. A test on industry data revealed that the extension for removing the drift inside a states history is insufficient. On the other hand, the runtime parameter extractor performed well even on industrial data, thereby reducing the required parameters to one. This is a first step towards and possibly into self-learning.

Kurzfassung

Die Überwachung von Prozessen ist ein weites Feld und dient zur Erkennung von Störungen oder der Notwendigkeit von vorbeugender Wartung. Normalerweise wird hierbei ein Modell des Prozesses oder zumindest ein tieferes Verständnis vorausgesetzt. Aufgrund der zunehmenden Komplexität ist eine detaillierte Modellierung jedoch oft schwer oder unmöglich. Es ist jedoch bekannt, dass ein System auch ohne ein Modell überwachbar ist. Hierbei werden Annahmen über das System getroffen und minimale Systeminformationen verwendet. Einer dieser Ansätze basiert auf Feature-Extraction, und die Qualität der Erkennung von charakteristischen Mustern in Prozessdaten hat einen entscheidenden Einfluss auf die Qualität der Prozessüberwachung. Die vorliegende Arbeit baut auf CCAM auf, einem Überwachungssystem welches auf lokalem Clustering basiert. Hierbei werden Confidence Functions verwendet, um einen Wert einem Cluster zuzuordnen. In der vorliegenden Arbeit wird versucht, die Erkennung eines Signal-Sprungs, welcher nicht abrupt sein muss, auch während eines starken Drifts auftreten darf und mit Rauschen überlagert ist, zu verbessern. Dazu wurden limitierende Annahmen von CCAM entfernt, um nachfolgend den entsprechenden Teil von CCAM zu analysieren. Diese Analyse ist zweifach, einerseits von einem mathematisch internen Aspekt und andererseits von einem statistischen externen Aspekt. Auf Basis dieser Analyse wurden mögliche Verbesserungen definiert und anhand unterschiedlicher Signaltypen systematisch getestet. Die Erweiterungen von CCAM wurden unter der Annahme von weißem Signalrauschen definiert und funktionieren in diesem Rahmen exzellent. Bei der Anwendung auf Prozesssignale, die der industriellen Realität näher sind, zeigte sich jedoch eine schlechtere Performance. Diese Limitierungen stammen großteils aus Teilbereichen von CCAM, die in der vorliegenden Arbeit nicht modifiziert wurden. Ein Test an einem echten Industriesignal zeigte insbesondere, dass der verwendete Algorithmus zur Entfernung einer lokalen Drift, welcher die Erkennung von Sprüngen erleichtern sollte, noch verbesserungsbedürftig ist. Im Gegensatz dazu hat die Extraktion von Parametern während der Laufzeit für alle getesteten Signale gut funktioniert. Diese Erweiterung ist ein erster Schritt in Richtung selbstlernender Systeme, die für praxisnahe modellfreie Prozessüberwachung von großer Wichtigkeit sind.

Contents

Acronyms

CCAM	The name used for CCAM as defined by [1] in chapter 5.
CCAMb	The name used for CCAM without the limiting assumptions in chapter 5.
FCoE	The name used for FCoE form chapter 2 in chapter 5.
FIR	A description used for filters, which defines their response to an impulse. FIR filters responds with a finite number of non-zero values to an impulse. Usually a FIR filters have a non-recursive form..
IIR	A description used for filters, which defines their response to an impulse. IIR filters responds with an infinite number of non-zero values to an impulse..
myCCAM	The name used for the improved algorithm in chapter 5.
RANSAC	An iterative algorithm to guess parameters for a defined model, e.g., a first order polynomial.

List of Symbols

$\mathcal{C}_{\text{coconf}}$	The co-confidence of the algorithm.
$\mathcal{C}_{\text{conf}}$	The confidence of the algorithm.
$\mathcal{C}_{\text{sacoconf}}$	A confidence defined by how representative samples are. It is meant for a sorted co-confidence.
$\mathcal{C}_{\text{scoconf}}$	The values of Δ_{norm} evaluated by the co-confidence function and sorted ascendingly.
$\mathcal{C}_{\text{saconf}}$	A confidence defined for how representative samples are. It is meant for a sorted confidence.
$\mathcal{C}_{\text{sconf}}$	The values of Δ_{norm} evaluated by the confidence function and sorted descendingly.
$\mathcal{C}_{\text{valcoconf}}$	The values of a difference evaluated by the co-confidence function.
$\mathcal{C}_{\text{valconf}}$	The values of a difference evaluated by the confidence function.
Δ	The difference between a new value and the history.
Δ_{LinAp}	The difference between a new value and the history. Bot aspects got their drift removed. The incline is calculated by using fitting a polynomial of degree one with least quadratic error.
Δ_{MeanDiff}	The difference between a new value and the history. Bot aspects got their drift removed. The incline is calculated by using the mean of discrete differences of the history.
Δ_{norm}	The "normalized" distance between a new value and the history.
ϵ	The normalized threshold of the single-point solution($\epsilon = \frac{x_t}{\sigma}$).

f_{diff}	A functions which takes the current state history and the new input value and calculates a drift removed difference.
$f_{fc_{\text{valcoconf}}}$	A functions which returns a functions, which converts values to a co-confidence.
$f_{fc_{\text{valconf}}}$	A functions which returns a functions, which converts values to a confidence.
$f_{\text{parameter}}$	A functions which takes the current state history and parameters for the confidence and co-confidence function and generates new parameters for the confidence and co-confidence functions.
fc_{sacoconf}	A co-confidence function defined by how representative samples up to a sample amount are. It is meant for a sorted co-confidence.
fc_{saconf}	A confidence function defined by how representative samples up to a sample amount are. It is meant for a sorted confidence.
$fc_{\text{valcoconf}}$	A function converting values to a co-confidence.
fc_{valconf}	A function converting values to a confidence.
γ	The normalized step height ($\gamma = \frac{\text{step height}}{\sigma}$).
$\mathcal{H}$	History of the step detection algorithm.
$\mathcal{H}_{\text{P}}$	A timely ordered set of past input values and if they were marked .
μ	The mean of the distribution which has been normalized to σ.
n	An array position of Δ.
n_{t}	A numeric position representing a specific solution for $fc_{\text{saconf}}(n) \geq fc_{\text{sacoconf}}(n)$.
P_{detected}	The likelihood that a that a jump was detected within s_{b}.

Φ	The integral over the normal distribution over normalized values($\Phi(x, \mu)$).
ψ	The normal distribution over normalized values($\psi(x, \mu)$).
P_{stay}	The likelihood that a that a state is kept.
s_{a}	The specified sample amount for the algorithm.
s_{b}	Sample amount accepted into state which are incorrect.
σ	The square root of the variance of the normal distribution (variance= σ^2).
$\mathcal{S}_{\text{SCCVC}}$	A set of ranges in $\mathbb{R}_{0+}$ for which $f_{\mathcal{C}_{\text{valconf}}}(x) \geq f_{\mathcal{C}_{\text{sacoconf}}}(n)$ is true for each n .
$\mathcal{S}_{\text{SCVCC}}$	A set of ranges in $\mathbb{R}_{0+}$ for which $f_{\mathcal{C}_{\text{saconf}}}(n) \geq f_{\mathcal{C}_{\text{valcoconf}}}(x)$ is true for each n .
$\mathcal{S}_{\text{step}}$	A step function.
$\mathcal{S}_{\text{triangle}}$	A triangle function.
θ	The normalized sample threshold of the single-point solution($\theta = \frac{n_{\text{t}}}{s_{\text{a}}}$ at first later $\theta = \frac{n_{\text{t}} - s_{\text{b}}}{s_{\text{a}}}$).
v	The new/next input value.
$\tilde{v}$	The to σ normalized new/input value($\tilde{v} = \frac{v}{\sigma}$).
x	A numeric value of the sorted array Δ at the position n.
x_{t}	A numeric value representing a specific solution for $f_{\mathcal{C}_{\text{valconf}}}(x) \geq f_{\mathcal{C}_{\text{valcoconf}}}(x)$.
ζ_{LinAp}	The calculated incline from the $\mathcal{H}$, using a QR-Factorization .
$\zeta_{\text{LinApDelay}}$	The incline of the Historic values calculated through QR-Factorisation, approximating a polynomial of first order with minimal quadratic error. It does not use all historic values rather the second and third quarter.

ζ_{MeanDiff}	The calculated incline from the $\mathcal{H}$, using mean of discrete differences.
$\zeta_{\text{MeanDiffDelay}}$	The incline of the Historic values calculated through discrete differences. It does not use all historic values rather the second and third quarter.
ζ_{SGolay}	The incline of the Historic values calculated through Savitzky-Golay derivative first order Finite Impulse Response (FIR) filter and taking the mean of the second and third quarter.

List of Figures

Chapter 1

Introduction

Monitoring systems can lead to actions taken, which improve longevity of the system and can help to maintain quality of the product. This is nothing new, we have done it for centuries. Early third century BC, Ktesibios of Alexandria invented, or at least is credited for inventing a float valve to regulate the level in water clocks [2]. A float valve, in technical terms is an analogue feedback loop regulating the water level of something, commonly a tank. A bit more recent, before the third industrial revolution, processes were under manual control, in other words humans got values displayed (Monitoring) and then action were taken to correct errors. Through the third industrial revolution many processes got automated, thereby improving production quantity and quality [3]. These forms of automation/control have one thing in common: there is a form of system model present. Be it the innate understanding by a human or explicitly mathematically/statistically defined model; But not all systems have models of them due to complexity. Truly getting rid of any form of model is unlikely, but pushing the creation of model into an automated process or creating a broadly applicable model with parameters which can be automatically detected might be the way to go. The first approach goes towards a neural network, while the second one more towards a general expert system. It should be noted that a general expert system might be a contradiction, as it would be an expert for a general, universally applicable system.
Neural networks have shown to be a good solution in complex tasks and through hardware advances they seem to be a possible approach to the problem. Figuratively speaking, they do not need a predefined model of a system as they incorporate it trough training. To go a bit more into detail they are fully mathematically defined algorithms, but it is often not fully understood how the solution is the desired solution. They are commonly referred to as black boxes because of that. They often get initialized with random parameters and then through training they try to approach local minima. This training process is an optimization process and neural networks themself can be put in the non-linear adaptive filtering category [4].

There are multiple approaches of training a neural network, but they usually have one big problem: they require massive quantities of data, but at the same time if bad data is used the overall performance of the neural network is worsened. There were many approaches to reduce the data amount of neural networks. One of the most prominent one is transfer learning. Simplified, data from other regions can be used to train a neural network or a pre-trained neural network can be used, which has shown to reduce data amount required significantly [5].
Building a general system is the same as making general assumptions about systems. A general assumption that usually holds true for systems is causality. A system should only change iff some form of input changes. An input can be anything from a defined input signal to changes in the environment. For this definition it is irrelevant if it is measured or unmeasured. When factoring in the environment the label context-aware usually given to a system. Context-aware and context-awareness are to the best of the authors knowledge not formally defined, however, many descriptions are used, varying in level of detail.
Context-awareness (in ubiquitous computing) seems [6][7] to have been introduced by Schilit in 1994, which states "[...] context-aware software adapts according to the location of use, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time." [8]. This could be adapted to: context-aware systems adapt or react to the environment or changes of it. Environment is intentionally used with its broad definition as anything might interact with the sensor/algorithm/system. Let us put the assumption of causality into a more tangible scenario, a continues/on demand water heater. Figure 1.1 displays the essential of a continues water heater. There is only one valve for the gas and no valve for the water. We shall focus on the energy input, let us say gas flow rate and it shall have a feedback loop measuring the temperature of the hot water. This example was chosen as there are multiple industrial systems which have a similar principle and at the same time a far higher complexity.

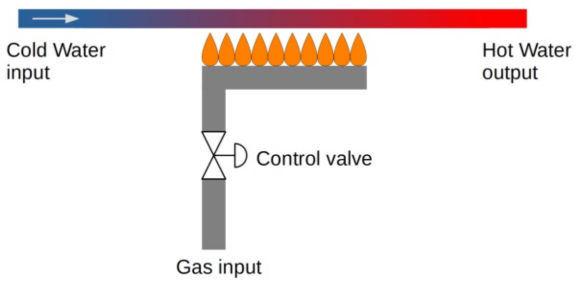


Fig. 1.1: The base principle behind a continues water heater.

If the fuel flow rate changes it could be to a number of reasons, some of them them are:

- target temperature was changed (direct, expected)
- warm water production has started/increased thereby requiring more energy to keep the temperature constant (indirect, expected)
- the ambient temperature has changed, the water input temperature has changed (indirect, explainable)
- part of the thermal isolation has broken off, the fuel line is leaking (indirect, unexpected)

A direct reason is quite easy to demine where it originates. If the set value changes the actuating variable has to change. On the other hand, an indirect reason is quite a bit harder to figure out. For the case of change in production, be it on/off/increase, the connection between the change in volume and the temperature is not essential for the function. It can function without it and not modelling it decreases the complexity and cost of the model and later for the build system; But having the complexity means a more consistent temperature when such a change happens and a higher grade of automation. But regardless of the grade of automation, there is a value that can be measured which is known to influence the temperature and therefore the fuel flow rate. Explainable changes can be measured and the physical model could be extended by corresponding parts, but most likely statistical data would be used to define a norm. The last aspect is the change of the physical object in an unexpected manner. Unexpected, as in they are not dependent on a realistically measurable value during production. Usually of destructive nature, e.g. wear and tear. Measuring or estimating wear and tear at runtime commonly requires highly complex models, be they mathematical or statistical in nature.

If no complex model is desired the base assumption of causality can still be used to define normal and abnormal behaviour. Using the whole systems information, it can easily be described as: if and only if an input changes an output is allowed to change within a certain time frame. This would only require the information of what is an input and what is an output, rather than a defined model of the system. The problem changes to what is a change in input/output.

Feature extraction can be used to feed such an algorithm instead of raw data, thereby abstracting the data. Figure 1.2 displays the base idea.

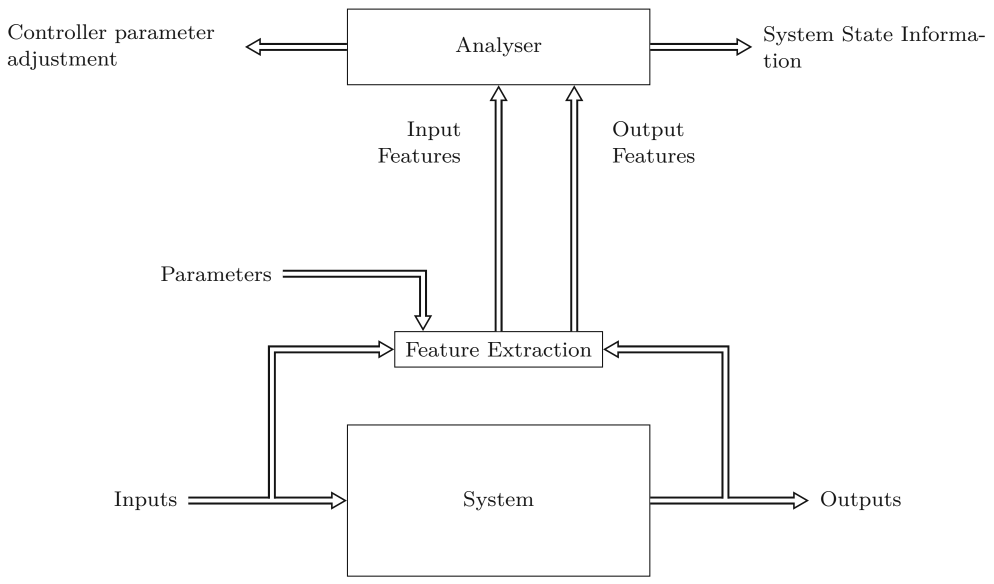


Fig. 1.2: A Feature extraction gathering data from all inputs and outputs of a system, which then could be analysed and used to control the system.

There are multiple possible features that can be extract from signals. For this thesis, jumps are considered relevant, but they by themself are an interpretation of a change-point detection. To detect a point-change is no miracle. Some of the existing algorithms are described in chapter 2. They can easily answer the question: has the point changed? To answer the question of how it has changed, the difference between jump and drift needs to be defined. To define a jump might not be seen as hard at first, as there is a mathematical definition for a jump, which also exists in the discrete realm. However, real systems usually do not exhibit a perfect, instantaneous jump. Figure 1.3 displays the step response of a simple system. The similarities with a jump are rather meagre. It has a slope and an overshoot. If the signal is measured in a finer granularity it could easily be mistaken for a drift. The answer to what is and what isn't, is purely system dependent. It should be noted that drifts and jumps can occur at the same time, therefore searching for zero degree "slopes" is not the answer. However it is discussed in chapter 2.

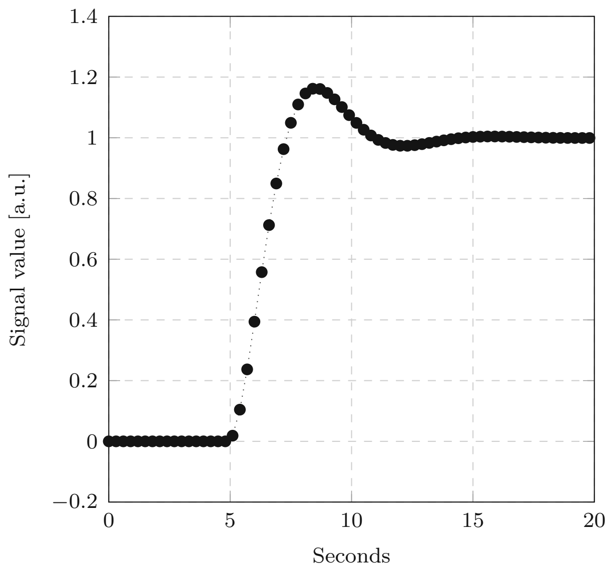


Fig. 1.3: Example of what should be considered a jump. It is the shifted step response of the system $\frac{1}{s^2+s+1}$ taken at specific timings.

By improving and possibly automating feature extraction, higher level algorithms could be empowered to produce better current machine state information and possible react to patterns, thereby possibly enabling pre-emptive actions to improve quality, or longevity by scheduling maintenance. Such a monitoring system could then be used to improve cheap systems, where process control would be to costly, while at the same time requiring less data than a neural network. A second possible use case would be systems where process control has been driven to its limits. If the complexity of the system exceeds the current possibilities of process control, a grey-box monitoring system might be the only viable approach.
As a side-note the goal is to enhance current systems and process control and not replace it. A fully modelled system will in all likelihood always be superior to such a grey-box approach, as it holds the true system information.

1.1 Scientific Problem and Contribution

Under the base assumption that in systems semi-stationary states exist, which are represented by semi constant values, frequencies, amplitudes, patterns, it can be argued that context is the reason why an output values changes when no input value has changed. An algorithm should be found which can interpret a point-change as a jump, even if a drift is simultaneously occurring. This algorithm shall be an online algorithm with little delay, as it is considered important to detect sudden changes fast. It should have stable parameters which should be able to be gathered from historic data. This

algorithm should also be resource efficient as it should be used on all inputs and outputs of an system. Computing the parameters from historic values is allowed to use an offline algorithm and resource efficiency has no major role.
The approach of this thesis is to remove assumptions that would limit a found algorithm with high potential (CCAM [1]). This is followed up by an analysis of the modified algorithm and afterwards, based on the analysis, extensions are formalized to improve the behaviour. The analysis is done from two different approaches: a mathematical internally defining approach and a statistical externally characterizing approach. Through the defining and characterising analysis improvements to the algorithm can be formalized, namely:

- Approximating the drift and removing it.
- Using dynamic fuzzy functions to improve the probability of forming a state.
- Using local information to extract fuzzy function parameters during runtime, even if non are provided initially. [1]

The improvements have a focus on resource efficiency and have a small individual analysis.
The base algorithm, the algorithm with the limiting assumptions and the fully adapted algorithm are at the end tested on multiple data sets, in order to experimentally verify the analysis and investigate the improvements and their cross-interaction. The data sets that are used are:

- Synthetic-Optimal:
  A data set that is created to show strengths and weaknesses of the of the algorithm and adaptations.
- Synthetic-Signal-Mimicking:
  A data set which was generated based on a statistical analysis of industrial data. The advantage is that it contains additional information that the real industrial data is missing.
- Industrial data:
  True industrial data, which has been anonymized.

[1]Providing no parameters is technically speaking not possible, so the initial parameters are set outside of a useful range.

1.2 Structure of the Thesis

The structure of the thesis is quite linear. Chapter 2 displays existing algorithms and makes a selection. They are categorized in offline and online algorithms. This categorisation has an importance as the final algorithm shall be an online algorithm. The offline category exists as some might be turned to online algorithms, or might be useful for extensions. In chapter 3 an existing algorithm was taken in its base form and adapted to suit the general nature of the problem. This adapted algorithm was then analysed from two viewpoints. The two viewpoints are a mathematical, what types of solutions exist and a statistical, how the algorithm behaves. In chapter 4 the analysis from the previous chapter is used to form further possible improvements for the algorithm and they are directly tested on datasets which are thought to show their strengths and weaknesses. That chapter concludes with a fusion of the adaptations. Chapter 5 tests the adaptation from chapter 3, the multiple adaptations from chapter 4 and the base algorithm on multiple datasets visualizing the benefits of the adaptations and their shortcomings. Chapter 6 concludes this thesis and describes possible further improvements.

Chapter 2

Related Work

In general two different types of algorithms exist online and offline. Online algorithms have a strict causal requirement while offline algorithms do not. The desired algorithm for the jump interpretation has to be an online algorithm, while the automatic/simplified parameter extractor is allowed to be an offline algorithm. Depending on the algorithm it might be possible to convert an offline algorithm to an online one by introducing a delay, but because of the temporal constrain on the desired algorithm it is unlikely. The main benefits of offline algorithms are that, they usually create better results than online ones as they have access to more information. The offline algorithm section mainly focuses on the parameter extractor while the online algorithm section focuses on the jump detector.

2.1 Offline Signal Algorithms

Offline Algorithms have a big scope. They reach from linear filters to signal reconstruction. Often there are hardly any temporal run-time constrains on the algorithm. However, there is a special niche where offline algorithms are used in an online fashion. This niche creates sets of data each time-step, like an image, they are still considered "offline" as there is no restriction on what data from the entire dataset is used, but at the same time very strict runtime constraints exist.
Iterative approaches are also commonly seen for offline algorithms. The desired signal which should be analysed is a piecewise semi constant signal. Jumps are allowed to be instantaneous or a transient process. These two conditions make the problem significantly harder. Ideally the signal should be decomposed to its base components, such as noise and individual sections separated by jumps. Such a decomposition is believed to prove useful when configuring a step detector.

2.1.1 Filter Approach

Matched/Correlation filters are the first looked upon. They are also usually found in the online algorithm domain, as a delay is all that is required to transform them. They are commonly used in very noisy environments when looking for specific patterns. They are also used as an offline algorithm inside image processing, in that scenario they are treated as pseudo offline as they do not move over the time axis, only over the image axes. They can reliably detect a pattern even with heavy noise. A step could be interpreted as a pattern and in fact there is of course a response which can be interpreted as a "found" pattern.
Normally, a pattern is considered to be found when a defined threshold value is surpassed and usually the intervals of patterns are also known. Both of these constraints cannot be universally defined for all signals, which might lead to the conclusion that the parameter extractor requires signal specific parameters itself. If such parameters are required and they are not easy to generate then the advantage of having the automatic/easy parameter extractor could be lost.

Image processing filters have for a long time concerned themself with edges, be it finding them or preserving them. Some of them are:

- median filter, removes outliers,
- bilateral filter, smooths while preserves edges,
- canny edge detection, uses a derivative and then thresholds, to define possible and definite edges, then iteratively defines used edges.

All of them are seen as acausal filters for the purpose of this thesis, with a defined filter width. Therefore they could be brought into the online domain by a delay. Median filters are likely going to be used as they are figuratively speaking low passes, which allow close to ideal jumps; And at the same time they do not require any signal information. A bilateral filter might also be interesting if signal information is present, as they smooth the signal in a linear fashion, but limit the information over which they smooth, in order to preserve edges. Such a filter might be useful to sharpen an edge, but they require a selecting algorithm, commonly a threshold, which would be dependent on signal information. Canny edge detection is similar in this regard, it would also require signal information to define edges; But the steps up to the this requirement might be useful.

2.1.2 Iterative Algorithms

First order spline fitting algorithm might also be an interesting approach with the assumption that jumps have at least two knots in close proximity. As an example, let us assume a jump from 0 to 1 over the course of 100 measurements, the overall amount of measurements is 2000. Let there be white Gaussian noise with a standard deviation of 0.1 . The spline shall be first order, in other words linear interpolation between spline points. A greedy search[1] algorithm is used and defined by the maximum of the absolute difference between spline and true signal. The inserted point shall be at the position of the maximum absolute difference and the value of the median of the 11 closest points. The median is only used to reduce the influence of the noise. Figure 2.1 displays the example with the first two iterations, as more are not needed. It should be noted that the displayed algorithm is by no means sophisticated. The two most relevant problems are that a termination condition is missing and that no automatic spline point merging exists to remove over-fitting.

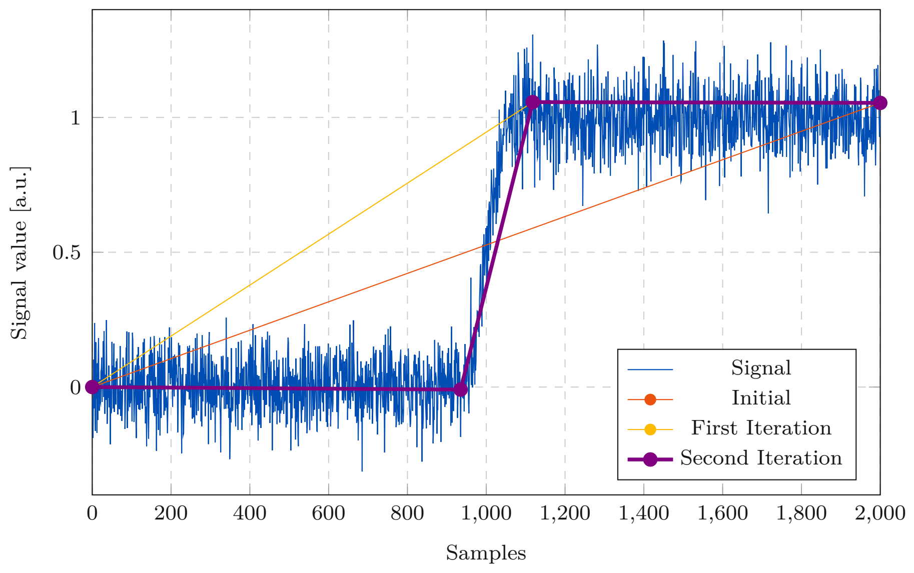


Fig. 2.1: Simple spline fitting algorithm demonstrated with iterations over a simple jump.

Piecewise constant signals(PWC) is a term which describes similar problems and a suit of algorithms [9]. Unsurprisingly there are algorithms which might solve the problem of this thesis, but it seems like they are offline algorithms rather than online ones which makes them only applicable for the parameter generator. The second problem is that some of them force the signal to be truly constant which contradicts the

[1]https://en.wikipedia.org/w/index.php?title=Greedy_algorithm&oldid=1028149539

assumption of this thesis. Even so, there might be algorithms which could be adapted to solve the problem.

Stepwise jump placement is an algorithm from the PWC suite which places jumps at likely locations and connects these with constant values [10]. It has a high similarity to spline fitting, but forces the signal to be truly constant between jumps. The jump positions are determined by a greedy search. There is also the inverse approach of defining each point as a jump and then removing the least likely one iteratively. The one major problem with it is that drifts can not be recreated with this algorithm. The algorithm might be adapted to handle them, but the question arises what the difference between this and a spline fitting algorithm would be.

0-degree Spline fitting was mentioned in [9], but no explicit paper was found. The base idea is quite interesting and might be modified to sharpen the jumps found by another algorithm. By redefining the "0-degree" condition to allowed slopes and only passing values of suspected jumps into the algorithm it might be possible to sharpen jumps. As an example let us reuse the jump from first order spline fitting. By running a RANdom SAmple Consensus (RANSAC)[2] algorithm for a Polynomial of zeroth order and using the mean of captured values as fitting. It should return the value and the position of first and last fitting value. By removing the found values from the signal, dominant offset values can be found iteratively. Figure 2.2 displays the example. As RANSAC needs an evaluation function a simple threshold of the distance was chosen with the value of 0.2 .

[2]https://en.wikipedia.org/w/index.php?title=Random_sample_consensus&oldid=983497063

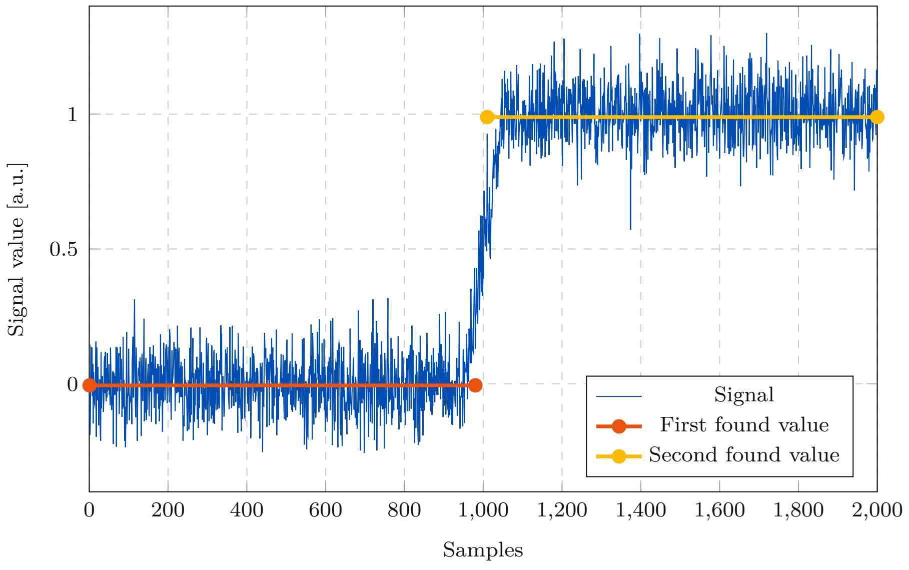


Fig. 2.2: Simple constant value fitting algorithm over a simple jump.

It is quite obvious that this algorithm could not solve the problem if a drift is present. By changing the the fitting function to use the surrounding values for a polynomial first order fitting and using an error function with linearly increasing error up to a point could alter the algorithm to accept drifts. Figure 2.3 displays the adapted example. It should be noted that the algorithm used to display is in the stage of explorative-prototype, it does not always produce the desired result. There are ways to remove unwanted results, such as using only connected segments of signal for error function evaluation. Connected segments in this case could mean that no more than 2 values in sequence are allowed to be outside the allowed distance to the spline.

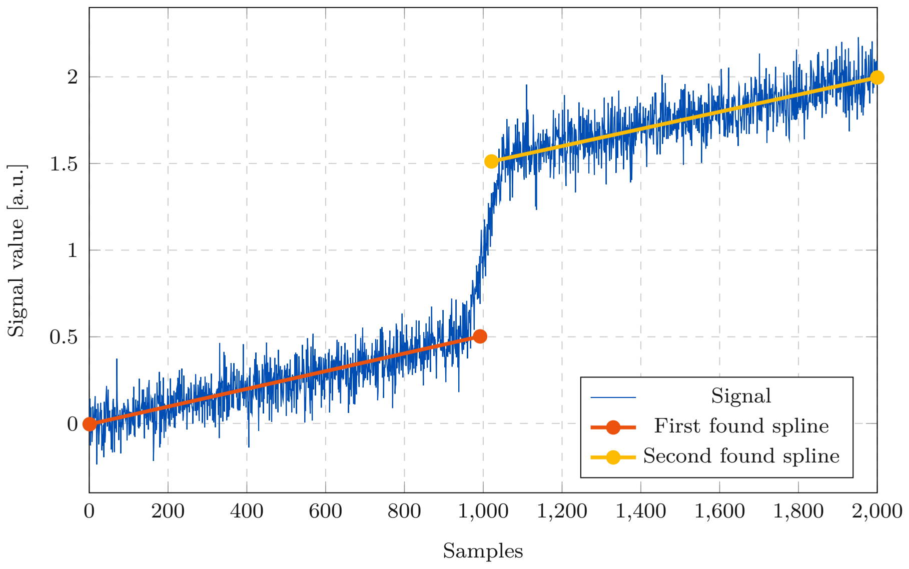


Fig. 2.3: Simple linear value fitting algorithm over a simple jump with drift.

A major problem with this approach is that it requires an evaluation function. Usually such an evaluation function is data specific. It would require this information from an overarching algorithm which would pass a suspected jump; Conversely, this also means that such an overarching algorithm has regions in which, in its opinion, no jump occurs, from which the required information might be obtained.

2.2 Online Signal Algorithms

Online algorithms are algorithms designed to process data during run-time. Some algorithms have strict delay requirements, especially ones that actively control a system. Delay in this context is the amount of measurements required past a current time-step in order to generate the response for this time step. Such delays are inherent to the algorithm and can not be solved by more processing power. As an example a linear causal filter has no delay as the algorithm produces the answer for that time-point by only using past information. Comparing that to a correlation filter, a delay of about half the order (that is the same as about half the size of the correlation reference) is usually present. Because the algorithm is intended to be used at some point as a part of actively controlling a system no inherent delay is wished. That the interpretation of the signal might have a delay due to the signal is probably unavoidable, but making it the only delay should make controlling easier and increase safety. Through this restriction matched/ correlation and median filters are unusable.

2.2.1 Commonly used Algorithms

CUSUM is an old and well researched approach [11][12]. It works on the base that a signal has a constant offset and noise. The integral over the noise should be zero. By subtracting the prior known offset, jumps and drifts can be detected. The detection is done by summing up the distances to the offset, with the twist that the sum can not get negative, if it would, it would be 0 instead. The same is done for the negative aspect to gain the ability to detect negative point-changes as well. This is a sequential method which is very resource efficient. It needs the expected mean value, and a threshold value. CUSUM through its integrating behaviour is very sensitive to slow changes and inaccuracies. These slow changes and inaccuracies can create a problem if CUSUM is running continuously. Such a problem could be solved by regularly resetting the algorithm. To reduce downtime, two CUSUMs could be run simultaneously and be reseted alternatingly. Another Problem with CUSUM is that it detects steps and drifts. Drifts should not be detected and the mean value of a signal is unknown. Both problems could be mitigated by using a sliding average filter for the mean but the question arises if this simple adaptation might destroy the advantages of the algorithm itself.

Shiryaev-Roberts procedures are mentioned usually as well when CUSUM is mentioned [13]. The base premises is that signal distributions are known. It seems to be exceptionally good in terms of average delay to detection. This trade is highly desired. It seems there are studies on variations on this algorithm which incorporate an unknown post change distribution [14].The problem with this algorithm is that the pre-jump distribution has to be known. If the drift would be known it could be compensated in the algorithm either prior to calculation or as a time dependent distribution. But the drift is unknown. Theoretically the more data is available the more computations the algorithm has to do, but as shown in [15] less data can be used to approximate a solution.

Exponentially weighted moving average (EWMA) is another common algorithm when talking about point-changes [16]. It is a very light weight highly effective Infinite Impulse Response (IIR) low-pass filter. Therefore, it would be easy and cost efficient to implement in hardware. It could be used with a threshold to detect point-changes. As normally with low-pass filters fast changes are not instantly visible. This alone could already disqualify EWMA, but it is very effective in determining the offset and low frequencies of a signal which could make it suitable to be used as the required offset information for CUSUM.

2.2.2 Simple Adaptations

CUSUM modified with EWMA might create an algorithm which could solve the problem. As these are modifications the true algorithm description is required. Let x_n be the new incoming value and O the predefined offset. CUSUM creates a negative and a positive sum of the differences to the predefined value,

$$\text{CUSUM}_{+,0} := 0, \tag{2.1}$$

$$\text{CUSUM}_{+,n} := max(\text{CUSUM}_{+,n-1} + x_n - O, 0), \tag{2.2}$$

$$\text{CUSUM}_{-,0} := 0, \tag{2.3}$$

$$\text{CUSUM}_{-,n} := min(\text{CUSUM}_{-,n-1} + x_n - O, 0). \tag{2.4}$$

$$\tag{2.5}$$

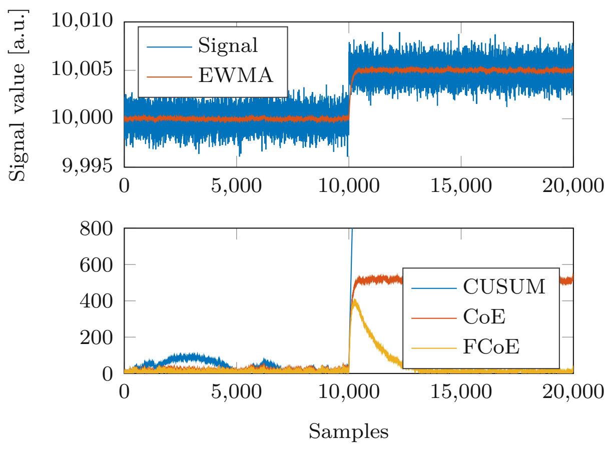
By modifying the CUSUM algorithm to use EWMA as offset O_n with the EWMA parameter p,

$$\begin{aligned}
\mathrm{O}_0 &:= x_0, && (2.6)\\
\mathrm{O}_n &:= (1-p)\cdot \mathrm{O}_{n-1} + p\cdot x_n, && (2.7)\\
\mathrm{CoE}_{+,0} &:= 0, && (2.8)\\
\mathrm{CoE}_{+,n} &:= max(\mathrm{CoE}_{+,n-1} + x_n - \mathrm{O}_n, 0), && (2.9)\\
\mathrm{CoE}_{-,0} &:= 0, && (2.10)\\
\mathrm{CoE}_{-,n} &:= min(\mathrm{CoE}_{-,n-1} + x_n - \mathrm{O}_n, 0), && (2.11)\\
& && (2.12)
\end{aligned}$$

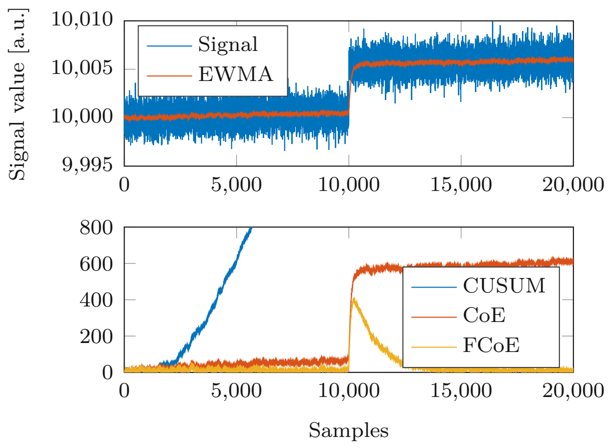
the offset does not need to be defined at the start. Such a modification is not fully drift resistant as there will be a delay between EWMA having the correct offset and CUSUM over EWMA (CoE) requiring it. By introducing a coefficient q into CoE to slowly forget the past,

$$\begin{aligned}
\mathrm{FCoE}_{+,0} &:= 0, && (2.13)\\
\mathrm{FCoE}_{+,n} &:= max(\mathrm{FCoE}_{+,n-1}\cdot q + x_n - \mathrm{O}_n, 0), && (2.14)\\
\mathrm{FCoE}_{-,0} &:= 0, && (2.15)\\
\mathrm{FCoE}_{-,n} &:= min(\mathrm{FCoE}_{-,n-1}\cdot q + x_n - \mathrm{O}_n, 0), && (2.16)\\
& && (2.17)
\end{aligned}$$

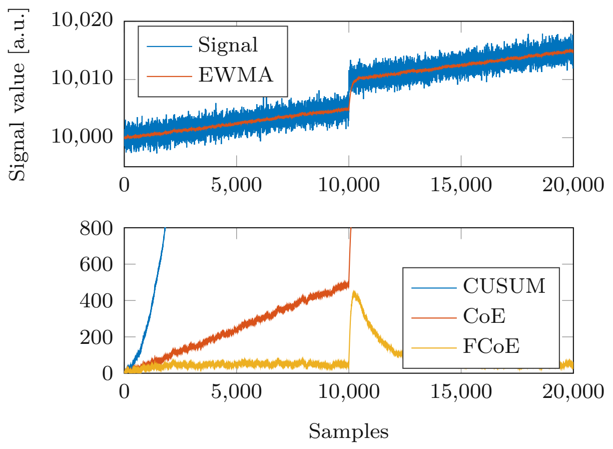
might fix the problem of the delay in offset. Figure 2.4 shows CUSUM in it base form and the two modified forms. Sub-figure a shows that all three could solve the problem with a simple threshold, as expected. Sub-figure b demonstrates the sensitivity of CUSUM, in this case undesired. It also visualizes the cumulative error of CUSUM over EWMA, this cumulative error is not as dominant as in sub-figure c, where it would invalidate a threshold approach. A forgetful CUSUM over EWMA (FCoE) could solve the problem in all three demonstrated cases.



(a) No drift



(b) A drift of 1 over the whole range

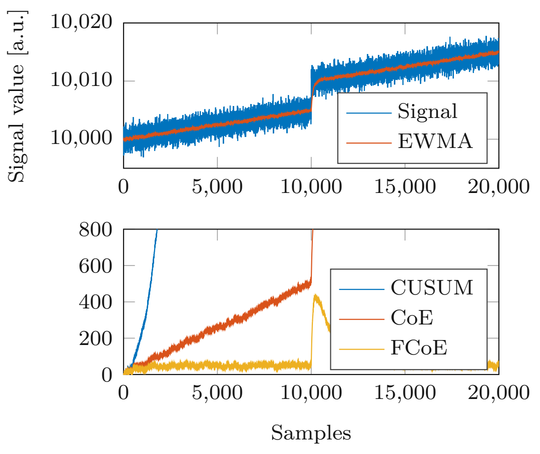


(c) A drift of 10 over the whole range

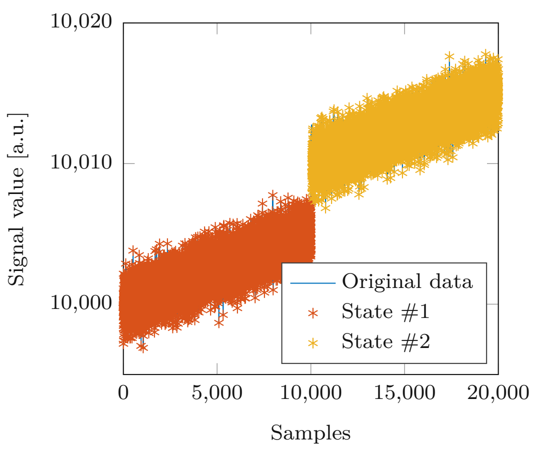
Fig. 2.4: CUSUM and two adaptations of it visualized with a drifting signal and a step.

CUSUM, Shiryaev-Roberts procedures and EWMA are algorithms designed to detect point-changes, not steps. This is an important differentiation. The source of the change is irrelevant for them. CUSUM and Shiryaev-Roberts procedures use the entire past through a quasi integrating behaviour. For them it is not a question of whether they detect the change but rather when. The integrating behaviour is the root of the sensitivity and why the detection is guaranteed. On the other hand a non-integrating behaviour would mean that drifts would not be recognisable and jumps might not be detected.

CCAM has a state detector that tries to cluster values into states as an online algorithm [1]. For this it uses a changing window with a maximal size and discrete differences between the incoming value and all values inside the window. If a step of significant change enters this window it might be detected. If it is detected it is likely that it is detected rather early. The algorithm only uses the window for local information to detect a step, or state change. Therefore, the longer the step is undetected the less likely it is to be recognized . This sounds highly negative, but the advantage is that by using only local information it is drift resistant by nature. If CCAM's state detector and the CUSUM and the prior introduced adaptations are used with a more extreme signal the advantage of CCAM's state detector become visible. Figure 2.5 and Figure 2.6 show the capabilities of CCAM, even when identical parameters are used for both signals. It should be noted that CUSUM could also solve the problem but would need specialized parameters for that. It should also be noted that due to the high parameter stability of CCAM acceptable parameters should be easy to find.

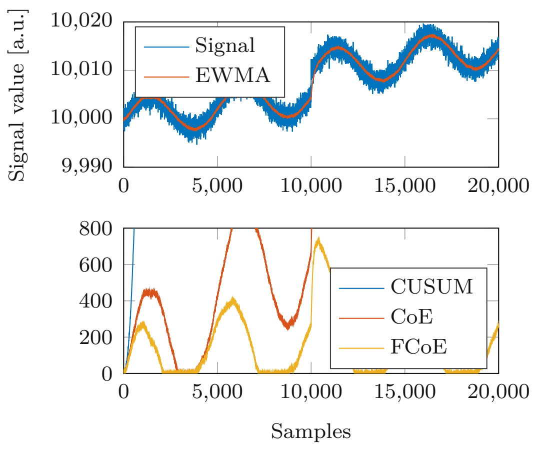


(a) CUSUM and adaptations

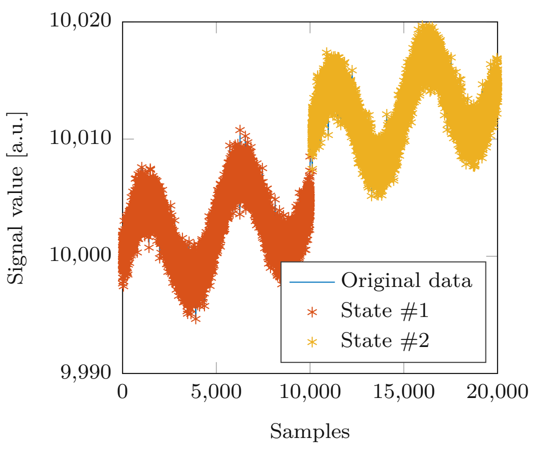


(b) CCAM's state detector

Fig. 2.5: CUSUM and adaptations, and CCAM's state detector visually compared, with a simple drift.



(a) CUSUM and adaptations



(b) CCAM's state detector

Fig. 2.6: CUSUM and adaptations, and CCAM's state detector visually compared, with a sine and a drift.

Chapter 3

Analysis of the Step Detector

CCAM is a context-aware algorithm which uses feature extraction as one of its methods to calculate a "health status" of a system [1]. Through discussions with the author of the original algorithm and supported up by [17], the author of this thesis was informed that the algorithm was developed and tuned for medical signals, such as heart rate, breath rate. Such signals usually are positive and stay in a specified range, e.g. the heart rate of a normal human stays in the range of 40-200 beats per minute, depending on multiple factors. CCAM has been tested and further improved to work in other areas than medical and it has shown potential. However, the assumptions for medical data remain and can easily lead to problems with the domain change. Removing these assumptions changes the algorithm in its behaviour.

In general the whole CCAM algorithm can be summarized with the following two questions:

- Was a step detected?
- If not, is it drifting?

If the first question is answered with yes, then it leaves the current state and it either finds an old fitting state or creates a new state. If it neither can find an old state nor create a new one for a prolonged time the system is considered "Broken". If it is in a state it is considered "OK" and if its drifting it is considered "Drift". These questions are asked for all signals simultaneously and a state represents multiple signals and not only one. The logic behind the algorithm can be seen in a more detailed manner in Figure 3.1.

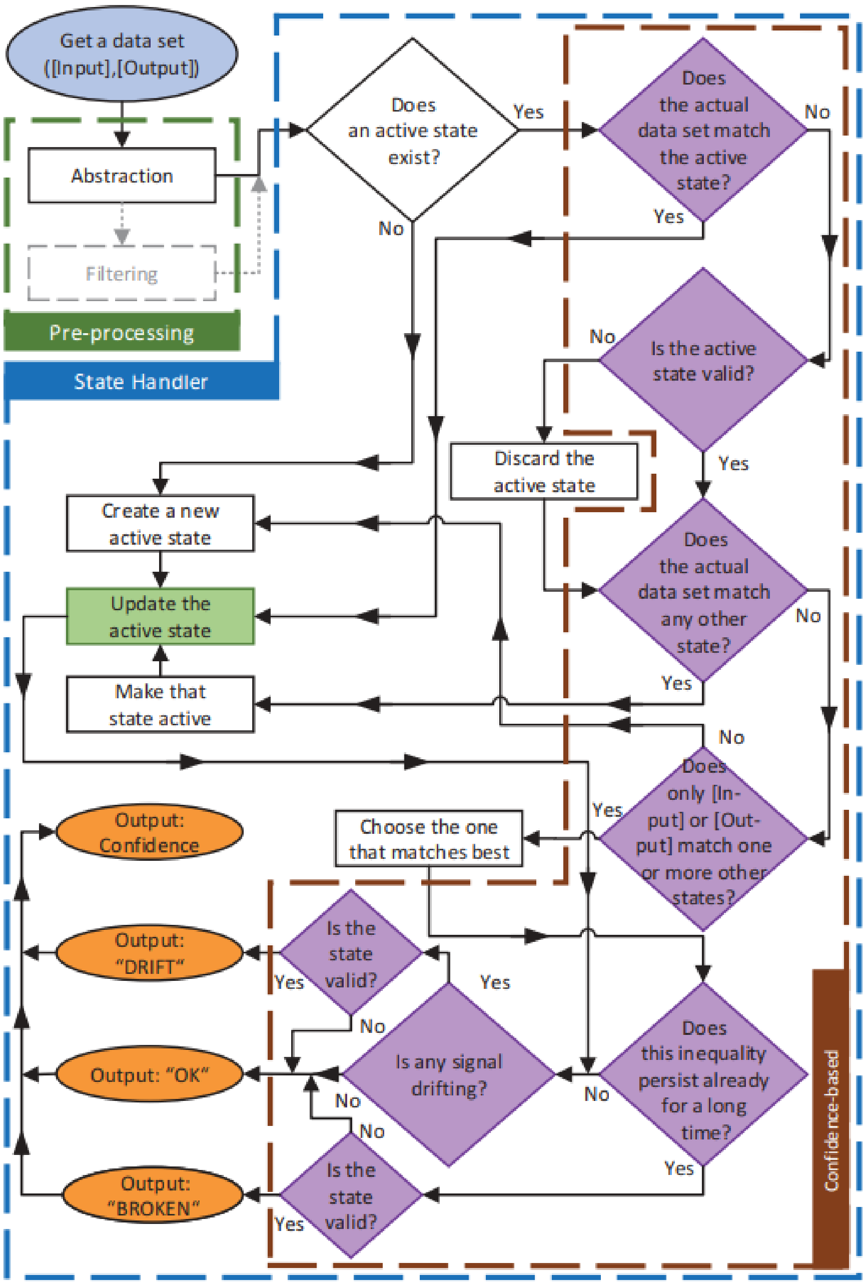

Get a data set ([Input],[Output])
Abstraction
Filtering
Pre-processing
State Handler
Does an active state exist?
Yes
No
Does the actual data set match the active state?
No
Yes
Is the active state valid?
No
Yes
Discard the active state
Does the actual data set match any other state?
No
Yes
Create a new active state
Update the active state
Make that state active
No
Does only [Input] or [Output] match one or more other states?
Yes
Choose the one that matches best
Output: Confidence
Output: "DRIFT"
Output: "OK"
Output: "BROKEN"
Is the state valid?
Yes
No
Is any signal drifting?
Yes
No
Does this inequality persist already for a long time?
No
Yes
Is the state valid?
Yes
No
Confidence-based

Fig. 3.1: Flow chart of the CCAM system as proposed in [1].

Simplified explained the step detector is a a clustering algorithm, that means it tries to assign an input to a "cluster". In this case a cluster is called state. A state is defined by a number of recently added values to that state. It assigns a value to a state depending on a confidence, which in turn uses the distance to the other values in that state. A state is founded by the first value assigned to it. This happens when there is no old state fitting for a value. Afterwards the cluster is extended by each assigned value up to a prior specified length. Once this length is reached the new values pushes out the oldest value. How it exactly works is described in the following section.

3.1 Original Algorithm

This section describes the step detector as defined by [1]. A custom nomenclature is used to improve readability, as a lot of symbols or alter forms of them are used throughout the thesis. However, they are connected to the original algorithm as described by Table 3.1.

3.1.1 Definition

The algorithm has one new input value v_i. The subscript i is which signal of an array of signals is used.
The algorithm has a history of the past values in the current state. This will be called $\mathcal{H}_{i,j}$. The subscript j is which point in the history is used.
The first step of the algorithm is to calculate the normalized distance $\Delta_{\text{norm},i,j}$ over v_i to $\mathcal{H}_{i,j}$ and then to "normalize" it to v_i,

$$\Delta_{\text{norm},i,j} = \left| \frac{v_i - \mathcal{H}_{i,j}}{v_i} \right| . \tag{3.1}$$

Then a confidence and co-confidence function need to be defined. The used parameters for them are $d_\text{a}, d_\text{b}, d_\text{c}, d_\text{d}$. The second step is to have $\Delta_{\text{norm},i,j}$ evaluated by the confidence function and label the result as $\mathcal{C}_{\text{valconf},i,j}$, and evaluated by the co-confidence function and label the result as $\mathcal{C}_{\text{valcoconf},i,j}$,

$$\mathcal{C}_{\text{valconf},i,j} = \begin{pmatrix} \frac{d_\text{a} - \Delta_{\text{norm},i,j}}{d_\text{a} - d_\text{b}} & : d_\text{a} < \Delta_{\text{norm},i,j} < d_\text{b} \\ 1 & : d_\text{b} \leq \Delta_{\text{norm},i,j} \leq d_\text{c} \\ \frac{d_\text{d} - \Delta_{\text{norm},i,j}}{d_\text{d} - d_\text{c}} & : d_\text{c} < \Delta_{\text{norm},i,j} < d_\text{d} \\ 0 & : \text{other} \end{pmatrix} , \tag{3.2}$$

$$\mathcal{C}_{\text{valcoconf},i,j} = \begin{pmatrix} \frac{\Delta_{\text{norm},i,j} - d_\text{b}}{d_\text{a} - d_\text{b}} & : d_\text{a} < \Delta_{\text{norm},i,j} < d_\text{b} \\ 0 & : d_\text{b} \leq \Delta_{\text{norm},i,j} \leq d_\text{c} \\ \frac{\Delta_{\text{norm},i,j} - d_\text{c}}{d_\text{d} - d_\text{c}} & : d_\text{c} < \Delta_{\text{norm},i,j} < d_\text{d} \\ 1 & : \text{other} \end{pmatrix} . \tag{3.3}$$

$\mathcal{C}_{\text{valconf},i,j}$ and $\mathcal{C}_{\text{valcoconf},i,j}$ are taken from [1] but were altered as there seems to be a typo in the source algorithm[1].

Next $\mathcal{C}_{\text{valconf},i,j}$ and $\mathcal{C}_{\text{valcoconf},i,j}$ need to be sorted over j. This is not explicitly stated, but implicitly by [1]. $\mathcal{C}_{\text{valconf},i,j}$ is sorted descendingly and $\mathcal{C}_{\text{valcoconf},i,j}$ is sorted ascendingly. These are called $\mathcal{C}_{\text{sconf},i,k}$ and $\mathcal{C}_{\text{scoconf},i,k}$ respectively. It should be noted that the subscript changed from j to k as it is now an amount and not a point in the history.

With these sorted values it needs to be decided how representative they are. For this purpose, confidences were defined over the sample amount for $\mathcal{C}_{\text{sconf},i,k}$ and $\mathcal{C}_{\text{scoconf},i,k}$. They are called $\mathcal{C}_{\text{saconf},i,k}$ and $\mathcal{C}_{\text{sacoconf},i,k}$,and are linearly increasing and decreasing respectively,

$$\mathcal{C}_{\text{saconf},i,k} = \left(\begin{array}{ll} 1 & : k \geq s_{\text{a}} \\ \frac{k}{s_{\text{a}}} & : 0 \leq k < s_{\text{a}} \end{array} \right) , \tag{3.4}$$

$$\mathcal{C}_{\text{sacoconf},i,k} = \left(\begin{array}{ll} 0 & : k \geq s_{\text{a}} \\ \frac{s_{\text{a}}-k}{s_{\text{a}}} & : 0 \leq k < s_{\text{a}} \end{array} \right) . \tag{3.5}$$

s_{a} is a specified sample amount.

Next $\mathcal{C}_{\text{sconf},i,k}$ and $\mathcal{C}_{\text{saconf},i,k}$ need to be combined. They are combined by using the "and" operation on $\mathcal{C}_{\text{sconf},i,k}$ up to k and then and-ing it with $\mathcal{C}_{\text{saconf},i,k}$. The same but with the "or" operator is done on $\mathcal{C}_{\text{scoconf},i,k}$ and $\mathcal{C}_{\text{sacoconf},i,k}$ respectively,

$$\mathcal{C}_{\text{conf},i,k} = \left(\bigwedge_{q=1}^{k} \mathcal{C}_{\text{sconf},i,q} \right) \wedge \mathcal{C}_{\text{saconf},i,k} , \tag{3.6}$$

$$\mathcal{C}_{\text{coconf},i,k} = \left(\bigvee_{q=1}^{k} \mathcal{C}_{\text{scoconf},i,q} \right) \vee \mathcal{C}_{\text{sacoconf},i,k} . \tag{3.7}$$

It should be noted that k is the number of samples and the index in an ordered array, as well as that the first sample would be index 1 and not 0.

Lastly, a state is held if any $\mathcal{C}_{\text{conf},i,k} \geq \mathcal{C}_{\text{coconf},i,k}$ over k. If a state is held $v_i$is pushed to the front of $\mathcal{H}_{i,j}$. If it is not held, old states are checked and entered or if none are found a new state is created.

The connection in nomenclature between this thesis and [1] is shown in Table 3.1.

[1]That it actually is a typo is confirmed when checking figure 1. in [1]

This Work Symbols	Symbols from [1]	Comments
s_{a}	s_a	
$d_{\mathrm{a}}, d_{\mathrm{b}}, d_{\mathrm{c}}, d_{\mathrm{d}}$	$d_{\mathrm{a}}, d_{\mathrm{b}}, d_{\mathrm{c}}, d_{\mathrm{c}}$	
v_i	$v_{i,new}$	
$\mathcal{H}_{i,j}$	$v_{h_{i,j}}$	
$\Delta_{\mathrm{norm},i,j}$	$d_{i,j}$	
$\mathcal{C}_{\mathrm{valconf},i,j}$	$c_{sv,i,j}$	
$\mathcal{C}_{\mathrm{valcoconf},i,j}$	$c_{dv,i,j}$	
$\mathcal{C}_{\mathrm{sconf},i,k}$	$c_{sv,i,k}$	Note the k instead of j
$\mathcal{C}_{\mathrm{scoconf},i,k}$	$c_{dv,i,k}$	Note the k instead of j
$\mathcal{C}_{\mathrm{saconf},i,k}$	$c_{ss,i,k}$	
$\mathcal{C}_{\mathrm{sacoconf},i,k}$	$c_{ds,i,k}$	
$\mathcal{C}_{\mathrm{conf},i,k}$	$c_{b,i}$	
$\mathcal{C}_{\mathrm{coconf},i,k}$	$c_{n,i}$	

Tab. 3.1: Connection of names from this work to [1].

3.1.2 Example

To give an example of this algorithm the values from Table 3.2 were chosen. Because the step detector is the desired component only one signal is used and the subscript i is ignored. Figure 3.2 visualizes the past input values which have been moved into $\mathcal{H}$. Figure 3.3 visualizes $\mathcal{C}_{\mathrm{valconf}}$, $\mathcal{C}_{\mathrm{valcoconf}}$, $\mathcal{C}_{\mathrm{saconf}}$ and $\mathcal{C}_{\mathrm{sacoconf}}$.

Name	Value
s_{a}	10
$d_{\mathrm{a}}, d_{\mathrm{b}}, d_{\mathrm{c}}, d_{\mathrm{d}}$	$[-8, -2, 2, 8]$
$\mathcal{H}$	$[4.6, 3.5, 1.9, 9.1, 7.4, 2.6, 5.9, 1.4, 1.1, 11.0]$
v	1

Tab. 3.2: Values for the example.

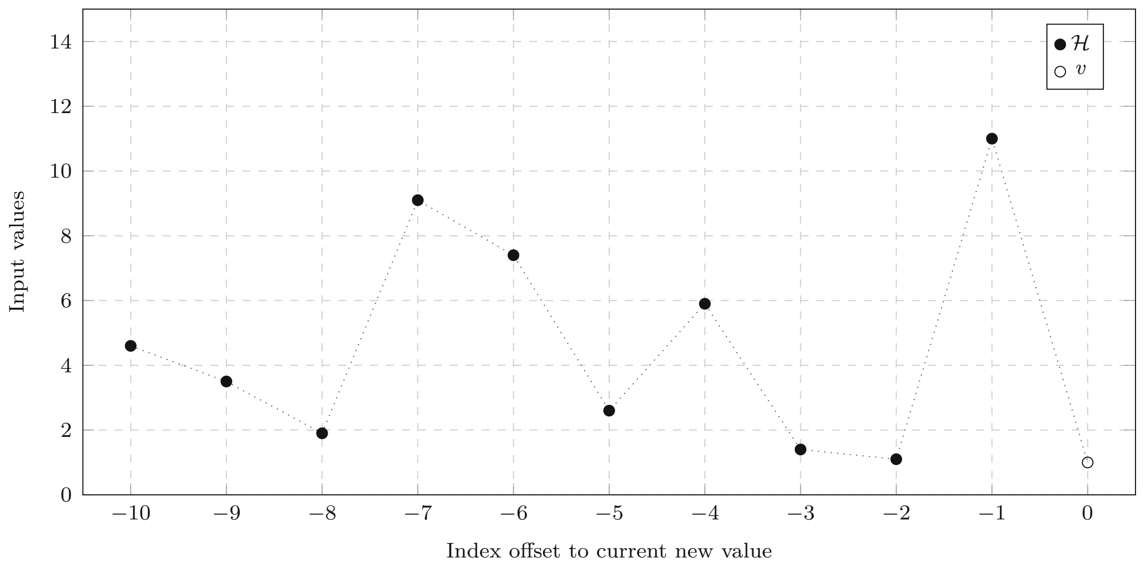
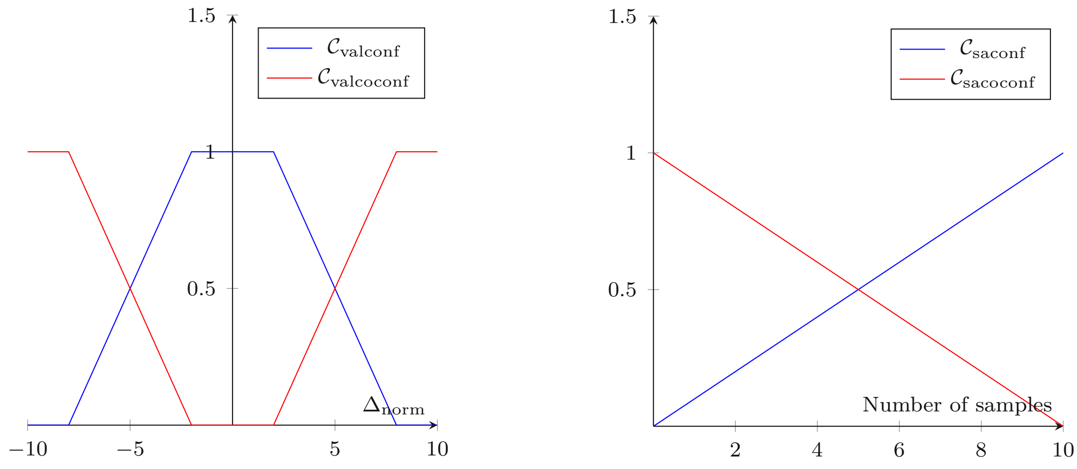


Fig. 3.2: The visual representation of the history and the new value.



(a) $\mathcal{C}_{\text{valconf}}$ and $\mathcal{C}_{\text{valcoconf}}$ **(b)** $\mathcal{C}_{\text{saconf}}$ and $\mathcal{C}_{\text{sacoconf}}$

Fig. 3.3: Visual representation of the defined functions.

Table 3.3 shows the resulting values for the current value. Because $\mathcal{C}_{\text{conf}} \geq \mathcal{C}_{\text{coconf}}$ has at least one true value, the state would be held, this is also shown by Figure 3.4 . It should be noted that $\mathcal{C}_{\text{saconf}}$ and $\mathcal{C}_{\text{sacoconf}}$ technically speaking have a value for $k = 0$, but it is unused by the algorithm so it is not displayed. It is only relevant if s_{a} is 0.

Value Name	Values										Comments
Δ_{norm}=	3.6	2.5	0.9	8.1	6.4	1.6	4.9	0.4	0.1	10.0	
$\mathcal{C}_{\text{valconf}}$=	0.7	0.9	1.0	0.0	0.3	1.0	0.5	1.0	1.0	0.0	not sorted
$\mathcal{C}_{\text{valcoconf}}$=	0.3	0.1	0.0	1.0	0.7	0.0	0.5	0.0	0.0	1.0	not sorted
$\mathcal{C}_{\text{valconf}}$=	1.0	1.0	1.0	1.0	0.9	0.7	0.5	0.3	0.0	0.0	sorted
$\mathcal{C}_{\text{valcoconf}}$=	0.0	0.0	0.0	0.0	0.1	0.3	0.5	0.7	1.0	1.0	sorted
$\mathcal{C}_{\text{saconf}}$=	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	
$\mathcal{C}_{\text{sacoconf}}$=	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	0.0	
$\mathcal{C}_{\text{conf}}$=	0.1	0.2	0.3	0.4	0.5	0.6	0.5	0.3	0.0	0.0	
$\mathcal{C}_{\text{coconf}}$=	0.9	0.8	0.7	0.6	0.5	0.4	0.5	0.7	1.0	1.0	
$\mathcal{C}_{\text{conf}} \geq \mathcal{C}_{\text{coconf}}$=	F	F	F	F	T	T	T	F	F	F	

Tab. 3.3: Calculated values from the example. F is for false and T is for true.

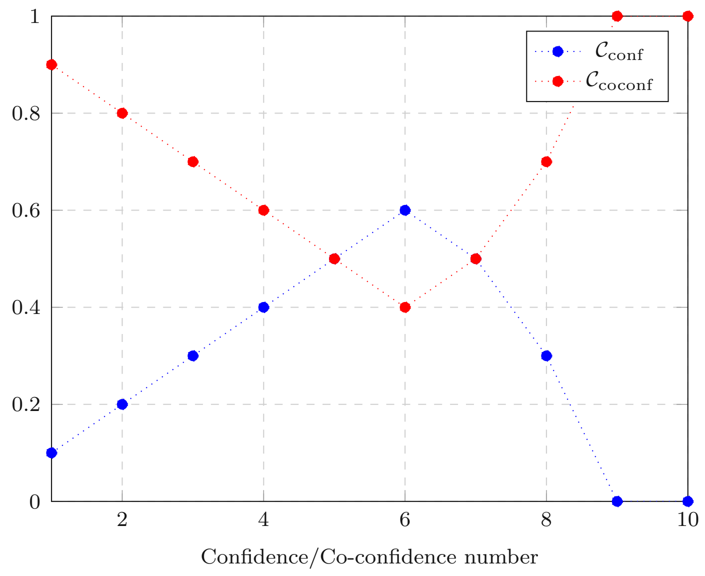


Fig. 3.4: Visual representation of $\mathcal{C}_{\text{conf}}$ and $\mathcal{C}_{\text{coconf}}$.

3.2 Necessary Adaptations

As already mentioned in the introduction to this chapter, CCAM and its predecessor were based on assumptions from the medical field [18]. These assumptions are believed to be that the noise is proportional to the offset, and the noise is small compared to the offset and must not reach values close to 0. Values inspected were such as heart-rate, breath-rate, temperature and so on. These assumptions likely lead to design derisions that will cause problems for general signals. These problems need to be addressed and corrected otherwise the algorithm will not work reliable with a general data set.

3.2.1 The Problems

The first problem is the normalization of the difference. If a signal would get 0 as a new value, Δ_{norm} would be undefined and if it would get a value close to 0, Δ_{norm} would be enormous. This would cause the algorithm to be unable to keep any state.

The second problem is a semi problem. The normalized distance or distance in general has only one sign. The problem is that $\mathcal{C}_{\text{valconf}}$ and $\mathcal{C}_{\text{valcoconf}}$ are defined over $\mathbb{R}$ but will only ever be used $\mathbb{R}_{0+}$. Everything below 0 will never be used. This could quite easily lead to misinterpreting the algorithm.

The third problem is a semi problem as well. $\mathcal{C}_{\text{sconf},i,j}$ is a sorted array, $\wedge$ in numerical terms is a minimum, therefore $\left(\bigwedge_{j=1}^{k} \mathcal{C}_{\text{sconf},i,j}\right)$ has to be at one side of the sub-array $j = 1...k$. Using $\wedge$ is good to demonstrate the thought behind it but might overcomplicate the the algorithm.

The fourth problem is that the equations chosen for $\mathcal{C}_{\text{valconf}}$ and $\mathcal{C}_{\text{valcoconf}}$ as well as $\mathcal{C}_{\text{saconf}}$ and $\mathcal{C}_{\text{sacoconf}}$ are suboptimal. To be more precise the relationships of $a = 1 - b$ for both pairs of functions. This relationship and a few constraints lead to transforming this algorithm from a fuzzy logic algorithm to a threshold algorithm for the state. This does not mean that a higher level algorithms would not get a probability for a state, only that the decision of "Is it in a state?" is a single threshold based decision. As this is a serious redefinition of the algorithms foundation a proof is provided in section 3.3.

The fifth problem is part of building and re-entering a state. If a value is not part of a state and no old state is found it creates a new state. Let us assume that the value is an outlier. By pure chance the following value is close enough to not leave that new state. A full state may be build because of an outlier. This would create a false state change.

3.2.2 Adaptation of the algorithm

The found solutions are used in the following sections, as this is accepted as the new base algorithm.

For the first problem redefining Δ_{norm} to a simple distance is enough,

$$\Delta = |v - \mathcal{H}| \; . \tag{3.8}$$

This also leads to the problem that each signal will need individual $\mathcal{C}_{\text{valconf}}$ & $\mathcal{C}_{\text{valcoconf}}$ functions. Therefore the index is removed and the signals decoupled.

For the second problem stating the function domains should be enough,

$$f_{\mathcal{C}_{\text{valconf}}} : \mathbb{R}_{0+} \longrightarrow [0,1] \,, \tag{3.9}$$
$$f_{\mathcal{C}_{\text{valcoconf}}} : \mathbb{R}_{0+} \longrightarrow [0,1] \,. \tag{3.10}$$

For the third problem replacing $\left(\bigwedge_{j=1}^{k} \mathcal{C}_{\text{sconf},j}\right)$ with $\mathcal{C}_{\text{sconf},k}$ is the solution as they are mathematically the same. It should be noted that the index i has been removed in accordance to the firsts problems solution.

For the fourth problem an approach is needed to define $\mathcal{C}_{\text{valconf}}$ and $\mathcal{C}_{\text{valcoconf}}$ as well as $\mathcal{C}_{\text{saconf}}$ and $\mathcal{C}_{\text{sacoconf}}$. For this the algorithm needs to be understood. There are two approaches, a mathematical one (see section 3.3) and a statistical one (see section 3.4). The author provides an example how they can be defined to solve this problem (see section 3.3.3.3), but it should be noted that this example is just a simple one and was not tested in any form.

For the fifth problem it is necessary to change how a state is build. By defining that a state has a purely temporary phase where it is actively compared to all other fully build states would force a state created by an outlier to be dropped. This decision shall be based on the highest confidence. If a true state change should happen it is assumed the the confidence of the values post-jump fit better into the newly created state than into the pre-jump state. Because this is resource demanding this temporary phase shall be $s_{\text{a}}/10$ long. This solution has also a downside, especially when pushing against the sensitivity limits of the algorithm itself; But in most cases it will only have a positive effect. For the negative effect to occur an old state has to exist and the pre-jump values have to be far to close to post-jump values.

3.3 Mathematical Analysis

The mathematical analysis aims to define the algorithms behaviour depending on the functions used for $\mathcal{C}_{\text{valconf}}$ and $\mathcal{C}_{\text{valcoconf}}$ as well as $\mathcal{C}_{\text{saconf}}$ and $\mathcal{C}_{\text{sacoconf}}$. For this constraints were defined as a generalization and simplification.

3.3.1 Definitions

The following definitions are used for the analysis.

Distance

$$\Delta = |v - \mathcal{H}| \tag{3.11}$$

The equation is identical to subsection 3.2.2, but for consistency shown.

Value Confidence

$$f_{\mathcal{C}_{\text{valconf}}} : \mathbb{R}_{0+} \longrightarrow [0,1] \tag{3.12}$$
$$f_{\mathcal{C}_{\text{valcoconf}}} : \mathbb{R}_{0+} \longrightarrow [0,1] \tag{3.13}$$

Again these quations are identical to subsection 3.2.2, but for consistency shown. Constraints are added for these two functions. $f_{\mathcal{C}_{\text{valconf}}}$ has to be decreasing, and $f_{\mathcal{C}_{\text{valcoconf}}}$ has to be increasing,

$$\forall x, y \in \mathbb{R}_{0+} : x < y \Rightarrow f_{\mathcal{C}_{\text{valconf}}}(x) \geq f_{\mathcal{C}_{\text{valconf}}}(y)\,, \tag{3.14}$$
$$\forall x, y \in \mathbb{R}_{0+} : x < y \Rightarrow f_{\mathcal{C}_{\text{valcoconf}}}(x) \leq f_{\mathcal{C}_{\text{valcoconf}}}(y)\,. \tag{3.15}$$

There are two reasons for these constraints. First, the the shown examples in [1] fulfil these constrains. Second, many noise distributions have some sort of maximum and from that maximum the likelihood decreases with distance.

Sample Confidence

The sample confidence and co confidence are also defined as increasing and decreasing, respectively,

$$f_{\mathcal{C}_{\text{saconf}}} : \mathbb{N}_0 \longrightarrow [0,1] \quad \forall x, y \in \mathbb{N}_0 : x < y \Rightarrow f_{\mathcal{C}_{\text{saconf}}}(x) \leq f_{\mathcal{C}_{\text{saconf}}}(y)\,, \tag{3.16}$$
$$f_{\mathcal{C}_{\text{sacoconf}}} : \mathbb{N}_0 \longrightarrow [0,1] \quad \forall x, y \in \mathbb{N}_0 : x < y \Rightarrow f_{\mathcal{C}_{\text{sacoconf}}}(x) \geq f_{\mathcal{C}_{\text{sacoconf}}}(y)\,. \tag{3.17}$$

Again the argument is made that the example in [1] is part of this group and that more samples can be trusted more than fewer samples.

Confidence Combination

$$\mathcal{C}_{\mathrm{conf},k} = \mathcal{C}_{\mathrm{sconf},k} \wedge \mathcal{C}_{\mathrm{saconf},k} \tag{3.18}$$

$$\mathcal{C}_{\mathrm{coconf},k} = \mathcal{C}_{\mathrm{scoconf},k} \vee \mathcal{C}_{\mathrm{sacoconf},k} \tag{3.19}$$

These equations behave identical to the original definition but are less computationally complex and give the opportunity to connect $\mathcal{C}_{\mathrm{sconf},k}$ and $\mathcal{C}_{\mathrm{saconf},k}$ as well as $\mathcal{C}_{\mathrm{scoconf},k}$ and $\mathcal{C}_{\mathrm{sacoconf},k}$ directly.

Order (Anti-)Isomorphism

Strictly increasing bijective functions are order isomorph. Respectively strictly decreasing bijective functions are order anti-isomorph. Order Isomorphism simplified means that if an ordered set is given to a function the resulting set is also ordered in the same "direction". Anti-Isomorphism would mean the "inverse direction".

3.3.2 Analysis

The analysis will take the following steps.

1. Show and prove that sorting prior to using the value confidence function is the same as sorting after the value confidence function. Thereby being able to connect the sample amount and true values. It should be noted that if it was allowed to sort the input rather than the output, it would be needed to sort the input in an ascending manner which would lead to $\mathcal{C}_{\mathrm{valconf}}$ being decreasing and $\mathcal{C}_{\mathrm{valcoconf}}$ ascending, which is exactly what is desired.

2. Show the origin of all possible solutions for keeping the state.

Sorting

The value confidence function is a surjective decreasing function over $\mathbb{R}_{0+}$. Let f be a surjective decreasing function. This function can be split up into two subfunctions, the first one has all injective parts the second one has the rest. Let $\mathcal{D}$ be the domain of a surjective decreasing function f. Let $\mathcal{D}_\mathrm{b}$ be the parts of $\mathcal{D}$ for which an inverse function from $f(\mathcal{D})$ exist, so the injective subdomain. Let $\mathcal{D}_\mathrm{s}$ be $\mathcal{D}\backslash\mathcal{D}_\mathrm{b}$,

$$\begin{aligned} f :&\mathcal{D} \longrightarrow f(\mathcal{D}), \\ f =& \left\{ \begin{array}{ll} f_\mathrm{b} & : \mathcal{D}_\mathrm{b} \longrightarrow f_\mathrm{b}(\mathcal{D}_\mathrm{b}), \exists f_\mathrm{b}^{-1} \\ f_\mathrm{s} & : \mathcal{D}_\mathrm{s} \longrightarrow f_\mathrm{s}(\mathcal{D}_\mathrm{s}) \end{array} \right| \begin{array}{c} \mathcal{D} = \mathcal{D}_\mathrm{b} \cup \mathcal{D}_\mathrm{s}, \mathcal{D}_\mathrm{b} \cap \mathcal{D}_\mathrm{s} = \emptyset, \\ f(\mathcal{D}_\mathrm{b}) \cap f(\mathcal{D}_\mathrm{s}) = \emptyset \end{array} . \end{aligned} \tag{3.20}$$

It should be noted that either part might be empty, but it does not change the analysis. The injective part is automatically bijective and decreasing, making it strictly decreasing. A strictly decreasing function is order anti-isomorph.
f_s is a decreasing surjective function which has no injective parts. In other words for each possible value in $f_\mathrm{s}(\mathcal{D}_\mathrm{s})$ multiple values in $\mathcal{D}_\mathrm{s}$ exist,

$$\forall x \in \mathcal{D}_\mathrm{s}, \exists y \in \mathcal{D}_\mathrm{s} : x \neq y, f_\mathrm{s}(x) = f_\mathrm{s}(y)\,, \tag{3.21}$$

because if a value only had one counterpart this value would be injective thus not be part of $\mathcal{D}_\mathrm{s}$.
$\mathcal{D}_\mathrm{s}$ can be split even further. Each sub range of $\mathcal{D}_\mathrm{s}$ which evaluates to the same value by f_s is placed in a corresponding $\mathcal{D}_{\mathrm{s},k}$ with a function $f_{\mathrm{s},k}$,

$$\forall k \in f_\mathrm{s}(\mathcal{D}_\mathrm{s}), \forall x \in \mathcal{D}_{\mathrm{s},k} : f_{\mathrm{s},k}(x) := k\,, \tag{3.22}$$

$$\mathcal{D}_\mathrm{s} = \bigcup_{k}^{f_\mathrm{s}(\mathcal{D}_\mathrm{s})} \mathcal{D}_{\mathrm{s},k}\,, \tag{3.23}$$

$$\forall a, b \in f_\mathrm{s}(\mathcal{D}_\mathrm{s}) : a \neq b \implies \mathcal{D}_{\mathrm{s},a} \cap \mathcal{D}_{\mathrm{s},b} = \emptyset\,. \tag{3.24}$$

Because f_s is decreasing there is an order to $\mathcal{D}_{\mathrm{s},k}$ over k. If $f_\mathrm{s}(a) < f_\mathrm{s}(b)$ then all elements of the set where a falls into, are greater then all elements of the set b falls into

$$a \in \mathcal{D}_{\mathrm{s},A}, b \in \mathcal{D}_{\mathrm{s},B} : f_\mathrm{s}(a) < f_\mathrm{s}(b) \implies \forall x \in \mathcal{D}_{\mathrm{s},A}, \forall y \in \mathcal{D}_{\mathrm{s},B} : x > y\,. \tag{3.25}$$

As f_s is decreasing an increasing set of $\mathcal{D}_{\mathrm{s},k}$ shall create a partly decreasing set consisting of one numerical value by applying $f_{\mathrm{s},k}$. Because k is one numerical value every permutation of $\mathcal{D}_{\mathrm{s},k}$ creates a partly ordered set. Therefore a strictly increasing set which creates a partly decreasing set exists.
The last part which is missing is to put f_s and f_b in relationship. By taking a value

from $\mathcal{D}_\mathrm{b}$ and from $\mathcal{D}_\mathrm{s}$ an order can be established by comparing these values once f is applied to it,

$$a \in \mathcal{D}_\mathrm{b}, b \in \mathcal{D}_\mathrm{s} : \begin{cases} f(a) > f(b) & \implies \quad a \leq b \\ f(a) < f(b) & \implies \quad a \geq b \\ f(a) = f(b) & \implies \quad a \in \mathcal{D}_\mathrm{s} \quad \text{impossible !} \end{cases} , \tag{3.26}$$

because $\mathcal{D}$ is split into $\mathcal{D}_\mathrm{b}$ and $\mathcal{D}_\mathrm{s}$, the values once f is applied can not be identical as that would contradict the split itself.
In conclusion an increasing set will create a partly ordered decreasing set when a decreasing surjective function is applied. Therefore a partly ordered increasing set will result in a partly ordered decreasing set, which is required in order to say that sorting is allowed to take place prior to applying the value confidence function.
The same steps can be taken for the value co confidence function, which is an increasing surjective function, therefore a partly ordered increasing set will result in a partly ordered increasing set.
This leads to the conclusion that both value based functions need an ascendingly sorted array as an input, in order to remove the sorting in a later step.

Possible Solutions

The condition to holding a state is that any $\mathcal{C}_\mathrm{conf}$ is greater or equal to $\mathcal{C}_\mathrm{coconf}$ for a specific k,

$$\exists k : \mathcal{C}_{\mathrm{conf},k} \geq \mathcal{C}_{\mathrm{coconf},k} \iff \text{keep state} . \tag{3.27}$$

By using the redefinitions for $\mathcal{C}_\mathrm{conf}$ and $\mathcal{C}_\mathrm{coconf}$ as defined by Equation 3.18 and Equation 3.19 (again shown for convenience) ,

$$\begin{aligned} \mathcal{C}_{\mathrm{conf},k} =& \mathcal{C}_{\mathrm{sconf},k} \wedge \mathcal{C}_{\mathrm{saconf},k} , \\ \mathcal{C}_{\mathrm{coconf},k} =& \mathcal{C}_{\mathrm{scoconf},k} \vee \mathcal{C}_{\mathrm{sacoconf},k} \end{aligned}$$

and inserting them into the condition, allows the condition to be split into 4 individual equations that $\mathcal{C}_\mathrm{scoconf}$ being greater or equal to $\mathcal{C}_\mathrm{scoconf}$ and $\mathcal{C}_\mathrm{sacoconf}$,

$$\mathcal{C}_{\mathrm{sconf},k} \geq \mathcal{C}_{\mathrm{scoconf},k} , \tag{3.28}$$

$$\mathcal{C}_{\mathrm{sconf},k} \geq \mathcal{C}_{\mathrm{sacoconf},k} \tag{3.29}$$

and $\mathcal{C}_{\text{saconf}}$ being greater or equal to $\mathcal{C}_{\text{scoconf}}$ and $\mathcal{C}_{\text{sacoconf}}$,

$$\mathcal{C}_{\text{saconf},k} \geq \mathcal{C}_{\text{scoconf},k}\,, \tag{3.30}$$

$$\mathcal{C}_{\text{saconf},k} \geq \mathcal{C}_{\text{sacoconf},k}\,. \tag{3.31}$$

This split is allowed because $\wedge$ is used on the greater side and $\vee$ on the smaller side. Therefore all 4 equations have to be fulfilled for the state to be kept. By using the previous prove and accepting that $f_{\mathcal{C}_{\text{valconf}}}$ and $f_{\mathcal{C}_{\text{valcoconf}}}$ take the same sorted Δ and are not sorted afterwards to create $\mathcal{C}_{\text{sconf}}$ and $\mathcal{C}_{\text{scoconf}}$, each mention of them can be replaced by their respective function, as they are not dependent on k but rather on the value in the k place of the array.
$\mathcal{C}_{\text{saconf}}$ and $\mathcal{C}_{\text{sacoconf}}$ can also be transformed into their respective functions. They are only dependent on k and not on any value.
In other words, the connection between k and the the value in the k place of the array is intentionally severed. Let $x \in \mathbb{R}_{0+}$ and $n \in \mathbb{N}$ then the following equations show this transformation,

$$f_{\mathcal{C}_{\text{valconf}}}(x) \geq f_{\mathcal{C}_{\text{valcoconf}}}(x)\,, \tag{3.32}$$

$$f_{\mathcal{C}_{\text{saconf}}}(n) \geq f_{\mathcal{C}_{\text{sacoconf}}}(n)\,, \tag{3.33}$$

$$f_{\mathcal{C}_{\text{saconf}}}(n) \geq f_{\mathcal{C}_{\text{valcoconf}}}(x)\,, \tag{3.34}$$

$$f_{\mathcal{C}_{\text{valconf}}}(x) \geq f_{\mathcal{C}_{\text{sacoconf}}}(n)\,. \tag{3.35}$$

The first two Equitations stay so to say in their respective domain, they compare a value confidence with a value co-confidence and a sample confidence with a sample co-confidence. This makes it easy to calculate a condition for x and n which needs to be fulfilled based on the functions. The solution for x is a range with either $[0, x_{\text{t}}]$ or $[0, x_{\text{t}})$, depending on the functions used and both part of $\mathbb{R}_{0+}$. It is also figuratively allowed for x to be infinite, which would just mean that the range would be $\mathbb{R}_{0+}$. For n the solution is also a range but takes the shape of $[n_{\text{t}}, s_{\text{a}}]$, part of $\mathbb{N}_{0+}$.
For the third and fourth equations the conditions are a little more difficult to define. It is basically mapping "$\mathbb{R}_{0+} \times [1, s_{\text{a}}] \longrightarrow$ Boolean". Because $[1, s_{\text{a}}]$ is a subset of $\mathbb{N}$ a

range can easily be numerically evaluated. This range is in $\mathbb{R}_{0+}$ for every n. For the third equation it is called $\mathcal{S}_{\mathrm{SCVCC}n}$,

$$\begin{aligned}
&\forall n \in [1, s_\mathrm{a}], \forall x \in \mathcal{S}_{\mathrm{SCVCC},n}, \\
&\quad \forall y \in \mathbb{R}_{0+} \backslash \mathcal{S}_{\mathrm{SCVCC},n} : \\
&\qquad y > x, \\
&\qquad f_{\mathcal{C}_\mathrm{valcoconf}}(x) \leq f_{\mathcal{C}_\mathrm{saconf}}(n), \\
&\qquad f_{\mathcal{C}_\mathrm{valcoconf}}(y) > f_{\mathcal{C}_\mathrm{saconf}}(n)
\end{aligned} \tag{3.36}$$

and for the fourth equation it is called $\mathcal{S}_{\mathrm{SCCVC}n}$,

$$\begin{aligned}
&\forall n \in [1, s_\mathrm{a}], \forall x \in \mathcal{S}_{\mathrm{SCCVC},n}, \\
&\quad \forall y \in \mathbb{R}_{0+} \backslash \mathcal{S}_{\mathrm{SCCVC},n} : \\
&\qquad y > x, \\
&\qquad f_{\mathcal{C}_\mathrm{valconf}}(x) \geq f_{\mathcal{C}_\mathrm{sacoconf}}(n), \\
&\qquad f_{\mathcal{C}_\mathrm{valconf}}(y) < f_{\mathcal{C}_\mathrm{sacoconf}}(n)
\end{aligned} \quad . \tag{3.37}$$

It should be noted that $\mathcal{S}_{\mathrm{SCVCC}n}$ is continues and if it is not empty it will start at 0. This is due to the fact that $f_{\mathcal{C}_\mathrm{valcoconf}}(x)$ is a increasing function and it has to be smaller or equal to $f_{\mathcal{C}_\mathrm{saconf}}(n)$.
$\mathcal{S}_{\mathrm{SCVCC}n}$ has another useful feature. Because $f_{\mathcal{C}_\mathrm{saconf}}(n)$ and $f_{\mathcal{C}_\mathrm{valcoconf}}(x)$ are increasing, the range $\mathcal{S}_{\mathrm{SCVCC},n}$ can only increase or stay the same when n increases,

$$\forall n \in [1, s_\mathrm{a} - 1] : \mathcal{S}_{\mathrm{SCVCC},n} \subseteq \mathcal{S}_{\mathrm{SCVCC},n+1} \ . \tag{3.38}$$

An equivalent argument can be made for $\mathcal{S}_{\mathrm{SCCVC},n}$, but with the difference that $f_{\mathcal{C}_\mathrm{valconf}}$ and $f_{\mathcal{C}_\mathrm{sacoconf}}$ are decreasing, and because both are decreasing it leads to the same conclusion that $\mathcal{S}_{\mathrm{SCCVC}n}$ can only increase or stay the same when n increases,

$$\forall n \in [1, s_\mathrm{a} - 1] : \mathcal{S}_{\mathrm{SCCVC},n} \subseteq \mathcal{S}_{\mathrm{SCCVC},n+1} \ . \tag{3.39}$$

By having defined conditions for these four equations the possible solutions can be defined. Table 3.4 summarizes the found mathematical contributions.

Equations	Constraint for keeping state
$fc_{\text{valconf}}(x) \geq fc_{\text{valcoconf}}(x)$	$x \in [0, x_t]$ or $x \in [0, x_t)$
$fc_{\text{saconf}}(n) \geq fc_{\text{sacoconf}}(n)$	$n \in [n_t, s_a]$
$fc_{\text{saconf}}(n) \geq fc_{\text{valcoconf}}(x)$	$x \in \mathcal{S}_{\text{SCVCC},n}$
$fc_{\text{valconf}}(x) \geq fc_{\text{sacoconf}}(n)$	$x \in \mathcal{S}_{\text{SCCVC},n}$

Tab. 3.4: The conditions and the equations they stem from that a state is kept.

The three possible solutions:

1. No Solution/ Trivial Solution. No solution exists therefore a state can never be kept.

2. Singe Point Solution. This solution is defined by x_t, n_t. Note that $\mathcal{S}_{\text{SCCVC}}$ and $\mathcal{S}_{\text{SCVCC}}$ could create a single point, but this would be treated as a special case of the multi point solution.

3. Multi Point Solution. This solution is shaped by $\mathcal{S}_{\text{SCCVC}}$ and $\mathcal{S}_{\text{SCVCC}}$. x_t,n_t also come into effect but they are not the main focus.

3.3.3 The three Solutions

This subsection analyses the three possible solutions. The analysis is done by interpreting the solution, showing a scenario or conditions so that a specific solution is created. If useful a graphical representation of the solution is shown.

3.3.3.1 No Solution/ Trivial Solution

This solution will never be able to keep a state. This can be achieved by fulfilling any of the following interpreted conditions.

x_t	The value confidence and co-confidence function never intersect and $\forall x \in \mathbb{R}_{0+} : f_{\mathcal{C}_{\text{valconf}}}(x) < f_{\mathcal{C}_{\text{valcoconf}}}(x)$, so it is never more likely that a signal is in a state rather than not in the "current" state. It should be noted that if a signal has no state the current value is used as the first value of a new state, therefore every value is put into a state even if that state is not kept.
n_t	The sample confidence and co-confidence function never intersect and $\forall n \in [1, s_a] : f_{\mathcal{C}_{\text{saconf}}}(n) < f_{\mathcal{C}_{\text{sacoconf}}}(n)$, so there is no number of samples at which it could be said that a value confidence could be trusted enough to keep a state.
$\mathcal{S}_{\text{SCCVC}}$	There is no possible value or sample amount that the value confidence would ever be more expressive than the trust that there are not enough samples.
$\mathcal{S}_{\text{SCVCC}}$	The trust in the sample amount is never enough to outweigh confidence that the values are not good enough.

3.3.3.2 Single Point Solution

The single point solution, which might also be referred to as threshold solution, is like the name suggests a single point in a two dimensional value space which defines the entire solution. This point is (n_t, x_t). It should be noted that $\mathcal{S}_{\text{SCCVC}}$ & $\mathcal{S}_{\text{SCVCC}}$ might collapse to a single point, this would be considered as a special case of the multipoint solutions and is therefore excluded.

There are two cases which have to be considered. The first case is that x_t is the intersections of value confidence and co-confidence function and therefore is part of the range. The second case is that x_t is defined by a discontinuous function and an not included boundary value has to be used for the range. This just has to be kept in mind. It is possible to argue that the non included case does not exist as the algorithm is implemented on a physical machine, therefore a boundary value can be found at the accuracy limit of the used data-type.

Through the characteristics of $\mathcal{S}_{\text{SCCVC}}$ and $\mathcal{S}_{\text{SCVCC}}$ it is easy to check if a solution falls into the category of single point solution. It has to be checked if the range defined with x_t is a part of the union between $\mathcal{S}_{\text{SCCVC}}$ and $\mathcal{S}_{\text{SCVCC}}$ at the n_t position,

$$[0, x_\mathrm{t}] \subseteq \{\mathcal{S}_{\mathrm{SCCVC},n_\mathrm{t}} \cap \mathcal{S}_{\mathrm{SCVCC},n_\mathrm{t}}\} \implies \text{single point solution}\,, \tag{3.40}$$

$$[0, x_\mathrm{t}) \subseteq \{\mathcal{S}_{\mathrm{SCCVC},n_\mathrm{t}} \cap \mathcal{S}_{\mathrm{SCVCC},n_\mathrm{t}}\} \implies \text{single point solution}\,, \tag{3.41}$$

of course depending on the case.

Figure 3.5 visualizes this single point solution. This visualization however does not incorporate $\mathcal{S}_\mathrm{SCCVC}$ and $\mathcal{S}_\mathrm{SCVCC}$ as they are not important for this type of solution. The sorted Δ_k can be plotted into this plot with $k = n$ and $x = \Delta_n$.

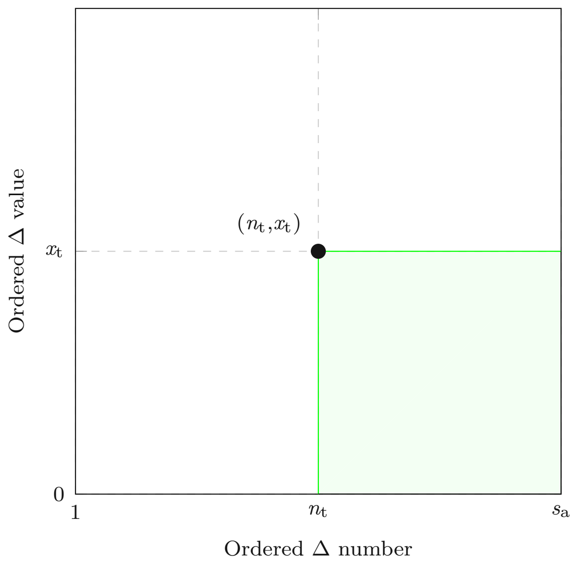


Fig. 3.5: Visual representation of the single point solution. If even one value falls into the green space it would keep the state.

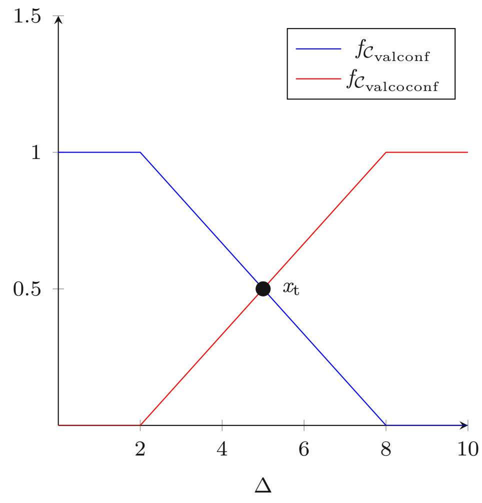
Example

Let us reuse the example from subsection 3.1.2 . Table 3.5 is an exact copy shown for convenience.

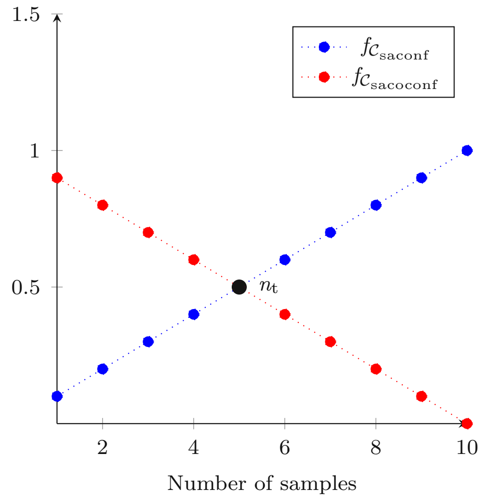
Name	Value
s_a	11
$d_\mathrm{a}, d_\mathrm{b}, d_\mathrm{c}, d_\mathrm{d}$	$[-8, -2, 2, 8]$
$\mathcal{H}$	$[1.1, 13.1, 4.6, 1.4, 9.1, 1.9, 11.0, 7.4, 2.6, 3.5, 5.9]$
v	1

Tab. 3.5: Values for the example. They are the identical values from subsection 3.1.2.

$f_{\mathcal{C}_\mathrm{valconf}}$ and $f_{\mathcal{C}_\mathrm{valcoconf}}$ are defined over $\mathbb{R}_{0+}$, therefore only the values of d_c, d_d are relevant. By searching the intersection of $f_{\mathcal{C}_\mathrm{valconf}}$ and $f_{\mathcal{C}_\mathrm{valcoconf}}$, x_t can be calculated and it evaluates to 5. The intersection of fc_{saconf} and fc_{sacoconf} evaluates to 5 as well. This is displayed in Figure 3.6.



(a) fc_{valconf} and $fc_{\text{valcoconf}}$



(b) fc_{saconf} and fc_{sacoconf}

Fig. 3.6: Visual representation of the defined functions and their intersections.

The last missing part is the sorted Δ. Table 3.6 shows the relevant and resulting values. These values would result in a plot as shown in Figure 3.7. In this example the state would be kept. It should also be noted that 3 values of Δ fulfil the criteria to hold the state, this is the same number as in subsection 3.1.2. The only difference is that the confidence is unknown and would need to be calculate separately.

Symbol	Value	Remarks
s_a	10	
n_t	5	
x_t	5	case $[0, x_t]$
Δ	$[0.1, 0.4, 0.9, 1.6, 2.5, 3.6, 4.9, 6.4, 8.1, 10.0]$	sorted

Tab. 3.6: Relevant calculated values for the example.

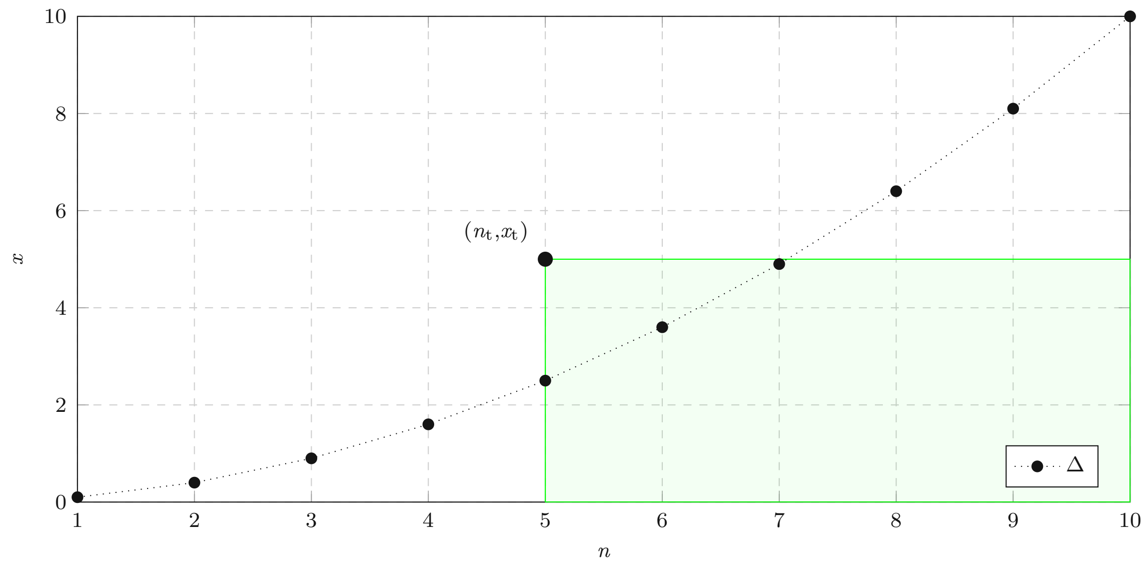


Fig. 3.7: A visual example of a single point solution and applied to a sorted Δ.

Problem with $a = 1 - b$

There is a problem with fc_{valconf} & $fc_{\text{valcoconf}}$ and fc_{saconf} & fc_{sacoconf} if they are defined in the manner $a = 1 - b$, with a, b being the confidence and co-confidence functions respectively. The problem is that the single point solution is the only possible solution if a non trivial solution exists. By applying this to the 4 conditional equations (3.32, 3.33, 3.34 and 3.35) the following four equations are logical consequences,

$$fc_{\text{valconf}}(x) \geq fc_{\text{valcoconf}}(x) \quad \Rightarrow \quad fc_{\text{valconf}}(x_t) \geq 0.5\,, \tag{3.42}$$

$$fc_{\text{saconf}}(n) \geq fc_{\text{sacoconf}}(n) \quad \Rightarrow \quad fc_{\text{saconf}}(n_t) \geq 0.5\,, \tag{3.43}$$

$$fc_{\text{saconf}}(n) \geq fc_{\text{valcoconf}}(x) \quad \Rightarrow \quad fc_{\text{saconf}}(n_t) \geq 1 - fc_{\text{valconf}}(x_t)\,, \tag{3.44}$$

$$fc_{\text{valconf}}(x) \geq fc_{\text{sacoconf}}(n) \quad \Rightarrow \quad fc_{\text{valconf}}(x_t) \geq 1 - fc_{\text{saconf}}(n_t)\,. \tag{3.45}$$

The last two equations are identical. By inserting the first two equations into the third one, an inequation is created. This is always true because the left side of the inequation is greater or equal to 0.5 and the right side is smaller or equal to 0.5 . The left side has to be greater or equal to the right side. Therefore it is a tautology.
Therefore if n_t and x_t exist and the defining functions are in the connection of $a = 1 - b$ it will always be a single point solution. The constraints of fc_{valconf}, $fc_{\text{valcoconf}}$, fc_{saconf} and fc_{sacoconf} should still be remembered.

3.3.3.3 Multi Point Solution

The multi point solution is mainly defined by $\mathcal{S}_{\text{SCCVC}}$ and $\mathcal{S}_{\text{SCVCC}}$. x_{t} and n_{t} still have a role of constraining the edges of the solution. As the name suggests it can be represented by multiple point in a two dimensional value space, which defines the entire solution. $\mathcal{S}_{\text{SCCVC}}$ and $\mathcal{S}_{\text{SCVCC}}$ represent the main conditions of this solution which are as following:

$\mathcal{S}_{\text{SCCVC}}$	Is a value x at the position n good enough for that position considering the trust-worthiness of that position in the sorted Δ?
$\mathcal{S}_{\text{SCVCC}}$	Is the value x at the position n not too bad for that position considering the trust-unworthiness of that position in the sorted Δ?

Both conditions need to be fulfilled for at least one n. They are figuratively asking the inverse questions of each other, but inverse in this case does not mean $a = 1 - b$, it is expressed by four independent functions. The two conditions from the point solution still apply so $x \in [0, x_{\text{t}}]$ / $x \in [0, x_{\text{t}})$ and $n \geq n_{\text{t}}$. This can be represented by a plot. An example is shown in Figure 3.8.

It can be simplified to a 0 to $x_{\text{t},n}$ range by using the characteristics of $\mathcal{S}_{\text{SCCVC},n}$ and $\mathcal{S}_{\text{SCVCC},n}$, the maximum of the conjunction of them and the range defined by x_{t},

$$x_{\text{t},n} := \max(\{[0, x_{\text{t}}] \cap \mathcal{S}_{\text{SCCVC},n} \cap \mathcal{S}_{\text{SCVCC},n}\})\,. \tag{3.46}$$

$$x_{\text{t},n} := \max(\{[0, x_{\text{t}}) \cap \mathcal{S}_{\text{SCCVC},n} \cap \mathcal{S}_{\text{SCVCC},n}\})\,. \tag{3.47}$$

Then a state is held if $n \geq n_{\text{t}}$ and $x_n \leq x_{\text{t},n}$. This simplification visually displayed is where the name comes from. The simplification is shown in Figure 3.9. It should be remembered that sample number is a discrete value. It should also be noted that this example was not generated by functions, it only exists to show the mechanism of this type of solution.

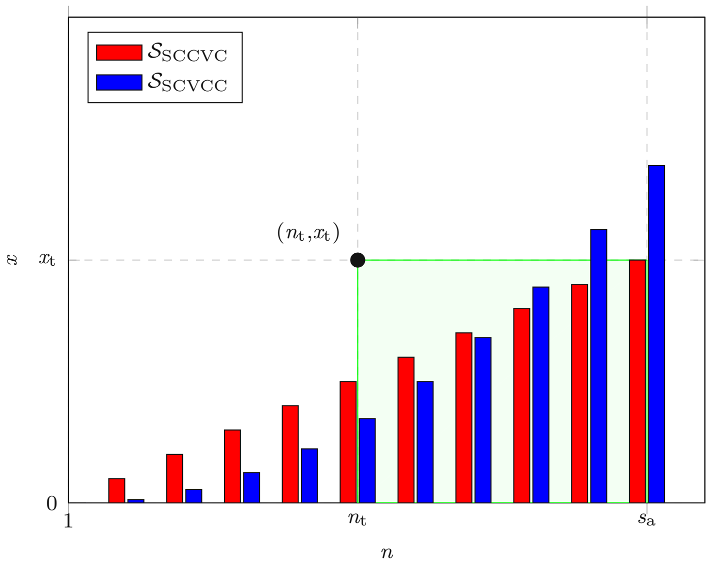


Fig. 3.8: Visual representation of the multi point solution. If even one value falls into the green space and is inside both bars it would keep the state.

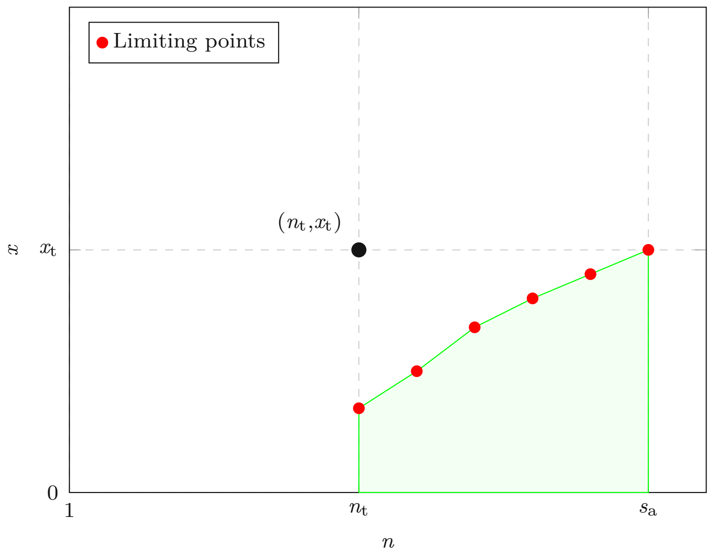


Fig. 3.9: Visual representation of the simplified multi point solution. If even one value falls into the green space it would keep the state.

Example

This example will construct a multipoint solution and not use one, as the usage is graphically identical to the single point solutions but with a differently shaped field. Table 3.7 show the desired values for the multipoint solution. Multi-points is what $x_{\text{t},n}$ shall follow inside of (n_t, x_t), it will shape $\{\mathcal{S}_{\text{SCCVC},n} \cap \mathcal{S}_{\text{SCVCC},n}\}$.

Name	Value/Function
s_a	10
x_t	10
n_t	3
multi-points	$n^2/5$

Tab. 3.7: Defining the desired values for the example.

To start of, any function must be defined, for this the author arbitrarily chooses $f_{\mathcal{C}_\text{sacoconf}}$ and it is defined as

$$f_{\mathcal{C}_\text{sacoconf}} := 1 - \frac{n}{s_\text{a}}\,. \tag{3.48}$$

Next $\mathcal{S}_\text{SCCVC}$ can be defined, by using the wished solution for the multi-points,

$$\begin{aligned} \forall n \in [1, s_\text{a}] :& \\ x_{\text{t},n} :=& n^2/5 \\ \mathcal{S}_{\text{SCCVC},n} :=& [0, x_{\text{t},n}] \end{aligned}\quad . \tag{3.49}$$

Because $f_{\mathcal{C}_\text{sacoconf}}$ and $\mathcal{S}_\text{SCCVC}$ are defined, $f_{\mathcal{C}_\text{valconf}}$ can be calculated. It should be noted that $f_{\mathcal{C}_\text{valconf}}$ will have discrete values and between those values the function is allowed to have any form as long as the constraint of decreasing is not broken. By having defined x and the resulting confidences these discrete positions can be calculated,

$$\begin{aligned} \forall n \in [1, s_\text{a}] :& \\ f_{\mathcal{C}_\text{valconf}}(x_{\text{t},n}) :=& f_{\mathcal{C}_\text{sacoconf}}(n) \\ & \Downarrow \\ f_{\mathcal{C}_\text{valconf}}(x_{\text{t},n}) :=& 1 - \frac{n}{s_\text{a}} \end{aligned}\quad . \tag{3.50}$$

A visual representation of $f_{\mathcal{C}_\text{valconf}}$ is shown in Figure 3.10.

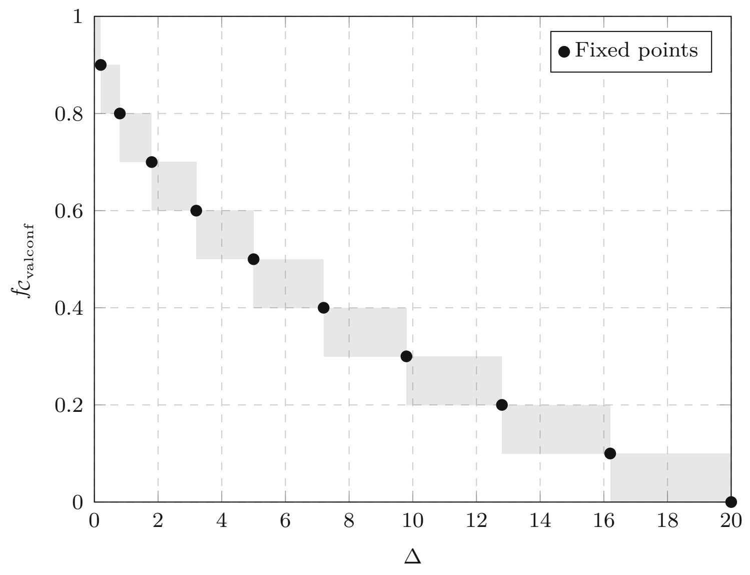


Fig. 3.10: Visual representation of the $f_{\mathcal{C}_{\text{valconf}}}$. The grey patches are the areas where the function has to be inside but no closer definition is given so far.

For the sake of simplicity of this example the author assumes n to be in $\mathbb{R}_+$ to calculate $f_{\mathcal{C}_{\text{valconf}}}$. This leads to $f_{\mathcal{C}_{\text{valconf}}}$ being defined continuously,

$$f_{\mathcal{C}_{\text{valconf}}}(x) := \begin{cases} 1 - \frac{\sqrt{x \cdot 5}}{s_{\text{a}}} & : 0 \leq x \leq 20 \\ 0 & : \text{else} \end{cases} \tag{3.51}$$

and visualised in Figure 3.11.

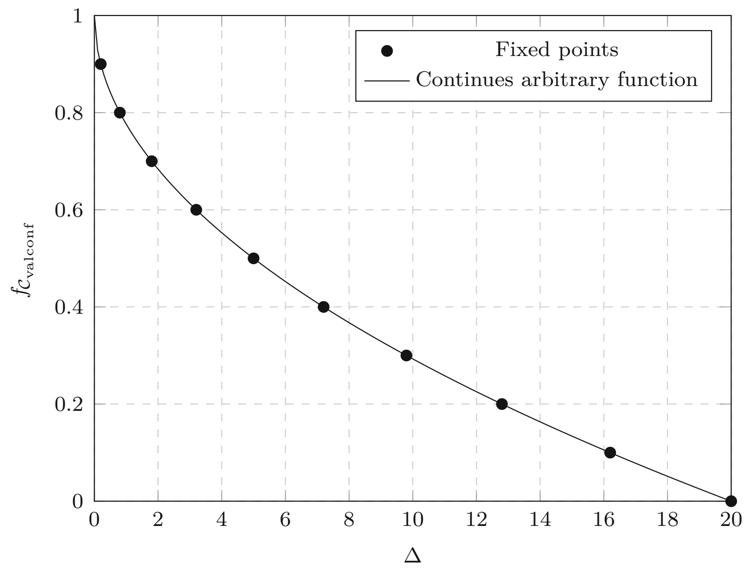


Fig. 3.11: Visual representation of the $f_{\mathcal{C}_{\text{valconf}}}$. The continues arbitrary function is an arbitrary function which fits the conditions.

With fc_{valconf} fully defined, $fc_{\text{valcoconf}}$ can be partly defined. $fc_{\text{valcoconf}}$ has to be smaller or equal $fc_{\text{valconf}}(x_t)$ up to x_t, at x_t it has to be $fc_{\text{valconf}}(x_t)$ and beyond x_t it has to be greater or equal to $fc_{\text{valconf}}(x_t)$, while of course being increasing as a function,

$$\forall x \in \mathbb{R}_+, x < x_t : fc_{\text{valconf}}(x) \geq fc_{\text{valcoconf}}(x)\,, \tag{3.52}$$
$$\forall x \in \mathbb{R}_+, x = x_t : fc_{\text{valconf}}(x) = fc_{\text{valcoconf}}(x)\,, \tag{3.53}$$
$$\forall x \in \mathbb{R}_+, x > x_t : fc_{\text{valconf}}(x) \leq fc_{\text{valcoconf}}(x)\,. \tag{3.54}$$

Again an arbitrary function is chosen which fulfils the requirements, it is defined as:

$$fc_{\text{valcoconf}}(x) := \begin{cases} fc_{\text{valconf}}(x_t) \cdot e^{\frac{x-10}{10}} & : 0 \leq x \leq 20 \\ 1 & : \text{else} \end{cases} \tag{3.55}$$

and both the conditions and chosen function are visualized in Figure 3.12.

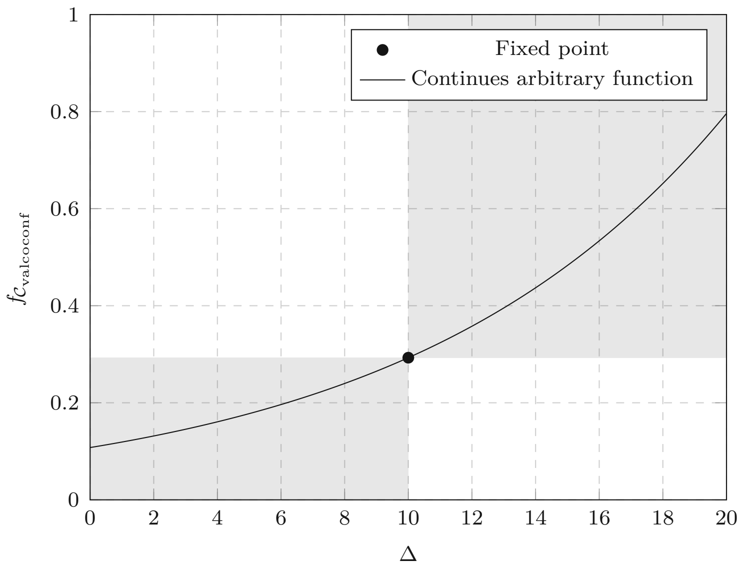


Fig. 3.12: Visual representation of the $fc_{\text{valcoconf}}$. The grey patches are the areas where the function has to be inside. The continues arbitrary function is an arbitrary function which fits the conditions.

The last function missing is fc_{saconf}. It is defined by fc_{sacoconf} and n_t. fc_{saconf} again has defined conditions. It has to be smaller or equal up to n_t, at n_t it has to be fc_{sacoconf} and after it has to be greater or equal to fc_{sacoconf},

$$\forall n \in [1, n_t - 1] : fc_{\text{saconf}}(n) \leq fc_{\text{sacoconf}}(n)\,, \tag{3.56}$$
$$n = n_t : fc_{\text{saconf}}(n) = fc_{\text{sacoconf}}(n)\,, \tag{3.57}$$
$$\forall n \in [n_t + 1, s_a] : fc_{\text{saconf}}(n) \geq fc_{\text{sacoconf}}(n)\,. \tag{3.58}$$

Again an arbitrary function fitting the conditions is defined as:

$$f_{\mathcal{C}_{\text{saconf}}}(n) := \begin{cases} \frac{f_{\mathcal{C}_{\text{sacoconf}}}(n_t)}{3} \cdot n & : 1 \leq n \leq 4 \\ 1 & : \text{else} \end{cases} \tag{3.59}$$

and both things are shown in Figure 3.13.

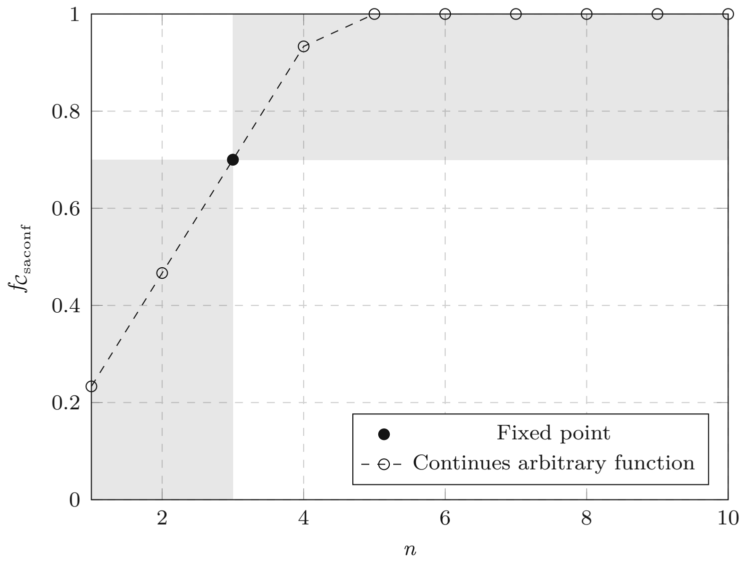


Fig. 3.13: Visual representation of the $f_{\mathcal{C}_{\text{saconf}}}$. The grey patches are the areas where the function has to be inside. The continues arbitrary function is an arbitrary function which fits the conditions.

With $f_{\mathcal{C}_{\text{saconf}}}$ all functions are defined. As well as n_t, x_t and $\mathcal{S}_{\text{SCCVC}}$. $\mathcal{S}_{\text{SCVCC}}$ has to be calculated. Table 3.8 shows the result of this calculation. It also shows $\mathcal{S}_{\text{SCVCC}}$ and their combination.

Position	$\mathcal{S}_{\text{SCVCC}}$ [calculated]	$\mathcal{S}_{\text{SCCVC}}$ [defined]	$\{\mathcal{S}_{\text{SCVCC}} \cap \mathcal{S}_{\text{SCCVC}}\}$
1	$[0, 7.7266]$	$[0, 0.2]$	$[0, 0.2]$
2	$[0, 14.6581]$	$[0, 0.8]$	$[0, 0.8]$
3	$[0, 18.7127]$	$[0, 1.8]$	$[0, 1.8]$
4	$\mathbb{R}_{0+}$	$[0, 3.2]$	$[0, 3.2]$
5	$\mathbb{R}_{0+}$	$[0, 5.0]$	$[0, 5.0]$
6	$\mathbb{R}_{0+}$	$[0, 7.2]$	$[0, 7.2]$
7	$\mathbb{R}_{0+}$	$[0, 9.8]$	$[0, 9.8]$
8	$\mathbb{R}_{0+}$	$[0, 12.8]$	$[0, 12.8]$
9	$\mathbb{R}_{0+}$	$[0, 16.2]$	$[0, 16.2]$
10	$\mathbb{R}_{0+}$	$[0, 20.0]$	$[0, 20.0]$

Tab. 3.8: The calculated results for $\mathcal{S}_{\text{SCVCC}}$ and the defined values for $\mathcal{S}_{\text{SCCVC}}$ and their combination.

The final act is to visualize the resulting multipoint solution. Figure 3.14 shows the simplified solution.

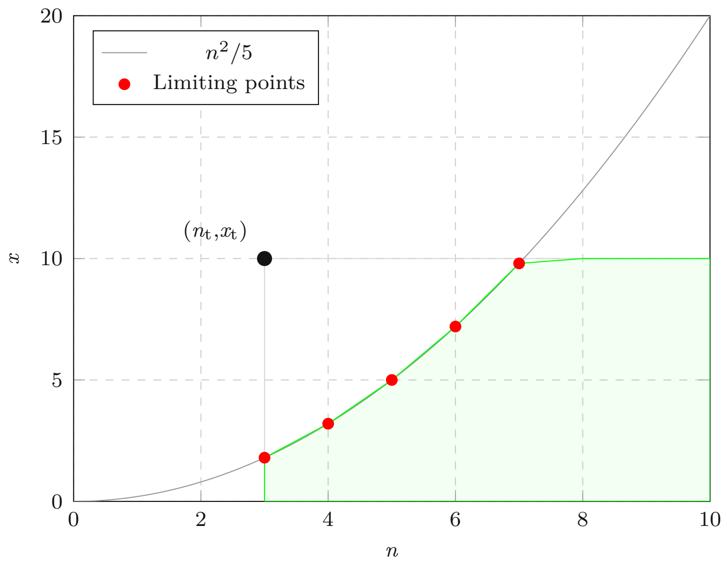


Fig. 3.14: Visual representation of the simplified multi point example and $n^2/5$ as it was one of the main constraints. If even one value falls into the green space it would keep the state.

3.4 Behaviour Analysis

So far the mathematical aspect of when and under what condition a state is lost has been analysed. This section focuses on the likelihood of staying or leaving a state under the condition of a white noise and a sharp step. The single point solution is analysed as the multipoint solution can be constructed by using multiple single point solutions, but it would not add anything for the goal of this section. The goal of this section is to deepen the understanding of the algorithm, while using the single-point solution. This understanding could then be used to define the solution itself, based on the signal attributes.

3.4.1 Analysis

Prior to starting the analysis a few things need to be defined.

1. The noise used is white Gaussian noise, so a normal distribution.
2. μ of the distribution is normalized to σ.
3. The used variable for the distribution x is also normalized to σ.
4. A step is instantaneous and pre & post step noise exists.
5. The step height is normalized to σ and is called γ.
6. The new input value v from the mathematical analysis is normalized to σ and is called $\tilde{v}$.
7. The value threshold x_t from the mathematical analysis is also normalized to σ and is called ϵ.
8. The sample amount threshold n_t from the mathematical analysis is normalized to s_a and is called θ, it will be redefined during the chapter.

Due to the normalisation to σ, the distribution has no longer a σ in the exponential term and by keeping the definition that the are of the integral form neg. to pos. infinity has to be 1, σ vanishes from the distribution,

$$\psi(x,\mu) = \frac{1}{\sqrt{2\cdot\pi}} \cdot e^{-\frac{1}{2}\cdot(x-\mu)^2}\,, \tag{3.60}$$

This normalisation has no effect on the integral,

$$\Phi(x, \mu) = \int_{-\infty}^{x} \psi(x', \mu)\, dx' \,. \tag{3.61}$$

Both of these functions are shown in Figure 3.15.

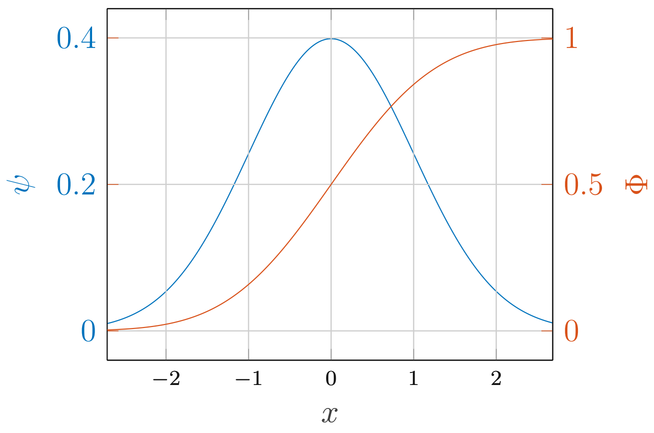
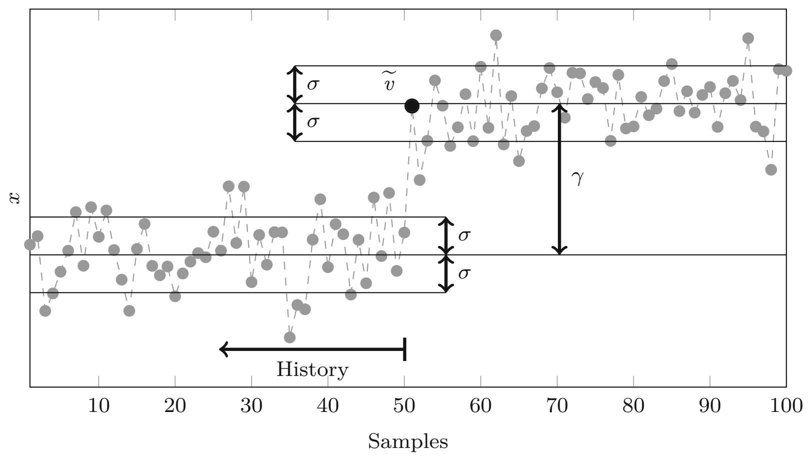
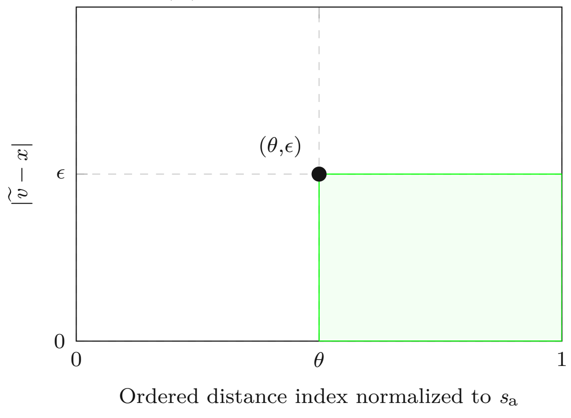


Fig. 3.15: $\psi(x, 0)$ and $\Phi(x, 0)$ graphically represented.

Figure 3.16 visualizes the used variables as part of a time-series and as the single-point solution.



(a) Time series



(b) Single-point Solution

Fig. 3.16: Two sub-figures which display where the normalized parameters can be found in a signal or in the algorithm.

For the first step let us assume the history length as infinite, therefore the history has a perfect distribution. The first question is which ϵ is needed for Δ to contain θ of the history,

$$\int_{\widetilde{v}-\epsilon}^{\widetilde{v}+\epsilon} \psi(x',0)\,dx' \geq \theta\,, \tag{3.62}$$

or for a given ϵ how big is $\widetilde{v}$ allowed to be? Figure 3.17 shows this over $\widetilde{v}$ and ϵ.

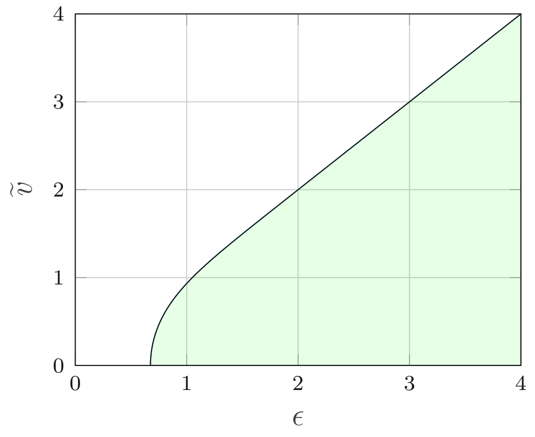


Fig. 3.17: The green area displays value combinations of ϵ and $\widetilde{v}$ which would hold a state. θ is considered 0.5 and the black line is the border and was calculated by transforming it to $\epsilon(\widetilde{v})$. It should be noted that the history is considered infinitely long.

By answering the first question, it is known which value would keep a state and which would loose a state. By assuming that the input value is a step plus noise,

$$\widetilde{v} = \gamma + \text{noise}\,, \tag{3.63}$$

two questions can be answered:

1. How likely is it to stay in a state, when assuming γ is 0?
2. How likely is it to detect a jump, when assuming γ is not 0?

Referring to question 1, the probability of staying in a state can be calculated using a distribution-weighted mean over the Boolean equation with the common substitution that true equals 1 and false equals 0. This results in changing the integral boundaries from negative & positive infinity to $-$&$+$ the outline from Figure 3.17 for a given ϵ,

$$\tilde{x}(\epsilon,\theta) = \underset{\text{for } \widetilde{v}}{\text{solve}} \left(\int_{\widetilde{v}-\epsilon}^{\widetilde{v}+\epsilon} \psi(x',0)\, dx' = \theta \right) , \tag{3.64}$$

$$P_{\text{stay}}(\epsilon,\theta,\gamma) = \int_{-\tilde{x}(\epsilon,\theta)}^{\tilde{x}(\epsilon,\theta)} \psi(x,\gamma)\, dx\,, \tag{3.65}$$

the solution is visualised in Figure 3.18 for $\gamma = 0$.

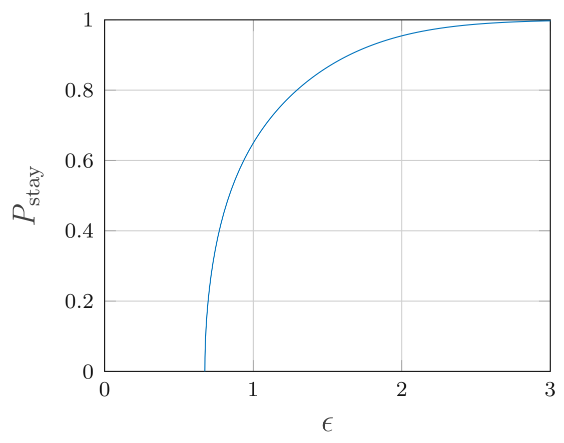


Fig. 3.18: The likelihood that a state is kept P_{stay} dependent on ϵ. θ is considered 0.5 and γ is considered 0 for this plot. It should be noted that due to numerical instability the inverse function was calculated ($\epsilon(P_{\text{stay}})$).

The same calculations can be done while varying θ. The result is shown in Figure 3.19. It should be noted that there are numerical inaccuracies inside the graph, visible at the top. The top should extend all the way back to $\epsilon = 4$. The reason for these inaccuracies is that the inverse function was calculated, because $P_{\text{stay}} \approx 0$ has higher relevance.

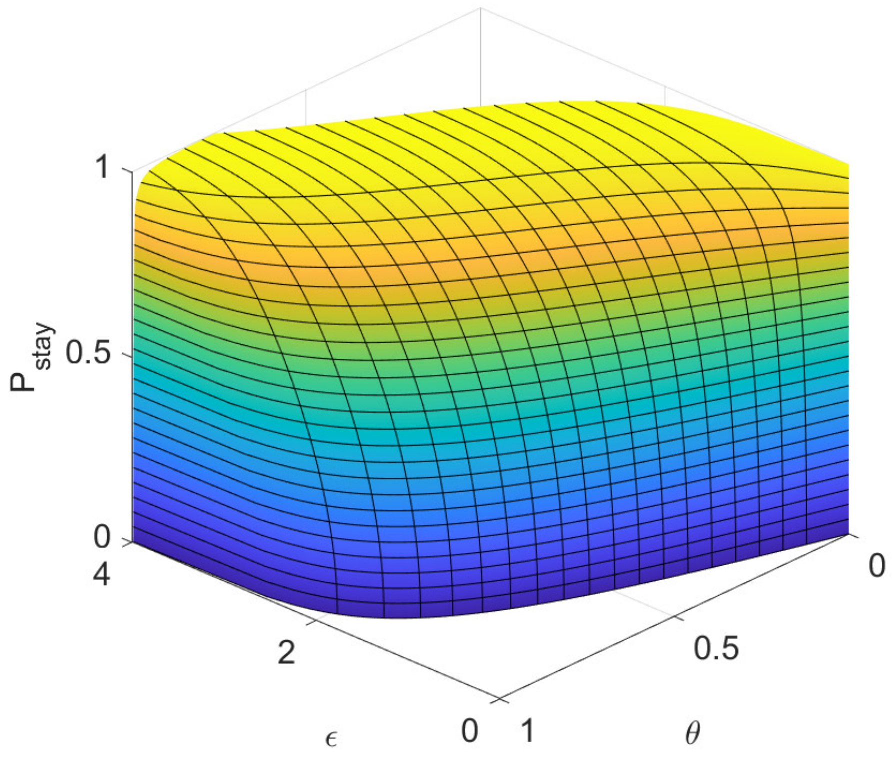


Fig. 3.19: The likelihood that a state is kept P_{stay} dependent on ϵ and θ. γ is considered 0 for this plot. It should be noted that due to numerical instability the inverse function was calculated ($\epsilon(P_{\text{stay}}, \theta)$).

The bottom of Figure 3.19 looks like the error-function over ϵ while the upper part looks like a shifted shrunken one. At $\theta = 0$ the calculation does not make sense as P_{stay} should be 1, but as soon as $\theta \neq 0$ it is correct again.
Referring to question 2: it is not as straightforward as the first question, because the jump does not need to be detected with the first value, it only needs to be detected at some point. With the base assumption that each incorrectly accepted values s_{b} fits perfectly into the post jump distribution, θ needs to be modified. The probability that it stays inside the state depending on θ is shown in a 3D multi-isosurface plot in Figure 3.20.

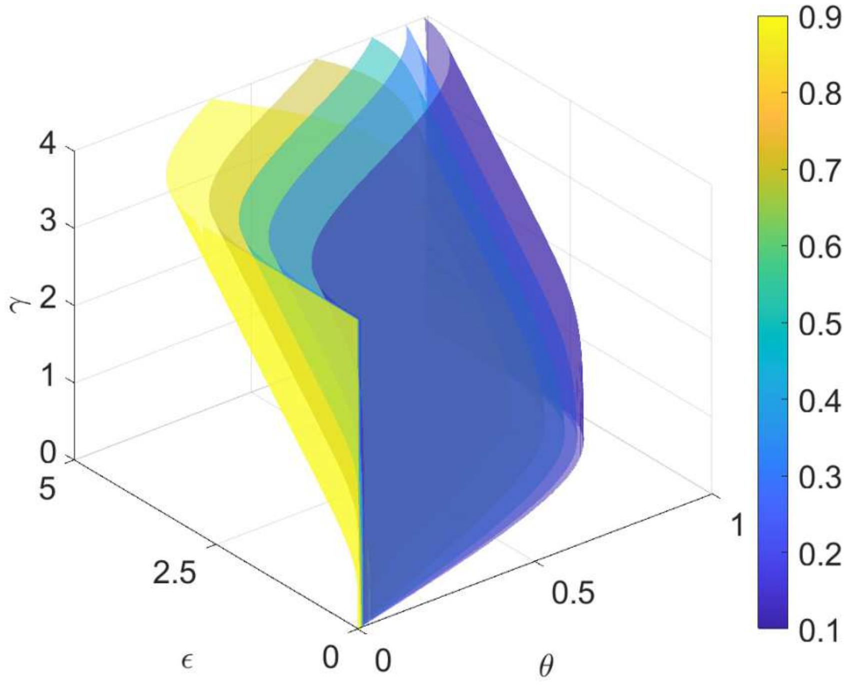


Fig. 3.20: The colour-coded likelihood that a state is kept P_{stay} dependent on ϵ, θ and γ. The layers have the values 0.9, 0.7, 0.5, 0.3, 0.1 .

Figure 3.20 displays quite a bit about the behaviour of the algorithm. The probability that it stays in the state decreases as the normalized sample threshold θ increases. This is as expected. The second aspect that can be seen is that there is a trade-of between the normalized sample threshold θ and the normalized value threshold ϵ , this is also as expected.

Making θ dependent on the incorrectly accepted values s_b reduces the amount needed to fulfil the condition as s_b increases, since s_b fulfil the condition as per definition,

$$\theta(s_\text{b}, s_\text{a}) = \frac{n_\text{t}(s_\text{a}) - s_\text{b}}{s_\text{a}} . \tag{3.66}$$

It should be noted that n_t can depend on the sample amount s_a. This possible dependency is defined by $f_{\mathcal{C}_\texttt{saconf}}$ and $f_{\mathcal{C}_\texttt{sacoconf}}$. In the base form of CCAM as defined in [1] it results in at least 50% of the samples,

$$n_\text{t}(s_\text{a}) = \lceil 0.5 \cdot s_\text{a} \rceil , \tag{3.67}$$

this will be used in this section if not stated otherwise. By applying the modified θ the likelihood can be calculated and is shown in Figure 3.21. Note, Figure 3.21 displays the part of Figure 3.20 where s_b is in the range of 0 to 0.5, because it is a coordinate transformation of one axis.

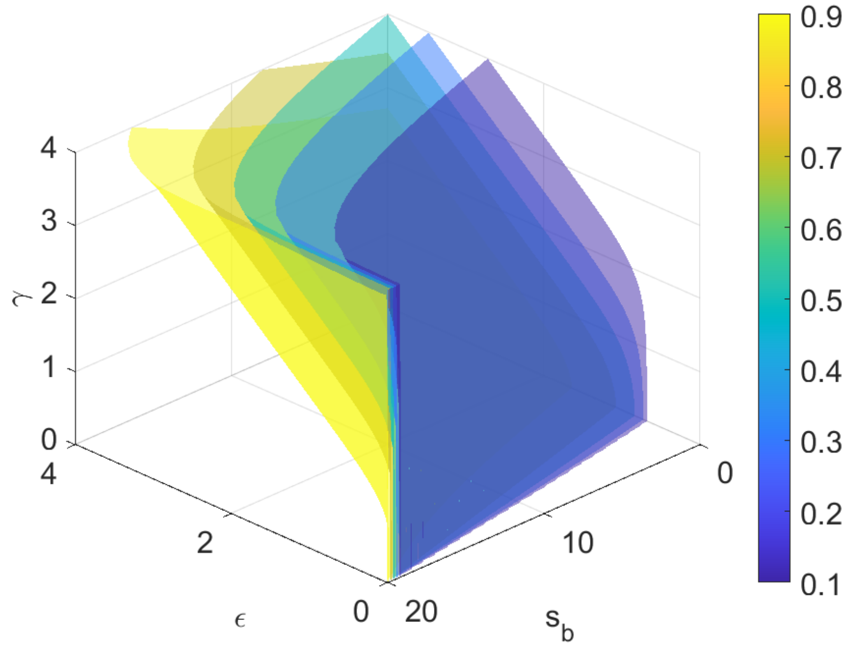


Fig. 3.21: The colour-coded likelihood that a state is kept P_stay dependent on ϵ, s_b and γ. The layers have the values 0.9, 0.7, 0.5, 0.3, 0.1 . s_b has been used as a continues number for this figure which it actually is not. s_a is considered as 40 for this plot and n_t is considered 20. It should be noted that there are visual computational errors in this Figure.

Figure 3.21 shows a lot of the algorithms behaviour. The more values are wrongfully accepted the less likely a jump is detected (increasing s_b direction). The lower the normalized value threshold ϵ is the more likely a jump is detected and the higher the normalized step height γ is the more likely a jump is detected. The next interesting question is how likely is a jump detected within s_b. This can be calculated by multiplying the likelihood that it stays over s_b and subtracting it from 1,

$$P_\mathrm{detected}(\epsilon, s_\mathrm{b}, \gamma) = 1 - \prod_{s_\mathrm{b}'=0}^{s_\mathrm{b}} P_\mathrm{stay}(\epsilon, s_\mathrm{b}', \gamma)\,. \tag{3.68}$$

Figure 3.22 shows the result of this calculation. As expected, the more values are past the jump the higher the probability, but with a decreasing efficiency.

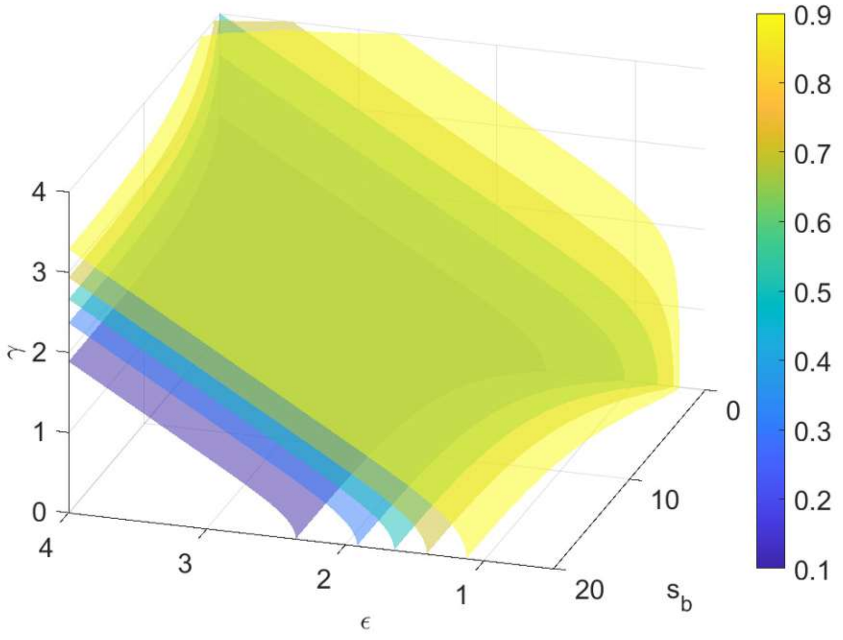


Fig. 3.22: The colour-coded likelihood that a jump was detected P_{detected} within s_{b}, dependent on ϵ, s_{b} and γ. The layers have the values 0.9, 0.7, 0.5, 0.3, 0.1 . It should be remembered that s_{b} is a discrete value and the values are simply interpolated to create a surface. s_{a} is considered as 40 for this plot and n_{t} is considered 20.

The likelihood that a jump is detected within and depending on s_{a} is another interesting question. It is shown in Figure 3.23.

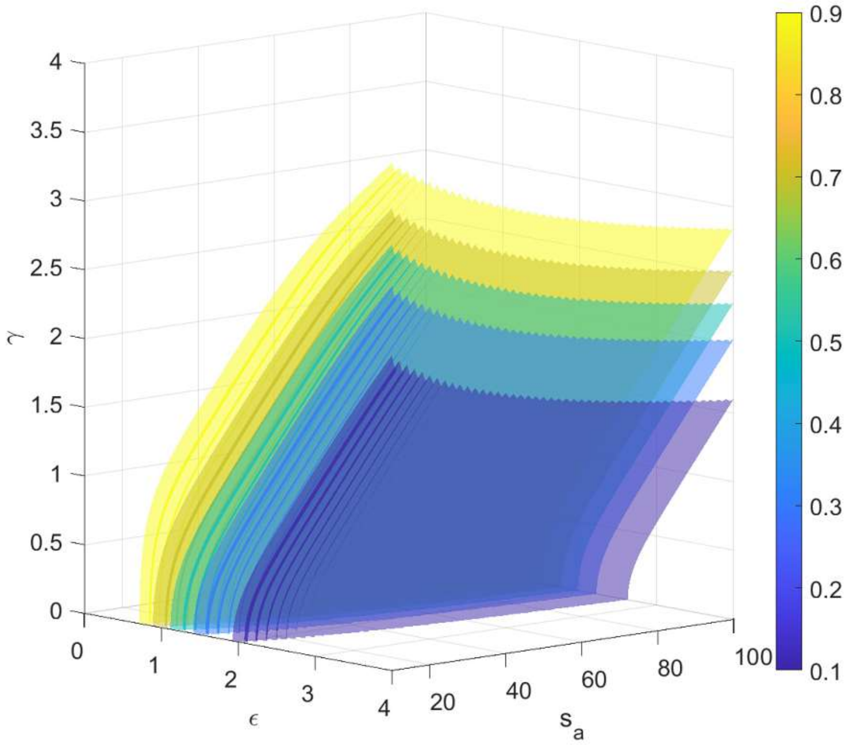


Fig. 3.23: The colour coded probability that a state is lost P_{detected} depending on γ, s_{a} and ϵ with s_{b} being s_{a}. The layers have the values 0.9, 0.7, 0.5, 0.3, 0.1 . It should be noted that if n_{t} is smaller or equal 0 the probability of loosing the state is considered 0. n_{t} is used as defined in equation 3.67. It should be noted that s_{a} starts at 10 as the simplification of an infinitely long history is far too inaccurate underneath it.

The likelihood that a jump is detected within a fixed s_{a} depending on n_{t} is another interesting question. It is shown in Figure 3.24.

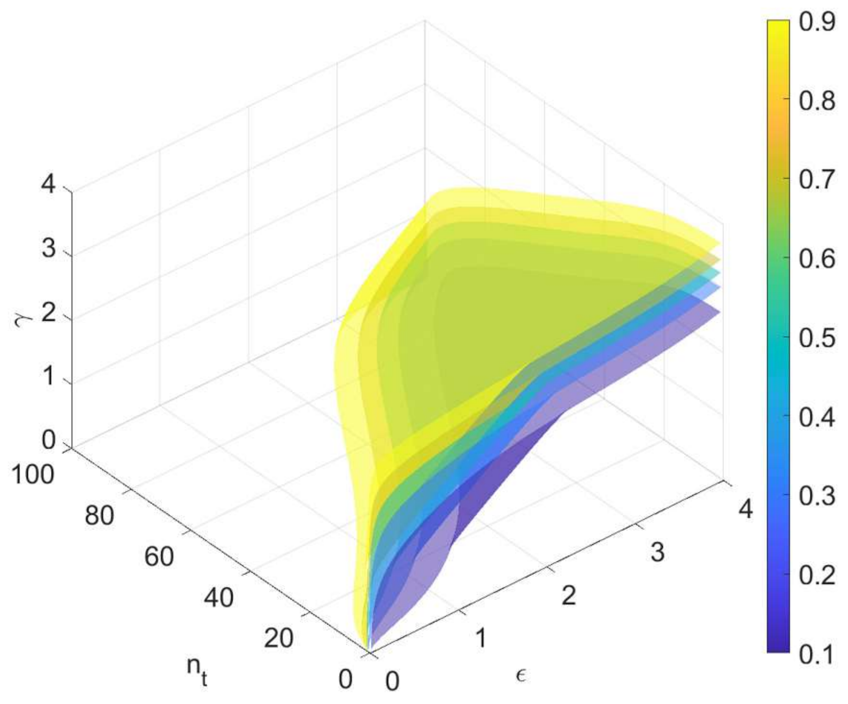
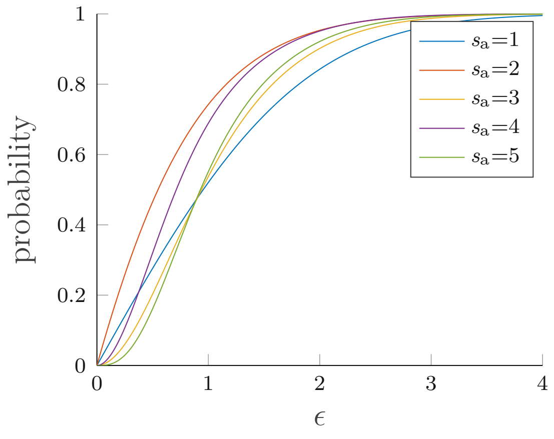


Fig. 3.24: The colour coded probability that a state is lost depending P_{detected} on γ, n_{t} and ϵ with s_{b} being n_{t}. The layers have the values 0.9, 0.7, 0.5, 0.3, 0.1 . It should be noted that if θ is smaller or equal 0 the probability of loosing the state is considered 0. s_{a} is set to 100. n_{t} starts at 1 in this graph.

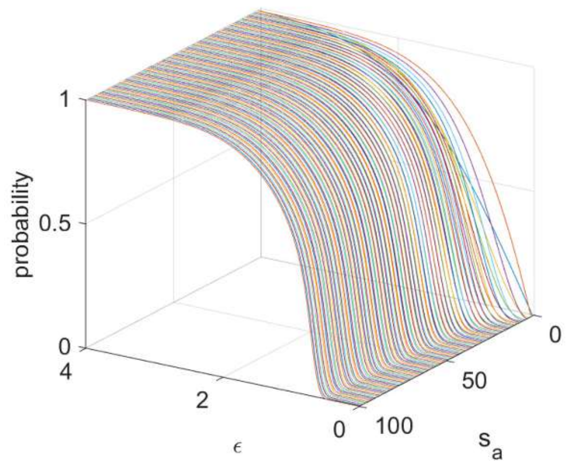
The next logical step is to look at the behaviour when s_{a} is changing. Calculating this however is not that easy. The original simplification of an infinitely long history can no longer be used. The correct formulation is basically a sum over binomial probability. The probability is the integral over the distribution with the limits of $x \pm \epsilon$ and then multiplying the new value distribution and integrating from negative infinity to positive infinity,

$$\int_{-\infty}^{+\infty} \psi(x,\gamma) \cdot \sum_{i=n_{\text{t}}(s_{\text{a}})}^{s_{\text{a}}} \binom{s_{\text{a}}}{i} \cdot \left(\int_{x-\epsilon}^{x+\epsilon} \psi(x',0)\, dx' \right)^{i} \cdot \left(1 - \int_{x-\epsilon}^{x+\epsilon} \psi(x',0)\, dx' \right)^{s_{\text{a}}-i} dx\,. \quad (3.69)$$

The evaluation is shown in Figure 3.25.



(a) s_a range is 1 to 5



(b) s_a range is 1 to 100

Fig. 3.25: The probability that enough values are inside θ to fulfil the condition of n_t, dependent on ϵ and s_a. It should be noted that n_t is used as defined in equation 3.67, which causes the at first glance strange behaviour. γ is 0 for this plot.

Figure 3.25 (b) seems to be slowly approaching the behaviour of Figure 3.18, which is to be expected since Figure 3.18 is the same probability but with an infinitely long history.

By varying γ, the probability of keeping a step can be calculated. The numeric evaluation is shown in Figure 3.26. It has a very interesting behaviour, it does not really change depending on s_a. This is interesting as it means that keeping a state is seemingly not effected by s_a, but detecting a state change is effected.

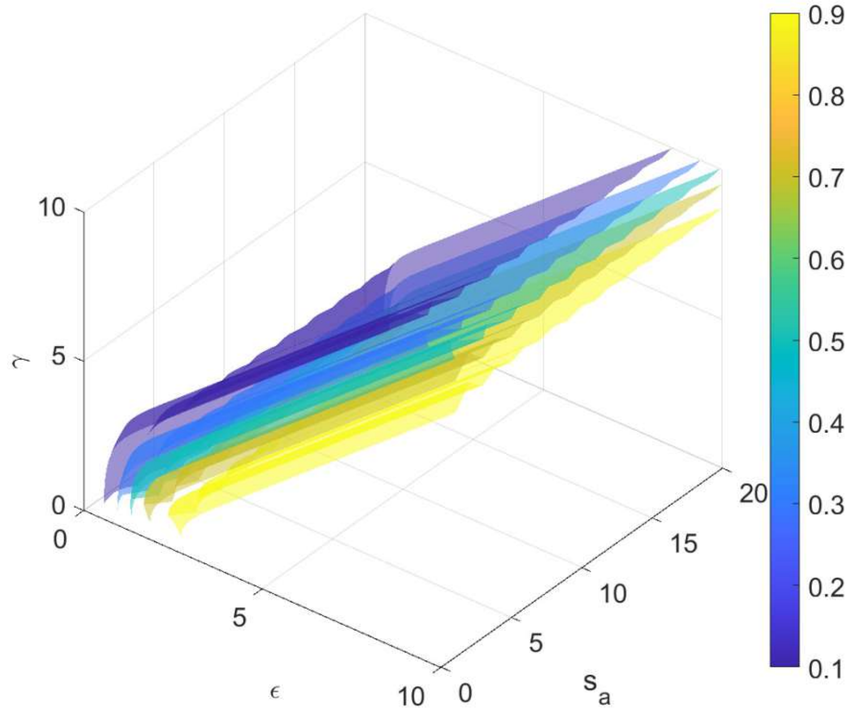


Fig. 3.26: The colour coded probability that a state is kept depending on γ, s_{a} and ϵ. The layers have the values 0.9, 0.7, 0.5, 0.3, 0.1 .

There is one more question which so far has not been asked. How likely is a state even found? The answer to this is straightforward. A state is found if the temporary state which is created after a state is lost can reach the maximum number of historic values. In other words the temporary state is not allowed to have any errors in it, otherwise it is dropped and a new temporary state is created. Also if a value fits to an existing state better than the building state, the existing state is prioritised, entered and the temporary one is deleted. In other words for each s_{a} the state has to be kept and would need to be of higher confidence than all old ones. Calculating the probability that a state is kept regardless of old states can be done and is visualized in Figure 3.27.

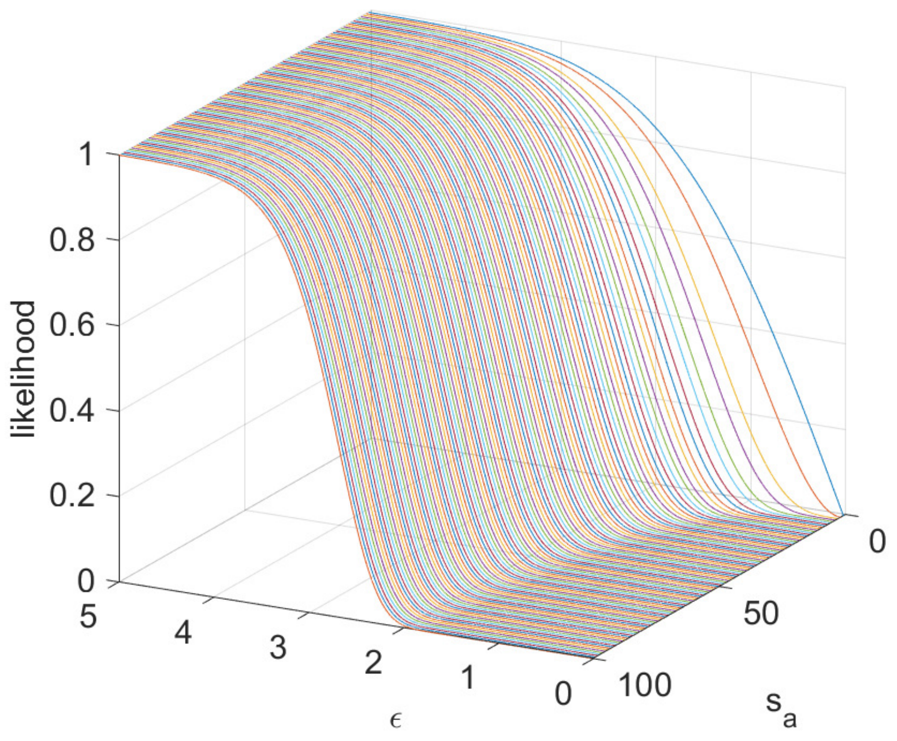


Fig. 3.27: The likelihood that a state is kept over s_a and ϵ.

3.4.2 Essential Findings

3.4.2.1 Parameters and their effects

There is a four-way connection between the normalized value threshold ϵ, the normalized sample threshold θ, the normalized step height γ and the probability that a state is kept. Fixating the probability creates a surface over ϵ, θ and γ as shown in Figure 3.20; But, the normalized step height γ is a parameter that is defined by the unknown signal and should not be predictable. The normalized value threshold ϵ and the normalized sample threshold θ on the other hand are freely choosable within their bounds. Due to the fact that a jump can be found within a time frame even after it has occurred increases the probability that it is found. This however requires a discrete approach, imposing rules onto the normalized sample threshold θ. Figure 3.24 display the probability that a jump was found within a time-frame after it has occurred(, with roughly speaking $n_t \approx 100 \cdot \theta$, relevant for the figure).
Successfully building a state under ideal condition seems to define a kind of minimum normalized value threshold ϵ. Figure 3.27 displays this. It seems like a normalized value threshold ϵ of around and above 4 is good and below 2 is unusable, for the used configuration of $\theta \approx 0.5$.

Increasing or decreasing the sample amount s_a is another aspect that needs to be discussed. Increasing it has a positive and a negative effect.
The positive effect of increasing s_a is that a jump is more likely to be found, displayed in Figure 3.23. Thereby increasing the sensitivity of the algorithm.
The negative effect of increasing s_a is that a state is harder to build, displayed in Figure 3.27. This in turn would require a higher normalized value threshold ϵ in order to counteract this effect and thereby decrease the sensibility.

3.4.2.2 Resulting possible improvements

There a multiple possible improvement resulting from this analysis.
The advantages of increasing s_a could be used if the the disadvantages could be removed. A possible approaches is introducing a changing normalized value threshold ϵ depending on the progress of building the state, thereby being able to define a higher one while building and a lower one while searching for jumps. Another approach could be granting full state "rights" at a fraction of the defined s_a. This would circumvent the disadvantages, by not introducing them in the first place.
Gathering parameters for the normalized value threshold ϵ should be possible from historic values and possible even at runtime. Through the minimal required normalized value threshold ϵ for a specific normalized sample threshold θ and a minimal reliable detectable normalized step height γ a compromise for the normalized value threshold ϵ can be statically defined. Using such a statically defined value and σ from the signal the required parameter for a single-point solution can be calculated. This in turn would require σ to be known at all times(, this also implies Gaussian white noise). Calculating it from the historic values is easy. Approximating it at runtime can include resource expensive algorithms. However, due to the fact that a local history is sorted cheap approximations might be usable.

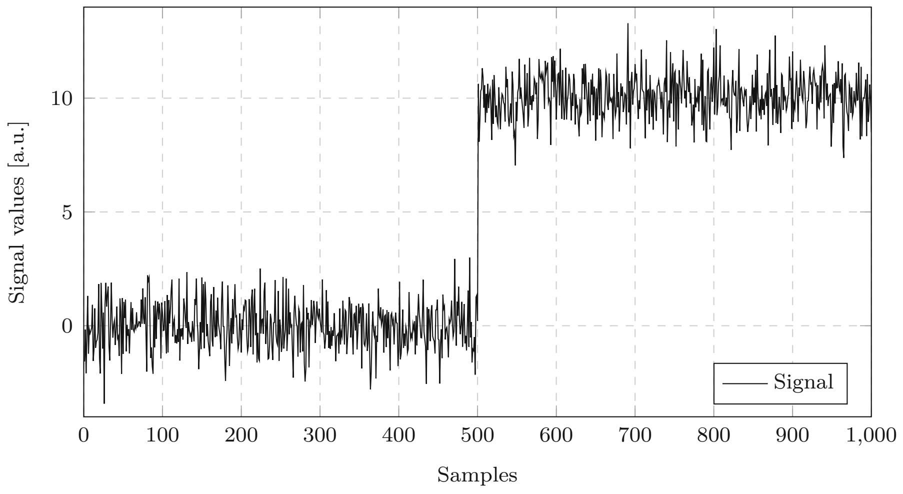
Chapter 4

Extension

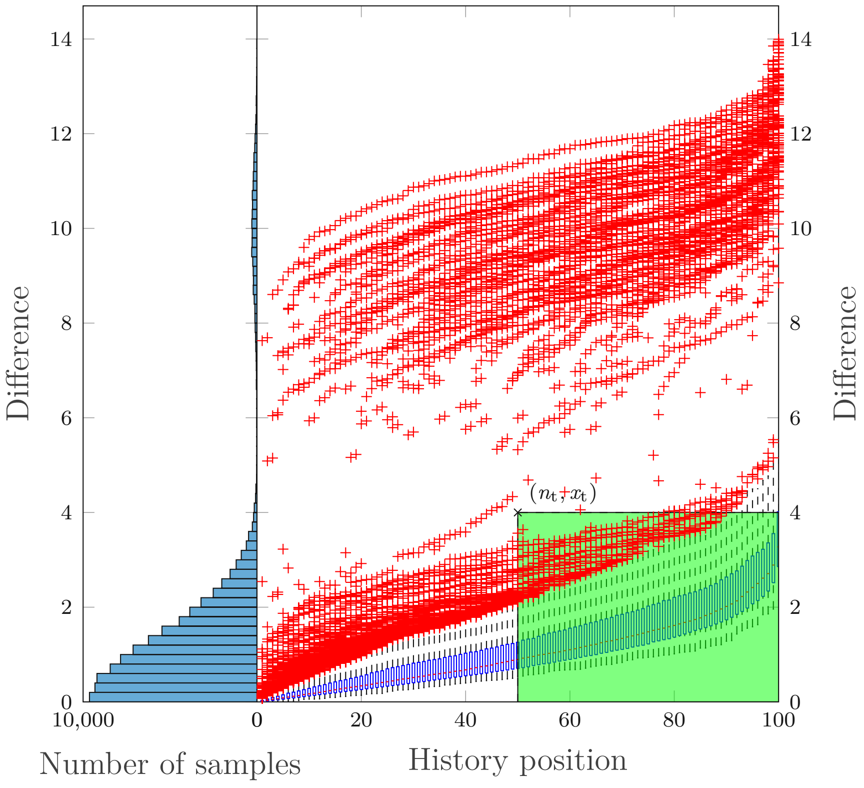
Based on the mathematical and statistical analysis multiple extensions can be formalized for the algorithm. These range from information for the developer, useful information for testing and possibly for deployment, to modifying the algorithm to improve its capabilities. It starts with a toolset to view the signal from the eyes of the algorithm. This toolset is meant for signal analysis and to check the statements of the automatic parameter detector. The automatic parameter detector is the second section and describes a possible solution to the problem. This is followed by three sections modifying different parts of the algorithm, with the goal to make the algorithm more reliable, increase sensitivity and decrease errors. This chapter finishes by combining the introduced extensions, as some negative side-effects are removed by that fusion.

4.1 Toolset for working with Data

The toolset is a simple tool that gives a user/ developer information about a signal; And if also given the used confidences, it can be used to interpret the data and make assumptions about the result that would be created by the step detector algorithm. To achieve this it displays a histogram of the distance Δ between an new input value v and the local history $\mathcal{H}$ (see section 3.3), of the whole signal and a box-plot for every index of Δ, for the whole signal. If the confidences are given, the solution space from subsection 3.3.3 can be overlayed.
Simply put, the toolset shows the step up to and including Δ in a statistical way that can then be interpreted by a user afterward. Figure 4.1 visualizes this. On the left side a histogram of all Δ is displayed and on the right side the box-plot for each index of Δ . How this data can be interpreted is shown later in this section.



(a) Signal



(b) Toolset response

Fig. 4.1: An example of a signal and the toolset response. In the toolset response on the left side is a histogram of Δ and on the right the box-plot for each sample position, overlayed with the keep state condition diagram. The maximum history length is 100 and the chosen functions result in a simple threshold condition also known as single-point solution at (50,4) .

4.1.1 Definition

The toolset creates Δ for the signal once enough historic values exist to fill the history. Let $\mathcal{S}$ be the signal and $\|\mathcal{S}\|$ the length of it, then it creates Δ starting at the position s_a in $\mathcal{S}$ until $\|\mathcal{S}\|$,

$$\forall x \in [s_\mathrm{a}+1, \|\mathcal{S}\|], \forall n \in [1, s_\mathrm{a}] : \Delta_n^x := |\mathcal{S}_x - \mathcal{S}_{x-n}| \; . \tag{4.1}$$

Then the histogram is created over all Δ_n^x.
Afterwards Δ_n^x is sorted over n, for each x. At the end a box-plot is created over the sorted Δ, over index n and the simplified representation of keeping a state from subsection 3.3.3 is added on top, which has been numerically calculated. The histogram displayed is the histogram of all values of Δ_n^x.

4.1.2 Interpretation

Using Figure 4.1(b) certain assumptions can be made:

1. The jump is likely detected, visible because the values of the second, smaller histogram peak are outside the keeping condition.
2. There might be a state drop during one of the states, but unlikely. Visible because no value barely misses the keep state condition. By contrast, if the keep condition would have been set to roughly 2.2 at n_t there would probably be multiple incorrect state losses.
3. The height of x_t could be slightly decreased when a states history is full. Visible because the single point solution captures all outliers of a suspected state at n_t and is still bigger than the maximum of those outliers.

To see if those assumptions are correct the state detector algorithm is applied to the signal and unsurprisingly it is exactly as predicted. This is visualized in Figure 4.2. It should be noted that x_t is set to $4 \cdot \sigma$ in accordance to subsection 3.4.2, therefore it was very unlikely that an incorrect state loss would happen, a state should build successfully. By also setting the jump height to $10 \cdot \sigma$ it would be nearly impossible that a jump would not be detected again in accordance to section 3.4. The reason this example was chosen is that it is unambiguous and easy to see and understand.

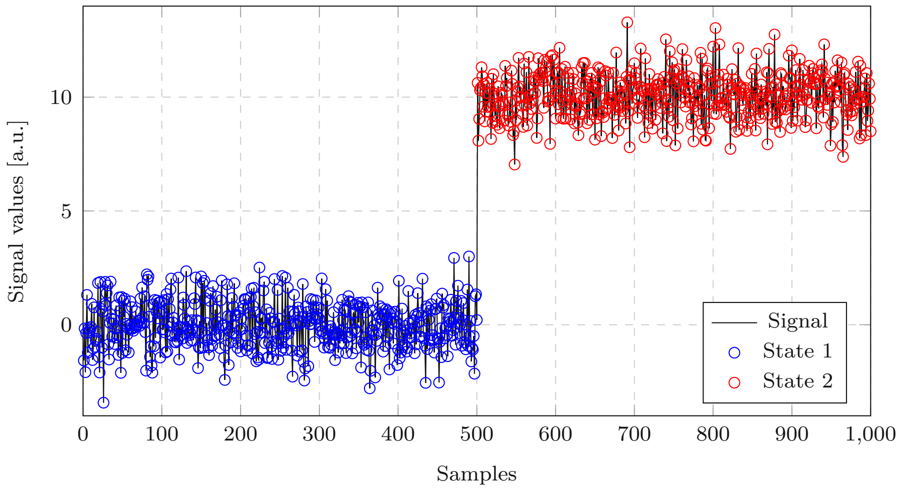


Fig. 4.2: The example from Figure 4.1(a) but with the state detector algorithm applied and the states visually displayed.

4.2 Automatic Parameter Detection

Automatic parameter detection is the first step to create a step detector that is capable of handling all possible signals. If the required parameters can be determined automatically from the historical values, it might be possible to transfer it to the runtime of the algorithm.

There are multiple approaches to how such a automatic parameter deducer could function, some are:

1. Give no parameters and let the algorithm figure out what could work.
2. Give it a parameter that describes an inherent property of the signal, like a time constant.
3. Let a user draw/select portion of a historic signal to use that information to define what a state should look like.

To be fair the last approach is very user intensive, but should create good parameters. The first approach is the most desired one, the second one is acceptable under the condition that the required parameter is highly stable. It could be possible to transform the second approach to the first one by using an iterative approach. The automatic parameter deduction algorithm that was implemented uses one parameter, which is s_a. The base assumption of the algorithm is that there are more "stable", non-jumping segments of the signal than jumping ones. It starts with guessing where jumps are, by searching for a step through correlation and then isolates patches which are unlikely jumps. Then the Δ of these patches are calculated, which is the same calculation as in section 4.1. Afterwards selected quantiles are made, which are then used to calculate a threshold for the single-point solution. This threshold then gets transformed into the confidence functions in a predefined manner.

4.2.1 Definition

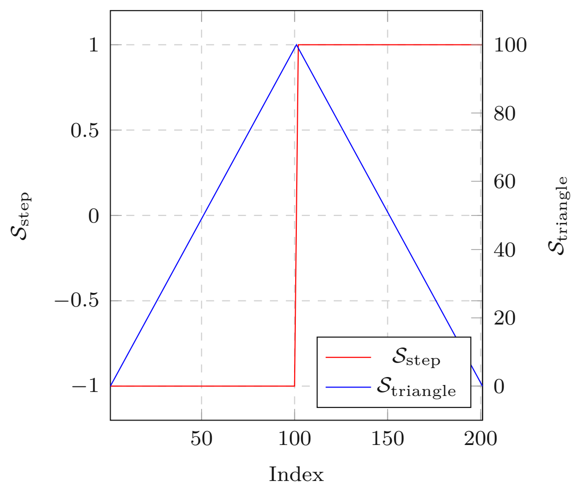
Let s_a be the history length. Let $\mathcal{S}$ be a sample. Let $\mathcal{S}_\mathrm{step}$ be a step signal for the convolution, it has s_a times "-1" followed by "0" followed by s_a times "1",

$$\mathcal{S}_{\mathrm{step},n} := \begin{cases} -1 & : 1 \leq n \leq s_\mathrm{a} \\ 0 & : n = s_\mathrm{a} + 1 \\ 1 & : s_\mathrm{a} + 1 < n \leq 2 \cdot s_\mathrm{a} + 1 \end{cases} . \tag{4.2}$$

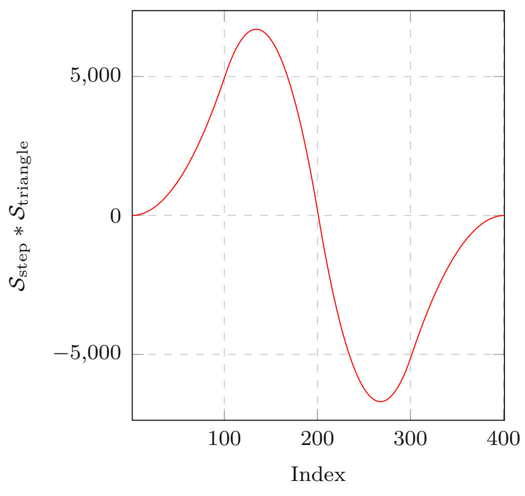
Let $\mathcal{S}_\mathrm{triangle}$ be a triangle signal with the maximum of s_a in the middle, starting and ending with 0 and the length of $2 \cdot s_\mathrm{a} + 1$,

$$\mathcal{S}_{\mathrm{triangle},n} := \begin{cases} n - 1 & : 1 \leq n \leq s_\mathrm{a} + 1 \\ -n + 1 + 2 \cdot s_\mathrm{a} & : s_\mathrm{a} + 1 < n \leq 2 \cdot s_\mathrm{a} + 1 \end{cases} . \tag{4.3}$$

Because convolutions are associative they can be calculated and displayed. They and their convolution are shown in Figure 4.3.



(a) $\mathcal{S}_\mathrm{step}$ and $\mathcal{S}_\mathrm{triangle}$



(b) Convolution of Step and Triangle

Fig. 4.3: The graphical representationism of the convolution kernel for a s_a of 100. Note that that the signals have a length 201 and 401 respectively.

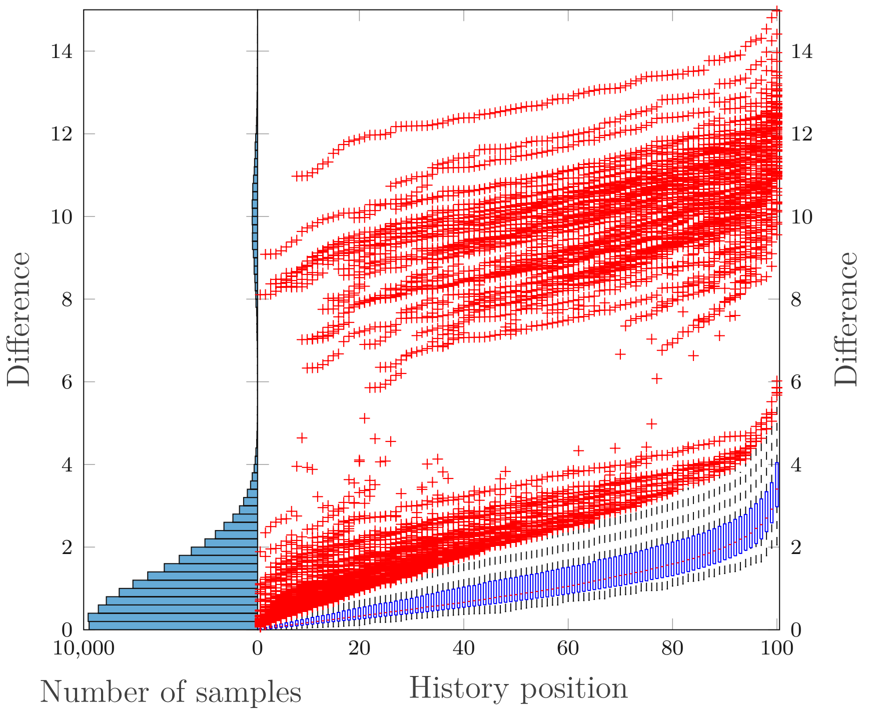
It should be noted that the convolution of step and triangle is highly similar to derivative of gauss. This is picked up in the analysis of the algorithm.
The absolute of the convolution of $\mathcal{S}$ and $\mathcal{S}_{\text{step}}$ and $\mathcal{S}_{\text{triangle}}$ is called $\mathcal{T}$,

$$\mathcal{T} := |\mathcal{S} * \mathcal{S}_{\text{step}} * \mathcal{S}_{\text{triangle}}| \tag{4.4}$$

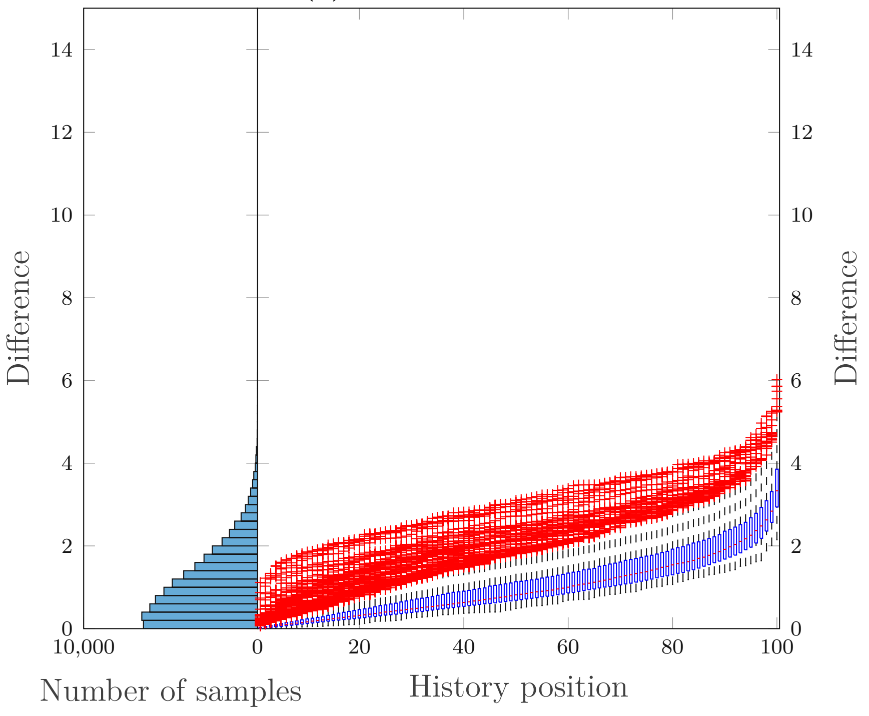
and it is similar in nature to a smoothed derivation.[1] This means there are now two possible interpretations for $\mathcal{T}$. The higher $\mathcal{T}$ is, the more it has in common with a high step. This is due to the fact that convolution can be used to find a specific pattern inside a signal. The pattern that should be found is the step and then a triangle in order to make found positions more dominant. Therefore higher values in $\mathcal{T}$ are more likely to be jumps.
The second interpretation is that it is a smoothed derivation. This means that places of high incline also have a high $\mathcal{T}$. Jumps are likely to have the highest incline. Therefore high peaks in $\mathcal{T}$ are likely to indicate jumps.
Finding peaks can be done by a simple local maxima. The quarter of peaks with the highest values are defined as jumps. It was experimentally determined that a quarter is acceptable. It was a trade-off between removing to much from the signal and keeping jumps. The positions of these maxima are used to segment the input signal and for each of those segments half of s_{a} is also removed at each end, in order to remove possible slopes from non-instantaneous jumps. Each segment, which is longer than s_{a} can now be used to generate independent statistics, in the same manner as the toolset did. The results of those statistics are merged at the end. The comparison to the pure toolset is shown in Figure 4.4. There are a few visible aspects:

1. The jump is not visible.
2. There are less overall values.
3. Median and quantiles are slightly lower, this is also the data skew that was intended to be removed.

[1] It should be noted that finite convolution depending on definition can decrease the size of the result or pad the signal with zeros. Padding the signal with zeros would introduce jumps at both ends. To combat this the input signal was mirrored on both ends and the middle section was used. This only influences the borders of the signal.



(a) Toolset of data



(b) Toolset of divided data

Fig. 4.4: Comparison of Toolset for base data and by Automatic parameter deduction splitted data, for a history length of 100 on an equivalent dataset as Figure 4.1.

With this the majority of the work has been done. The last missing piece is to define the functions. As already described a threshold of about $4 \cdot \sigma$ is desired for Gaussian noise, so the desired single point solutions is $n_\mathrm{t} := 0.5 \cdot s_\mathrm{a}$ and $x_\mathrm{t} := 4 \cdot \sigma$ and by using the relationship of $a = 1 - b$ for confidence and co-confidence functions the single-point solution can be enforced. The sample confidence and co-confidence are simple linear functions from 0 to 1 and respectively 1 to 0,

$$fc_{\mathrm{saconf}}(k) = \begin{cases} 1 & : k \geq s_\mathrm{a} \\ \frac{k}{s_\mathrm{a}} & : 0 \leq k < s_\mathrm{a} \end{cases} , \tag{4.5}$$

$$fc_{\mathrm{sacoconf}}(k) = \begin{cases} 0 & : k \geq s_\mathrm{a} \\ \frac{s_\mathrm{a}-k}{s_\mathrm{a}} & : 0 \leq k < s_\mathrm{a} \end{cases} . \tag{4.6}$$

The x_t is a bit harder to find, as simply using the value were about 99.9936658% of all values fall below is highly prone to errors. By finding two points and extrapolating, a higher reliability is produced. But on the other hand, by taking two points an implicit assumption is made that it follows a certain distribution. This algorithm is a proof of concept. The explicit assumption is made that it is a one sided Gaussian normal distribution. How it could/should be handled in the future is discussed in subsection 4.2.3. Another thing is by using Δ, it is not the original Gaussian distribution, it is the convolution of the distribution, however there is a simple conversion between them, the factor is $\sqrt{2}$. Let a and b be points in the value confidence/co-confidence function that describe the points where the functions have a probability of 1/0 and 0/1 then the mean of both values will be the single-point solution threshold. If a drift is happening the drift would be part of the noise and the noise would would not be purely random any more. It would cause the difference to gain an offset, changing the distribution to an equal distribution rounded of by a one sided Gauss. This also means that the sorting of values influences a drift-less signal stronger than a strongly drifting one. So using a specific position in the history will create an error. a and b shall be:

$$a := \mathrm{Quantil}(\Delta, 0.9) \tag{4.7}$$
$$b := a + 7.6 \cdot (\mathrm{Quantil}(\Delta, 0.95) - a) \tag{4.8}$$

By having a and b defined, fc_{valconf} and $fc_{\mathrm{valcoconf}}$ can be defined as:

$$\begin{aligned} fc_{\mathrm{valconf}} : & \quad \mathcal{R}_{0+} \rightarrow [0,1] \\ fc_{\mathrm{valconf}}(x) = & \begin{cases} 1 & : x \leq a \\ \frac{b-x}{b-a} & : a < x < b \\ 0 & : b \leq x \end{cases} \end{aligned} \tag{4.9}$$

$$\begin{aligned} f_{\mathcal{C}_{\text{valcoconf}}} : \quad & \mathcal{R}_{0+} \to [0,1] \\ f_{\mathcal{C}_{\text{valcoconf}}}(x) = & \begin{cases} 0 & : x \leq a \\ \frac{x-a}{b-a} & : a < x < b \\ 1 & : b \leq x \end{cases} \end{aligned} \tag{4.10}$$

4.2.2 Analysis of the Algorithm

As this algorithm is crude in nature the analysis is held short. There are three main aspects that need to be addressed:

1. The similarities to Canny edge detector.
2. The value stability.
3. The required amount of data.

The similarities to Canny edge detector are highly interesting. The Canny edge detection uses the derivation of Gauss. It is used as an image processing kernel. It uses two thresholds to determine strong and weak edges. In short, Canny edge detection is based on the idea that by using a Gaussian kernel, smoothing is done and by using the derivation of the kernel, the derivation of the data is produced. The true similarity between the algorithm described in this thesis and Canny is the similarity of the kernel used. The similarity is displayed in Figure 4.5. The similarity is a positive surprise because Canny edge detection has already proven itself to be capable of finding edges, in other words a mathematically highly similar approach has proven itself to be capable of finding jumps, but the algorithm described in this thesis uses a statistical threshold rather than a fixed one, so there can easily be false positives and false negatives. Canny edge detection was originally avoided as an approach as the algorithm in its base form uses signal specific parameters in the form of a hysteresis. The goal of this section was to find signal specific parameters and requiring them in order to get them does not make sense; However there are variations of canny edge detection, usually with the prefix auto or zero-parameter, which remove this criteria and they also rely on statistics. The difference that remains is that the value domain is explicitly defined, usually $0 - 255$, which is unwanted for this algorithm. Therefore modifying canny edge detection to solve this problem could have also been a viable approach.

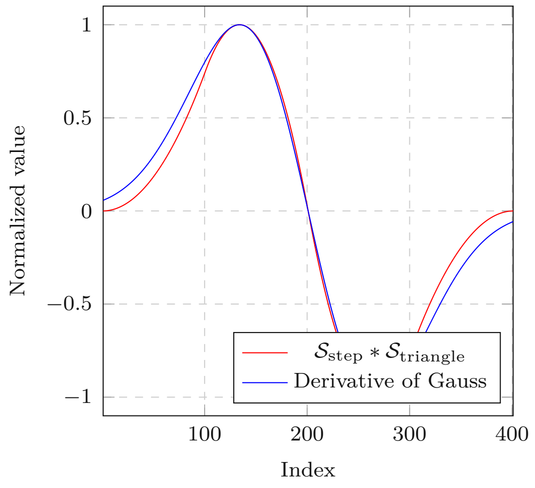
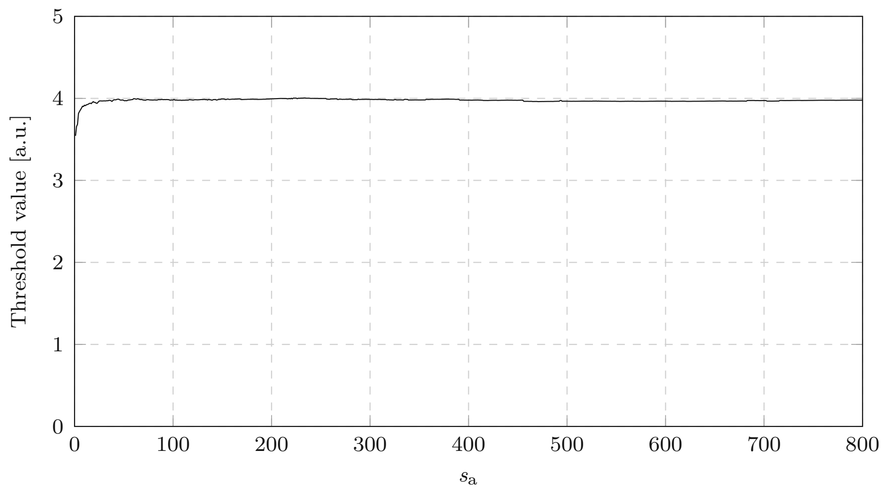
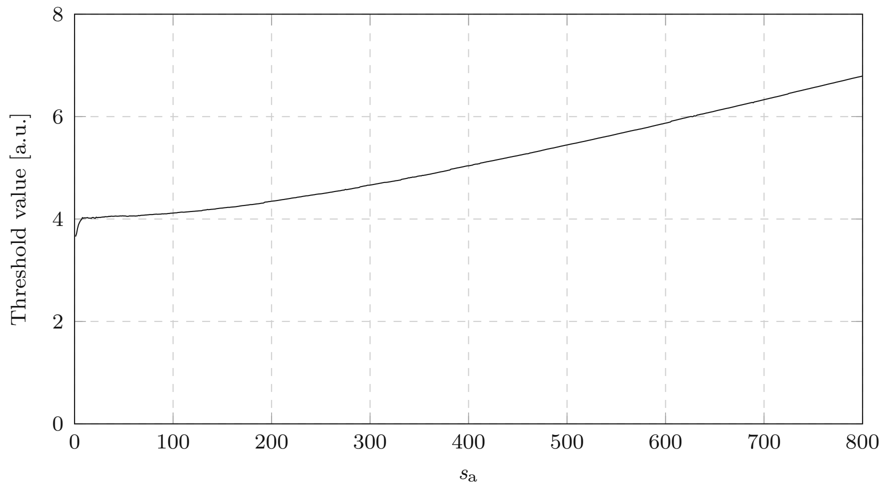


Fig. 4.5: The comparison between derivation of Gauss and convolution of $\mathcal{S}_{\text{triangle}}$ and $\mathcal{S}_{\text{step}}$. Gauss was fitted by subtracting the max of the convolution from the middle and multiplying it by $\sqrt{2}$ and using that as σ for Gauss. Both signals maxima were normalized to 1. The used s_{a} is 100.

The value stability is highly important. For a constant signal there should not be changes in the generated single-point solution. On the other hand for a drifting signal, changes over the used s_{a} need to occur as the drift itself is part of the noise distribution. Both versions are displayed in Figure 4.6 and it is visible that there are slight errors, due to the simplistic calculation method, but their influence is in the range of $\pm 10\%$ at low s_{a} and around $\pm 2\%$ at higher s_{a}.



(a) Threshold value calculated over drift free signal



(b) Threshold value calculated over drifting signal

Fig. 4.6: Visualisation of the single-point solution over the sample amount. The drift strength is 0.0050σ per sample.

The required amount of data is interesting to look at. The less data is available, the more problems the jump removal causes, as it will likely generate more false positives than true jumps. The second aspect to that is, that if little data is available it will behave like an inferior Toolset. If the amount of data is too small the jump removal will simply remove all available data, which results in the algorithm retuning no answer. Figure 4.7 displays the behaviour of being unable to find a solution for less than 300 data points.

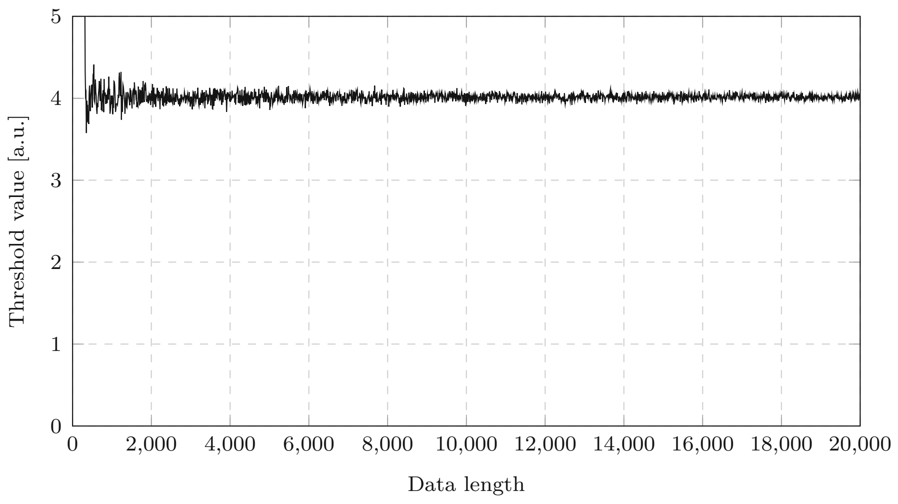


Fig. 4.7: The evaluation of the algorithm over an example similar to Figure 4.1, but with a variable data length. The used s_{a} is 100. Note that the first data point is at 310.

4.2.3 Summary and possible Improvements

As the proposed algorithm is in a state of prove of concept there are multiple possible improvements. However there are two major ones. Improving the jump detection and guessing the real distribution.

Improving the jump detection could be done by first letting the algorithm run through the available data, then let the step detector run with the generated parameters, creating states and then use the state information to create data fragments for the parameter generator. Doing this iteratively while a segment is long enough might create better parameters for specific data. This relies on the assumption that if there are no jumps in a segment and it is split arbitrarily then the solution should be the solution from the last iteration, or a value close to it, thereby validating the result.

Guessing the real distribution could be done by trying to match multiple ones and using the best fitting one. By having access to the real distribution of the signal an ideal parameter could be calculated. This fitting of distributions of course requires the signal to be divided into jump free segments.

4.3 Integrated Preprocessing

The power of preprocessing is nothing new and was deliberately not used, because it would change the signal and not the capabilities of the algorithm. There are multiple aspects of preprocessing that could improve how easy the algorithm stays inside a state while at the same time improve its sensibility. Integrating a data preprocessing algorithm into the step detector could result in such improvements while at the same time not being dependent on parameters. By targeting the history inside the step detector a sort of preprocessing can be done without effecting the data itself.
There is only one important question for the algorithm "How far is the current value from the values in the history?". Minimizing this for non jumps while leaving a jump unaffected would optimize the jump detector. Drifts are one aspect which widens the distribution while being easily removable locally. By making an approximation for the current incline, the incline could be removed for each value inside $\mathcal{H}$ and the new input value. By removing the drift the remaining distribution, offset and a possible jump become more visible. In order to not integrate a jump into these calculations the new value is not allowed to be considered for the drift calculation. On a detected jump this drift remover has to be reset/deactivated until enough values exist to reliably determine the incline again.
There are multiple approaches to calculating an incline, from discrete differences to linear approximation using least quadratic error. The two main differences are the value stability and processing power needed.

4.3.1 Definition

The two approaches looked at are mean of discrete differences and linear approximation using least quadratic error.

Mean of differences is easy and computationally cheap compared to the linear approximation. It is akin to a linear filter. It is the subtraction of the lower half from the upper half, divided by half of the length rounded down and then again divided by half of the length rounded up. In the case of having an uneven history length the middle value is ignored,

$$\zeta_{\text{MeanDiff}} := \frac{\sum_{i=s_\text{a}-\lfloor s_\text{a}/2\rfloor+1}^{s_\text{a}} \mathcal{H}_i - \sum_{i=1}^{\lfloor s_\text{a}/2\rfloor} \mathcal{H}_i}{\lceil s_\text{a}/2\rceil \cdot \lfloor s_\text{a}/2\rfloor} \tag{4.11}$$

This incline then can be removed from the history by removing a polynomial of first order from the history and the new value. The new value is treated as the $s_\mathrm{a}+1$ value for the polynomial, the resulting Δ is called Δ_MeanDiff,

$$\Delta_\mathrm{MeanDiff} := \left| (v - (s_\mathrm{a}+1)\cdot\zeta_\mathrm{MeanDiff}) - \left(\mathcal{H} - \begin{bmatrix}1\\2\\3\\\vdots\\s_\mathrm{a}\end{bmatrix}\cdot\zeta_\mathrm{MeanDiff}\right)\right| \tag{4.12}$$

Linear approximation using least quadratic error is computationally more complex. By using a simple matrix of the zeroth and first values for a first-order polynomial, an overdetermined linear system of equations with offset and incline can be "solved" by a QR-Factorization,

$$\mathcal{A}^{s_\mathrm{a}\times 2} := \begin{bmatrix}1, & 1\\1, & 2\\1, & 3\\\vdots & \vdots\\1, & s_\mathrm{a}\end{bmatrix}, \tag{4.13}$$

$$\mathcal{A}^{s_\mathrm{a}\times 2}\cdot\begin{bmatrix}\text{offset}\\\zeta_\mathrm{LinAp}\end{bmatrix} = \mathcal{H}^{s_\mathrm{a}\times 1}, \tag{4.14}$$

$$\begin{bmatrix}\text{offset}\\\zeta_\mathrm{LinAp}\end{bmatrix} = \mathcal{A}^{s_\mathrm{a}\times 2}\backslash\mathcal{H}^{s_\mathrm{a}\times 1}, \tag{4.15}$$

it should be noted that the cases s_a is 1 or 2 are ignored in this case. Again the history and the new value get adjusted creating the difference called Δ_LinAp,

$$\Delta_\mathrm{LinAp} := \left| (v - (s_\mathrm{a}+1)\cdot\zeta_\mathrm{LinAp}) - \left(\mathcal{H} - \begin{bmatrix}1\\2\\3\\\vdots\\s_\mathrm{a}\end{bmatrix}\cdot\zeta_\mathrm{LinAp}\right)\right| \tag{4.16}$$

Both adaptations have a problem with smaller s_a. The reason for this is that both adaptations can not differentiate between occurring noise and a drift. On the other hand, the base algorithm has no problem with an occurring drift, at a low s_a. Therefore both adaptations only become active if a certain history length has been surpassed.

The chosen minimal history length is 5 and to have a smooth transition the weighted average is used in the interval from 5 to 10,

$$l := |\mathcal{H}|\,, \tag{4.17}$$

$$\Delta_{\text{used}} = \begin{cases} \Delta & : l < 5 \\ \Delta \cdot \left(1 - \frac{l-5}{5}\right) + \Delta_{\text{adaptation}} \cdot \frac{l-5}{5} & : 5 \leq l \leq 10 \\ \Delta_{\text{adaptation}} & : 10 < l \end{cases}\,. \tag{4.18}$$

4.3.2 Adaptation of the Step-detector

The adaptation of the step detector is quite simple. It requires an additional function with which the Δ is calculated. By extending the algorithm by this additional configurability it can be can adapted in the future if a better method is found.

4.3.3 Analysis of the adaptation

The main focus of the both methods is to eliminate the drift. The base algorithm is by nature to some degree resistant to drifts, because it only looks at data in close proximity, but by increasing the history length the drift becomes more noticeable. At the same time the longer the history is, the easier it is to detect smaller jumps, as there are more opportunities to loose the state, while at the same time the single point solution can be tailored more strictly. There are four aspects to analyse:

1. How does the adaptation affect drift-less signals?
2. How does the adaptation affect drifting signals?
3. How does the adaptation affect signals with a pattern?
4. How does the adaptation affect non-instantaneous jumps?

The following analysis calculates the single-point solution the same way the toolset would.

Drift-less signals are of major importances. If the adaptation affects them strongly all advantages might be lost. Figure 4.8 displays the calculated single-point solution for a signal without a jump and no drift, over s_{a}. It is clearly visible that the adaptation has problems with small s_{a}. It becomes practically indistinguishable around the 200 mark for s_{a}. The unadjusted versions of the displayed signals show the behaviour of the incline adaptations if they would be allowed at low s_{a}.

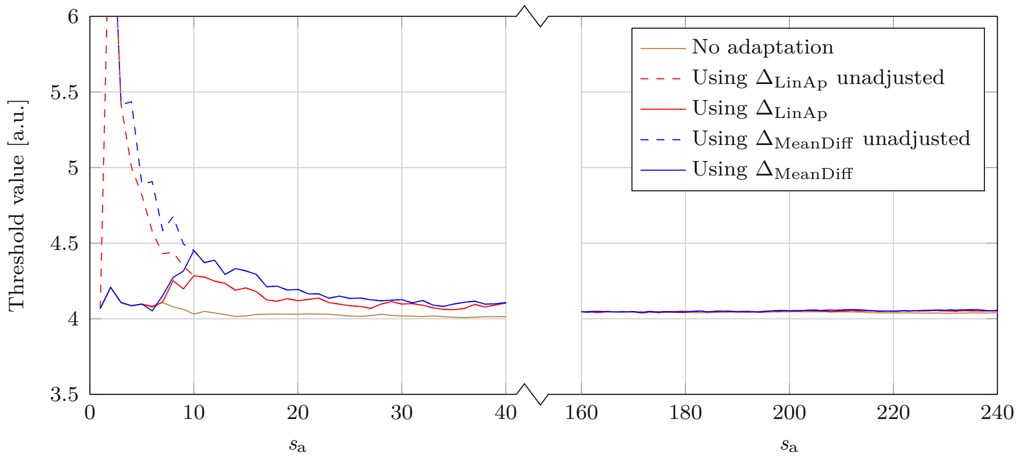


Fig. 4.8: Comparison of the threshold extracted from a jump-less signal, with no drift using different ways to calculate Δ, over s_a.

Drifting signals should highlight this adaptation. Unsurprisingly they do, as displayed in Figure 4.9. They provide a good solution even though a drift is applied to the signal. Using the adaptation would provide the ability to detect a step even when it would happen very inconveniently from the eyes of the algorithm, as displayed by Figure 4.10. The reason why it would be considered inconvenient for the algorithm is because the jump is opposing the drift, therefore creating values which have been recently encountered and accepted by the state. In other words with an s_a of 200 it finds enough values inside the history for the post-jump values to keep the state. The second negative aspect is that a higher threshold is required to reliably keep a state in the first place.

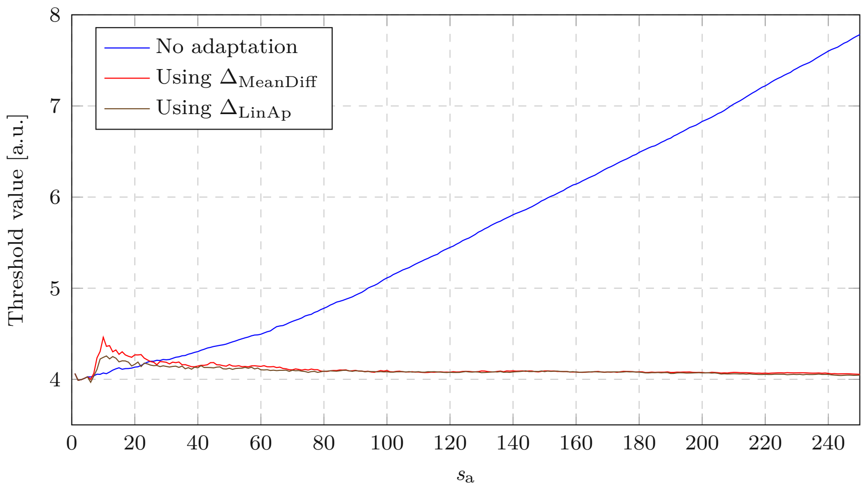
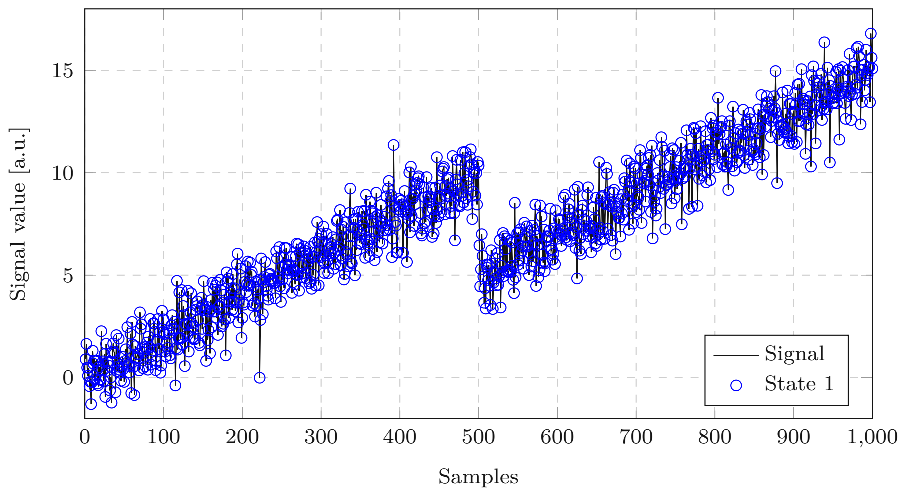
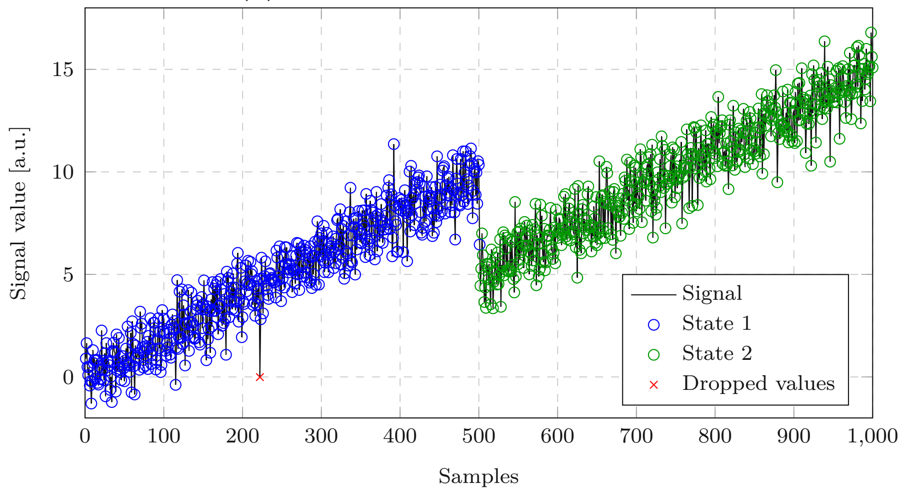


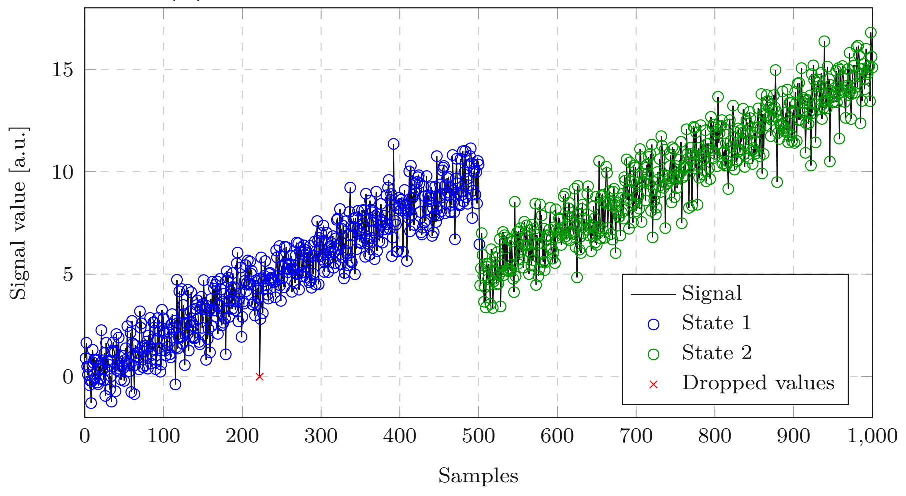
Fig. 4.9: Comparison of the threshold extracted from a jump-less signal, with a drift of strength 0.02σ per step, using different ways to calculate Δ, over s_a.



(a) Step-detector no adaptations.



(b) Step-detector with $\Delta_{\mathrm{MeanDiff}}$ adaptation.



(c) Step-detector with Δ_{LinAp} adaptation.

Fig. 4.10: Comparison of the step-detector on a strong drift with a negative jump. The jump height is 5σ and the drift strength is 0.02σ per value. All have a s_{a} of 200 and they use a single-point solution threshold of $6.8\sigma/4.08\sigma/4.04\sigma$ respectively, as calculated by Figure 4.9.

.

Pattern containing signals are an interesting case. They are mainly looked at because the patterns can be interpreted as a changing drift. Figure 4.11 shows the suggested single-point solution of the base algorithm and both adaptations for a sine signal, with a pattern length of 500 samples. There seems to be a kind of resonance peak in both adaptations, which seemingly separates the solution between viewing the pattern as a local drift or as being part of the distribution. Δ_{MeanDiff} and Δ_{LinAp} have significant differences above 600. It is unsurprising that changing the length of the pattern, the displayed response in Figure 4.11 scales accordingly.

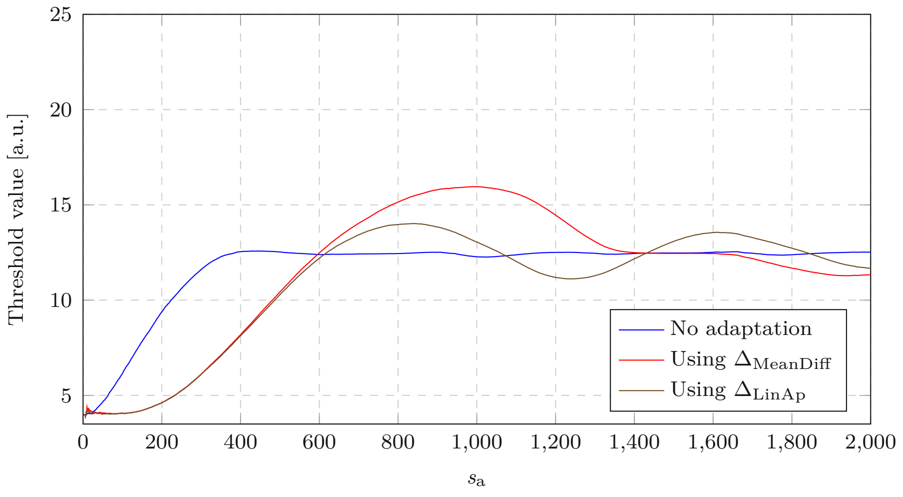


Fig. 4.11: Comparison of the threshold extracted from a jump-less signal, without a drift, with an sine pattern of length 500 samples, using different ways to calculate Δ, over s_a. Note the difference in scaling compared to Figure 4.8 and Figure 4.9

Non-instantaneous jumps are highly common in data and can be viewed as suddenly strongly drifting signals. Strongly drifting signals are no problem for this extension, as they adapt to the drift. In this case, this might be a negative. If a jump is not found fast, it starts to be interpreted as a drift rather than a jump. The root of this problem stems from the indirect redefinition of the difference between a jump and a drift. So far it was purely defined by the incline. With this adaptations the value of the incline is of no concern any more, but rather the speed with which it changes. The speed is of course relative to the history length. Both adaptations are outperformed by the base algorithm in therms of sensitivity on non-instantaneous jumps, when they are brought to the sensitivity limit of the base algorithm. The sensitivity limits were explored in section 3.4. A possible fix is to limit the the positions inside the history from which the information for the incline is collected, thereby introducing a kind of lag element into the algorithm. As always lag elements can have very negative effects, especially on resonance phenomena. The second problem with reducing the amount of information used is the value stability. Mathematically speaking, the reduction in used historic values is quite simple. By using only values around the middle of the history and defining the amount of values used to 50% of the history length, the algorithm has a delay of $25\% s_\mathrm{a}$ and a value stability equal to $50\% s_\mathrm{a}$. For the mean of discrete differences, the new incline is calculated in the following manner:

$$s_\mathrm{aMiddle} := \lceil s_\mathrm{a}/2 \rceil \tag{4.19}$$

$$s_\mathrm{aQuarter} := \lceil s_\mathrm{a}/4 \rceil \tag{4.20}$$

$$L_\mathrm{LB} := s_\mathrm{aMiddle} - s_\mathrm{aQuarter} + 1 \tag{4.21}$$

$$L_\mathrm{UB} := s_\mathrm{aMiddle} + s_\mathrm{aQuarter} - 1 \tag{4.22}$$

$$\zeta_\mathrm{MeanDiffDelay} := \frac{\sum_{i=s_\mathrm{aMiddle}}^{L_\mathrm{UB}} \mathcal{H}_i - \sum_{i=L_\mathrm{LB}}^{s_\mathrm{aMiddle}} \mathcal{H}_i}{s_\mathrm{aQuarter} \cdot s_\mathrm{aQuarter}} \tag{4.23}$$

For the linear approximation the new incline would be calculated in the following manner, again only using the middle of the historic data:

$$s_\mathrm{aMiddle} := \lceil s_\mathrm{a}/2 \rceil \tag{4.24}$$

$$s_\mathrm{aQuarter} := \lceil s_\mathrm{a}/4 \rceil \tag{4.25}$$

$$L_\mathrm{LB} := s_\mathrm{aMiddle} - s_\mathrm{aQuarter} + 1 \tag{4.26}$$

$$L_\mathrm{UB} := s_\mathrm{aMiddle} + s_\mathrm{aQuarter} - 1 \tag{4.27}$$

$$L_\mathrm{AU} := 2 \cdot s_\mathrm{aQuarter} - 1 \tag{4.28}$$

$$\mathcal{A}^{L_\mathrm{AU} \times 2} := \begin{bmatrix} 1, & 1 \\ 1, & 2 \\ 1, & 3 \\ \vdots & \vdots \\ 1, & L_\mathrm{AU} \end{bmatrix} \tag{4.29}$$

$$\begin{bmatrix} \text{offset} \\ \zeta_\mathrm{LinApDelay} \end{bmatrix} = \mathcal{A}^{L_\mathrm{AU} \times 2} \backslash \mathcal{H}_{L_\mathrm{LB} \ldots L_\mathrm{UB}}^{L_\mathrm{AU} \times 1} \tag{4.30}$$

[2] There are also filters which can approximate the derivate. One of them is SavitzkyGolay [3] filter. It has multiple components to it. The used one is the first derivative for a first order fit, as an asynchronous FIR filter. The asynchronicity is no problem in this case as it is desired that a delay is still part of it. The filter has a frame length which

[2]The ... operator here used means that only values from left side of ... to right side of ... are used.

[3]`https://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter` `https://aip.scitation.org/doi/pdf/10.1063/1.4822961`

shall be roughly $s_a/4$, it has to be uneven, so adding one if is even. It also has to be at least one longer than the order, which leaves it to be at least 3. To increase the value stability, the mean over half of the values in the middle is done. This operation again is a digital FIR filter. In other words both can be combined and by padding the edges of the coefficients with zeros, a matrix with filter coefficients can be created which gets cycled through depending on the current s_a. This incline shall be called ζ_{SGolay}. Figure 4.12 visualizes the effect of the introduced lag. As expected they have worse resonance phenomena. But without them it would not be possible to detect slow non-instantaneous jumps and would be outperformed by the base algorithm.

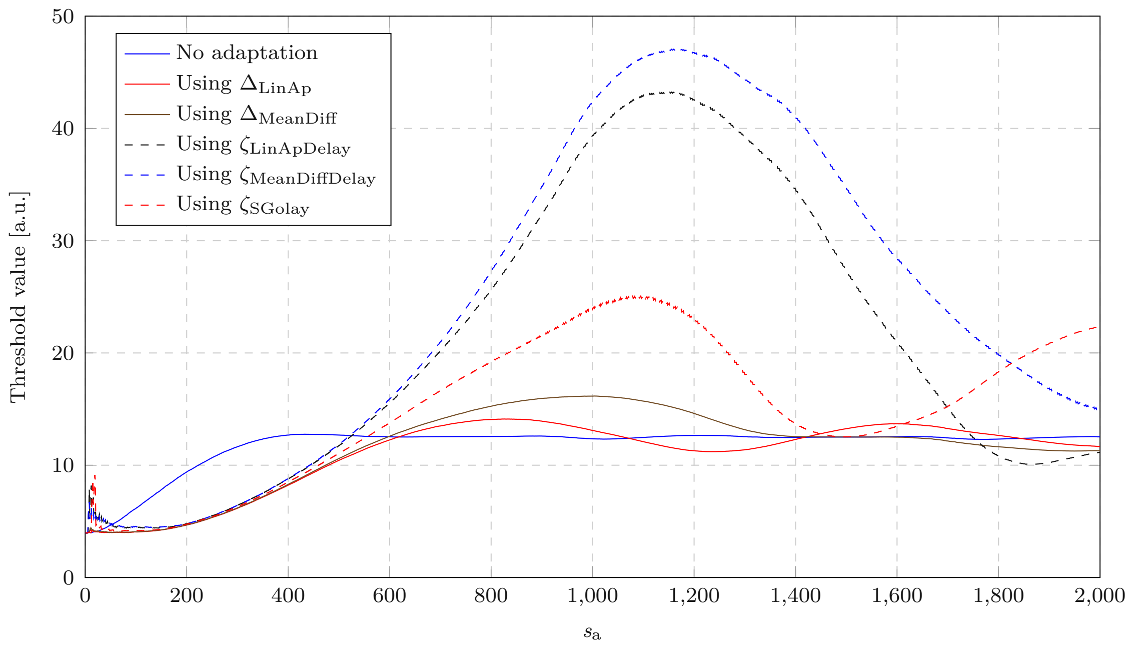


Fig. 4.12: Comparison of the threshold extracted from a jump-less signal, without a drift, with an sine of length 500 samples using different ways to calculate Δ, over s_a. A focus is on the 3 new ways of calculation.

4.3.4 Summary and possible Improvements

The introduced adaptations in this extension were designed to make the algorithm more drift resistant, which all of them fulfilled. On signals with constant or slowly changing incline the adaptations produce an excellent result. If patterns are part of the signal, resonance phenomena can occur. These phenomena have a negative effect on the sensitivity as they require the single-point solution to include higher values.

From the perspective of the computational resources required, linear approximation with least quadratic error requires the calculation of a pseudo inverse each cycle. The size of the required matrix , if the delayed version is used, is $0.5 \cdot s_a$. Compared to

that, the mean of discrete differences is FIR filter of order $0.5 \cdot s_{\mathrm{a}}$, and the history array doubles as filter array. The filter coefficients can easily be calculated. As a side-note the SavitzkyGolay approach would also be a FIR filter, but of size close to $0.75 \cdot s_{\mathrm{a}}$. The filter coefficients can be calculated, but it requires folding making it a lot more costly than the mean of discrete differences. For both FIR filters the coefficients can be precomputed and stored as constants, increasing runtime performance drastically.
As can be seen from non-instantaneous jumps, algorithms that work better can be found, especially when using algorithms specially designed for line fitting, derivative calculation. One of the possible approaches could be RANSAC. A full RANSAC algorithm could be implemented and as a cost function $f_{\mathcal{C}_{\mathrm{valcoconf}}}(2 \cdot \text{"distance"})$ could be used. The reason that it might be beneficial to use twice the distance for the $f_{\mathcal{C}_{\mathrm{valcoconf}}}$ when used by RANSAC is that the co-confidence function is at 0.5 when the input is at $4 \cdot \sigma$, so by using twice the distance this would push 0.5 to the distance of $2 \cdot \sigma$. The downside of using the co-confidence function for RANSAC would be that it becomes dependent on it. If it were incorrect, it could not calculate an incline. As seen in the following extensions, the assumption of having a usable co-confidence function at all times should not be made. A second aspect why linear filters are prioritised is that linear filters can be easily implemented in hardware, which would decrease the computation time of them drastically.

4.4 Dynamic Value Confidences and Phantom History

It was already multiple times mentioned that having a higher threshold while building a state could be advantageous. Section 3.4 ended with the likelihood that a state is found and concluded that a value close to 4σ is close to optimal. But using 4σ as a threshold has the problem that the sensitivity is also somewhere around 4σ depending on the acceptable error rate. Modifying x_{t} to follow a trajectory over the current history length can dramatically increase the chances of finding a state and at the same time improve sensitivity once it is fully build. This sounds good at first, but there are problems that come with it. If building a state is made easier, this also means building an incorrect state is made easier. Also by changing the single-point solution, the confidence it produces shifts. This shift can give a building state a higher confidence than an existing one, making it easier to create incorrect states. In other words the creation of states will be the preferred method of handling outliers, which is incorrect behaviour. To counteract this, only the decision of keeping a state should be influenced by the dynamic threshold. This has the benefit of making it easier to build a state and at the same

time increasing sensitivity while giving a correct confidence at all times.
Figure 4.13 visualizes the probability of keeping a state depending on the history length for 4σ. To get the total probability of building a state the product of up-to the chosen maximum history length needs to be calculated. This leads to the conclusion that if building a state fails it is more likely for it to fail at the begin of building than at the end.

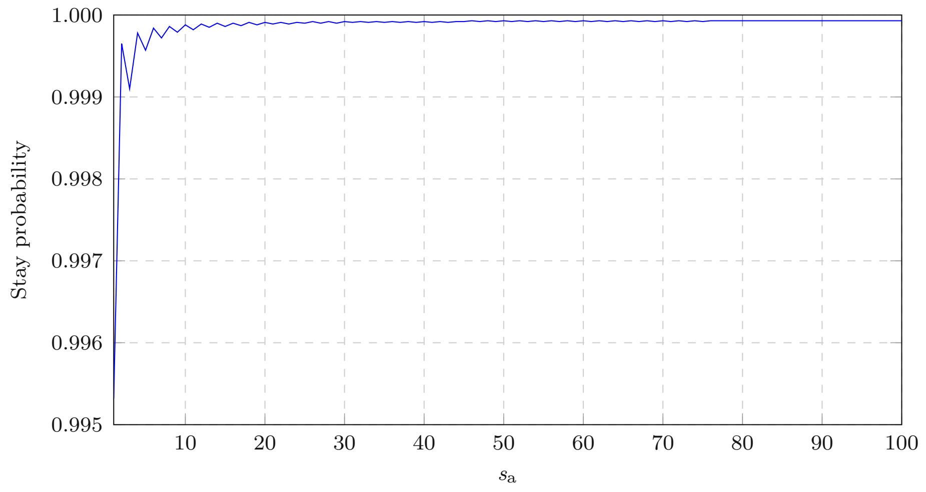


Fig. 4.13: The probability that a state is kept depending on the the current history length for a single-point solution threshold of 4σ.

By using the trajectory shown in Figure 4.14 the probability of finding a state can be increased from 98.52% to 99.77% or even 99.91% if non linear trajectories are allowed. After the state is build it would previously stay at the threshold of 4σ with an error rate of 0.006851%. Reducing the threshold to 3.5σ for an error rate of 0.0492% per sample or to 3σ for an error rate of 0.2799% per sample, would improve the sensitivity. As soon as the state is fully build the error rate is no longer that problematic, as once the algorithm leaves the state it should quickly re-enter it.

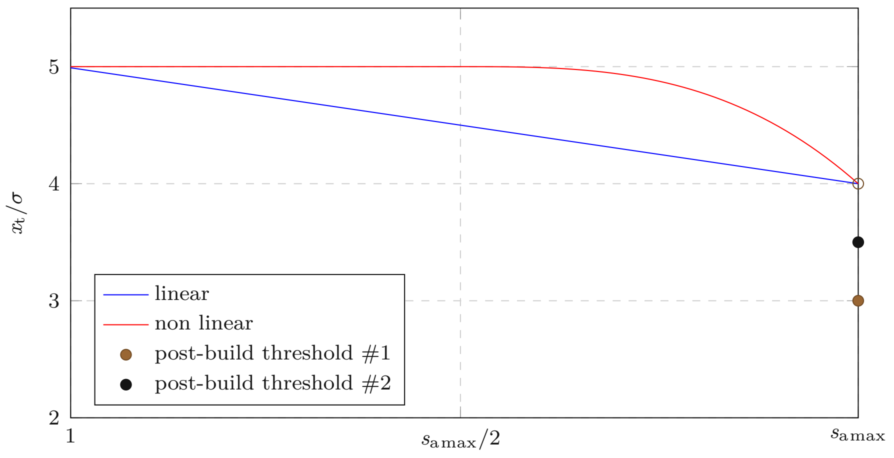
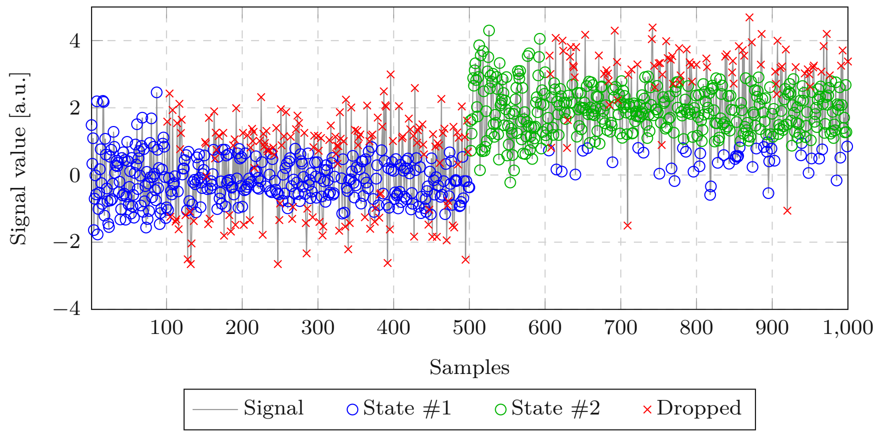
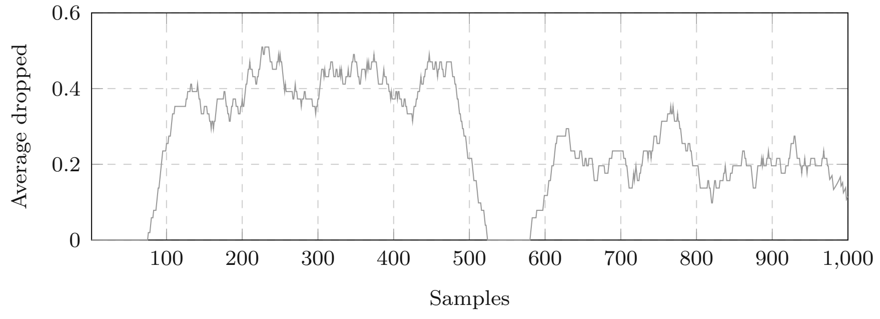


Fig. 4.14: Possible trajectories of the single-point threshold over the history length.

To demonstrate this behaviour as well as the accuracy increase a post-build threshold of 1σ (error rate of 35.1338% per sample) was chosen in Figure 4.15 and a jump-height of 2σ was found, which under normal circumstances is close to impossible. It should be noted that a post-build threshold of 1σ is far to low to be sensible. It demonstrates how much easier it is to create a state using the dynamic threshold than to keep it, as no values are dropped until position 100. The fact that there are values other than 0 in sub-figure (b) prior to position 100 is due to the fact that it is an acausal smoothed curve. Once the first state is fully build, it switches to a single-point solution threshold of 1σ and the errors start occurring. That the error rate is not exactly 0.35 is due to the fact that it is random and has a small window size. At the value of 500 it detects a jump of height 2σ. Again for the next 100 values no drops occur. At the 600 mark the value drops starts happening again, but this time it is sometimes found that the values fit well into the first state, which is highly incorrect. Finding that the first state as a good fit also explains why the average of dropped values is to low in the post-jump region.



(a) signals and states



(b) moving average of dropped values

Fig. 4.15: Using a non-linear dynamic state building single-point solution threshold with a maximum history length of 100, and a post-build single-point solution threshold of 1σ. Sub-figure (a) displays where which state was found and which values were dropped. Sub-figure (b) displays the moving average of dropped values with a window size of 51.

There is a second major problem. Increasing the sensitivity is useless if only a state is lost, a new state has to be successfully build as well. A state is build successfully if and only if all values fit into the new state, which this adaptation already takes care of. The second condition is that the values fit better in this new state than all old states. That a building state fits best is by far not guaranteed. If the sensitivity is increased to such an extent that the values overlap with another state, it can get quite likely that a new state cannot be built, as the values just re-enter the old state. This shifts the old state towards the new states value domain. Once it is shifted far enough it will accept all new values. Figure 4.16 visualizes this phenomena.

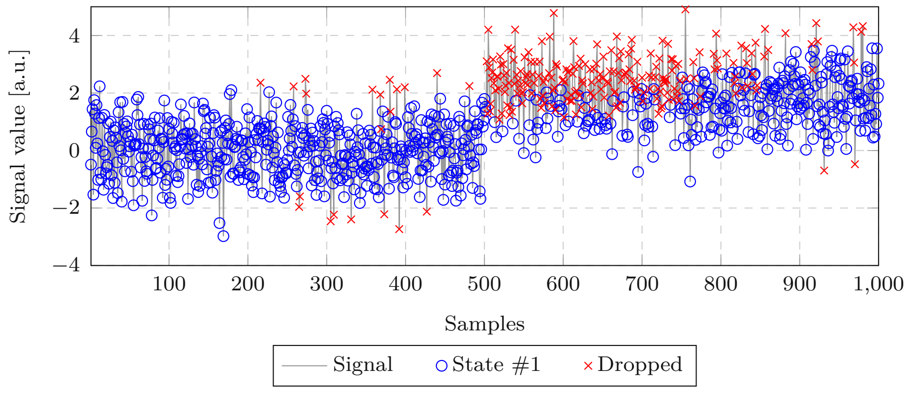


Fig. 4.16: Displays how the sensitivity can be good enough, but a new state can not be build. s_a is 200, a post jump threshold of 2σ.

The algorithm successfully leaves the old state and tries building a new one. Then one value fits into the old state and therefore the algorithm re-enters it, dropping all values of the previously building state. By storing which values were discarded, the algorithm can estimate whether such behaviour is occurring. When such a behaviour occurs, artificially filling the history of a new state increases the probability to fill the history completely before an input value fits better into an old state than the building one. This artificial filling of the history is called phantom history. Figure 4.17 demonstrates the effectiveness of the phantom history on the same data and configuration as Figure 4.16.

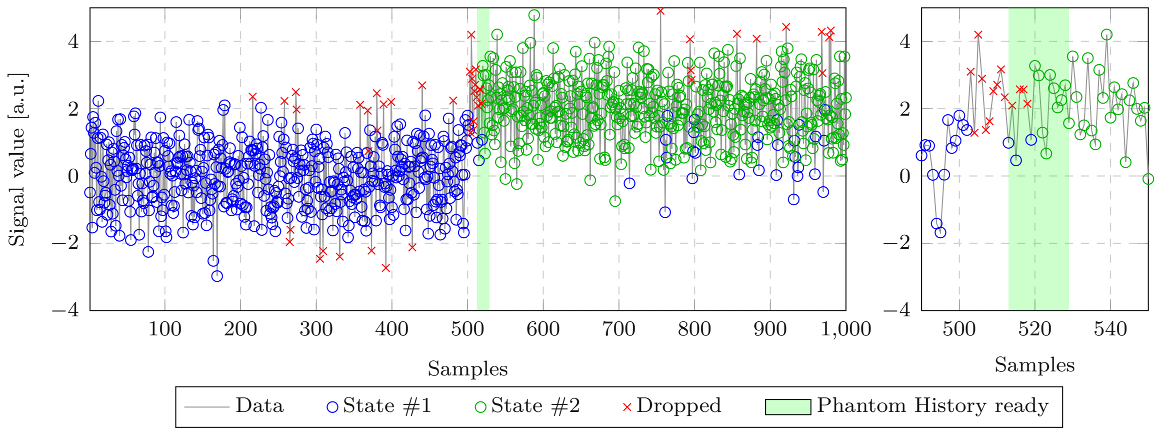


Fig. 4.17: Displays how the phantom history can improve finding a state. s_a is 200, a post jump threshold of 2σ. It uses the same configuration as Figure 4.16 but with the phantom history. "Phantom History ready" means that it will be used when a new state is created.

4.4.1 Definition

There are two things to define in this adaptation.

The Dynamic Value Confidence is quite simple to define. The new definition range is:

$$\mathbb{R}_{0+} \times \mathbb{N} \longrightarrow [0,1]^2 \,. \tag{4.31}$$

The behaviour of the function itself is a little more complicated. Let s_{acurrent} be the current history length and s_{amax} the maximum history length. Let the functions $c(x)$ and $d(x)$ take the place for the the parameters inside fc_{valconf} and $fc_{\text{valcoconf}}$,

$$a(x,b) := \begin{cases} 1 & : x \leq c(b) \\ \frac{d(b)-x}{d(b)-c(b)} & : c(b) < x < d(b) \\ 0 & : d(b) \leq x \end{cases} \,, \tag{4.32}$$

$$fc_{\text{valconf adaptive}}(x, s_{\text{acurrent}}) = \{a(x, s_{\text{acurrent}}), a(x, s_{\text{amax}})\} \,, \tag{4.33}$$

$$a(x,b) := \begin{cases} 0 & : x \leq c(b) \\ \frac{x-c(b)}{d(b)-c(b)} & : c(b) < x < d(b) \\ 1 & : d(b) \leq x \end{cases} \,, \tag{4.34}$$

$$fc_{\text{valcoconf adaptive}}(x, s_{\text{acurrent}}) = \{a(x, s_{\text{acurrent}}), a(x, s_{\text{amax}})\} \,. \tag{4.35}$$

The functions $c(x)$ and $d(x)$ define how the threshold behaves over s_{a}. The first return value of $fc_{\text{valconf adaptive}}$ and $fc_{\text{valcoconf adaptive}}$ shall be used to determine whether a state is kept and the second one for all confidence purposes other than keeping a state.

The Phantom history shall be a history that contains $s_{\text{amax}}/10$ of the most recent input values. These values shall be marked if that value either caused a change of state or was part of a dropped state that was smaller than $s_{\text{amax}}/10$. If more than 50% of the phantom history is marked and a new state would be created, it will change the way the state is created. This change basically violates the boundaries of what belongs to a state. It takes a number of most recent values equal to the number of marked values inside the phantom history. It should be noted that because only the number of marked values is used and not the marked values itself, it is likely that the values which are used are part of a fully build state. This is intentional and desirable behaviour, as it represents the new state better than only the dropped values could.

4.4.2 Adaptation of the Step-detector

Incorporating this adaptation into the algorithm is a bit more tricky. The two adaptations that have to be made are modifying confidence calculation/keeping a state and adding a Phantom history to modify building a state.

Adapting Confidence and keeping a state requires that $f_{\mathcal{C}_{\text{valconf}}}$ and $f_{\mathcal{C}_{\text{valcoconf}}}$ return twice the amount of values. One set for the purpose of keeping a state and the second set for calculation of confidence. To make it easy to use, the algorithm should be capable of handling a function pair which does not use this adaptation.

The Phantom history requires multiple changes to be implemented. First, a history array with additional information has to be added, to define whether that history value has caused a state change or belonged to an unfinished state that was dropped due to an old better fitting state. Second, a counter to has to be implemented to count the current length of the building state. This is necessary because the history of the building state is no longer identical to the amount of time that the state was active. This counter is increased each time a value is accepted into the building state. When the state is dropped the counter is used to mark that amount of most recent values in the phantom history. The counter is reset when a new state is created and it is ignored as soon as a state is no longer compared to old states. The third change is when a new state is initialized, the phantom history has to be examined and if more than half of it is marked, count the amount of marked values and use that amount to extend the history of that newly created state by this amount of most recent values. This extension of state history will include values claimed by old states and values claimed by dropped states preceding it. The phantom history has the length of $s_{\text{amax}}/10$. This is the same amount that is required by a new state to leave the active comparison phase. In other words it will shorten or even remove the comparisons to older states.

4.4.3 Analysis of the adaptation

The adaptation has two main focuses, building a state more reliable and improving sensitivity. There are four aspects to analyse:

1. How does the adaptation affect building a state?
2. How does the adaptation affect sensitivity?
3. How does the adaptation affect drifting signals?
4. How does the adaptation handle old good fitting states?

Building a state is one of the main aspects of the adaptation. As already discussed the likelihood that a state is build is higher, but a likelihood of 98.52% is not bad to start with. It gets important once we accept that $f_{C_{\text{valconf}}}$ and $f_{C_{\text{valcoconf}}}$ might not be ideally defined. Variations in the threshold have an effect on the likelihood. Figure 4.18 displays the effect of the likelihood of finding a state with a mismatch of σ between the signal and $f_{C_{\text{valconf}}}$ and $f_{C_{\text{valcoconf}}}$. With a 10% error in sigma the chance of finding a state drops to around 95% and with a 20% error it would drop to around 87%. At the same time using linear or non linear trajectories for the threshold with an error of 20% would drop to around 96% or 98% respectively. Such an approach would increase parameter stability, since small errors in the definition of the parameters would have even less impact.

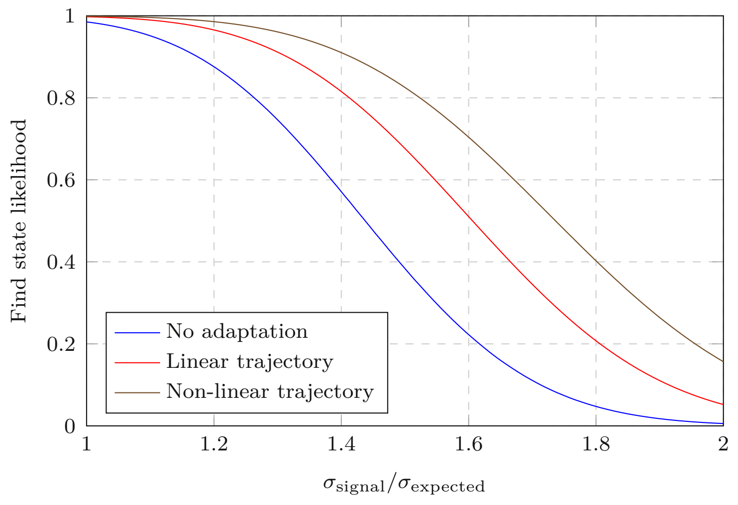


Fig. 4.18: The likelihood that a state is found even when σ of a signal is different than the σ $f_{C_{\text{valconf}}}$ and $f_{C_{\text{valcoconf}}}$ were defined for. The chosen s_a is 100.

Sensitivity is the second main aspect of the adaptation. Figure 4.19 visualizes the mathematical probability that an old state is lost at any point and that enough values are outside the old state to bring a new one beyond the comparison phase, depending on step height, for s_a being 40. It should be noted that the calculation time of the algorithm used seemed to be factorial with s_a. The probability that it is inside or outside the old state changes depending on the amount of previous values belonging to the old state. The calculation does not take the confidence comparison between building and old state into account. In other words the true probability should be between the solid line and the dashed line, for each colour.

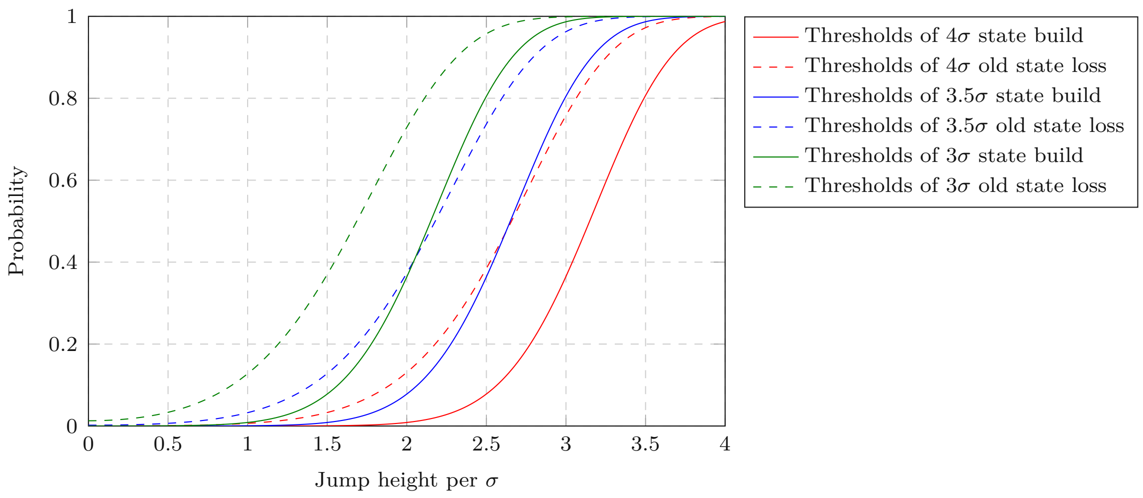


Fig. 4.19: The probability that a state is lost and the probability that enough values are not kept to build a new state beyond the comparison phase depending on the step height.

Drifting Signals are of major importance. It is to be expected that this adaptation should reduce drift acceptance. Figure 4.20 shows this behaviour. It is clear that if this extension is wanted a form of drift removal is required.

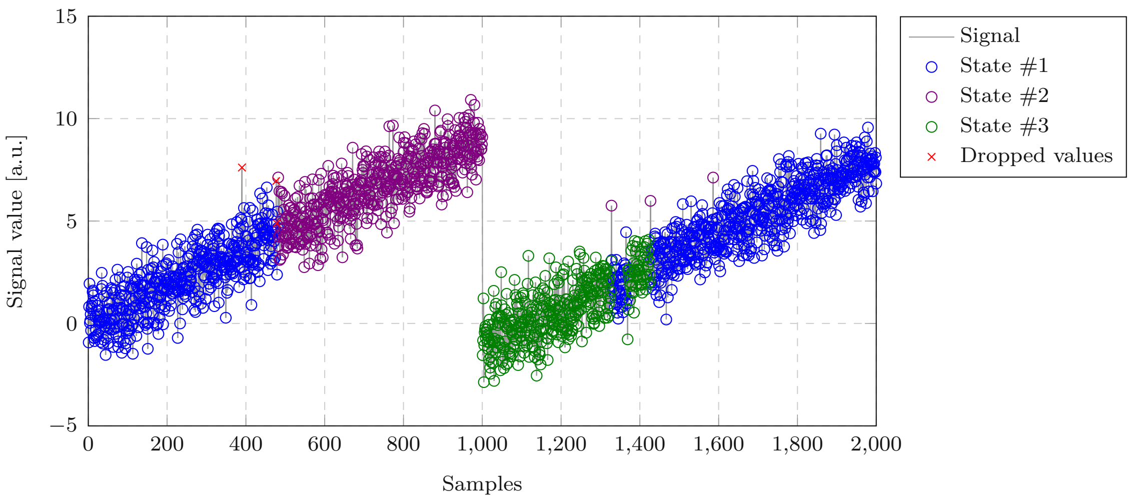


Fig. 4.20: Multiple states are created due to the drift. The drift has a strength of 0.009σ per value. The jump height is -10σ and an appropriate single-point solution threshold of 4.9σ was chosen. It should be noted that the standard algorithm was able to solve this without any problems. A post-build threshold of 3.675σ $(3/4 \cdot 4.9)$ was chosen.

Handling multiple good fitting states comes into focus because of the higher probability that a state loss occurs. Figure 4.21 demonstrates what can happen. A special focus is in the point around 7500 where the state changes only because of the noise. This problem has to do with the way old states are handled. Proximity of a state has no influence on the probability of it. This is a problem with the algorithm that is more common with this extension.

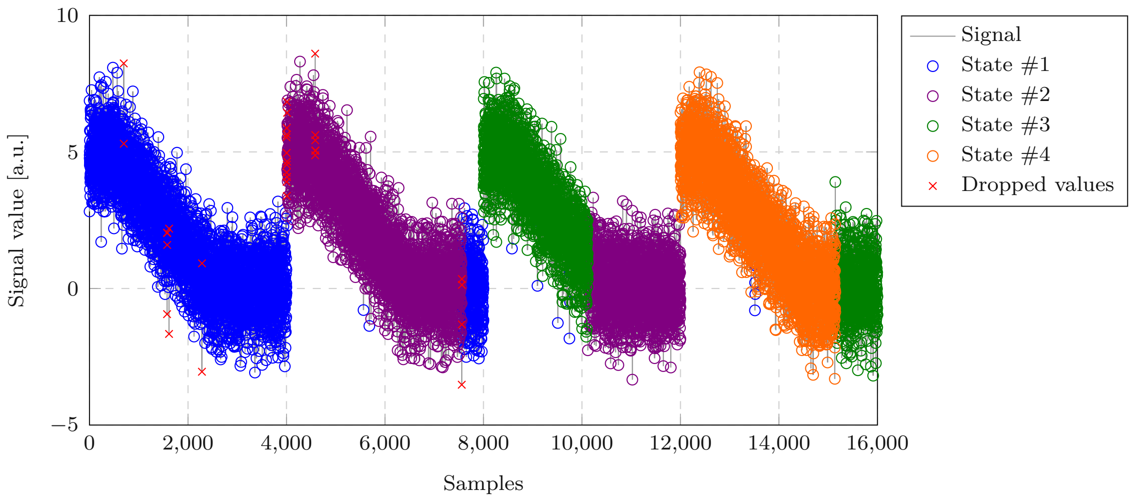


Fig. 4.21: A signal with a dominating pattern that ends around the same value. A single-point solution threshold of 4.6σ was chosen. A post-build threshold of 3.45σ $(3/4 \cdot 4.6)$ was chosen.

4.4.4 Summary and possible Improvements

The introduced adaptations were designed to increase sensitivity and the probability that a state is built. They accomplish the intended purpose. However, the analysis is more on the negative side about the sensitivity increase, as it comes with negative aspects. These negative aspects are less drift resistance and a higher likelihood of false state changes due to old states. The drift resistance could be mitigated by using the Integrated Preprocessing. Mitigating the problem of re-entering old states could be done by limiting which states can be entered, checking if states are equivalent/ indistinguishable and then disabling/deleting them, or by giving states in proximity the edge on re-entering them. The phantom history adaptation is only required if the desired state is partly within an old state. The usefulness of it is questionable, if such a sensitivity is not explicitly desired.

4.5 Runtime Adaptation

The problem of having incorrect parameters is twofold. Using parameters that are too big will reduces sensitivity, while having parameters that are too small will reduce the possibility that a state is being found at all. By using an algorithm that extracts parameter information at runtime such problems should be reduced. Making a full statistical analysis with all already occurred values would be very resource intensive. So using only local and low resource algorithms would be preferable. Using the algorithm from Automatic Parameter Detection only on the current Δ and feeding these values into a low-pass filter might achieve the desired effect. Automatic Parameter Detection in short calculates the 4σ threshold value by extrapolating it from two quantile values. The question arises as to how stable that would be and whether drifts could be a problem.

4.5.1 Definition

There are three things to define. First, the parameter extraction from the current history. Second, the used low-pass filter; And third, the calculation of fc_{valconf} and $fc_{\text{valcoconf}}$.

Parameter extraction from the current history can be done surprisingly easy. As already mentioned the parameter extraction shall be done exactly as in Automatic Parameter Detection:

$$a_{\text{extracted}} := \text{Quantile}(\Delta, 0.9) \tag{4.36}$$
$$b_{\text{extracted}} := a_{\text{extracted}} + 7.6 \cdot (\text{Quantile}(\Delta, 0.95) - a_{\text{extracted}}) \tag{4.37}$$

The low-pass filter is a bit more tricky. It shall force changes to occur slowly. It shall take multiple full history lengths to apply them fully. A FIR low-pass would be of a high order, which is undesired. This leaves only IIR low-pass approaches. Using an IIR low-pass of first order can be enough. As already mentioned EWMA is a very light weight and highly effective low-pass filter. If applied to a step it approaches the steady value in the manner of:

$$V \cdot \left(1 - e^{-\frac{t}{\tau}}\right) \tag{4.38}$$

V being the height of the step. As defined, it shall take multiple history lengths to get close to V so τ should roughly be s_a. For the IIR low-pass this means that it can be defined as:

$$y_i = y_{i-1} \cdot \left(1 - \frac{1}{s_\mathrm{a}}\right) + \frac{1}{s_\mathrm{a}} \cdot x_i \tag{4.39}$$

x being the input and y being the output. It should be noted that it is only an approximation between the exponential equation and the discrete equation. The error between them is $\sum_{i=2}^{\inf} 1/(i! \cdot (-s_\mathrm{a})^i)$ simply calculable by Taylor approximation.
A low-pass shall exist for all parameters.

The calculation of $f_{\mathcal{C}_\mathrm{valconf}}$ and $f_{\mathcal{C}_\mathrm{valcoconf}}$ is not hard to do, but hard to define. Using Lambda calculus[4] simplifies the definition, but would make it harder to read. So the middle way was chosen, which was first defined in a more classical mathematical sense, which should be understandable and then defining it using the lambda calculus. Let $f_{f\mathcal{C}_\mathrm{valconf}}$ be a function which returns a confidence function as an answer:

$$\begin{aligned} f_{f\mathcal{C}_\mathrm{valconf}} :\ & \mathbb{R}^2 \longrightarrow (\mathbb{R} \to [0,1]) \\ & (a,b) \mapsto \left(x \mapsto \begin{cases} 1 & : x \leq a \\ \frac{b-x}{b-a} & : a < x < b \\ 0 & : b \leq x \end{cases} \right) \end{aligned} \tag{4.40}$$

$$f_{f\mathcal{C}_\mathrm{valconf}} = \lambda a. \left(\lambda b. \left(\begin{cases} 1 & : x \leq a \\ \frac{b-x}{b-a} & : a < x < b \\ 0 & : b \leq x \end{cases} \right) \right) \tag{4.41}$$

Let $f_{f\mathcal{C}_\mathrm{valcoconf}}$ be the function which returns the co-confidence function,

$$\begin{aligned} f_{f\mathcal{C}_\mathrm{valcoconf}} :\ & \mathbb{R}^2 \longrightarrow (\mathbb{R} \to [0,1]) \\ & (a,b) \mapsto \left(x \mapsto \begin{cases} 0 & : x \leq a \\ \frac{x-a}{b-a} & : a < x < b \\ 1 & : b \leq x \end{cases} \right) \end{aligned}, \tag{4.42}$$

$$f_{f\mathcal{C}_\mathrm{valcoconf}} = \lambda a. \left(\lambda b. \left(\begin{cases} 0 & : x \leq a \\ \frac{x-a}{b-a} & : a < x < b \\ 1 & : b \leq x \end{cases} \right) \right). \tag{4.43}$$

[4] `https://en.wikipedia.org/wiki/Lambda_calculus`

4.5.2 Adaptation of the Step-detector

Thanks to the definition the adaptation of the algorithm is quite easy. Feeding the sorted Δ into the parameter extractor and those results into the low-pass filter. The results of the low-pass are then fed into the lambda expressions which return a set of functions used for value confidence and co-confidence. These functions simply replace the predefined confidence and co-confidence functions.
There are two possibilities of updating the functions either only on fully build states or at all times regardless of state. Both of these versions are looked at.
It should be noted that the updates of the parameters is only allowed to occur on the comparison with the active state. On comparisons with inactive states, updating the parameters are not allowed.

4.5.3 Analysis of the Adaptation

There are multiple aspects to check that might be problematic. The following behaviours are analysed:

- How do the parameters change for a steady signal?
- How do the parameters change for a drifting signal and a suddenly drifting signal?
- How does the adaptation influence building a state?
- How do the algorithms behave if the initial parameters are to small?

Analysing all aspects for both versions does not make sense. E.q. updating at all times and only updating when a state is fully build can only create differences if a state is not fully build. Also if no state can be build fully initially, it does not make sense to check the behaviour of the version that only updates when a state is fully build.

Steady Signal should not be affected by this extension. That however is not the case. The problem with simple statistical methods is that if the data set is not big enough they get inaccurate. This is also the case for this algorithm. Figure 4.22 displays this problem. The desired output would be a steady line at 4. It is quite interesting because the prevailing range is 3-4. Reducing x_t was one of the main goals of Dynamic Value Confidences and Phantom History. It had a problem with drifts.

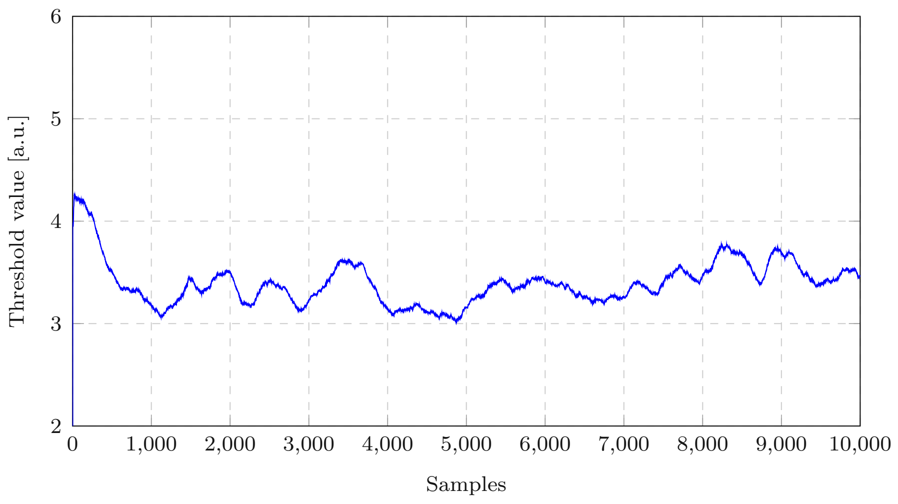


Fig. 4.22: Single-point solution value threshold over a signal with 10000 samples and a noise of 1 and an s_a of 200. It should be noted that the step-detector defined 2 samples, as not part of the state. It should be noted that in this case the difference between both versions are so miniscule that only one is shown.

Drifting Signals are of major importance for this algorithm. The adaptation has shown in the stead signal test that it reduces the single-point solution threshold. A similar behaviour can be seen in this example. It is lower than the solution from Automatic Parameter Detection, but it exhibits an intended behaviour. Roughly speaking the threshold should be "threshold of drift free signal" + "drift strength" $\cdot s_\mathrm{a}/2$. In other words if it works correctly the threshold for a drift strength of 0.01 per sample and an s_a of 200 should be 4-5. Figure 4.23 displays this trait. Figure 4.24 again displays this trait with a stronger drift. It is also the first to show a truly noticeable difference between both versions. The version which only update on fully build states can not create a fully build state initially, therefore it is doomed to repeat to try to build a state again and to fail again. But there is a chance that a state is build and once it happens the single point-solutions threshold practically jumps.

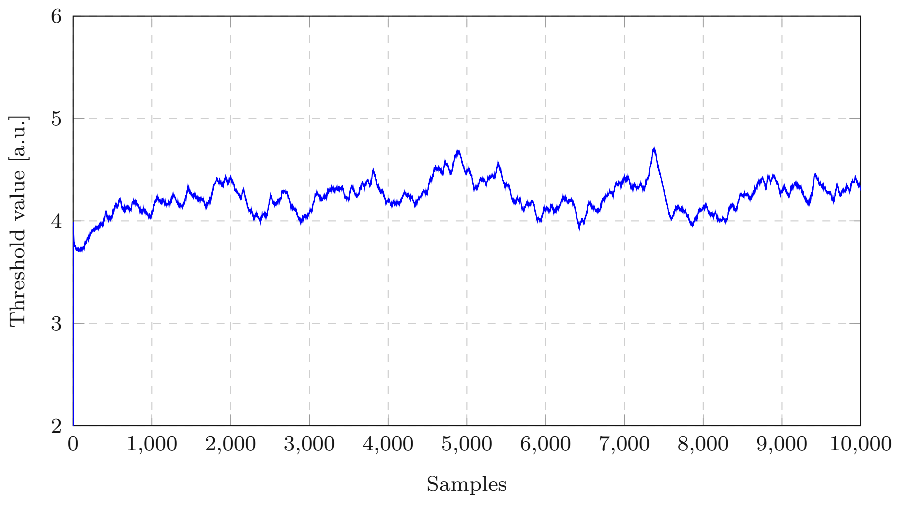


Fig. 4.23: Single-point solution value threshold over a signal with 10000 samples and a noise of 1, a drift strength of 0.01 per sample and an s_a of 200. It should be noted that in this case the difference between both versions are so miniscule that only one is shown.

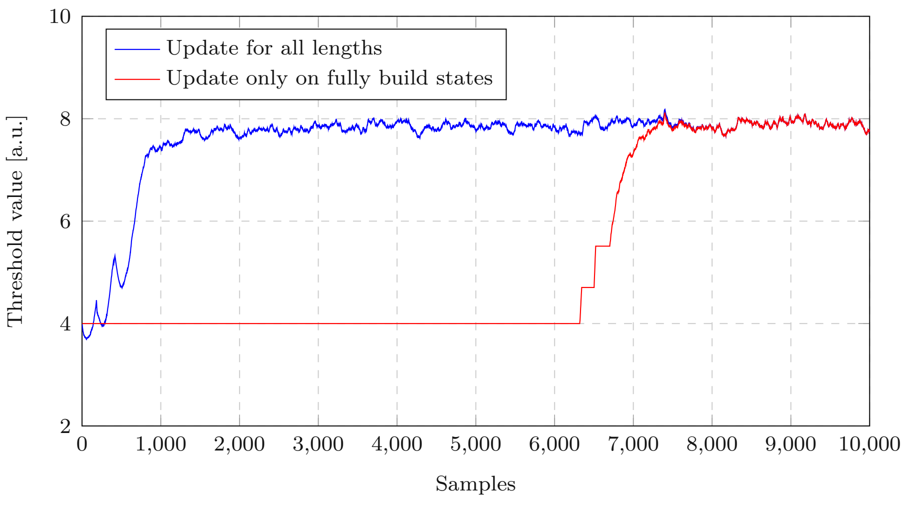


Fig. 4.24: Single-point solution value threshold over a signal with 10000 samples and a noise of 1, a drift strength of 0.03 per sample and an s_a of 200. It should be noted that the initial parameters are to low and that states are dropped at the start of the signal.

Figure 4.25 displays what happens when the signal suddenly begins to drift. The adaptation can handle it, although the single-point solution was reduced to less than 4 pre drift. The interesting aspect is that it seems to be more reactive once its drifting, but that is incorrect. The values that are being "flattened" by the IIR low-pass are bigger, because the drift is part of the distribution and therefore the sorting of Δ, which normally would also act as a sort of smoothing filter, gets partly disabled, which gives the impression that it is more reactive.

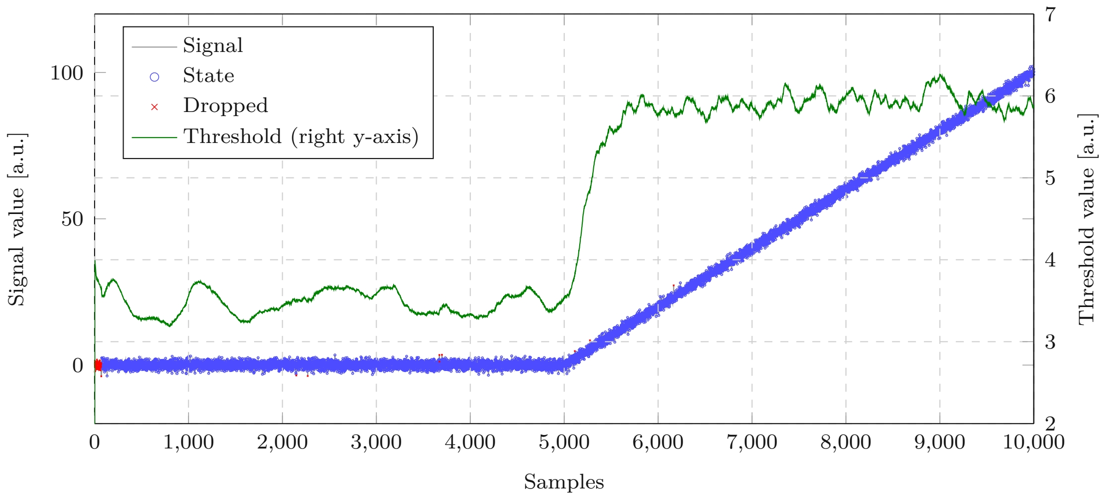


Fig. 4.25: A signal with a sudden drift of 0.02σ and the currently used/calculated threshold on the right y-axis. An s_a of 200 was used. It should be noted that the difference between both versions are so miniscule that only one is shown.

Building a State is interesting with these adaptations. By default, the thresholds gets anchored to around 3.5σ, which has a significantly lower change to create a state than 4σ would have. Figure 4.26 displays this negative effect. Setting s_a to 3000 can enforce the problem to some extent. It can find a state at the beginning while the threshold is high but once it has reached the 3.5-4 domain and the jump occurs it has a problem. It should be noted that there is a chance that this problem could occur but under normal circumstances it is unlikely. Solving this problem is not easily done. Using the phantom history and a dynamic building threshold from Dynamic Value Confidences and Phantom History would probably be beneficial. If they are adjusted to a default threshold value of 3.5σ, it could negate the problem. Note the threshold jumps at 3 points in the figure. This is due to the values being outlies and by chance are also quite far apart, resulting in an huge value, which due to the nature of the filter gets passed on to the threshold. Also it was only tested for the variation which updates at all times.

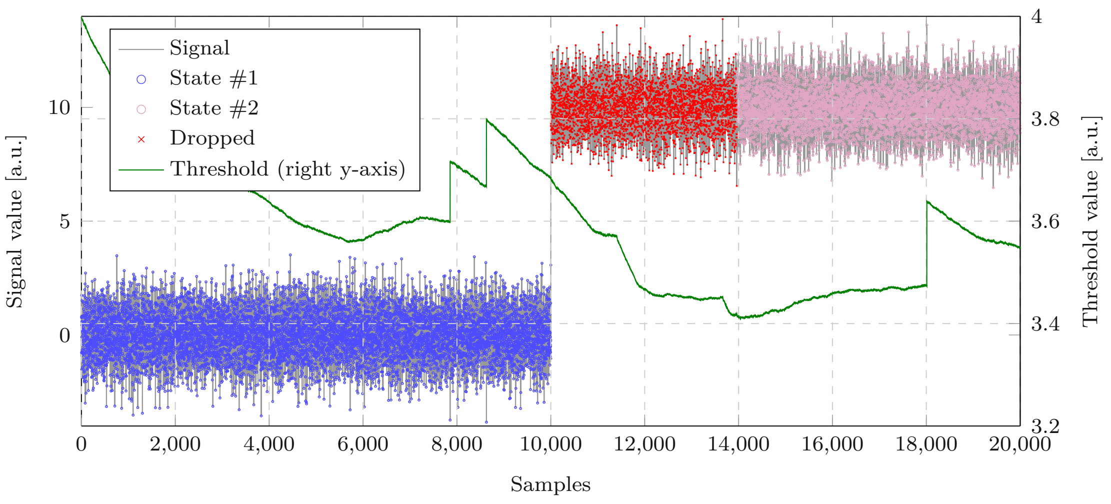


Fig. 4.26: A signal with a jump which is being segmented. s_a is 3000.

Incorrect parameters is what this adaptation aims to improve, but having to small parameters might be a problem as no state can be formed. It is clear that the version that only updates on fully build states can not solve this problem. Figure 4.27 shows how the adaptation behaves when the initial parameters are 0.

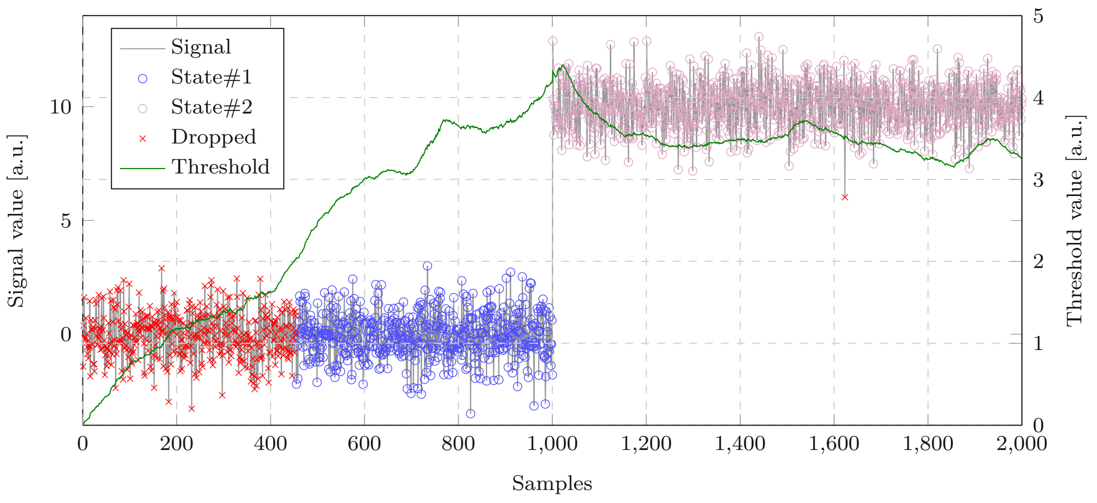


Fig. 4.27: The algorithm starts with a single-point solution threshold of 0. s_a is 100.

There is more to analyse inside the adaptation. As mentioned inside the Definition an IIR low-pass exists for all parameters. Viewing the behaviours of the individual parameters reveals what happens internally. Figure 4.28 displays this. As a reminder the definitions of the fc_{valconf} and $fc_{\text{valcoconf}}$:

$$fc_{\text{valconf}} : \mathbb{R}_{0+} \longrightarrow [0,1]$$

$$fc_{\text{valconf}}(x) = \begin{cases} 1 & : x \le a \\ \frac{b-x}{b-a} & : a < x < b \\ 0 & : x \ge b \end{cases}, \tag{4.44}$$

$$fc_{\text{valcoconf}} := 1 - fc_{\text{valconf}} . \tag{4.45}$$

Figure 4.28 displays the behaviours of the internal parameters. Up until about 200 they are identical, after which small temporary states are formed. This allows the adaptation to differentiate between the quantiles used, which is sufficient to further increase the size of the temporary states. At around 400 the threshold is big enough to create a state of size 50, which further differentiates the quantiles enabling b to increase more quickly. At around 450 the last temporary state is dropped, but another big state is build and because the threshold increases fast enough it can be fully build.

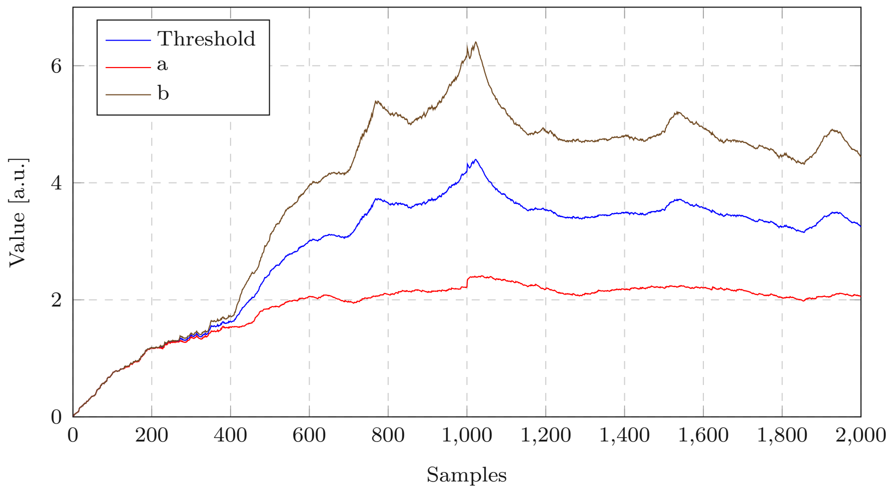


Fig. 4.28: Same data as Figure 4.27, but showing internal behaviour. Around 450 the first permanent state is found.

4.5.4 Summary and possible Improvements

The adaptation aimed to fit fc_{valconf} and $fc_{\text{valcoconf}}$ to the signal during the runtime. It achieves it better than expected. It can not quite reach the desired threshold value and stays to low. This makes it harder for it to find a state. Using Dynamic Value Confidences and Phantom History could help out, but it should not modify the post build threshold. It is capable of finding a good solution even for drifting signals and even if the initial parameters are useless. It seems that the version that updates parameters regardless of state is superior. Looking at this extension from the standpoint of resource consumption, it turns out to be surprisingly cheap. It requires array accesses and basic mathematical operators. The values do not need to be sorted as they already are from the algorithm.

4.6 Combination of useful Adaptations

Three algorithms that modify the step-detector have been demonstrated. They were shown in isolation to demonstrate their effect on the behaviour of the step-detector. Merging them together into a unified adaptation is the last act of this chapter. Some of the adaptations have weaknesses that are strengths of others. These strengths should be achieved in positions that make later weaknesses irrelevant. For example, the weakness that the drifts make it harder to keep a state for a to small post build threshold is irrelevant if the drift was compensated for by the Runtime Adaptation.
This section does not have an analysis nor a summery or possible improvements as they are done in the following chapters.

Integrated Preprocessing is taken in its entirety. The used algorithm to approximate the incline is ζ_{SGolay}.

From Dynamic Value Confidences and Phantom History the phantom history and the increased state building threshold are used. The post build threshold is unused because it would collide with the Runtime Adaptation. This part should make it significantly easier to build a state.

The whole Runtime Adaptation is used. It actually has a synergy with the phantom history. When the phantom history is activated, the history of a state is artificially filled, which makes it possible to calculate a better threshold.

4.6.1 Algorithm Steps

In this subsection the steps that the modified step-detector takes are roughly explained, as otherwise it would be scattered through out the thesis. The steps are explained in an abstract manner. With an precise definition of how certain parameters or steps are calculated, the steps shown in this subsection are sufficient to fully define the algorithm. It is a simplification to make the whole algorithm more tangible. It starts with necessary data structures that exist inside the algorithm. It uses a simplified flowchart extended by an overlay of special interactions and the whole chart is explained. In the next subsection the mathematical definitions are displayed.

The Required Data Structures and their internal parts are the following:

• State	
– Id	
– Dropped	*Stores if that state was dropped*
– History[]	*Array containing the states old input values*
• PhantomHistory	
– History[]	*Array containing old input values*
– Dropped[]	*Array containing if that value is marked*
• OldStates[]	*Array containing all old states*

The Required Functions for the extensions and their respective domains:

- IntegratedPreprocessing:
$$\mathbb{R}^a \times \mathbb{R} \longrightarrow \mathbb{R}^a_{0+} \tag{4.46}$$
The function removes an assumed drift from the history and input value, then subtracts the input value from the history and makes the absolute of these values. Afterwards they are sorted ascendingly.

- RuntimeAdaptation:
  - Parameter extraction and smoothing:
$$\mathbb{R}^a_{0+} \times \mathbb{R}^b \longrightarrow \mathbb{R}^b \tag{4.47}$$
    The function takes the difference and the parameters from the previous cycle and returns updated and smoothed ones.
  - Function creation:
$$\mathbb{R}^b \longrightarrow \left(\mathbb{R}^a_{0+} \to [0,1]^{a\times 2}\right)^2 \tag{4.48}$$

> The function takes parameters and creates $f_{\mathcal{C}_{\text{valconf}}}$ and $f_{\mathcal{C}_{\text{valcoconf}}}$. The generated functions produce two sets of confidences, one for keeping a state and the other for the confidence used. This can create the behaviour of keeping a state even if the confidence is low.

It should be noted that from a programming perspective, these two functions can be masked within one function and the parameters can be stored internally. This would simplify the implementation and if this combination would be passed as a function or functor it would make the whole algorithm easily configurable.

The Simplified Flowchart is displayed in Figure 4.29. It also shows the extensions, their positions and special data interactions.

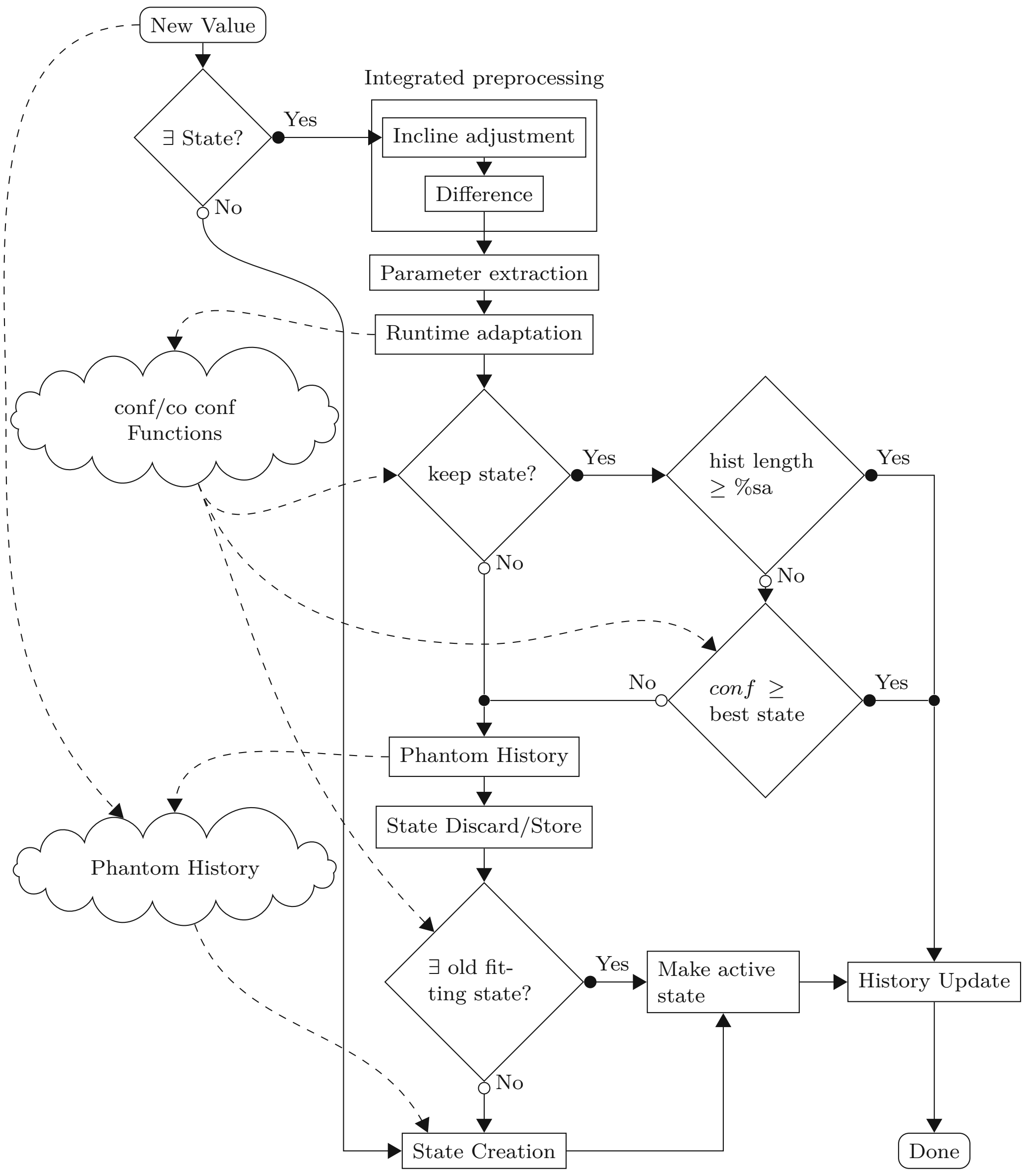


Fig. 4.29: Simplified flow chart of the algorithm, extended by special data interactions represented by clouds and dashed lines.

Each new value is pushed into the phantom history, shifting the oldest value out. The length of the phantom history is $\lfloor s_a/10 \rfloor$. A zero/false is pushed into the phantom history dropped array, pushing the oldest value out as well.
If no current state exists a new one is created and the history of the state is updated. If a current state exists the new value and the history of the current state get fed into the integrated preprocessing function creating a difference array. From the difference

the parameters for the confidence and co-confidence functions are extracted. These parameters (or initial values) are then passed to the runtime adaptation to create $f_{\mathcal{C}_{\mathrm{valconf}}}$ and $f_{\mathcal{C}_{\mathrm{valcoconf}}}$. These functions get stored, which is represented by the dashed line. In combination with the defined $f_{\mathcal{C}_{\mathrm{saconf}}}$ and $f_{\mathcal{C}_{\mathrm{sacoconf}}}$ it is calculated if the state is kept by using the dynamic aspects of these functions and the confidence is calculated by using the static aspects of those functions.
If the state is kept and it is a fully build state, the history is updated and the algorithm returns the states-id and the confidence.
If the state is kept, but it is not a fully build state the current progress of building the state is checked (the current length of the history) and if it is large enough the history is updated and the algorithm returns the states-id and the confidence.
If the progress of building the state is not far enough, the input value is used to calculate the confidences of all fully build states. If there is no state better fitting than the current one, the history is updated and the algorithm returns the states-id and the confidence. However, if there is a better fitting state than the current state, the length of the current history is used to mark that amount of resent phantom history values, then the state is discarded and the better fitting one is made active, which then updates its history and the algorithm returns the states-id and the confidence.
If the state is not kept the algorithm differentiates between a fully build state and a building one. If it is a fully build state the phantom history marks the most recent history value, afterwards the state is stored. If the state is currently building the length of the current history is used to mark that amount of resent phantom history values and then the state is discarded.
Afterwards it is checked if a better fitting state exists.
If a better fitting state exists it is made active, then the history is updated and the algorithm returns the states-id and the confidence.
If no better fitting state exists a new state is created. If at least 50% of the phantom history is marked, the history of the created state gets filled with the amount of dropped values going back from the most recent one. If this is triggered it will most likely also include values belonging to other states, this is intended. If not enough recent values were dropped only the new input value gets pushed into the history.
Afterwards the algorithm returns the states-id and the confidence.
Updating the history means adding the new input value to the history and if the state is currently building it just extends the history. If the state building is finished the oldest value gets pushed out.

4.6.2 Definition

The full definition of the mathematical functions of the modified step-detector is shown in this subsection. It is shown in its entirety, repeating parts of 3.1.1 Definition. The full definition is given for consistency.

Let s_a be the sample amount. So far it has been used in a dual state of being the current and the maximal sample amount. In this definition the current version is called s_a and the maximum is called $s_{\mathrm{a}\max}$.

Let $\mathcal{H}$ be the history of the current state and v the new input value.

Let f_diff be a function which takes the current states history and the new input value and calculates a drift-removed sorted absolute difference from this,

$$f_\mathrm{diff} : \mathbb{R}^a \times \mathbb{R} \longrightarrow \mathbb{R}^a_{0+} \, . \tag{4.49}$$

There are multiple ways of doing this. The variation chosen for this extension is:

$$\begin{aligned}
q :=& \max(3, \lfloor s_\mathrm{a}/4 \rfloor) && (4.50)\\
r :=& q + \mathrm{mod}(q+1, 2) && (4.51)\\
\mathcal{S}_\mathrm{sgolay} :=& \text{ Savitzky-Golay-Filter}(\text{Order} = 1, \text{Derivative} = 1, \text{size} = r) && (4.52)\\
s1 :=& \left\lfloor \frac{s_\mathrm{a} - 2 \cdot q}{2} \right\rfloor && (4.53)\\
s2 :=& \left\lceil \frac{s_\mathrm{a} - 2 \cdot q}{2} \right\rceil && (4.54)\\
\mathcal{S}_\mathrm{summean} :=& \left[\; 0^{1\times s1}, 1^{1\times 2\cdot q}, 0^{1\times s2} \; \right] / (2 \cdot q) && (4.55)\\
\mathcal{S}_\mathrm{SgolayMean} :=& \mathcal{S}_\mathrm{sgolay} * \mathcal{S}_\mathrm{summean} && (4.56)\\
& \text{forced to the size of } \mathcal{S}_\mathrm{summean}
\end{aligned}$$

$$f_\mathrm{drift}(\mathcal{H}, v) = \left| (v - (s_\mathrm{a}+1) \cdot \mathcal{S}_\mathrm{SgolayMean} \cdot \mathcal{H}) - \left(\mathcal{H} - \begin{bmatrix} 1 \\ 2 \\ \vdots \\ s_\mathrm{a} \end{bmatrix} \cdot (\mathcal{S}_\mathrm{SgolayMean} \cdot \mathcal{H}) \right) \right| \tag{4.57}$$

$$f_\mathrm{used}(\mathcal{H}, v) = \begin{cases} |v - \mathcal{H}| & : s_\mathrm{a} \leq 12 \\ |v - \mathcal{H}| \cdot (17 - s_\mathrm{a})/5 + f_\mathrm{drift}(\mathcal{H}, v) \cdot (s_\mathrm{a} - 12)/5 & : 12 < s_\mathrm{a} < 17 \\ f_\mathrm{drift}(\mathcal{H}, v) & : s_\mathrm{a} > 17 \end{cases} \tag{4.58}$$

$$f_\mathrm{diff}(\mathcal{H}, v) = \mathrm{sort}\left(f_\mathrm{used}(\mathcal{H}, v), \text{'ascending'}\right) \tag{4.59}$$

Let Δ be the result of f_{diff},

$$\Delta := f_{\text{diff}}(\mathcal{H}, v)\,. \tag{4.60}$$

Let $f_{\text{parameter}}$ be a function which uses confidence and co-confidence parameters and the current states history to calculate new parameters,

$$f_{\text{parameter}} : \mathbb{R}^{s_{\text{a}}} \times \mathbb{R}^{b} \longrightarrow \mathbb{R}^{b}\,. \tag{4.61}$$

Again there are multiple ways to calculate this and the chosen one is:

$$f_{\text{filter}}(x_{\text{new}}, x_{\text{old}}) = x_{\text{old}} \cdot \left(1 - \frac{1}{s_{\text{a}}}\right) + \frac{1}{s_{\text{a}}} \cdot x_{\text{new}} \tag{4.62}$$

$$a_{\text{extracted}} := \text{Quantile}(\Delta, 0.9) \tag{4.63}$$

$$b_{\text{extracted}} := a_{\text{extracted}} + 7.6 \cdot (\text{Quantile}(\Delta, 0.95) - a_{\text{extracted}}) \tag{4.64}$$

$$f_{\text{parameter}}(\Delta, [a, b]) = \{f_{\text{filter}}(a_{\text{extracted}}, a), f_{\text{filter}}(b_{\text{extracted}}, b)\} \tag{4.65}$$

Let $f_{fc_{\text{valconf}}}$ and $f_{fc_{\text{valcoconf}}}$ be functions which generate functions,

$$f_{fc_{\text{valconf}}} : \mathbb{R}^{b} \longrightarrow \left(\mathbb{R}^{a}_{0+} \to [0,1]^{a\times 2}\right)\,, \tag{4.66}$$

$$f_{fc_{\text{valcoconf}}} : \mathbb{R}^{b} \longrightarrow \left(\mathbb{R}^{a}_{0+} \to [0,1]^{a\times 2}\right)\,. \tag{4.67}$$

There are multiple ways to define these functions, the chosen ones are:

$$f_{\text{Shape1}}(\mathcal{X}, a, b) = \left\{ \begin{array}{ll} 1 & : x \le a \\ \frac{b-x}{b-a} & : a < x < b \\ 0 & : b \le x \end{array} \right| \forall x \in \mathcal{X} \tag{4.68}$$

$$f_{\text{Shape2}}(\mathcal{X}, a, b) = \left\{ \begin{array}{ll} 0 & : x \le a \\ \frac{x-a}{b-a} & : a < x < b \\ 1 & : b \le x \end{array} \right| \forall x \in \mathcal{X} \tag{4.69}$$

$$f_{\text{Dyn}}(x, s_{\text{a}}) = \begin{cases} x \cdot \frac{5}{3.5} & : s_{\text{a}} < s_{\text{amax}}/2 \\ x \cdot \frac{5-1.5\cdot(2\cdot s_{\text{a}}/s_{\text{amax}}-1)^3}{3.5} & : s_{\text{a}} \ge s_{\text{amax}}/2 \end{cases} \tag{4.70}$$

$$f_{fc_{\text{valconf}}}(a, b) = \lambda a. \left(\lambda b. \left(\left\{ \begin{array}{c} f_{\text{Shape1}}(\mathcal{X}, f_{\text{Dyn}}(a, \|\mathcal{X}\|), f_{\text{Dyn}}(b, \|\mathcal{X}\|)) \\ f_{\text{Shape1}}(\mathcal{X}, a, b) \end{array} \right\}\right)\right) \tag{4.71}$$

$$f_{fc_{\text{valcoconf}}}(a, b) = \lambda a. \left(\lambda b. \left(\left\{ \begin{array}{c} f_{\text{Shape2}}(\mathcal{X}, f_{\text{Dyn}}(a, \|\mathcal{X}\|), f_{\text{Dyn}}(b, \|\mathcal{X}\|)) \\ f_{\text{Shape2}}(X, a, b) \end{array} \right\}\right)\right) \tag{4.72}$$

The result of the results of these two functions and the result of the sample confidence functions,

$$\mathcal{C}_{\text{saconf}} := \begin{cases} 1 & : k \geq s_{\text{a}} \\ \frac{k}{s_{\text{a}}} & : 0 \leq k < s_{\text{a}} \end{cases} , \tag{4.73}$$

$$\mathcal{C}_{\text{sacoconf}} := \begin{cases} 0 & : k \geq s_{\text{a}} \\ \frac{s_{\text{a}}-k}{s_{\text{a}}} & : 0 \leq k < s_{\text{a}} \end{cases} , \tag{4.74}$$

can be combined in the following manner:

$$\mathcal{C}_{\text{sconf}} := \left(f_{f_{\mathcal{C}_{\text{valconf}}}}(a, b)\right)(\Delta) \tag{4.75}$$

$$\mathcal{C}_{\text{scoconf}} := \left(f_{f_{\mathcal{C}_{\text{valcoconf}}}}(a, b)\right)(\Delta) \tag{4.76}$$

$$\mathcal{C}_{\text{conf},k,j} = \mathcal{C}_{\text{sconf},k,j} \wedge \mathcal{C}_{\text{saconf},k} \tag{4.77}$$

$$\mathcal{C}_{\text{coconf},k,j} = \mathcal{C}_{\text{scoconf},k,j} \vee \mathcal{C}_{\text{sacoconf},k} \tag{4.78}$$

It should be noted that there are two indices. Two are necessary because each confidence function returns two confidences, one for the question of keeping a state and one for everything else,

$$\text{keep} := \bigvee_{q=1}^{s_{\text{a}}} \left(\mathcal{C}_{\text{conf},q,1} \geq \mathcal{C}_{\text{coconf},q,1}\right), \tag{4.79}$$

$$\text{confidence} := \bigvee_{q=1}^{s_{\text{a}}} \mathcal{C}_{\text{conf},q,2} . \tag{4.80}$$

How and at which point these functions are used is described in the previous subsection. A formal mathematical definition is still missing in this subsection. Let $\mathcal{H}_{\text{P}}$ be the phantom history,

$$\mathcal{H}_{\text{P}} \subset \left\{\mathbb{R}^{\lfloor s_{\text{a}}/10 \rfloor}, \{0, 1\}^{\lfloor s_{\text{a}}/10 \rfloor}\right\} \tag{4.81}$$

and use the indexes j, k, with $k = 1$ being old input values and $k = 2$ being the information if a value is marked. It is used to change the initial history of a created state,

$$l := \lfloor s_{\text{amax}}/10 \rfloor , \tag{4.82}$$

$$q := \text{sum}(\mathcal{H}_{\text{P},1\ldots l,2}) , \tag{4.83}$$

$$\text{initHist}(\mathcal{H}_{\text{P}}) = \begin{cases} v & : q < l/2 \\ \mathcal{H}_{\text{P},1\ldots q,1} & : \text{else} \end{cases} , \tag{4.84}$$

if at least 50% of the values are marked.

Chapter 5

Testing the algorithm

This chapter tests the improved algorithm compared to FCoE from chapter 5, CCAM as defined by [1] and CCAM without the limiting assumptions. The limiting assumptions of CCAM where first discussed in section 3.2 and are the following:

- The noise is small compared to the average of the signal.
- The noise is proportional to the average.
- The signal may not approach or reach 0.

To make it easy to differentiate between the three CCAM algorithms, the following names were chosen:

- The improved algorithm is called myCCAM.
- CCAM as defined by [1] is called CCAM.
- CCAM without the limiting assumptions is called CCAMb.

There are three types of data tested.

1. Synthetic Data
2. Synthetic-Signal mimicking Data
3. Industrial Data

The Synthetic Data exist to show improvements and limitations of myCCAM, while at the same time comparing it to the other algorithms.
The Synthetic-Signal-mimicking Data are signals created by using statistical data of an industrial signal from multiple machines. The signals were created to test drift error detections. The statistical parameters of the real data analysed were mean, sigma,

kurtosis and skew. These parameters were extracted from the industrial data using a moving window approach. To better match the industry data further outlier were introduced, which scale the noise by 4 with a predefined chance. Sections of semi constant parameters were found in the industrial data and these parameters, forced to constants, were used to create multiple data sets. One of these data sets was used in this thesis and out of the 9000 generated signals in this data set 3 were chosen, which have different levels of difficulty. Going from easy to unreasonably hard for the algorithm in its current form.
The Industrial data is, as the name suggests, data from an industrial setting. This data was filtered and downsampled prior to receiving it. The downsample is somewhere in the range of 1 every 1 000 to 100 000. It is done to conserve storage space. This downsample might obscure true information about the signal, which can not be reconstructed. As this is industrial data units could be provided for the signal, but it is intentionally not done to hide the origin of the data.

5.1 Synthetic Data

As already mentioned this section is about testing myCCAM on computer generated data. myCCAM does not get usable initial parameters in order to check whether the runtime parameter adaptation can take the role of predefined parameters. This is a deliberate disadvantage compared to all other algorithms. For CCAMb parameters were chosen defined by statistical parameters of the signal, or determined experimentally and then iteratively optimized for these signals. For CCAM good fitting parameters were experimentally determined. The problem with CCAM is that for some signals the parameters would need to change over time in order to fit the signal. For FCoE parameters were determined experimentally and the main focus is on solving the problem with a threshold value. The chosen history length for all CCAM algorithms is 100.

5.1.1 Simple Step

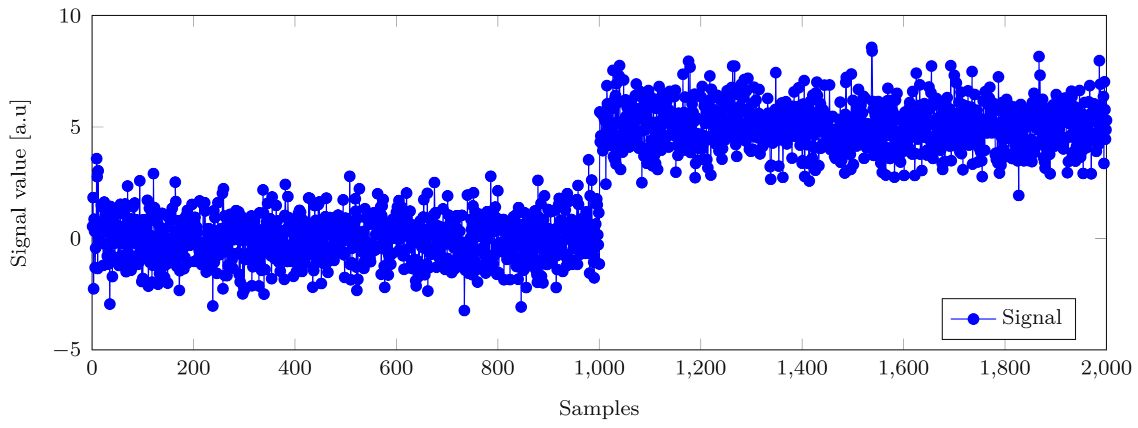


Fig. 5.1: The signal the algorithms are tested on in this subsection.

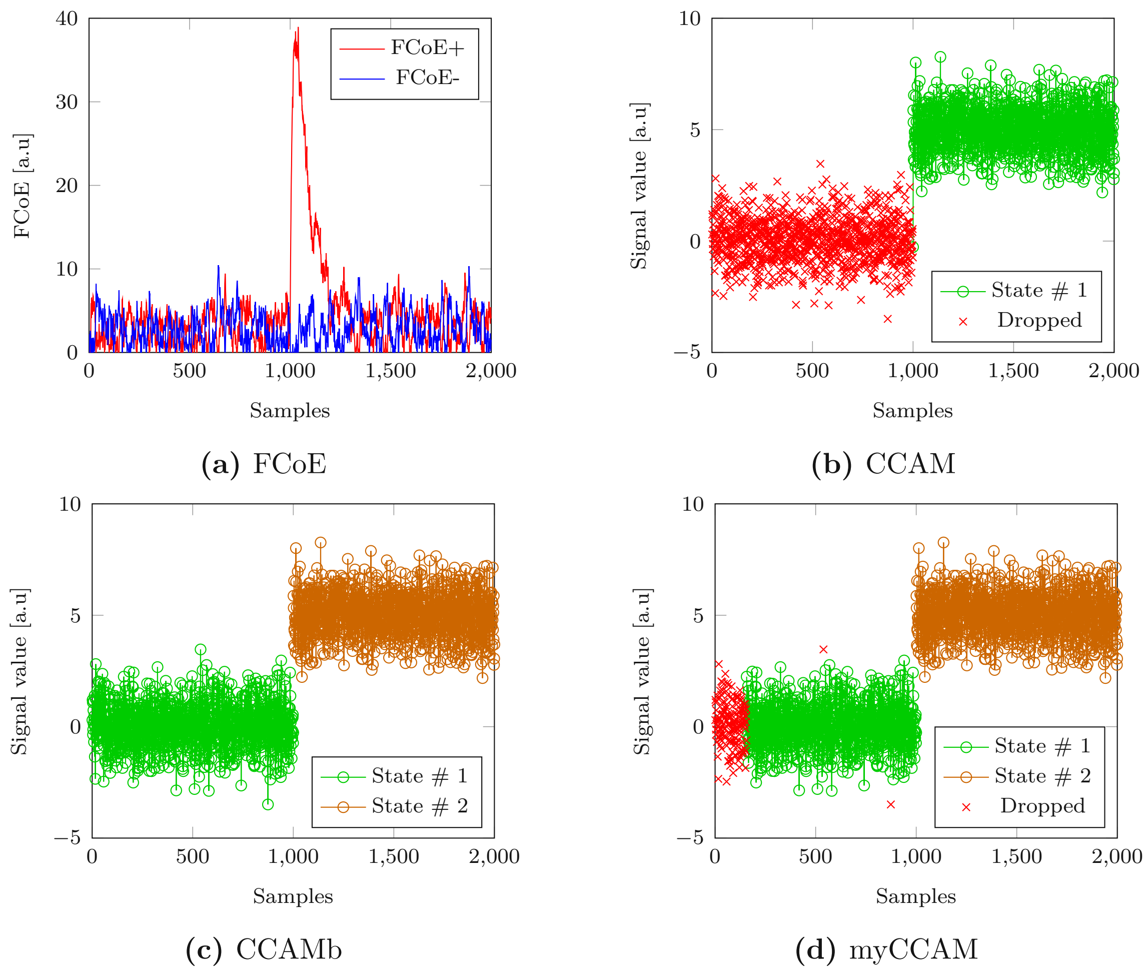


Fig. 5.2: The results of the algorithms.

Figure 5.1 shows the signal for this test. It is a very simple signal and easy to detect. Figure 5.2 displays the response of the algorithms. FCoE could solve it without a problem as expected. CCAM has a problem at around 0 as expected. It is working against its assumptions in that area. Once the signal has jumped it can find and hold a state. CCAMb can find and hold both states without a problem. myCCAM needs some time to find fitting parameters and has 2 errors. The errors are due to the parameters not being quite big enough as described in section 4.5.

5.1.2 Strong drift with negative Step

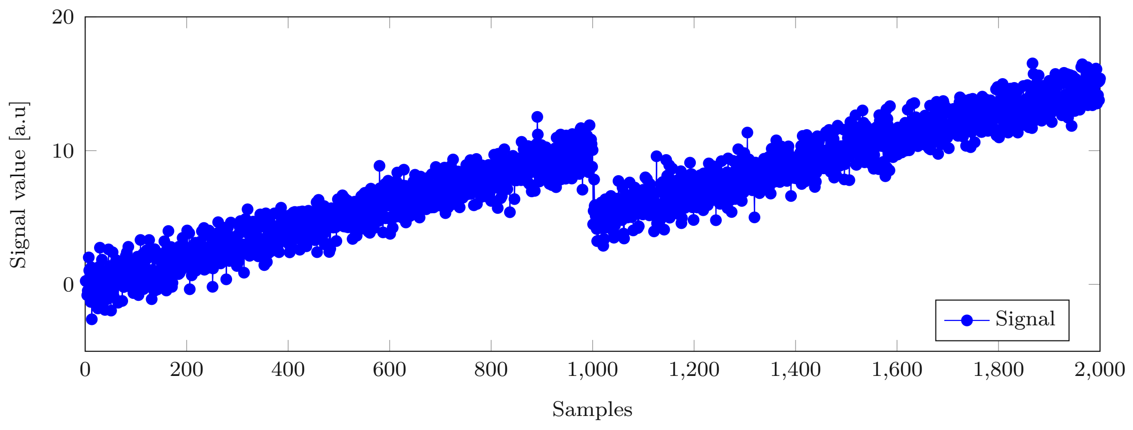
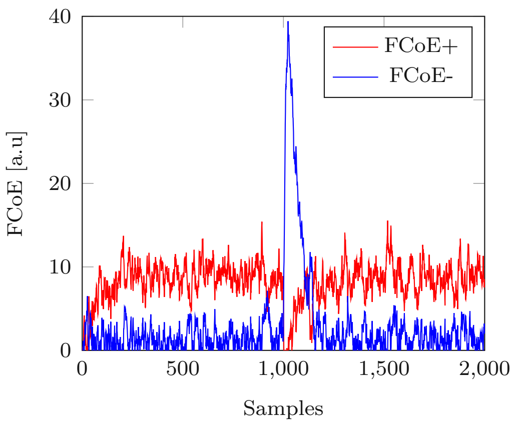
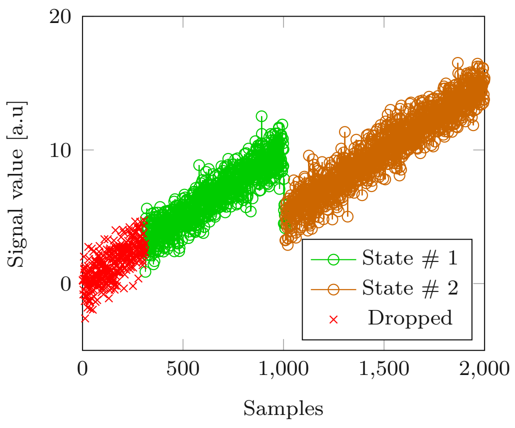


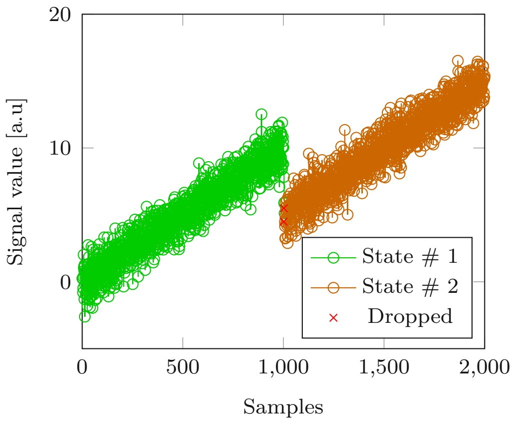
Fig. 5.3: The Signal the algorithms are tested on in this subsection.



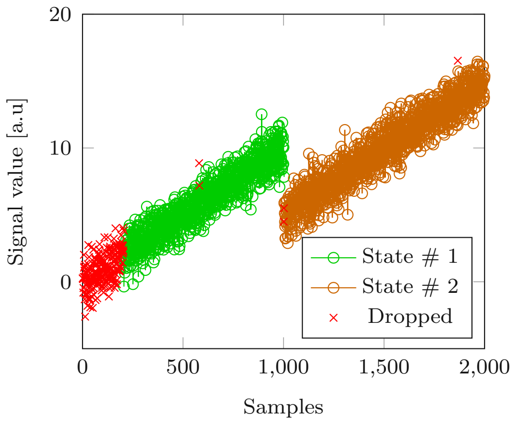
(a) FCoE



(b) CCAM



(c) CCAMb



(d) myCCAM

Fig. 5.4: The results of the algorithms.

Figure 5.3 shows the signal for this test. It is a a bit harder than the last subsection, but still quite easy. Figure 5.4 displays the response of the algorithms. FCoE could again easily solve the problem. CCAM again has a problem at around 0 as expected. Once it reaches a certain value it can solve this problem. CCAMb can find and hold both states without a problem. myCCAM needs some time to find fitting parameters and has 3 outliers. All CCAM algorithms had a bit of a struggle finding the precise position of the step. This is due to the signal by chance creating a non-instantaneous jump. This is visible in Figure 5.5.

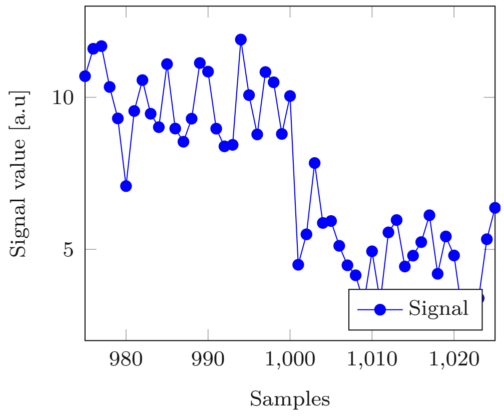


Fig. 5.5: Close-up of the jump of Figure 5.3.

5.1.3 Strong drift and Sine with negative Step

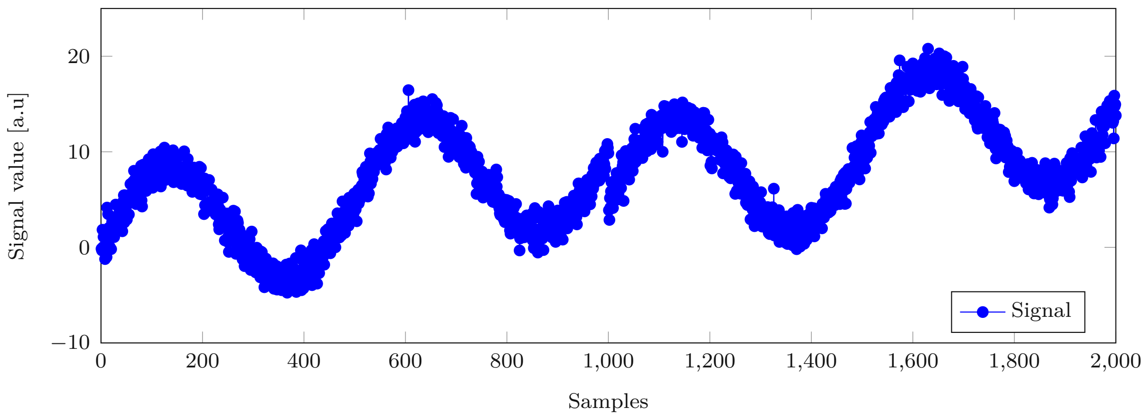
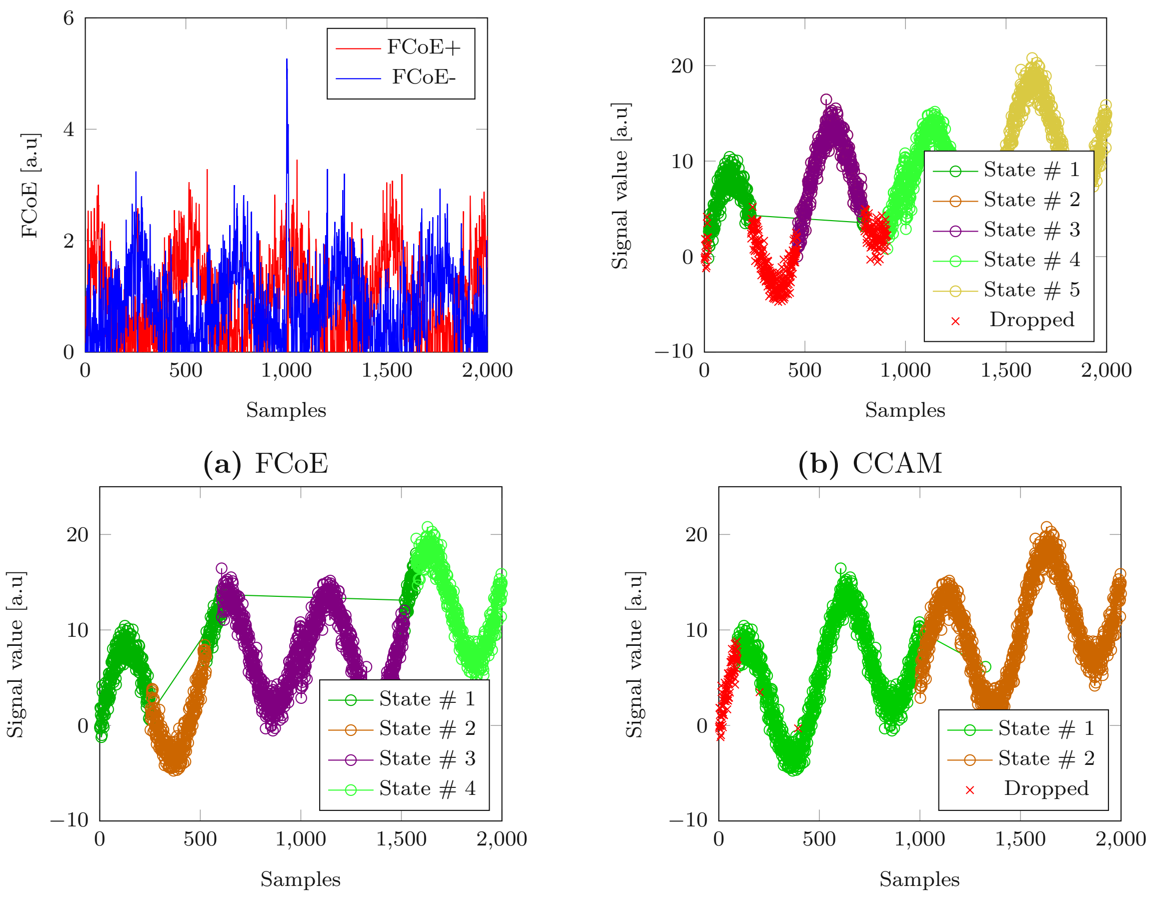


Fig. 5.6: The signal the algorithms are tested on in this subsection.



(a) FCoE (b) CCAM (c) CCAMb (d) myCCAM

Fig. 5.7: The results of the algorithms.

Figure 5.6 shows the signal for this test. It is quite hard to successfully segment the signal. FCoE can barely solve the problem, but it could easily be interpreted as an outlier. CCAM can not solve the problem and regularly enters a zone where it can not hold a state at all. CCAMb has the problem that it can either find too many or only one state. The state change is not detectable because it simply is to close to the old data. myCCAM can solve this problem, but has two outliers and one incorrect state assumption which was simply found by chance. This state change represents one of the problems that still remain inside myCCAM. This state is only held for one value and therefore should not be counted as a true state change. It should be noted that in signals of this type myCCAM has a chance to re-enter the first state due to an outlier. This is one of the problems already discussed, and is due to the fact that all old state have equal opportunities regardless of proximity, only dependent on the confidence.

5.1.4 Strong drift and fast Sine with negative Step

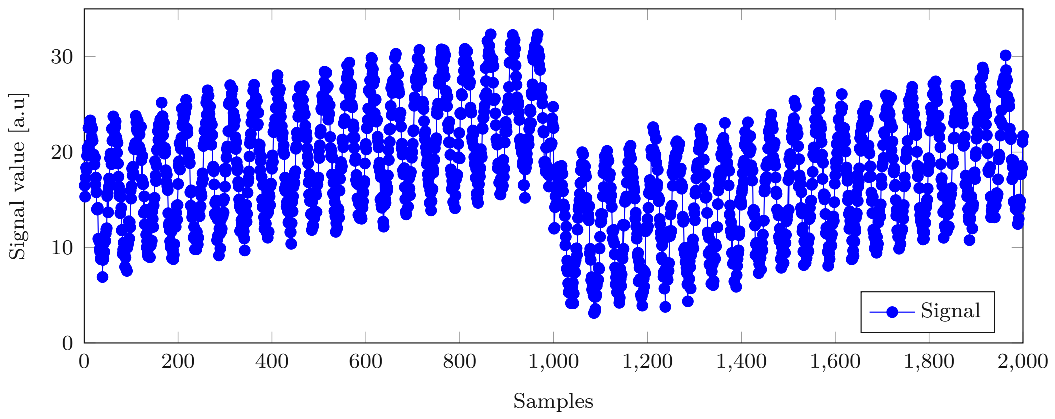


Fig. 5.8: The signal the algorithms are tested on in this subsection.

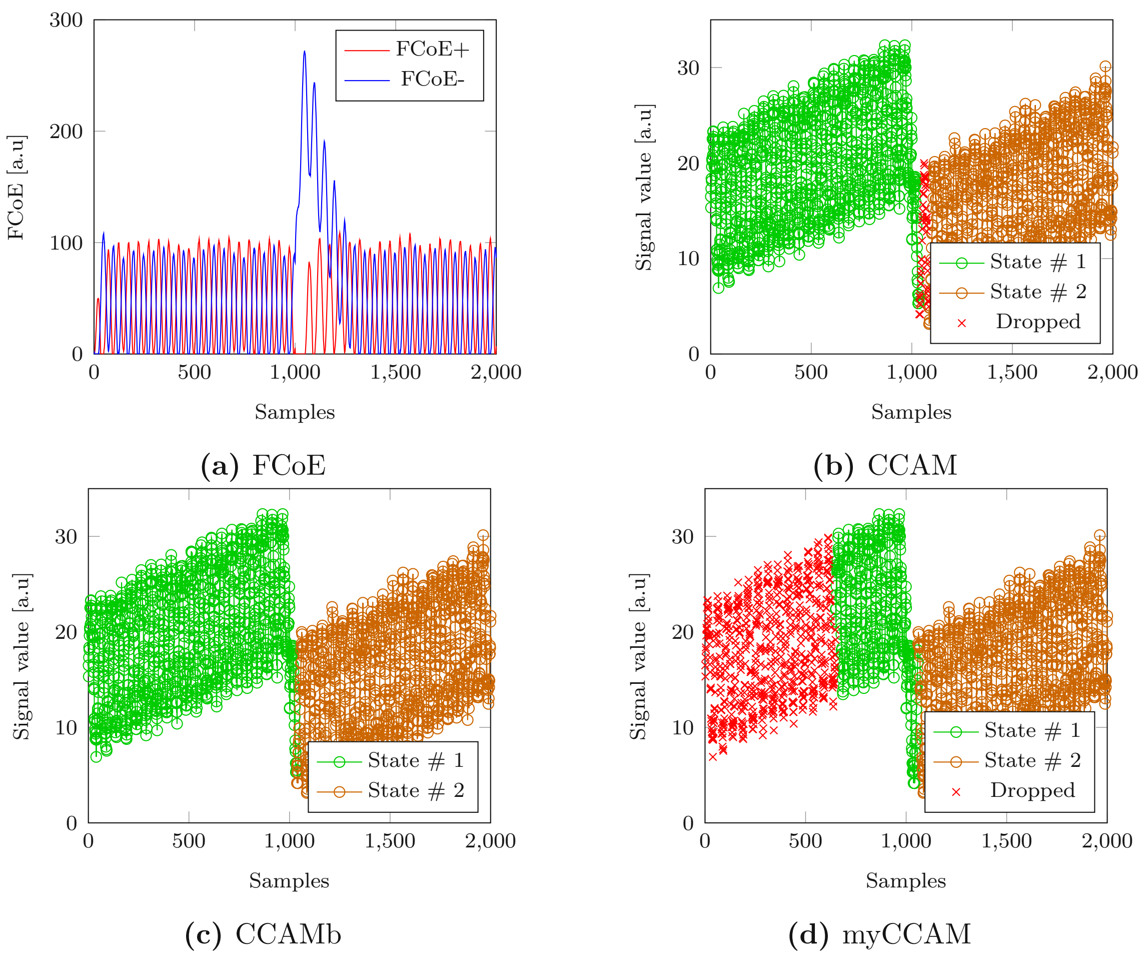


Fig. 5.9: The results of the algorithms.

Figure 5.8 shows the signal for this test. It was meant to display the problem of entering the resonant behaviour, by being at the peak of it, from section 4.3 and for myCCAM to fail because of it. However, it is clearly visible in Figure 5.9 that myCCAM can solve the problem. Every other algorithm can also solve it. myCCAM needed about 30 values more than CCAMb to detect the jump. CCAM needs an offset to be added so that it could be compared. FCoE could detect it without a problem. This example is the first problem with a non Gaussian noise distribution. The noise distribution is displayed in Figure 5.10 . As mentioned myCCAM was not intended to be able to solve it easily, but it still managed. The likely explanation for that is that the Runtime Adaptation (section 4.5) is able to counteract the negative effect. The negative effect is also likely the reason for the delayed detection. This test was also re-tested with 10 times the data length to make sure that it was not by coincidence or that the Runtime Adaptation just did not have time to change the parameters strongly enough to ignore the jump. The same behaviour was found for myCCAM and again it was about 30 samples slower than CCAMb. It is also no coincidence that myCCAM needs longer to calculate the parameters, this is also a reoccurring behaviour.

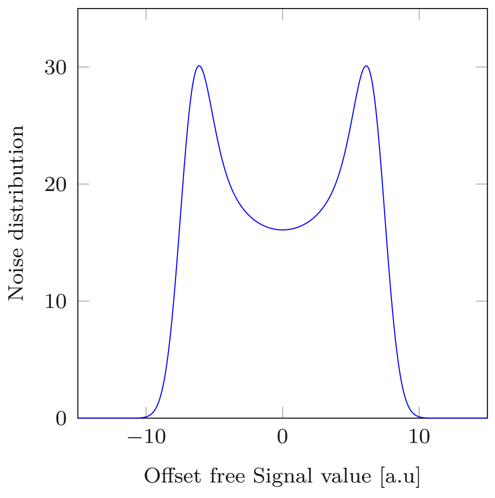


Fig. 5.10: The noise distribution for this example.

5.1.5 Steps with overshoot

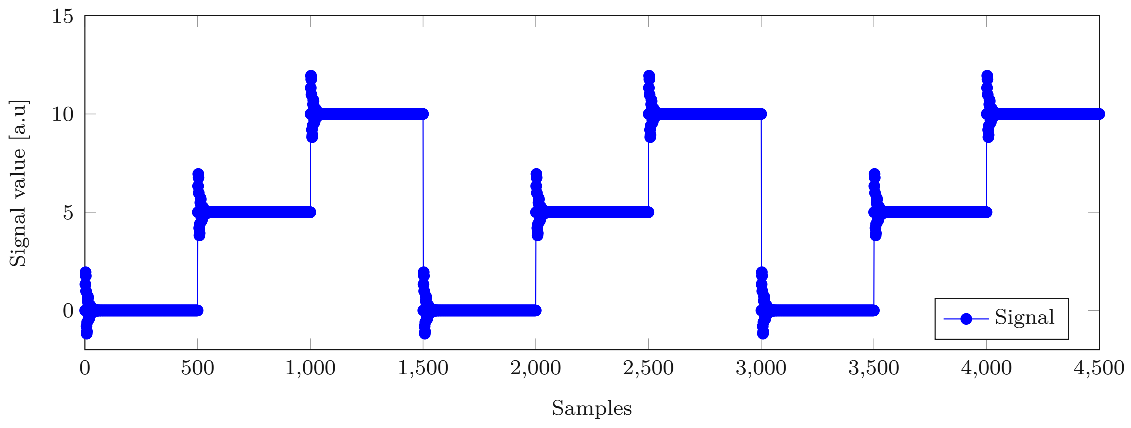
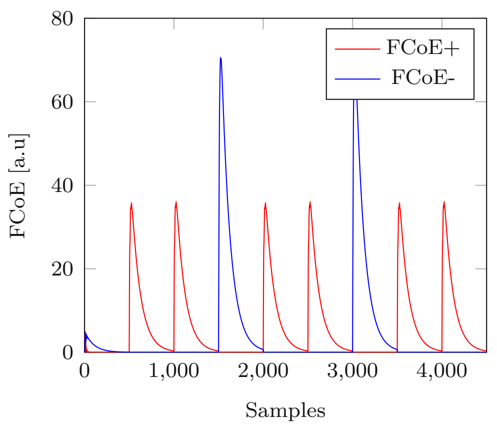
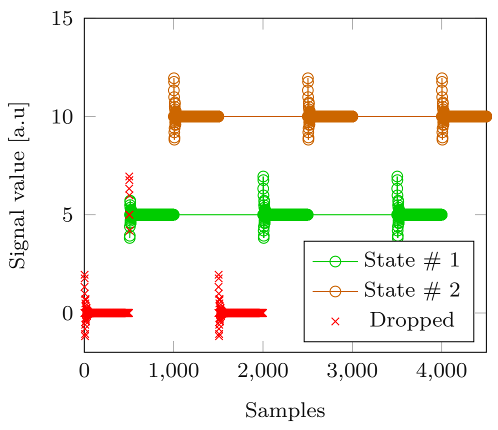


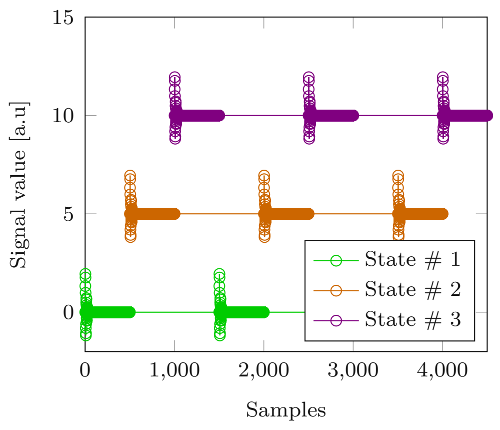
Fig. 5.11: The signal the algorithms are tested on in this subsection.



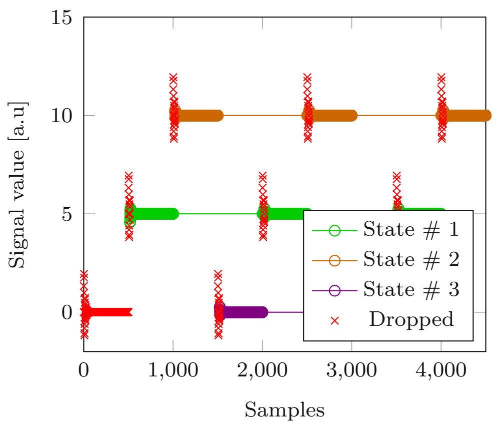
(a) FCoE



(b) CCAM



(c) CCAMb

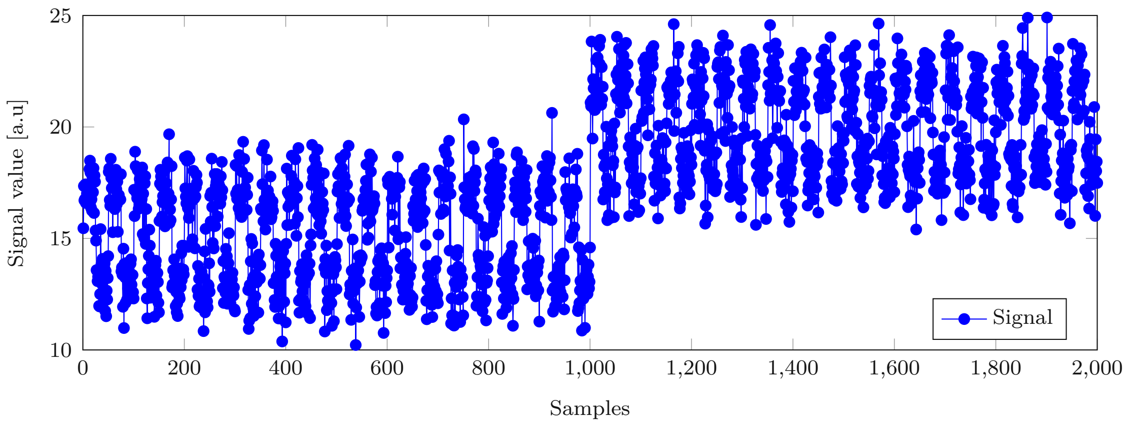


(d) myCCAM

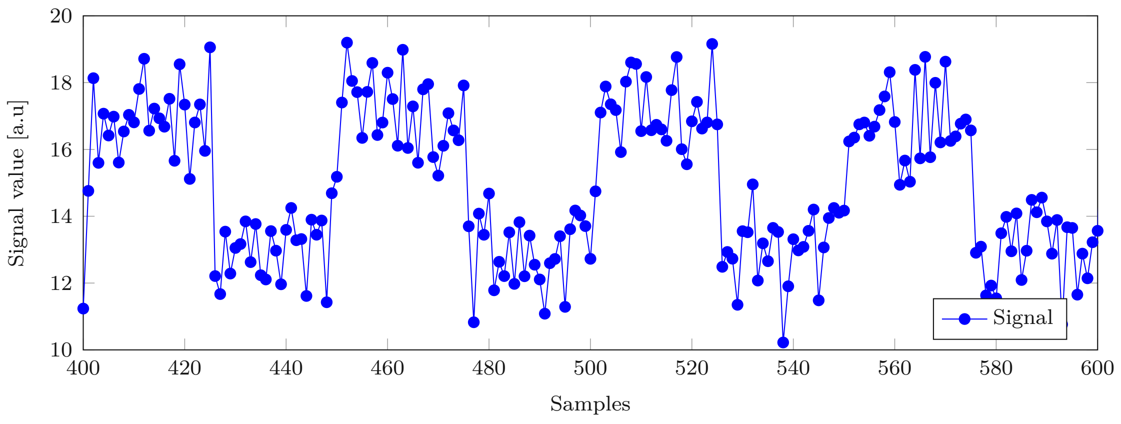
Fig. 5.12: The results of the algorithms.

Figure 5.11 shows the signal for this test. It tries to mimic parts of Current Amplitude form Figure 6 from [1]. It has an overshoot and quickly stabilizes before it jumps again. Figure 5.12 displays the results and FCoE can solve the problem easily. CCAM as expected has a problem with the state at zero. CCAMb can solve this without ever dropping a state. myCCAM has a problem at the start as the extracted parameters are zero once the signal is stable. In other words it tries to approach zero from the negative side and the parameters have to be positive or do not make any sense. It can not hold any state because of that. Once the first jump happens the parameters are increased strongly enough for a state to be held. The parameters extracted locally can not prepare myCCAM for the post jump distribution. Therefore there are always dropped states at the start. This creates a delay of about 20 samples before the correct state is found and used. In addition, once the parameters are increased above zero it can find the state at zero. It should be noted that if the initial parameters would have been set to 0 it could have found the very first state.

5.1.6 Miniature Steps as part of the Noise

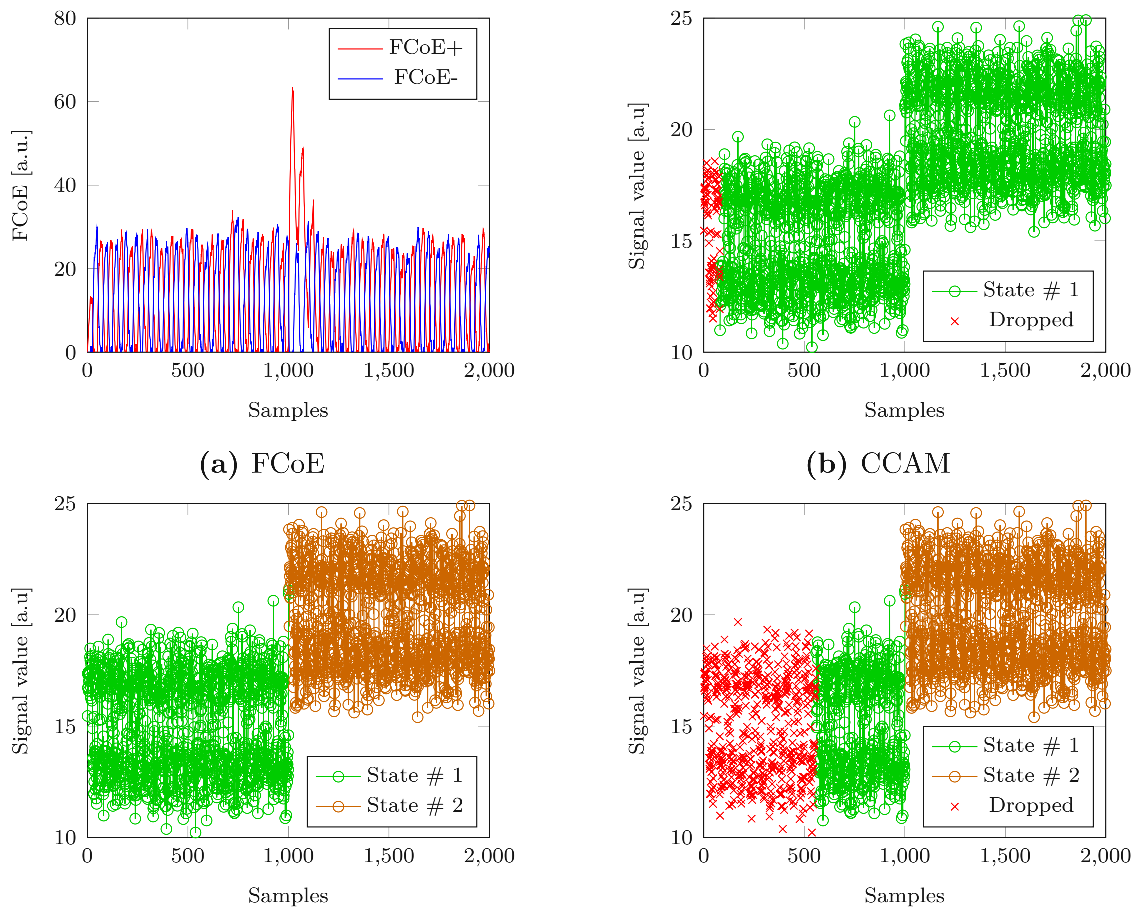


(a) The full signal



(b) Close-up of the signal

Fig. 5.13: The signal the algorithms are tested on in this subsection.



(a) FCoE (b) CCAM

(c) CCAMb (d) myCCAM

Fig. 5.14: The results of the algorithms.

Figure 5.13 shows the signal for this test. It has steps as part of the noise. Figure 5.14 shows the results. FCoE can solve the problem. CCAM can not solve the problem. A lower threshold would make it impossible to find the lower state. CCAMb can solve the problem. myCCAM can also solve the problem, but needs more time to find the parameters. The distribution in this case would be two separate white Gaussian noise distributions with 4σ between the peaks.

5.2 Synthetic-Signal-mimicking Data

This section is focusing on data which was generated from statistical and internal system information.

5.2.1 First Signal

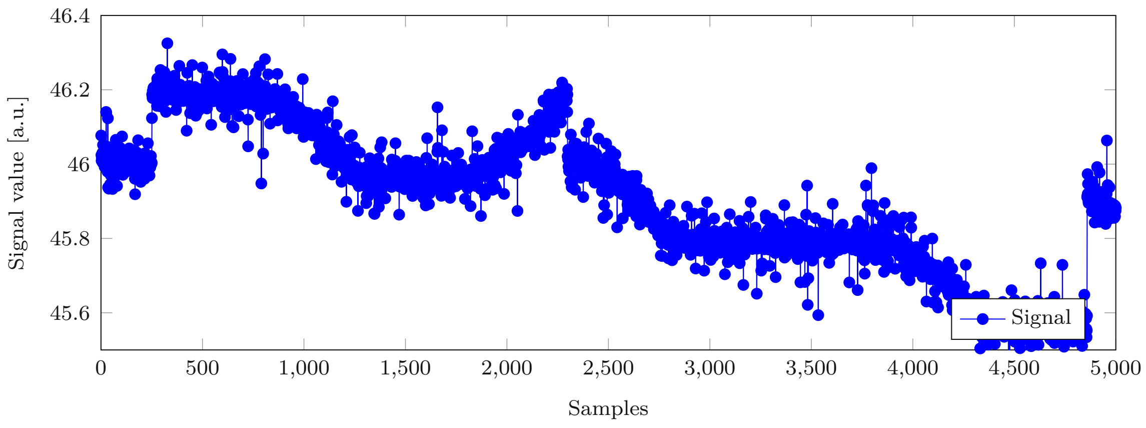


Fig. 5.15: The signal the algorithms are tested on in this subsection.

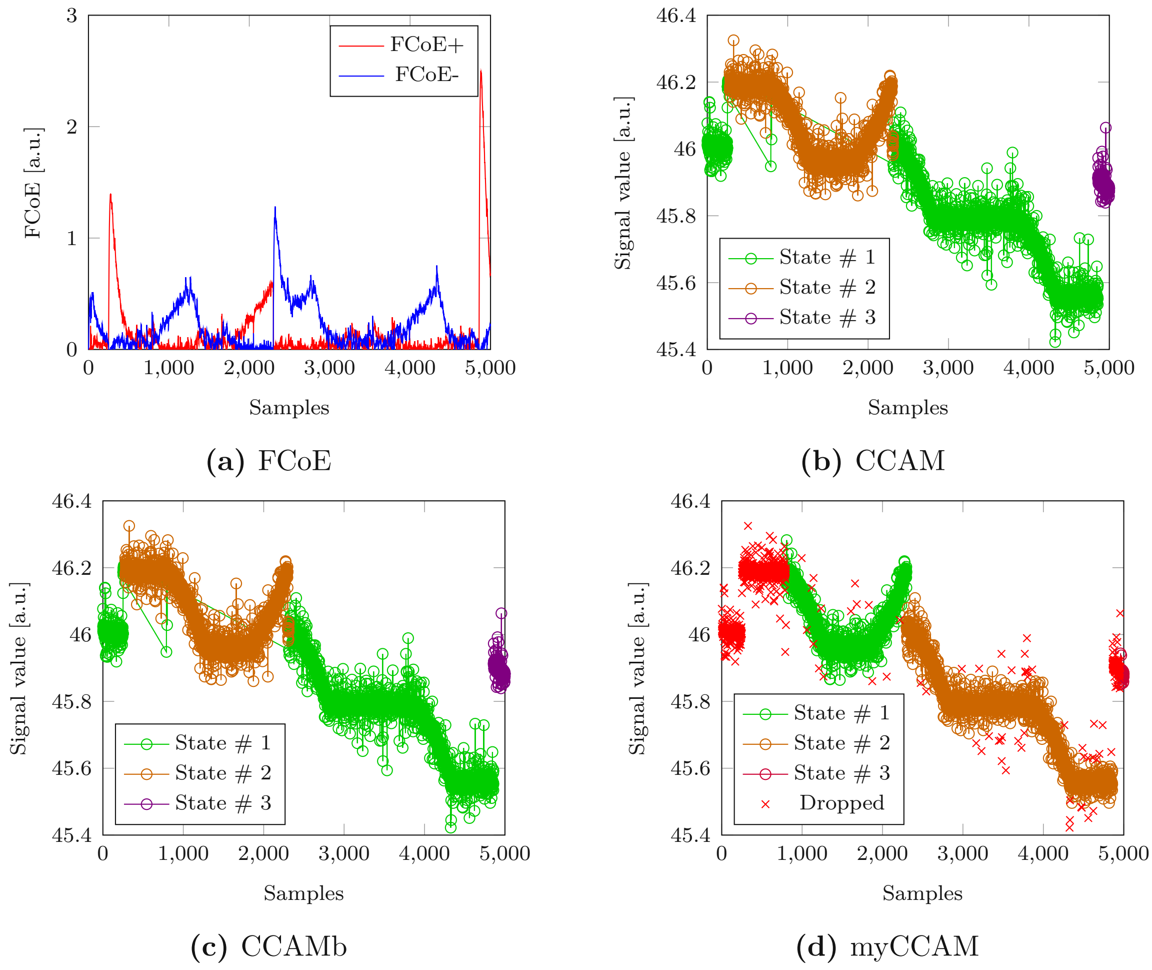


Fig. 5.16: The results of the algorithms.

Figure 5.15 shows the signal for this test. Figure 5.16 displays the response of the algorithms. FCoE can solve it. CCAM and CCAMb can solve it. For CCAM the assumptions do not hinder it. myCCAM has outliers. This is due to the runtime parameter adaptation not being able to handle the distribution. In this case it managed to segment the signal nearly correctly. It can not segment it while it still has incorrect parameters and at the end it can not find the state correctly as there is an outlier while building it. These outliers are a big problem for the algorithm to successfully build a state.

5.2.2 Second Signal

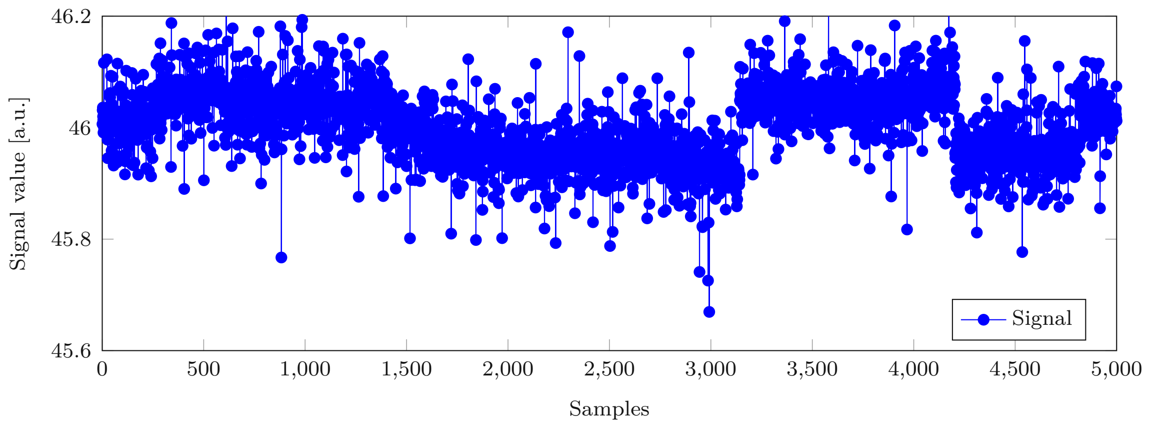
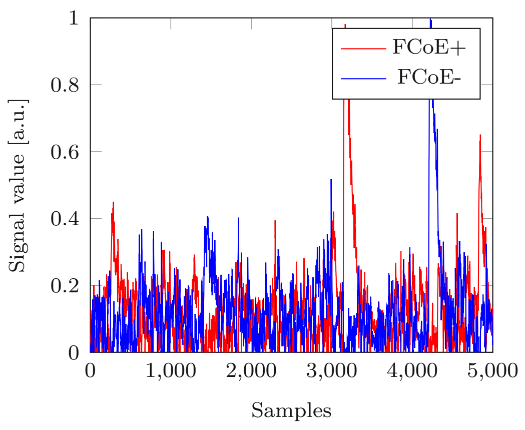
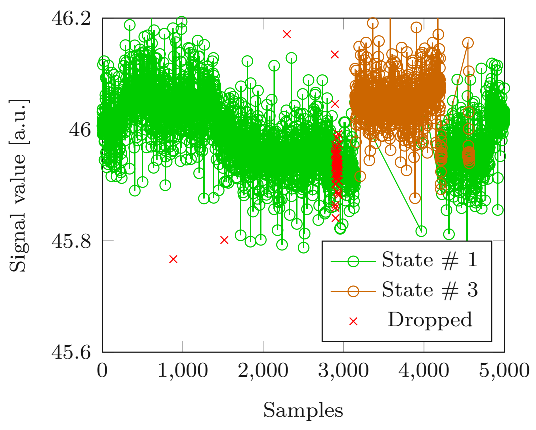


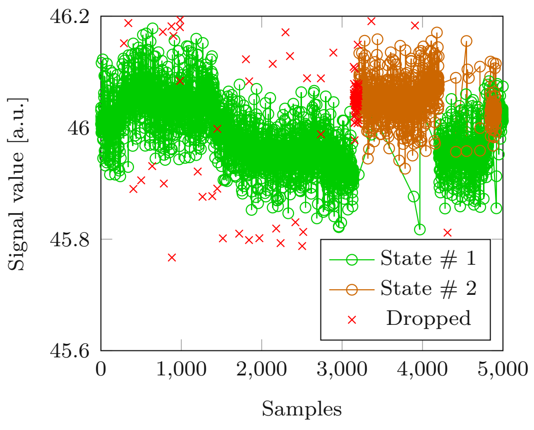
Fig. 5.17: The Signal the algorithms are tested on in this subsection.



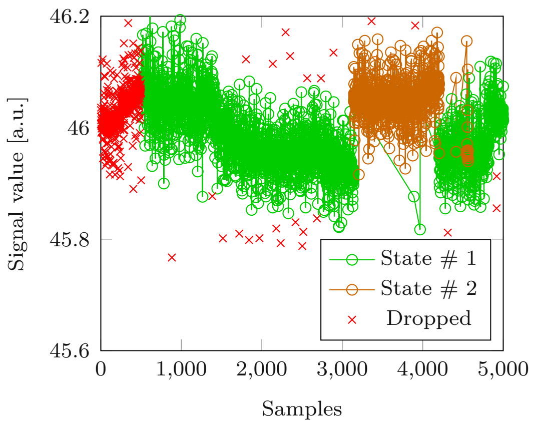
(a) FCoE



(b) CCAM



(c) CCAMb



(d) myCCAM

Fig. 5.18: The results of the algorithms.

Figure 5.17 shows the signal for this test. Figure 5.18 displays the response of the algorithms. FCoE can solve it barely. All CCAM based algorithms can not detect the last step. CCAM finds the remaining two jumps, but has a large dropped area in the middle. CCAMb can also solve the two remaining jumps and somehow finds the third jump, but the finding aspect might just be re-entering the second state due to an outlier, which appears likely on closer inspection. In order to find the jump reliably, the threshold value would have to be smaller, but then it could no longer find a state successfully. myCCAM has a similar problem with extracting a to big threshold. A second problem with this is that it switches state due to an outlier and keeps the incorrect state too long. This would be an incorrect state change.

5.2.3 Third Signal

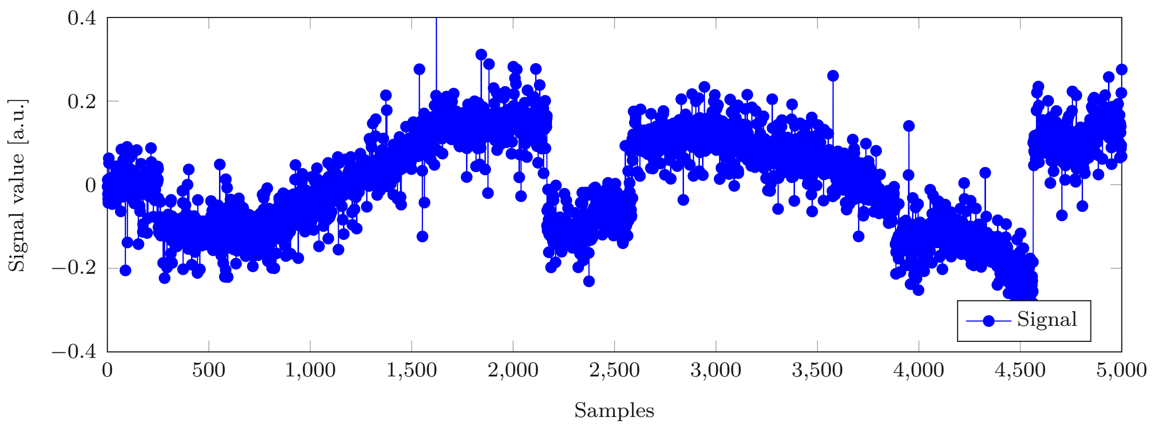
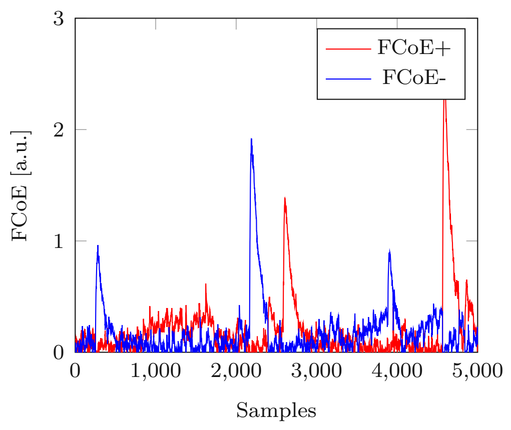
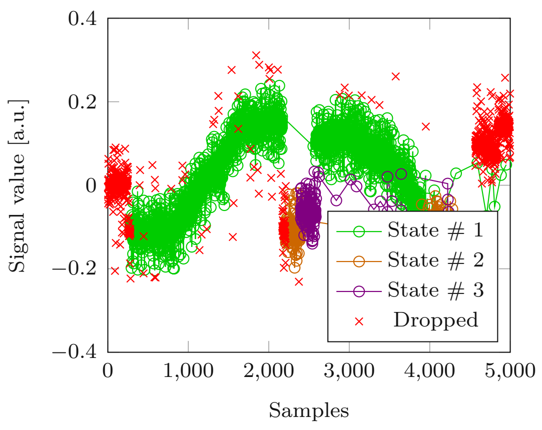


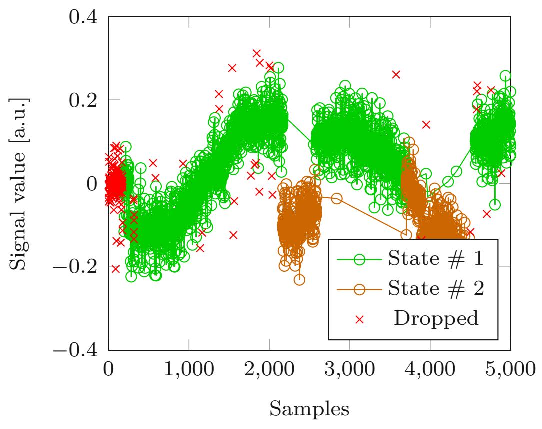
Fig. 5.19: The signal the algorithms are tested on in this subsection.



(a) FCoE



(b) CCAMb



(c) myCCAM

Fig. 5.20: The results of the algorithms.

Figure 5.19 shows the signal for this test. Figure 5.20 displays the response of the algorithms. CCAM is missing due to it fighting against its assumptions and there is no point in displaying it. FCoE only can find 6 out of the 9 jumps. It should be noted that the last "found" jump has a value difference of 0.005 compared to an incorrect peak. CCAMb can find 5 out of the 9 jumps. myCCAM does even worse and only finds 3 jumps and one incorrect one. This again displays the problems of the algorithm with this kind of data. That there are actually 9 jumps can easily be seen in Figure 5.21 which is the signal after it has passed through a median filter of the fifth order, so a delay of 2 samples. Afterwards it has been segmented by myCCAM with suitable initial parameters. It should be noted that due to remaining outliers two states could not be build. Figure 5.21 also shows that if preprocessing is possible it should be used, as it makes the signals easier to interpret.

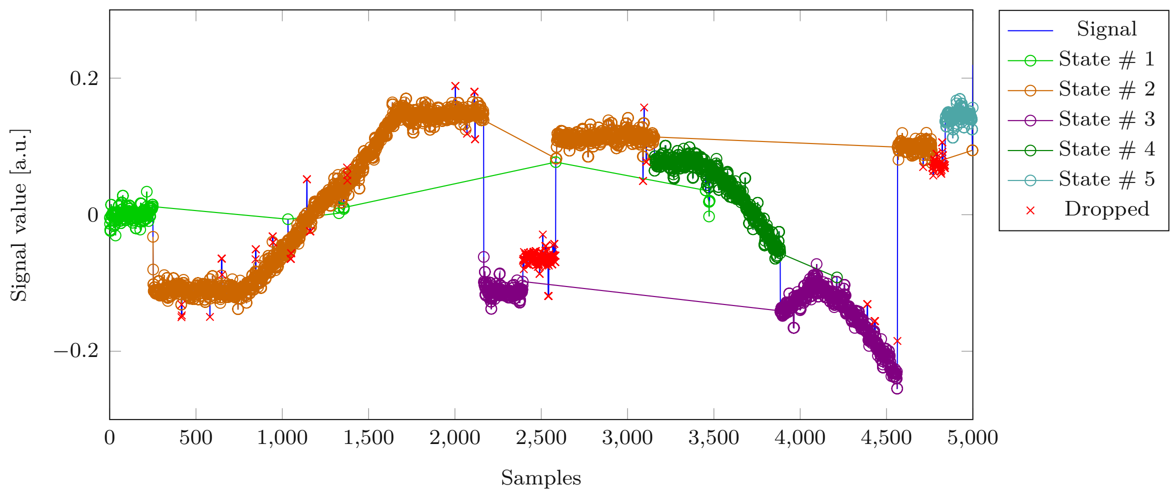


Fig. 5.21: The signal from Figure 5.19 filtered with a median filter and then segmented by myCCAM.

5.3 Industrial Data

Industrial data is a bit more problematic than generated data. There is, so to speak, no known truth. Interpreting it is the only option to define success. In some cases interpretation is easy, other times multiple interpretations are possible. In other words the results will be subjective. The main focus is on interpreting the data as a human would.

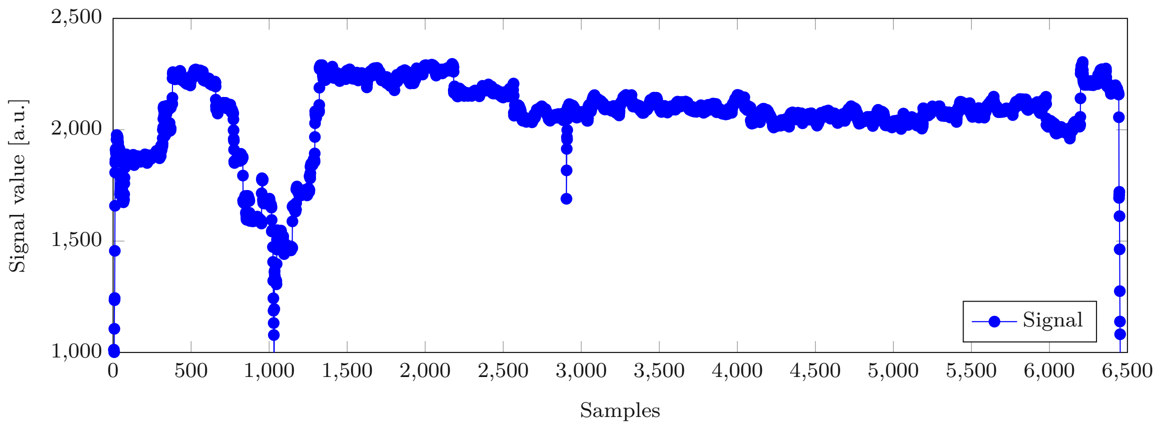


Fig. 5.22: The signal the algorithms are tested on in this subsection.

Figure 5.22 displays the test signal. Prior to sample number 1300 short states seem to exist, they seem to have a roughly stable value. This is the first inspected range and it is visible in Figure 5.23. One of the smaller states that wants to be found has a length of about 20 samples, this forces the algorithm to have a history length of less than 20 samples. A length of 15 was chosen. Due to this short length some adaptations are per default disabled. The first one is entering a temporary phase where other states are actively checked. When a new state is created, it by definition enters a temporary phase which has the length of one tenth the length of the history. Because one tenth of 15 is one it skips this phase. Integrated preprocessing gets deactivated, because it is known that the behaviour is poor with short history lengths. This technically decreases sensitivity, which in this case does not matter. The runtime parameter adaptation reacts a lot quicker and therefore is a lot more sensitive to the values.

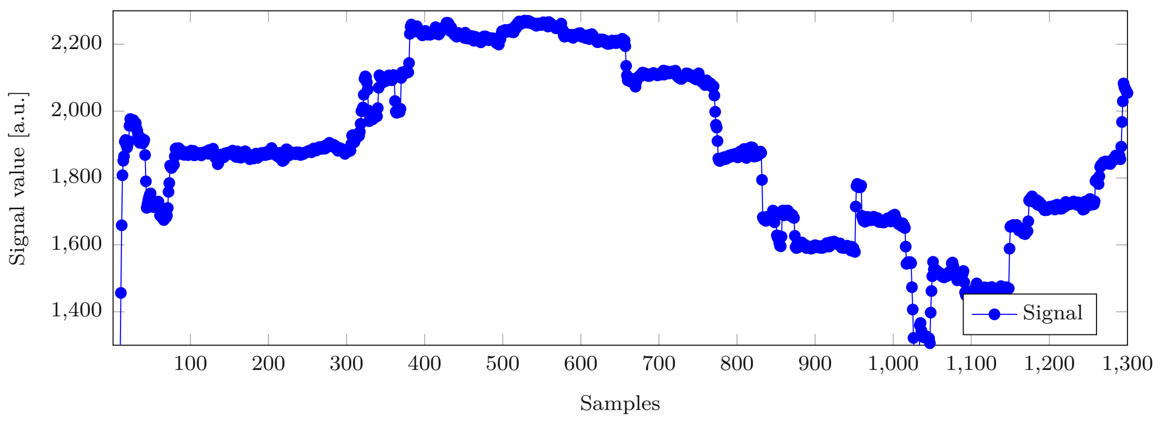
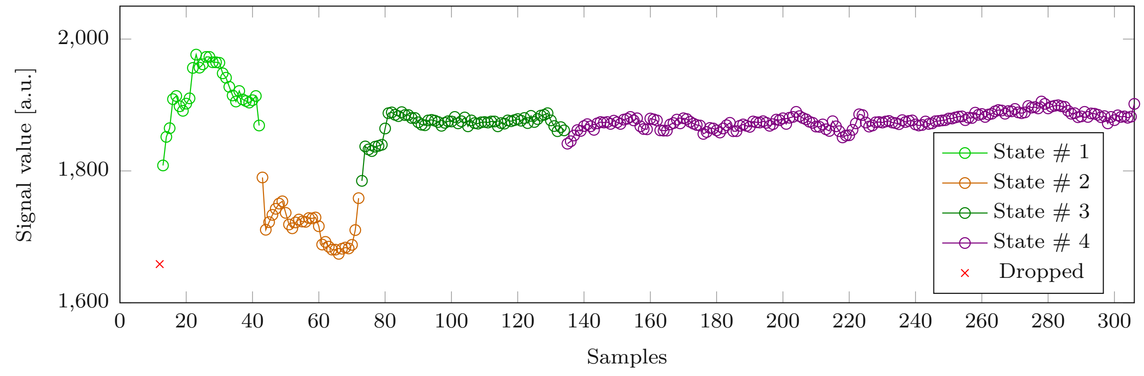
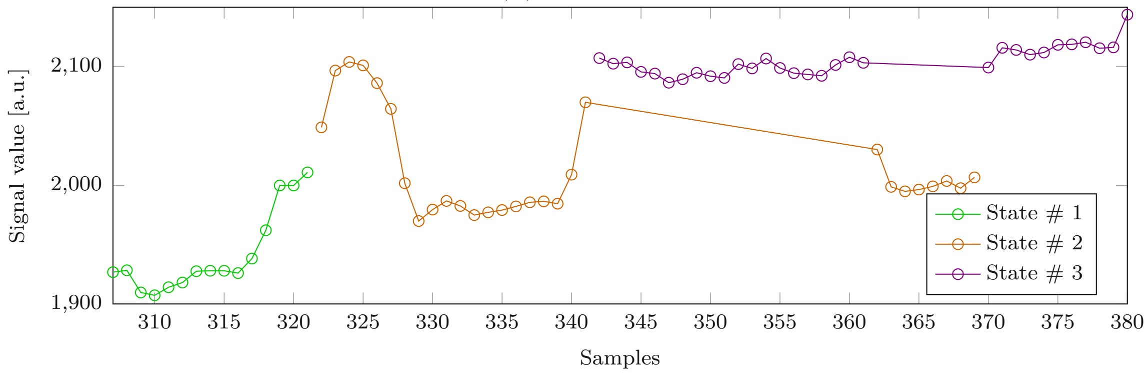


Fig. 5.23: Sub-range 0-1300 of Figure 5.22.

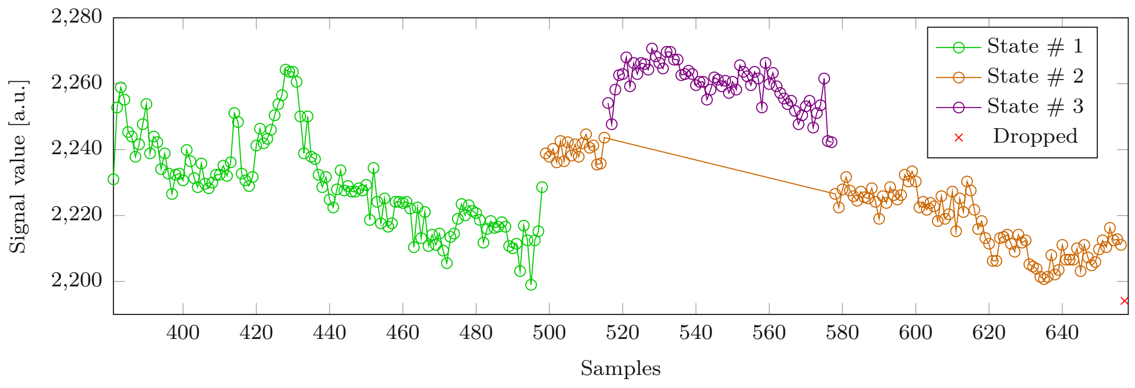
myCCAM created more than 15 states, which makes the graphs almost illegible, as either colours have to be reused or they have to be too similar to be easily differentiated. So instead selected passages are shown in Figure 5.24. The rest of the signal was easily segmented and does not give additional information. Sub-figure (a) and (d) display how it would have been use-full if the state could have been re-entered. Sub-figure (b) displays the small state that was desired to be found, it has the name State # 3 in the Sub-figure. It also displays the dynamic confidence function in full action, in combination with runtime parameter adaptation at around 325. Sub-figure (c) displays an interesting behaviour. It displays 3 different states, in truth there might be anything from 4 to only 1 state. Looking at the signal on that scale makes the segmentation plausible.



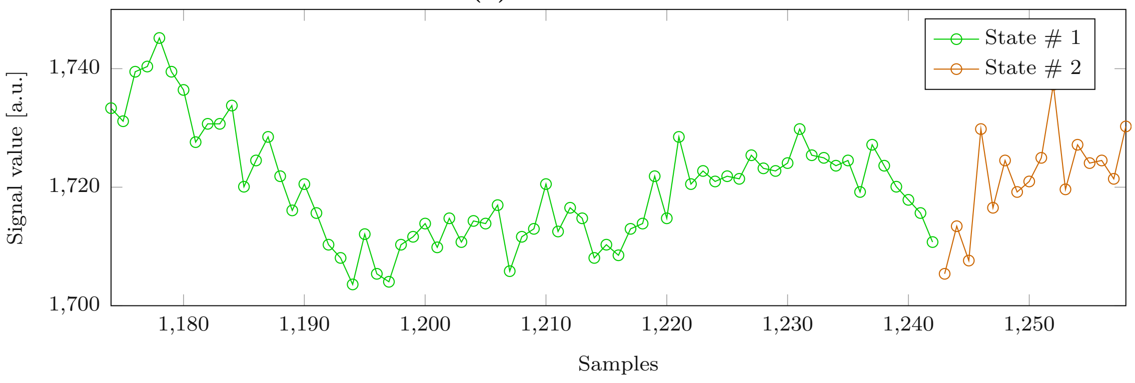
(a) 0 to 306



(b) 307 to 380



(c) 381 to 658



(d) 1174 to 1258

Fig. 5.24: The results of myCCAM split into parts to make them easier to display.

On the other hand, Figure 5.24 (c) might only be a noise pattern inherent to the signal. This idea is supported by the test-signal range 2550 to 4100, displayed in Figure 5.25. These patterns might be state changes with strong negative drifts or a sawtooth noise. If they should be part of the noise a history length of at least 250-300 is required due to the automatic parameter deduction. Due to the shape of that noise the internal preprocessing will make problems as there is a dominant drift in the "noise". This would force a further increase in the history length to about 600-800. This in turn makes it impossible to find results from 0 to about 1300. Figure 5.26 displays what was found with a history length of 600. It is quite noticeable that there are many dropped values, this is due to the amount of data available at those positions. By removing the integrated preprocessing the history length can be set to 250, therefore being able to build the desired states as well. This is displayed in Figure 5.27. It should be noted that there are 2 state changes which do not make sense. The first one from state 3 to 2 at around 3100 might actually be a true state change, but it quickly switches back into the old state. The switching back or the initial switch would be incorrect. The second switch is right before another seemingly true state change. This makes sense from the point of view of the algorithm, but not from a global view. By extracting the threshold information of the runtime parameter adaptation and using the maximum of of the desired area as the threshold for CCAMb a similar result can be achieved with a lower history length. This in turn means that the decision to use only the active states history for the parameter adaptation was bad for this signal. It could use its own history, which could get cleared with a state change, but that would also mean an increase in processing power required. On the other hand granting full state right at a fraction of the history length could also be used to solve this.
To display the change in behaviour of taking the smaller history length of 15 samples Figure 5.28 is displayed. In this figure 24 different states are visible and some share the same colour and markers. It is clear that this does not segment the data correctly. The reason for it is that it figuratively speaking tries to detect jumps while riding the pattern. Also the parameters which are generated at runtime do not incorporate the pattern.

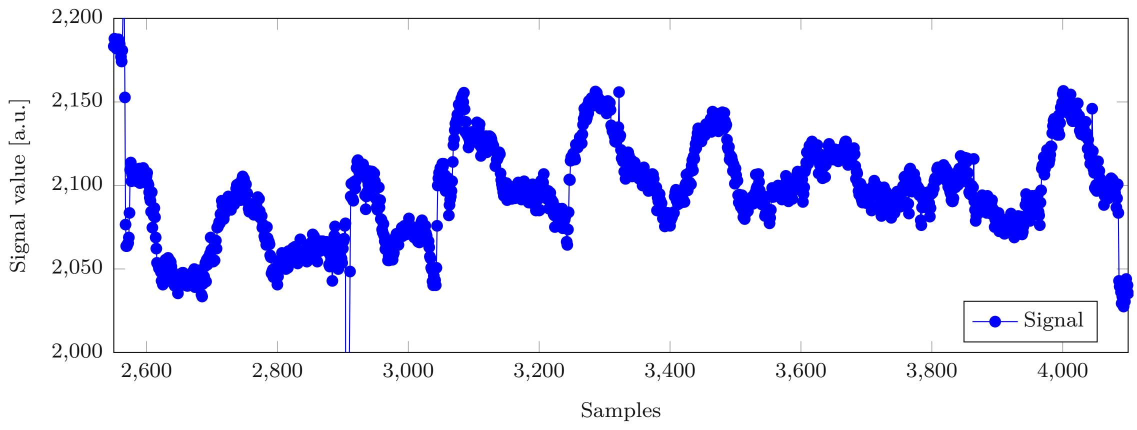


Fig. 5.25: Sub-range 2550-4100 of Figure 5.22.

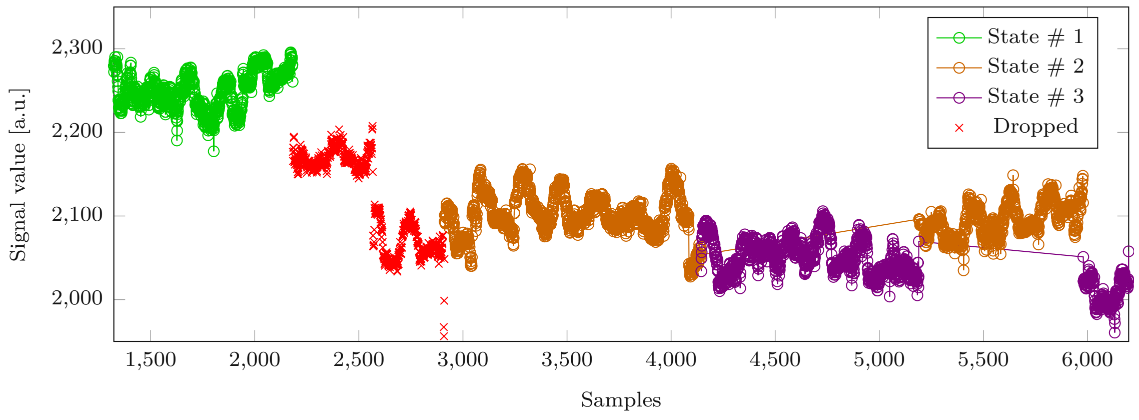


Fig. 5.26: myCCAM result for the sub-range 1323-6198 of Figure 5.22 with a history length of 600.

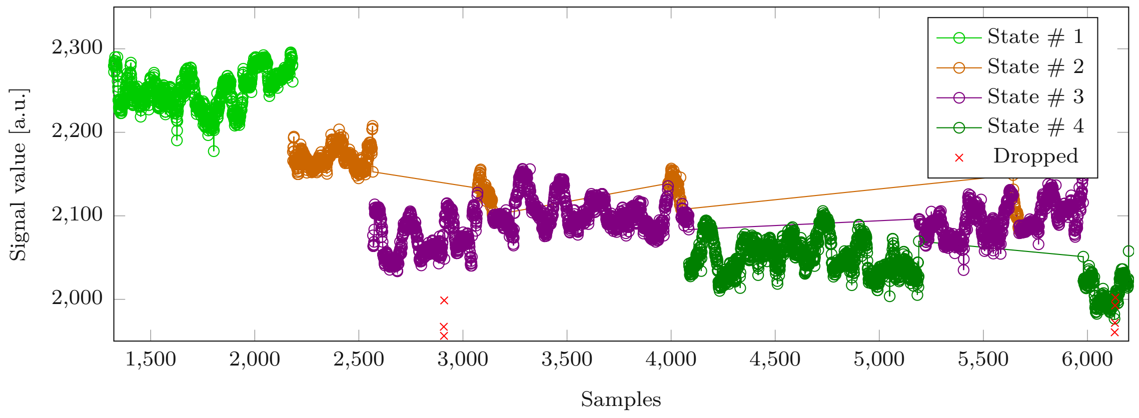


Fig. 5.27: myCCAM without integrated preprocessing result for the sub-range 1323-6198 of Figure 5.22 with a history length of 250.

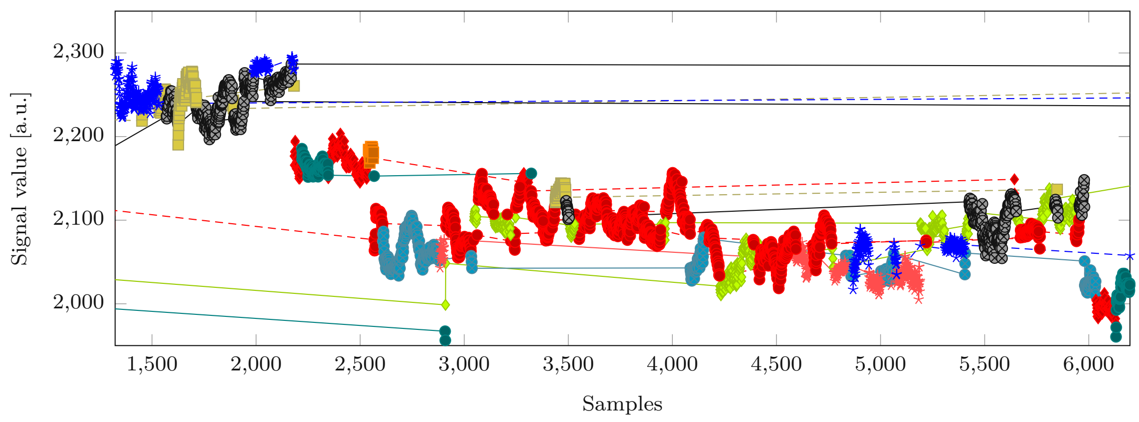


Fig. 5.28: myCCAM without integrated preprocessing result for the sub-range 1323-6198 of Figure 5.22 with a history length of 15. Note multiple states may share colours and markers and no legend is provided as 24 individual states are visible.

Chapter 6

Conclusion and Outlook

6.1 Conclusion

The main parts of this thesis are: the analyses, the extensions and the tests.
The analyses differ in their perspectives and approach. The mathematical analysis, focused on the mechanism within the algorithm, in order to figure out how and why a state is held. The behaviour analysis, analysed the algorithm from the outside, by using a defined distribution, probabilities of keeping/building a state were calculated. Each of the extensions targeted a specific problem, while at the same time trying to minimize computational power.
The tests were performed on different sets of data, to show the behaviour of the algorithms and extensions. The fully customized algorithm was essentially given one parameter while the rest of them were extracted during the runtime. On one of the tests one extension needed to be disabled to detect most of the desired states.

The Mathematical Analysis revealed that there are multiple types of solutions, categorized according to their main contributors.
These solutions are the representation of the confidence and co-confidence functions and describe if a state is kept, depending on the ordered distance of a new input and the states history.
The Single-Point Solution, also called threshold solution, is a simple solution and easy to enforce. It was used for the entire thesis. It is fully describable by two thresholds that can be represented by a single point.
The Multi-Point Solution defines multiple of such points. The thesis has shown that it can be generated. It is not yet known whether the Multi-Point Solution has any advantages over the Single-point Solution.

The Behaviour Analysis was done for the Single-Point Solution and white Gaussian noise. It brought the noise parameters into relationship with the Single-Point Solution parameters. It shows the probability of keeping or loosing a state and focuses on the interconnection of the parameters. The width of the noise distribution influences indirectly the sensitivity. To improve this the extension: Integrated Preprocessing was created. The value threshold of the Single-Point solution has a minimum viable value dependent on the noise width, to successfully build a state. To improve this the extension: Dynamic Value Confidences and Phantom History was created. The noise distribution might not be known a priori or might change at runtime, which would lead to incorrect parameters. To counteract this the extension: Runtime Adaptation was created.

Extension: Integrated Preprocessing targets the history of a state to remove drifts, as drifts would widen the noise distribution. It is able to eliminate drifts, thereby increasing the sensitivity and reducing errors. However, the tests have shown that incorrect behaviour occurs easily. This incorrect behaviour can be suppressed by increasing the history length, which is sometimes undesirable since small desired states might be merged into larger ones or cannot be found. This leads to the conclusion that the defined extension in its current state is insufficient.

Extension: Dynamic Value Confidences and Phantom History aimed at the problem of successfully finding a state. It increased the probability of finding a state immensely, at the cost of being unable of finding small state changes while building a state. The adaptation decouples keeping of a state and the confidence in this state, during and after building it. It made it possible to use very restrictive definitions for keeping a state, which in turn increase sensitivity and at the same time making building the state very likely. It should be noted that increasing sensitivity also increases the error rate.
The Phantom History was originally designed to make jumps detectable when the pre and post jump distributions overlap significantly. This can be achieved, but the sensitivity, which is required to do so, might yield an undesirably high error rate. However, it has a beneficial influence on the Runtime Adaptation, where it significantly improves the speed with which usable parameters are approached.

Extension: Runtime Adaptation aimed to calculate the parameters at run time and it achieved it to a certain extent. It underperformed, which had positive and negative side effects. The positive is that it increases sensitivity, but the downside is that it also

increases the error rate. At the moment CCAM cannot handle errors correctly. Let us assume that two states exist, A and B and their signal value domains are interchangeable. It can happen, due to an outlier that the algorithm exists A. The next input value should belong to A again, but it could enter A or B because they are interchangeable and by chance B is slightly better fitting. This would display an incorrect state change from A to B, with an outlier in between. This behaviour is also possible if the value domain of A and B only overlap, but are not interchangeable. In this case B would morph into As value domain. Again an incorrect behaviour. This behaviour was visible in the tests. How a state is re-entered and then kept is the problem.
The test have shown that the extensions works great for the distribution it was designed for. It also worked for other distributions, but depending on the distribution it took longer to find fitting parameters or extracted a to big value threshold. None the less it was still a good approximation for the desired values.

Weather it is learning or not is another thing that needs to be addressed with the Runtime Adaptation. This topic arises as neural networks are currently a big topic. The answer to this question is a bit more problematic. A possible close comparison for the whole algorithm would be a non-linear adaptive filter. In this comparison, the current states history and the new input value would be the input signals, the two outputs would be boolean value of keeping/dropping a state and the confidence value, and the parameters would be functions. In general for adaptive filters the adaptation is an optimisation problem. In other words how do the parameters need to be changed to create the desired output. The next aspect is that neural networks can be considered as specific non-linear adaptive filters and what they describe as learning are optimisation algorithms [4]. The equivalent of the optimisation algorithm in the mapping would be the Runtime Adaptation. If the Runtime Adaptation is an optimisation algorithm then yes, yes it would be learning. It might fit into the subcategory of Stochastic programming or Robust optimisation algorithms. The author's personal opinion is that this should not be considered as learning, it is purely reacting to the data, and is not sophisticated enough; But it shows that there is room for a learning algorithm, one that uses more data and considers the reaction of the algorithm as well.

The Tests showed that although the fully extended algorithm (myCCAM) got in essence only one parameter, it outperformed or at least was equal to the base CCAM algorithm. Comparing it to CCAM without the limiting assumptions (CCAMb) and Forgetful CUSUM over EWMA (FCoE) yields mostly equivalent results. In the few cases where they are ahead it is due to human adjustments in parameters that either stem from system knowledge, in this case noise parameters, or from an iterative approach, optimizing the parameters over multiple test-runs. This optimisation was not done to the fully extended algorithm (myCCAM), which had no adjustments made when comparing it to other algorithms. At the same time the test have shown that there are still fundamental flaws in the algorithm. These deficiencies became evident in the tests, all the more so when realistic signals were used. Especially long patterns with strong drift like behaviour seems to create problems. It is to be expected that patterns will occur in some real signals as they appear in the environment on a 24 hour basis.

6.2 Future Work

There are multiple possible improvements to be done:

- Change the algorithm to allow outliers while building a state, or redefine the conditions of a fully build state. By redefining this, sensitivity and minimal state length might be separated. This could lead to small states being found and acknowledged.
- Redefine how an old state is entered and what happens afterwards. A possible approach is to use time proximity to give a recent state the advantage on re-entering a state. This could reduce false positives. Also comparing the current state against all old states after re-entering it for a specific time period could decrease wrong state changes, which happen due to outliers and are kept by chance.
- Define a confidence for the state. At the moment there is only a confidence of how likely a value belongs to a state, but no confidence of how certain the algorithm is that the current state is actually a true state rather than an outlier only. Such a confidence would be distribution dependent.
- Analyse the Multi-Point Solution. By analysing it, it might reveal that a distribution specific solution has a better sensitivity, while at the same time has minimal to no disadvantages.
- Make the Behaviour Analysis for different distributions. Afterwards compare the optimal values of different distributions. This will probably lead to redefining the Runtime (parameter) Adaptation.
- Redefine Integrated Preprocessing. The currently defined version is insufficient for patterns and requires a very long history length. Separating this adaptation from

the states history might be a possible way to proceed forward. The resonance phenomena can possibly also be eliminated by using a sort of low-pass for the incline calculated over time. Such a low-pass would also make it possible to remove the delay of the algorithm, if changes are only allowed to happen slowly. However, if the slope of the distribution were to change drastically in the event of a jump, the post jump state might not be found due to the only slowly changing incline.

- Improve/Redefine Runtime Adaptation:
  - Increase the speed at which the parameters are found. This could be done by entering an algorithm state after being unable to find a signal state for a certain time, at which the the low-pass is "sped up", by changing the parameters to allow higher frequencies.
  - Make a distribution estimator and use statically defined methods to extract the required parameters and create functions accordingly. This could also be done by a neural network that could be trained with an optimal parameter-function selection defined by humans. It might also be an interesting idea to superimpose the resulting functions depending on the decision probability, as multiple different distributions might occur at the same time.
  - Use the results of the algorithm and the past to define optimal parameters. By using the past signal values and the past state information, outliers can be identified and more optimal parameters can be extracted as a result. At the same time it might be possible to detect jumps by using an acausal/offline algorithm to further be able to segment the data to extract optimal parameters. In other words to learn from hindsight. Such an algorithm might truly be called learning, but at the same time it is likely to require a lot more resources.

Bibliography

[1] M. Götzinger, N. TaheriNejad, H. A. Kholerdi, A. Jantsch, E. Willegger, T. Glatzl, A. M. Rahmani, T. Sauter, and P. Liljeberg. "Model-free condition monitoring with confidence". In: *International Journal of Computer Integrated Manufacturing* 32.4-5 (2019), pp. 466–481. DOI: 10.1080/0951192X.2019.1605201.

[2] W. Y. Svrcek, D. P. Mahoney, and B. R. Young. *A real time approach to process control*. eng. Third edition.. Chichester, England ; West Sussex, England: John Wiley & Sons, 2014. ISBN: 1118684753.

[3] Wikipedia contributors. *Automation — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Automation&oldid=1034438050. [Online; accessed 27-July-2021]. 2021.

[4] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, and S. Marcos. "Neural Networks and Nonlinear Adaptive Filtering: Unifying Concepts and New Algorithms". eng. In: *Neural computation* 5.2 (1993), pp. 165–199. ISSN: 0899-7667.

[5] S. J. Pan and Q. Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[6] P. Brown, J. Bovey, and X. Chen. "Context-aware applications: from the laboratory to the marketplace". In: *IEEE Personal Communications* 4.5 (1997), pp. 58–64. DOI: 10.1109/98.626984.

[7] Wikipedia contributors. *Context awareness — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Context_awareness&oldid=1027869538.

[8] B. Schilit, N. Adams, and R. Want. "Context-Aware Computing Applications". In: *1994 First Workshop on Mobile Computing Systems and Applications*. 1994, pp. 85–90. DOI: 10.1109/WMCSA.1994.16.

[9] L. M. A. and J. N. S. *Generalized methods and solvers for noise removal from piecewise constant signals. I. Background theory.* 2011. URL: http://doi.org/10.1098/rspa.2010.0671.

[10] L. M. A. and J. N. S. *Generalized methods and solvers for noise removal from piecewise constant signals. II. New methods.* 2011. URL: http://doi.org/10.1098/rspa.2010.0674.

[11] E. S. Page. "Continuous Inspection Schemes". In: *Biometrika* 41.1/2 (1954), pp. 100–115. ISSN: 00063444. URL: http://www.jstor.org/stable/2333009.

[12] G. A. Barnard. "Control Charts and Stochastic Processes". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 21.2 (1959), pp. 239–271. ISSN: 00359246. URL: http://www.jstor.org/stable/2983801.

[13] M. Pollak and A. G. Tartakovsky. *On Optimality Properties of the Shiryaev-Roberts Procedure.* 2007. arXiv: 0710.5935 [math.ST].

[14] Y. Xu and Q. Wang. "Asymptotical Optimality of Change Point Detection With Unknown Discrete Post-Change Distributions". In: *IEEE Signal Processing Letters* 27 (2020), pp. 695–699. DOI: 10.1109/LSP.2020.2983312.

[15] M. Pollak. "The Shiryaev-Roberts Changepoint Detection Procedure in Retrospect - Theory and Practice". In: *Second International Workshop in Sequential Methodologies (IWSM).* 2009. URL: http://mat.izt.uam.mx/profs/anovikov/data/IWSM2009/plenary%20papers/IWSMPollak.pdf.

[16] D. Leung and J. Romagnoli. "Chapter 6.4 - Fault Diagnosis Methodologies for Process Operation". In: *Software Architectures and Tools for Computer Aided Process Engineering.* Ed. by B. Braunschweig and R. Gani. Vol. 11. Computer Aided Chemical Engineering. Elsevier, 2002, pp. 535–556. DOI: 10.1016/S1570-7946(02)80024-4. URL: https://www.sciencedirect.com/science/article/pii/S1570794602800244.

[17] A. Anzanpour, I. Azimi, M. Götzinger, A. M. Rahmani, N. TaheriNejad, P. Liljeberg, A. Jantsch, and N. Dutt. "Self-awareness in remote health monitoring systems using wearable electronics". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017.* 2017, pp. 1056–1061. DOI: 10.23919/DATE.2017.7927146.

[18] A. Anzanpour, I. Azimi, M. Götzinger, A. M. Rahmani, N. TaheriNejad, P. Liljeberg, A. Jantsch, and N. Dutt. "Self-awareness in remote health monitoring systems using wearable electronics". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2017.* 2017, pp. 1056–1061.

Code of Conduct

Hiermit erkläre ich, dass die vorliegende Arbeit gemäSS dem Code of Conduct Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Daniel Schnöll
Wien, 29.09.2021