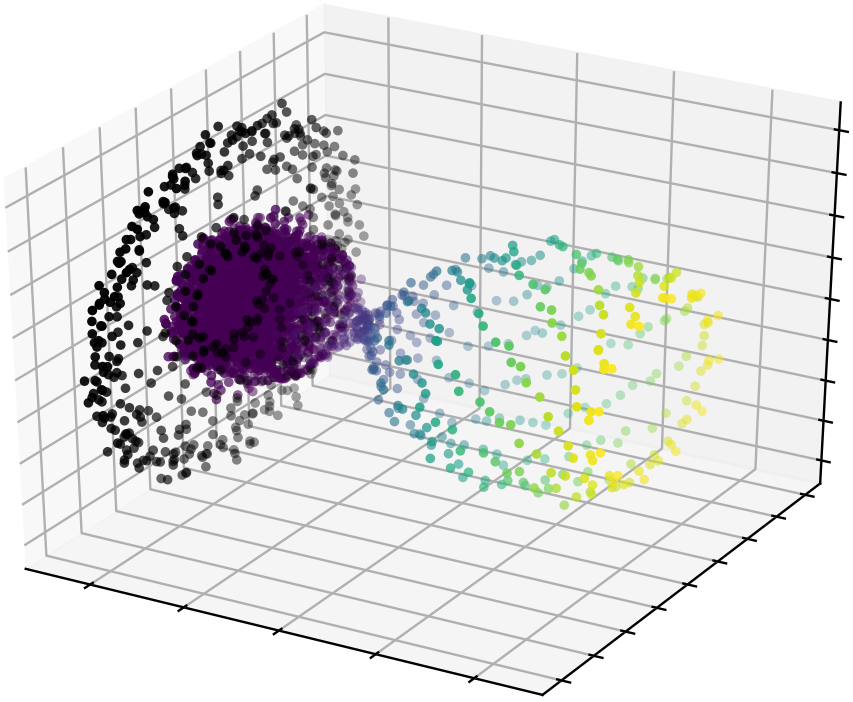

Process-level Anomaly Detection in Industrial Control Systems

Near real-time anomaly detection in large-scale industrial control systems

Master's thesis in Computer Science and Engineering

VIREN SINAI NADKARNI
DAVID STRÖM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Process-level Anomaly Detection in Industrial Control Systems

Near real-time anomaly detection in large-scale industrial control systems

VIREN SINAI NADKARNI
DAVID STRÖM

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Process-level Anomaly Detection in Industrial Control Systems
Near real-time anomaly detection in large-scale industrial control systems
VIREN SINAI NADKARNI
DAVID STRÖM

Supervisor: Magnus Almgren, Department of Computer Science and Engineering
Advisor: Wissam Aoudi, Department of Computer Science and Engineering
Advisor: Mattias Gustin, ABB
Examiner: Erland Jonsson, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Projection of PASAD subspace for anomalous sensor measurements in the Tennesse-Eastman process. Values with normal behaviour (indigo) cluster within trained model (black). Anomalous values (warmer shades) exhibit significant deviation from the trained model.

Typeset in LaTeX
Gothenburg, Sweden 2019

Process-level Anomaly Detection in Industrial Control Systems
Near real-time anomaly detection in large-scale industrial control systems
VIREN SINAI NADKARNI
DAVID STRÖM
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Over the last decade, Industrial Control Systems (ICSs), which manage critical infrastructure such as power, water and gas distribution systems, are increasingly being targeted by sophisticated cyberattacks. It is of paramount importance that necessary safeguards are in place for these systems to avoid potentially catastrophic damage. Intrusion Detection Systems (IDSs) can be used to monitor computer systems for signs of attacks and are commonly of two types: signature-based or anomaly-based. Signature-based IDSs work by using a database of known traffic patterns to identify malicious activity. Attacks against ICSs are specialised and crafted to exploit specific protocol semantics and setup. As such, building a signature database which incorporates all attack properties is difficult. This has led to a growing interest in doing anomaly-based intrusion detection using information from the industrial processes, such as sensor readings and control commands.

Research has shown that process-level anomaly detection can identify a large range of attack types, but so far there have been limited insights into whether process-level anomaly detection is suitable for modern ICS software. Questions such as if the cost of processing a large number of signals is reasonable, if it is feasible to integrate anomaly detection into existing ICS software, need a deeper understanding.

This study aims to evaluate the suitability of using process-level anomaly detection in production-grade ICS software. The platform is provided by ABB, a major international supplier of ICSs. We focus on two time series algorithms: Process-Aware Stealthy Attack Detection (PASAD) and Auto-Regression (AR) modelling.

Our findings show that both methods can successfully be used in large-scale ICS software. AR gives throughput one magnitude higher than PASAD, while PASAD is better at detecting stealthy attacks and attacks in noisy signals. PASAD can also leverage GPU capabilities, but needs buffering to outperform CPU implementations. The design of PASAD means that it requires a large amount of memory to model signals which have many values representing the normal behaviour. On the whole, we find that process-level anomaly detection can be a reliable complementary security mechanism for ICS deployments.

Keywords: anomaly detection, intrusion detection, industrial control systems, electrical grid

Acknowledgements

Our heartfelt thanks go to our supervisor, Magnus Almgren, for giving us the opportunity to do this thesis and being an awesome mentor; Wissam Aoudi for offering us insights from his anomaly detection research; Mattias Gustin, ABB, for providing access to the codebase that has formed the backbone of this study; and Nitesh Srivastava, ABB, for walking us through the codebase.

We would also like to thank our examiner, Erland Jonsson, for his valuable support and feedback.

Lastly, credits to Albin Hellqvist and Albert Overland for peer reviewing our report, and Daniel Posch and Jesper Rask for being the opposition in our defense.

Viren Sinai Nadkarni & David Ström, Gothenburg, June 2019

Contents

List of Figures **xiii**

List of Tables **xv**

Acronyms **xvii**

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Aim . . . 2
1.3 Contribution . . . 2
1.4 Scope . . . 3
1.5 Outline . . . 3

2 Background **5**
2.1 Intrusion Detection Systems . . . 5
2.1.1 Signature-based Systems . . . 6
2.1.2 Anomaly-based Systems . . . 7
2.2 Process-level Anomaly Detection . . . 7
2.2.1 Process-Aware Stealthy Attack Detection . . . 7
2.2.2 Auto-Regressive Modelling . . . 8
2.3 Thread Pools . . . 8
2.3.1 Lock Convoys . . . 10
2.4 OpenCL . . . 11
2.5 Industrial Control Systems . . . 11
2.5.1 Supervisory Control and Data Acquisition Systems . . . 12
2.5.2 Remote Terminal Units . . . 12
2.6 Case study: ABB Electrical Grid . . . 13
2.6.1 Remote Server Protocol . . . 13
2.6.2 Remote Communication Server . . . 13
2.6.3 PCU400: Process Communication Unit . . . 13
2.6.4 Drivers . . . 16
2.6.4.1 X05: Database Loader . . . 17
2.6.4.2 X97: Simulator Remote Terminal Unit Master . . . 17
2.6.5 Excel Data Engineering Tool . . . 17

3 Related Work **21**

3.1 Anomaly Detection in Production Settings . . . 21
3.1.1 Process-Aware Stealthy Attack Detection . . . 21
3.1.2 Auto-Regressive and Linear Dynamic Statespace Modelling . . 22
3.2 Anomaly Detection using Neural Networks . . . 23
3.3 Entropy-based Anomaly Detection Methods . . . 23

4 Design **25**
4.1 Choice of Anomaly Detection Methods . . . 25
4.2 Reference Implementation . . . 25
4.3 Core Implementation . . . 26
4.4 Modular Design . . . 26
4.5 Addressing Lock Convoys . . . 27
4.6 GPU-Accelerated Scheme . . . 28
4.7 Visualising Output . . . 29
4.8 Integration with PCU400 . . . 29

5 Implementation **31**
5.1 Reference Implementation . . . 31
5.2 Core Implementation . . . 31
5.2.1 Basic Linear Algebra Subprograms Libraries . . . 32
5.3 Processing Multiple Signals . . . 33
5.4 GPU-Accelerated Scheme . . . 33
5.5 Visualising Output . . . 35
5.6 Integration with PCU400 . . . 36
5.6.1 Integration with X97 . . . 36

6 Evaluation **39**
6.1 Setup . . . 39
6.2 Datasets and Correctness . . . 40
6.3 Scalability . . . 40
6.4 Basic Linear Algebra Subprograms Libraries . . . 42
6.5 Resource Utilisation . . . 44
6.6 Signal Frequency Distributions . . . 44
6.7 Simulating Anomalies . . . 47
6.8 Training and Subspace Sizes . . . 55
6.9 GPU-Accelerated Scheme . . . 57

7 Discussion **59**
7.1 Results . . . 59
7.1.1 Suitability . . . 59
7.1.2 Integration . . . 60
7.1.3 Accuracy . . . 61
7.2 Limitations . . . 61
7.3 Ethics and Sustainability . . . 62
7.4 Future Work . . . 62

8 Conclusion **65**

Bibliography **67**

A Observations **I**

List of Figures

2.1 PASAD and AR departure scores for a sample signal 9
2.2 Thread pool design 10
2.3 Example PCU400 setup 14
2.4 PCU400 architecture 15
2.5 PCU400 supervisor watchdog operation 16
2.6 PCU400 internals and drivers 18
2.7 Role of Excel Data Engineering Tool in PCU400 19

4.1 Hierarchy of modules 26
4.2 Multi-queue design 27
4.3 Integrating anomaly detection module in PCU400 30

5.1 Signal subspace projection for PASAD OpenCL 34
5.2 Parallel summing of array 35
5.3 Integrating PASAD in PCU400 37

6.1 Mean PASAD and AR throughput versus number of signals 41
6.2 PASAD throughtput for different BLAS libraries 42
6.3 AR throughtput for different BLAS libraries 43
6.4 Utilisation of CPU cores 45
6.5 Signal frequency distributions 46
6.6 AR on signal frequency distributions 47
6.7 PASAD on signal frequency distributions 48
6.8 Anomalous periodic signals 49
6.9 PASAD and AR departure scores for anomalous periodic signals . . . 50
6.10 PASAD projections for anomalous periodic signals 51
6.11 Noisy periodic signals 52
6.12 PASAD and AR departure scores for noisy periodic signals 53
6.13 PASAD projections for noisy periodic signals 54
6.14 PASAD throughput for various training and subspace sizes 56
6.15 PASAD OpenCL throughput 57

List of Tables

2.1 Low-level, high-level and external drivers supported by PCU400 . . . 17

A.1 PASAD throughput for various BLAS implementations II
A.2 AR throughput for various BLAS implementations III

Acronyms

AE . . . . . Auto-Encoder
AIC . . . . . Akaike Information Criterion
API . . . . . Application Programming Interface
AR . . . . . Auto-Regression

BLAS . . . . . Basic Linear Algebra Subprograms

CAG . . . . . Control Agent
CIC . . . . . Common Information Criterion
CLK . . . . . Clock Controller
CPU . . . . . Central Processing Unit
CSV . . . . . Comma Separated Values

DCU . . . . . Data Communication Unit
DOM . . . . . Document Object Model
DSP . . . . . Digital Signal Processor

FPGA . . . . . Field Programmable Gate Array

GAN . . . . . Generative Adversarial Network
GCC . . . . . GNU Compiler Collection
GPGPU . . . . . General Purpose GPU
GPU . . . . . Graphics Processing Unit

HID . . . . . Human Interface Device
HIDS . . . . . Host Intrusion Detection System
HLD . . . . . High-Level Driver

HMI . . . . . Human-Machine Interface
HTML . . . . . Hypertext Markup Language
HTTP . . . . . Hypertext Transfer Protocol

ICS . . . . . Industrial Control System
IDS . . . . . Intrusion Detection System
IED . . . . . Intelligent Electronic Device
IP . . . . . Internet Protocol
IPC . . . . . Inter-Process Communication
IPS . . . . . Intrusion Prevention System
IT . . . . . Information Technology

LAN . . . . . Local Area Network
LAPACK . . . . . Linear Algebra Package
LDS . . . . . Linear Dynamic Statespace
LLD . . . . . Low-Level Driver

MAC . . . . . Media Access Control
MinGW . . . . . Minimalist GNU for Windows
MKL . . . . . Math Kernel Library

NIDS . . . . . Network Intrusion Detection System
NTP . . . . . Network Time Protocol

OS . . . . . Operating System
OT . . . . . Operational Technology

PASAD . . . . . Process-Aware Stealthy Attack Detection
PCU . . . . . Process Communication Unit
PLC . . . . . Programmable Logic Controller
POSIX . . . . . Portable Operating System Interface

RCS . . . . . Remote Communication Server
RFC . . . . . Request for Comments
RSP . . . . . Remote Server Protocol

RTU . . . . . Remote Terminal Unit

SCADA . . . . . Supervisory Control and Data Acquisition System

SSA . . . . . Singular Spectrum Analysis

SUP . . . . . Supervisor

SVD . . . . . Single Value Decomposition

SWaT . . . . . Secure Water Treatment

TCP . . . . . Transmission Control Protocol

TRP . . . . . Trap Handler

UDP . . . . . User Datagram Protocol

XLD . . . . . External High-Level Driver

XML . . . . . Extensible Markup Language

1

Introduction

This decade has witnessed a proliferation of cyberattacks that have targeted national infrastructure. One such attack that gained global media coverage was the Stuxnet worm, which targeted enrichment centrifuges used by the Iranian nuclear program [30]. Since then, use of control systems for managing critical infrastructure has expanded, increasing the potential devastation that may be caused by exploiting these systems. It is therefore in every stakeholder's interest to secure these systems before they can be hijacked to cripple transportation networks, utility services, financial systems and cause damage to property and life.

The term Industrial Control System (ICS) is used to describe several kinds of control and instrumentation systems used for industrial process control. In industrial process control, automatised production of a product is performed while maintaining prescribed standards of safety and consistency. Such systems are controlled by Programmable Logic Controllers (PLCs) and Supervisory Control and Data Acquisition Systems (SCADAs), which are designed to offer high-level supervision from remote co-ordination centres. These systems are classified as Operational Technology (OT), rather than Information Technology (IT).

Historically, PLCs and SCADAs used to be locally managed and air-gapped from all networks. This caused a growing sense of false security leading more focus to be put only on performance and efficiency [10]. However, with the current trend of increasing connectivity, these systems are getting exposed to potential cyberattacks.

ICSs manage processes rather than data [46]. Because of the proprietary nature of protocols employed in ICSs and the rarity of attacks, there is a lack of necessary data and domain knowledge for traditional signature-based Intrusion Detection Systems (IDSs) to be an effective security mechanism. As such, recent research has looked into performing anomaly detection directly on process variables, built up from passively listening to changing variables within the ICS. The idea is that an attacker would not be able to completely hide the intended goal of somehow disrupting an ICS process — this would be apparent through a change in process variables. These detection methods have had limited testing in production environments, and there is a need to determine if such approaches would be suitable for a large-scale ICSs.

1.1 Motivation

Previous research has looked at whether process-level anomaly detection can realistically be used to find anomalies at a non-holistic level. This has involved implementing and testing such methods on a small test suite to prove their viability, often processing no more than a dozen signals in highly controlled environments. Resulting tests have shown that process-level anomaly detection methods can be used to accurately detect several types of attacks. What has not been studied so far is the associated overhead and limitations of these algorithms when running under the constraints of a more realistic scenario. In particular, our understanding of the behaviour of such methods in OT is limited. The importance of learning about these constraints is amplified by the fact that there are a few SCADA vendors in the world. This means that the findings of this study could have a large impact on the future of the global SCADA market.

1.2 Aim

The goal of this study is to investigate the suitability of using process-level anomaly detection in a production-grade ICS. ICSs have some of the strictest performance requirements and thus, it is important to determine if process-level anomaly detection is viable in large-scale settings.

In particular, this study aims to answer the following questions:

1. Which process-level anomaly detection methods are appropriate for ICSs?
2. Can process-level anomaly detection be integrated into commercial ICS software to achieve high performance?
3. How well do the selected anomaly detection methods handle various types of ICS signals?

Performance and scalability are critical to ICSs. Being complex distributed systems, thousands of readings are generated per second that are required to be monitored and logged without affecting normal operation. As such, it is necessary that the design and implementation of the accompanying anomaly detection system also have these traits. Part of our goal is to understand the implications of these requirements and be effective in integrating anomaly detection in ICS codebases.

1.3 Contribution

In order to assess the performance of process-level anomaly detection for large-scale ICSs, this thesis evaluates two process-level anomaly detection algorithms in production ICS software. This is preceded with a survey of existing anomaly detection algorithms that use various methods: statistical modelling, neural networks-based and

entropy-based. The intention is to justify the choice of these two specific anomaly detection algorithms based on the various constraints of OT settings.

We evaluate a thread pool scheme that uses independent workers to extract full performance from given hardware. This is followed by a GPU-based scheme to leverage its massive parallelisation capabilities. A set of quality metrics — throughput, impact of number of pool workers, utilisation ratio and scalability — have been measured. These measurements are then put into perspective by comparing them with the system requirements experienced by ABB, which is one of the largest SCADA suppliers in the world.

1.4 Scope

There exist a multitude of methods for performing process-level anomaly detection. The central motivation with the selection of algorithms relies on their ability to operate at an acceptable accuracy. It is also the case that many methods use neural networks or other elaborate techniques, which makes it challenging to reason about processes. This means that they are generally difficult to evaluate. In this thesis, only two methods for process-level anomaly detection have been chosen as representative algorithms. By focusing on these two algorithms, more time can be spent on the evaluation of each, rather than digging into details of alternative methods.

This study is carried out using production ICS software. The software supports relevant configuration modes, which are used in internal testing by the ICS manufacturer. As such, it can be assumed that it exhibits a high degree of realism. Using this approach allows for rapid prototyping and testing since no physical devices need to be placed and configured. Scaling is as easy as changing a few lines in a configuration file.

1.5 Outline

This report is organised into following chapters:

- Chapter 1 introduces the problem of identifying disruptors in industrial processes, the scope of this thesis and its contributions.
- Chapter 2 delves into background information and related core concepts. This includes intrusion detection systems, anomaly detection approaches, multithreading, and ABB architecture and terminologies.
- Chapter 3 presents other related techniques for performing anomaly detection which have not been focused on in this study.
- Chapter 4 lays out the high-level design of the anomaly detection modules that have been integrated into ABB software, particularly, for achieving high throughput and providing a visualisation interface.

- Chapter 5 discusses the low level specifics of integration with ABB software, implementation of the thread pool scheme, and GPU-accelerated anomaly detection.
- Chapter 6 elaborates on the metrics that are important for measurements, how measurement has been done, what simulation scenarios have been tested, and results of the simulations.
- Chapter 7 talks about the suitability of process-level anomaly detection in ICSs, limitations of this study, associated ethical challenges, and potential future work.
- Chapter 8 concludes the report with a summary of this study.

2

Background

This chapter introduces the background knowledge and concepts required to understand this study. We start with a quick review of Intrusion Detection Systems (IDSs) in Section 2.1. This is followed by an overview of anomaly detection methods that have been selected for evaluation in Section 2.2. Section 2.3 covers multi-threading designs used to maximise CPU utilisation. Section 2.4 covers OpenCL, a framework that allows the use of GPUs for general purpose computing. Section 2.5 explains Industrial Control Systems (ICSs) and its constituent components such as Supervisory Control and Data Acquisition Systems (SCADAs) and Remote Terminal Units (RTUs). The chapter concludes with Section 2.6, where a case study of a major SCADA supplier and their system terminology is presented.

2.1 Intrusion Detection Systems

Intrusion Detection Systems (IDSs) are software applications which monitor the network and perform analysis on systemic events in order to try and detect potential malicious activity. Some IDSs also perform network policy enforcement and raise alarms if any violations are identified. The idea is to warn the security team early enough so that proper actions can be taken to safeguard the network against attackers. Some types of IDSs can respond to attacks automatically by trying to hinder or block the attacks entirely. These systems are called Intrusion Prevention Systems (IPSs).

IDSs can be categorised based on its location within the network topology [9]. Host Intrusion Detection Systems (HIDSs) are deployed on end-points of the network, such as computers and terminals. They monitor the incoming and outgoing traffic for that specific system and send alerts to the administrator in case of an intrusion. Network Intrusion Detection Systems (NIDSs) are installed at strategic points in the network topology such as gateways, and can monitor the traffic flowing through the entire subnet.

Another way of classifying IDSs is based on the method used to detect intrusions: signature-based and anomaly-based [9]. Anomaly-based IDSs are effective in identifying zero-day attacks, which are exploits that are not known to the maintainers of the software and thus remain unmitigated. However, it also suffers from fairly

high false-positive rate because it can flag legitimate traffic that is not known to be *normal*.

There exist another type of IDSs called specification-based IDSs which are almost exclusively used for network intrusion detection [45]. Specification-based IDSs promise a low rate of false alarms in comparison to signature-based IDSs, but are less capable than anomaly-based IDSs in identifying novel attacks. These systems use *specifications*, which are basically extended finite state automata that represent valid and invalid states of respective protocols. Specifications are compiled from resources such as protocol standardisation documents and RFCs. The major disadvantage of specification-based IDSs is the need to manually develop the specifications, which is a time-consuming task.

2.1.1 Signature-based Systems

In signature-based IDSs, the network traffic is monitored for certain patterns and byte sequences. These patterns, known as signatures, need to be known in advance and are uniquely associated with past instances of attacks. As such, signature-based IDSs work best when the signature databases are comprehensive and up to date.

There are some problems associated with this approach. If there are a large number of attack patterns, then a significant amount of storage space is required for the signature database. An efficient algorithm is also required to search the attack pattern search space when trying to determine if new data is part of an attack or not. Moreover, signature-based IDSs are unable to catch zero-day exploits, i.e. newly constructed exploits.

Unfortunately, signature databases fail to capture the vast range of protocols, specifications, and design-specific contexts in OT domains. ICSs are proprietary in design and do not always use standardised protocols for communication. Furthermore, attacks against ICSs are rare, requiring extensive preparation and research due to relatively higher complexity, smaller attack surface, and necessity to exploit low-level semantics [10].

However, signature-based IDSs offer some redeeming features which make them useful despite their associated problems. It is comparatively rare for signature-based IDSs to raise false alarms given accurate rules. As such, if an alarm is raised, there is rightly a cause for concern.

As an example, there exists a popular open source signature-based IDS used to secure IT networks called Snort [11]. A rule in Snort could be as follows:

```
alert tcp 10.20.30.40 any -> any 443 ( msg:     "Alert!";
                                       content: "evilstring"; )
```

This rule will raise an alarm if there is an incoming TCP packet from `10.20.30.40` to any destination IP on the subnet for port `443`, with packet body containing the string `evilstring`. Similar rules would have to be crafted for each unique attack pattern.

There are comprehensive rule lists which are compiled and maintained by the security community [12]. Ultimately, this leads to the problem of having to know what types of attacks the system will be likely to face, and as such, choosing the right rule lists. Without proper knowledge, this could lead to some attack patterns not being accounted for, or for the rule lists to contain more rules than necessary, causing the system to waste resources.

2.1.2 Anomaly-based Systems

Anomaly-based IDSs learn how *normal traffic* looks like and aim to isolate abnormal and unusual observations [9]. It employs a statistical or machine-learning approach to build a model of legitimate behaviour of the system, and any traffic that does not fit into this model is treated as anomalous or malicious.

Since the normal behaviour is defined by what data is used for training, it is of utmost importance that this data does not contain any attacks and that it captures a complete view of the process behaviour. If there are attacks in the training data, then the same attacks will not be caught in the future. If the training data does not contain a holistic view of the process behaviour, a lot of false alarms will be raised, leading to efforts being wasted on non-existent problems.

The prime benefit that anomaly-based IDSs have over signature-based counterparts is their ability to identify zero-day exploits, i.e. novel and previously unknown attacks. This makes anomaly-based IDSs ideal for systems where historical attacks are scarce and the operational pattern is predictable to a high degree, such as ICSs.

2.2 Process-level Anomaly Detection

Process-level anomaly detection is concerned with detecting anomalies in an industrial process. This translates into processing time series of raw sensor values and control commands, rather than the higher level problem of detecting specific protocol exploits that signature-based IDSs commonly deal with. Industrial processes have an associated system of sensors and actuators with static topology, regular communication patterns, deterministic signal noise, and more. These characteristics allow for construction of algorithms that incorporate these assumptions into the anomaly model to give more accurate predictions.

2.2.1 Process-Aware Stealthy Attack Detection

In the paper by Aoudi et al., an approach by the name of Process-Aware Stealthy Attack Detection (PASAD) based on Singular Spectrum Analysis (SSA) and some common properties of ICSs has been explored [7]. PASAD tries to determine if drifting sensor values are caused by a change in the system behaviour. This is done

by learning the behaviour of the signal time series when represented as lagged vectors. The lagged vectors form a signal space which is decomposed into eigenvectors. The eigenvectors with largest contribution to the signal is extracted and forms the deterministic part of the signal subspace. All the lagged vectors used in the training phase are then projected into this subspace and averaged together, forming a centroid. New lagged vectors get projected into the subspace and are compared with the constructed centroid. If the newly arrived lagged vectors behave similarly to the learnt system behaviour, the projection will lie close to the centroid. If this is not the case, it indicates the occurrence of an anomaly.

2.2.2 Auto-Regressive Modelling

A well-known method for doing process based anomaly detection is the linear Auto-Regression (AR) modelling with control limits employed in the paper by Hadžiosmanovič et al. [21]. While this simple approach works for some attack patterns, it fails to detect more subtle attacks, such as ones which cause damage over time from small perturbations that fit within the accepted noise level of process variables [7].

In this method, a number of coefficients and an error bound are estimated from a time series of training data. These are then used in order to try and predict the next value in the time series. An auto-regressive model of order p would have the following formula:

$$x_i = \omega_0 + \omega_1 \cdot x_{i-p+1} + ... + \omega_p \cdot x_{i-1} + \epsilon \tag{2.1}$$

where i is the sensor value that is being predicted, ϵ is the error bound, and ω_j represents the jth coefficient.

There are variations to how the coefficients and control limits are chosen. The algorithms employed by Hadžiosmanovič et al. used Burg's method for estimating coefficients with Akaike Information Criterion for choosing the number of coefficients and Sherwart control limits.

Figure 2.1 shows PASAD and AR departure scores for an anomalous signal. PASAD is responsive and persistent to small variations in signal properties. This can also be seen when PASAD subspace is projected into three dimensions, as illustrated on the cover page. AR can initially detect the deviation in signal behaviour but fails to retain the departure score for persisted deviation.

In this report, N indicates training size, L denotes lag vector size and r indicates subspace size. Unless explicitly stated otherwise, the lag vector size used is half the training size.

2.3 Thread Pools

Modern computers are equipped with multiple processing cores per CPU. Multi-threading is a common approach to achieve high levels of concurrency and utilising

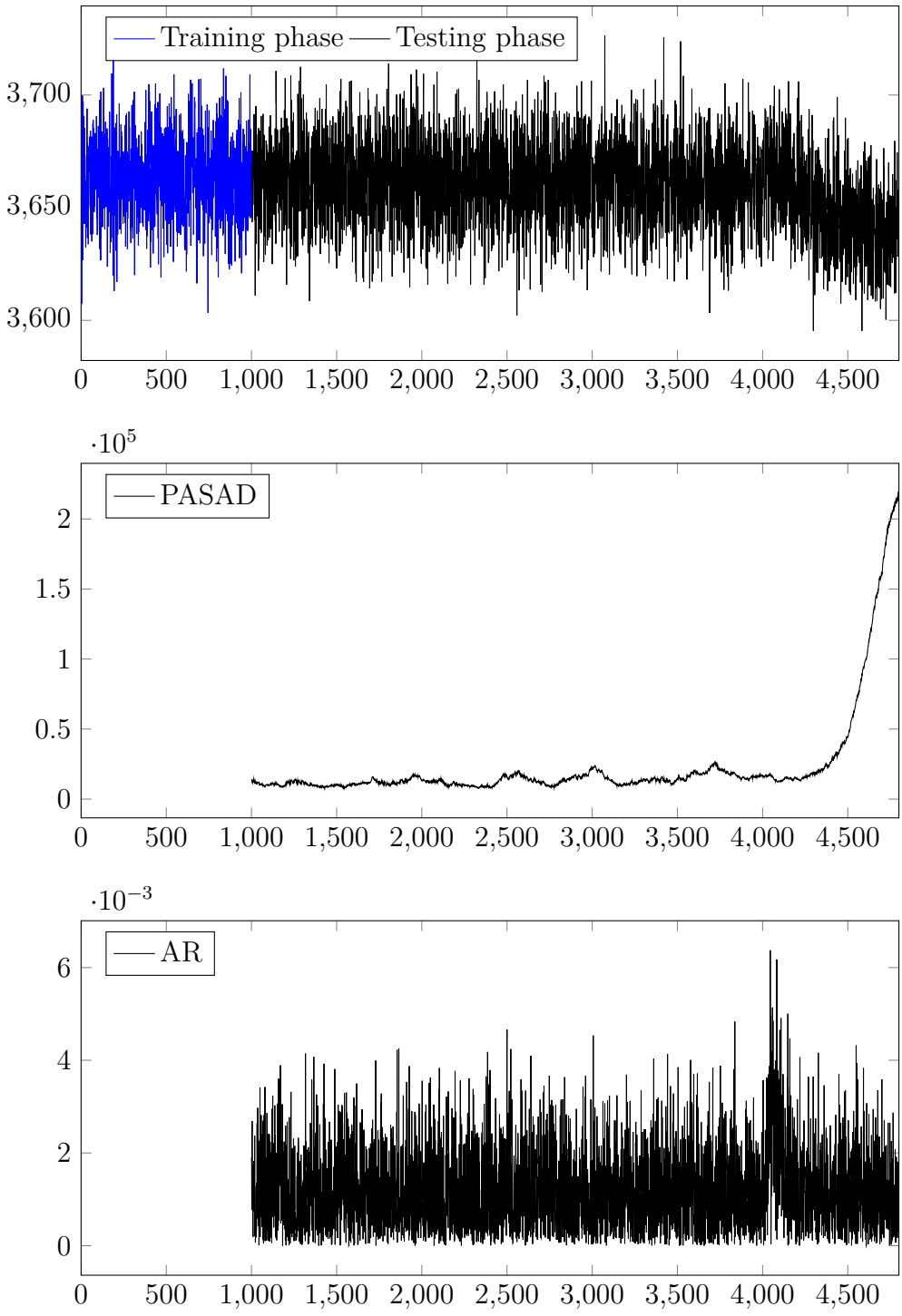

Figure 2.1: PASAD and AR departure scores for a sample signal ($N = 1000$, $L = 500$, $r = 30$). PASAD subspace can also be projected into a 3D space, as shown on the cover page.

all available cores. Each thread has its own programmatic sequence of instructions and follow an independent execution flow.

There is additional overhead when repeatedly spawning and destroying threads. It is particularly wasteful when the threads perform a single function and are short-lived. A way to minimise this unnecessary overhead is to maintain a pool of threads that are reused after every execution. This design paradigm is called *thread pooling*. Incoming work items to be processed are inserted into a queue. A worker thread picks a work item from the front of the queue and begins processing. On finishing, the worker picks the next work item from the queue, or goes to standby if the queue is empty. The number of worker threads is set as per available CPU cores. This idea is illustrated in Figure 2.2.

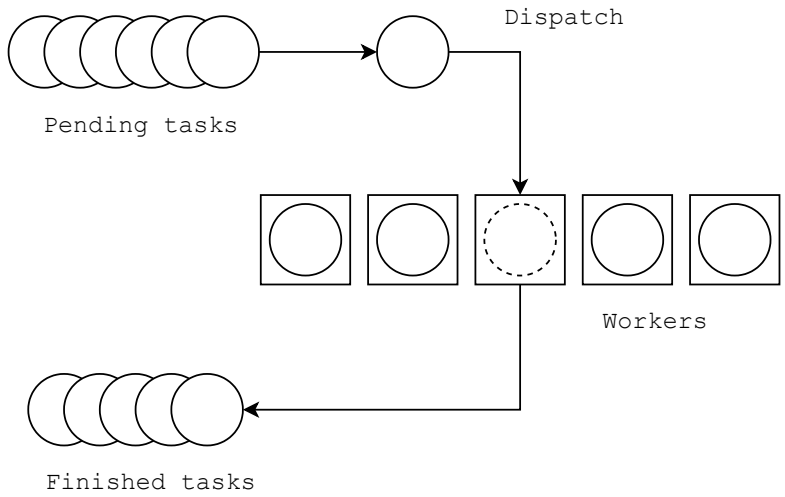

Figure 2.2: Thread pool design. Standby worker threads pick tasks from the queue and process them.

If multiple worker threads try to access a shared resource at the same time, it may lead to undefined behaviour. This contention between the threads is called *racing*. The shared resource could be an external device such as a camera or a printer, or it could be a block of memory. The sections of program code which perform the access to these shared resources are termed as *critical sections*. OSs provide synchronisation structures such as mutexes and semaphores to avoid such contention. They work by allowing only a single thread to be present in the critical section at a time.

2.3.1 Lock Convoys

A side effect of using mutexes is the *lock convoy effect*. Every time a thread tries to acquire a lock currently held by another thread, it fails and causes a context switch by giving up its time-slice. These repeated context switches and failure to use the assigned scheduling time-slice can cause a significant drop in performance. When multiple threads are processing related work items, only the thread holding the lock will make progress, while the remaining threads will wait instead of spending that

time processing other items. Unlike deadlocks — where execution flow grinds to a halt — lock convoys do make progress, but cause CPU cycles to be wasted.

2.4 OpenCL

OpenCL is a standard and a computing framework that allows cross-vendor and cross-platform heterogeneous parallel computing, providing uniform access to CPUs, GPUs, FPGAs, and DSPs [19]. Even though GPUs operate at a lower clock frequency than CPUs, they consist of far larger number of cores, which allows greater parallelisation and faster processing of stream pipelines. Their use has traditionally been limited for graphics processing such as video games. These pipelines were later found to be also suitable for mathematical and scientific workloads. With OpenCL, the idea is to allow easier means to offload computations to GPUs to improve performance.

The OpenCL standard is maintained by the Khronos Group, a non-profit consortium made of representatives from various hardware and software companies from the industry. The latest stable release is OpenCL 2.2. [18]

OpenCL provides an abstraction layer for a uniform programmatic access to computing devices, irrespective of their build and architecture [48]. It makes a distinction between *host device* (i.e. CPU), and *computation unit* (i.e. GPU). In a similar manner, program logic is also separated into *host code* and *kernel*. Host code can be written in any high-level language and makes use of language-specific OpenCL APIs. The kernel is implemented in OpenCL C, a superset language of C99. This dialect is designed to enable high parallelism by providing specialised data structures for synchronisation among the pipelines. It has a modified C standard library with additional functions for mathematical and scientific workloads, and lacks support for pointers, recursion, and variable-sized arrays.

Within the kernel, computation logic is split into *work-items*. The computation unit manages multiple work-items in groups called *work-groups*. The memory follows similar hierarchical structure and is divided into *global*, *local* and *private* memory. The global memory is large and can be read by any work-item, but is slow and computationally expensive. The local memory is limited and shared by all work-items in the same work-group. Finally, each work-item has its own private memory, access to which is restricted to the that work-item.

2.5 Industrial Control Systems

Industrial Control System (ICS) is a collective term that encompasses various systems that are used to control an industrial process. An industrial process is a series of steps which, when followed, enable controlled production of some goods or services. A typical industrial process would be the manufacturing of some product —

such as a car — but could also be how an electric utility company makes sure power is reliably and safely delivered.

Modern industrial processes are generally controlled by Programmable Logic Controllers (PLCs) or Remote Terminal Units (RTUs). In the case of a manufacturing plant, a single controller would be responsible for managing and making sure that all steps in the industrial process are working as intended. The ICS would in turn be the collection of all controllers, communication lines and master stations connected to them.

ICSs are generally closed-source, proprietary, and run non-standard protocols that rely on security-by-obscurity [10]. Furthermore, attacks on ICSs tend to be rare and highly specific. For these reasons, there is a lack of necessary data and domain context to build signature-databases, and hence, it is harder to maintain signature-based systems for ICSs.

2.5.1 Supervisory Control and Data Acquisition Systems

Supervisory Control and Data Acquisition Systems (SCADAs) are a component of ICSs that are critical in modern industrial infrastructure and control everything from water, electricity, gas distribution, and more [29]. Modern SCADAs use real-time information from the industrial process in order to be efficient and refine performance.

SCADAs have a hierarchy of control layers that manage the industrial process at different fidelities. The idea is to provide full system information and allow complete control from a centralised location.

Typically, the top layer has a Human Interface Device (HID) which allows for a high-level overview and control of the process. The lowest levels are where sensors, control valves and other field devices exist. The intermediate levels facilitate grouping of lower layers in order to turn the high-level commands from the top layer into executable instructions for the lowest layer.

In this report, the terms *SCADA*, *central stations* and *master stations* have the same meaning and are used interchangeably.

2.5.2 Remote Terminal Units

Remote Terminal Units (RTUs), also referred to as Intelligent Electronic Devices (IEDs), are devices that monitor and control ICS field components. RTUs relay analogue and digital signals, metrics, and control commands between master stations and actuators. RTUs also compile telemetry data from sensors which is used to monitor the overall health of an ICS. Most modern RTUs have some degree of automation and perform a closed-loop control.

2.6 Case study: ABB Electrical Grid

ABB is a multinational corporation operating in equipment and services for industrial control and automation. Because they are one of the leading ICS suppliers in the world, they are a good representative of the design and scale of large-scale ICSs.

This section explores the protocols, modules and terminologies employed by ABB in their commercial SCADA system. The focus is on PCU400, a general purpose Process Communication Unit (PCU) designed and manufactured by ABB. We cover the internal architecture of PCU400, its high-level operation, and configuration process.

2.6.1 Remote Server Protocol

Remote Server Protocol (RSP) is a communication protocol used internally within the PCU400 system [23]. It is based on RP-570, a now deprecated protocol that connected sub-stations with front-end computers [4]. Two versions of RSP exist: RSP v1 is the legacy version that lacks support for RTU floating point data. RSP v1 suffers from out-of-order issues because packets are not timestamped. RSP v2 allows use of floating point data and uses timestamped packets. In addition, it supports custom control commands and network-level multi-casting, switching and redundancy.

All internal communication in PCU400 happens in RSP. Data from RTUs and master stations that use third party protocols are first converted to RSP before analyses and relays.

2.6.2 Remote Communication Server

Remote Communication Server (RCS) refers to the set of components that interface with SCADA. On production systems, RCSs connect to PCUs over RSP buses. Primary responsibilities of RCS include controlling the PCU, uploading configuration databases to the PCU, SCADA supervision, and routing the messages originating in SCADA. Because there are multiple physical communication lines connecting to SCADA, part of the routing process involves forwarding messages over the correct communication line.

In the setup used in this thesis, RCS is replaced by the X05 Database Loader. X05 is covered in Section 2.6.4.1.

2.6.3 PCU400: Process Communication Unit

PCU400 is a general purpose PCU capable of maintaining communication with RTUs, IEDs and substations. It can act as a protocol converter, a concentrator or a

communication front-end for SCADA. It also offers a Human-Machine Interface for data supervision, simulation and process control [3].

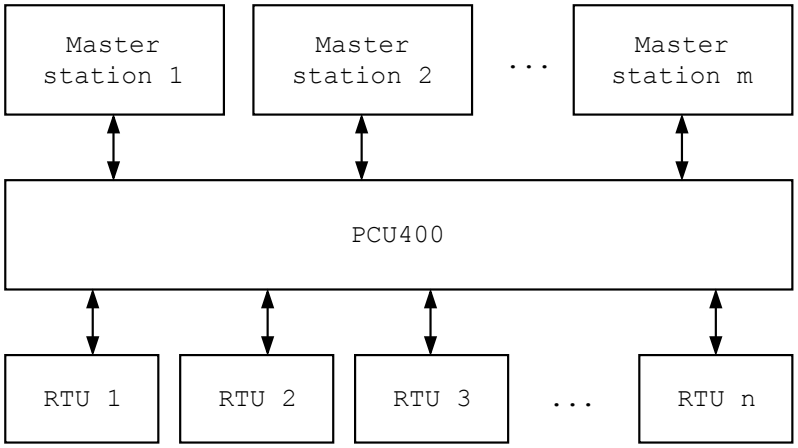

Figure 2.3: Example PCU400 setup.

In a typical configuration, PCU400 maintains links with one or more master stations on one end, and RTUs on the other end. This is illustrated in Figure 2.3. In this configuration, PCU400 acts as a multiplexer, i.e. data from RTUs are spread out to all the connected master stations.

There are several manufacturers of RTUs in the market that cater to a wide variety of industrial applications. These RTUs employ the use of different proprietary protocols that do not always adhere to standards. As such, PCU400 has built in support for a large number of protocols. Furthermore, it provides necessary APIs and runtime environments so that non-standard protocols can be supported.

PCU400 supports Microsoft Windows 7 or later [3]. Although inherently a x86 (32-bit) software, it can run in x64 (64-bit) environments. It makes use of multiple Windows services, such as system clock, network stack, and standard drivers through the Win32 API. The internal system itself consists of several components as shown in Figure 2.4. All of these run as independent userspace processes and communicate amongst themselves via shared memory. These components are described in greater detail below:

Supervisor The Supervisor (SUP) is a software watchdog that monitors other processes in the system. All supervised processes periodically send a heartbeat message to the supervising process as illustrated in Figure 2.5. The SUP directly supervises Control Agent (CAG), Clock Controller (CLK), Trap Handler (TRP) and Data Communication Unit (DCU). On the other hand, the DCU supervises all External High-Level Driver (XLD) processes. Any terminated process is automatically restarted by the supervising process. SUP itself can optionally be supervised by a hardware watchdog. This essentially works by SUP relaying heartbeats to the hardware watchdog. Absence of heartbeats from SUP causes the hardware watchdog to initiate a complete system reboot.

Control Agent The CAG provides local/remote control, logging and diagnostic

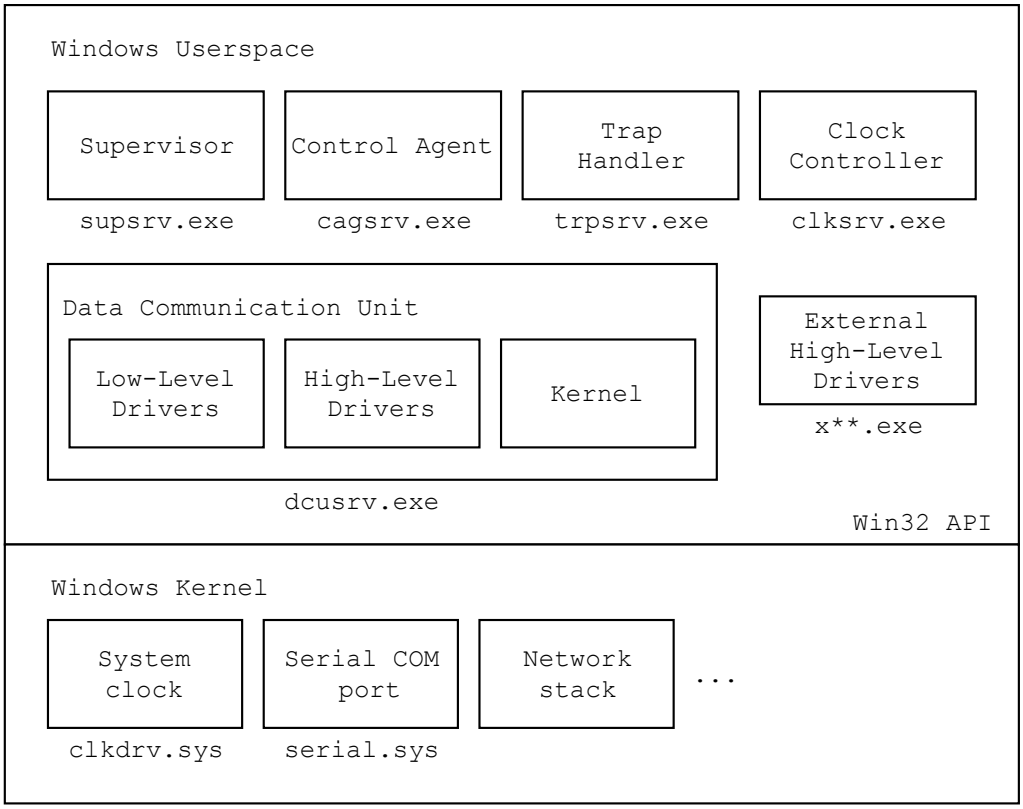

Figure 2.4: PCU400 architecture. Each service component runs as a userspace process.

services for the PCU400 system.

Trap Handler The TRP archives error messages from the various processes within the PCU400 system. These messages can assist in analysis in event of a major issue. All messages above a pre-defined threshold importance level are unconditionally recorded by TRP.

Clock Controller The CLK provides a high-accuracy clock for all PCU400 processes. CLK is independent from the operating system clock and supports synchronisation via minute pulses over the parallel port. High-accuracy synchronisation can be done over Network Time Protocol (NTP).

External High-Level Drivers PCU400 protocol specifications can be implemented to run a individual process. Such processes are termed as XLDs. XLDs are discussed in greater detail in Section 2.6.4.

Data Communication Unit The DCU consists of the kernel and handlers for different protocols. The DCU is responsible for task scheduling, internal message passing, and providing runtime environment and libraries for protocol handlers and other PCU400 internal processes.

Low- and High-Level Drivers Communication port drivers consist of various Low-Level Drivers (LLDs) for physical ports such as serial ports and Local

Area Network (LAN). Drivers are discussed in greater detail in Section 2.6.4.

PCU400 also provides web-based access for simulation, supervision, and remote control of the system. An interface is provided for controlling and viewing status information for communication lines, RTUs, and process objects. There is also support for simulation of changes in process objects; this stops communication with process objects and simulated master station connections can be tested. Various security safeguards are built-in which include password authentication, IP address whitelisting, ability to use a non-standard HTTP port, and appropriate warnings for critical actions.

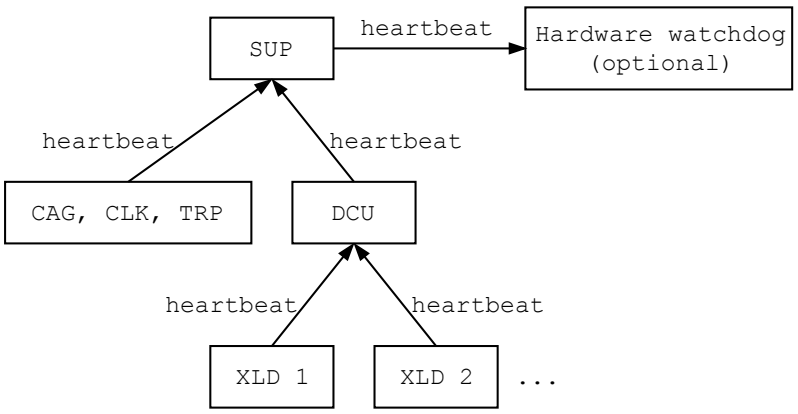

Figure 2.5: PCU400 supervisor watchdog operation.

2.6.4 Drivers

In the context of PCU400, communication protocols put forth the rules of intercommunication with software modules, local and remote devices over physical lines. Examples include RP-570 (which is used for RTU communication), RSP, and standard internet protocols such as TCP/IP. For protocol support, PCU400 systems make use of *drivers* (also referred to as *handlers*). These are separate software modules that describe the logical processes for communicating in that protocol. High-Level Driver (HLD) and External High-Level Driver (XLD) are two ways of implementing these modules on the architectural level. The internal organisation of drivers in PCU400 is shown in Figure 2.6

Within the PCU400 system, RSP is used for all internal communication. Both HLDs and XLDs essentially translate the external protocol to RSP. An HLD runs as an internal task within the DCU process, while XLD runs as a Windows process external to PCU400 system. Functionally, both XLD and HLD are equivalent, but differ in implementation, and this difference is transparent to the end user.

By convention, the protocol handlers are named as *Hnn*, *Lnn* or *Xnn*. The *H* in the name means that a handler is an HLD, *L* stands for LLD and *X* stands for XLD. *nn* is a number between 0 and 99. All XLDs communicate with PCU400 over a special HLD, H01. Furthermore, all XLDs and HLDs are associated with an LLD

to facilitate sending and receiving data over physical ports. Some of the supported LLDs are listed in Table 2.1.

Table 2.1: Select low-level, high-level and external drivers supported by PCU400.

Interface	Driver ID
Windows serial port	L41
TCP/IP socket server	L45
TCP/IP socket client	L46
UDP/IP socket client/server	L47
Kerberos socket server	L49
Switched line	L71
RP570 RTU Master Emulator	H13
RP570 RTU Master	H20
Database loader	X05
Modbus Serial RTU Master	X30
Modbus IP RTU Master	X31
RP570/71 RTU Master	X57
Simulator RTU Master	X97

We now describe two protocols that are in scope of this thesis: X05 database loader and X97 simulator RTU master.

2.6.4.1 X05: Database Loader

The X05 module is responsible for reading the XML configuration generated by the Excel Data Engineering Tool and loading it onto the PCU400 protocol module databases [2]. This data contains configuration parameters that are used by protocol modules when they initialise. Once the protocol modules finish loading, PCU400 is ready to begin handling communication along the connected communication lines and attached devices.

2.6.4.2 X97: Simulator Remote Terminal Unit Master

The X97 module in PCU400 simulates RSP data from RTUs, and is typically used in testing scenarios. It works by receiving a database specification of the data and sending simulation data to the RCS. The behaviour of the simulation output can be controlled by the Excel Data Engineering Tool. Some example parameters for the simulator output are the number of indications per second, duration of signal, output value step size, etc.

2.6.5 Excel Data Engineering Tool

The Excel Data Engineering Tool is used to draw up schematics for process data and flows for PCU400 [1]. It supports stand-alone, non-redundant PCU400 con-

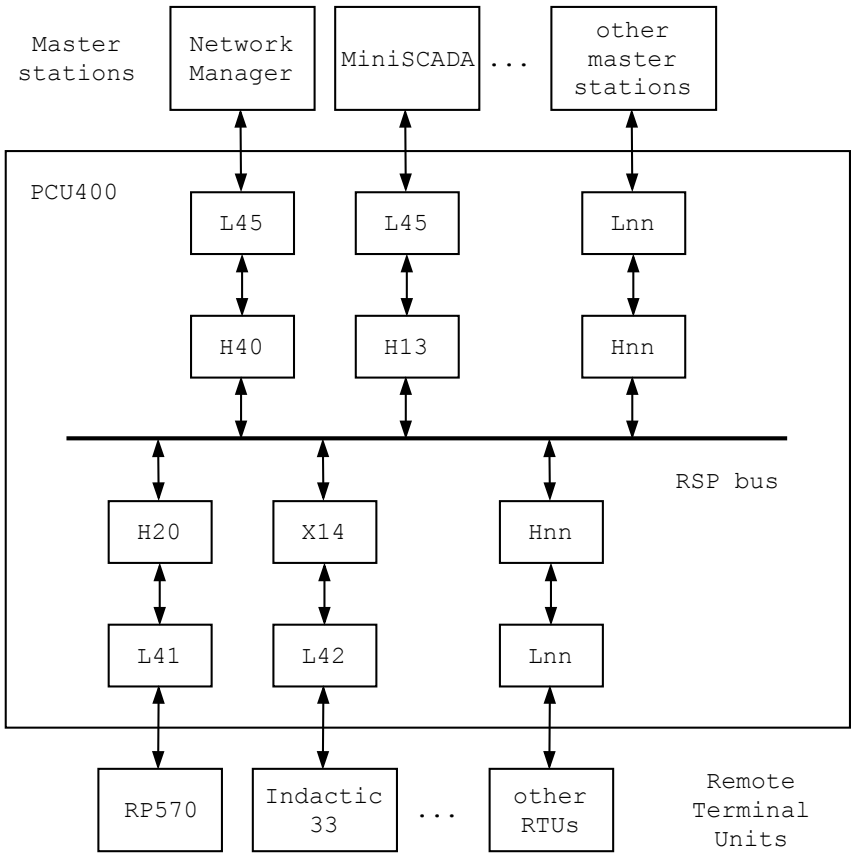

Figure 2.6: PCU400 internals and drivers. XLDs and HLDs convert all communication to RSP, while LLDs allow interfacing with hardware ports.

figurations; this includes the gateway mode configuration used in this study. The tool requires Microsoft Excel 2000 or later, and has been written in Visual Basic for Applications, an event-driven programming language based on Visual Basic 6. The data engineered by the tool is exported as an XML file, referred to in the PCU400 ecosystem as a *database*. The X05 database loader reads this XML database and pushes the configuration to the PCU400. This provides the PCU400 protocol modules with necessary data to communicate with connected components, such as RTUs. The process is illustrated in Figure 2.7

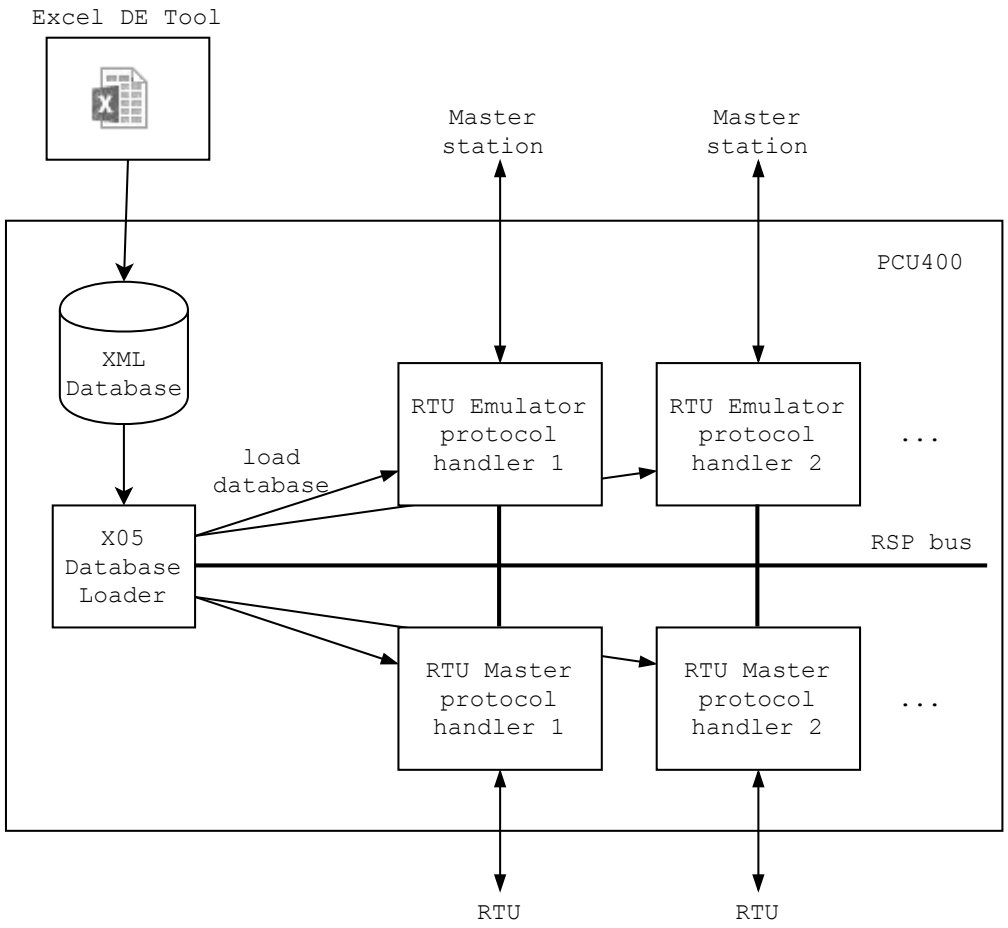

Figure 2.7: Role of Excel Data Engineering Tool in PCU400 for simulating RTU data.

3

Related Work

The use of time series anomaly detection at process-level is a relatively novel approach. In this chapter, we briefly present related research in this area. Section 3.1 provides an overview on studies that have involved deployment of PASAD and AR in production environments. Section 3.2 talks about studies in which neural networks have been used to perform anomaly detection. Finally, in Section 3.3, we cover studies that achieve anomaly detection by measuring the variation of system entropy.

3.1 Anomaly Detection in Production Settings

This section summarises related experiments that have tested time series anomaly detection in live production systems. We begin with a PASAD study in which anomalies were identified in the operation of a paper factory. This is followed by a study on Auto-Regression and Linear Dynamic Statespace modelling, two other time series approaches that have been tested in production settings. These studies have been chosen particularly for their similarity with this thesis in regards to their setup, i.e. either a live or a simulated production system. Performance and scalability are not the focus of these studies, in contrast to the goal of our study.

3.1.1 Process-Aware Stealthy Attack Detection

Almgren et al. were the first to run PASAD in a production environment [6]. PASAD was deployed at a paper factory in Sweden and observed over a span of several months. The ICSs in this case were ABB PM866 controllers with Modbus for intercommunication. Modbus is a widely used PLC communication protocol that can operate over Ethernet and TCP/IP. The Modbus protocol defines various types of data traffic but only the types that represent a continuous stream of values are useful for time series anomaly detection.

Data was recorded by intercepting the Modbus traffic using a packet capture and post-processing system. This system, based on libpcap and Bro, was installed on a Raspberry Pi hooked to data lines [42, 47]. A PASAD model was trained using this data over a span of eight days. The departure threshold was determined by

analysing 15 days of normal operation and setting the value to slightly over the maximum attained score. This was followed by a continuous run of the system for 75 days which was interspaced with regular scheduled downtimes.

It was observed that PASAD was able to handle these downtimes gracefully. The authors highlighted the difficulty of distinguishing interrupted signal stream (as is the case with downtimes) from operational failures of the anomaly detector. This study successfully demonstrated that PASAD could be installed in a production setting. A notable point is that this setup processed a single signal stream, while our study aims to analyse how PASAD handles a large number of signal streams. Another difference is the hardware: our study is based on ABB PCU400, a production software that runs on a server-grade computer, and not a resource-constrained single-board device like the Raspberry Pi. This difference makes a significant impact on the ability to process a larger number of signals. In production, PCU400 systems handle data points at rates that are in the magnitude of millions per minute [15]. As such, our study deals with situations where high frequency of incoming signal values is expected.

3.1.2 Auto-Regressive and Linear Dynamic Statespace Modelling

In the study by Urbina et al., the authors explore physics-based approaches for detecting stealthy attacks on ICS processes [49]. System identification was used to try and determine a physical model for the system being monitored. This was constructed using either AR modelling or Linear Dynamic Statespace (LDS). Both methods were tested using a stateless and stateful approach. Evaluation was performed on a *traditional attack* model and the proposed *stealthy attack adversary* model.

Both setups were tested in three configurations: a small testbed of a water treatment plant, real world data from a large scale SCADA system of Modbus traffic from a U.S. water treatment plant, and simulations of an electrical power grid. The testbed used was controlled by a total of six main PLCs and six backups, one for each stage in the water treatment process. Each of the PLCs dealt with only a single sensor and actuator. The large number of signals managed in this setup makes it similar to this thesis in the scale of data processed. The final tests were performed on a simulation of a single signal, i.e. the primary electrical frequency in a power grid under an attack based on one used in the joint military training exercise *Aurora* [56]. Their findings showed that stateless approaches have poorer performance than their stateful counterparts for all of the tests performed.

3.2 Anomaly Detection using Neural Networks

One of the goals of this thesis is to understand and incorporate the performance and system requirements posed by a large-scale ICS. This includes not only having low false-positive rate, high accuracy, and high performance, but also more subtle things, such as reproducibility and ease of analysing the model. Methods based on neural networks are generally an ill fit due to their complex relationship between component layers, neurons, and output, making analysis of attack detection difficult. In contrast, for PASAD and AR modelling, identifying why an input signal was marked as anomalous is more straightforward. This is because the PASAD and AR models rely on deterministic calculations, which can be interpreted and justified.

Generative Adversarial Neural Networks: Li et al. conducted a study about using Generative Adversarial Networks (GANs) to perform anomaly detection on multivariate time series [31]. The proposed method models both the generator and adversary as a Long Short Term Memory-Recurrent Neural Network. These were trained as a two-player zero-sum min-max game and were tested on part of the Secure Water Treatment (SWaT) dataset [17]. Their method had a detection accuracy of at best 94%, which is low compared to the one obtained from PASAD, and about the same as the one obtained for AR modelling on the same dataset.

Auto-Encoders: An approach based on Auto-Encoders (AEs) was explored by Oh and Yun, in which the authors try to detect failing machinery based on the sounds it makes [37]. The idea was that anomalous sound would give a high reconstruction error when decoding. The method was tested on sounds recorded from inside a Surface-Mount Device assembly machine which is responsible for mounting electronics components on a printed circuit board. Results showed that even with a low sensitivity threshold, their method could detect the abnormal intermittent clacking sounds in real time. The sounds of running without lubricants, like grease, were much harder to detect outright, but could be detected from analysis over a longer time period. Unlike the GAN approach, this method has a continuous output, making it easier to compare the relationship between input and its output. Despite this advantage, it suffers from a complex inner model which is hard to analyse. There is also the non-trivial problem of having to design the convolutional neural network used for learning the system behaviour. To get a well functioning model, this approach requires a tailor-made convolutional neural network design for each signal to be analysed. This is in contrast with AR modelling and PASAD, which require no signal specific configuration and have simpler inner models for analysis.

3.3 Entropy-based Anomaly Detection Methods

In the context of information science, *entropy* refers to the degree of randomness or disorder. Two entropy-based anomaly detection methods were explored in a paper by Agogino and Tumer [5]. The first method worked by measuring system-wide Shannon entropy, while the second method used an automated clustering technique

called Pearson Correlation, which calculated Shannon entropy for each separate cluster. The methods were tested on data captured from 148 sensors for a duration of 520 seconds on three separate test firings of a shuttle engine. The data contained an unknown ratio of anomalies added by a separate testing team. For each signal, 13,000 samples were taken for each test. Just like AR modelling and PASAD, this method is agnostic to the specific signal types being analysed. In contrast to the previous two methods, it has trouble distinguishing entropy generated from valid system transitions to those generated by anomalies. This makes it ill-suited for systems where frequent system changes expected. This is also true in systems with predictable system changes where an attacker could time the attacks to occur in lockstep with the system transition to hide the attack. This is not a problem with PASAD and AR, since system changes are incorporated into the model and are unlikely to be detected as anomalies.

4

Design

This chapter delves into the high-level designs that were formulated with an aim to achieve high scalability and performance. These qualities are important for Industrial Control Systems (ICSs) that are deployed in production, which, in some cases, receive an extremely large number of signal values per unit time.

The chapter starts with Section 4.1, where we justify the choice of PASAD and AR for evaluating process-level anomaly detection in ICSs. Sections 4.2 and 4.3 detail the design for reference and core implementations respectively. Section 4.4 provides an overview of the hierarchy of designed modules. Section 4.5 describes an alternate design which avoids the lock convoy effect arising from thread pools. In Section 4.6, an experimental scheme to run PASAD on GPUs with OpenCL is presented. Section 4.7 specifies how signal input and anomaly detection output will be visualised using a near real-time graph plotting tool. Finally, Section 4.8 covers how anomaly detection modules have been integrated into the PCU400 system.

4.1 Choice of Anomaly Detection Methods

Large-scale ICSs have high performance requirements. Because of this, the chosen process-level anomaly detection method needs to be fast, accurate, and easy to analyse. From the process-level anomaly detection methods described in Chapter 3, this property seems to best fit AR modelling and PASAD, as they outperform the other algorithms surveyed in some or all of speed, accuracy, and simplicity. As such, AR modelling and PASAD act as representative for process-level anomaly detection algorithms in this study.

4.2 Reference Implementation

Reference implementations are used in software development as a correct and idealised version of the program logic. All requirement-specific derivations and customisations are based upon reference implementations. It makes sense to have reference implementations in high-level languages. This is because high-level languages generally provide all the convenience features for performing, e.g. mathematical operations. This eliminates the need to be concerned with low-level details of the

exact implementation of mathematical operations and allows full focus to be put on implementing the actual algorithm.

In order to verify implementation correctness of PASAD and AR, reference implementations have been written in Python. Python has been chosen due to its flexibility and vast collection of libraries that support scientific computing.

4.3 Core Implementation

Core implementations are made with specific environments and requirements in mind. In our case, the core implementations for PASAD and AR would be integrated with existing systems to carry out anomaly detection. Ideally, this would be done so that it seamlessly integrates with the system structure and workflow. Therefore, the key priorities for developing the core implementation are performance, ease of integration, and scalability. Low-level languages offer the highest degree of control over program implementation at a machine level, generally giving it better performance than higher level languages. It also gives maximum control over execution environments by allowing finer control in the form of low-level routines, such as memory management and execution flow control with threading constructs. For these reasons, C and C++ have been used for writing the core implementation.

4.4 Modular Design

In order to make the use of PASAD and AR modules as flexible as possible, they have been implemented as modular libraries. This way, they could either be used in stand-alone form, or they could be imported into another codebase.

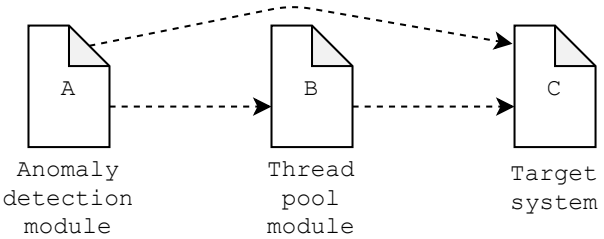

Figure 4.1: Hierarchy of modules

As illustrated in Figure 4.1, the modules have been organised as follows:

(A) contains anomaly detection classes and associated routines.

(B) contains high-performance threading scheme to ensure maximum CPU utilisation, and imports (A).

(C) is the target system. As per requirements, it can either only import the anomaly detection module (A), or the thread-pool implementation (B) which in turn will import (A).

4.5 Addressing Lock Convoys

ICSs deployed in production receive a high number of signals and signal values from various parts of the system. These signals are monitored for telemetry and to ensure that the system is operating within safe limits. In order to ensure quick response to potential anomalies, it is important that the system design is able to provide a throughput that is high enough to enable near real-time anomaly detection. One way to achieve high throughput is to use multi-threading or multi-processing for handling the incoming signal values in parallel. For multi-threading, we use the thread pooling scheme described in Section 2.3 due to its flexible processing of work items — in our case, enabling threads to work on any incoming signal.

In an effort to combat the lock convoy effect occurring in the thread-pool scheme, an alternative approach has been proposed, but not implemented. In this, a single thread would exclusively be responsible for processing multiple number of signals. Worker threads would never have to wait for each other and would always be able to make progress. This idea is illustrated in Figure 4.2.

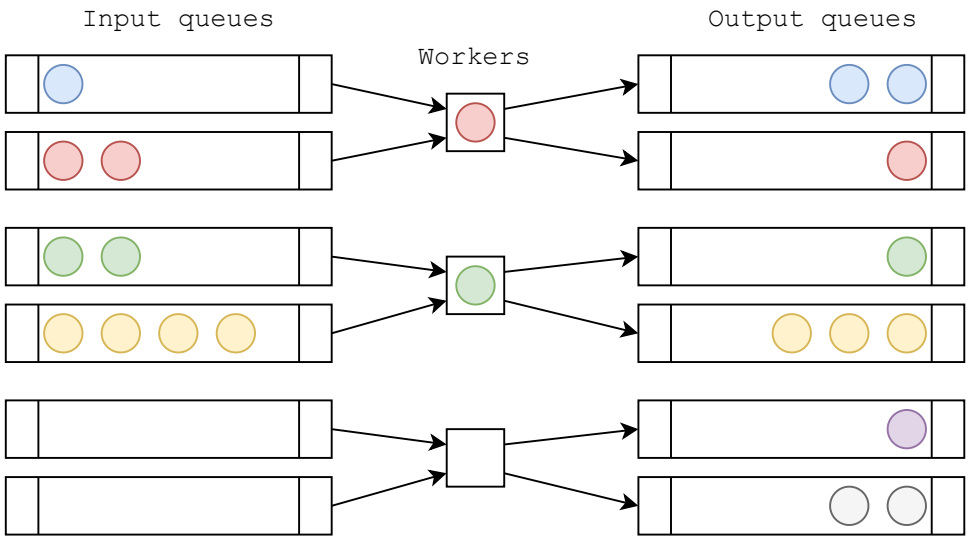

Figure 4.2: Multi-queue design

A weakness with this multi-queue approach is that if the rate of measurements are imbalanced between signals, some threads would have more values to process than others. Removing the lock convoy-effect comes at a price of potentially having an uneven amount of values to process among the work threads.

There is also the problem with the dispatcher thread not being able to fill the signal queues faster than the signal values are processed by the workers. This is because

performing a single iteration of the PASAD or AR testing phase is a relatively fast operation. This leads to worker threads simply waiting for the signal queues to fill up instead of utilising such time for processing. A way to circumvent this imbalance is to spawn multiple dispatcher threads so that they can individually fill up signal queues.

The multi-queue approach involves more mutexes and threads than the basic thread pool scheme, and hence, has much larger overhead. However, there is less coordination between threads because work is more separated. When a constant rate of signals and values need to be processed, this scheme might be beneficial to performance.

4.6 GPU-Accelerated Scheme

Graphics Processing Units (GPUs) are processors designed for computer graphics applications. They employ special pipeline design structures that allow for large blocks of computations to be processed in parallel. Besides graphics processing, some GPUs can also be used for general purpose computing. In such cases, they are referred to as General Purpose GPUs (GPGPUs). GPU hardware design makes them very suitable for stream processing — this covers datasets that can be divided into subsets, each subset having little to no need for intercommunication between processing streams, such as the linear algebra operations used by PASAD.

Several frameworks make interfacing with GPU easy. API frameworks such as DirectX and Vulcan [20] are optimised for 3D graphics routines. OpenCL provides vendor-independent APIs for implementing GPGPU applications [19]. CUDA is another GPGPU framework for NVIDIA GPUs [35].

To create a GPU-native implementation of PASAD and AR using one of the above frameworks, it would need the whole algorithm to be implemented as a native GPU kernel, including all the linear algebra and other necessary routines. An alternative is to use a CPU-bound implementation of the algorithm done in high-level languages such as C++, but are linked to a GPU BLAS library. Notable GPU-offloaded BLAS libraries are clBLAS and cuBLAS, which use OpenCL and CUDA respectively [13, 34].

In this study, we create a native OpenCL kernel for the testing phase of PASAD. The implementation is not intended to be performance optimal or even a reference implementation; instead it is only meant to serve as a proof-of-concept for future work. Because the training phase involves multiple mathematical operations which would be hard to implement in OpenCL, such as calculating Single Value Decomposition (SVD), we have opted to only perform the testing phase. Moreover, training is rarely time critical and happens only once, thus having a GPU implementation for training would likely not yield much benefit to system operators.

PASAD has been chosen over AR modelling for the GPU implementation due to its higher potential for parallelisation. The negative aspects of going with PASAD

is that more data is needed to be transferred to the GPU. Furthermore, the implemented operations are more complex, which means that there is a higher likelihood for creating bugs in the implementation. The testing phase for AR modelling involves a single matrix multiplication of two small matrices, while for PASAD there are multiple matrix operations and calculations. Thus, the overhead of transferring data to and from the GPU has a relatively lower impact on overall performance for PASAD than it has for AR.

We also try to further reduce overhead by buffering the incoming signal, so that multiple departure scores can be calculated in parallel. By increasing the number of values buffered, throughput improves at the expense of latency. As such, the importance of latency contra throughput would have to be determined on a case by case basis during deployment.

4.7 Visualising Output

AR modelling and PASAD produce departure scores based on the time series of input data. A visualisation tool has been designed in order to get an intuitive sense of how signal input and algorithm parameters affect departure score. The visualiser consists of a stand-alone graphing tool that is able to plot signal values and their corresponding departure scores in real-time. To make the visualiser as modular as possible, sockets have been used for Inter-Process Communication (IPC), allowing it to be put on a completely different system if needed. However, all performance tests in this report have been run without the visualiser so as to avoid interference with observations.

4.8 Integration with PCU400

In order to evaluate the feasibility of integrating the anomaly detection algorithms in a large-scale ICS environment, we integrate them into the PCU400 system. The intention is to verify that the anomaly detection algorithms do not interfere with existing control structures, and that appropriate signal values can be extracted with reasonable effort.

As covered in Subsection 2.6.3, the DCU is the *brain* of PCU400. It is responsible for translating RTU-specific protocols into RSP which is understood by ABB systems, and RSP commands into protocol specific commands for the RTUs. Therefore, the ideal location for the anomaly detection module is to have it in the PCU400, just at the step occurring after translation to RSP and before sending the information to the SCADA system. This way, the anomaly detection module has a consistent way to extract signal values and allows anomaly scores to be included with signal information sent to SCADA systems. The flow is made clearer in Figure 4.3.

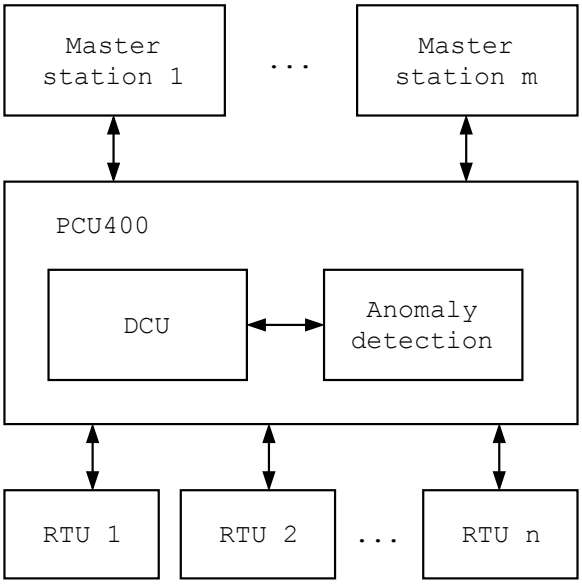

Figure 4.3: Integrating anomaly detection module in PCU400

5

Implementation

This chapter explains how the software for evaluating process-level anomaly detection has been constructed and lists the languages, tools, and libraries that have been used in its implementation. We start with reference implementations for the anomaly detection algorithms in Section 5.1, and move on to core implementations of PASAD and AR modelling modules in Section 5.2. Section 5.3 talks about how multi-signal processing has been implemented. Section 5.4 provides implementation details for the GPU-accelerated scheme. The development of the visualisation tool for viewing signal input and departure score is presented in Section 5.5. A brief overview of PCU400 codebase and development setup is presented in Section 5.6. Integration of these modules with ABB X97 simulator is discussed in Subsection 5.6.1.

In this report, anomaly detection algorithms are referred to by their acronyms — PASAD — and the implemented modules are indicated in monospace typeface — `Pasad`.

5.1 Reference Implementation

For both PASAD and AR modelling, reference implementations have been implemented within Jupyter Notebook using Python 3.6. Jupyter Notebook provides a web-based mathematical computation environment [28]. It allows program instructions to be written in a series of input/output cells along with rich text, LaTeX equations, plots and images. Python is a popular high-level interpreted language that emphasises code readability and is extensively used in scientific computing [38]. Since the Python standard library lacks support for advanced mathematics, other open-source libraries have been used. Scipy provides support for linear algebra [27]; Numpy allows use of large arrays and multi-dimensional matrices [50]; and Matplotlib is used for generating plots to visualise the inputs and results [24].

5.2 Core Implementation

Both PASAD and AR have been implemented as C++ classes called `Pasad` and `Ar` respectively. Each signal is associated with its own instantiation of these classes and is only trained for that signal. The classes provide of a number of helper methods,

such as for constructing the signal subspace `train(...)`, or to feed new values to the model `receive_value(...)`. Parameters such as the number of values to use for training, the size of the lagged vector space, and the size of the subspace to use, have to be specified in the constructor of the object, meaning they can not be changed after instantiation. Both classes train the model for specified number of values and switch to testing when the number of received values exceed the training size. All distance measurements are stored within the model, but is something which can be moved to persistent storage if required.

For PASAD, in order to process the `lagged_size`th value received, a vector of the previous `0, ..., lagged_size-1` values is stored for each signal. When a new value is received, the vector is shifted one step to the right and the new value inserted, giving a lagged vector of values with index `1, ..., lagged_size-1, lagged_size`. Because this is a stateful operation, it is not possible to have multiple threads processing the same signal at a time.

For AR, we have leveraged an open source implementation for the training phase [39], and have implemented the testing phase ourselves. It exposes the same API function names as `Pasad`, thus making it easy to switch classes if needed. Internally, the training and testing switching also works in a similar manner.

A chief dependency of the implementation is the Armadillo library [44]. It provides native capability for some basic linear algebra operations such as matrix addition and multiplication. For more advanced operations such as eigen decomposition, Armadillo wraps around a BLAS library.

5.2.1 Basic Linear Algebra Subprograms Libraries

Basic Linear Algebra Subprograms (BLAS) is a specification for low-level linear algebra routines such as vector-scalar multiplication and matrix multiplication. The specification is accompanied by a Fortan implementation, which is the reference implementation [40]. Colloquially, this implementation is referred to as *the* BLAS library. A related library is the Linear Algebra Package (LAPACK) library, which provides higher-level linear algebra operation, such as matrix factorisation and eigenvalue solvers [43]. Both BLAS and LAPACK specification and libraries are developed and maintained by the Netlib project [33].

There are many other BLAS implementations that seek to improve performance using alternative approaches. In this study, we focus on the reference BLAS library, OpenBLAS, and Intel MKL. OpenBLAS is a performance-optimised fork of GotoBLAS [52, 53, 54], while Math Kernel Library (MKL) is a proprietary BLAS implementation for Intel processors offering hardware-optimised performance [26].

5.3 Processing Multiple Signals

For time series algorithms, there is a strict requirement that values are processed in the order they are received. Even though a queue is used in the core implementation, there is no guarantee that the values will be processed in order. This is because of the context switches performed by the scheduler. A worker may be preempted mid-execution, causing another worker to process a signal value out of order. To prevent this, synchronisation primitives, such as mutex locks, can be used. A mutex lock offers a synchronisation mechanism in multi-threaded environments by preventing concurrent access to critical sections of program code. We have opted for using a fair mutex to ensure threads gets woken up in the correct order for processing.

The thread pooled design covered in Section 2.3 spawns a set number of threads to process the incoming signals. These threads are woken up one-by-one when a new value is enqueued on the work queue. A hashmap, pairing signal identifiers to instantiated objects, is used to keep track of which `Pasad` or `Ar` object is associated with incoming signals. The main thread instantiates a new object if it is missing for a given signal, and runs the `receive_value(...)` method with the received value to be processed. A fair mutex is also associated with each instantiated object, guaranteeing that only one worker can process the signal at any time.

For execution control and safety, C++ Thread Pool Library and Yet Another Mutex Collection provide thread pools and fair mutexes respectively [51, 55]. These libraries use Boost for its foundation, notably for lock-free queues [41].

5.4 GPU-Accelerated Scheme

We have experimented with creating a GPGPU implementation of the PASAD testing phase. In order to decrease complexity, we have focused only on a specific setup of PASAD where the training size is 200, the lagged vector size 100, and the subspace size is 5. This assumption allows us to maximise the use of worker threads for matrix multiplications.

To synchronise read and write operations to the various memory hierarchies of OpenCL, *barriers* have been used. Barriers work by specifying that all threads executing the barrier will wait until all other threads accessing either global or local memory are finished with write and read operations.

The parallelised scheme stores the incoming signal in global memory due to its large size. However, it might be sensible to copy over the current signal index into local memory for improved access time, based on the hardware capabilities, lagged vector size and subspace size.

The basic formula for calculating a departure score D_i for signal value $\mathbf{x_i}$ is:

$$\mathbf{x_i} = (x_{i-L+1}, x_{i-L+2}, ..., x_i) \tag{5.1}$$
$$\tilde{\mathbf{c}} = \mathbf{U^T} \cdot \mathbf{c} \tag{5.2}$$
$$D_i = ||\tilde{\mathbf{c}} - \mathbf{U^T x_i}||^2 \tag{5.3}$$

where L is the lagged vector size, $\mathbf{c}$ is the training centroid, and $\mathbf{U}$ is the signal subspace. Since the projected centroid is a constant, an expanded version is used instead to reduce the number of operations needed to calculate the departure score:

$$\mathbf{p} = \mathbf{U^T} \cdot \mathbf{x_i} \tag{5.4}$$
$$D_i = ||\tilde{\mathbf{c}} - \mathbf{U^T} \cdot \mathbf{x_i}||^2 \tag{5.5}$$
$$= ||\tilde{\mathbf{c}}||^2 - 2 \cdot \tilde{\mathbf{c}} \cdot \mathbf{p} + ||\mathbf{p}||^2 \tag{5.6}$$

Using Equation 5.6, $||\tilde{\mathbf{c}}||^2$ is instead computed only once in the training phase.

For the GPGPU implementation, multiple work-groups are used to process different indices of the incoming signal, each group processing 100 (i.e. the fixed lagged vector size) departure scores. The parallelisation begins by having 100 threads in each work-group multiply the value at its thread index of the first row in $\mathbf{U^T}$ with its thread index in the current lagged signal $\mathbf{x_i}$, as shown in Figure 5.1. At this point, a barrier ensures that the writes of all workers are synchronised.

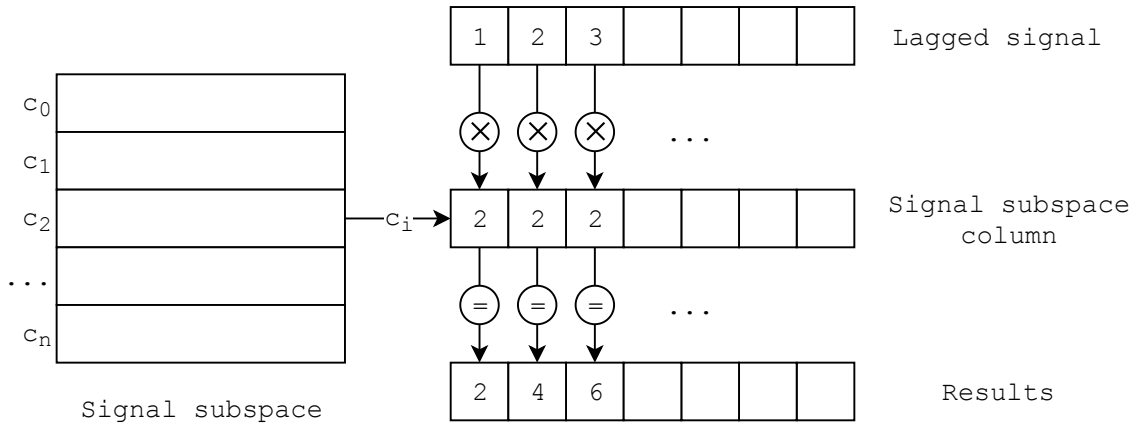

Figure 5.1: Signal subspace projection for PASAD OpenCL.

A parallel summing scheme is then used, where the threads with a thread index less than 50 (half of the lagged vector size) sum the value at their thread index with the value at index 50 positions higher. This is followed by a barrier and is repeated, each time halving the offset and number of threads used in the sum until there is only a single thread left, as shown in Figure 5.2. The resulting sum is then located in the first index position. This process is looped for each row of $\mathbf{U^T}$ until $\mathbf{p}$ has been calculated.

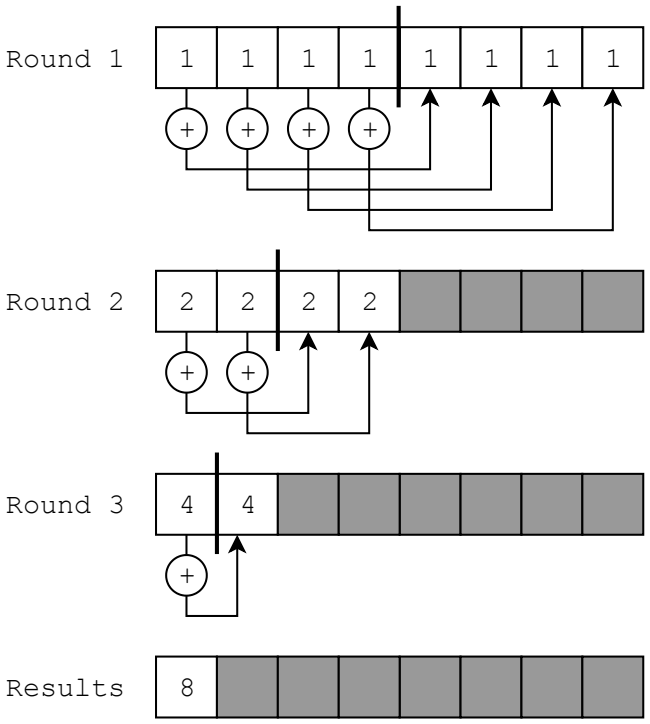

Figure 5.2: Parallel summing of array with example values.

$||\mathbf{p}||^2$ is then computed by having each thread index less than the subspace size squaring the value at its index, followed by a barrier and a parallel sum. In a similar fashion, $2 \cdot \tilde{\mathbf{c}} \cdot \mathbf{p}$ is computed by having each thread multiply the value at its index in the projected centroid with the same index in the lagged signal. A parallel sum and a barrier is applied afterwards to get the resulting value. Lastly, the departure score is computed and stored in a shared buffer by the first thread in the work-group. This entire process repeats 100 times (i.e. the lagged vector size) for each work-group until the buffered signal is finished processing.

5.5 Visualising Output

Software modules work best when directly integrated into the target system. This holds for the anomaly detection modules as well, which would ideally be part of the core ICS software. A consequence of this would be that any way to visualise the inputs and other information of interest will have also be provided by the target system. In order to provide an implementation independent way to visualise the data, namely the input signal values and computed departure scores, a stand-alone program called `liveplot.py` has been implemented. This has been done in Python 3.6 and the Matplotlib plotting library. A C/C++ API has also been written to allow integration with the PASAD class. The API provides straightforward means to send signal values and departure scores to `liveplot.py`.

`liveplot.py` is capable of real-time plotting over socket IPC. On Linux, Unix domain sockets `AF_UNIX` are used. On Windows, `AF_INET` sockets are used, because the lack of POSIX compatibility. Sockets support both stateful TCP and stateless UDP connections. However, using TCP connections is a poor choice because communication is local and hence, reliable. The extra overhead of establishing and maintaining the connection is wasteful given that local connections are lossless. Moreover, plotting is a secondary function; ensuring that departure score calculation in near real-time is of higher priority. It is acceptable if some data points are lost due to failure to obtain CPU time. Hence, on both platforms, UDP connections have been used.

5.6 Integration with PCU400

For the purpose of this study, ABB provided access to the source code for X05 and X97, as well as binary and license files for PCU400. The target platform for PCU400 is Windows and the primary integrated development environment used by the ABB development team is Visual Studio 2013. The provided codebase consisted of following Visual Studio projects:

- `domLite:` support functions for DOM and HTML used by web control panel
- `toolkitLib:` foundation and base classes for all XLDs
- `x05:` source code for X05
- `x97:` source code for X97

PCU400 requires a license file in order to operate with full functionality. In absence of a valid license file, PCU400 operates in a time-limited demo mode. The license file contains permission information about the communication protocols that can be used, number of communication lines, addresses, and information about the computer systems it is allowed to run on. For the latter, the license file is linked to a specific computer's MAC and public IP address. For this thesis, two static IP addresses were requested from Chalmers IT department. ABB supplied the license files associated with these static IP addresses.

5.6.1 Integration with X97

In PCU400, the X97 module is capable of simulating RTUs in scenarios such as testing, when real RTUs may not be available. It works by generating mock signal values and feeding them to the DCU. The specifics of these signals such as the protocols they use, data types, addresses, physical lines, etc. are configured using the Excel Data Engineering Tool. Select supported data types of interest include:

- `Ind:` indications, which work like booleans
- `Amv:` analogue measured values

- `Dmv`: digital measured values
- `Cmd`: object commands

The exact properties of the signals can not be set through the Excel Data Engineering Tool. As such, it is difficult to finely control the attributes of the signals, especially the necessity to introduce anomalies. This could have been added to X97 signal generation logic programmatically, but it would still be difficult to change the properties if required without updating the code and recompiling the project. For these reasons, the functionality to read the signal values from a CSV file has been implemented for X97. This way, signal values with required anomalous behaviour could be fed into X97 in a more straightforward manner.

Figure 5.3 shows the proof-of-concept modifications that have been made to X97 to integrate PASAD. The CSV parser has been implemented as a C++ library and merged into the `x97` project. `Amv`s correspond to floating point values that PASAD and AR take as input, and hence have been used for tests and evaluations. Only `Pasad` has been integrated into PCU400, but `Ar` has the same API and it is straightforward to switch the anomaly detection method if needed.

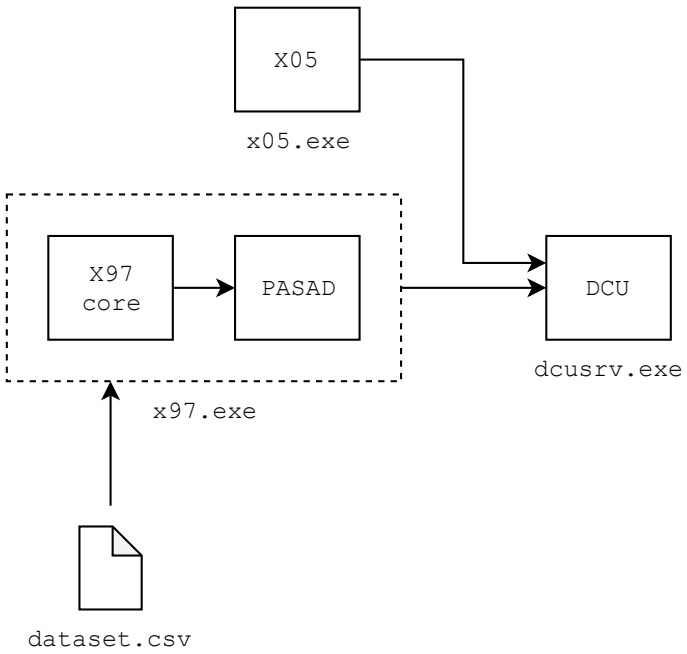

Figure 5.3: Integrating PASAD in PCU400.

In PCU400, the configuration management is handled by X05. It reads various configuration files for various drivers and serves as a configuration store when these modules are running. The DCU provides APIs and routines for reading configuration parameters. These calls have been used to obtain parameters such as path to the CSV file, training size, lagged space size, and subspace size.

As explained in Section 5.2.1, BLAS and LAPACK provide the foundation for performing matrix operations, a core part of PASAD and AR. The Armadillo project provides a pre-compiled set of static-linked `lib` libraries, dynamic-linked `dll` li-

braries, and headers only for 64-bit version of Windows. However, PCU400 is 32-bit software. We have built the 32-bit version of BLAS and LAPACK static- and dynamic-linked libraries using the Minimalist GNU for Windows (MinGW) [32]. MinGW is an open-source development toolkit which packages GNU Compiler Collection (GCC), Binutils, with several Windows-specific modifications and header files. Consequently, this means that after integration, DCU requires additional files in its working directory, namely, `liblapack.dll`, `libblas.dll`, `libgfortran-5.dll` and `libgcc_s_sjlj-1.dll`.

The Armadillo project recommends using Linux for high-performance workloads. The limited scope of this thesis meant that PCU400 could not be ported and tested on Linux. In Chapter 6, we test the thread pool module on Linux and compare the performance with different BLAS libraries.

6

Evaluation

This chapter covers the evaluation of anomaly detection modules developed as part of this study. The evaluation has been done with the purpose of determining the suitability of these modules for production ICSs. Special focus has been put on determining if process-level anomaly detection is able to manage a large number of signal values that production ICSs generate.

We begin with a description of the testbed in Section 6.1. Information about the test data used for the evaluation is presented in Section 6.2. There are different BLAS libraries that offer varying performance; a practical comparison of them is covered in Section 6.4. Section 6.5 verifies that the implementation makes full use of available CPUs to extract maximum performance. In Section 6.6, we present the behaviour of the implementation for various signal frequency distributions. Section 6.7 tests PASAD and AR with various anomalous sinusoidal signals. The chapter concludes with a performance analysis of PASAD OpenCL with various degrees of buffering in Section 6.9.

6.1 Setup

Performance of any application greatly depends on the underlying system capabilities. Outside of the program-specific optimisations, the CPU and the GPU have a large impact on speed of execution. The idea with the choice of hardware has been to be as broad in scope as possible. Processors from different vendors, Intel, AMD, ARM and NVIDIA, with very different capabilities have been used. A brief summary of each processor is presented below.

Intel Core i3-7100U is a laptop-grade CPU launched in 2016 [25]. This chipset has been designed with power efficiency in mind at the expense of performance. The 'U' in chipset name indicates that it is designed to have ultra low power consumption. This processor has two cores with two hardware threads per core. Each core runs at a clock speed of 2.4 GHz.

AMD Ryzen 7 1800X is a high-performance CPU launched in 2017 targeting the desktop market [14]. The chipset features eight cores with two hardware threads per core. Each core has a base clock rate of 3.4 GHz, which can be increased up to 4 GHz for demanding applications.

ARM Mali T628 is a low-power GPU released by ARM Holdings in 2012 [8]. The target for this GPU is mobile applications such as smartphones. In this study, we use this GPU onboard the ODROID-XU4 single board computer for developing and testing the OpenCL implementation of PASAD [22]. Mali T628 has a core clock rate of 600 MHz and includes support for OpenCL 1.2. A notable design choice in case of ODROID-XU4 is that the GPU lacks independent memory and shares the host memory. This can be used to achieve performance benefits when using OpenCL; data can be passed to GPU as memory pointers instead of copying the buffers back and forth.

NVIDIA GeForce GTX 1080 is a top-end desktop graphics card by NVIDIA Corporation [36]. This GPU features 2,560 cores with a base clock rate of 1,607 MHz and comes with 8 GB of graphics memory.

All tests have been performed on Ubuntu 18.04.2 LTS 64-bit, running Linux 4.18.0.

6.2 Datasets and Correctness

In some of the benchmarking experiments that follow, we have used the attack datasets developed for the Tennessee-Eastman process [7, 16]. In the Tennessee-Eastman process, the readings of all measured sensors are termed as XMEAS, and all actuator commands as XMV. XMEAS consist of 41 signals, while XMV consist of 12 manipulated signals that correspond to actuator commands. As such, each dataset has 41 signals, and 4,800 values per signal, totalling 196,800 values per dataset.

In addition, mock datasets have been created to perform some of the visual analysis, signal distribution, and throughput tests. This data has been constructed with the help of Python `numpy.signals` module, adding random noise and by sampling and repeating the Tennessee-Eastman datasets.

In order to ensure correctness of the C++ implementation of PASAD and AR, the departure scores and results have been cross-checked with those from the Python reference implementation.

6.3 Scalability

Production-grade ICSs process a large number of signals and signal values. As such, it is important to determine how well PASAD and AR modelling is able to the fulfil the processing requirements of these systems. We have verified this by measuring the variation of throughput depending on the number of signals being processed. Figure 6.1 displays throughput as a function of number of signals for PASAD and AR. For these tests, only the time to process the values are included in the measurements. The enqueueing gets finished before processing begins.

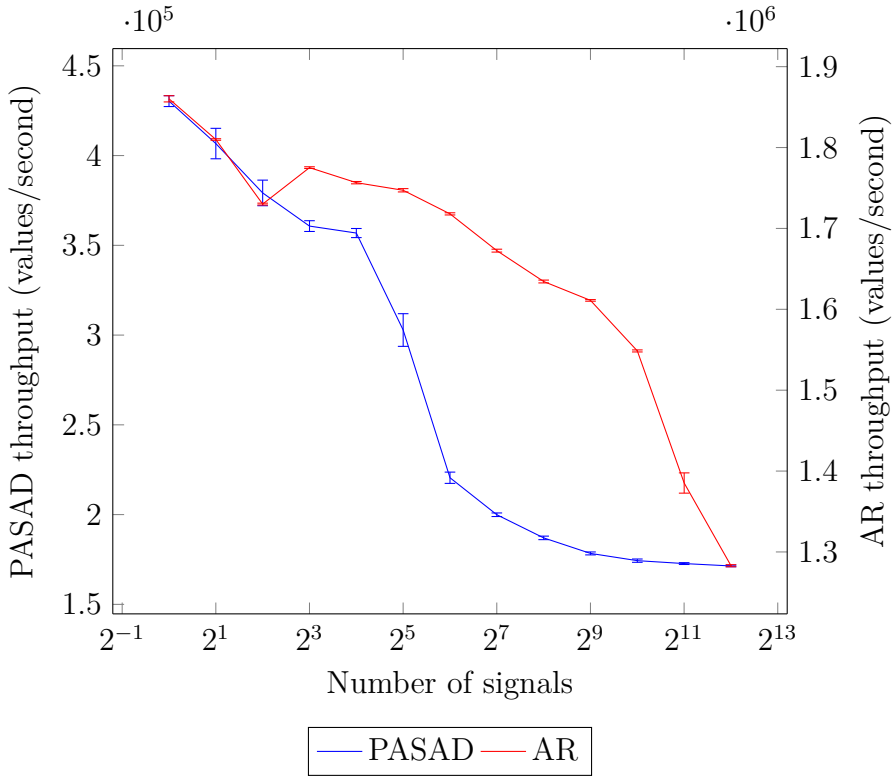

Figure 6.1: Mean PASAD and AR throughput versus number of signals for five runs ($N = 500$, $L = 250$, $r = 30$, Intel i3, two workers for PASAD, one worker for AR). The error bars represent standard deviation.

The performance curve of PASAD experiences a decline of 60.17% when increasing the number of signals from 2 to 4,096. There is a sharp decline when the number of signals are increased from 16 to 64, experiencing a drop of 38.20% throughput. For AR, the decline is not as steep and only 31.03% over the entire range. We have found that trying to model 8,192 PASAD models depleted more memory than was available on the test hardware. While the program is able to train all the PASAD models, it terminates when enqueueing the values. This is likely due to the overhead of both having the CSV in memory and the values on the queue.

For 8,192 signals, we have measured a usage of 4.4 GB of memory for training all PASAD models. Extrapolating this, a total of around 35 GB of memory would be needed to model 65,000 signals.

If the decline in performance follows around 8% for PASAD each time the number of signals is doubled, then processing 65,000 signals would have a throughput of 113,000 signal values per second. For AR, if the decline averages around 4%, processing 65,000 signals would result in a throughput of around 968,000 signal values per second.

6.4 Basic Linear Algebra Subprograms Libraries

The throughput of the thread pool module has been tested in various conditions: on CPUs of different types and number of cores, with different number of workers, and when linked to different BLAS libraries. The difference between each of the libraries have been discussed previously in Section 5.2.1. For these tests, the time taken by training, enqueueing, and processing has been included when calculating the processing rate.

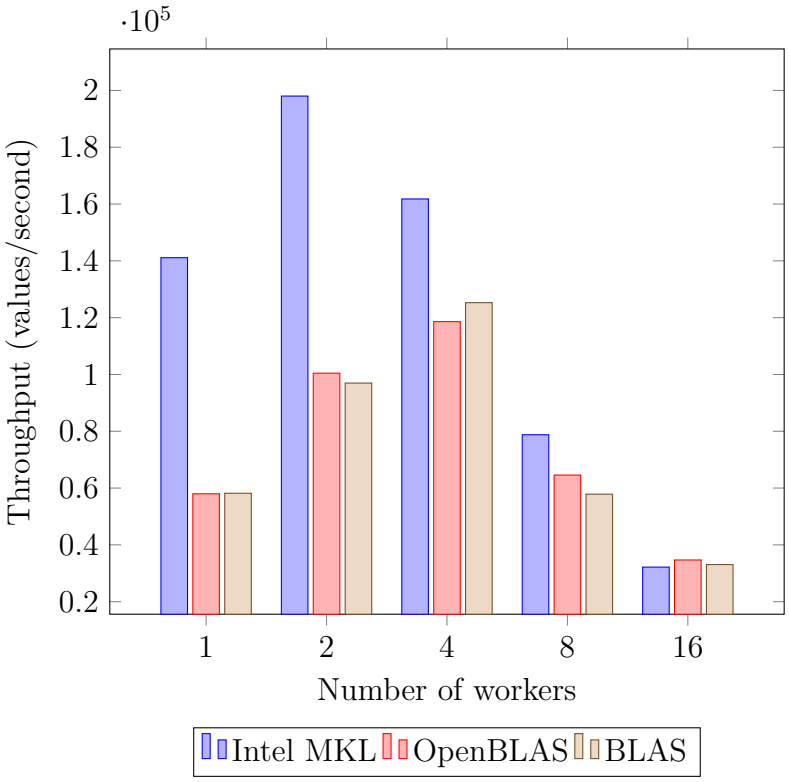

Figure 6.2: Mean PASAD throughput for different BLAS implementations for ten runs ($N = 500$, $L = 250$, $r = 30$, AMD Ryzen 7).

PASAD: As can be seen in Table A.1, PASAD performs best when using Intel MKL because of the hardware-level optimisations. With MKL, the standard deviation of all runs is also the lowest when the number of workers is equal to the number of CPU cores. Irrespective of core count, the highest throughput is obtained when two workers are being used.

The highest performance when using OpenBLAS highly depends on worker-CPU combination. For a four-core CPU, two workers give the best performance, although the difference is not large when compared to four workers. On an eight-core CPU, eight workers give maximum throughput.

BLAS has higher throughput on both four-core and eight-core CPUs when using four workers.

AR: In case of AR, as seen in Table A.2, there is little difference between libraries

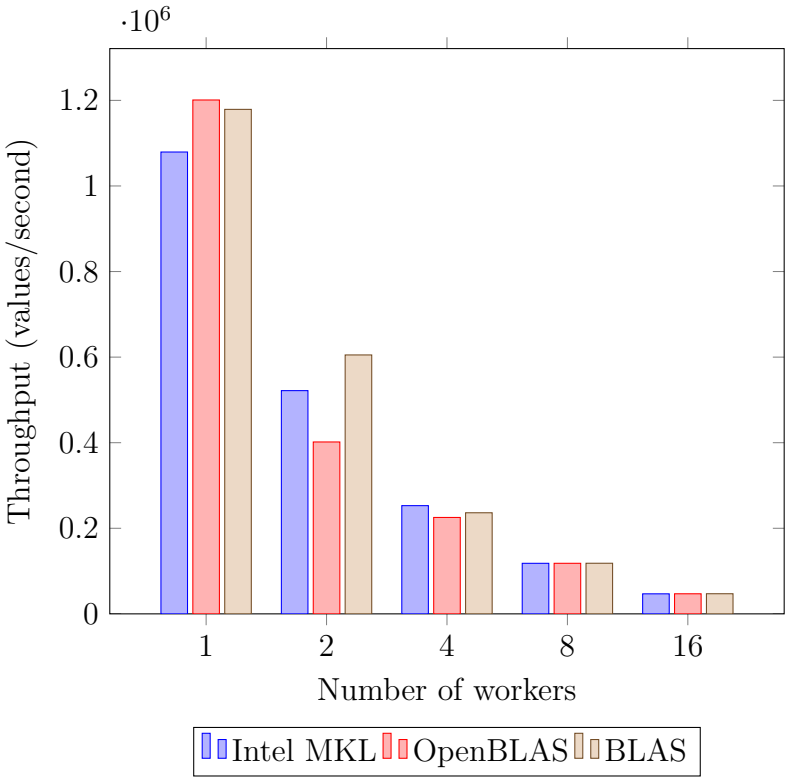

Figure 6.3: Mean AR throughput for different BLAS implementations for ten runs ($N = 500$, AMD Ryzen 7).

in relation to the number of workers. On a four-core CPU, two workers give the best performance. Even between libraries, there is little difference in throughput rates. On the eight-core CPU, using a single worker gave the highest throughput.

Figures 6.2 and 6.3 show mean throughput for the different libraries and number of workers for PASAD and AR, respectively.

The reason why using half the number of cores for PASAD processing produces a higher throughput than using all cores for is not entirely clear. It could be due to the fact that the clock rate of each core is high enough that multi-processing does not add any benefits, rather reduces performance due to context switches. If this is the case, for scenarios where the time it takes for PASAD to calculate each departure score is increased, the system might benefit from adding more threads. Normally, this would be the case when using a larger model (increased training and subspace size).

The discrepancy between throughput versus the number of workers is even more pronounced in AR than it is for PASAD. Using just a single worker yields the highest performance. It should be noted that AR inherently requires fewer linear algebra operations, which means that most of the time is spent CPU-bound.

6.5 Resource Utilisation

The core focus of the design for the thread pool module has been to extract highest possible performance by maximising CPU utilisation. Figure 6.4 shows utilisation for each CPU core when using different number of workers in various PASAD runs. These tests include the training, enqueueing, and processing steps in its measurements. While this example run has used PASAD, the CPU behaviour is caused by the thread pooling scheme; irrespective of the method of anomaly detection used.

Each worker takes up one CPU core. Workers may not necessarily be assigned to the same core throughout the execution. This can be seen in subplot Figure 6.4.a, where the single worker is initially assigned to `CPU0` but is then reassigned to `CPU2` sometime around the 14th second. The assignment of threads to CPU cores is handled by the OS scheduler and is tuned to platform-specific properties and other criteria. As such, it is generally not possible to control this assignment from user-space.

An initial peaking of other CPU cores can also be observed in Figure 6.4.a and 6.4.b. This is due to the dispatcher thread that is initially spawned to send signal values to the anomaly detection engine. This thread reads the CSV dataset and enqueues values to the thread pool queue.

6.6 Signal Frequency Distributions

The multi-processing coordination scheme presented in Section 5.3 has been developed to process multiple signals. A drawback with this scheme is that the order in which values get enqueued affects processing performance due to the lock convoy effect described in Section 4.5. We have developed tests which have varying signal enqueueing frequency. This has been done to assess how large of an effect different signal distributions have on the processing throughput. Simplified forms of Uniform, Stick, Pareto and Normal distributions have been chosen for this purpose. The choice of these distributions give a good range of possible signal distribution extremes, as illustrated in Figure 6.5.

The interleaving of different signals has been incorporated into the multi-signal processing scheme by having the values enqueued on different iterations of the enqueueing loop. This has been specified in a CSV file, which defines how many values are to be enqueued for each signal, as well as the rate of interleaving. An enqueuing rate of i denotes that a value from that signal should be enqueued every ith iteration. The signal distributions are then approximated by changing the enqueueing rate.

To ensure consistent settings between runs, each test enqueues an equal number of values for processing in the testing phase. Each distribution also limits the number of values for each signal to guarantee that the interleaving is consistent throughout the processing step. Without this guarantee, it is not certain that the results accurately reflect the system throughput. This is because enqueueing at different rates but

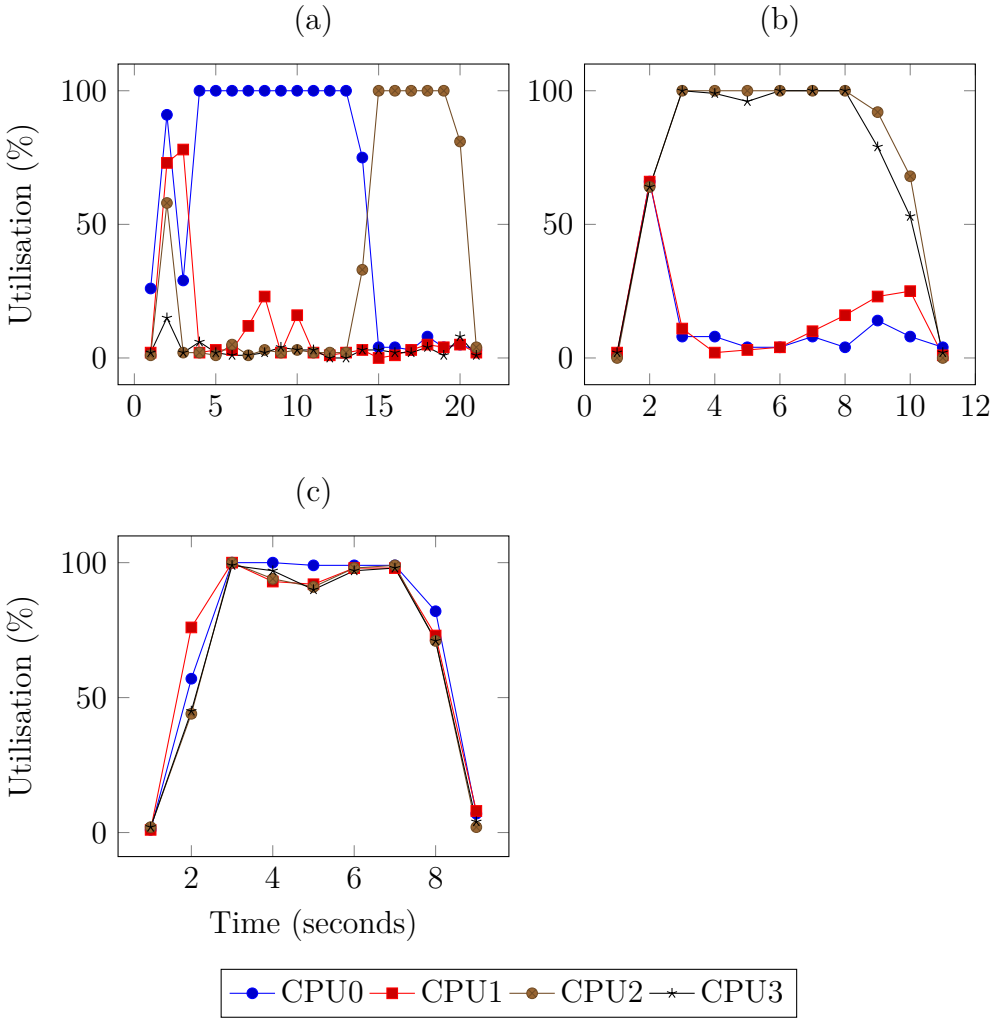

Figure 6.4: Utilisation of CPU cores for PASAD using thread-pooling with (a) single worker, (b) two workers, and (c) four workers ($N = 500$, $L = 250$, $r = 30$, Intel i3).

with an equal number of values would cause one signal to finish before the other, effectively changing the resulting distribution.

The Pareto distribution has been approximated by enqueuing the first signal on every iteration, the second signal on every second iteration, the third on every third iteration, and so on. This way, i values will have been enqueued for the first signal, $i/2$ values for the second, $i/3$ for the third, and so on.

The Normal distribution has been approximated in a similar way, but double the number of signals are enqueued for every signal except the first. When i values have been enqueued for the first signal, the second and third will have $i/2$ values enqueued each, fourth and fifth $i/3$ each, and so forth.

For the Uniform distribution, each signal has its values enqueued at the same rate, requiring no change to the enqueueing mechanism.

For the Stick distribution, each signal, except the first, has its values enqueued at

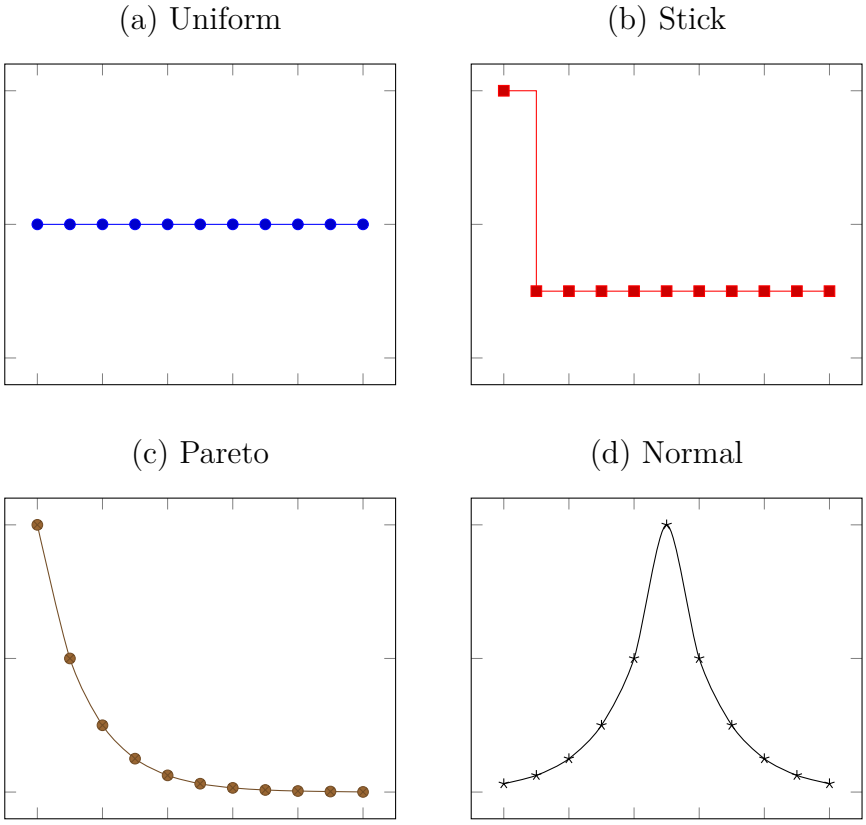

Figure 6.5: Probability density functions for (a) Uniform, (b) Stick, (c) Pareto, and (d) Normal distributions. The representations have been normalised to fit in the same range. Only their shape is of importance. In tests, these represent the frequency of arrival of signal values.

the same rate. The first signal enqueues its values four times faster than other signals; when it has enqueued i values, others will have enqueued $i/10$ each.

The following tests only measure the processing speed after all values have been enqueued.

Figure 6.6 shows, unsurprisingly, throughput is about the same for all distributions and number of signals processed since only a single thread is used. As a reference, the difference in throughput for the uniform distribution when using one signal versus 41 signals is roughly 9.6%.

Figure 6.7 shows the processing throughput for PASAD when enqueueing the signals according to Uniform, Stick, Pareto, and Normal distribution. The enqueueing distribution does not seem to have a large impact on the throughput when using PASAD. Irrespective of the distribution, the throughput trends follow a similar path, with the difference between highest and lowest measured throughput showing a drop of 36.8% across the tested range.

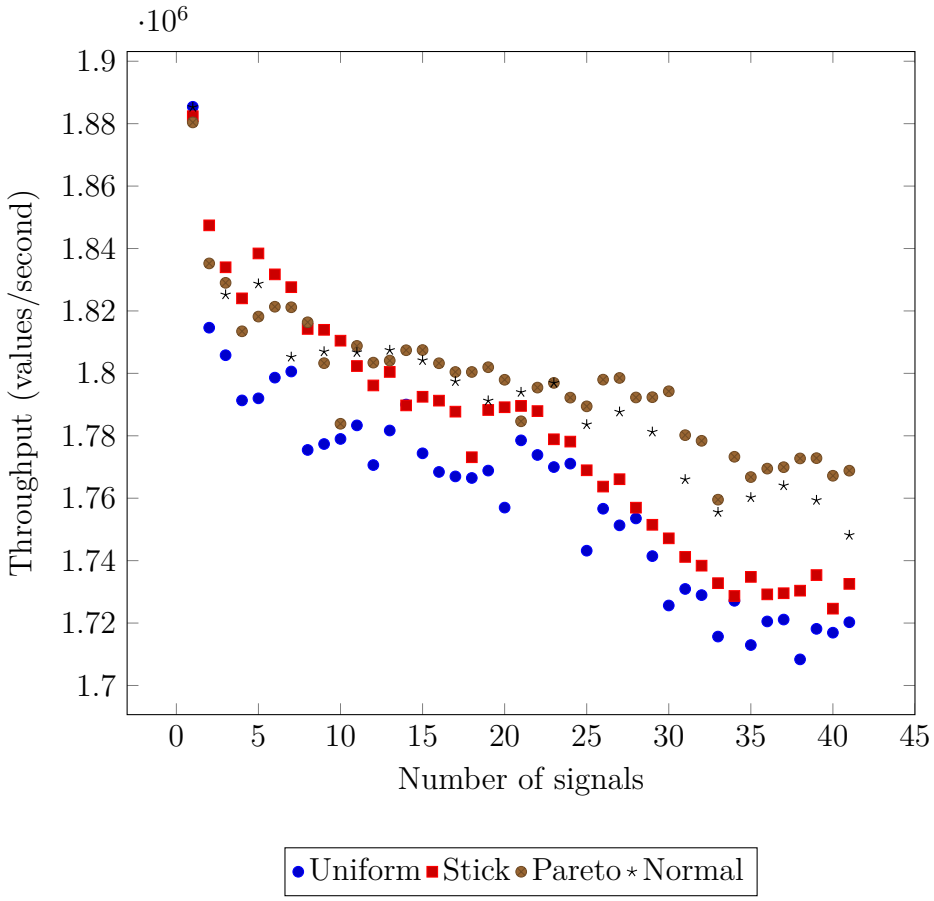

Figure 6.6: Throughput for AR per number of signals. Each test consists of processing 900,000 signal values in total (guaranteeing that a minimum of 5,000 values are processed for each signal), divided among the signals to ensure consistent load according to the respective distribution ($N = 500$, $L = 250$, $r = 30$, Intel i3, one worker).

6.7 Simulating Anomalies

Both the AR and PASAD algorithms produce departure scores which indicate that signal properties have diverged from normal behaviour. A threshold is usually set by the operator in order to decide at what point the departure score goes from within a tolerable error level to an anomaly. Our approach for determining if process-level anomaly detection is suitable for ICSs is instead based solely on reasoning about the behaviour of the departure score for different anomalous signal scenarios. For anomalies, the departure score should be clearly distinguishable from its normal.

A script for generating five different anomaly scenarios has been written. The generated periodic signals had: (a) change in polarity, (b) drift, (c) noise, (d) change in amplitude, and (e) change in frequency. For these tests, a sinusoidal wave has been used as the base, with 6,000 normal values and 1,000 anomalous values. The anomalous input signals are illustrated in Figure 6.8. The corresponding AR and PASAD scores are shown in Figure 6.9 and 6.10.

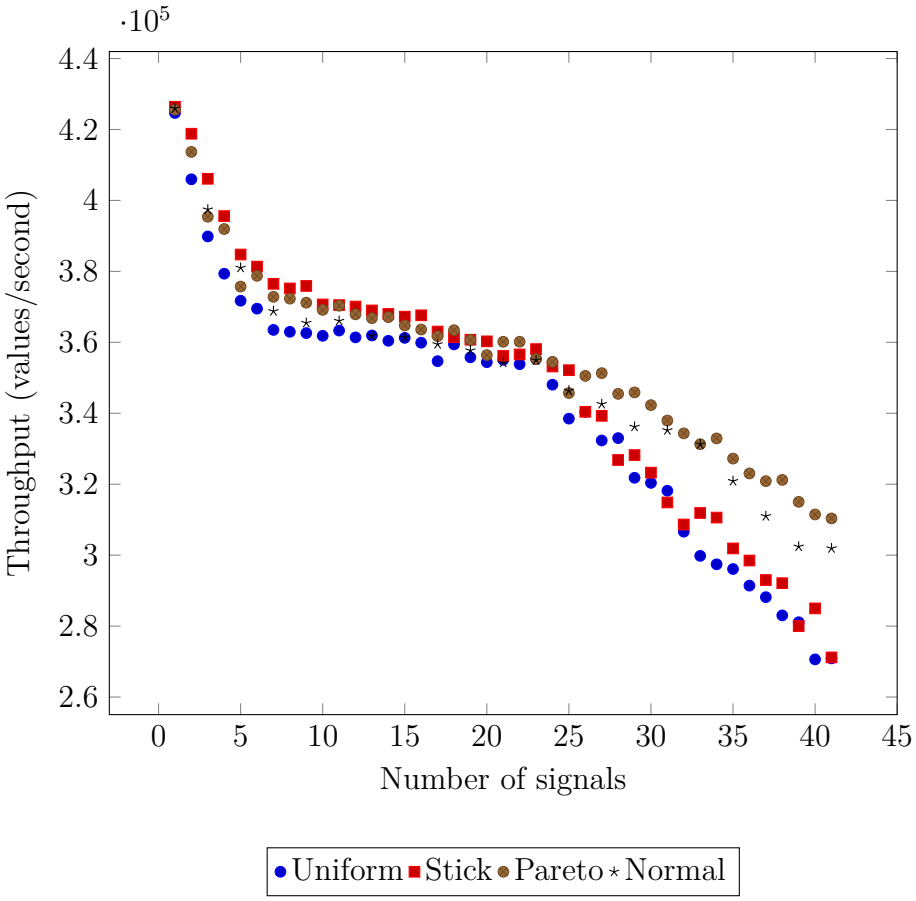

Figure 6.7: Throughput for PASAD per number of signals. Each test consists of processing 900,000 signal values in total (guaranteeing that a minimum of 5,000 values are processed for each signal), divided among the signals to ensure consistent load according to the respective distribution ($N = 500$, $L = 250$, $r = 30$, Intel i3, two workers).

A problem with these tests is that they assume the signals to be perfectly noise free. This is an idealised scenario that is highly unlikely to be the case in a real system. As such, we have performed additional tests where noise is added to the signal incrementally to observe how PASAD and AR modelling departure scores get affected.

Figure 6.11 shows this noisy version of the frequency shift signal attack with increasing levels of noise added to it. The uppermost signal have normally distributed noise added to each point with a mean of the original value and a standard deviation of 2.5% the signal range span. For the frequency shift attack the signal range is $[-1, 1]$, giving a span of 2 and standard deviation of $\sigma = 0.025$. This normal distribution implies that for a value x, the new value after adding noise $\tilde{x}$ will be have a 68.24% probability of ending up within one standard deviation from the original value, $P(x - 0.025 > \tilde{x} < x + 0.025) = 68.24\%$. It also implies that the value will have a 95.45% of ending up within two standard deviations of the original value, $P(x - 0.05 > \tilde{x} < x + 0.05) = 95.45\%$.

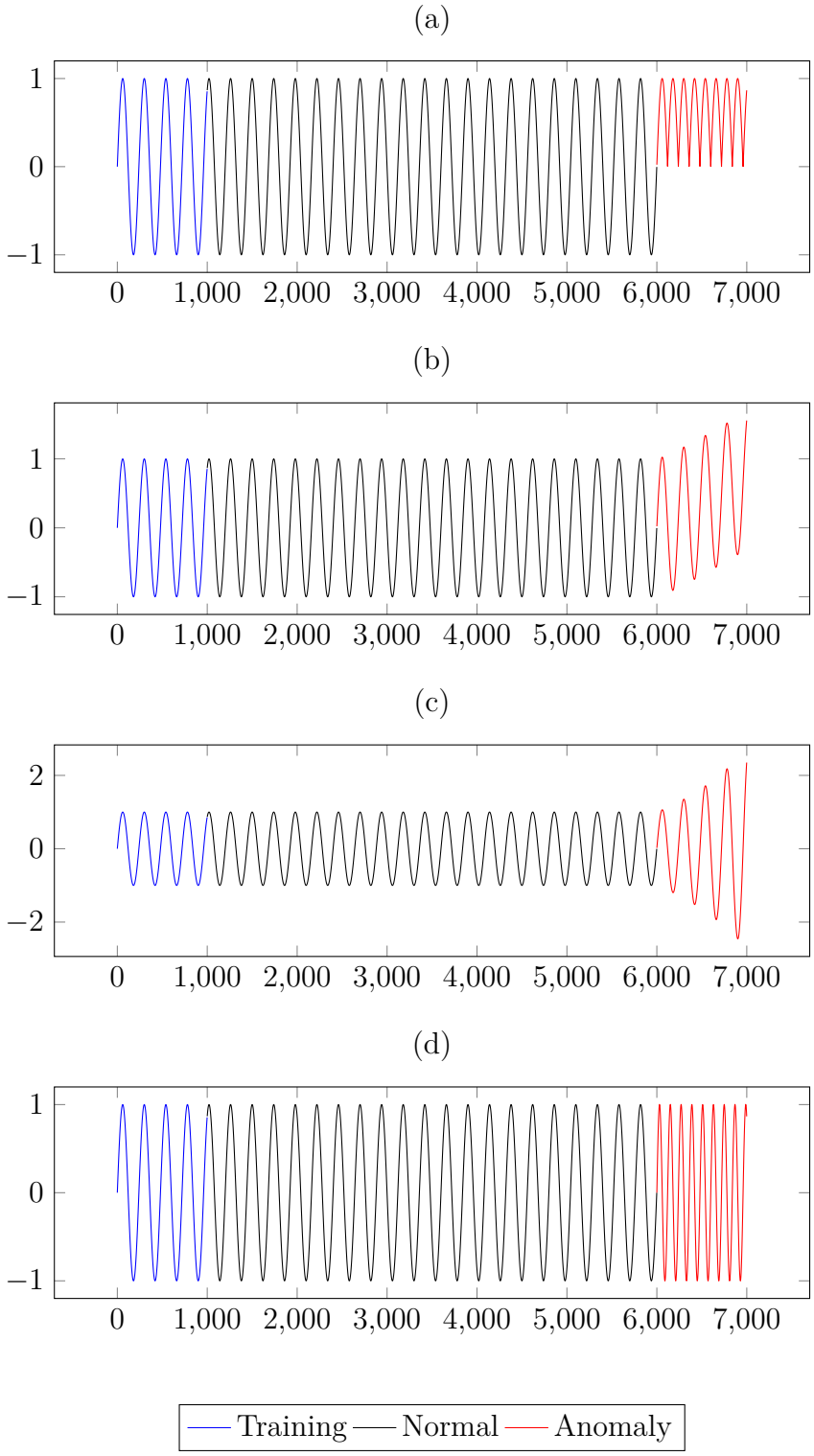

Figure 6.8: Series of simulated anomalous periodic signals exhibiting: (a) change in polarity, (b) drift, (c) change in amplitude, (d) change in frequency.

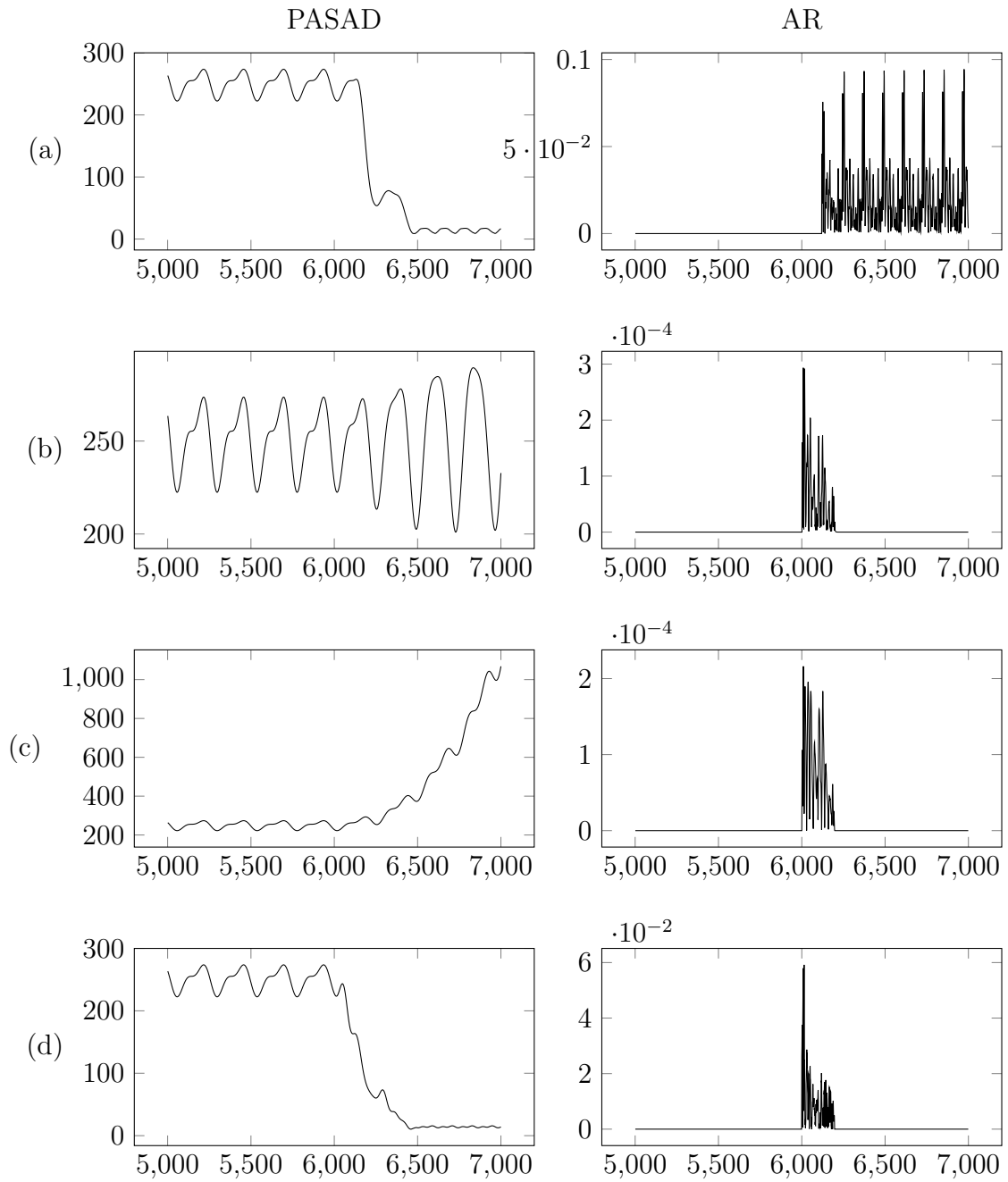

Figure 6.9: PASAD and AR departure scores for anomalous periodic signals ($N = 500$, $L = 250$, $r = 30$), exhibiting: (a) change in polarity, (b) drift, (c) change in amplitude, (d) change in frequency.

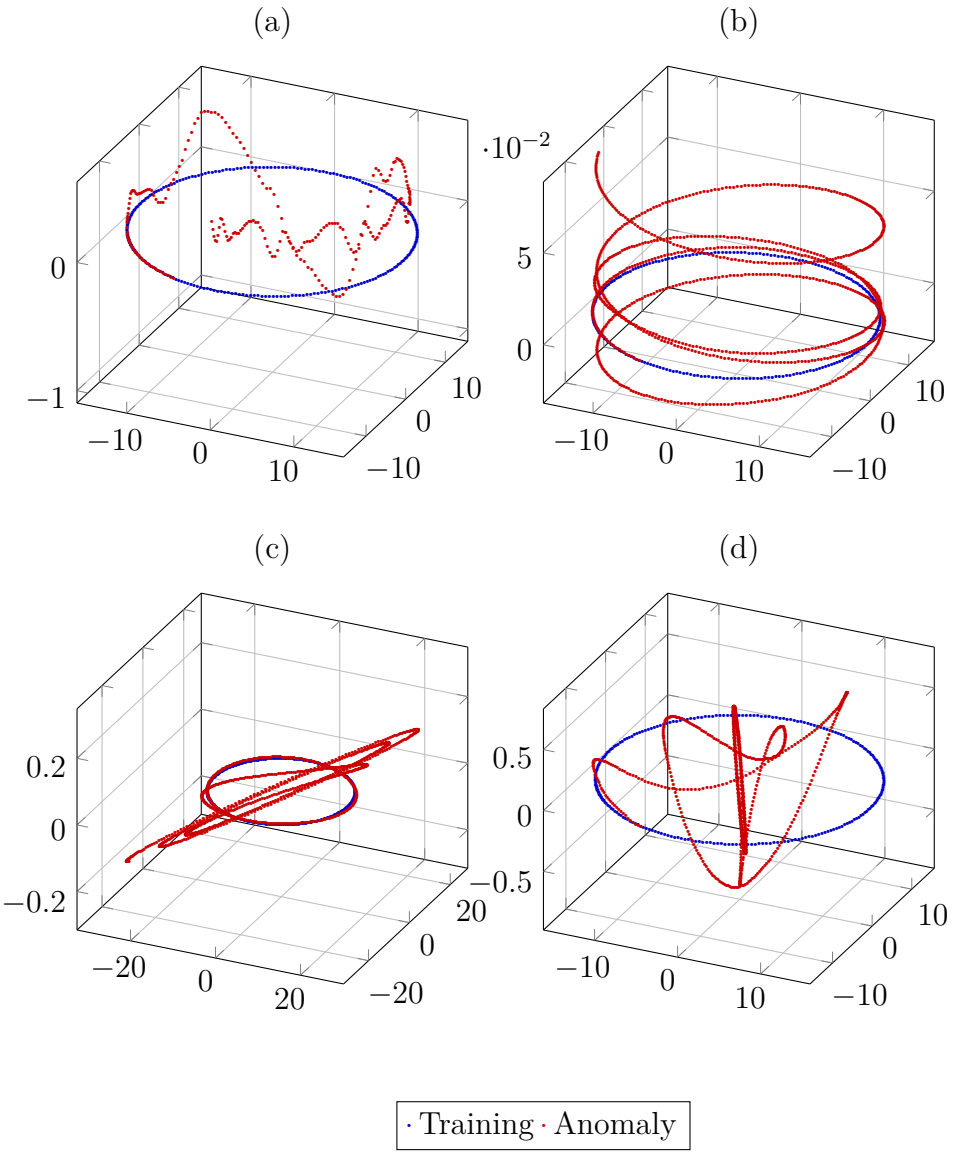

Figure 6.10: PASAD subspace projections for anomalous periodic signals ($N = 500$, $L = 250$, $r = 30$), exhibiting: (a) change in polarity, (b) drift, (c) change in amplitude, (d) change in frequency. Values with normal behaviour overlap exactly on training values and are not illustrated for clarity.

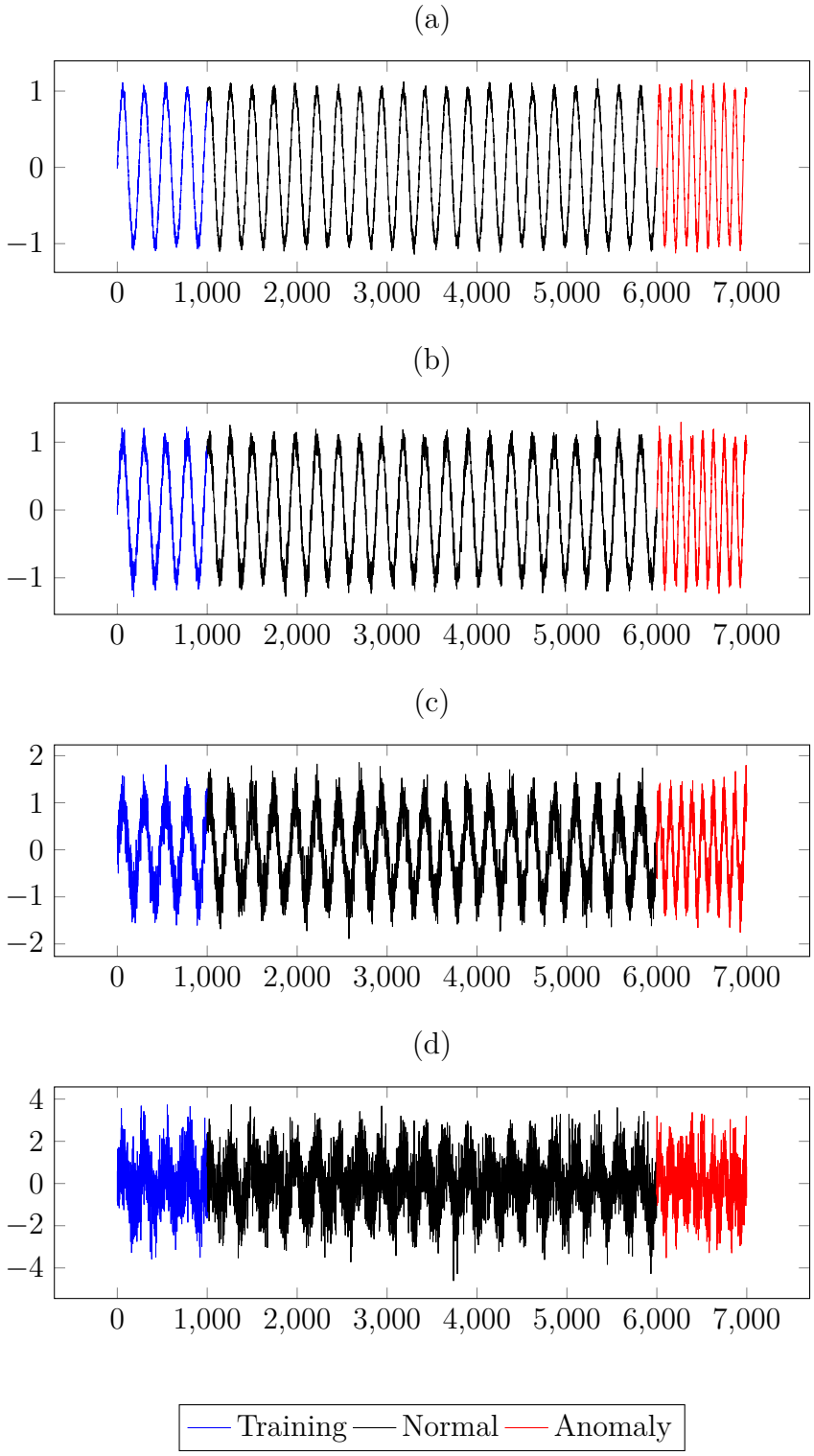

Figure 6.11: Versions of signal in Figure 6.8.d with normal variate noise relative to total signal span: (a) 5% noise, (b) 10% noise, (c) 30% noise, and (d) 100% noise.

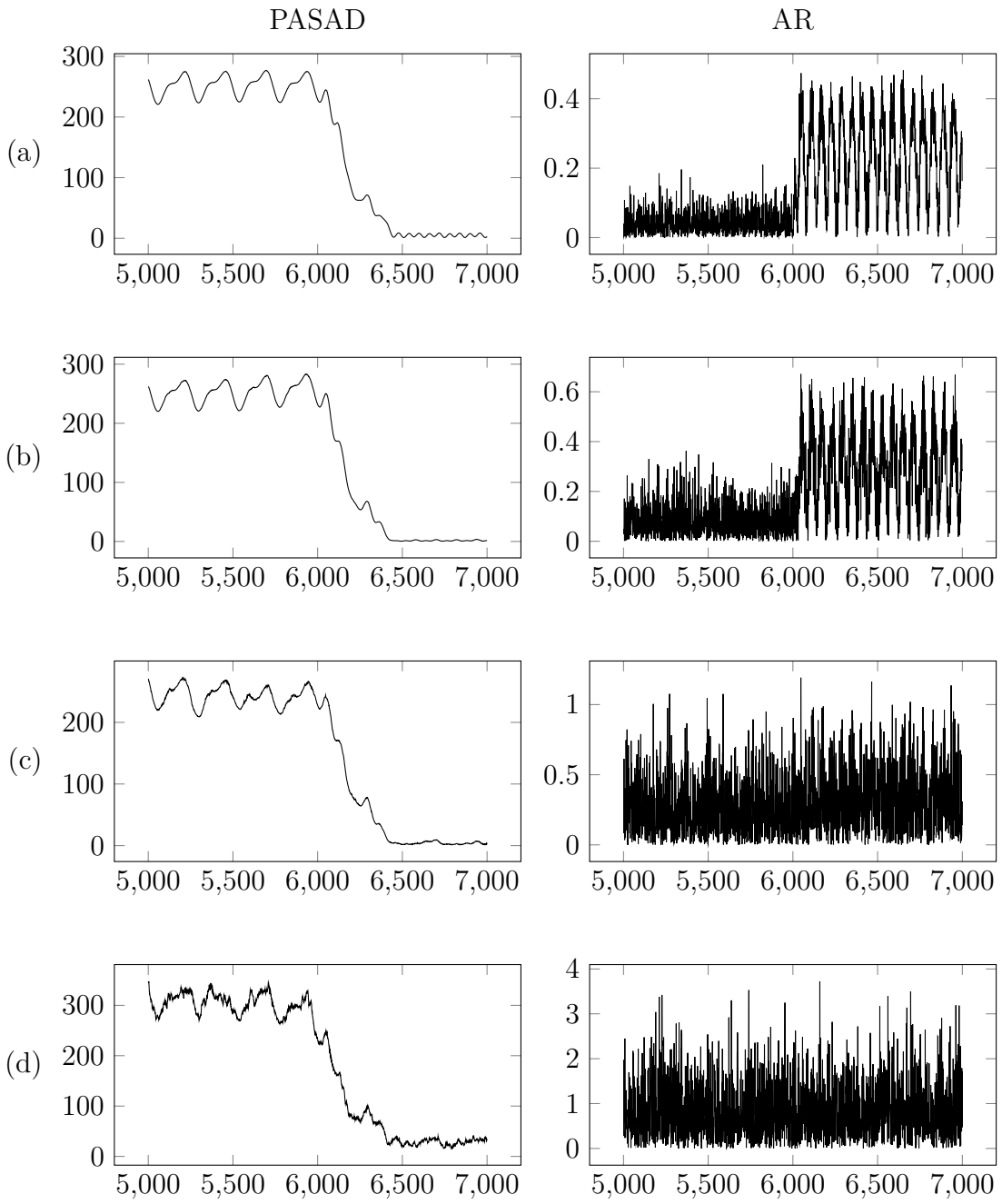

Figure 6.12: PASAD and AR departure scores for signal in Figure 6.8.d ($N = 500$, $L = 250$, $r = 30$) with normal variate noise relative to total signal span: (a) 5% noise, (b) 10% noise, (c) 30% noise, and (d) 100% noise.

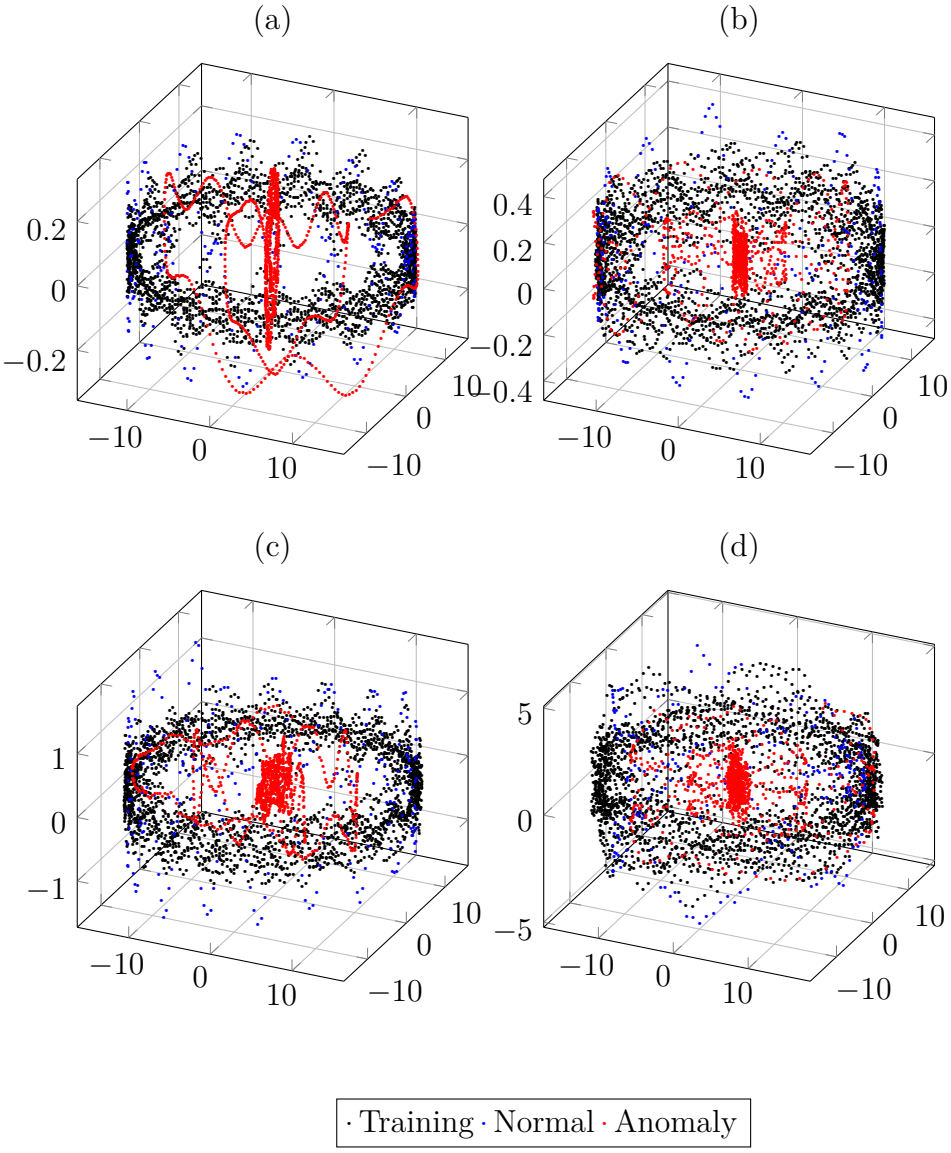

Figure 6.13: PASAD projections for signal in Figure 6.8.d ($N = 500$, $L = 250$, $r = 30$) with normal variate noise relative to total signal span: (a) 5% noise, (b) 10% noise, (c) 30% noise, and (d) 100% noise.

With this in mind, we decided on four tests where the accuracy of the sensor have a 95.45% probability of reading a value within a fraction of the signal range. The uppermost figure shows a signal with noise equivalent with reading values with a 5% error to the total signal span. This is followed by signals with 10%, 30%, and 100% signal errors.

Departure scores for AR modelling and PASAD can be seen in Figure 6.12. The resulting PASAD subspace projection can be seen in Figure 6.13. Here the attack is easily visible in all of the plots, despite the high level of noise. While AR is easily able to detect both the 5% and 10% noisy signals, the 30% and 100% signals are obscured by the base level of noise. For PASAD, the attack is noticeable for all the levels of noise but this change is exhibited in a non-intuitive way. It fails to capture the relationship between this centroid and the normal values, since the departure score denotes distance between the projected centroid and lagged signal values. In this case, the lagged vectors during training have been hovering at a close to constant distance from the centroid. The attack then shifts this relationship by swerving the lagged vector directly into the centroid, causing the departure score to get close to zero but breaking the learnt relationship.

To summarise, all attacks are very apparent when using AR and PASAD. However, in the tests where noise has been added, attacks are much more apparent when using PASAD than they are when using AR modelling.

6.8 Training and Subspace Sizes

Throughout this thesis, we have used training size of 500 and subspace size of 30 for PASAD. These values have been decided after running some initial tests on example data and determining that these are reasonable choices for that data. In a real system, these would instead depend on how many values encompass the normal system behaviour, sampling rate, and how deterministic the signal is.

If normal system behaviour repeats every 24 hours, then using a training size of 500 implies that the average sampling rate should be around once every 3 minutes. This may include increased and decreased sampling during certain hours of the day. If instead the signal repeats every hour, the average sampling rate should be every 7th second. This also depends on the degree of subsampling involved. If the system behaviour contain a lot of duplicate values, the signal can be subsampled while still giving an accurate reflection of the normal system behaviour.

With this in mind, we have designed tests which compare throughput for PASAD when varying the training and subspace sizes, as shown in Figure 6.14. Since throughput for uniform enqueuing distributions do not change very fast, an exponential increase for training and subspace size has been used. This also gives a better idea of the general trend. The upper bound for the training and subspace size has been set after discussions with our thesis supervisors, comprising of upper limits used previously in related research. We also take into account the behaviours of production ICS signals gleaned from discussions with ABB [15].

One limit with this test is that simply reading in the CSV dataset containing the values leads to a large amount of memory being used. This can be avoided by reading in the datafile in batches, freeing memory as values gets processed. However, this causes extra delays for storage I/O and impacts the calculation of throughput. Consequently, we have opted to set an upper bound of 4,096 for the training size. For a real system, these values would instead be streamed over the network, only requiring storing a limited number of values.

In the following test, only the time taken to process all values after they have been enqueued is included.

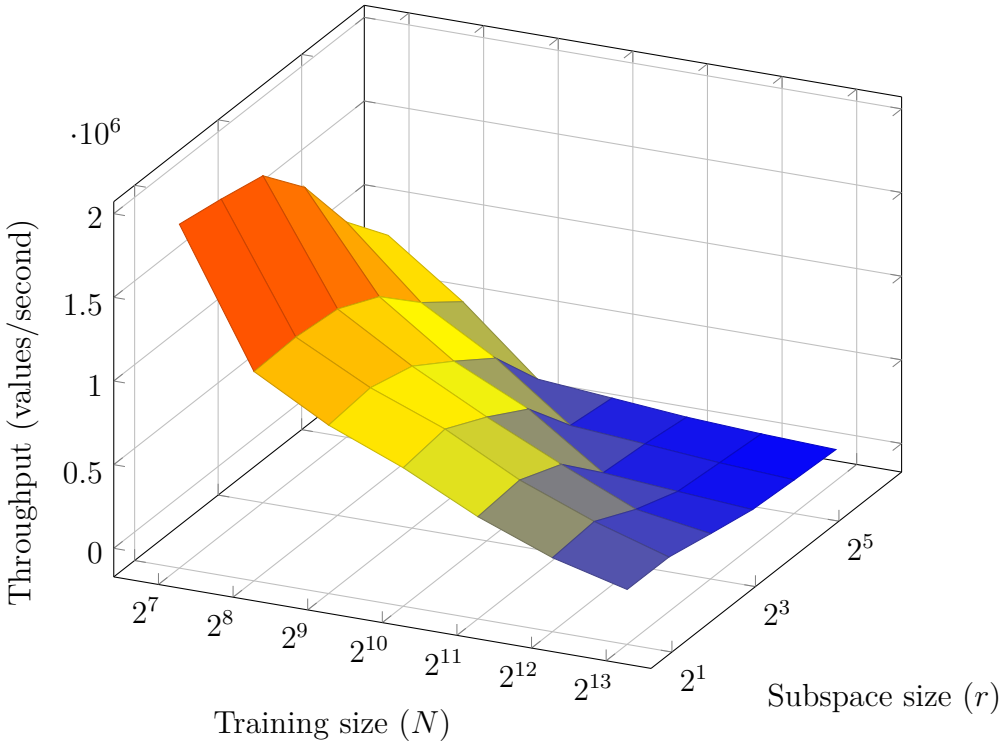

Figure 6.14: PASAD throughput for various training and subspace sizes on a single signal with 120,000 values (Intel i3, two workers).

From Figure 6.14, it is apparent that doubling the subspace size has much less impact on PASAD performance than doubling the training size. Increasing the subspace size from 2 to 64 while using a training size of 128 has the effect of decreasing performance by 55.4%. In comparison, increasing the training size from 128 to 512 has the same impact, decreasing throughput by 55.7%.

Going from a training size of 128 to 8,192 — when the subspace size is two — results in a 91.6% decrease in performance. On the other end, when the subspace size is 64, performance is decreased by 98.1% for the same range in training size. This relationship does not seem to hold when instead comparing subspace size while varying the training size. When going from a subspace size of 2 to 64 using a training size of 128 yields a 55.4% decrease in performance. The same range results in a 87.6% decrease when the training size is 4,096 — doubling five times.

6.9 GPU-Accelerated Scheme

While the OpenCL implementation of PASAD was never fully completed, the kernel of the program ended up fully functioning. As such, we have been able to run throughput tests for one iteration of the kernel.

Figure 6.15 shows the recorded throughput when different levels of buffering are used for different GPUs. The parallel computing capabilities of Mali T628 quickly max out when trying to calculate any more than 400 departure scores at a time. In contrast, GTX 1080 is easily able to accommodate calculating a very large amount of departure scores in parallel. It is worth noting that this is done at the cost of latency, so for signals where early attack detection is critical, GPU processing is no better than using the CPU.

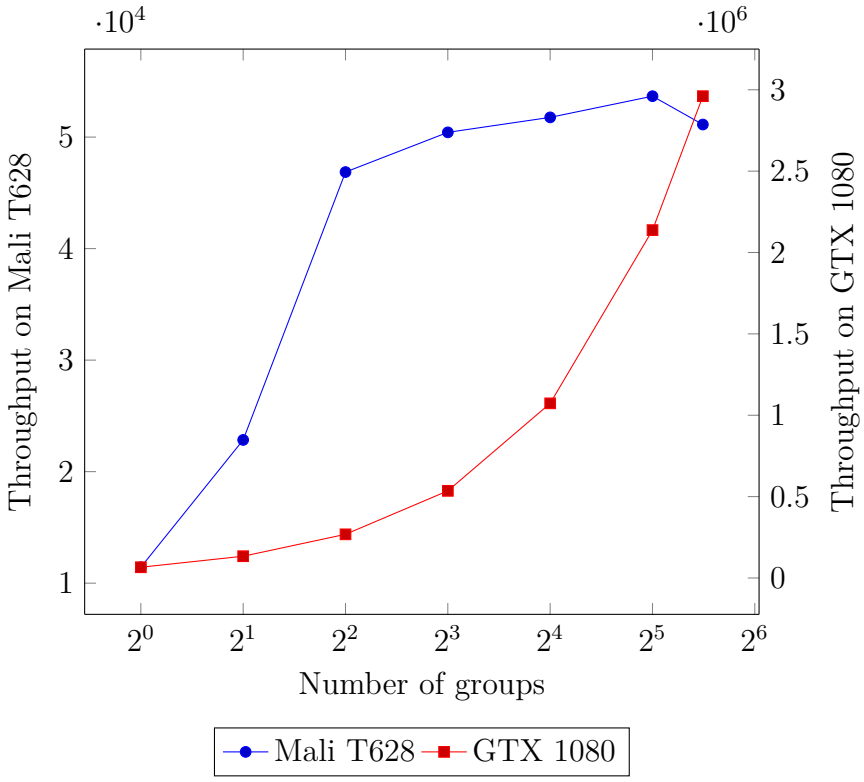

Figure 6.15: PASAD OpenCL kernel throughput versus degree of buffering on Mali T628 and GTX 1080 ($N = 200$, $L = 100$, $r = 5$).

7

Discussion

This chapter summarises the work done in this study. We discuss the insights gained from the tests and the limitations of the evaluation, as well as the larger ethical impact of this work. Section 7.1 contains a detailed discussion on whether the research goals have been fulfilled and provides conclusions that can be drawn from the test results presented previously. Section 7.2 reiterates the scope and limitations of this study. Section 7.3 gives a write-up on ethical and sustainability concerns. Finally, Section 7.4 concludes this chapter with pointers for possible future work to elevate this area of research.

7.1 Results

At the start of this report, in Section 1.2, we put forth three questions relating to the suitability of using process-level anomaly detection in ICSs. This study has aimed to provide answers to these research questions. We now present our conclusions below.

7.1.1 Suitability

The first question has been to determine if process-level anomaly detection methods are appropriate for production-grade ICSs. To determine this, we initially have surveyed various methods used in research and ended up with AR modelling and PASAD as two promising candidates. These were selected due to their accuracy, speed, and transparency.

We have had discussions with ABB representatives to learn about the landscape of current and future SCADA systems. Their largest project to date has been a SCADA system for a national power grid, which would handle up to 750,000 signals [15]. Out of these, around 10% would need to be monitored. The system is expected to generate around 10,000 values per second.

With all this information in mind, we have explored different methods to achieve high throughput when processing many signals. As can be seen in Figure 6.1, we have been able to generate a throughput of 1.3 million signal values per second for AR and 230,000 signal values per second for PASAD when processing 4,096 signals.

Our tests show that the thread pool scheme is able to maximise CPU utilisation, allowing for high performance. However, high utilisation may not be suitable for all types of ICSs. Some setups may require a certain fraction of the CPU left unused for emergency routines or other operational purposes. Therefore, when deploying the anomaly detection module, these systems would require limiting how much CPU time the process has available.

The tests from running PASAD with various training and subspace sizes show that memory is the most important concern when deciding if the method is applicable to a specific system and setup. For systems where the signals are highly regular and repetitive, PASAD is applicable for a very large number of signals, even on very limited hardware. Increased signal complexity can imply that a larger subspace is needed to model the normal signal behaviour, increasing the memory requirements. Lagged vector size also has a great impact on memory and can be large in systems where the normal behaviour cycles very slowly. If the signal can be effectively subsampled, the amount of memory needed can be greatly reduced.

The biggest factor affecting the viability of using PASAD is the number of signals that needs to be monitored. The large power grid that ABB has designed would require at least $75,000 \cdot 250 \cdot 30 \cdot 8/1024^3 \approx 4.2$ GB, which is more than most embedded systems can provide but well within the range of a modern desktop. If all signals need to be monitored, then about 42 GB of memory would be required, which is far above most desktop computers but well within the range of a dedicated server.

We have also looked at using GPUs to achieve high throughput. On the GTX 1080, we were able to achieve a processing rate of 3 million values per second when buffering 4,500 values in every batch. The throughput when buffering only a few values, the performance is worse than on both CPUs tested. For systems where a high degree of buffering is acceptable, GPUs can be used.

7.1.2 Integration

The second research question asks about the possibility of integrating process-level anomaly detection into existing production-grade ICS software. To answer this question, we have implemented the selected anomaly detection algorithms as software modules, which have then been integrated into commercial SCADA software and its simulation module. This shows that given good design, process-level anomaly detection can be integrated into existing SCADA software with relative ease.

The decision to split functions into modules has been beneficial for testing and integration with the PCU400. Our implementation has cross-compatibility between Linux and Windows and this is reflected in the choice of libraries used. The cross-compatible nature of the modules means that the anomaly detection can easily be placed in a wide variety of locations in the SCADA.

Due to the prominent market position ABB has in the global SCADA marketplace, there is a good potential that integrating process-level anomaly detection can greatly

impact the security of current and future SCADA systems.

7.1.3 Accuracy

While ABB could not provide sample data, it was noted that monitoring electrical current frequency is of special interest, and possibly the most important metric in the power grid. Because of this, our tests have been designed around anomalous sinusoidal signals.

From Section 6.7, it is clear that both methods can be used to detect anomalies with good certainty for a wide variety of attacks. However, when the noise-to-signal ratio is increased, the accuracy of AR modelling rapidly tanks while that of PASAD remains accurate. As such, for systems where only a mild level of noise is present, both methods can be used, while only PASAD is usable if the normal system behaviour includes a fair amount of noise.

From visual analysis of PASAD departure scores, it is apparent that the current way of computing it is not sufficiently adequate for identifying anomalies. This is because the departure score does not capture the relationship between lagged vectors and the centroid in a sufficient manner. Therefore, the approach taken for calculating PASAD departure scores need to be revisited.

7.2 Limitations

The behaviour of any algorithm depends on the system and the environment in which it runs. The ABB simulation package is fairly sophisticated given that ABB itself relies upon it for internal testing and inspection. However, the simulator is incapable of generating anomalies, meaning that it cannot realistically model a production system. This is a major limitation to the accuracy of how our results may translate to a deployed system.

Another point to consider is that process-level anomaly detection is only effective in attack scenarios meant to disrupt some control process. It will not be able to address the more general problem of determining if the system has been infiltrated. If the intent of the attacker is to simply spy on the control process without actually disrupting it, some other security measures would be required.

We have looked at very specific model setups for both AR and PASAD. Most tests assume that the normal system behaviour is captured in 500 signal values or less. This assumption is based on manual experimentation of departure score behaviour with sample data and generated data. By assuming these models to be representative of common signal behaviour, we can simplify our tests but this comes at a potential loss of accuracy.

We have explored the throughput for a somewhat small number of signals. This is due to the memory requirements when having a large number of PASAD models.

To allocate a PASAD model with lagged vector size 250 and subspace size 30 for 750,000 signals ends up requiring at least $750,000 \cdot 250 \cdot 30 \cdot 8/1024^3 = 41.9$ GB of memory.

We have not looked at how the performance of PASAD is affected by having multiple models with variable size. It is unrealistic that a large system with a diverse array of sensors would require the same model setup for each signal. For systems with highly variable models the applicability of our findings may vary.

Finally, all tests have been run on a very limited set of hardware, somewhat low-end compared with what is typically using for OT. As such, the applicability of the results obtained in this thesis is likely to vary.

7.3 Ethics and Sustainability

We identify two major ethical issues with our study:

- **Inferring privacy-invasive information from detected anomalies**
  Since this thesis does not use real ICS data, there are no direct privacy concerns. However, our findings show that anomaly detections can be suitably integrated into ICSs, and deploying them in production could lead to privacy concerns. For instance, anomalies in power usage could be used to infer individual actions or routines. There should be appropriate regulatory protocols put in place to avoid misuse of this information.
- **Inferring information about internal system design**
  Because this study is in the public domain and goes into details of ABB systems' internal structure, malicious actors could exploit architecture specific details when constructing future attacks. To prevent this from happening, we have made sure to not include specific information about system internals to the best of our abilities.

The sustainability aspect depends on how the system is designed and where it is deployed. Energy use is a significant concern and the extra computations involved with anomaly detection would add to it. The choice of signals on which to run anomaly detection is important. Running it on signals that are not expected to reflect system behaviour, or that are not critical, would lead to wasted energy. On the other hand, while the anomaly detection would increase energy usage, discovery of faulty parts could enable timely repair or replacement, and might offset the increased energy use.

7.4 Future Work

In this section, we present ways this area of research could be expanded. This includes possible stand-alone studies, as well extensions to our work and evaluation.

- **Evaluating BLAS libraries that offload to GPU**
  In this study, we only test the performance of implemented modules when linked against different CPU-based BLAS libraries. There are BLAS implementations that are GPU-bound, notably cuBLAS and clBLAS [13, 34]. cuBLAS is developed and supported by NVIDIA for their line of GPUs, while clBLAS is an open-source project which uses OpenCL internally. Future work could extend the evaluation in this study to cover these implementations as well, and see how they fare in industrial applications.
- **Implementing PASAD as a pure OpenCL or CUDA kernel**
  The PASAD OpenCL implementation works by each work-group calculating the full departure score at a time for 100 departure scores each. To do this, a lot of barriers have to be used to ensure all threads are processing the same step in the calculation. This forces the PASAD models to have a rather small fixed size, since each index in the lagged vector needs to have a corresponding worker thread. An alternative approach could instead have each work-group process the individual steps for all the departure scores, such as calculating the first value in the signal projection. This may increase throughput at the expense of higher memory usage since fewer barriers would be used but memory can not be reused.
- **Testing PASAD and AR on real data**
  Although we have used production-grade ICS software in this study, the data used to run the tests has been simulated. ABB were unable to provide production datasets due to the sensitive nature of their business. Using data from ICSs deployed in the real world would help understand the behaviour of these systems more accurately.
- **Processing multiple signals on GPU**
  The current PASAD OpenCL implementation can only process a single signal at time. The implementation could be extended to process multiple signals at the same time. This way, less time could be spent buffering and more time processing.
- **New method for calculating PASAD departure scores**
  The current method of using Euclidean distance to calculate PASAD departure score fails to capture the relationship between lagged signal vectors and the centroid. Alternative approaches could be explored that use clustering or machine-learning techniques.

8

Conclusion

The importance of Industrial Control Systems (ICSs) in modern infrastructure is growing and at the same time they are increasingly being targeted by cyberattacks. These attacks, irrespective of their sophistication, manifest in the form of minute variations in system behaviour. Identifying these operational anomalies could be an effective way to detect malicious actions and take necessary defensive measures. Several studies have looked into the viability of doing so, but insights into using these methods in large-scale production systems is limited.

In this study, we have investigated the feasibility of integrating two anomaly detection algorithms into ICS software: Process-Aware Stealthy Attack Detection (PASAD) and Auto-Regression (AR) modelling. We have designed and implemented a scheme to extract high performance from available hardware using a thread pooled design. The anomaly detection modules have been implemented and integrated into commercial ICS software developed by ABB, a major Supervisory Control and Data Acquisition System (SCADA) supplier. We have also experimented with an OpenCL version of PASAD to further accelerate the throughput by leveraging massive parallel capabilities of GPUs.

We find that it is possible to achieve high throughput using this scheme and capable hardware. PASAD fares very well in detecting anomalies in periodic sinusoidal signals, which are relevant to the electrical domain. This is also true for incremental noisy versions of the same. In comparison, AR can identify the initial deviation of the anomaly but fails to detect sustained irregularities. However, it has the benefit of having a lower memory footprint and significantly higher throughput. The OpenCL implementation of PASAD yields high throughput at the expense of increased latency.

In conclusion, our findings show that process-level anomaly detection is suitable for large-scale ICSs. PASAD and AR can play a reliable complementary role in Operational Technology (OT) security. Integrating and deploying them into commercial SCADA systems would have a wide-ranging positive impact for ICS security.

Bibliography

[1] ABB. *Excel DE User Guide PCU400/OPC400, Operator Manual*. [Internal; 3AJK000007-021 Rev. I, October-2016].

[2] ABB. *PCU400 Configuration Implementation Manual*. [Internal; 3AJK000007-150 Rev. I, September-2018].

[3] ABB. *PCU400 Manual, Windows*. [Internal; 3AJK000007-195 Rev. H, September-2018].

[4] ABB. *REC 501 - RP 570 Protocol Description, Technical Description Manual*. `https://library.e.abb.com/public/9a5c1896695487e6c2256a7200361578/REC501RP570_EN_A.pdf`. [Online; accessed 15-February-2019].

[5] Adrian Agogino and Kagan Tumer. 'Entropy based anomaly detection applied to space shuttle main engines'. In: *2006 IEEE Aerospace Conference*. IEEE. 2006, pp. 7–14.

[6] Magnus Almgren, Wissam Aoudi, Robert Gustafsson, Robin Krahl and Andreas Lindhé. 'The Nuts and Bolts of Deploying Process-Level IDS in Industrial Control Systems'. In: *Proceedings of the 4th Annual Industrial Control System Security Workshop*. ACM. 2018, pp. 17–24.

[7] Wissam Aoudi, Mikel Iturbe and Magnus Almgren. 'Truth Will Out: Departure-Based Process-Level Detection of Stealthy Attacks on Control Systems'. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2018, pp. 817–831.

[8] ARM. *Mali-T628 GPU*. `https://www.arm.com/ja/products/multimedia/mali-cost-efficient-graphics/mali-t628.php`. [Online; accessed 26-April-2019].

[9] Stefan Axelsson. *Intrusion detection systems: A survey and taxonomy*. Tech. rep. Chalmers University of Technology, 2000.

[10] Eric Byres and Justin Lowe. 'The myths and facts behind cyber security risks for industrial control systems'. In: *Proceedings of the VDE Kongress*. Vol. 116. Citeseer. 2004, pp. 213–218.

[11] Cisco. *Snort Homepage*. `https://www.snort.org`. [Online; accessed 04-March-2019].

[12] Cisco. *Snort Subscriber Rule Set Categories*. `https://www.snort.org/rules_explanation`. [Online; accessed 04-March-2019].

[13] Advanced Micro Devices. *clBLAS: a software library containing BLAS functions written in OpenCL*. https://github.com/clMathLibraries/clBLAS. [Online; accessed 22-April-2019].

[14] Advanced Micro Devices. *Ryzen 7 1800X Processor*. https://www.amd.com/en/products/cpu/amd-ryzen-7-1800x. [Online; accessed 23-April-2019].

[15] *Discussions with Mattias Gustin, ABB*. [Held on 30-January-2019, 01-March-2019 and 06-May-2019].

[16] James J Downs and Ernest F Vogel. 'A plant-wide industrial process control problem'. In: *Computers & chemical engineering* 17.3 (1993), pp. 245–255.

[17] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo and Aditya Mathur. 'A dataset to support research in the design of secure water treatment systems'. In: *International Conference on Critical Information Infrastructures Security*. Springer. 2016, pp. 88–99.

[18] Khronos Group. *OpenCL 2.2 Specification*. https://www.khronos.org/registry/OpenCL/specs/2.2/html/OpenCL_API.html. [Online; accessed 25-March-2019].

[19] Khronos Group. *OpenCL Overview*. https://www.khronos.org/opencl/. [Online; accessed 22-April-2019].

[20] Khronos Group. *Vulkan Overview*. https://www.khronos.org/vulkan/. [Online; accessed 22-April-2019].

[21] Dina Hadžiosmanović, Robin Sommer, Emmanuele Zambon and Pieter H Hartel. 'Through the eye of the PLC: semantic security monitoring for industrial processes'. In: *Proceedings of the 30th Annual Computer Security Applications Conference*. ACM. 2014, pp. 126–135.

[22] Hardkernel. *ODROID-XU4 User Manual*. https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf. [Online; accessed 25-April-2019].

[23] Ivan Hidajat. 'A prototype of a full-scale SCADA system installation using an operator training simulator module as power grid'. MA thesis. KTH Royal Institute of Technology, 2016.

[24] John D Hunter. 'Matplotlib: A 2D graphics environment'. In: *Computing in science & engineering* 9.3 (2007), p. 90. URL: https://matplotlib.org/.

[25] Intel. *Core i3-7100U Processor*. https://ark.intel.com/content/www/us/en/ark/products/95442/intel-core-i3-7100u-processor-3m-cache-2-40-ghz.html. [Online; accessed 23-April-2019].

[26] Intel. *Math Kernel Library*. https://software.intel.com/en-us/mkl. [Online; accessed 02-May-2019].

[27] Eric Jones, Travis Oliphant and Pearu Peterson. *SciPy: Open source scientific tools for Python*. http://www.scipy.org/. [Online; accessed 05-April-2019].

[28] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay et al. 'Jupyter Notebooks-a publishing format for reproducible computational workflows.' In: *ELPUB*. 2016, pp. 87–90. URL: https://jupyter.org/.

[29] Ronald L Krutz. *Securing SCADA systems*. John Wiley & Sons, 2005.

[30] Ralph Langner. 'Stuxnet: Dissecting a cyberwarfare weapon'. In: *IEEE Security & Privacy* 9.3 (2011), pp. 49–51.

[31] Dan Li, Dacheng Chen, Jonathan Goh and See-kiong Ng. 'Anomaly detection with generative adversarial networks for multivariate time series'. In: *arXiv preprint arXiv:1809.04758* (2018).

[32] MinGW.org. *MinGW and MSYS Project Homepage*. http://mingw.org/. [Online; accessed 12-April-2019].

[33] Netlib. *What is Netlib*. http://www.netlib.org/misc/faq.html#2.1. [Online; accessed 02-May-2019].

[34] NVIDIA. *cuBLAS*. https://developer.nvidia.com/cublas. [Online; accessed 22-April-2019].

[35] NVIDIA. *CUDA*. https://developer.nvidia.com/cuda-zone. [Online; accessed 22-April-2019].

[36] NVIDIA. *GeForce GTX 1080 Graphics Cards*. https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080/. [Online; accessed 25-May-2019].

[37] Dong Oh and Il Yun. 'Residual error based anomaly detection using auto-encoder in smd machine sound'. In: *Sensors* 18.5 (2018), p. 1308.

[38] Travis E Oliphant. 'Python for scientific computing'. In: *Computing in Science & Engineering* 9.3 (2007), pp. 10–20. URL: https://www.python.org/.

[39] Todd A Oliver, Nicholas Malaya, Rhys Ulerich and Robert D Moser. 'Estimating uncertainties in statistics computed from direct numerical simulation'. In: *Physics of Fluids* 26.3 (2014), p. 035101. URL: https://github.com/RhysU/ar.

[40] BLAS Project. *BLAS: Basic Linear Algebra Subprograms*. http://www.netlib.org/blas/. [Online; accessed 05-April-2019].

[41] Boost Project. *Boost C++ Libraries*. https://www.boost.org/. [Online; accessed 05-April-2019].

[42] Bro Project. *Zeek Network Security Monitor*. https://www.zeek.org. [Online; accessed 13-March-2019].

[43] LAPACK Project. *LAPACK: Linear Algebra PACKage*. http://www.netlib.org/lapack/. [Online; accessed 05-April-2019].

[44] Conrad Sanderson and Ryan Curtin. 'Armadillo: a template-based C++ library for linear algebra'. In: *Journal of Open Source Software* 1.2 (2016), pp. 26–32. URL: http://arma.sourceforge.net/.

[45] Ramasubramanian Sekar, Ajay Gupta, James Frullo, Tushar Shanbhag, Abhishek Tiwari, Henglin Yang and Sheng Zhou. 'Specification-based anomaly detection: a new approach for detecting network intrusions'. In: *Proceedings of the 9th ACM conference on Computer and communications security*. ACM. 2002, pp. 265–274.

[46] Keith Stouffer and Joe Falco. *Guide to supervisory control and data acquisition (SCADA) and industrial control systems security*. 2006.

[47] Tcpdump/Libpcap. *Tcpdump/Libpcap Public Repository*. `https://www.tcpdump.org`. [Online; accessed 13-March-2019].

[48] Jonathan Tompson and Kristofer Schlachter. 'An introduction to the opencl programming model'. In: *Person Education* 49 (2012).

[49] David I Urbina, Jairo A Giraldo, Alvaro A Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell and Henrik Sandberg. 'Limiting the impact of stealthy attacks on industrial control systems'. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 1092–1105.

[50] Stefan Van Der Walt, S Chris Colbert and Gael Varoquaux. 'The NumPy array: a structure for efficient numerical computation'. In: *Computing in Science & Engineering* 13.2 (2011), p. 22. URL: `https://docs.scipy.org/doc/numpy/`.

[51] vit-vit. *CTPL: C++ Thread Pool Library*. `https://github.com/vit-vit/ctpl`. [Online; accessed 05-April-2019].

[52] Qian Wang, Xianyi Zhang, Yunquan Zhang and Qing Yi. 'AUGEM: automatically generate high performance dense linear algebra kernels on x86 CPUs'. In: *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE. 2013, pp. 1–12.

[53] Zhang Xianyi. *OpenBLAS: an optimized BLAS library based on GotoBLAS2*. `http://www.openblas.net/`. [Online; accessed 02-May-2019].

[54] Zhang Xianyi, Wang Qian and Zhang Yunquan. 'Model-driven level 3 BLAS performance optimization on Loongson 3A processor'. In: *2012 IEEE 18th International Conference on Parallel and Distributed Systems*. IEEE. 2012, pp. 684–691.

[55] Yohhoy. *YAMC: Yet Another Mutex Collection*. `https://github.com/yohhoy/yamc`. [Online; accessed 05-April-2019].

[56] Mark Zeller. 'Myth or reality—Does the aurora vulnerability pose a risk to my generator?' In: *2011 64th Annual Conference for Protective Relay Engineers*. IEEE. 2011, pp. 130–136.

A

Observations

CPU	Workers	Intel MKL				OpenBLAS				BLAS			
		min	mean	max	sd	min	mean	max	sd	min	mean	max	sd
Intel Core i3	1	67.5	81.0	98.3	8.8	26.3	32.3	35.2	3.2	19.9	25.4	28.3	2.9
7100U 2.4 GHz	2	123.7	134.3	147.1	7.0	38.3	45.5	49.3	3.4	44.6	46.7	49.2	1.7
(4 cores, 8	4	114.6	123.2	132.3	5.4	39.2	44.9	50.2	3.2	55.4	59.0	60.3	1.6
threads)	8	39.5	41.4	42.8	1.1	20.3	23.6	26.6	1.9	28.8	29.5	29.8	0.3
	16	16.1	19.9	22.3	1.9	10.7	12.9	14.5	1.1	17.0	18.4	20.3	1.1
AMD Ryzen 7	1	124.9	141.1	157.8	11.7	57.7	57.9	59.3	0.5	56.6	58.1	49.2	1.2
1800X 3.4 GHz	2	187.4	198.0	219.6	11.0	87.4	100.4	115.4	8.7	89.3	96.9	110.0	7.1
(8 cores, 16	4	143.6	161.8	176.9	8.8	89.3	118.6	143.4	19.1	108.9	125.2	155.0	13.6
threads)	8	70.3	78.7	82.5	3.8	54.8	64.5	78.4	9.3	53.0	57.8	66.8	4.8
	16	29.4	32.1	35.3	2.0	31.5	34.6	39.0	2.4	28.7	33.0	38.7	2.8

Table A.1: PASAD throughput when linked against various BLAS implementations ($\cdot 10^3$ values/second)

CPU	Workers	Intel MKL				OpenBLAS				BLAS			
		min	mean	max	sd	min	mean	max	sd	min	mean	max	sd
Intel Core i3	1	737.0	818.9	873.8	51.2	829.7	837.3	851.1	6.3	815.6	840.5	859.4	11.7
7100U 2.4 GHz	2	1060.6	1110.2	1163.5	33.2	1025.3	1057.3	1091.4	21.9	991.3	1053.9	1105.6	36.3
(4 cores, 8	4	185.2	207.3	219.0	11.4	145.7	191.7	214.4	23.1	191.3	205.4	217.2	8.6
threads)	8	62.8	68.4	73.4	3.7	64.0	68.5	72.0	2.8	64.1	67.6	70.7	2.4
	16	34.5	37.1	38.3	1.2	34.6	36.9	38.4	1.3	34.5	36.4	38.1	1.0
AMD Ryzen 7	1	1063.6	1079.3	1102.2	11.6	1171.6	1200.9	1306.6	39.2	1161.2	1178.9	1068.8	15.0
1800X 3.4 GHz	2	365.6	521.7	718.4	137.6	401.7	401.7	784.6	150.3	284.5	605.0	1244.9	307.1
(8 cores, 16	4	223.6	252.9	387.6	51.1	225.3	225.3	241.1	7.2	217.2	236.2	337.1	35.9
threads)	8	116.9	118.1	118.7	0.5	118.1	118.1	119.2	0.7	117.4	118.2	118.9	0.4
	16	46.2	46.8	47.4	0.3	46.4	46.9	47.4	0.3	46.2	47.0	48.3	0.6

Table A.2: AR throughput when linked against various BLAS implementations ($\cdot 10^3$ values/second)