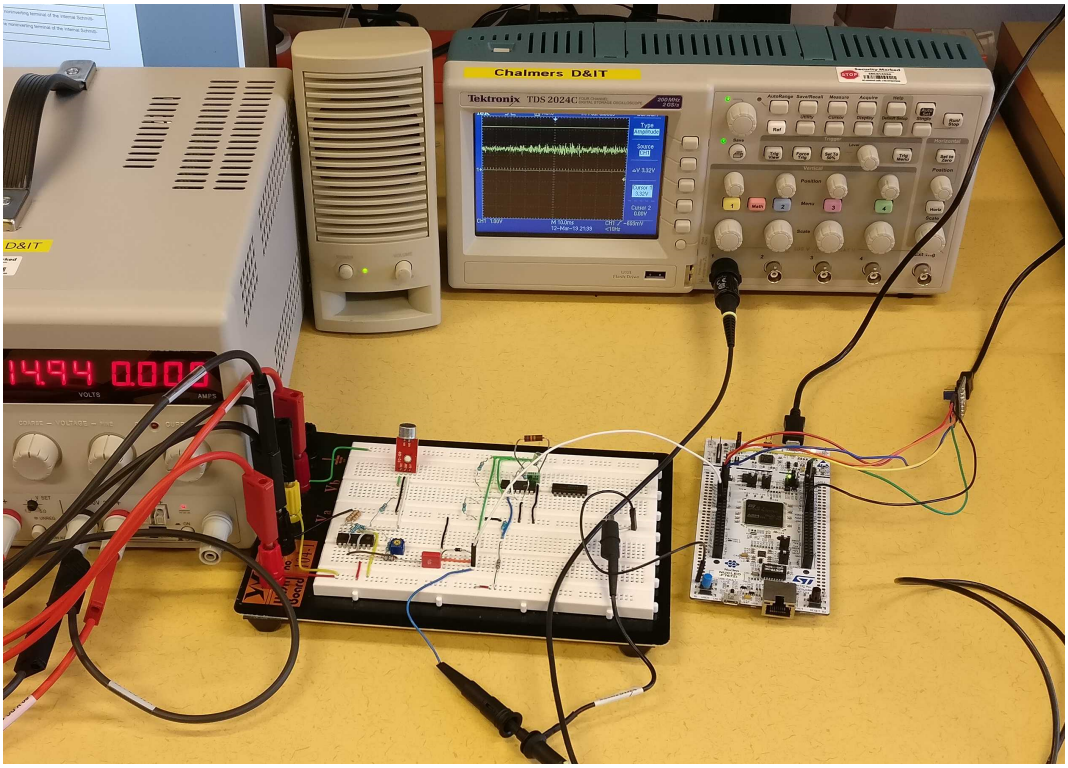
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Investigating Process-Aware Attack Detection on Embedded Systems

Master's Thesis in Computer Science and Engineering

ALBIN HELLQVIST
ALBERT OVERLAND

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

Master's thesis 2019

Investigating Process-Aware Attack Detection on Embedded Systems

ALBIN HELLQVIST
ALBERT OVERLAND

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

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

Investigating Process-Aware Attack Detection on Embedded Systems

ALBIN HELLQVIST
ALBERT OVERLAND

Supervisor: Magnus Almgren, Department of Computer Science and Engineering
Advisor: Wissam Aoudi, Department of Computer Science and Engineering
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: The intrusion detection system when using the microphone sensor.

Typeset in LaTeX
Gothenburg, Sweden 2019

Investigating Process-Aware Attack Detection on Embedded Systems
ALBIN HELLQVIST
ALBERT OVERLAND
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

In many industrial settings, there are multiple processes that need to be monitored and controlled. Examples of such processes include controlling the flow of water in a hydroelectric plant or managing the temperature in an industrial water boiler. The systems supervising these processes are called Industrial Control Systems (ICSs).

In some cases, ICSs are in control of critical infrastructure which makes them a worthwhile or profitable target for adversaries. Furthermore, ICSs are increasingly becoming targets of cyber attacks due to their increased network connectivity and integration into previously isolated systems. In addition, the advent of Internet of Things (IoT) increases the number of systems that can be targeted by similar cyber attacks. Since ICSs encompass a variety of different applications, each having its specific requirements, current methods of detecting attacks are oftentimes application-specific and not scalable. In response to the increased need for application-agnostic security, attack-detection methods with the capability of only using sensory data for detecting attacks have recently been proposed in the literature.

These recently proposed attack-detection methods are to be run in ICS or IoT environments where power consumption is of concern in addition to limited hardware resources. Consequently, the scope and the aim of this thesis is to implement and evaluate one of these recent types of methods on a resource-constrained embedded system. For this task, a state-of-the-art attack-detection method was chosen together with a suitable embedded system on which the method was implemented. Additionally, a test environment consisting of three different sensors was set up in order to have real data for the evaluation of the system.

The results show that the chosen attack-detection method is able to detect various types of attacks in real time when running on the resource-constrained embedded system. Furthermore, by tweaking certain parameters, the method could possibly run on less powerful embedded systems or with better resource utilization. Additionally, the results show that the embedded system, together with the attack-detection method, can potentially be used in resource-constrained ICS or IoT environments to detect attacks in real time.

Keywords: Industrial control systems, Internet of Things, computer security, intrusion detection system, anomaly-based attack detection, embedded systems, microcontroller, resource-constrained devices.

Acknowledgements

We would like to extend thanks to our supervisor Magnus Almgren and advisor Wissam Aoudi for providing us with the opportunity of carrying out this Master's thesis at Chalmers University of Technology. The two of them have helped us with extensive guidance and support during the thesis including the writing process, sharing of their technical expertise and help with the presentation. We would also like to thank our examiner Erland Jonsson for taking his time and reviewing our thesis.

Additionally, we are grateful for the provision of hardware components supplied to us by Lars Norén and E-sektionens Teletekniska Avdelning at Chalmers University of Technology. Furthermore, we would like to thank Viren Sinai Nadkarni and David Ström for their peer review of our thesis and patience during our noisy experiments performed in our shared workspace.

Finally, we would like to thank our families for their continued support during our time at Chalmers University of Technology.

Albin Hellqvist, Gothenburg, May 2019

Albert Overland, Gothenburg, May 2019

Contents

List of Figures

List of Tables

Nomenclature

ADC	Analog-to-Digital Converter
CAN	Controller Area Network
CPU	Central Processing Unit
HMI	Human-Machine Interface
ICS	Industrial Control Systems
IDS	Intrusion Detection System
IoT	Internet of Things
IT	Information Technology
PASAD	Process-Aware Stealthy-Attack Detection
PC	Personal Computer
PLC	Programmable Logic Controller
RAM	Random Access Memory
RC	Resistor-Capacitor
RPM	Revolutions Per Minute
RS-232	Recommended Standard 232
SDRAM	Synchronous Dynamic Random Access Memory
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver-Transmitter
VFP	Vector Floating Point

1

Introduction

Industrial Control Systems (ICS) are widely used to control industrial processes such as manufacturing, power regulation and nuclear plants. In recent years, these control systems have been increasingly integrated with a variety of network types. Additionally, with the advent of Internet of Things (IoT), more and smaller devices are connected to the Internet, oftentimes enabling their functionality to be remotely controlled. Consequently, this technology could be subject to problems associated with ICS, such as cyber attacks.

The increased connectivity suffers from an elevated exposure to cyber attacks. These attacks could, for instance, degrade the performance of an industrial process or even completely disrupt the system. These cyber attacks come in many forms, e.g., they could be stealthy in the sense that the attacks only modify process parameters slightly, thus making them difficult to detect. The goal of the attack could be to affect a competing company or to alter the process infrastructure of a sovereign state. An example of such an attack is the Stuxnet attack, which was used to sabotage the Iranian nuclear program [8].

In response to the cyber attacks on ICSs, there has been an increased interest in researching ways to improve the security of ICSs. Many new types of Intrusion Detection Systems (IDSs) have been developed to provide automatic detection of such attacks on ICSs [14]. In addition, remotely controlled IoT devices could utilize similar IDS prevention mechanisms to safeguard against attacks.

Both IoT devices and ICSs span a number of different applications, each with unique sensor data collection and data management. Consequently, detection methods for these systems are often application specific. Thus, there is an increasing need for detection methods that are general enough to be implemented on a variety of unique systems. As a consequence, there are several new application-agnostic methods for tackling these attacks that have been proposed in recent literature, such as the autoregressive model presented by Hadžiosmanović et al. [11] and the process-aware detection model presented by Aoudi et al. [2]. What is common among these methods is that they all use anomaly-based intrusion detection mechanisms. Anomaly-based intrusion detection has the property of checking if currently collected values from processes are deviating from what is deemed expected. What is considered normal operation is determined by passively monitoring process behavior.

Another method of detecting attacks is to use a signature-based IDS. In contrast to anomaly-based intrusion detection, the signature-based IDS monitors known states or parameters of the system that are changed in a predictable pattern when attacks occur. The advantage of anomaly-based intrusion detection is that there is no need for known attack signatures which enables detection of previously unknown attacks. Disadvantages include a higher false-positive rate depending on the method used [21].

There will always be limits to the amount of overhead that is acceptable in a given system and this needs to be weighted versus the cost of lowered security. It is especially important to minimize the overhead for small IoT devices that are resource-constrained thus needing all the performance available. As a consequence of these constraints, an implementation of an anomaly-based IDS on a small, embedded system and its subsequent evaluation would give further insight as to whether or not this is an avenue worth pursuing. If shown to be feasible, security of existing and future IoT devices and ICS would greatly benefit. Since the system will be designed as a stand-alone system, it could potentially work as an IDS alongside a critical industrial process in an airgapped manner. Consequently, the stand-alone system would not be subject to the same attacks as the critical industrial process.

1.1 Aim

The aim of the thesis is to investigate and evaluate a potentially viable embedded IDS solution to be used in ICS and IoT environments. Ensuring the security of, for example critical infrastructural systems, is required to responsibly deploy them. As a result, improving the security of these systems can be viewed as an enabler of these technologies.

As mentioned in the introduction, there are two types of IDSs, signature-based and anomaly-based. The first one uses known attack signatures to detect potential attacks on the system. The second IDS uses knowledge of what constitutes normal behavior and compares this to the current behavior. Consequently, anomaly-based IDS has the advantage of being able to detect attacks previously unknown due to its process-agnostic properties. Hence, the technical aim of the thesis consists of investigating the viability of implementing an anomaly-based intrusion detection algorithm on an embedded system with limited hardware resources. In addition, a suitable test environment is to be developed to enable evaluation all the way from the sensor collecting values, to the output of the attack detection algorithm running on the embedded system.

1.2 Problem Formulation

The project aims to provide answers to the following questions:

1. What needs to be taken into consideration when constructing a test environment consisting of sensors to be used in an IDS?
2. Is it feasible to run an anomaly-based intrusion detection algorithm in real time on a resource-constrained embedded system?
3. Can the intrusion detection system detect attacks that are generated with the test environment?
4. What are the resource requirements in terms of computational requirements, memory requirements and power consumption?
5. What parameters of the system can be modified and how would they affect the execution time, resource requirements and detection capabilities?

1.3 Limitations

The project is subject to the following limitations:

1. The testing of the embedded system will be done in a non-industrial, controlled environment.
2. The attack-detection algorithm is tested on one piece of hardware that will be over-dimensioned, in terms of performance and memory, to allow some leeway in the implementation and evaluation. Consequently, some parameters such as power draw could be overestimated.

1.4 Outline

To begin with, the reader is presented with an introduction to the topic in Chapter 1. It is followed by an overview of previous work related to IDS in Chapter 2. Presented in Chapter 3, is an explanation about different types of attacks, intrusion detection algorithms, embedded systems and the knowledge needed to understand the components that the test environment is constructed of.

In Chapter 4, motivation for choice of attack-detection algorithm and embedded system as well as high-level choices made when designing the system is presented. In Chapter 5 are the low-level choices made when designing the system. For example, circuit diagrams and implementation of the attack-detection algorithm on the embedded system.

The evaluation of the system as well as answers to the questions posed in the problem formulation are presented in Chapter 6. Finally, a discussion of the results, work and conclusion of the thesis are shown in Chapter 7 and Chapter 8.

2

Related Work

This chapter serves as a compilation of similar work that has been carried out within the field of intrusion detection in industrial control systems. The chapter begins by introducing the reader to related work in the general area of ICS intrusion detection, and gets more specific toward the end.

2.1 Background

Traditionally, intrusion detection systems are classified by dividing them into two categories, anomaly-based and misuse-based detection (also called signature-based detection) [14]. Anomaly-based IDS stores information about the system's previous intended operation and compares this information with the current operation of the system [18]. This comparison allows the anomaly detection IDS to detect if changes to the system's parameters fall within what is expected, and if not, an alarm can be raised. On the other hand, misuse-based detection uses a number of rules to determine whether or not a suspected attack is occurring. The rules are defined ahead of time and are configured accordingly to already known attack signatures or patterns [12].

Historically, ICS security has been focused on keeping systems isolated from networks or relying on using non-standard interfaces and protocols between components [16, 35]. In contrast, newer ICSs are often connected to networks outside of the specific domain of control systems such as the Internet [16]. Additionally, increased connectivity combined with the increased prevalence of Internet of Things devices further expands the available attack surface of ICSs [6].

When reviewing related work within ICS security, current research is focused on anomaly-based detection methods since they allow capturing an extensive number of attacks. This includes new attacks with unknown signatures which may otherwise evade misuse-based detection methods, such as zero-day exploits. What allows this is that anomaly-detection is application agnostic by nature, i.e., its method of determining an attack merely consists of comparing it to a previous, normal behavior.

2.2 Overview of Intrusion Detection

The purpose of this section is to introduce the reader to related work that identify the need and challenges of implementing intrusion detection in ICSs.

A review of intrusion detection on ICSs is presented in a survey performed by Han et al. [14]. This survey argues for the need of intrusion detection in ICSs in addition to detailing the specific security requirements of such a system and that these requirements are not fulfilled by traditional Information Technology (IT) security techniques. Furthermore, the survey presents information about the traditional taxonomy of intrusion detection in ICSs, i.e., anomaly-based intrusion detection and misuse-based intrusion detection. Additionally, the authors propose their own extension to the traditional taxonomy with the aim of providing a more comprehensive classification.

In the book by Macaulay and Singer [19], the authors identify the differences between traditional security approaches within the field of IT and security in ICSs. For instance, the authors emphasize that the results of a successful attack on an ICS almost certainly will have some type of physical consequence. When contrasted to attacks on IT systems, the effects of an attack rarely result in physical damage. Additionally, it is pointed out that systems in already existing ICSs are oftentimes resource-constrained which may make it unfeasible to augment existing systems with traditional IT security techniques.

In a study by Caselli et al. [7], the viability of using fingerprinting techniques in ICSs is investigated. Fingerprinting techniques are used to gain information about devices in a network by either passively monitoring traffic or actively sending requests to the devices. The information gathered can be used to find potential vulnerabilities in the system, such as outdated software or the use of legacy hardware. The authors recognize a number of difficulties related to using traditional fingerprinting approaches in ICSs. One of these difficulties presented are the number of unique embedded systems that are usually present in an ICS setting. The high variety of embedded systems makes the fingerprinting task more laborious since existing fingerprinting methods are oriented toward personal computers with commonly used operating systems.

2.3 Intrusion Detection for Embedded Systems

This section presents four different intrusion detection algorithms implemented on embedded systems. These four algorithms were chosen because they represent a wide variety of different approaches to potential intrusion detection algorithms. First an IDS algorithm implementation on a neural network is presented, followed by an implementation on a smart meter. Then, an implementation of an autoregressive model on a Programmable Logic Controller (PLC) is described. Lastly, a process-based anomaly detection method is presented.

Neural Network-based Anomaly Detection on a Smart Sensor

First, in the paper written by Macia-Perez et al. [20], an anomaly detection algorithm is implemented on a smart sensor in a distributed IDS context. The smart sensor consists of an ARM-based industrial board with a Linux variant aimed at embedded systems. Here, the authors use an artificial neural network to train expected behavior of TCP connections to the device. The expected behavior is obtained by inspecting a number of parameters associated with TCP connections. When the training phase is done, incoming connections are tested against the artificial neural network which will determine if the connection is anomalous or not.

Compared to the work carried out in this thesis, the proposed implementation on a smart sensor is generally on a higher level of abstraction, such as making design choices with respect to the required architecture of a distributed IDS. As such, the main focus of the paper was not to determine the most ideal anomaly detection algorithm that can be implemented. Instead, the authors investigated how different network loads affected the detection capabilities of the implemented IDS.

While the authors acknowledge the importance of resource-constrained embedded systems, the hardware used in the paper is a rather resourceful device compared to what is used in this thesis. The prototype in the paper has a rated power consumption of 4.5 W compared to the 0.33 W for the hardware chosen for this thesis. Additionally, the smart sensor runs the IDS in an operating system environment which introduces some overhead that could be avoided if implemented on a more machine-oriented level.

Memory-Constrained Intrusion Detection on a Smart Meter

Second, in the paper by Tabrizi and Pattabiraman [33], the authors aim to create a general method for developing IDSs for resource-constrained devices with an emphasis on generating no false positives. In order to avoid any false positives, the authors use a method based on static analysis which inherently can not produce false positives. To clarify, the static analysis employed in the paper uses monitoring and modeling of so-called invariants, i.e., behaviors of the system that do not change during execution of the program under normal circumstances. There are different abstraction levels of invariants, such as functional invariants that describe a control flow which remains static, and more concrete invariants such as specific function calls being made. With concrete invariants formalized, a model capable of verifying these invariants can be developed.

Furthermore, the authors formulate the generation of the IDS as an optimization problem with the memory capacity of the embedded system as a constraint. This constraint is combined with other constraint parameters associated with the overall coverage of security properties to allow the optimization of both. The resulting IDS is implemented on a smart meter that left 4 MB memory available for the IDS model, though trials with 2 MB and 1 MB were also tested. The authors note

that the most severe memory constraint tested (1 MB) had a considerable impact on the detection rate.

In contrast to the work performed in this thesis, the method presented by Tabrizi and Pattabiraman [33] needs an extensive analysis of the system to determine the invariate variables used to build the model. However, the model will ensure no false positives which is not generally true for anomaly-based models.

Autoregressive Anomaly Detection on a PLC

Third, a process-based anomaly detection method where data is captured directly from a sensor is proposed by Hadžiosmanović et al. [11]. In this paper the authors first identify normal process behavior represented as a time series of sensor values. Afterwards, an autoregressive model is applied on the time series to enable predicting what the next value in the time series should be. A significant mismatch, defined as a threshold, between the predicted value and measured sensor value would indicate some type of anomaly or attack and should be further investigated.

To test the anomaly detection method, the authors used a PLC and a Human-Machine Interface (HMI) workstation. Here, the PLC was configured to simulate an industrial process while the HMI workstation monitored the output of the process. This hardware setup was used in conjunction with Bro (now called Zeek), a network security monitor, to collect and prepare process data for the autoregressive modeling.

The work performed by Hadžiosmanović et al. [11] has little focus on resource-constrained devices and is concentrated on the actual verification of the proposed method. This is supported by the fact that the use of Bro is designed for higher level operating systems that utilize extensive resources, such as Linux, FreeBSD and Mac OS X [28].

Process-Aware Anomaly Detection on a Raspberry Pi

Fourth, a process-aware anomaly detection method is presented by Aoudi et al. [2]. Instead of using autoregressive modeling to predict future values, this method compares a series of recently sampled data to a representation of normal system behavior. The representation of normal system behavior is derived by applying the algorithm to a time series of data where the algorithm learns what is normal behavior and associates it with a number of parameters. The sampled data is evaluated and compared with training vectors, previously computed from the normal behavior. Any disparity between the outputs of the two phases is denoted as the departure score. The score is then compared to a user-defined threshold to determine whether or not the anomaly should raise an alarm.

The algorithm presented by Aoudi et al. [2] seems to detect more types of attacks in

addition to a higher detection rate than other process-aware attack-detection methods. However, there is currently a lack of evaluating an implementation of this process-aware attack detection on resource-constrained hardware. Thus, an implementation on resource-constrained hardware would serve to help determining the scalability of the proposed method.

That being said, there is one example of implementation on hardware and subsequent deployment presented by Almgren et al. [1]. Here, the authors implement the algorithm in the paper written by Aoudi et al. [2] on a Raspberry Pi 3+. The device is placed inside a paper factory and connected to a Modbus network to monitor sensor information. The sensor information consisted of information directly related to the water content of the paper surface being produced. The deployment of the algorithm on the Raspberry Pi 3+ at the paper factory was successful and showed stable behavior during its 75-day stay. While deployed, no attacks were performed on the monitored process.

2.4 Summary

Among the reviewed work, the most closely related work is found in the paper written by Almgren et al. [1], since it consists of implementing an IDS on an ICS on an embedded system. What differentiates this work from the work performed in this thesis, is that the paper by Almgren et al. [1] is mainly focused on the actual deployment and pertinence of the algorithm on a piece of hardware. Meanwhile, the investigation in this thesis is more focused on testing and evaluating the limits of the algorithm in terms of resources. Low power consumption is essential to broaden the field of potential applications that use process-based IDSs. This can be achieved by removing redundant features such as operating systems in addition to reducing the memory and clock speed manyfold.

3

Technical Background

This chapter intends to function as a resource for explaining technical concepts needed to fully understand the design and implementation parts of the thesis.

3.1 Types of Attacks

Attacks are often categorized into many different groups, but for the experiments done in this thesis, they have been divided into the following two types: direct attacks and stealthy attacks, which are presented in this section.

Direct attacks within the scope of intrusion detection systems are generally defined as attacks that aim to rapidly result in physical damage caused to the system by heavily changing control parameters or sensor readings. While direct attacks are generally simple to detect, the damage resulting from such an attack is oftentimes severe and immediate. A real-world example of a direct attack is the December of 2015 Ukraine power grid cyber-attack which led to power outages for approximately 225 000 customers [17].

The main characteristics of stealthy attacks is that they aim to go undetected while gradually inflicting physical harm, resulting in performance degradation of the attacked system. A stealthy attack can be performed by changing control parameters or sensor readings slightly, or over short periods of time, so that they can be reasonably misinterpreted for noise. One of the most famous attacks on an industrial control system that was performed in a stealthy manner is the Stuxnet attack on the Iranian Nuclear Program, first uncovered in July of 2010 [29]. The attack aimed to slow down the nuclear program's progress by changing the speed of the centrifuges in a uranium enrichment facility in Natanz, Iran. Continuously changing the speed of the centrifuges in short bursts wore them down until they were left unusable and had to be replaced.

3.2 Anomaly-based Attack Detection

There are quite a few methods using anomaly-based attack detection such as those presented earlier in Section 2.3. One common theme among these methods is that they use a training phase where sensor data is used to derive the normal behavior of a process or a system. They also share the need of a detection phase where the earlier derived normal behavior is used together with recently collected sensor data to compute if the process or the system is acting outside of its normal behavior. Since they usually share these two steps (training phase and detection phase), an anomaly-based attack detection method which uses these steps is presented in this section. An algorithm called PASAD was chosen to demonstrate how the training and detection phases work.

The algorithm is from the paper written by Aoudi et al. [2]. The algorithm in the paper, which was briefly mentioned in Chapter 2, is a process-aware stealthy-attack detection algorithm. Being process aware means that it uses sensor data for its detection phase. The algorithm consists of a training phase and a detection phase which are presented subsequently.

3.2.1 Training Phase

The training phase of the PASAD algorithm is an offline procedure which means it can be performed on a Personal Computer (PC) by having a time series of a chosen process' sensor measurements. Since this phase would be unnecessary to do on an embedded system, the math behind the training phase will not be elaborated upon. Nevertheless, the interested reader can read about the details in the paper written by Aoudi et al. [2]. However, in order to understand the detection phase, a short walkthrough of the training phase is done.

In the training phase, a time series of N sensor measurements are used to derive the normal behavior. A lag parameter L is defined which tells how many sensor values are needed when calculating the departure score, which is a score corresponding to the degree of deviation from the normal behavior. The lag parameter also has a substantial impact on the computational and space complexity.

After various mathematical operations have been run on the time series of N sensor measurements and the chosen lag parameter L, an L-by-L matrix consisting of L eigenvectors is produced. The L eigenvectors represent the behavior of the signal in the underlying time series. The eigenvectors are then sorted such that the eigenvectors with the highest eigenvalues are ordered first. After this, a so-called statistical dimension r is defined to tell how many of the sorted eigenvectors give an accurate description of the signal. The remaining eigenvectors represent the noise of the signal. The result of taking the r leading eigenvectors from the L-by-L matrix produces an L-by-r matrix denoted as U. The U matrix is then transposed into an r-by-L matrix U^{T} which is used in the detection phase to project a vector onto the

Table 3.1: Summary of the vectors and matrix used for the equations in this section.

Name	Type	Description
D	Scalar	The departure score for whether an alarm should be raised or not when compared to a predefined threshold.
$\tilde{c}$	Column vector	Represents the centroid that the signals used for the training phase form when they are projected onto the signal subspace $\mathcal{L}^r$.
w	Column vector	Represents the weight distribution of the r leading eigenvectors.
U^{T}	Matrix	Represents the r leading eigenvectors and is used when projecting a vector onto the signal subspace $\mathcal{L}^r$.
x	Row vector	Represents the L latest sensor measurements.
p	Column vector	Resulting vector when projecting the x vector onto the signal subspace $\mathcal{L}^r$.
y	Column vector	Resulting vector when comparing the centroid $\tilde{c}$ and the p vector.

r-dimensional linear subspace $\mathcal{L}^r$ spanned by the r leading eigenvectors (done by multiplying the matrix and the vector).

In addition to the matrix U^{T}, a c vector is computed from the sensor measurements from the time series. This c vector is then transformed into a $\tilde{c}$ vector representing the centroid of the signal subspace $\mathcal{L}^r$. Consequently, $\tilde{c}$ consists of r elements. Besides this, a w vector of r elements is created, representing the weight distribution of the r leading eigenvalues.

3.2.2 Detection Phase

The detection phase uses a lag vector, called the x vector, which contains the L latest sensor measurements. The detection phase is built around projecting the x vector onto the signal subspace $\mathcal{L}^r$ and then comparing the projection's difference from the centroid $\tilde{c}$. This is mathematically done by calculating the squared Euclidean distance between the projected $U^{\mathrm{T}}x$ vector and the centroid $\tilde{c}$. The mathematical expression including the weight distribution w vector for this is shown in Equation 3.1 where $\circ$ denotes the Hadamard product (entrywise product). Descriptions of the terms used for all equations in this section is shown in Table 3.1.

$$D = ||w \circ (\tilde{c} - U^{\mathrm{T}}x)||^2 \tag{3.1}$$

In order to simplify Equation 3.1, it has been separated into three steps. The first step is shown in Equation 3.2, where the x vector is projected onto the signal subspace $\mathcal{L}^r$.

$$\begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_r \end{bmatrix} = \begin{bmatrix} U_{1,1} & U_{2,1} & \cdots & U_{L,1} \\ U_{1,2} & U_{2,2} & \cdots & U_{L,2} \\ \vdots & \vdots & \ddots & \vdots \\ U_{1,r} & U_{2,r} & \cdots & U_{L,r} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_L \end{bmatrix} \tag{3.2}$$

After that, in the second step, the projected vector is compared to the centroid $\tilde{c}$ which produces a temporary vector denoted as y. This corresponds to $\tilde{c} - p$ where $p = U^{\mathrm{T}}x$, and is shown in Equation 3.3.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix} = \begin{bmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ \vdots \\ \tilde{c}_r \end{bmatrix} - \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_r \end{bmatrix} \tag{3.3}$$

After the first and second steps we have $||w \circ y||^2$ left to compute. The third step is shown in Equation 3.4 where the previously computed y vector from Equation 3.3 is multiplied with the w vector using the Hadamard product. This results in a vector which then has its norm squared, expressed as a scalar product, resulting in a scalar value corresponding to the departure score D.

$$D = \begin{bmatrix} w_1 y_1 & w_2 y_2 & \cdots & w_r y_r \end{bmatrix} \cdot \begin{bmatrix} w_1 y_1 \\ w_2 y_2 \\ \vdots \\ w_r y_r \end{bmatrix} = w_1^2 y_1^2 + w_2^2 y_2^2 + \cdots + w_r^2 y_r^2 \tag{3.4}$$

3.3 Analog Signal Handling

Even though the detection algorithm of choice is run in the digital domain, the measured signals used as input to the algorithm are analog in nature. This necessitates some kind of conversion of input signals from the analog to the digital domain. The conversion from analog to digital domain is performed using an Analog-to-Digital Converter (ADC).

The input signal to an ADC is oftentimes consisting of a voltage. This voltage needs to be within a voltage range defined by the ADC's specification, i.e., $U_{\max}$ and $U_{\min}$. The ADC converts the voltage at the input to a digital value where the range of this value is determined by the ADC's number of bits N. N number of bits allow $2^N - 1$ quantization levels to be represented, which can be translated to the

voltage resolution of the ADC. The voltage resolution of the ADC, i.e., the smallest voltage difference it can detect on its input is determined by Equation 3.5.

$$\Delta V_{\mathrm{q}} = \frac{|U_{\max} - U_{\min}|}{2^N - 1} \tag{3.5}$$

where:

ΔV_{q}	= Voltage resolution	[V]
$\lvert U_{\max} - U_{\min}\rvert$	= Full scale voltage range	[V]
N	= Number of bits	[-]

To illustrate, a 12-bit ADC with a full scale voltage range of 3.3 V has a voltage resolution of approximately 806 µV. This translates to one of the 4096 steps available for a 12-bit ADC being equal the 806 µV. In order to take advantage of all quantization levels provided by the ADC, it is recommended to make the analog input signal span the entire full scale voltage range. However, it is common for analog signals from sensors to have amplitudes many times smaller than the ADC's full scale voltage range. This could be rectified by amplifying the circuit as explained in Section 3.3.1.

In addition, the output of an analog sensor is oftentimes spanning both positive and negative voltages. Since most ADCs use unipolar voltages, i.e., $U_{\max}, U_{\min} \geq 0$, it is necessary to convert the negative voltages to positive voltages by introducing an offset. A method for solving this is explained in Section 3.3.2.

If the expected frequency content of the measured signal is known, one could take advantage of this by filtering out signals with unwanted qualities. This is elaborated on in Section 3.3.3. Additionally, if a high apparent input resistance to a circuit is needed, which is the case for some capacitive sensors, a method of achieving this is presented in Section 3.3.4. Furthermore, every system that samples a continuous signal and aims to correctly identify the signal's properties needs to fulfill sampling rate requirements. This is expanded upon in Section 3.3.5.

3.3.1 Amplifier

The amplification of signals with small amplitudes can be obtained by using an operational amplifier circuit in a non-inverting configuration using two resistors as shown in Figure 3.1.

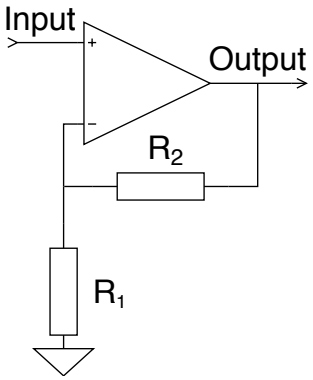


Figure 3.1: Non-inverting operational amplifier circuit.

The resistors R_1 and R_2 determine the multiplicative gain of the circuit by the relationship in Equation 3.6.

$$A_{\mathrm{V}} = 1 + \frac{R_2}{R_1} \tag{3.6}$$

where:

A_{V} = Gain of the circuit [-]
R_1, R_2 = Resistance [Ω]

3.3.2 Bipolar to Unipolar Converter

Converting a voltage range that has both positive and negative voltages to only positive voltages can be performed by using an operational amplifier summing circuit shown in Figure 3.2. This type of circuit allows a direct voltage offset (bias) to be added on top of the input signal, resulting in the addition of the two. Knowledge of the maximum expected negative value on the input can be used to derive a sufficient DC bias to ensure that the output signal will not, in theory, become negative. The resistors R_1 and R_2 decide the weights of the voltages U_{Input} and U_{Bias}. The output of the circuit is determined by Equation 3.7.

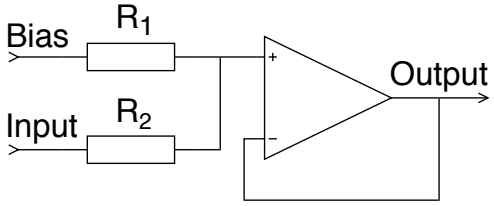


Figure 3.2: Operational amplifier summing circuit.

$$U_{\text{Output}} = U_{\text{Bias}} \cdot \frac{R_1}{R_1 + R_2} + U_{\text{Input}} \cdot \frac{R_2}{R_1 + R_2} \tag{3.7}$$

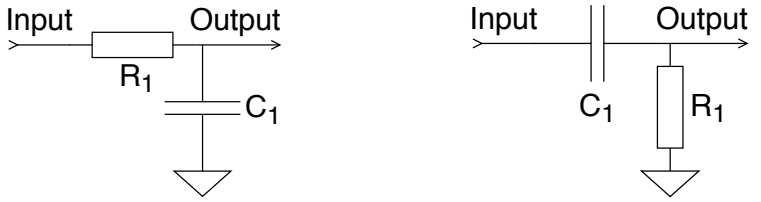
where:

U_{Output}	= Output voltage of the circuit	[V]
U_{Bias}	= Bias voltage	[V]
U_{Input}	= Input signal	[V]
R_1, R_2	= Resistance	[Ω]

3.3.3 Filters

When the desired frequency content in the input signal has been determined, it is sensible to suppress signals with frequencies outside of the chosen spectrum to minimize the out-of-band noise. A method to suppress the unwanted high frequency components is to pass the signal through a low-pass filter. The low-pass filter has a cutoff frequency f_c which is where the attenuation of the filter starts to have a significant impact on the signal. Usually, the cutoff frequency is defined as the frequency where the attenuation of the signal is $-3.01\,\text{dB}$.

A simple way to construct a low-pass filter is using a resistor and a capacitor as shown in Figure 3.3a. This filter is a first-order filter, meaning that it only consists of one frequency dependent element. As a consequence, the attenuation roll-off rate in the stop band is limited to $-20\,\text{dB/decade}$. Higher-order filters provide steeper roll-off for the price of needing more components. The cutoff frequency f_c of the low-pass filter shown in Figure 3.3a can be derived from the resistor R_1 and the capacitor C_1 as presented in Equation 3.8.



(a) Low-pass filter. **(b)** High-pass filter.

Figure 3.3: First-order low- and high-pass filters.

$$f_c = \frac{1}{2\pi \cdot R_1 \cdot C_1} \tag{3.8}$$

where:

f_c	= Cutoff frequency	[Hz]
R_1	= Resistance	[Ω]
C_1	= Capacitance	[F]

In a similar fashion one can construct a high-pass filter by switching the places of the resistor and the capacitor as shown in Figure 3.3b. The cutoff frequency of the filter is still derived by the same method as shown in Equation 3.8. The only significant difference is that frequency components below the cutoff frequency are now attenuated instead of above the cutoff frequency.

3.3.4 Bootstrap Circuit

Many types of vibration sensors have a significant capacitance associated with them which needs to considered when designing how they should be implemented. This has to do with the fact that the input resistance of the circuit that the capacitive sensor is connected to can substantially influence the output amplitude of the sensor. To understand why this is the case, the capacitance of the sensor and the resistance of the rest of the circuit can be thought of as a capacitor connected in series to an input resistor. As a result, the capacitor and the input resistor form a high-pass filter as explained in Section 3.3.3. This can lead to problems if the desired function of the circuit is to pick up low frequency components since they will be heavily attenuated by the high-pass filter. Since it is impractical to change the capacitance of the sensor, the designer needs to adapt the resistance of the external circuit to achieve the desired frequency response.

As a result, depending on the capacitance of the sensor being used in addition to how low frequencies are to be measured, incredibly high resistance values may be needed. To more clearly illustrate how high resistance values are needed to prevent the high-pass filter from attenuating low frequency signals, an example is provided. Consider that the desired cutoff frequency of the high-pass filter needs to be around 1 Hz. Also assume that a capacitive vibration sensor is to be connected to a measurement circuit, and that the required resistance of the measurement circuit needs to be determined. In addition, assume that the vibration sensor used has a capacitance of 300 pF. According to Equation 3.8, the required resistance to achieve a cutoff frequency of 1 Hz is shown in Equation 3.9.

$$R_1 = \frac{1}{2\pi \cdot 1\,\mathrm{Hz} \cdot 300\,\mathrm{pF}} \approx 530\,\mathrm{M\Omega} \tag{3.9}$$

Such high resistor values are uncommon and often need to be specially purchased. If one does not have access to such resistors, there is an alternative in the form of a bootstrap circuit. A bootstrap circuit uses an operational amplifier in a positive feedback configuration to increase the apparent resistance.[1] Using a bootstrap circuit, it is possible to achieve apparent resistance values of over hundreds of megaohms, while the component with the highest actual resistance can be as low as a few megaohms. The bootstrap can be constructed from an operational amplifer, three resistors and a capacitor as shown in Figure 3.4.

[1] Apparent resistance is the result of a voltage potential divided by a current which results in a perceived resistance being present, even though no actual resistor is used.

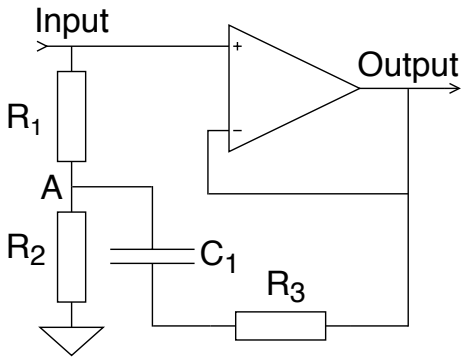


Figure 3.4: Bootstrap circuit to increase the apparent resistance.

The capacitor C_1 will prevent any direct current to impact the positive feedback of the circuit, meaning it will only feedback signals that vary in time such as sine waves. The resistors R_3 and R_2 form a voltage divider from the output of the circuit, which results in a voltage potential at point A as shown in Equation 3.10.

$$V_A = U_{\text{Output}} \cdot \frac{R_2}{R_2 + R_3} \tag{3.10}$$

where:

V_{A}	= Voltage at A	[V]
U_{Output}	= Output of the circuit	[V]
R_2, R_3	= Resistance	[Ω]

The apparent resistance from the input signal is expressed in Equation 3.11. This is under the assumption that the input resistance to the operational amplifier is much larger than the desired apparent resistance, at least by an order of magnitude.

$$R_{\text{e}} = \frac{U_{\text{Input}}}{\frac{U_{\text{Input}} - V_{\text{A}}}{R_1}} = \frac{U_{\text{Input}} \cdot R_1}{U_{\text{Input}} - V_{\text{A}}} \tag{3.11}$$

where:

R_{e}	= Effective (apparent) resistance at Input	[Ω]
V_{A}	= Voltage at A	[V]
U_{Input}	= Input of the circuit	[V]
R_1	= Resistance	[Ω]

Since the operational amplifier is configured as a voltage follower, assuming an ideal operational amplifier, the input is equal to the output, i.e., $U_{\text{Input}} = U_{\text{Output}}$. This simplifies Equation 3.10 and Equation 3.11 into Equation 3.12.

$$R_{\mathrm{e}} = \frac{R_1}{R_3} \cdot (R_2 + R_3) \tag{3.12}$$

where:

R_{e} = Effective (apparent) resistance at Input [Ω]
R_1, R_2, R_3 = Resistance [Ω]

To summarize, in order to reduce the filtering effects of the sensor's capacitance, an incredibly high input resistance is needed for the measurement circuit to which the sensor is connected. If such high resistance values are not available when building the measurement circuit, one can use the bootstrap circuit to emulate the needed resistance using resistance values that are orders of magnitude lower.

3.3.5 Sampling Theorem

Since the embedded system uses an ADC, it converts analog signals to a discrete representation by the method of sampling. This means that the sampling rate of the ADC needs to be determined which leads to a limitation of the frequency components that can be reliably measured, as decided by the Nyquist–Shannon sampling theorem. This theorem states that the minimum sampling rate f_{s} of the system needs to be higher than twice the bandwidth B of the sampled signal in order to avoid aliasing. Aliasing is when multiple signals with different frequencies can not be distinguished from one another due to a low sampling rate. The mathematical formulation of the sampling theorem is shown in Equation 3.13.

$$f_{\mathrm{s}} > 2B \tag{3.13}$$

where:

f_{s} = Sampling rate [Hz]
B = Signal bandwidth [Hz]

3.4 Embedded Systems

In this section, hardware components and parameters of three different embedded systems are presented. The embedded systems are the following: the Raspberry Pi 1 Model A+ presented in Section 3.4.2, the Arduino Due presented in Section 3.4.3 and the STM32F767ZI presented in Section 3.4.4. One of these systems will be chosen in Section 4.3 to be used for the implementation of the attack-detection algorithm that will be chosen in Section 4.2. These embedded systems are all commercially available and were chosen because they provide a variety of different different hardware components and parameters. This involves computational performance, amount of memory, power consumption, the availability of

analog-to-digital conversion and double-precision floating-point support. Potentially useful peripherals and interfaces will also be listed if available. Examples of these are Serial Peripheral Interface (SPI), Universal Asynchronous Receiver-Transmitter (UART), Controller Area Network (CAN) and Ethernet. Before the embedded systems are presented, a brief description of the terms used is given in Section 3.4.1.

3.4.1 Microcontroller Features

Some of the concepts and terms used in the upcoming sections may be unknown to the reader, hence, this section is intended to give a brief description of the terms used. The descriptions are presented in a list for the sake of simplicity.

- **Static Random Access Memory (SRAM)** is a type of Random Access Memory (RAM), where the memory does not need frequent refreshing of the stored bits. This results in lower power consumption compared to those memories that require frequent refreshes. The memory is also faster compared to the SDRAM (which is discussed next), however, it can only store about 1000 times less data than the SDRAM.
- **Synchronous Dynamic Random Access Memory (SDRAM)** is a type of RAM where the memory does need to be frequently refreshed in order to keep the correct bits. Because of this, the power consumption is higher than that of the SRAM. Furthermore, it is clocked together with the processor which results in slower performance. Furthermore, the size of the memory can be about 1000 times larger than the SRAM.
- **Flash memory** is a non-volatile type of memory which often comes in two forms, NOR or NAND depending on the gates used in the circuit. Flash memories are very fast compared to ordinary magnetic storage, with the disadvantage of being more expensive. It also has a limited amount of write cycles per gate before that gate is non functional. Further explanation of flash memories can be seen in Section 3.5.
- **Single-precision floating-point format** is a format used when one needs to store a decimal or a big numerical value. It uses 32 bits (4 bytes) and is able to store values between $1.2 \cdot 10^{-38}$ and $3.4 \cdot 10^{38}$. Furthermore, it can typically represent 7 or 8 decimals.
- **Double-precision floating-point format** uses 64 bits (8 bytes) instead of 32 bits (4 bytes) and is able to store values between $2.2 \cdot 10^{-308}$ and $1.8 \cdot 10^{308}$. Furthermore, it can typically represent 15 or 16 decimals.
- **Analog-to-Digital Converter (ADC)** is a peripheral which takes an analog signal and converts it into its digital representation. Usually, a sampling rate can be defined which determines the frequency of how often the ADC should convert values. Furthermore, since the digital representation is

discrete, a resolution is used which tells how many bits are used for the discrete representation of the analog signal. Further explanation of the ADC can be seen in Section 3.3.

- **Serial Peripheral Interface (SPI)** is a communication interface that is used to enable a serial link between a master, for example a microcontroller, and slaves such as other peripherals. Examples of other peripherals could be an external math coprocessor or non-volatile storage. The transfer speed is typically a few megahertz.
- **Universal Asynchronous Receiver-Transmitter (UART)** is an interface used to facilitate serial communication between two devices. It is asynchronous, meaning that the devices do not need to be synchronized using any common clock or similar. It is also typically simple to use as well as widely used. A more detailed description of asynchronous serial communication is given in Section 3.6.
- **Controller Area Network (CAN)** is a serial asynchronous protocol used in a two-wire bus network. CAN allows only one node on the network to broadcast data at any instant even though all nodes are allowed to initiate sending and receiving of data. This means that each node in the network needs to check if the bus is idle before beginning to broadcast data on it. The CAN protocol is often used in networks where the intended nodes consist of sensors or actuators.

3.4.2 Raspberry Pi 1 Model A+

The first presented embedded system is the Raspberry Pi 1 Model A+ which uses an ARM1176JZF-S Central Processing Unit (CPU) running at a clock frequency of 700 MHz [27]. Additionally, it features 512 MB of SDRAM and a micro SD card slot for non-volatile memory.

Aside from the CPU, a Vector Floating-Point (VFP) coprocessor is also provided, capable of storing 16 different double-precision floating-point values [4]. It can perform double-precision floating-point loads, stores, additions, subtractions and multiplications at low clock cycle cost. Since the load and store, i.e., the operations that need to transfer data from and to the coprocessor, are fast, the data transfer time will not to be a factor in the choice of embedded system.

The Raspberry Pi does not feature any built-in ADC so an external component for this is needed. Two ADCs have been considered, the first one, MCP3004 which has a 10 bit resolution and the second one, ADS1015 which has a resolution of 12 bits [23][15]. The first ADC can provide a sampling rate of 200 kS/s, while the second ADC can provide a sampling rate of a 3.3 kS/s.

In addition to the items already presented, the board features several peripherals and interfaces such as SPI and UART [10]. Regarding the power consumption, the board

is rated at around 180 mA at 5 V under normal use according to the manufacturer, which gives a power draw of 0.9 W [9].

3.4.3 Arduino Due

The second presented embedded system is the Arduino Due which currently is the most powerful board in the Arduino series. It uses an AT91SAM3X8E microchip which is based on an ARM Cortex-M3 processor that can be run at clock frequencies up to 84 MHz depending on the configuration [3]. Additionally, it features 96 kB of SRAM and 512 kB of non-volatile flash memory. The flash memory can be set to have a read width of 128 bits for better access speed, or a width of 64 bits for lower power consumption [5]. Each read operation typically takes only a few clock cycles.

The Arduino Due does not have any hardware support for single- nor double-precision floating-point calculations. A math coprocessor can be connected through its SPI interface, but both the math coprocessor and the speed of the SPI protocol might too slow or complex to be used in this thesis [3]. However, the processor can run floating-point operations by software with the disadvantage of using several hundred clock cycles per operation.

The board has 12 analog inputs in the form of ADCs where each one can be set to have a precision up to 12 bits [3]. The sampling rate is somewhat based on the clock frequency of the CPU and is able to have a sampling rate of around 100 kS/s. Furthermore, the board features several peripherals and interfaces such as CAN, UART and the previously mentioned SPI. As for the power consumption, no figures are given by any vendor, still amateur estimates found online seem to place it around 90 mA on 7 V which gives a power draw of 0.63 W.

3.4.4 STM32F767ZI

The third presented embedded system is the STM32F767ZI which is based on the ARM Cortex-M7 processor [32]. It runs on clock frequencies up to 216 MHz depending on the configuration. Additionally, it features 512 kB of SRAM and 2 MB of non-volatile flash memory. The flash memory can be set to work in single or dual bank mode where as for single bank mode it has a read width of 256 bits [31]. Each read operation typically takes only a few clock cycles.

Together with the processor comes a floating-point unit capable of performing both single-precision and double-precision floating-point operations [31]. Similar to the Raspberry Pi's VFP coprocessor, the floating-point unit has 16 different registers for storing double-precision floating-points numbers. It can also perform double-precision floating-point loads, stores, additions, subtractions and multiplications at a low clock cycle cost. The transfer speed to and from the floating-point unit will be omitted as it was for the Rasperry Pi.

The board has three ADCs that can have a sampling rate of up to 2.4 million samples per second [32]. Additionally, it has several UART, SPI and CAN interfaces which can be used for various applications. It also features a 10/100 Ethernet interface that can be used to transfer large amount of data to and from the device at high speed. As for the power consumption, it was calculated (with the help of STM32's CubeMX power consumption calculator) to be around 100 mA at 3.3 V which gives a power draw of 0.33 W.

3.5 Flash Memory

The fundamental behavior and structure of the flash memories used in embedded systems are important to comprehend in order to understand how writing and reading to and from the memory is performed. To begin with, the flash memories are usually divided into several sectors of equal or unequal sizes. An example of the STM32F767ZI's memory structure can be seen in Figure 3.5. Once a bit has been written, it cannot be rewritten before it has been erased, i.e., when erasing, the bit is set to 1, which is the default value. The erasing can only be performed sector-wise, hence the need for dividing the memory into sectors. Additionally, the sectors have a limited amount of times in which they can be written to before they are rendered defective. However, reading from the memory can be performed anywhere as many times as needed.

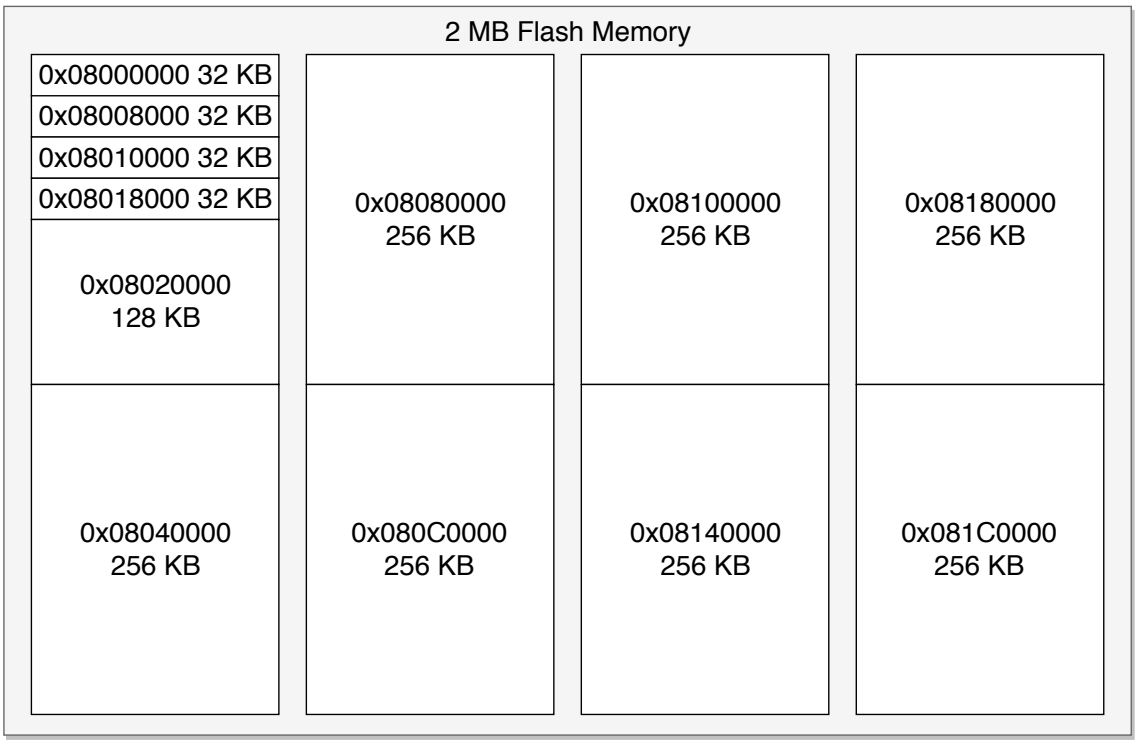


Figure 3.5: The addresses and sizes of the sectors in the flash memory of an STM32F767ZI.

The limitations of the writing and erasing of bits make smaller flash memories unsuitable for cases where many rewrites on the same location are needed. Consequently, the flash memories are usually used to store the program code in addition to any data which the program might use. This leaves the responsibility of storing volatile data to the RAM. Furthermore, having the memory structured into sectors with fixed addresses (see Figure 3.5) makes it easier to determine where the data should lie and be read from.

When flashing data on the flash memory from an external source (e.g., a PC), the data often comes from a binary file. The binary-file consists of raw data and could for example be training data used for the detection phase in an anomaly-based attack detection algorithm. Since the binary-data could be flashed for example on the memory address `0x08020000` and forward, the program running on the embedded system could be programmed to read its data from that memory address. Consequently, embedded systems often work with memory addresses for the flash memory and not with file systems when locating data.

3.6 Asynchronous Serial Communication

Serial communication is widely used in computer systems when transferring data between two or more devices. There are several serial protocols which define the underlying electrical characteristics and timing of the signals such as Ethernet, Recommended Standard 232 (RS-232) [24], SPI and Universal Serial Bus. Since serial communication is of interest in this thesis, parts of the RS-232 protocol, which is commonly used in embedded systems, will be explained in this section.

The RS-232 standard supports both asynchronous and synchronous transmissions. Asynchronous transmissions are when the two connected devices are not sharing a clock signal, and thus rely on that the receiving side can notice that the line is being transmitted on. On the other hand, synchronous transmission uses a shared clock signal which helps the receiving side to know when a new transmission is being performed. The difference between the two transmission types are that asynchronous transmissions are slower. However, they rely on less circuitry and omits the need for a common clock signal. The data package (or frame) used for asynchronous transmissions will be expanded upon in this section due to its simple enabling of transferring data.

There are several interfaces that can utilize the RS-232 standard to enable communication between devices where one example of such an interface is UART. The interface works by reading parallel data from a bus, and then translating it to a serial representation which is then passed through a voltage level converter to match the RS-232 standard. The RS-232 standard enables communication between devices by defining voltage levels to be used during the transmission in addition to pinout.

Before an actual transmission can be performed between two devices, they need to agree upon the baud rate, if a parity bit is to be used, how many data and stop bits

they should use and some additional parameters. These terms will be explained as they come below. The line between the two asynchronously communicating devices is by default held at the high voltage level when no transmissions are being performed. An example of an asynchronous serial transmission of 8 bits can be seen in Figure 3.6. When the transmitting side starts sending a data frame, the data frame begins with a start bit held at a low voltage level to indicate that a transmission has started. The following bits after the start bit are the agreed number of data bits containing the data, which usually are between 5 to 9 bits. Afterwards, a parity bit can be sent to be used for error checking, this is however optional. Finally, 1 or 2 stop bits follow the parity bit or the data bits, which indicate that the transmission of the frame is done.

Figure 3.6: Example of the clock pulses in an asynchronous serial transmission of 8 bits using RS-232.

In order for the two communicating devices to know the speed of the transmission, they agree upon the so-called baud rate. The baud rate is the total number of bits (including start, data, parity and stop bits) that can be transferred per second. Typical baud rates for asynchronous serial transmissions are between 9600 bits per second and 115 200 bits per second.

3.7 Summary

In this chapter, the technical background needed to understand the design choices and the following implementation was presented. First the reader was introduced to a classification of attacks that can target ICSs and what the characteristics of these attacks are. In addition, a description of a method of anomaly-based attack detection was presented as well as how it uses a training phase, a detection phase and what resource requirements are present. Furthermore, there was an explanation of what components are involved in converting an analog signal to its digital representation. Following was a presentation of three candidate embedded systems and their specifications. Finally, details about flash memory and asynchronous serial communication were presented.

4

Design

This chapter elaborates on the higher level design decisions that have been taken when designing the embedded intrusion detection system. There are multiple considerations and tradeoffs to be made when designing an intrusion detection system for an industrial control system. The design consists of choosing the detection algorithm to be used in addition to what type of embedded system it should be implemented on. Furthermore, the top-level system design is presented including the design of the test environment, memory management in addition to the buffering of incoming data and how to handle data transfers to and from the microcontroller.

4.1 Design Overview

A simple model of an industrial control system anomaly detection algorithm on an embedded system can be divided into three main stages as shown in Figure 4.1. First, the system needs to be able to collect measurement data using an analog sensor. The analog sensor would typically be some kind of monitoring sensor used in an industrial setting. Second, the output of the analog sensor needs to be converted to the digital domain in order to prepare the measurement data for digital processing. This digitization process is performed using an analog-to-digital converter. Third, the quantized measurements are fed into an anomaly detection algorithm running on a processor. Depending on the input, the IDS algorithm outputs a value indicating whether or not an anomaly has been detected depending on a tunable threshold. The main components used for carrying out the three processes, in addition to the direction of data between them can also be seen in the figure.

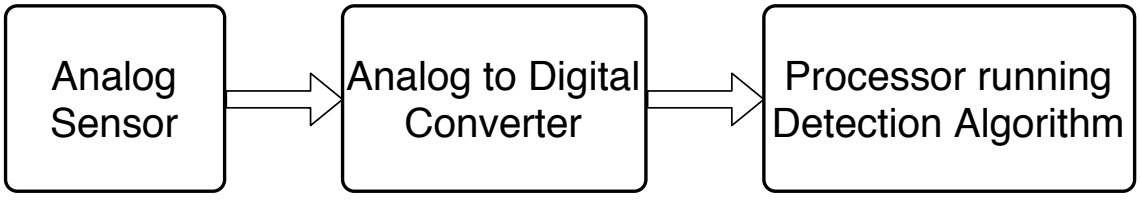


Figure 4.1: A simple model of an ICS anomaly detection system.

With an abstract model of the main components available, one needs to choose

what type of hardware should be used for each stage in Figure 4.1. When choosing the hardware it is recommended to use hardware that incorporates as many of the stages as possible in order to simplify the design. Luckily, there is readily available hardware which oftentimes has the capability of performing analog-to-digital conversion in addition to supporting the execution of algorithms in software. One of these types of hardware is the microcontroller which is a common component in many embedded systems. There are countless of microcontrollers to choose from, each with different computational performance, memory sizes and peripherals. In order to choose what type of microcontroller to be used, one must take into account the performance requirements of the algorithm. In addition, the characteristics of the analog-to-digital converter, if available, need to be considered. Examples of such characteristics include how fast and often an analog value can be converted and how many bits of resolution the converted value is composed of.

When choosing the algorithm to be run on a resource-constrained embedded system, it is of interest to choose between algorithms that are resource-efficient in order to minimize hardware requirements. One such type of algorithm was chosen, namely an anomaly-based detection algorithm which is further elaborated on in Section 4.2. Furthermore, the final choice of embedded system is presented in Section 4.3 with an overview of the system shown in Section 4.4.

4.2 Choice of Anomaly-based Attack Detection

Many factors play a role when looking at candidate anomaly-based attack-detection algorithms to be implemented on an embedded system. Such factors can be the detection rate or false alarm ratio. When it comes to resource-constrained embedded systems, other factors need consideration, such as the hardware requirements to actually run the algorithm. The algorithm needs to be compact enough memory-wise since the resource-constrained embedded system typically only has a few megabytes of available memory. In addition, the computational requirements also need to be low enough to be able to run on clock speeds which usually are around a few hundred megahertz. Furthermore, the architecture of the processor and memory to run the algorithm can severely hinder the computational effectiveness. To illustrate, if the algorithm needs a high degree of parallel processing and only has access to a CPU instead of a GPU, the performance will most likely suffer.

There are three types of anomaly detection algorithms considered. The number and types of detection algorithms investigated were decided after an initial survey. The first one is an autoregressive model, meaning it is based on values previously collected to predict what the next sampled value is expected to be. The second algorithm uses a neural network to train what represents normal behavior. Then it provides a function that represents a window of expected values where this window has a user-defined size. The third and final algorithm is a process-aware attack detection algorithm that uses methods from linear algebra to provide a noise resistant detection model.

4.2.1 Autoregressive Anomaly Detection

In a paper presented by Hadžiosmanović et al. [11], an autoregressive intrusion detection model is proposed. This autoregressive model uses two different stages for detecting anomalies in a time series of data. These two stages consist of a training stage and an detection stage. First, the autoregressive model needs to be trained in order to understand what constitutes normal behavior of a given time series. To do this, the training stage is fed a time series of data where the data is corresponding to what is expected during normal behavior. Afterwards, the output of the training stage contains a number of coefficients which will be used in the detection stage where the anomaly detection takes place.

The detection stage of autoregressive modeling is characterized by using a linear function, where the input to the function consists of recently collected values in the form of a time series. Each value in the time series is multiplied by the corresponding coefficient determined by the training stage. The resulting value is a scalar which has the purpose of representing the expected behavior of the time series. This value can be compared to future values to detect any deviation from what is expected. In order to change how far back in time previous values should influence the currently predicted value, one can change the number of coefficients to be used. The optimal number of coefficients to use depends on the characteristics of the time series used.

Finally, the autoregressive model is completed by adding a stochastic term to the linear function. This stochastic term is a randomly varying value which emulates noise in the model. The exact characteristics of the noise, such as its distribution, mean value and variance are dependent on the specifications of the time series used in the training phase. In a simple sense, the stochastic term can be interpreted as a measure of the regression model's accuracy.

4.2.2 Neural Network-based Anomaly Detection

A machine learning-oriented anomaly detection method using a neural network is presented by Shalyga et al. [30]. This method of detecting attacks uses an algorithm based on neural networks. A neural network is built from a number of layers where each layer consists of several nodes. The layers can be thought of as a number of vertical columns situated next to one another. Each layer in the neural network is connected by its nodes to the two adjacent layers in every possible combination. This means that the leftmost and rightmost layers are only connect to one other layer since they only have one adjacent layer. These two layers are responsible for handling the input and output of the neural network. By training the neural network on data representing normal behavior of a time series or other process, the network becomes increasingly adept at predicting this behavior.

The basic idea presented by Shalyga et al. [30] is to provide the neural network with parameters related to the previous behavior of a system deemed normal which can be summarized as a system state. The exact parameters used to derive this system state

varies between applications. For example, it can be attributes of a TCP connection such as port number, bytes received or sent, response time, connection attempts and many other properties.

The neural network uses the system state to come up with a function that predicts expected values in the future. The number of predicted values, i.e., how many data points into the future are to be predicted can be determined by setting, what the authors call, a forecasting horizon parameter. For example, setting the forecasting horizon parameter to a value h results in h number of data points being predicted in the future.

Since neural networks can have a number of different architectures, such as number of layers, it is of interest to determine the most optimal architecture to solve the problem at hand. To solve this, the method uses a genetic algorithm to automatically determine the most suitable architecture. A genetic algorithm provides a way of determining increasingly better architectures with respect to some constraint. Genetic algorithms utilize a process similar to natural selection by choosing potential solutions from a pool of candidate architectures. After, the potential solutions picked from the pool are run against each other until only the best performing solution is left.

4.2.3 Process-Aware Stealthy-Attack Detection

A process-aware anomaly detection algorithm has been introduced by Aoudi et al. [2] named PASAD. Similarly to the other anomaly detection algorithms, PASAD uses a training phase to determine the expected behavior of a time series of data. The training phase uses singular spectrum analysis as a mean to distinguish process noise from the actual deterministic behavior of the process being monitored. As such, PASAD provides a way of detecting the behavior of the process with a lower influence from noise.

When the training phase is completed, a detection phase is used which takes each input value collected and compares it to the result of the training phase. Consequently, the detection phase can determine whether or not the recently collected values are in line with the normal behavior of the time series used in the training phase.

The output value from the detection phase used for quantifying the degree of deviation from the normal behavior is called the departure score. This departure score is an accumulation of how much the recently acquired values depart from what is expected of the monitored process. Monitoring the departure score of a chosen process can be used to derive a threshold value which establishes the acceptable variation of the mentioned score. If the detection phase produces an output which exceeds the threshold, a potential attack on the system could be occurring and an alarm of choice can be triggered.

4.2.4 Choosing the Algorithm

The advantage of using an autoregressive model over the other two algorithms comes down to its simplicity of the detection stage. The detection stage to make a prediction is simple since it consists solely of a sum of multiplications between incoming values and the coefficients from the training stage.

The neural network method where the architecture is chosen by a genetic algorithm could theoretically become incredibly proficient at detecting anomalies in a time series of data. However, this hinges on if the algorithm is given extensive time for training in addition to having formidable hardware resources available. This being said, even with access to powerful hardware, the time it takes to come up with the optimal architecture still takes a long time task [30]. For example, it took around 33 hours to find the optimal architecture for the dataset used when using a server-grade 6-core CPU along with a GPU with 64 GB RAM [30].

Another reason that makes neural networks an ill fit on current microcontrollers is that neural networks run most efficiently on hardware which provides a high degree of parallelization. This is due to the inherent structure of neural networks which can take advantage of parallel computing architectures. Microcontrollers are generally using CPUs for processing which have a sequential architecture as opposed to GPUs which has lots of hardware support for parallel execution. This being said, there have been neural network implementations on microcontrollers, although limited in size [22].

Using PASAD allows for better detection rates and the ability to detect more types of attacks than the autoregressive model [2]. One of the reasons for the better performance is that the algorithm is highly insensitive to noise. As a result, it is better at detecting attacks that are stealthy in nature, i.e., trying to hide within noise levels of the process. However, PASAD does use more resources for its detection phase, especially when it comes to memory usage. The additional memory used is due to the increased storage size of the results of the algorithm's training phase which is used to describe the normal behavior. However, the memory usage does not seem to be absurdly high in comparison to what is available on many commercially available microcontrollers. The scenario with the highest memory usage presented in the paper by Aoudi et al. [2] uses around 1.7 MB of data which is a reasonable size of the flash memory available on an microcontroller.

Due to the ill-fitted architecture of microcontrollers when implementing neural networks and the better performance of PASAD as opposed to the autoregressive model, PASAD was the algorithm of choice to be implemented on the embedded system.

4.3 Choice of Embedded System

In this section, motivations will be presented for choosing the embedded system which is used for the implementation of the PASAD algorithm. The decision will be made between the following embedded systems that were presented in Chapter 3: Raspberry Pi 1 Model A+, Arduino Due and the STM32F767ZI. The choice to include these specific embedded systems was motivated by the fact that they are widely used as well representing a wide range of features, e.g., different clock frequencies, memory sizes and floating-point support. A summary of the embedded systems can be seen in Table 4.1.

Table 4.1: Summary of the embedded systems.

Embedded system	**Raspberry Pi 1 Model A+**	**Arduino Due**	**STM32F767ZI**
Clock frequency	700 MHz	84 MHz	216 MHz
RAM	512 MB SDRAM	96 kB SRAM	512 kB SRAM
Non-volatile memory	SD card	512 kB flash memory	2 MB flash memory
Floating-point support	Single/double precision	No	Single/double precision
Built-in ADC	No, SPI connected ADC	Yes	Yes
ADC resolution	10 bit or 12 bit	12 bit	12 bit
ADC sampling rate	200 kS/s or 3.3 kS/s	100 kS/s	2.4 MS/s
Additional peripherals	SPI and UART	CAN, SPI and UART	CAN, Ethernet, SPI and UART
Power consumption[1]	0.9 W	0.63 W	0.33 W

[1]These numbers should not be seen as a good representation of the embedded systems' real power consumption. This is discussed further in Section 4.3.5.

A few metrics and features will be used for the comparison to decide upon which one of the embedded systems that is most suitable for the task. These metrics and features include computational performance, amount of memory, power consumption, the availability of analog-to-digital conversion and double-precision floating-point support.

The computational and memory requirements will be based on the most demanding experiment from paper written by Aoudi et al. [2]. The experiment, called *DA2*, consists of a direct attack on a pressure sensor for a reactor, resulting in anomalous

behaviour in a controller that uses this sensor. It had an L value of 5000 and an r value of 43. With these parameters and when the use of double-precision floating points is assumed, one knows that the experiment requires a storage size of $(2 \cdot r + r \cdot L) \cdot 8$ bytes. If we use an L value of 5000 and an r value of 43, we get that $(2 \cdot 43 + 43 \cdot 5000) \cdot 8 = 1\,720\,688$ bytes are needed to be stored for the training vectors with the chosen experiment as a basis. Thus, around two megabytes of memory are needed.

4.3.1 Computational Performance

When it comes to performance, the Raspberry Pi is a lot faster with its 700 MHz clock frequency compared to the Arduino Due's 84 MHz and the STM32F767ZI's 216 MHz. However, the Raspberry Pi's RAM is of SDRAM type which means that it has a slower access time than the SRAM which the other microcontrollers feature. Despite the slower access time of the SDRAM, this difference gets cancelled out to some extent since the other microcontrollers do not have enough RAM to store all the training vectors. This results in the need for real-time transfer of data between the RAM and the flash memory which can be costly.

Moreover, the Arduino Due does not have hardware support for floating-point calculations which forces the system to perform these operations in software instead of hardware. Consequently, an approximated overhead of 200 extra clock cycles per floating-point operation is assumed. The overhead is based on various online sources and was roughly estimated in order to be able to compare the Arduino Due's performance to the other embedded systems. This constraint makes the Arduino Due impractical to use if the algorithm is to be run in real time. The reason for this is made clear by using a simple example. Assume that one uses an L value of 5000 and an r value of 43, then it would take the Arduino Due $(43 \cdot 5000 \cdot 200)/(84 \cdot 10^6) = 0.51$ seconds to compute the restricting matrix multiplication in the algorithm which is considered to be slow. It is also important to note that the necessary data is assumed to already be located in the RAM, which due to the memory size constraints, is not possible.

4.3.2 Memory

The Raspberry Pi outperforms the other microcontrollers in terms of memory, because of its large RAM. The size of its RAM makes it possible to store all the training vectors without having to load parts of them from any non-volatile memory in real time. The STM32F767ZI's RAM can hold all training vectors for many of the parameters used among the experiments, or about one third of the training vectors used in the most memory consuming experiment. Because of its clock frequency and transfer speed from the flash memory, it is considered feasible to temporarily transfer the necessary vectors from the flash memory to the RAM in real time. Furthermore, implementing a real-time memory transfer for the Arduino Due would be

complex due to its limited RAM size. Not to mention that it does not have enough non-volatile memory to store training vectors for the more demanding experiments, consequently, they would need to be transferred through a peripheral.

4.3.3 Analog-to-Digital Converter

The Arduino Due and the STM32F767ZI have built-in ADCs, all of them supporting a resolution of 12 bits. These ADCs are considered to be adequate when it comes to the sampling rate and resolution requirements of commonly used sensors. However, the Raspberry Pi does not feature any built-in ADC and therefore needs an external ADC connected to it. Two external ADCs were looked into, one of them having a resolution of 10 bits and a sampling rate of 200 kS/s. The other one has a resolution of 12 bits and a sampling rate of 3.3 kS/s. Only having a resolution of 10 bits gives a resolution span between 0 and 1023 which is severely limiting compared to the resolution span between 0 and 4095 that a 12 bit ADC can provide. Additionally, only having a sampling rate of 3.3 kS/s is not feasible when sampling sensors that are outputting fast-varying signals. The reason for this is that we need a sampling rate of twice those waves' frequencies due to the Nyquist-Shannon sampling theorem. This results in limiting the ADC to only collecting fast-varying sine waves up to approximately 1650 Hz. Besides this, having an external ADC adds more complexity to the system than a built-in ADC.

4.3.4 Peripherals

All of the microcontrollers feature SPI and UART peripherals. SPI can be good for connecting arbitrary external components and UART is often considered an easy and non-complex way of being able to communicate with PCs. Furthermore, the Arduino Due and the STM32F767ZI have support for CAN which can be useful for potentially future work of the thesis. The STM32F767ZI also features an Ethernet interface which too could be of use for future work.

4.3.5 Power Consumption

Approximating power consumption beforehand is difficult, or even impossible in many cases since there is no way of knowing how much power the PASAD algorithm will draw when deployed on a system. The type and size of the RAM, non-volatile memory and caches (if present) have a big impact on the power consumption. Additionally, having peripherals on top of this makes the estimation of the power consumption even more difficult. As a consequence, the estimated power consumption of 0.9 W, 0.63 W and 0.33 W for the Raspberry Pi, Arduino Due and STM32F767ZI respectively, will not be considered applicable to this comparison. Instead, clock frequency, RAM type and support for built-in ADC will be examined when comparing the power consumption.

To begin with, the Raspberry Pi's clock frequency is considerably higher than the Arduino Due's and STM32F767ZI's clock frequency, resulting in higher power consumption. This is because the power consumption of a integrated circuit is approximately proportional to the clock frequency [36]. Furthermore, SDRAM uses more energy than SRAM, resulting in higher power consumption for the Raspberry Pi. Additionally, the Raspberry Pi does not have any built-in ADC and therefore needs an external ADC which most likely draws more power than a built-in ADC. It also uses an operating system which the other embedded systems do not have, which also adds to the power consumption.

When it comes to the power consumption of the Arduino Due and the STM32F767ZI, their power consumption will probably be somewhat similar. This is however based on the assumption that the Arduino Due does not use any external source from which it transfers the training vectors, since if that was the case, the power consumption of the Arduino Due would most likely be higher.

4.3.6 Choosing the Embedded System

The Raspberry Pi and STM32F767ZI are considered to be similar when it comes to their computational performance since both have high clock frequencies and support the double-precision floating-point format. The Arduino Due on the other hand, would probably be too slow computational-wise to run the algorithm in real time because of its lacking hardware support for the double-precision floating-point format.

Regarding the memory, the Arduino Due does not feature enough RAM nor flash memory in order to run the algorithm. It would be necessary to transfer parts of the training vectors used in the algorithm from an external source and this would not be an option worth pursuing in limited ICS or IoT environments. Nevertheless, the Raspberry Pi has far more RAM than what is needed which makes questionable to view it as a resource-constrained embedded system, which is the scope of this thesis. Consequently, the STM32F767ZI would be the most suitable choice for the algorithm when looking at the memory since it has adequate flash memory to store the training vectors.

As for the ADC support, the Arduino Due and the STM32F767ZI both have built-in ADCs that are considered to be feasible for the work done in this thesis. However, the Raspberry Pi does not have any built-in ADC which adds complexity to the system design since an external ADC is needed to be connected to the board. Additionally, the two external ADCs that were discussed for the Raspberry Pi are both not adequate for the work in this thesis.

All of the embedded systems feature similar support for peripherals such as SPI and UART. Besides this, when considering the power consumption, the Raspberry Pi will probably have the highest power consumption of the embedded systems. The reason for this is its unnecessarily powerful hardware and the use of an operating system. Furthermore, the difference in power consumption of the Arduino Due and the

STM32F767ZI is hard to distinguish.

In conclusion, a resource-constrained yet adequate embedded system is sought, so the choice was the STM32F767ZI. The primary reasons for this is its fast CPU, hardware support for double-precision floating-point format, fast and sufficient memory, built-in ADC and the low power consumption.

4.4 System Design

This section is intended to give a overview of the system design with motivations for the high-level design choices made. It was established in Section 4.3 that the STM32F767ZI microcontroller was the most suitable one for this thesis. As a result, the microcontroller will serve as the basis when dimensioning a system around it. The system consists of a test environment with sensors, the microcontroller running the attack detection algorithm and a PC. An overview of the system can be seen in Figure 4.2.

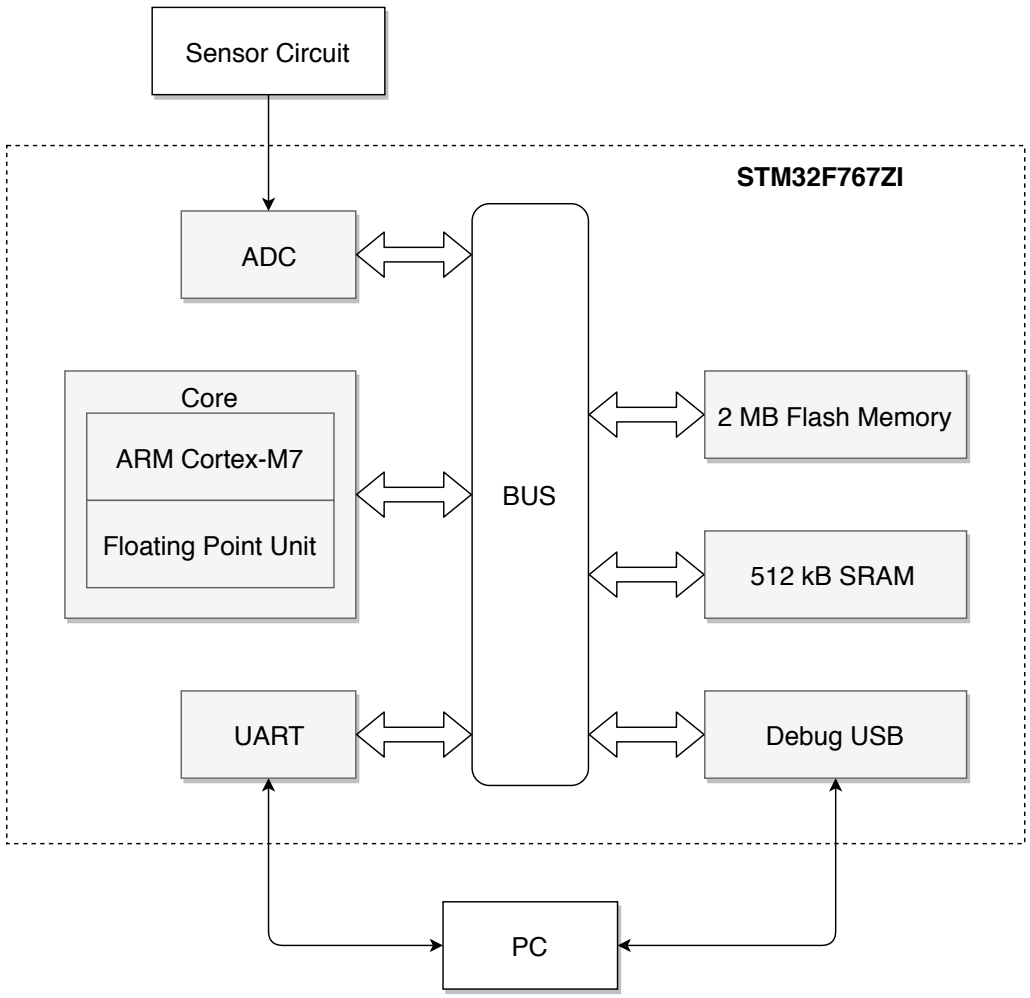


Figure 4.2: System overview where the STM32F767ZI is within the dotted lines.

The system design has been divided into several parts, beginning with the test environment consisting of the choice of sensors and their descriptions presented in Section 4.4.1. Then, the choice of buffering method used for the ADC as well as its details will be presented in Section 4.4.2. Afterwards, presented in Section 4.4.3, there is an explanation of how the data is used for the attack detection algorithm is handled between the RAM and the flash memory. Finally, the choices concerning how the data and the control commands should be transferred between the PC and the microcontroller is presented in Section 4.4.4. The actual implementation of the system is left to be described in Chapter 5.

4.4.1 Test Environment

This section introduces essential parts of the test environment used for evaluating the chosen intrusion detection algorithm. The section begins with motivating what types of sensors to be used in the test environment. When the types of sensors have been determined, details of each type of sensor chosen is expanded upon.

Choice of Sensors

When choosing what types of sensors to implement for the system, one must think ahead of how the choices will affect the practical aspects of evaluating the system. First, for practical reasons it is advisable to make the sensor choices as generic and flexible as possible in order to reduce the number of sensors that need to be implemented. This would allow testing of many different types input signals without using a myriad of different sensors. Second, the behavior of the sensors chosen should be deterministic enough to enable reproducibility of the tests being performed. Third, the sensors should preferably be somewhat related to sensors used in a real ICS where many different noise sources are usually present. The second and the third points are usually contradictory, since the presence of uncontrolled genuine noise sources will reduce the degree of reproducibility. In addition, at least one of the sensors should function in a non-complex way, such as producing a constant input signal to the ADC with low noise levels. This type of sensor can be used to check that the intrusion detection algorithm works for simple input signals before moving on to testing more complex ones.

Three types of sensors were decided to be used, a microphone sensor, a load sensor, and a vibration sensor. The microphone sensor allows controlled tests to be performed while still having some background noise present. Additionally, by placing the microphone in a quiet setting, strict control of noise or disturbances can be achieved. Additionally, complex testing cases can be emulated by playing multiple sound sources to the microphone at the same time. As a result, the microphone sensor allows easy testing of a variety of wave forms with different frequencies or shapes. Furthermore, to create simpler test cases with the microphone sensor, one can ignore the amplitude of the tone played and represent the played frequency as a value which would increase or decrease depending on the tone's frequency.

The rationale for choosing the load sensor is to provide a simple case where the output is constant as long as the load on the sensor is constant. However, even if the load sensor provides a simple case it could technically be used for more complex scenarios by placing a shaking object on top of the sensor. This would have the effect of introducing interference to the sensor depending on how violently the object placed on top shakes.

Lastly, while it is difficult to realistically imitate sensors that are present in industrial environments, the vibration sensor has the purpose of testing scenarios more closely related to a real ICS setting. The vibration sensor can be mounted on machinery in a workshop where lots of noise sources are present. Loading or changing the settings of the machinery being monitored by the vibration sensor can be used to introduce controlled disturbances to the system.

Microphone Sensor

The output of the microphone sensor circuit is twofold depending on application. Either a sine wave from the microphone or a DC voltage corresponding to the frequency of the sine wave output using a frequency-to-voltage converter. This allows two types of testing scenarios for the microphone sensor where the latter one requires less resources. The reason for using less resources in the second scenario comes from the fact that the minimum required sampling rate of the ADC does not depend on the actual frequency of the microphone sine wave. In other words, the circuit transforms the sine wave frequency to a DC voltage resulting in the removal of the sampling rate requirement. However, there is one aspect which still is dependent on the sampling frequency, namely the time it takes to detect changes in the DC voltage level when changing the input frequency. Nevertheless, one should keep in mind that information about the amplitude of the original sine wave is lost when performing this type of frequency-to-voltage conversion.

The rationale for using two types of outputs from the microphone sensor circuit is to widen the input frequency range of the signal without violating the Nyquist-Shannon sampling theorem. Soundwaves in the audible range are oftentimes up to 10 000 Hz which would require a sampling rate of at least 20 000 Hz. Such high sampling rates may or may not be feasible depending on available memory of the device storing the sampled data. Furthermore, common microphone sensors do not support frequencies outside of the audible range.

Load Sensor

While the microphone sensor produces constantly changing output values, the load sensor is more stable and only changes its output when the load changes. This makes the load sensor suitable as a reference case where barely any other factors than what it is loaded with influences the output. Background noise can influence measurements in the case of the microphone sensor and slightly distort the output.

Similarly, vibrations from adjacent machinery when using the vibration sensor could also affect its measurements.

The load sensor uses a strain gauge load cell with a capacity of 10 kg, choosing a relatively low maximum load value allows easy prototyping and testing. The strain gauge load cell changes its resistance value approximately linearly to the load being applied. This change in resistance can be detected and translated to a voltage that the ADC on the microcontroller can interpret.

Vibration Sensor

The output of the vibration sensor is similar to the microphone sensor. It also outputs a sine wave which is subject to sampling rate requirements. This being said, the vibration sensor used in this project is designed for low frequencies up to 40 Hz which severely reduces the sampling requirements compared to sound waves in the audible range.

The vibrations generated from machines such as lathes or fans often vary in intensity and is more difficult to control than, for example, the volume of a speaker. Consequently, to allow measuring different vibration scenarios with the sensor it is recommended to have a variable gain in the vibration sensor circuit. The reason for this is to match the signal's output amplitude of the sensor circuit to the full-scale voltage range of the ADC as explained in Section 3.3.

4.4.2 Buffering

The PASAD algorithm requires L number of sensor values and since the system uses an ADC to sample from a sensor, a buffer for the values sampled by the ADC is to be preferred. Three different types of buffers were considered, the first is shown in Figure 4.3, the second is shown in Figure 4.4 and the third is shown in Figure 4.5.

The first buffer type, shown in Figure 4.3, assumes that the ADC is not running concurrently along with the algorithm. Because of this, it first samples L number of values, then runs the algorithm and so on. The key idea of this type is simplicity, but it delays the run of the algorithm significantly since it has to wait for the values. It would also miss potentially important samples during the run of the algorithm since the sampling is not performed during that time. Another advantage apart from the simplicity is that this buffer could be used in cases where the algorithm and the sampling are not running all the time, e.g., to utilize less power consumption.

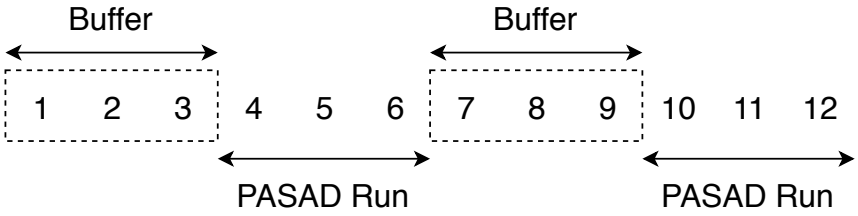


Figure 4.3: An example of the first potential buffer choice.

The second buffer type, shown in Figure 4.4, assumes that the run of the PASAD algorithm is slow compared to the sampling rate. By only putting half, or less, of the sampled values in a buffer, one would get an average of the sensor output. This buffer type could be useful in cases where fast-varying signals are not sampled, for instance in cases where a sensor is outputting a DC voltage.

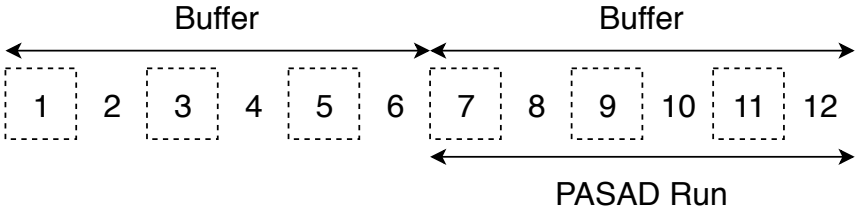


Figure 4.4: An example of the second potential buffer choice.

The third buffer type, shown in Figure 4.5, is a buffer where the L latest values are always stored. The advantage of this type is that PASAD always will have the latest values when it is run. However, the disadvantage is that if the L parameter is small, and PASAD is running too slow, we could lose values that never get used in the PASAD algorithm. The reason for this is that the buffer could potentially be refilled several times during the run of the algorithm when these criteria are fulfilled.

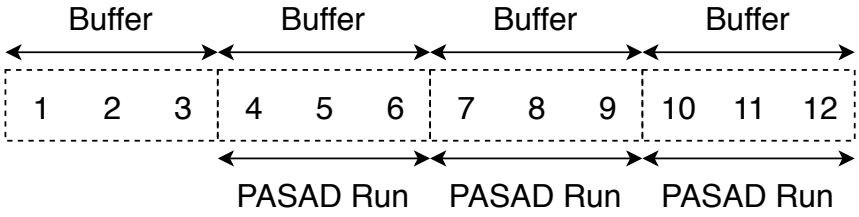


Figure 4.5: An example of the third potential buffer choice.

Choice of Buffer Type

The third buffer type was chosen because the buffer will always contain the latest sampled values possible, each time when PASAD is run. Since the system is to be run in real time, having the latest sampled values when the detection algorithm is run is preferable since an anomaly could be detected faster. Other reasons are that in cases where sine waves are sampled, a constraint is put on the sampling rate which may not be fulfilled by for example the second buffer type. The is due to the down sampling nature of the second buffer type. Additionally, the first buffer type assumes that the ADC and the PASAD algorithm are not running concurrently, which would be a waste of resource utilization.

Since the PASAD algorithm and the ADC run concurrently, it was decided that PASAD should copy the content of the buffer into its x vector at each run of the algorithm. If this was not the case, some of the lag values (sampled values) would be replaced during the run of the algorithm, resulting in erroneous calculations. Additionally, since PASAD requires the lag values to be used in order where the oldest value is first and the newest value is last, a mapping function between the buffer and the x vector is needed.

The mapping between the buffer and the x vector can be done simply by storing a pointer to the last written value. Then use the pointer as a marker, since we know that the value of the pointer plus one will be the oldest value which should lie first in the x vector. A modulo instruction is needed as well so that the pointer does not go outside the allocated memory space. An illustration of the mapping function can be seen in Figure 4.6.

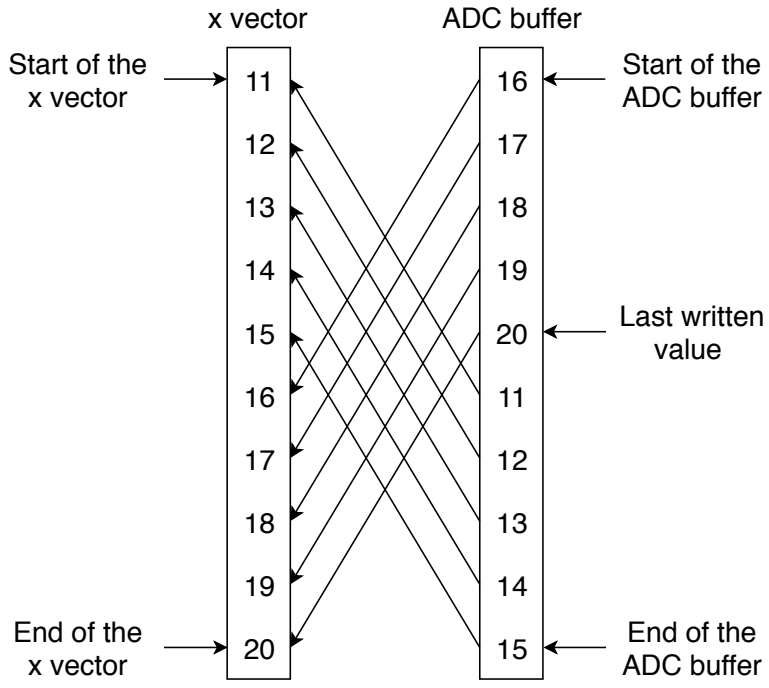


Figure 4.6: How the mapping between the buffer and the x vector is done.

4.4.3 Memory Management

The RAM size of the STM32F7676ZI is too small to hold all necessary training vectors needed for the PASAD algorithm when large parameters are used. However, the flash memory is sufficient to store all training vectors needed. Because of this, the flash memory will be loaded with all training vectors once, and the RAM will have to load vectors from the flash memory during the run of the algorithm. A model of the memory structure can be seen in Figure 4.7.

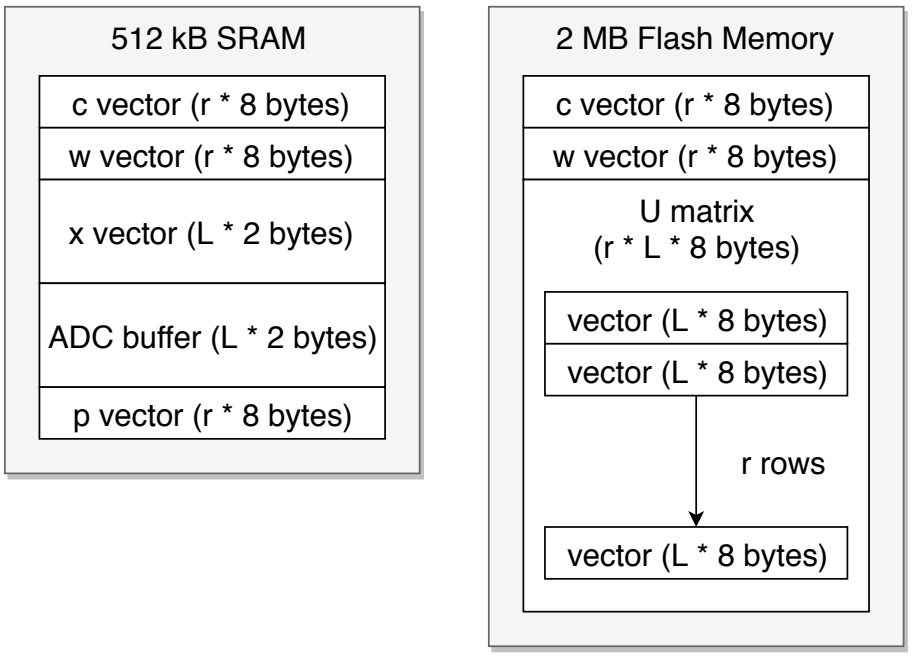


Figure 4.7: A model of the memory structure of the SRAM and the flash memory for PASAD.

As previously described in Section 4.4.2, the RAM already needs to have space for one ADC buffer and one x vector which each contain the L latest sampled values. Since both only use 16 bit integers, they together use $2 \cdot L \cdot 2$ bytes, which in the case of L being 5000, results in 20 kB of space. Since this memory usage is considered to be small and because the values are constantly changing, it was decided that they should lie in the RAM.

Furthermore, the c vector and the w vector use $2 \cdot r \cdot 8$ bytes, which in the case of r being 43, results in 688 B of utilized space. Because of this, it was decided that these two vectors should be loaded into the RAM before the algorithm is run and not be replaced during the run of the algorithm. Besides this, it was decided that the p vector occupying $r \cdot 8$ bytes should be in the RAM for holding temporary calculations since the execution time would otherwise suffer greatly.

4.4.4 Interfacing the Microcontroller with the PC

The STM32F767ZI microcontroller features a debug USB by default that can be used to flash the code of the program and potentially needed extra data on the flash memory from the PC. However, since it is a debug USB, it cannot be used to transfer any non debug data and commands between the microcontroller and the PC in real time. In order to send non debug data and commands between them in real time, an additional interface was sought. The choice of interface was between using UART or a client USB. The choice became UART because it was thought that a client USB would bring extra complexity to the system in the sense of programming it, but also the requirements of USB drivers for different operating systems. Furthermore, UART is often supported by many systems and could potentially be used to control the microcontroller from a simpler machine than from the PC. The asynchronous serial communication between the microcontroller and the PC was set to have a speed of 115 200 baud per seconds since this was thought to be the highest stable speed that could be used. This speed corresponds to a data rate of 11 520 bytes per seconds.

In order to verify the functionality of the PASAD algorithm when running on the microcontroller, test data sent in real time was thought necessary. When using the test data for the algorithm running on the microcontroller, the results could be compared to the same test data when used on the verified version of the algorithm running on the PC. Besides being able to transfer test data and commands, it was also thought necessary to have the option of transferring the departure score and the execution time of the algorithm running on the microcontroller, to the PC.

With the intention of facilitating all the needs of the microcontroller such as preparing the training vectors, sending commands and more, a program for the PC was developed. The program running on the PC was programmed to have the four features listed below.

1. Send a command to the microcontroller that it should start collecting values that can be used in the training phase.
2. Create a .csv file from the binary file extracted from the flash memory that contains the collected values to be used in the training phase.
3. Create a binary file from training vectors that can be used when flashing the microcontroller's flash memory.
4. Send a command to the microcontroller that it should start running the PASAD algorithm. Additional options were also added, such as sending test data to the microcontroller instead of using the ADC, and to send back the results to the PC.

5

Implementation

This chapter explains in detail how the top level design decisions laid out in Chapter 4 are realized. The first part includes the supporting sensor circuitry needed to enable the measurement of analog signals and how to prepare them for the microcontroller. The second part of the chapter elaborates on how the implementation of the intrusion detection system was accomplished, such as the software implementation of the intrusion detection algorithm on the microcontroller.

5.1 External Circuitry

As mentioned in Chapter 4, the only interfacing that needs to be physically implemented is between the analog sensors and the ADC. This section presents the following three types of sensors: a microphone sensor, a load sensor and a vibration sensor. Moreover, the external circuitry needed between the sensors and the ADC to function correctly are also presented.

5.1.1 Interfacing the Microphone with the ADC

The first sensor consists of an electret microphone, namely an FC-109 MAX9812 Microphone Amplifier Module, which outputs both positive and negative voltages approximately in the range $\pm 35\,\text{mV}$ at sound pressure levels of ($\approx 60\,\text{dB}$).

Since the analog-to-digital converter is designed to accurately convert voltages within $0\,\text{V}$ and $3.3\,\text{V}$, the interfacing circuit should raise the negative voltages above $0\,\text{V}$, which is performed by using a unipolar to bipolar conversion. Furthermore, using the entire available range of the ADC is recommended in order to use as many bits as possible for the quantized signal, e.g., using only half the range would waste half the resolution of the ADC. Consequently, the interfacing circuit should also amplify the signal level to span the entire available voltage range $0\,\text{V}$ to $3.3\,\text{V}$. Both of the operations can be performed by using operational amplifiers in combination with a few resistors. Raising the voltage can be done by using a summing circuit and amplifying the signal can be performed using a non-inverting operational amplifier where the values of the supporting resistors connected to the operational amplifier

determine the gain.

After managing to appropriately transform the signal, the signal is filtered by a simple resistor-capacitor circuit. Since the signals of interest are located in the human hearing range i.e., $< 20\,\text{kHz}$, signals with frequencies outside of this range should be attenuated in order to reduce noise. After the filter there is only need for one more stage before the signal is ready for the ADC. The final stage is to make sure that the input voltages actually stay reasonably within the $0\,\text{V}$ to $3.3\,\text{V}$ range. Since the output of the microphone depends on the input sound level, sounds that are well above the conversation sound level will result in signals with higher amplitudes than what the system is designed for. In order to make sure this does not happen, a pair of Schottky diodes are placed after the filter connected to the $3.3\,\text{V}$ source and ground respectively. Schottky diodes are useful in the application of overvoltage protection, this is due to their lower forward breakdown voltage when compared to other diodes. The forward breakdown voltage determines the voltage difference across the diode before it starts becoming electrically conductive. As a result, the diode will be faster at detecting that the voltage is too high and mitigate it.

With a working microphone circuit, the output of this circuit is ready to be sent to the ADC. However, the microphone circuit has also been extended by introducing another circuit that converts the frequency of the output signal to a constant voltage which can be fed to the ADC. This allows the microphone circuit to work in two modes, either by sending varying signals to the ADC that takes into account both its amplitude and frequency, or by sending a DC signal to the ADC where the amplitude is omitted.

This section is divided into four parts, first implementation of the the bipolar to unipolar conversion is presented. Then, the amplification stage is introduced followed by the circuit which provides the filtering and overvoltage protection. Finally, the frequency measurement circuit is described.

Bipolar to Unipolar Conversion

As mentioned previously, the bipolar to unipolar conversion can be performed by using a voltage summing circuit. Since the output of the microphone sensor is bipolar it will vary around $0\,\text{V}$. When trying to convert the output to be between $0\,\text{V}$ and $3.3\,\text{V}$ the voltage needs to be raised to $1.65\,\text{V}$. Using a voltage summing circuit where the resistor values and the source voltage have the effect of adding $1.65\,\text{V}$ as explained in Section 3.3.2. The circuit that adds the $1.65\,\text{V}$ is shown in Figure 5.1.

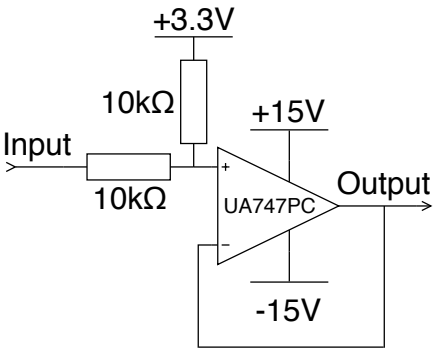


Figure 5.1: Bipolar to unipolar conversion circuit.

In the special case of the resistances in the summing circuit being equal to one another, Equation 3.7 from Section 3.3.2 results in Equation 5.1. Since the voltage $U_{\text{Bias}} = 3.3\,\text{V}$, the added voltage will be the desired 1.65 V.

$$U_{\text{Output}} = U_{\text{Bias}} \cdot \frac{1}{2} + U_{\text{Input}} \cdot \frac{1}{2} = 1.65\,\text{V} + U_{\text{Input}} \cdot \frac{1}{2} \tag{5.1}$$

where:

U_{Output} = Output of the circuit [V]
U_{Bias} = Bias voltage [V]

Amplifying the Signal

With a working bipolar to unipolar conversion circuit, it is highly recommended to amplify its output voltage to cover the entire range of the ADC between 0 V and 3.3 V. If this is not done, only a small part of the ADC's resolution is being used as explained in Section 3.3. The sound played to the microphone can be of different sound pressure levels depending on how loudly a sound is played. Therefore, it is not possible to choose a gain value which works for all different sound pressure levels. To resolve this, one of the resistors determining the gain of the amplifier is tunable, thus increasing the range of sound pressure levels that cover the range of the ADC. Using Equation 3.6 from Section 3.3.1, the tunable resistor chosen allows the circuit to have a minimum voltage gain A_{min} as shown in Equation 5.2 and a maximum voltage gain A_{max} as shown in Equation 5.3. The circuit that implements the non-inverting amplification is shown in Figure 5.2.

$$A_{\text{min}} = 1 + \frac{10\,\text{k}\Omega}{1\,\text{k}\Omega} = 11 \tag{5.2}$$

$$A_{\text{max}} = 1 + \frac{110\,\text{k}\Omega}{1\,\text{k}\Omega} = 111 \tag{5.3}$$

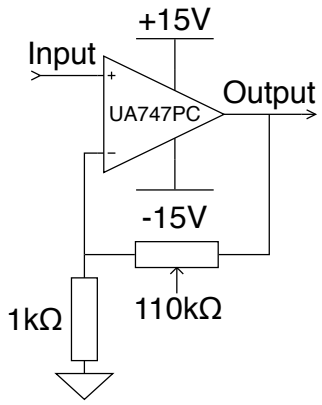


Figure 5.2: Non-inverting operational amplifier circuit with a tunable gain in the range between 1.1 to 11.1.

Filtering and Overvoltage Protection

The Resistor-Capacitor (RC) filter consists of a 8.2 kΩ and a 1 nF capacitor followed by a resistor and two Schottky diodes as shown in Figure 5.3. The Schottky diodes keep the voltage level at the output node (ADC) to the interval $[0-V_{\mathrm{FM}},\ 3.3+V_{\mathrm{FM}}]$ V where V_{FM} is the forward breakdown voltage. If the voltage at the output node exceeds the $3.3 + V_{\mathrm{FM}}$ V, current will flow from the output node to the source resulting in limiting the voltage at the output node. Similarly, a sufficient negative voltage at the output node would allow the Schottky diode connected to ground to allow current through to keep the voltage stabilized. To limit the current draw, it is common to place a resistor before the diodes. In this case it is not needed since the resistor used in the RC filter already limits the current adequately.

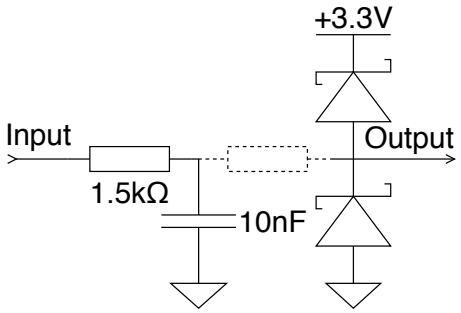


Figure 5.3: Lowpass filter followed by overvoltage protection diodes.

Combining the Stages

Putting all the stages together results in the complete circuit needed to convert the output of the microphone to the input of the ADC. The full schematic is shown in Figure 5.4.

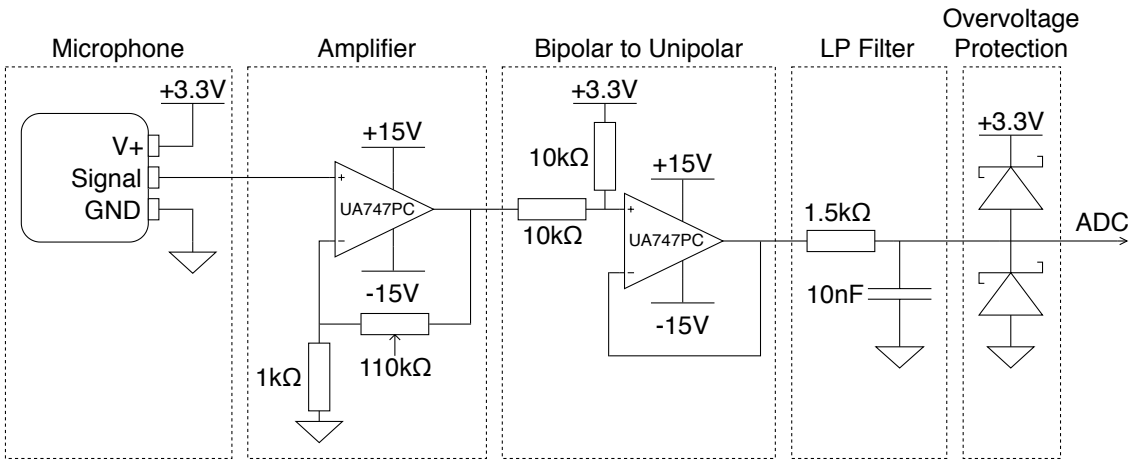


Figure 5.4: The combined stages form the path from microphone to ADC.

Frequency Measurement Circuit

The completed microphone circuit is extended with a frequency-to-voltage converter circuit that outputs a constant voltage at a specific input frequency. The shape of the input signal could have an arbitrary shape as long as it varies sufficiently in amplitude. As such, it is simple to connect the frequency-to-voltage converter's input to the output of the microphone circuit to enable frequency measurements.

The frequency-to-voltage converter circuit is shown in Figure 5.5. Similarly to the microphone circuit there is an amplifier stage and an overvoltage protection stage. However, the amplifier stage provides a tunable gain between 2 and 102 to provide better resolution at lower frequencies. The reason for achieving a better resolution at low frequencies is that the frequency-to-voltage converter's output is small at low frequencies, and large at high frequencies. If one only wants to measure low frequencies, increasing the gain is recommended. This has the effect of increasing the output voltage at the lower frequencies to cover the entire range of the ADC between 0 V and 3.3 V. The actual frequency-to-voltage component is an LM2907 from Texas Instruments [34].

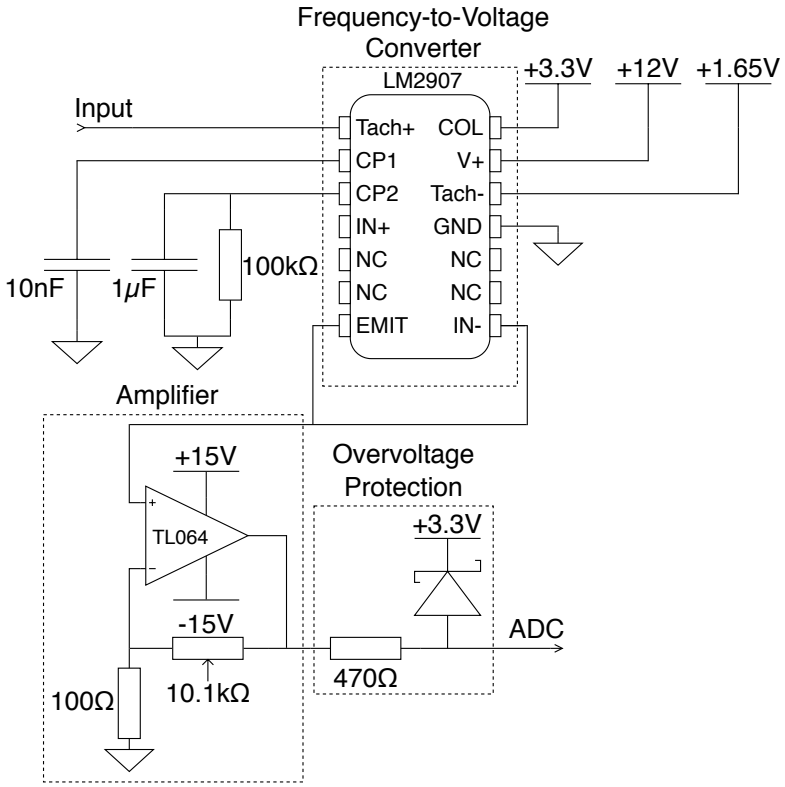


Figure 5.5: Frequency measurement circuit from the output of the microphone circuit to ADC.

5.1.2 Interfacing the Load Sensor with the ADC

The second sensor used consists of a load cell fastened between two plates. The two plates function as the surface where weights can be placed, i.e., applying a force to the load cell. A load cell is a weight measurement device consisting of a metal beam that bends when force is applied to it. Additionally, the load cell has a strain gauge that is connected onto the metal beam of the load cell which will bend along with the metal beam when force is applied to it. When the strain gauge is bent, the resistance of the strain gauge is also changed. This change in resistance can be measured in order to derive how much force has been applied on the load cell. However, the change in resistance is oftentimes extremely small which makes it difficult to measure. To allow such small changes in resistance, the strain gauge is connected to one of the legs in a Wheatstone bridge circuit which allows exceptionally precise measurement of the strain gauge's resistance [13].

Even though the Wheatstone bridge allows the detection of small changes in resistance, the resulting voltage from the Wheatstone bridge will still be extremely small and needs to be amplified [26]. An instrumentation amplifier can be used for this task since it has a high input impedance, thus limiting the effect it has on the input circuit. The instrumentation amplifier used to interface the output of the load cell is the INA217 which has an input impedance of 60 MΩ. The gain of the INA217

is determined by the size of the resistor R_G placed between the instrumentation amplifier's inputs $G+$ and $G-$. According to the datasheet, the gain of the INA217 is given by Equation 5.4 where the result of $R_G = 10\,\Omega$ is also shown. In addition, the load sensor circuit is shown in Figure 5.6.

$$G = 1 + \frac{10\,000\,\Omega}{R_G} = 1 + \frac{10\,000\,\Omega}{10\,\Omega} = 1001 \tag{5.4}$$

where:

G = Gain [-]
R_G = Resistance [Ω]

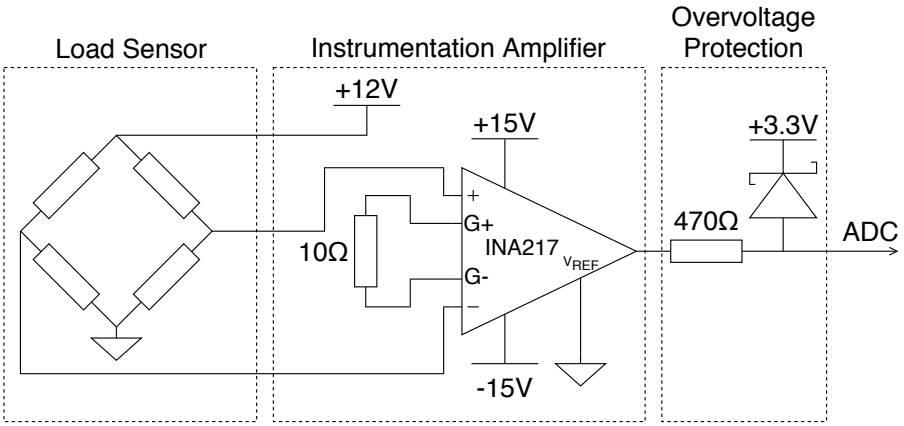


Figure 5.6: Load sensor circuit from sensor to ADC.

5.1.3 Interfacing the Vibration Sensor with the ADC

The third sensor to be implemented is the vibration sensor. This sensor is a capacitive piezoelectric sensor called the MiniSense 100. The Minisense 100 can detect vibrations up to 40 Hz with a sensitivity of 1.1 V/g, where g is the gravitational acceleration. The output of the sensor ranges between ±90 V depending on how powerful the vibrations are. However, providing vibrations that actually produce this high voltage requires an immense acceleration of over 80 g in either direction. As such, the output is usually rather small and needs amplification. However, the amplification could result in the voltage exceeding the intended range. To alleviate this, additional overvoltage protection diodes are placed at the sensor output. Moreover, the capacitive sensor creates a high-pass filter with the resistance of the circuit between the sensor and the ADC as explained in Section 3.3.4. Since the capacitance of the MiniSense 100 is 244 pF, the external resistance needs to be sufficiently large to avoid attenuating the low frequency signals. Using Equation 3.12 with $R_1 = 10\,\text{M}\Omega$, $R_2 = 100\,\text{k}\Omega$ and $R_3 = 1.8\,\text{k}\Omega$, the effective resistance seen by

the input signal is shown in Equation 5.5.

$$R_e = \frac{R_1}{R_3} \cdot (R_2 + R_3) = \frac{10\,\mathrm{M\Omega}}{1.8\,\mathrm{k\Omega}} \cdot (100\,\mathrm{k\Omega} + 1.8\,\mathrm{k\Omega}) \approx 566\,\mathrm{M\Omega} \tag{5.5}$$

The 244 pF capacitance together with the apparent 566 MΩ resistance results in a high-pass filter with a cutoff frequency shown in Equation 5.6.

$$f_c = \frac{1}{2\pi \cdot R_1 \cdot C_1} = \frac{1}{2\pi \cdot 566\,\mathrm{M\Omega} \cdot 244\,\mathrm{pF}} \approx 1.15\,\mathrm{Hz} \tag{5.6}$$

As mentioned in Section 3.3.4, the input impedance of the operational amplifier used in the bootstrap circuit needs to be a lot higher than the apparent impedance from the bootstrap circuit. Consequently, the operational amplifier TL064 was chosen since it has an input impedance of 1 TΩ. The realization of the bootstrap circuit is shown in Figure 5.7.

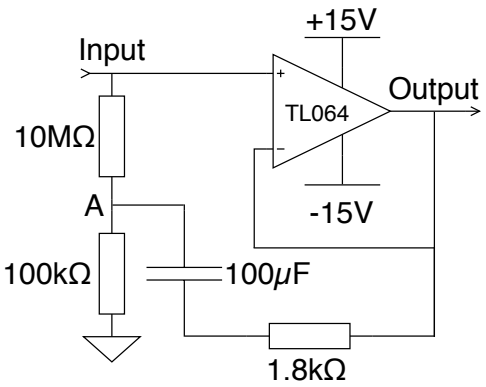


Figure 5.7: Bootstrap circuit providing high apparent impedance.

When taking measurements with the vibration sensor it is often a practical necessity to use wires to extend the physical range of the sensor. The wires will introduce some capacitance, and as a result slightly change the characteristics of the high-pass filter formed from the sensor capacitance and the bootstrap circuit. This can give rise to peaks in amplitude at the lowest frequencies below 1 Hz, to remedy this, a high-pass filter is placed after the bootstrap circuit with a cutoff frequency of approximately 0.72 Hz. Following the high-pass filter is an amplification stage, a bipolar to unipolar conversion circuit, a low-pass filter and overvoltage protection for the ADC similar to the microphone circuit. The low-pass filter is designed to have a cutoff frequency of approximately 34 Hz which is within the specified upper limit of 40 Hz according to the vibration sensor specification. The complete circuit is shown in Figure 5.8.

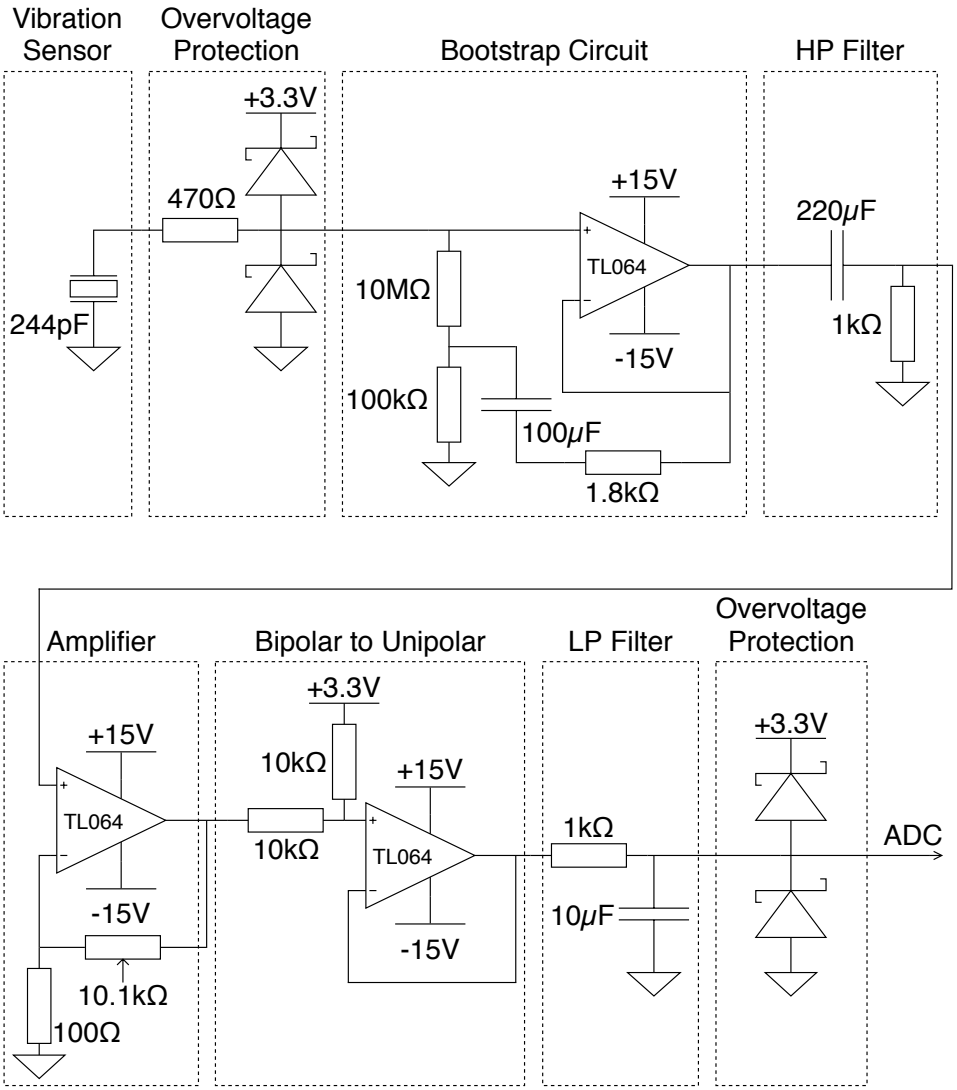


Figure 5.8: Vibration sensor circuit from sensor to ADC.

5.2 Implementing the Intrusion Detection System

This section describes how the implementation of the PASAD algorithm on the STM32F767ZI was performed. The section begins by explaining how the ADC was set up in order to have a more accurate sampling rate. Then, the training phase and detection phase are described, which both use the ADC to collect values. The collection of values by the ADC is performed using interrupts, i.e., running concurrently together with other software.

5.2.1 Timer-based Analog-to-Digital Converter

The ADC on the STM32F767ZI is rather limited when it comes to setting the desired sampling rate on it. Not only are there many parameters that can be changed, it would also be difficult to achieve a sampling rate close to the desired sampling rate because of the non-linear correlation between the sampling rate and the timer period. In order to have a more accurate sampling rate that also is easier to adjust, a timer peripheral is used that decides when the ADC should sample. The timer peripheral is running at 54 MHz, and the adjustment of the sampling rate works by setting the timer period to a specific value that can be derived from the formula in Equation 5.7. The timer works by counting up at each clock tick until the counter reaches the specified timer period. When reached, it sends a signal that instructs the ADC to sample one value.

$$T = \frac{54000000}{S} - 1 \tag{5.7}$$

where:

T = Timer period [s]
S = Sampling rate [Hz]

The division used in Equation 5.7 makes the formula have a non-linear correlation between the sampling rate and the timer period. However, it is still considered to be very precise under commonly used sampling rates. How to calculate the actual sampling rate is shown in Equation 5.8. To show its preciseness, when using a sampling rate of 33, the actual sampling rate is 33.000 013. When using a sampling rate of 7000, the actual sampling rate is 7000.26.

$$S_{\text{ASR}} = \frac{54000000}{\left\lfloor \frac{54000000}{S} \right\rfloor} \tag{5.8}$$

where:

S_{ASR} = Actual sampling rate [Hz]
S = Sampling rate [Hz]

5.2.2 Training Phase

Most parts of the training phase are done on the PC, where a MATLAB script is used to derive the training vectors from a given time series. The reason for not implementing this function on the microcontroller, is that there is not any reason for doing so since it is an offline procedure. In addition, the training phase is demanding in terms of computational performance and memory as well as requiring a subjective assessment when selecting the r parameter. This is done by plotting

the sorted eigenvalues of the eigenvectors, which then can be used to select the r parameter. Consequently, the training phase would be difficult to perform on a microcontroller.

In order to provide time series for the training phase, a small function was written for the microcontroller that collects values using the ADC. Additionally, it stores these values onto the flash memory which then can be read from by the PC. Since the test environment consists of different sensors, each with a unique sampling requirement, the property of changing sampling rate and number of samples to collect was implemented in the function. The function itself consists of setting up the ADC to run continuously at the provided sampling rate until the specified amount of samples are collected.

5.2.3 Detection Phase

The detection phase of the PASAD algorithm has been divided into two parts. The first part is where the initialization of the algorithm is performed once, i.e., fetching data from the flash memory to the RAM and precomputing addresses. The second part is where the departure score calculation of the algorithm is performed.

Initialization of the PASAD Algorithm

The initialization phase of the PASAD algorithm is presented as pseudocode in Algorithm 1 and its source code can be seen in Appendix A. During the initialization phase, the r and L parameters are loaded into the RAM from the flash memory (called Mem) as well as the performing the calculations of the addresses of the training vectors. After that, the c vector and the w vector are loaded to the RAM from the flash memory by using the previously calculated addresses. Additionally, the w vector is element-wise multiplied by itself in order to precompute as much as possible.

Departure Score Calculation of the PASAD Algorithm

The departure score calculation of the PASAD algorithm is presented as pseudocode in Algorithm 2 and its source code can be seen in Appendix B. It begins by storing the current to-be-written position of the ADC buffer in a variable j. Since this variable contains the to-be-written position, we know that it holds the currently oldest sampled value, which then is put first in the x buffer. Then, the rest of the ADC buffer is copied to the x buffer in sorted order. These operations can be seen at lines 1 to 6.

After the copying of the ADC buffer is done, the p vector, which will hold the projection of x onto the signal subspace $\mathcal{L}^r$, is cleared. This is performed at lines 8 to 10.

Algorithm 1 Initialization of the PASAD algorithm.

1: $r \leftarrow \text{Mem}[Base_{\text{addr}}]$
2: $L \leftarrow \text{Mem}[Base_{\text{addr}} + 4]$
3:
4: $c_{\text{addr}} \leftarrow Base_{\text{addr}} + 16$
5: $w_{\text{addr}} \leftarrow c_{\text{addr}} + r \cdot 8$
6: $U^{\text{T}}_{\text{addr}} \leftarrow w_{\text{addr}} + r \cdot 8$
7:
8: **for** $i = 0$ to $r - 1$ **do**
9: $\quad c_i \leftarrow \text{Mem}[c_{\text{addr}} + i \cdot 8]$
10: **end for**
11:
12: **for** $i = 0$ to $r - 1$ **do**
13: $\quad w_i \leftarrow \text{Mem}[w_{\text{addr}} + i \cdot 8]$
14: $\quad w_i \leftarrow w_i \cdot w_i$
15: **end for**

Algorithm 2 Departure score calculation of the PASAD algorithm.

1: $j \leftarrow Buffer_{\text{current}}$
2:
3: **for** $i = 0$ to $L - 1$ **do**
4: $\quad x_i \leftarrow Buffer[j]$
5: $\quad j \leftarrow (j + 1) \mod L$
6: **end for**
7:
8: **for** $i = 0$ to $r - 1$ **do**
9: $\quad p_i \leftarrow 0$
10: **end for**
11:
12: $DepartureScore \leftarrow 0$
13: $Row_{addr} \leftarrow U^{\text{T}}_{\text{addr}}$
14:
15: **for** $i = 0$ to $r - 1$ **do**
16: $\quad$ **for** $j = 0$ to $L - 1$ **do**
17: $\quad\quad p_i \leftarrow p_i + \text{Mem}[Row_{\text{addr}}] \cdot x_j$
18: $\quad\quad Row_{\text{addr}} \leftarrow Row_{\text{addr}} + 8$
19: $\quad$ **end for**
20:
21: $\quad y \leftarrow c_i - p_i$
22: $\quad DepartureScore \leftarrow DepartureScore + w_i \cdot y \cdot y$
23: **end for**
24:
25: **return** $<DepartureScore>$

Then, the departure score is cleared and the base address of the U^T matrix is stored as shown at lines 12 and 13 respectively.

After this, the departure score calculation of the PASAD algorithm is computed at lines 15 to 23. The outer loop is based on the r parameter, and the reason for this is that it can perform the necessary calculations for each row in the Equations 3.2 and 3.3 in Section 3.2.2. There is also an inner loop, based on the L parameter, which calculates the p vector holding the projected vector. After that, at line 21, the projected vector is compared to the centroid vector. The result of this comparison is squared and multiplied with the corresponding element from the weight distribution vector, in addition to being added to the departure score. This is performed at line 22 and corresponds to the operations from Equation 3.4 in Section 3.2.2. Finally, at line 25, the departure score is returned.

6

Evaluation

This chapter contains the results and methodology of the evaluation of the system. This includes verifying that the output of the algorithm is the same when running the intrusion detection algorithm on the embedded system as compared to running it on the reference PC. In addition, the execution time and detection capabilities are evaluated when applying a variety of attacks on the sensors. Moreover, an evaluation of the resources used is carried out. Finally, the impact of changing specific system parameters is estimated with respect to detection capabilities and resource utilization.

6.1 Methodology of Evaluation

Evaluation of the system is performed with respect to the metrics of interest, more specifically the execution time, memory usage, detection accuracy and power draw. The choice of these specific metrics was done with respect to common metrics used when evaluating embedded systems [25]. The parameters L and r were chosen as the parameters to be modified during the evaluation. These parameters are directly associated with the detection algorithm in which they decide the sizes of the training vectors as explained in Section 3.2.

When evaluating the system, it is of interest to provide a broad evaluation of the system while at the same time avoiding an exhaustive evaluation of all possible parameter combinations. For example, performing an exhaustive evaluation of r values between 3 and 43, and L values between 15 and 5000, would require testing $(43 - 3 + 1) \cdot (5000 - 15 + 1) = 204\,426$ different combinations. The choice of these specific L and r parameters in the example is to illustrate the number of combinations when using the most extreme values in the paper by Aoudi et al. [2].

To reduce the number of combinations to be tested, a subset of parameters were chosen, the values for the r parameter were set to 20 and 40, and the values for the L parameter were set to 1000, 3000 and 5000. In one experiment, where the execution time is evaluated against both L and r, the r parameter was set to 10, 20, 30 and 40 while the L parameter was set to 1000, 2000, 3000, 4000 and 5000. This exception was done to provide a more clear visualization of the execution times. The reason for choosing these exact values for the L and r parameters was to capture a

broad combination of the two.

When the subset of the L and r combinations has been chosen, the types of data sets to use need to be determined. The types of data sets chosen should represent a wide variety of signal types and attacks in order to provide a broad evaluation of the PASAD algorithm. Only using data sets with similar signal characteristics increases the risk of missing edge cases when performing the evaluation. Consequently, the evaluation uses five data sets: *DA1*, *DA2*, *SA1*, *SA2* and *SA3*. The five data sets consist of a number of different value ranges, as shown in Table 6.1. These data sets provide two types of direct attacks and three types of stealthy attacks, and have been used for evaluating the PASAD algorithm in the original paper by Aoudi et al. [2]. Picking the same data sets that were used in the original paper enables our evaluation to be compared to it, effectively providing a point of reference.

Table 6.1: Data sets used for evaluating the PASAD algorithm on the embedded system.

Experiment	Minimum value	Maximum value
DA1	40.675	66.896
DA2	31.467	32.968
SA1	31.405	32.953
SA2	0.16970	0.23889
SA3	122.86	122.94

6.2 Confirming Functional Correctness

Two systems are used when determining the functional correctness of the algorithm, the PC and the microcontroller. The purpose of using two systems is to provide a comparison between a highly resourceful system (the PC) and a resource-constrained system (the microcontroller). Additionally, the PC runs a version of the algorithm that has been confirmed to work. As a result, the PC version will function as a reference case for the comparison. Since the PC has much more resources in terms of memory and computational performance than the microcontroller, it is more suited to perform the training phase. This is mainly because the training phase of an anomaly detection algorithm usually performance requirements that are too high to be run on a microcontroller.

It was decided that the training phase running on the PC should use a subset of a known time series as input data. These time series consist of the five data sets presented in Table 6.1. The reason for choosing these data sets for determining the functional correctness is that they provide a wide range of signal types. In order to create a deterministic behavior when testing the functional correctness, the training vectors used by both the PC and the microcontroller need to be exactly the same. In addition, the input data during the detection phase also need to be

the same. As mentioned in Section 4.4.4, a function for receiving test data on the microcontroller was implemented, which makes it possible for both systems to use identical test data. With this in mind, the functional correctness of the algorithm consists of comparing the detection phases' mean departure scores on each of the systems. This comparison is presented in Table 6.2.

Table 6.2: Mean difference in departure score when the PASAD algorithm is run on the STM32F767ZI compared to the PC.

Experiment	**Mean departure score difference [%]**
DA1	$3.7462 \cdot 10^{-13}$
DA2	$1.0472 \cdot 10^{-12}$
SA1	$1.2686 \cdot 10^{-12}$
SA2	$9.9789 \cdot 10^{-14}$
SA3	$1.7572 \cdot 10^{-11}$

6.3 Execution Time

The execution time of the PASAD algorithm, when running on the STM32F767ZI, was measured by taking the CPU clock difference between the start and the end of one pass of the algorithm with the initialization step omitted. By doing this, the time for printing or sending results get left out from the measurement results. The reason for omitting the initialization step is that it does not have anything to do with the actual execution time of the algorithm. However, since the analog-to-digital converter used in the system, is interrupt-based, i.e., it runs concurrently with the algorithm, the sampling rate could potentially affect the execution time of the algorithm. To estimate to which extent it affects the execution time, different sampling rates were tested. In order to make the execution times as discernible as possible, large parameters were used which result in longer execution times. The parameters used were an L of 5000 and an r of 43, and the resulting execution times can be seen in Table 6.3.

Table 6.3: Execution times depending on the sampling rate when using an L of 5000 and an r of 43.

Sampling rate [Hz]	**Execution time [ms]**
500	22
1000	22
5000	23
10000	23

As seen in Table 6.3, the sampling rate did not affect the execution time to a high degree. Because of this, further evaluation concerning the ADC's influence was not performed. Nevertheless, evaluation of the execution time with different parameters used was performed and can be seen in Figure 6.1.

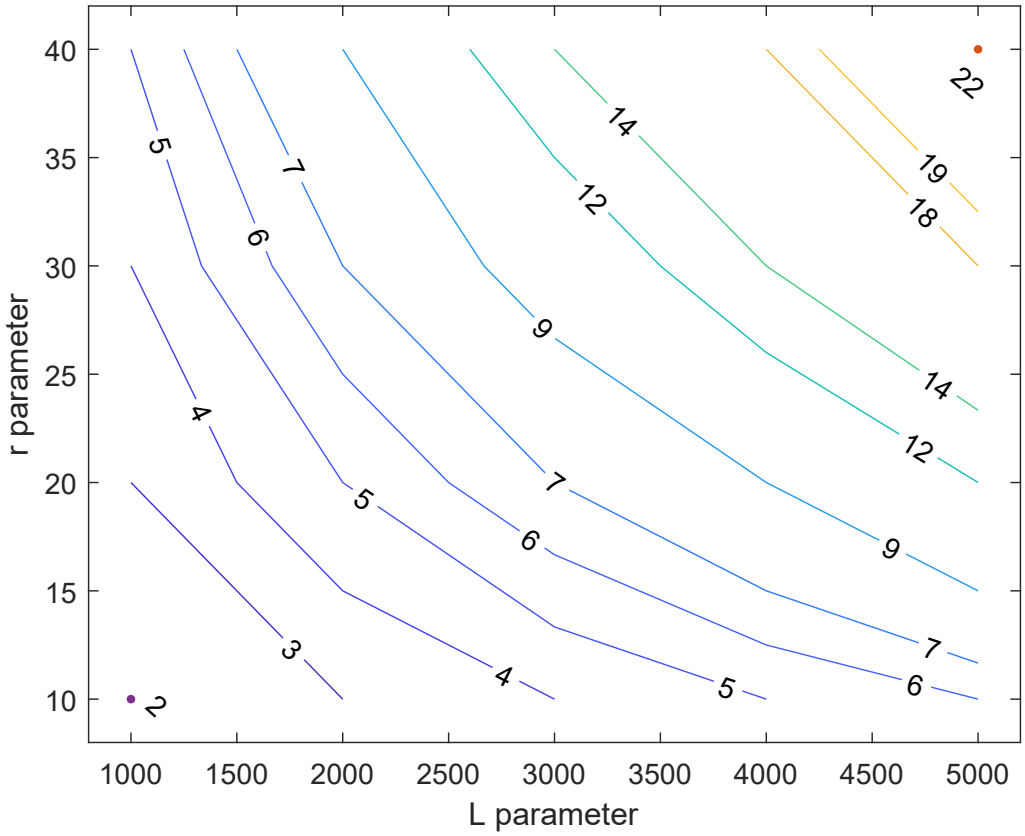


Figure 6.1: Contour map of execution times in milliseconds depending on the L and r parameters.

6.4 Attack Detection

This section describes the results and findings when attacks are performed on the sensors. It has been divided into subsections consisting of the microphone, vibration and load sensor. Several L and r parameters were tested in this section, however, only the combinations of the parameters that provided the best results are included. The reason for this is that each test case has its own optimal configuration for detecting the attacks. Finding the optimal configurations of L and r is still an open research question.

6.4.1 Using the Microphone Sensor

When using the microphone, one needs to have as quiet testing environment as possible to not interfere with the generated sounds. Thus, human speech, door slams and other noise were held to a minimum during the training and detection phases. In order to keep the testing as rigorous and simple as possible, the normal behavior of the experiment consisted of playing a 200 Hz tone with a specific amplitude. The algorithm was trained using this tone and then attacked by adding a 300 Hz tone with the same amplitude, followed by only using a 300 Hz tone, and finally by having no sound playing at all.

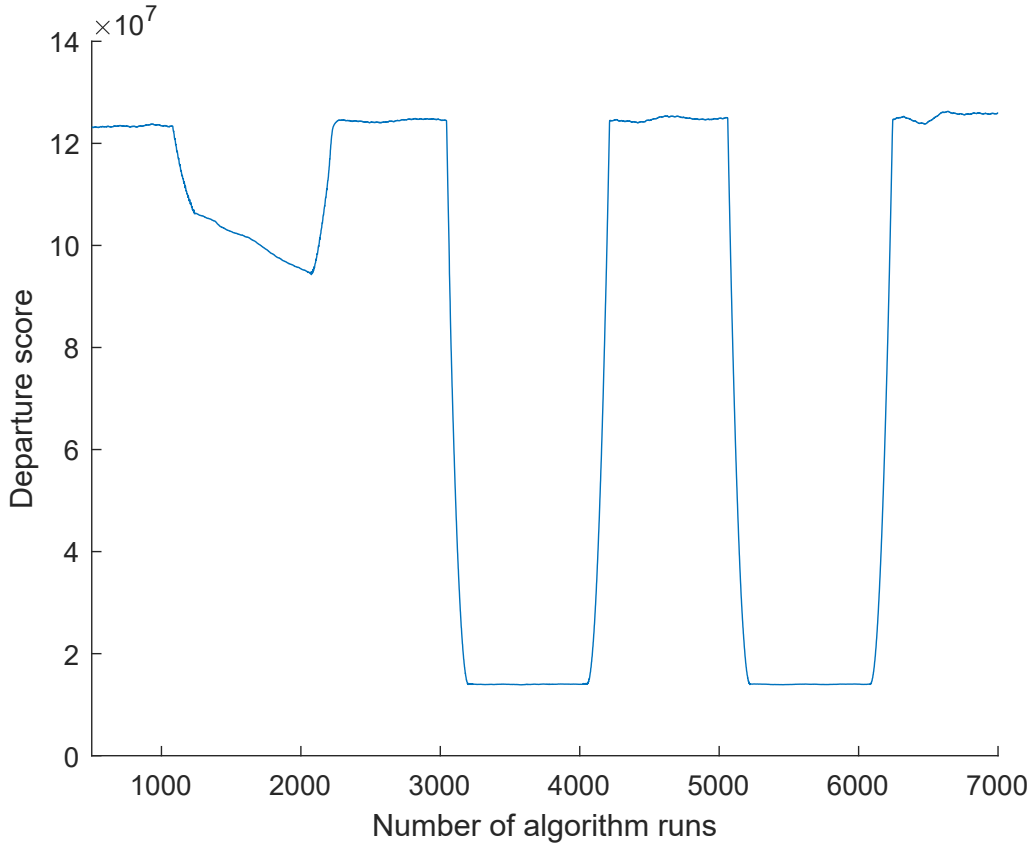


Figure 6.2: Resulting departure score when attacking the microphone sensor with three different types of attacks. An L of 3000 and an r of 6 were used for the training phase.

Initially, it was believed that the sampling rate of the ADC could be different when collecting samples for the training phase, and when collecting samples during the detection phase. However, the experiments showed that this was not true for fast-varying signals, and that the sampling rates need to be relatively close to one another to detect all attacks. Additionally, the configuration of the L and r parameters did not influence these findings. Another observation was made, namely that the resulting departure score from all the tests decreased instead of increased when the attacks

started. As such, all the attacks were detected but not accurately represented. One of the experiments is presented in Figure 6.2, where the first decrease in departure score corresponds to the added 300 Hz tone. The second decrease is caused by only playing the 300 Hz tone, and the final decrease by playing no sound at all.

After evaluating fast-varying signals with the microphone, the frequency measurement circuit was tested using the same attack scenarios. Since the output of the frequency measurement circuit consists of a DC value, sampling requirements are much lower as explained in Section 4.4.1. The experiments performed using the frequency measurement circuit, as opposed to those using the microphone sensor, were able to detect all attacks, even when using different sampling rates for the training and detection phases. In addition, the departure scores increased when attacked instead of decreased. One of the experiments is presented in Figure 6.3, where the first increase in departure score corresponds to the added 300 Hz tone. The second increase is caused by only playing the 300 Hz tone, and the final increase by playing no sound at all.

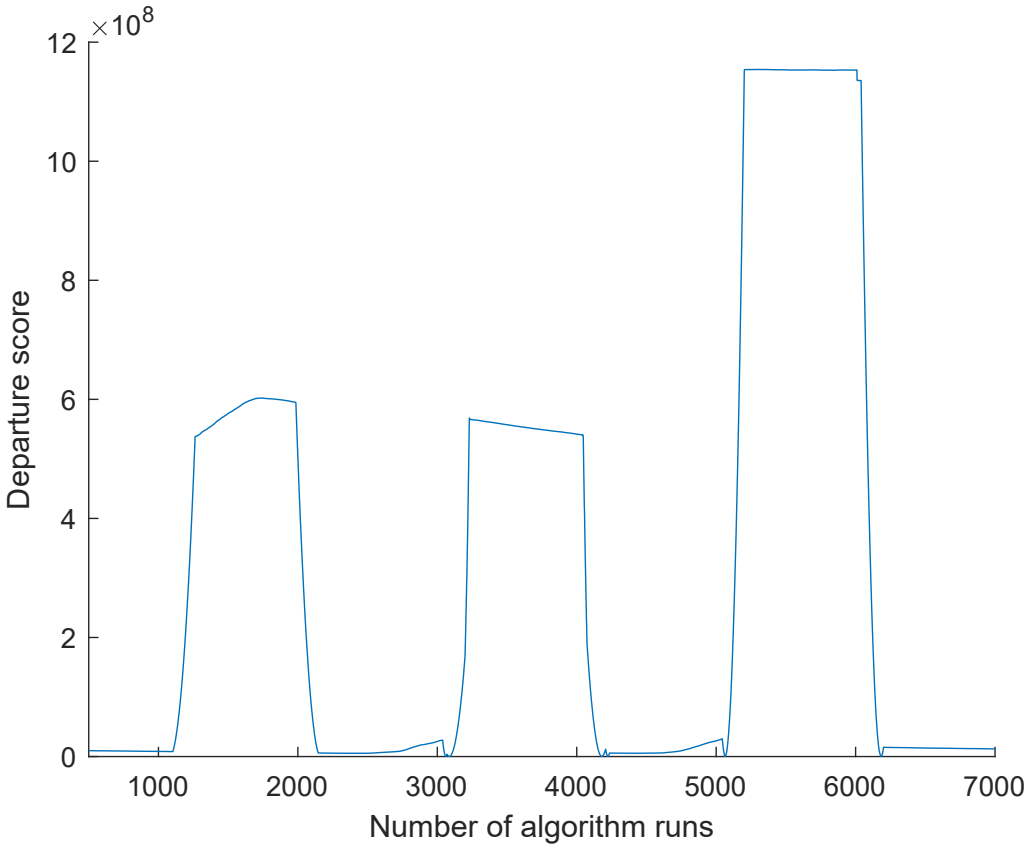


Figure 6.3: Resulting departure score when attacking the frequency sensor with three different types of attacks. An L of 3000 and an r of 6 were used for the training phase.

6.4.2 Using the Load Sensor

In contrast to the microphone sensor, there is not as much noise that can affect the load sensor. This is due to the static behavior of the load sensor which only changes its output when loaded with weights, thus not being subject to interference from background sounds. The load sensor was evaluated in two stages, first by training it without any weights on and second, by placing a spinning fan on top of the sensor in order to add noise to the measurement. Both stages were attacked by adding a small weight, which was barely visible when measuring the output of the load sensor on an oscilloscope.

The results show that all attacks in both experiments were detected and are shown in Figure 6.4 and Figure 6.5. The spikes that are seen in the figures correspond to not being able to unload the weights from the load sensor without accidentally adding some pressure.

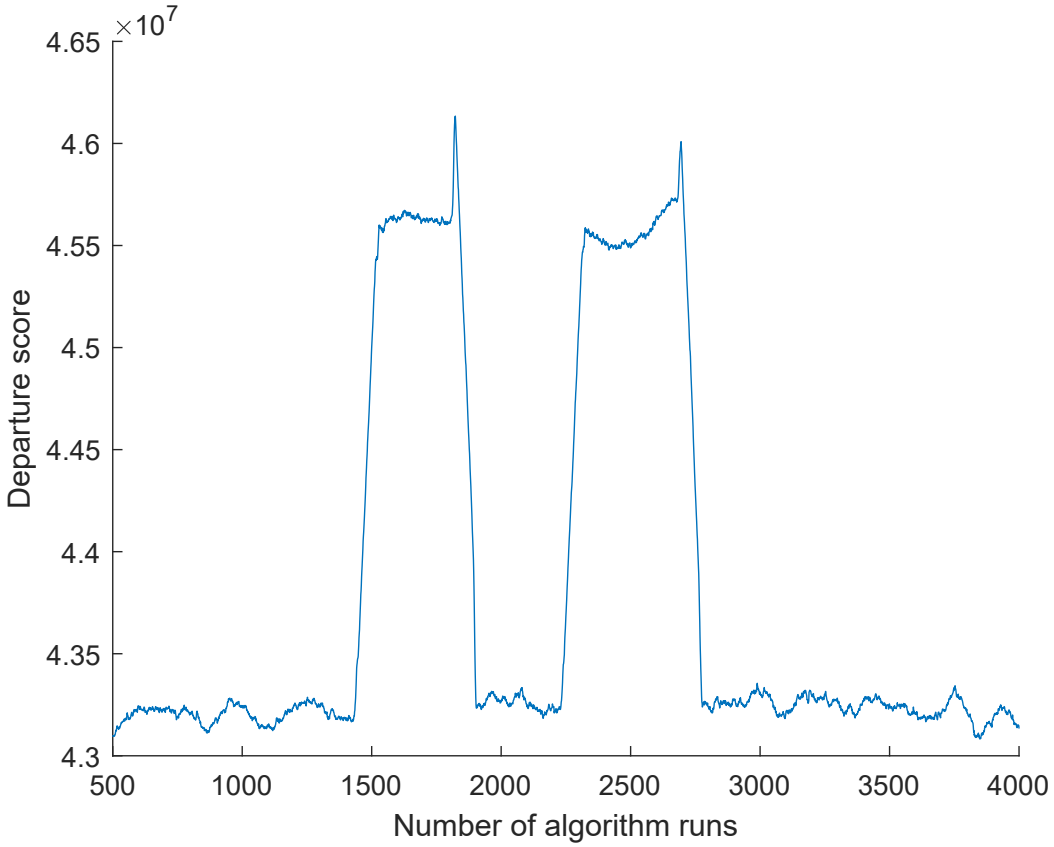


Figure 6.4: Resulting departure score when attacking the load sensor with one type of attack twice. An L of 3000 and an r of 18 were used for the training phase.

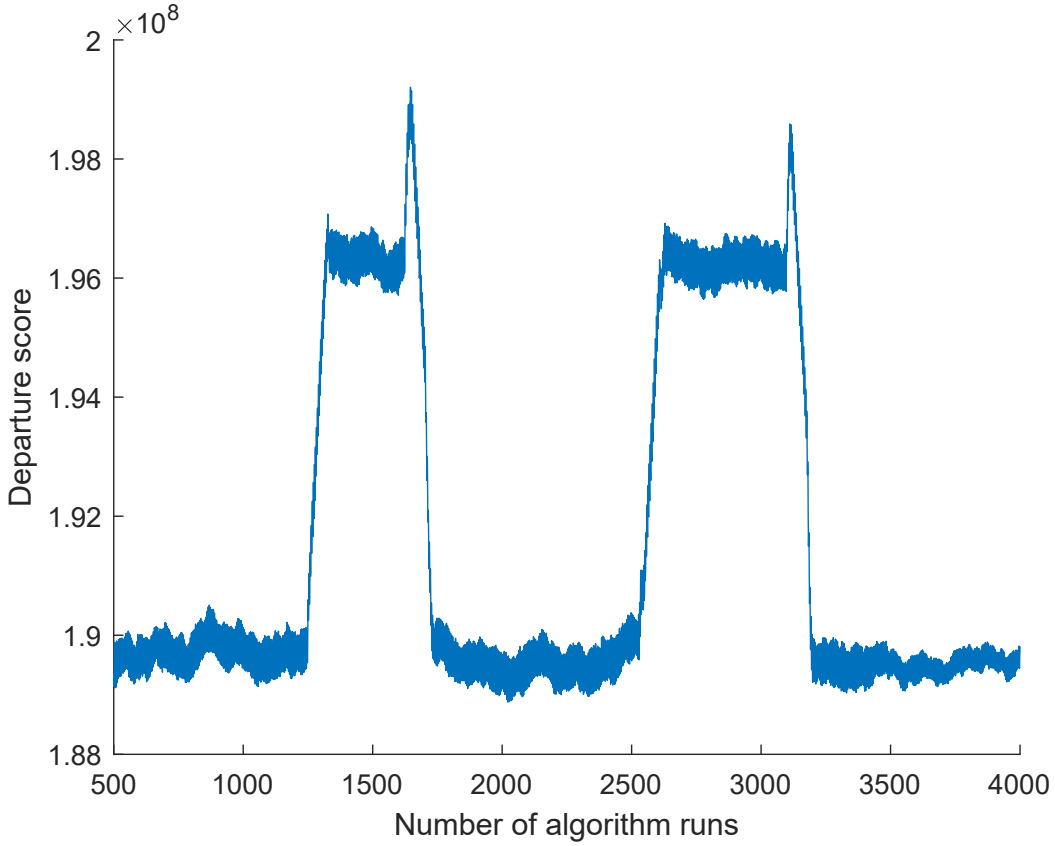


Figure 6.5: Resulting departure score when attacking the noisy load sensor with one type of attack twice. An L of 3000 and an r of 6 were used for the training phase.

Additional experiments were carried out where only noise was added to the load sensor as an attack, thus not changing the average output of the sensor. The results show that the algorithm was not able to detect these kinds of attacks.

6.4.3 Using the Vibration Sensor

Testing of the vibration sensor was performed on two types of industrial machinery, a lathe and a CNC drilling machine. These machines were chosen since they both were available for testing and because they had settings that influence the vibrations generated, more specifically, the Revolutions Per Minute (RPM) of the output axes. In both cases, the vibration sensor was fastened as close as possible to the rotating parts of the machine being tested. Then, the algorithm was trained on a specific RPM.

When testing the lathe, the algorithm was trained while the lathe was running at 800 RPM. Then, the lathe was attacked by stopping it in addition to changing the RPM to 395 RPM. The results show that the stop of the lathe was detected, but not the change of RPM. However, one needs to keep in mind that the vibrations

generated from the lathe were extremely small, in addition to different frequencies interfering from various gears in the lathe's gearbox. The attacks are presented in Figure 6.6.

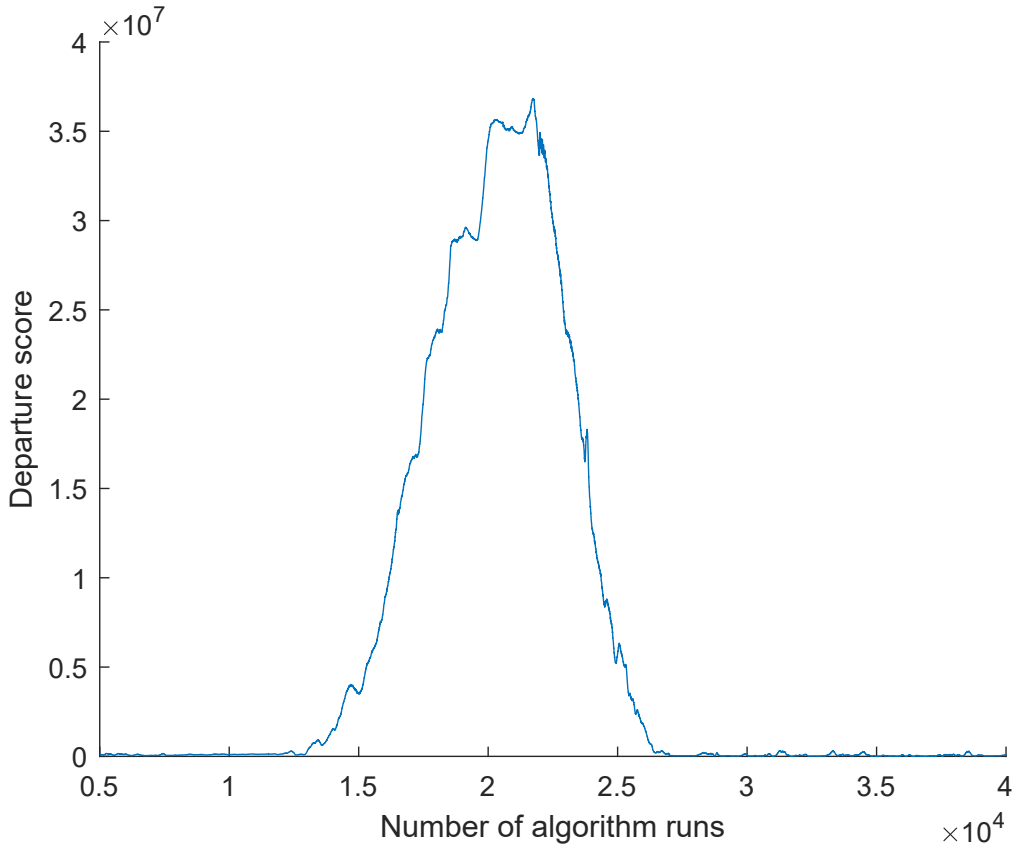


Figure 6.6: Resulting departure score when attacking the vibration sensor which is connected to a lathe with two types of attacks. The first attack is seen, as opposed to the second attack which begins at 25 000 on the x-axis. An L of 5000 and an r of 22 were used for the training phase.

The CNC drilling machine was running at around 500 RPM during the training phase of the algorithm. An attack was introduced by increasing the RPM to roughly 1000 RPM and then lowered to approximately 0 RPM. The results show that the algorithm was able to detect the increase of RPM but not the decrease to 0 RPM. Similarly to the lathe, the CNC drilling machine had a lot of interfering noise, even when running at 0 RPM. Additionally, it was difficult to adjust the RPM of the CNC drilling machine to exact RPM values. The results of the experiment is shown in Figure 6.7.

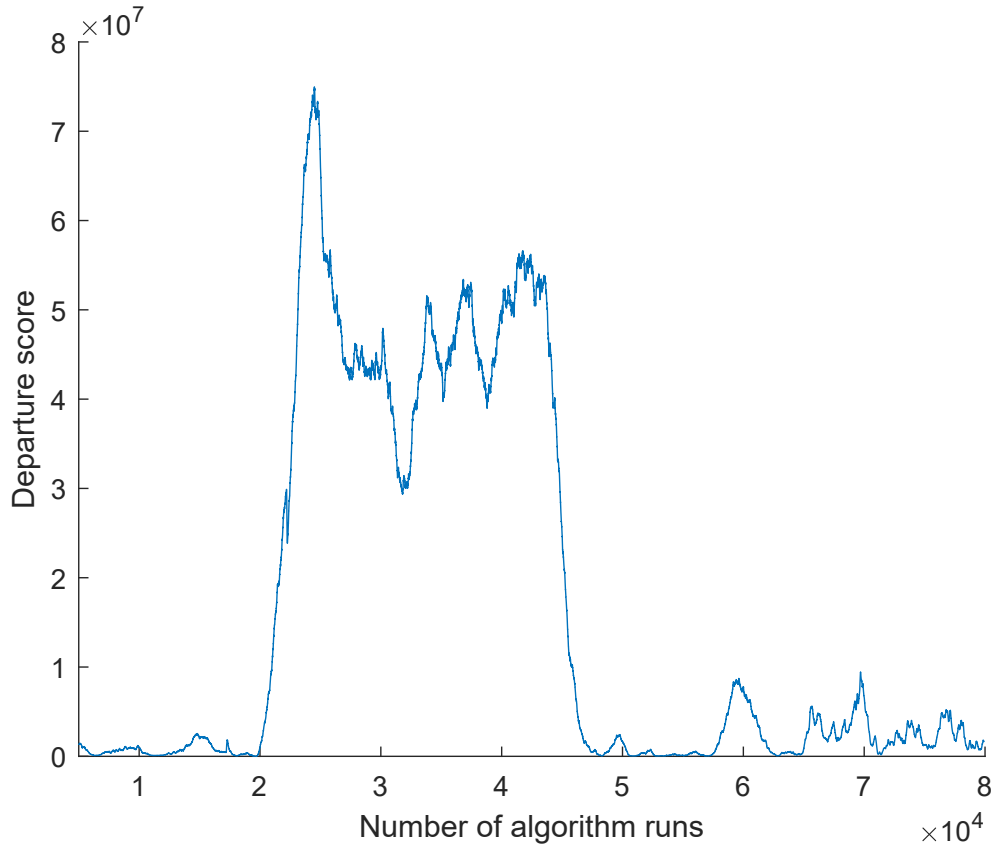


Figure 6.7: Resulting departure score when attacking the vibration sensor which is connected to a CNC drilling machine with two types of attacks. The first attack is seen, as opposed to the second attack which begins at 65 000 on the x-axis. An L of 5000 and an r of 6 were used for the training phase.

6.5 Resource Requirements

This section presents the requirements of the algorithm when implemented on the STM32F767ZI. This includes the computational requirements in addition to the memory requirements. For both the computational requirements and the memory requirements, the requirements are based on the sizes of the L and r parameters.

6.5.1 Computational Requirements

The computational requirements to run the PASAD algorithm are presented in this section. These computational requirements are presented as time complexities. Some limitations apply when calculating the time complexities, such as the overhead introduced by the interrupt-based ADC is omitted. This is because the ADC runs concurrently with the algorithm. The time complexities are based on Algorithm 2, in Section 5.2.3.

Time complexities of the algorithm when running on the embedded system:

- $\mathcal{O}(L)$ for copying the ADC buffer to the x vector.
- $\mathcal{O}(rL)$ for calculating the departure score.

6.5.2 Memory Requirements

This section presents the memory requirements of the PASAD algorithm when implemented on the embedded system. The memory requirements are derived from Figure 4.7, and are presented as space complexities for the flash memory and RAM, respectively.

Space complexities for the flash memory:

- $\mathcal{O}(r)$ for storing the $\tilde{c}$ vector, which is the centroid.
- $\mathcal{O}(r)$ for storing the w vector, which is the weight distribution.
- $\mathcal{O}(rL)$ for storing U^{T}, which is the matrix used to project a vector into the signal subspace.

Space complexities for the RAM:

- $\mathcal{O}(r)$ for storing the $\tilde{c}$ vector, which is the centroid.
- $\mathcal{O}(r)$ for storing the w vector, which is the weight distribution.
- $\mathcal{O}(L)$ for storing the ADC buffer, which is where the sampled values reside.
- $\mathcal{O}(L)$ for storing the x vector, which is the lag vector for the sampled values.
- $\mathcal{O}(r)$ for storing the p vector, which holds the projection of the x vector in the signal subspace.

In summary, the required memory for running the algorithm on the STM32F767ZI when using double-precision floating points will be: $(r + r + r \cdot L) \cdot 8$ bytes for the flash memory and $(r + r + L + L + r) \cdot 8$ bytes for the RAM.

6.5.3 Power Consumption

The power consumption was estimated by measuring the current draw and operating voltage directly on the embedded system. The power draw can be derived using Equation 6.1.

$$P = U \cdot I \tag{6.1}$$

where:

P = Power [W]
U = Voltage [V]
I = Current [A]

The results show that the sampling rate of the ADC did not measurably influence the power draw of the embedded system. Furthermore, there was no clear difference in in power draw when varying the L and r parameters. The results of the experiment are presented in Table 6.4.

Table 6.4: Power consumption depending on the L and r parameters.

L	***r***	**Current [mA]**	**Power draw at 3.24 V [mW]**
1000	20	99.7	323.0
1000	40	99.7	323.0
3000	20	99.7	323.0
5000	20	100.0	324.0
5000	40	100.0	324.0

6.6 Effects of Parameter Adjustments

This section evaluates the impact of using single- instead of double-precision floating points. Additionally, the effect on execution time when using a floating-point unit, is presented. The reason for choosing these experiments are that the results could potentially aid in evaluating the viability of implementing the algorithm on hardware that does not support double-precision floating points.

6.6.1 Double- to Single- Floating-Point Precision

The STM32F767ZI has supporting hardware for double-precision floating points, which is uncommon for other embedded systems. With this in mind, an experiment testing the suitability of using single- instead of double-precision floating points was performed. To achieve this, the compiler used for the code project of the microcontroller was modified to enable the system to only use its single-precision floating-point supporting hardware. Additionally, all calculations in the code that used double-precision floating points were changed to using single-precision floating points.

Various L and r parameters were chosen to provide a wide range of different execution times and memory usage. Also, in order to compare the mean difference in departure score between the two floating-point formats, test data were used and sent to the microcontroller from the PC. The test data were sought to have a variety of signal types, as a result, the experiments *DA1*, *DA2*, *SA1*, *SA2* and *SA3* were chosen, which is elaborated on in Section 6.1. The experiment, which compared execution times, memory usage and the mean differences in departure score between the two floating-point formats can be seen in Table 6.5.

Table 6.5: The differences in execution time, memory usage and the mean departure score when using single- instead of double-precision floating points. The values written within parentheses are the results from when double-precision floating points are used.

Experiment	L	r	**Execution time [ms]**	**Memory usage [B]**	**Mean departure score difference [%]**
DA1	1000	20	2 (3)	80 172 (160 336)	$9.3607 \cdot 10^{-5}$
DA2	1000	40	3 (5)	160 332 (320 656)	$6.9513 \cdot 10^{-4}$
SA1	3000	20	4 (7)	240 172 (480 336)	$3.9160 \cdot 10^{-1}$
SA2	5000	20	7 (12)	400 172 (800 336)	$3.2375 \cdot 10^{-3}$
SA3[1]	5000	40	13 (22)	800 332 (1 600 656)	$3.2322 \cdot 10^{5}$

[1]Further discussion regarding the high mean departure score difference of the *SA3* experiment is presented in Section 7.1.

6.6.2 Impact of a Floating-Point Unit

Not all embedded systems have hardware support for floating points, consequently, an experiment was performed in order to estimate the impact of having a floating-point unit. Various L and r parameters were chosen to provide a wide range of different execution times. Additionally, the compiler used for the code project of the microcontroller was modified so that the system would not use its floating-point hardware. By doing this, one can estimate the execution times of an embedded system that have similar specifications as the STM32F767ZI, but without a floating-point unit. The results of the experiment can be seen in Table 6.6.

Table 6.6: The differences in execution time when using no floating-point unit for calculating double-precision floating-point operations. The values written within parentheses are the results from when the floating-point unit is used.

L	r	**Execution time [ms]**
1000	20	16 (3)
1000	40	31 (5)
3000	20	45 (7)
5000	20	78 (12)
5000	40	155 (22)

7

Discussion

This section aims to discuss the findings during the evaluation part of the thesis, in addition to ethical considerations, sustainability and future work. First, the key results are presented and discussed. Second, the ethical considerations related to the thesis is elaborated upon. Third, potential future work that can be done to expand on the project is suggested.

7.1 Review of Results

In the beginning of the thesis, five research questions were proposed. Here, answers and discussions with respect to these questions are provided, in relation to the evaluation part of the thesis.

Considerations when Constructing a Test Environment

First, the aim was to find out what needs to be taken into consideration when constructing a test environment consisting of sensors to be used in an intrusion detection system. Here, it was established that constructing a test environment consisting of both analog and digital electronics can be quite challenging. There is a lot of interfacing between the analog and digital domains that needs to be performed. For example, making sure input signals are within acceptable voltage levels as to not damage the microcontroller. In addition, it is advisable to make use of the full resolution of the analog-to-digital converter in order to get the most accurate representation of the signal in the digital domain. Furthermore, the microcontroller needs to use a sampling rate that is sufficiently high, depending on the frequency of the measured signals or how quickly signal variations should be detected.

Moreover, to enable an efficient testing environment, it is important to provide a real-time communication link between the microcontroller and a PC. When the link was established, it enabled real-time transfer of test data, resulting in simplifying the evaluation of the system. When constructing this link, it was apparent that trying to implement this by using the USB protocol proved difficult. This resulted in lots of time wasted, and since there was no need for large data transfers, the

UART protocol was sufficient in addition to being simpler to implement.

Feasibility of the Intrusion Detection System

Second, the objective was to find out whether or not it is feasible to run an anomaly-based intrusion detection algorithm in real time, on a resource-constrained embedded system. It was shown in the evaluation that this was indeed possible. To begin with, in order to make subsequent evaluations of whether or not the algorithm can be run in real time, the functional correctness of the algorithm needed to be confirmed. The confirmation of the algorithm's functional correctness was performed, and was verified to produce the same departure scores as the PC, within insignificant rounding errors.

Furthermore, the execution time of the algorithm on the embedded system only took milliseconds to perform. However, the execution time of the algorithm is strongly dependent on the size of the L and r parameters. This results in slower performance when using large parameter values, still, the execution time seems to be scaling linearly. One thing that was noticed is that the execution time changed when moving around parts of the code that still were logically equivalent. It is believed that the reason for this is the use of the microcontroller's cache. This is most likely due to a higher cache miss rate when structuring the code in certain ways, resulting in slower execution time.

Detection Capabilities of the Intrusion Detection System

Third, when the execution time of the algorithm had been determined, the intrusion detection system was tested to see if it could detect various types of attacks, using the chosen sensors. The evaluation of the microphone sensor showed that it is required to use the same sampling rate when training the algorithm as when performing the detection phase. This held true for every signal tested which varied sufficiently, for example sine waves generated from sound. Additionally, the experiments performed using the microphone sensor show that attacks sometimes decrease the resulting departure score, instead of increasing it. However, this seems to be a problem with representing the departure score, rather than detecting the anomaly itself. On the other hand, the frequency sensor increased and never decreased the departure score when attacked, and still managed to detect all anomalies. However, since the output of the frequency sensor consisted of a DC signal, the sampling rate requirement is not needed.

The load sensor managed to detect all attacks, direct and stealthy, even with noise present. Furthermore, the experiments performed using the load sensor show that the size of the L parameter needs to be large when trying to detect stealthy attacks. The load sensor could not identify added noise as an anomaly, which supports the noise-resistant property of the PASAD algorithm.

The vibration sensor was tested on both a lathe and a CNC drilling machine where the attacks consisted of changing the revolutions per minute or stopping the machines. The test results from measuring these machines are a bit unclear. While stopping the lathe resulted in anomalies being detected, changing the RPM seemed to only increase the noise level and did not result in any anomaly being detected. To further investigate this, the PASAD algorithm was trained using the least significant eigenvectors instead of the most significant eigenvectors. This has the effect of training PASAD on the noise behavior instead of the underlying signal. When doing this, PASAD could detect the anomaly introduced by the additional noise generated by a change in RPM. However, when using PASAD in this way, turning off the lathe was not flagged as an anomaly.

When testing the vibration sensor on the CNC drilling machine, the results were similar to those of the lathe. Here, the change in RPM was detected by the algorithm while stopping the drill was not detected. However, stopping the CNC drilling machine consisted of turning the RPM down to 0, where the drill still vibrated a lot. This probably influenced the results to some extent. Additionally, concerning both machines, it is believed that detection was negatively impacted by the fact that there was a 50 Hz interference signal during the tests. While this signal was attenuated by a filter, it was still present to some degree. Furthermore, regarding the lathe, it was difficult to place the vibration sensor close to the rotating chuck. This resulted in capturing the vibrations from the gearbox, which introduces interfering frequencies at other RPMs than that of the rotating chuck.

Resource Requirements of the Intrusion Detection System

Fourth, the resource requirements were to be estimated. The resource requirements were evaluated in terms of computational requirements, memory requirements and power consumption. The computational and memory resources needed are mostly based on the L and r parameters. This means that for certain applications, these parameters could be smaller which could enable an even more resource-constrained system to use the algorithm. The most limiting factor between computational and memory resources, appears to be the size of the available memory. However, this will depend on how fast the departure score needs to be calculated, which ultimately makes it application dependent. The power consumption seemed to not vary significantly with different L and r parameters. In addition, a small increase in power consumption could potentially be seen for higher parameter values of L and r, but this could also be due to imprecise measurements of the microcontroller's current. When measuring the current, a simple multimeter was used, where the measured value varied with a few milliamperes. In any case, the power consumption of the microcontroller appears to be around 324 mW, irrespective of the L and r parameters.

Parameter Modifications of the Intrusion Detection System

Fifth and finally, changing system parameters and their resulting effects were investigated. More specifically, the impact on the system's performance when changing from double- to single-precision floating points or using no floating-point unit was tested. The metrics used were: execution time, mean difference in departure score, and resource requirements. The findings show that the execution time was heavily affected when switching to single-point precision, with about half. The reason for this is suspected to be due to only needing to fetch data from the flash memory of half the size. This is supported by the fact that fetching data from flash memory seems to be the limiting factor in the algorithm on the embedded system. Furthermore, the mean departure score difference between the two floating-point formats were fairly close in most cases, differing only with less than 1 %. However, in one scenario, the departure score deviated by 300 000 %. One potential explanation of this behavior, is that the test data consists of much higher values, which leaves less precision for the decimals of the floating point format.

Additionally, when not using an floating-point unit, the execution time suffered immensely. The execution time went up to between 5 and 7 times the normal execution time, depending on the size of the L and r parameters. Still, if execution time is of less concern, it could be of use for some applications.

7.2 Ethical Considerations and Sustainability

There are a couple of ethical and sustainability concerns regarding using the embedded system as an IDS, that one should take into consideration.

Since the device is collecting sensor data in order to function correctly, there are privacy concerns involved. Imagine a future where IoT devices are commonplace in homes, and embedded versions of the PASAD algorithm are used in these devices. Since the device used to detect attacks needs to collect data, there is a possibility of using this data in order to track peoples' behavior, with respect to consumer patterns and other integrity related concerns. There is a similar case to be made with industrial use of the mentioned technology. The data collected could possibly be leaked and used by competing companies to steal their intellectual property or to damage their reputation.

The system is meant to be an addition to already existing control systems, meaning that it will incur extra energy consumption (both manufacturing and operational energy consumption). In order to make the technology worth pursuing, the impact of a potential failure due to an attack that could be prevented by the system needs to outweigh the environmental impact. How to make this choice is not simple since it encompasses multiple fields, such as climate research and results from politics, e.g., environmental policies. However, since the thesis consists of implementing an IDS on a resource-constrained system with low power consumption, it could potentially

help with reducing the overall power consumption of industrial control systems by replacing existing IDSs. This would be one effort to improve and guide this type of technology toward a more sustainable path.

7.3 Future Work

This section presents suggestions for improvements that can be made to the thesis in addition to future work that can be carried out. The suggestions are presented in a list for simplicity.

- Improvements can be made to the test environment by specifically ordering components with near-perfect characteristics for what is tested for. Preferably implemented on a printed circuit board.
- Using different training vectors on the same machine in order to run multiple instances of the algorithm, possibly loaded from external sources, such as a network.
- Implementing and comparing the performance of this embedded system to other embedded systems without hardware support for double-precision floating points.
- Further testing of the system in real industrial environments is needed to test more realistic cases, over longer periods of time.
- Further testing of the double to single-precision floating-point capabilities.

8

Conclusion

The aim of this thesis was to investigate whether it is feasible to implement an anomaly-based intrusion detection algorithm on a resource-constrained embedded system in a manner that is directly interfacing with sensors. This included choosing the type of detection algorithm to be used, and the choice of embedded system to implement the algorithm on. The choice of algorithm was the process-aware stealthy-attack detection algorithm and the embedded system of choice was the STM32F767ZI from STMicroelectronics. Furthermore, the implemented algorithm on the embedded system was investigated, in addition to what needs to be taken into consideration when constructing a test environment to test the intrusion detection system. The test environment was used to test different types of attacks on the chosen sensors as well as measuring the execution time in various configurations. In addition, computational, memory, power requirements and the impact of system parameter modifications, were explored.

The major findings during the thesis suggest that it is indeed feasible to implement such an algorithm on a resource-constrained system. The execution time of the detection algorithm was found to be as low as milliseconds. Additionally, the execution time and memory usage were heavily dependent on two of the algorithm's configurable parameters, the size of the lagged vector L and the size of the statistical dimension r. As such, if large parameters are used, smaller embedded systems with less than 2 MB memory will have problems with storing the necessary data for the algorithm. Using double- or single-precision floating points also had a clear impact on the execution time and memory usage, where single-precision had less execution time and used less memory. However, the change of floating-point format resulted in erroneous calculations for some test cases, which needs further investigation. Moreover, not using a floating-point unit increased the execution time significantly, but could be used for applications where execution time is of less importance.

Furthermore, the output of the algorithm when run on the embedded system and a reference PC, resulted in the output of both cases being the same, within an insignificant rounding error. Direct and stealthy attacks were detected by the algorithm on the embedded system. However, attacks consisting of added noise were not detected. Detecting noise attacks could be detected if certain properties of the algorithm were changed. However, in this case, attacks that did not consist of noise could not be detected.

Further testing of the intrusion detection algorithm on the embedded system in real industrial environments needs to be performed in order to provide a more comprehensive evaluation. Still, the work presented in this thesis shows that implementing an IDS on a resource-constrained embedded system is possible, and is a path worth pursuing further.

Bibliography

[1] 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*. ICSS '18. San Juan, PR, USA: ACM, 2018, pp. 17–24. ISBN: 9781450362207. DOI: 10.1145/3295453.3295456. URL: http://doi.acm.org/10.1145/3295453.3295456.

[2] 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*. CCS '18. Toronto, Canada: ACM, 2018, pp. 817–831. ISBN: 9781450356930. DOI: 10.1145/3243734.3243781. URL: http://doi.acm.org/10.1145/3243734.3243781.

[3] Arduino. *Arduino Due*. URL: https://store.arduino.cc/due (visited on 03/25/2019).

[4] ARM Limited. *VFP11™ Vector Floating-point Coprocessor for ARM1136JF-S processor r1p5 Technical Reference Manual*. ARM DDI 0274H. July 2007.

[5] Atmel Corporation. *Atmel SAM3X / SAM3A Series*. Atmel-11057C. Mar. 2015.

[6] Misty Blowers, Jose Iribarne, Edward Colbert, and Alexander Kott. *The Future Internet of Things and Security of its Control Systems*. 2016. arXiv: 1610.01953 [cs.CY].

[7] Marco Caselli, Dina Hadžiosmanović, Emmanuele Zambon, and Frank Kargl. "On the Feasibility of Device Fingerprinting in Industrial Control Systems". In: *Critical Information Infrastructures Security*. Ed. by Eric Luiijf and Pieter Hartel. Cham: Springer International Publishing, 2013, pp. 155–166. ISBN: 9783319039640.

[8] James P. Farwell and Rafal Rohozinski. "Stuxnet and the Future of Cyber War". In: *Survival* 53.1 (2011), pp. 23–40. DOI: 10.1080/00396338.2011.555586. eprint: https://doi.org/10.1080/00396338.2011.555586. URL: https://doi.org/10.1080/00396338.2011.555586.

[9] Raspberry Pi Foundation. *Raspberry Pi Documentation*. URL: https://www.raspberrypi.org/documentation/faqs/#power (visited on 03/25/2019).

[10] Raspberry Pi Foundation. *Raspberry Pi Hardware*. URL: https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md (visited on 03/25/2019).

[11] 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*. ACSAC '14. New Orleans, Louisiana, USA: ACM, 2014, pp. 126–135. ISBN: 9781450330053. DOI: 10.1145/2664243.2664277. URL: http://doi.acm.org/10.1145/2664243.2664277.

[12] Faouzi Hidoussi, Homero Toral-Cruz, Djallel Eddine Boubiche, Kamaljit Lakhtaria, Albena Mihovska, and Miroslav Voznak. "Centralized IDS Based on Misuse Detection for Cluster-Based Wireless Sensors Networks". In: *Wireless Personal Communications* 85.1 (Mar. 2015), pp. 207–224. ISSN: 1572-834X. DOI: 10.1007/s11277-015-2734-2. URL: https://doi.org/10.1007/s11277-015-2734-2.

[13] Karl Hoffmann. *Applying the Wheatstone Bridge Circuit*. Rev. W1569-1.0 en. HBM Publication, 2001. URL: http://eln.teilam.gr/sites/default/files/Wheatstone%20bridge.pdf (visited on 04/14/2019).

[14] Yan Hu, An Yang, Hong Li, Yuyan Sun, and Limin Sun. "A survey of intrusion detection on industrial control systems". In: *International Journal of Distributed Sensor Networks* 14.8 (2018), p. 1550147718794615. DOI: 10.1177/1550147718794615. eprint: https://doi.org/10.1177/1550147718794615. URL: https://doi.org/10.1177/1550147718794615.

[15] Adafruit Industries. *Adafruit 4-Channel ADC Breakouts*. 2012. URL: https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/overview (visited on 03/25/2019).

[16] William Knowles, Daniel Prince, David Hutchison, Jules Ferdinand Pagna Disso, and Kevin Jones. "A survey of cyber security management in industrial control systems". In: *International Journal of Critical Infrastructure Protection* 9 (2015), pp. 52–80. ISSN: 1874-5482. DOI: https://doi.org/10.1016/j.ijcip.2015.02.002. URL: http://www.sciencedirect.com/science/article/pii/S1874548215000207.

[17] Robert M. Lee, Michael J. Assante, and Tim Conway. "Analysis of the Cyber Attack on the Ukrainian Power Grid". In: *document* (Mar. 2016).

[18] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. "Intrusion detection system: A comprehensive review". In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 16–24. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2012.09.004. URL: http://www.sciencedirect.com/science/article/pii/S1084804512001944.

[19] Tyson Macaulay. *Cybersecurity for Industrial Control Systems: SCADA, DCS, PLC, HMI, and SIS*. 1st. Boston, MA, USA: Auerbach Publications, 2012. ISBN: 9781439801963.

[20] F. Macia-Perez, F. J. Mora-Gimeno, D. Marcos-Jorquera, J. A. Gil-Martinez-Abarca, H. Ramos-Morillo, and I. Lorenzo-Fonseca. "Network Intrusion Detection System Embedded on a Smart Sensor". In: *IEEE Transactions on Industrial Electronics* 58.3 (Mar. 2011), pp. 722–732. ISSN: 0278-0046. DOI: 10.1109/TIE.2010.2052533.

[21] Federico Maggi, Matteo Matteucci, and Stefano Zanero. "Reducing false positives in anomaly detectors through fuzzy alert aggregation". In: *Information*

Fusion 10.4 (2009). Special Issue on Information Fusion in Computer Security, pp. 300–311. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2009.01.004. URL: http://www.sciencedirect.com/science/article/pii/S156625350900013X.

[22] N. J. Medrano-Marques, B. Martin-del-Brio, A. Bono-Nuez, and C. Bernal-Ruiz. "Implementing neural networks onto standard low-cost microcontrollers for sensor signal processing". In: *2005 IEEE Conference on Emerging Technologies and Factory Automation*. Vol. 2. Sept. 2005, pp. 967–972. DOI: 10.1109/ETFA.2005.1612776.

[23] Microchip Technology. *MCP3004/3008*. DS21295D. Feb. 2008.

[24] Axel Monteiro and Timothy R. Jordan. "Implementing communication between Windows PCs and test equipment using RS-232 and Borland C++ Builder". In: *Behavior Research Methods, Instruments, & Computers* 36.1 (Feb. 2004), pp. 107–112. ISSN: 1532-5970. DOI: 10.3758/BF03195556. URL: https://doi.org/10.3758/BF03195556.

[25] M. F. S. Oliveira, R. M. Redin, L. Carro, L. d. C. Lamb, and F. R. Wagner. "Software Quality Metrics and their Impact on Embedded Software". In: *2008 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software*. Apr. 2008, pp. 68–77. DOI: 10.1109/MOMPES.2008.11.

[26] Miro Oljaca, Peter Semig, and Collin Wells. *Connecting PGA900 Instrumentation Amplifier to Resistive Bridge Sensor*. Rev. SLDA032. Texas Instruments, 2015. URL: http://www.ti.com/lit/an/slda032/slda032.pdf (visited on 04/14/2019).

[27] Pololu. *Raspberry Pi 1 Model A+ 512MB*. 2016. URL: https://www.pololu.com/product/2760 (visited on 03/25/2019).

[28] The Bro Project. *The Zeek Network Security Monitor*. 2019. URL: https://www.zeek.org/download/ (visited on 04/04/2019).

[29] Paulo Shakarian, Jana Shakarian, and Andrew Ruef. "Chapter 13 - Attacking Iranian Nuclear Facilities: Stuxnet". In: *Introduction to Cyber-Warfare*. Ed. by Paulo Shakarian, Jana Shakarian, and Andrew Ruef. Boston: Syngress, 2013, pp. 223–239. ISBN: 9780124078147. DOI: https://doi.org/10.1016/B978-0-12-407814-7.00013-0. URL: http://www.sciencedirect.com/science/article/pii/B9780124078147000130.

[30] Dmitry Shalyga, Pavel Filonov, and Andrey Lavrentyev. "Anomaly Detection for Water Treatment System based on Neural Network with Automatic Architecture Optimization". In: *CoRR* abs/1807.07282 (2018). arXiv: 1807.07282. URL: http://arxiv.org/abs/1807.07282.

[31] STMicroelectronics. *RM0410 Reference manual*. RM0410 Rev 4. Mar. 2018.

[32] STMicroelectronics. *STM32F767ZI*. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f767zi.html (visited on 03/25/2019).

[33] F. M. Tabrizi and K. Pattabiraman. "Flexible Intrusion Detection Systems for Memory-Constrained Embedded Systems". In: *2015 11th European Dependable Computing Conference (EDCC)*. Sept. 2015, pp. 1–12. DOI: 10.1109/EDCC.2015.17.

[34] Texas Instruments. *LM2907 and LM2917 Frequency to Voltage Converter*. SNAS555D. Dec. 2016.

[35] Dominic Timpson and Esmiralda Moradian. "A Methodology to Enhance Industrial Control System Security". In: *Procedia Computer Science* 126 (2018). Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia, pp. 2117–2126. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.07.240. URL: http://www.sciencedirect.com/science/article/pii/S187705091831216X.

[36] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. 4th. USA: Addison-Wesley Publishing Company, 2010. ISBN: 9780321547743.

A

Data Structures and Initialization of the PASAD Algorithm

```
struct PASAD_Data {
    // ------------------------------------------------------------------------
    // Parameters
    // ------------------------------------------------------------------------

    uint32_t r_param;
    uint32_t l_param;
    double departure_score_limit;

    // ------------------------------------------------------------------------
    // Flash memory base addresses for the training vectors
    // ------------------------------------------------------------------------

    uint32_t base_address_c;
    uint32_t base_address_w;
    uint32_t base_address_ut;

    // ------------------------------------------------------------------------
    // Vector pointers to be allocated heap memory in the RAM
    // ------------------------------------------------------------------------

    // Vector holding the c vector (r * 8 bytes)
    double *c_vector;

    // Vector holding the weights (r * 8 bytes)
    double *w_vector;

    // Vector holding the projected vector (r * 8 bytes)
    double *p_vector;

    // Vector holding the x values (ADC data) (L * 2 bytes)
    uint16_t *x_vector_uint16_t;
```

```
33
34      // Vector holding the x values (test data) (L * 8 bytes)
35      double *x_vector_double;
36
37      // ----------------------------------------------------------------------
38      // Additional pointers
39      // ----------------------------------------------------------------------
40
41      // ADC buffer holding the x values (L * 2 bytes)
42      volatile uint16_t *x_buffer_uint16_t;
43
44      // Test data buffer holding the x values (L * 8 bytes)
45      double *x_buffer_double;
46
47      // ----------------------------------------------------------------------
48      // Runtime result variables
49      // ----------------------------------------------------------------------
50
51      double departure_score;
52      uint32_t execution_time;
53  };
54
55  static struct PASAD_Data* pasad_init(bool usingTestData) {
56      struct PASAD_Data *pasadData = malloc(sizeof(struct PASAD_Data));
57
58      // ----------------------------------------------------------------------
59      // Load the PASAD parameters from the flash memory
60      // ----------------------------------------------------------------------
61
62      pasadData->r_param = *(uint32_t*)(BASE_ADDRESS_TRAINING_VECTORS);
63      pasadData->l_param = *(uint32_t*)(BASE_ADDRESS_TRAINING_VECTORS + 4);
64      pasadData->departure_score_limit = *(double*)(BASE_ADDRESS_TRAINING_VECTORS + 8);
65
66      // ----------------------------------------------------------------------
67      // Calculate the vector base addresses for the flash memory
68      // ----------------------------------------------------------------------
69
70      pasadData->base_address_c = BASE_ADDRESS_TRAINING_VECTORS + 16;
71      pasadData->base_address_w = pasadData->base_address_c + pasadData->r_param * 8;
72      pasadData->base_address_ut = pasadData->base_address_w + pasadData->r_param * 8;
73
74      // ----------------------------------------------------------------------
75      // Allocate heap memory for the vectors and the buffer
76      // ----------------------------------------------------------------------
77
78      pasadData->c_vector = malloc(sizeof(double) * pasadData->r_param);
```

```
79      pasadData->w_vector = malloc(sizeof(double) * pasadData->r_param);
80      pasadData->p_vector = malloc(sizeof(double) * pasadData->r_param);
81
82      if (usingTestData) {
83          pasadData->x_vector_double = malloc(sizeof(double) * pasadData->l_param);
84          pasadData->x_buffer_double = calloc(pasadData->l_param, sizeof(double));
85      }
86      else {
87          pasadData->x_vector_uint16_t = malloc(sizeof(uint16_t) * pasadData->l_param);
88          pasadData->x_buffer_uint16_t = calloc(pasadData->l_param, sizeof(uint16_t));
89      }
90
91      // ------------------------------------------------------------------------
92      // Fill the c_vector with the training vector from the flash memory
93      // ------------------------------------------------------------------------
94
95      for (uint32_t r = 0; r < pasadData->r_param; r++) {
96          pasadData->c_vector[r] = *(double*)(pasadData->base_address_c + r * 8);
97      }
98
99      // ------------------------------------------------------------------------
100     // Fill the w_vector with the training vector from the flash memory
101     // ------------------------------------------------------------------------
102
103     for (uint32_t r = 0; r < pasadData->r_param; r++) {
104         pasadData->w_vector[r] = *(double*)(pasadData->base_address_w + r * 8);
105         pasadData->w_vector[r] *= pasadData->w_vector[r];
106     }
107
108     return pasadData;
109 }
110
111 static void pasad_clean(struct PASAD_Data *pasadData, bool usingTestData) {
112     free((void*)pasadData->c_vector);
113     free((void*)pasadData->w_vector);
114     free((void*)pasadData->p_vector);
115
116     if (usingTestData) {
117         free((void*)pasadData->x_vector_double);
118         free((void*)pasadData->x_buffer_double);
119     }
120     else {
121         free((void*)pasadData->x_vector_uint16_t);
122         free((void*)pasadData->x_buffer_uint16_t);
123     }
124 }
```

B

Departure Score Calculation of the PASAD Algorithm

```
static void pasad_departure_using_adc(struct PASAD_Data *pasadData) {
    uint32_t startTime = HAL_GetTick();

    // ------------------------------------------------------------------------
    // Storing the pointers in the stack, so that we skip the pointer operations
    // ------------------------------------------------------------------------

    uint32_t r_param = pasadData->r_param;
    uint32_t l_param = pasadData->l_param;
    uint32_t base_address_ut = pasadData->base_address_ut;

    double *c_vector = pasadData->c_vector;
    double *w_vector = pasadData->w_vector;
    double *p_vector = pasadData->p_vector;
    uint16_t *x_vector = pasadData->x_vector_uint16_t;
    volatile uint16_t *x_buffer = pasadData->x_buffer_uint16_t;

    // ------------------------------------------------------------------------
    // Filling the x_vector with the values from the x_buffer in order
    // ------------------------------------------------------------------------

    uint16_t j = counter;

    for (uint32_t l = 0; l < l_param; l++) {
        x_vector[l] = x_buffer[j];

        j = (j + 1) % l_param;
    }

    // ------------------------------------------------------------------------
    // Clearing the temporary p_vector
    // ------------------------------------------------------------------------
```

```
33
34      for (uint32_t r = 0; r < r_param; r++) {
35          p_vector[r] = 0.0f;
36      }
37
38      // ------------------------------------------------------------------------
39      // The PASAD algorithm
40      // ------------------------------------------------------------------------
41
42      double departure_score = 0.0f;
43      uint32_t address_row = base_address_ut;
44
45      for (uint32_t r = 0; r < r_param; r++) {
46          for (uint32_t l = 0; l < l_param; l++) {
47              p_vector[r] += *(double*)(address_row) * x_vector[l];
48              address_row += 8;
49          }
50
51          double y = c_vector[r] - p_vector[r];
52          departure_score += w_vector[r] * y * y;
53      }
54
55      // ------------------------------------------------------------------------
56      // Storing the results
57      // ------------------------------------------------------------------------
58
59      pasadData->departure_score = departure_score;
60      pasadData->execution_time = HAL_GetTick() - startTime;
61  }
```