"Intrusion Detection System for Electronic Communication Buses: A New Approach"

Matthew Spicer

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Mechanical Engineering

Alfred L. Wicks, Chair
Amos L. Abbott
Steve C. Southward

December 11, 2017
Blacksburg, Virginia

Keywords: CAN bus, Machine learning, Frequency analysis, Wavelets

"Intrusion Detection System for Electronic Communication Buses: A New Approach"

Matthew Spicer

Academic Abstract

With technology and computers becoming more and more sophisticated and readily available, cars have followed suit by integrating more and more microcontrollers to handle tasks ranging from controlling the radio to the brakes and steering. Handling all of these separate processors is a communication system and protocol known as Controller Area Network (CAN) bus. While the CAN bus is a robust system for sending messages, allowing control of the car through the CAN bus presents an opportunity for an outside party to interfere with the operations of a car. Any number of different methods could be used to hack the bus and take control of a car, including hacking into the bus remotely, plugging a small device into the on-board diagnostics port to the CAN bus, or swapping an existing node on the CAN bus for one that has been tampered with. This presents obvious safety risks, so to guard against this possibility, this paper will present an algorithm designed to recognize nodes based on the noise content of their signal so that any messages coming from an improper source can be flagged as suspicious.

The algorithm makes use of MATLAB® and Python to perform various transformations on the data and calculate features of the noise in a signal. These features are then passed through a statistical analysis which provides each one a score for how much useful information it contains. The best performing features are run through both a multilayer perceptron neural network and a support vector machine, and the results are compared. Each algorithm gives strong prediction performance, with prediction accuracies of 99.9% and 99.8% for the neural network and support vector machine, respectively.

"Intrusion Detection System for Electronic Communication Buses: A New Approach"

Matthew Spicer

General Audience Abstract

With technology and computers becoming more and more sophisticated and readily available, cars have followed suit by integrating more and more microcontrollers to handle tasks ranging from controlling the radio to the brakes. Handling all of these separate processors is a communication system and protocol known as Controller Area Network (CAN) bus. However, this presents an opportunity for an outside party to interfere with the operations of a car. An existing node for the CAN bus could be swapped out for one that has been tampered with, causing potentially fatal accidents. To guard against this possibility, this paper will present an algorithm designed to recognize nodes based on the noise content of their signal so that any new hardware will trigger a flag that an unrecognized source is trying to interfere. The algorithm makes use of the MATLAB® and Python programming languages to calculate certain characteristics of the noise in the signal and pass those through a machine learning algorithm. This algorithm is able to learn through mathematical means what each node "sounds like". With over 99% accuracy, we were able to predict which node sent a given signal.

Acknowledgments

I would like to thank Dr. Wicks for all of his guidance during my my time with the Mechatronics Lab. Not only did he help me with this project, but he allowed me a great opportunity to explore many different subject areas that I was previously unfamiliar with and find something that I really enjoyed doing. I would like to thank each of my committee members for their feedback and input to my work. I would also like to thank each member of the lab for all of the productive (and unproductive) conversations we've had. You've both given me good ideas and helped keep me sane, and I've learned a lot from working with all of you. Last but not least, I would like to thank my friends and family for putting up with me for so long and helping me to get to where I am now.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Security is always an important consideration for designing any system, and it is particularly important when people's lives are on the line, as is the case with automobiles. Car makers put a lot of effort into making their cars safe and frequently brag about crash safety ratings in their advertisements. However, one area of safety that has lagged behind is in securing the Controller Area Network (CAN) bus, which is used for intra-vehicular communications. Cars are becoming more and more integrated with electronics, with some cars having more than 50 electronic control units (ECUs) which control increasingly more powerful and complicated systems [1]. This increased complexity provides more vulnerabilities for exploitation, as well as more potential control for hackers. Modern cars have control of safety-critical processes through the CAN bus, such as steering and braking. This creates safety concerns since it allows a malicious third party with access to the CAN bus to potentially take control of a car. This thesis proposes a way of identifying the signals of the different nodes, or ECUs, on a CAN bus in the interest of being able to help detect intrusions and malicious activity on the bus. In theory, similar processes should also be broadly applicable to other electronic communication systems.

CAN bus hacking has started to get a lot of attention in recent years. There is a lot of publicly available information regarding interfacing with and reverse engineering a car's CAN bus. Additionally, many cheap, off-the-shelf products are available which assist in this process. A simple internet search for "CAN bus hacking" returns over $17,000,000$ results, with plenty of helpful and relevant information in at least the first two pages of results. These results include a forum for discussing CAN bus protocol, projects requiring interfacing with a CAN bus from a variety of platforms, sharing code snippets, and more [2]; several articles discussing how to interface with a CAN bus and highlighting off-the-shelf products [3, 4, 5, 6, 7], including how to use an Arduino Uno equipped with a CAN shield [8] and a system

that can be bought for less than $20 which allows control of the CAN bus through the on-board diagnostics (OBD-II) port (including control of critical features such as steering and braking) [9]; a blog detailing the design process for an "intelligent accessory control system for Jeep Wranglers" [10], including shared code, which directly uses the CAN bus to control accessories on the vehicle, such as the lights; a book titled *The Car Hacker's Handbook: A Guide for the Penetration Tester* written by a security researcher [11]; and a company which hosts car hacking challenges and conferences [12].

1.2 CAN Bus Hacking in the Literature

Since CAN packets are broadcast to all nodes on the bus and contain no built-in authentication, it is easy for components to both sniff the CAN network as well as pretend to be different ECUs to send CAN packets. With cars becoming more and more sophisticated, they have begun to integrate things such as Bluetooth and cellular/Wi-Fi networks. These interfaces provide access points for potential hackers to get into the CAN bus of a car [1]. Consequently, car hacking is becoming a hot topic. Even the government has started to get involved [13].

Researchers from the University of Washington and the University California San Diego were the first to begin the research into car hacking. They identified four main attack surfaces for a car. These are the OBD-II port as a direct physical attack, the CD slot as an indirect physical attack, the Bluetooth stack as a short-range wireless attack, and the cellular modem as a long-range wireless attack. The physical attacks, while disconcerting, are less scary since access to hardware will always provide hacking or sabotage opportunities. For instance, they could always just cut some cables on the CAN bus if they wanted to disrupt communication. However, they were able to remotely control some aspects of a vehicle both by exploiting a vulnerability in the Bluetooth stack of an ECU and compromising a cellular modem on the car, which fueled a lot of interest [14].

Inspired by this work, Miller and Valasek ([1], [15], [16], [17]) did extensive research into CAN bus hacking and reverse engineered the CAN bus on multiple cars to demonstrate the most feasible attacks. They also released all their code and tools in order to assist future researchers. We will discuss some of the highlights from these papers; however, the reader is encouraged to explore them if more detail is desired.

Miller and Valasek began by investigating both a 2010 Ford Escape and a 2010 Toyota Prius. They purchased the mechanic's tools for these vehicles and went to work reverse engineering the CAN communication for each vehicle. Once they understood the meaning of each message and were able to figure out the appropriate checksum and security algorithms, they began injecting their own messages to see how the bus would respond. One area of focus for them was diagnostic messages since these are extremely powerful. To use diagnos-

tic messages, a diagnostic session must be begun with the intended ECU. To do so requires an authentication via a cryptographic key. The ECU will send a seed, and the controlling ECU must reply with the correct computed result. If this is successful, the ECU will enter a diagnostic session. This allows extensive control over the ECU, including forcing it to pretend it is receiving given sensor values or disabling certain features, such as the brakes. However, they found this to be difficult to use because while they were able to reverse engineer the cryptographic key, once the car started moving, diagnostic sessions could not be entered and any existing session would be terminated [1]. Despite this, they were still able to demonstrate the ability to reprogram ECUs over the CAN bus; spoof dashboard readings such as speedometer, odometer, fuel remaining, and door ajar warnings; and even control the acceleration of the Toyota Prius [1]. Miller and Valasek then branched out to examining several more cars, including mostly newer models (from 2014) and classified possible remote attack surfaces (anything that performs wireless communication). These surfaces are listed and briefly explained below [15]:

- **Passive Anti-Theft System** - communication between chip in the ignition key and a device on the car; when the key is turned, its authenticity is verified through RF signals
  - Range: about $10\,cm$
  - Usefulness for attack: very low - small range and worst case is to prevent car from properly starting
- **Tire Pressure Monitoring System** - transmits real-time tire pressure data to an ECU
  - Range: $1\,m$
  - Usefulness: low - small range, but it is possible to crash the associated ECU
- **Remote Keyless Entry / Start** - short-range, encrypted radio communication to unlock or start a vehicle
  - Range: $5 - 20\,m$
  - Usefulness: low - could potentially be used for theft purposes to unlock or start the car without the proper key fob
- **Bluetooth** - intended for syncing a device, such as a phone, with the infotainment system in a car
  - Range: about $10\,m$
  - Usefulness: high - small range, but provides a reliable entry point to the automobile

- **Radio Data System** - intended for radio metadata for display (such as name of the radio station and song being played)
  - Range: several miles
  - Usefulness: moderate - difficult to successfully exploit
- **Telematics / Cellular / Wi-Fi** - connects the vehicle to a cellular radio for use with services such as OnStar
  - Range: Broad (as long as the car can have cellular communication)
  - Usefulness: very high - large range; it may not have direct access to the CAN bus, but it can be used to transfer data
- **Internet / Apps** - provides connected features, such as apps and internet browsers
  - Range: N/A
  - Usefulness: very high - opens up the door for web browser exploits and malicious apps

Miller and Valasek noted that safety-critical attacks generally require three stages. First, the attacker must gain access to an internal automotive network. Since the compromised node will likely not have direct control of the safety-critical features, the hacker will have to inject messages through the CAN bus in order to control actions such as steering, accelerating, or braking. The final stage is to reverse engineer the CAN messages to understand how to control the desired aspects of the vehicle [15]. Based on their analysis in this paper, they selected the 2014 Jeep Cherokee for another attempt at reverse engineering messages and remotely controlling the vehicle. They demonstrated access to the cellular device on the Jeep from anywhere in the country, and were able to remotely control many important vehicle systems, including steering, acceleration, brakes, and turning off the engine [16].

1.3 Intrusion Detection Systems

The National Institute of Standards and Technology (NIST) defines an intrusion as an attempt to compromise the confidentiality, integrity, and availability of a system by bypassing the security mechanisms of a computer or network. To guard against this, intrusion detection systems (IDS) are generally constructed. Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions [18]. Classical intrusion detection methodologies can be classified in three major categories: Signature-based Detection (SD), Anomaly-based Detection (AD) and Stateful Protocol Analysis (SPA). Most IDS actually use a combination of multiple categories to provide more robust and accurate detection [19]. Each of these categories will be explained,

and then a few proposed IDS for cars will be mentioned.

Signature-based detection, sometimes also called knowledge-based detection or misuse detection, looks for signatures, or patterns that correspond to a known attack or threat. SD requires comparison of known patterns against a monitored traffic of events to look for these signatures [19].

Anomaly-based detection, also known as behavior-based detection, works based on the 'normal' behavior of a system. In other words, it monitors activities on the network and looks for irregularities. It would generally need to be given some parameters to monitor and the distribution of those parameters under normal conditions. It then looks for deviations from that distribution in its observed network traffic. AD is often used alongside SD since SD will detect known attacks, and AD will detect the unknown attacks [19].

Stateful protocol analysis, or specification-based detection, requires an IDS that understands the protocol. Similar to AD, it compares network traffic to normal profiles. However, while AD looks at network-specific profiles, SPA uses vendor-developed generic profiles for specific protocols based on the international standards for that protocol [19].

Based on their research, Miller and Valasek suggested some precautions to take when building a CAN bus and what to look for to spot a potential attack. Their main suggestion for building a CAN bus is, while total isolation between ECUs with remote functionality for those which control safety-critical features may not be feasible, providing as much isolation as possible is a good idea. For example, the bus can be designed such that communication between such ECUs must go through a bridge ECU, providing an additional layer of security. For intrusion detection, they suggest monitoring the rate of messages on a CAN bus. Since most normal CAN packets will still be sent when a single node is compromised, the total messages will be a sum of the normal messages and the injected messages. Additionally, since the compromised node will generally not be the ECU typically controlling the safety-critical features, the node that does control these features will still be sending its normal messages. In order to get the car to listen to the injected messages, injected messages will need to be sent more frequently in order to make the car pay attention to them [15]. This approach to intrusion detection was further investigated and suggested in [20, 15, 21, 22]. Song et al added checking for recurring CAN ID in messages as a sign that a hacker was injecting messages on the bus [22]. However, one issue with this is that, as Miller and Valasek showed, once the initial node has been compromised, it is possible to incapacitate the safety-critical ECU, which would greatly reduce the number of messages that would need to be injected [16]. Other methods that have been suggested include encryption [23] and information-theoretic measures such as entropy and relative entropy based on information contained in the message [24].

1.3.1 Approach

This thesis will take a rather different approach to intrusion detection. Miller and Valasek noted that one of the main issues of analyzing CAN traffic is that there is no way to verify if a message came from the expected ECU or an attacker via a different, compromised ECU [17]. This thesis attempts to address that issue by "fingerprinting" the nodes on the bus so that it can be known which node is sending each packet. By determining which node is sending a message, any messages coming from an improper source can be identified as suspicious. For example, if a node from a telematics unit (such as OnStar), sends messages to control brakes or steering, this should instantly be marked as suspicious and untrustworthy.

We will perform this fingerprinting by analyzing the noise on the CAN bus. The idea behind this approach is that, although each node may have a theoretically identical transceiver which interacts with the bus, in reality, there will be slight differences between them due to the existence of finite manufacturing tolerances. These differences will manifest themselves at the smallest level, causing different patterns and likelihoods of "stray" electrons entering the bus as noise. Thus, when a particular node has control of the CAN bus, its unique noise pattern will be observable on the CAN bus. By monitoring the analog signal on the bus lines, we will propose to recognize each node's fingerprint. To do this, we treat the noise as a stochastic process and perform relevant statistical and signal analysis for characterization and make use of machine learning algorithms to map these characterizations to a classification of which node is "speaking".

This idea is similar to work done at Disney Research, where Yang and Sample are able to recognize individual electronic devices of the same type and model by measuring and quantifying their unique electromagnetic "signatures" [25]. They do this by using a software-defined radio module to measure external electromagnetic interference (EMI) from electronic devices. They then convert this to the frequency domain and store the most significant frequency magnitudes as the devices "EM-ID" [25]. They collect an EM-ID from each of the devices and store this in a database. To identify a device, they measure its external EMI and compare this to their database of EM-IDs. In their tests, they were able to identify devices with accuracies varying between 72% to 100%, depending on the device type [25].

This approach is not necessarily meant to be a replacement for traditional intrusion detection approaches. While it has potential to be a key component of intrusion detection systems, it is easy to envision it being combined with traditional approaches, such as a search for known attack signatures, in order to create an even more robust system.

In order to perform this classification, it is necessary to understand the process which generates noise, appropriate models for noise, how these models can be leveraged to extract features (ie, representative information describing the signal), and methods of learning differences between the features describing each node. The theory behind the methods used

for all of these things and the reason for choosing each method will be explained later.

The rest of this thesis is outlined as follows. Chapter 2 will discuss the CAN bus protocol. A brief history will be given (2.1), relevant standards will be mentioned (2.2), some advantages of the CAN protocol, such as its strong noise immunity, will be discussed (2.3), and the packet structure will be outlined and discussed (2.4). Chapter 3 will discuss the bulk of the relevant theory used for characterization and classification of the signals in this thesis. This will involve discussions on statistical analysis techniques, including hypothesis testing (3.1); stochastic models and types of noise (3.2); frequency analysis (3.3), wavelet analysis (3.4); additional features of interest (3.5), learning algorithms for classification (5.3); and feature selection techniques (5.2). Chapter 4 will discuss a proposition for how to fingerprint nodes on a CAN bus. All steps used in classifying the nodes - including preprocessing, feature extractions, training, and testing - will be discussed there. Chapter 5 will present the results obtained using the test CAN bus. An example of the measured noise and each of the resulting features will be presented. This thesis will close with a discussion on some issues regarding a physical implementation and a review of what's been accomplished in Chapter 6.

Chapter 2

CAN Bus Technology

2.1 Beginnings

Development of the CAN bus began in 1983 by engineers at by Robert Bosch GmbH to meet the needs of automotive engineers. The CAN bus was first introduced in 1986, providing a multi-master system with non-destructive arbitration to provide priority for the most important messages, as well as error handling and the automatic disconnection of faulty nodes. By mid 1987, Intel had already created the first CAN controller chip [26]. In 1991, the Mercedes W140 was the first car to include a CAN network, connecting five ECUs [27]. In 1993, version 2.0 of the Bosch CAN specification was standardized in ISO 11519-2, adding a desccription of the physical layer of the CAN bus, and an addendum was added with ISO 11898 in 1995, adding an extended frame format with a CAN identifier extension [26].

2.2 Standards

CAN bus has many associated standards that have been published by the International Standards Organization (ISO) and the Society of Automotive Engineers (SAE). The most relevant of these standards for cars are listed and briefly explained in Table 2.1.

2.3 Advantages

The Controller Area Network (CAN) bus is a robust communication protocol designed to be resistant to electrical interference. The protocol implements twisted pair, balanced line, differential signaling, and (sometimes) electrical shielding to provide good noise immunity. It also uses a synchronization protocol to ensure nodes on the bus are all in agreement about

ISO 11898 series	The main CAN standard specifying the physical and data link layers of CAN for use in road vehicles [28, 29, 30]
ISO 16845 series	Provides methodology for testing an implementation's conformance to ISO 11898 [31, 32]
ISO 15765 series	Specifies diagnostics for CAN vehicle network systems specified in ISO 11898 [33, 34, 35]
J2284 series	SAE Recommended Practice defining the Physical Layer and portions of the Data Link Layer for High-Speed CAN, including for use with CAN Flexible Data-rate (CAN FD) up to 5 Mbit/s[1] [36, 37, 38, 39, 40, 41]
J2411	SAE Recommended practice defining the Physical Layer and portions of the Data Link Layer for a single-wire CAN network [42]

Table 2.1: A table outlining the published standards important for CAN communication in cars.

the data being sent on the bus. This allows for communication speeds from $50\,kbit/s$ up to $1\,Mbit/s$ at bus lengths from $40\,m$ (at $1\,Mbit/s$) to $1\,km$ (at $50\,kbit/s$). For these reasons, it is the main choice for communication between different electronic control units (ECU) on cars and is also often chosen for use in other fields, such as robotics.

The combination of balanced line, twisted pair, and differential signaling is crucial to the high noise immunity provided by CAN bus. That the lines are balanced means there is equal but opposite current flowing in each signal line. This gives a field-canceling effect prevents cross-talk between the wires. Additionally, since external noise sources generally result from coupling of electrical and magnetic fields, the use of twisted pair causes external noise sources that are sufficiently far away to affect each line equally. This is because the twisting causes the noise source to see an approximately equal amount of each wire. Combining this with differential signaling through twisted pair CANH and CANL (CAN high and CAN low) lines provides strong external noise immunity through the use of strong common mode rejection. Any common noise from external sources is subtracted out, giving a stable differential signal and allowing for reliable communication.

Furthermore, signal reflections are prevented through the use of $120\,\Omega$ terminating resistors, which are specified to match the characteristic impedance of the line [43]. This is necessary since mismatched impedances can cause reflections back from the higher impedance towards the lower impedance. Current, like most things in nature, prefers to take the path of least resistance. The characteristic impedance of a transmission line can be calculated by equation 2.1, where L is inductance per unit length and C is capacitance per unit length [44].

$$Z = \sqrt{\frac{L}{C}} \tag{2.1}$$

The CAN bus also implements a synchronization protocol that keeps the bit rates of the receiving nodes aligned with the rate of the transmitting node. Nodes are synchronized on the bit edges to keep a frequent update and prevent nodes from drifting apart. To help this, bit stuffing is implemented. This means that, whenever there are 5 consecutive same bits, a bit of the opposite kind is stuffed in to keep the nodes in sync. This bit is ignored when parsing the message [45].

2.4 Packet Protocol

A packet on the CAN bus consists of different frames, which are shown in Figure 2.1 and outlined below [43]:

- **Start of Frame (SOF)** - single dominant bit indicating the beginning of a packet
- **Identifier (ID)** - establishes priority (the lower the binary value, the higher the priority) and purpose for the message
- **Remote Transmission Request (RTR)** - dominant when information is required from another node via a response
- **Identifier Extension (IDE)** - dominant indicates no extension; recessive indicates more identifier bits will follow
- **Reserved Bit (R)** - for possible use by future amendment
- **Data Length Code (DLC)** - 4 bits, contains number of bytes of data being transmitted
- **Data** - up to 64 bits of data
- **Cyclic Redundancy Check (CRC)** - checksum for error detection
- **Acknowledgement (ACK)** - every node receiving an accurate message submits a dominant bit; if a node detects an error and leaves this bit recessive, the message is ignored and a retransmission is attempted
- **End of Frame (EOF)** - 7 (recessive) bits marking the end of the frame (not stuffed)
- **Interframe Space (IFS)** - minimum of 7 (recessive) bits between frames (not stuffed)

Figure 2.1: The progression of a single CAN bus frame.

Most of these are fairly self-explanatory, and a more detailed understanding is not crucial for this thesis. However, we will go into more depth concerning the identifier segment since it is a crucial component for understanding how traffic on a CAN bus works.

Communication on CAN bus is done by broadcasting. This means that all nodes on a bus are all connected together through the same signal lines, and each node receives every message. To handle this type of communication, each message is given an identification number, or CAN ID, which lets each node know if the content of that message is relevant to them and how they should handle the message. Each message purpose will have a pre-specified ID. For example, the ECU connected to the speedometer will frequently send out a message updating the bus on what the vehicle's current speed is. This message will always have the same ID. Thus, when a node sees a message with this CAN ID, it knows that this message will contain the current speed of the vehicle, and will likely store this value in its appropriate location in memory for future use [43].

CAN IDs are assigned based on the priority of the message, which is used to handle contention for control of the bus through a process known as arbitration. This is used when multiple nodes on the bus want to send a message at the same time. The CAN ID is placed at the beginning of the CAN packet so that nodes must go through arbitration before being allowed to proceed with their message. This ensures that high priority messages are sent first to reduce latency for the most important processes. The standard CAN ID length is 11 bits, but it can be extended to 29 bits through the use of the identifier extension bit. Collisions among contending CAN ID bits are resolved using logical AND semantics. In other words, for the bus to read a 1, all nodes output must be 1, and if at least one node outputs a 0, then the line will be driven to 0. This works because the lines on the bus are passively biased to the low value (or logic 1)[2][3]. When a node vying for control of the bus detects a different signal on the bus than its output (ie, a dominant bit while it's sending a recessive bit) during the arbitration phase, it knows there is a higher priority message being sent, and it withdraws from arbitration. If a node reads its own bits on the bus throughout the whole arbitration process, it knows it is the winner of arbitration, and it continues transmitting the rest of its message while the other nodes switch to a passive listening mode [45]. This

[2]The two signal lines of the bus, CANH and CANL, are passively biased to $\approx 2.5\,V$. When in this state, CANH and CANL read the same thing; thus, it reads as a low value on the bus. When the bus reads high, CANH goes approximately $1.5\,V$ higher to $\approx 4\,V$, and CANL goes approximately $1.5\,V$ lower to $\approx 1\,V$ to create a $3\,V$ differential signal, which will read as a high value [43].

[3]It is worth noting that an interesting gotcha for CAN bus communication is that a high value maps to logical 0 (dominant), while a low value maps to logical 1 (recessive).

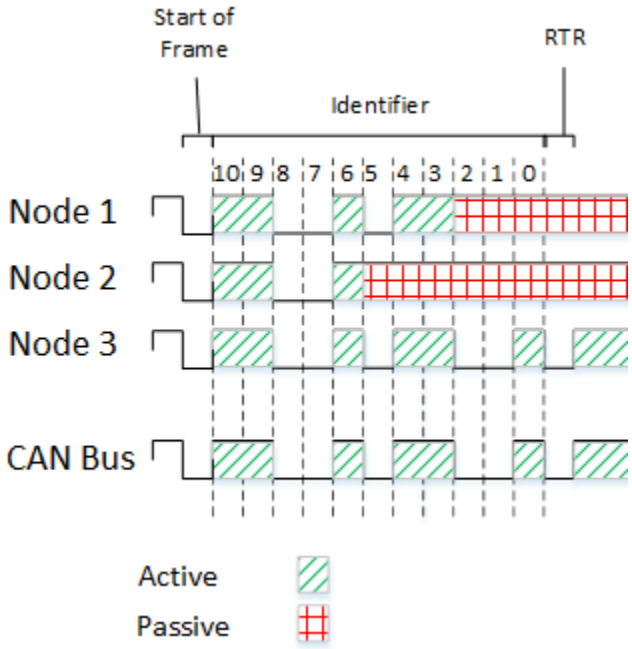


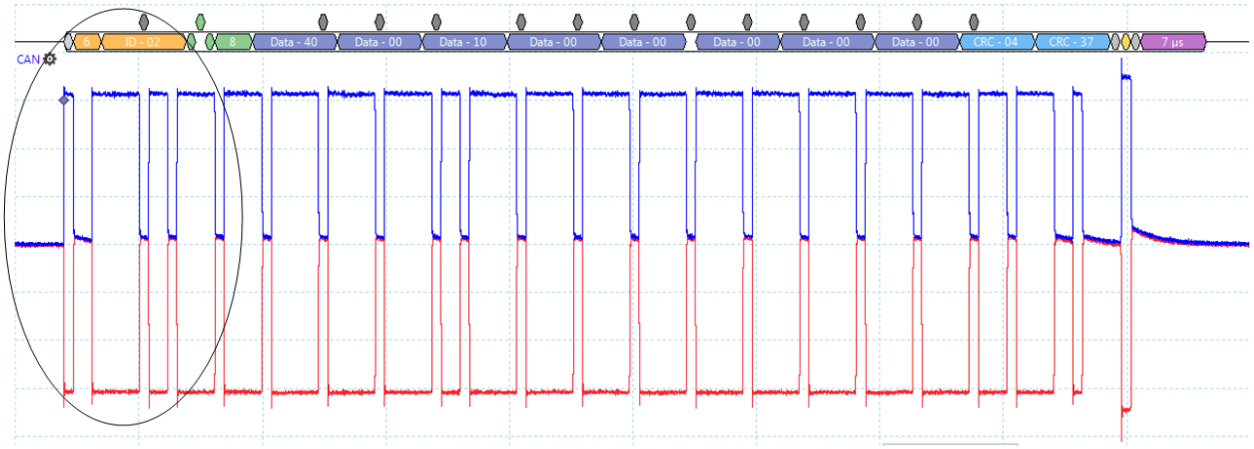
Figure 2.2: An example of arbitration on the CAN bus. In this scenario, node 3 wins the arbitration and continues with the rest of the message.

arbitration process is shown in Figure 2.2.

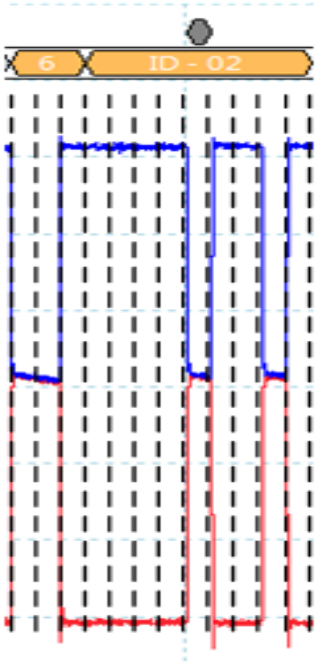
There are four message types on the CAN bus. They are: data frames (for transmitting data), remote frames (for requesting data), error frames (for flagging a detected error), and overload frames (for requesting an extra delay when a node is too busy) [43]. In order to understand how information is communicated on the CAN bus, we can ignore the less common error and overload frames and focus on data and remote frames. These message types are used by cars in three important ways. The standard data frame is used in an informative capacity. For example, the Power Steering Control Module (PSCM) will periodically broadcast the current position of the steering wheel. This kind of message does not have any physical effect - it simply is used to update values in memory. Another use is to request some sort of action from another ECU. For example, if the Adaptive Cruise Control (ACC) module determines the brakes need to be applied, it would request this action over the CAN bus. The third usage is even more powerful - diagnostic messages. Diagnostic messages are intended for communication between a mechanic's tool and ECUs. They can request that an ECU perform a test action or get diagnostic information. However, since they are only meant for use in a repair shop, these messages are usually ignored by a car in motion to prevent dangerous actions [17].

An example of a data frame CAN message with an ID of 0x602, collected with a Picoscope 5000 series oscilloscope, is displayed in Figure 2.3. A zoomed-in view of the identifier

section is displayed, and dotted lines indicate each bit. The bit marked by the gray hexagon above the hexadecimal values is a stuffed bit. In this case, the 0x6 in the identifier signals a data transfer, while the 0x02 indicates it is for the node associated with the identifier 0x02[4].



(a) An example data frame on a CAN bus. The message content is displayed in hexadecimal above the waveform.



(b) The identifier section of the CAN packet.

Figure 2.3: An example of a collected data packet from the CAN bus. A zoomed-in view of the identifier is shown.

[4]It is worth noting that cars are not necessarily likely to allocate their identifier in the same manner.

Chapter 3

Theory

3.1 Statistical Analysis

Any signal processing application always requires a thorough treatment of statistics. In this thesis, we are attempting to classify signals from different nodes on a CAN bus. This will require identification of some traits, or features, of the measured signal that consistently produce different values for each of the nodes. A feature can be evaluated for its discriminatory power based on statistical differences between the populations of a feature for each node. This section will discuss background regarding the concept of statistical populations and hypothesis testing for differences between populations.

3.1.1 Populations and Random Variables

In order to explain the relevant statistical analysis techniques, we will start by giving a background on statistical populations. This will provide necessary background for understanding how stochastic processes, such as noise, can be modeled, as well as how to differentiate between two (or more) different stochastic processes. In statistics, a population refers to the set of all items or events that represent something of interest to an experiment. For example, when drawing cards from a deck, the population would be the set of all cards in the deck. For convenience, a random variable is used to map items or events from a population to the real number line [46]. Continuing with the deck of cards example, we have a few choices for our random variable depending on what is of interest to the experiment. For example, we could decide that all we care about is the value of the card and not its suit or color. In this case, we could give all 2's a value of 2, all 3's a value of 3, etc. Alternatively, if we care about the suit, we could number each card 1-52, and the random variable would take on the value of whatever number had been assigned to a specific card. Depending on how we define the random variable, it will have some governing probability distribution which maps values of

the random variable to a probability of occurring. For the deck of cards, we would likely be dealing with a uniform distribution since each card is equally likely to be drawn.

Probability distributions can be described using probability mass or density functions. Probability mass functions (pmf) are used to describe discrete random variables, such as the cards from the last paragraph, where each value of $f(x)$ maps to a probability of observing the value x. Probability density functions (pdf) describe continuous random variables, such as a distance between two objects. For a continuous random variable, the probability of any specific value occurring is 0, so $f(x)$ gives a probability density for values around x. To get a probability, one must look at a range of values for x and find the integral for $f(x)$ over that range. Sometimes it is also convenient to deal with cumulative distribution functions (cdf). This function gives the probability of observing a value less than or equal to x. In other words, for the continuous case, we would define the cdf $F(x)$ as in equation 3.1 [46].

$$F(x) = \int_{-\infty}^{x} f(x)dx \tag{3.1}$$

Important properties of random variables and distributions include expected value, variance, covariance, correlation. The expected value of a random variable is defined as the weighted average of the random variable given, by equation 3.2. For an infinite sample, this would be equivalent to the mean. The variance of a random variable describes the average squared distance from the mean and is given in equation 3.3. Variance is the square of the more frequently used standard deviation. If we want to compare random variables, we can use covariance or correlation, which is a linear measure of the relationship between two random variables. Correlation is basically a scaled covariance, guaranteed to be between $[-1, 1]$, used to gain an intuitive understanding of how closely two variables are related. These are shown in equations 3.4 and 3.5. When dealing with time series, we often talk about auto- and cross-covariance, shown in equations 3.6 and 3.7, which define the relationship at different time lags [47].

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} xf(x)dx \tag{3.2}$$

$$\mathbb{V}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] \tag{3.3}$$

$$\mathbb{C}[X, Y] = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \tag{3.4}$$

$$\rho = \frac{\mathbb{C}[\mathbb{X}, \mathbb{Y}]}{\sqrt{\mathbb{V}[X]\mathbb{V}[Y]}} \tag{3.5}$$

$$C_{XX}(\tau) = \mathbb{E}[(X_{t_1} - \mathbb{E}[X_{t_1}])(X_{t_1+\tau} - \mathbb{E}[X_{t_1+\tau}])] \tag{3.6}$$

$$C_{XY}(\tau) = \mathbb{E}[(X_{t_1} - \mathbb{E}[X_{t_1}])(Y_{t_1+\tau} - \mathbb{E}[Y_{t_1+\tau}])] \tag{3.7}$$

A distribution of general interest, which is also quite relevant to this thesis, is the normal (Gaussian) distribution, with probability density function given in 3.8. The Gaussian distribution is so relevant thanks in large part to the central limit theorem, which states that the limiting distribution for any random variable defined as the summation of many independent random variables, no matter what their underlying distribution, will be a Gaussian distribution [46].

$$P[X = x] = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{3.8}$$

3.1.2 Hypothesis Tests

Now that we've built a little bit of background, we can start talking about applying statistical principles to hypothesis tests. Hypothesis tests are important for understanding similarities and differences between statistical populations. Hypothesis tests will be used later to evaluate features. Features which provide statistically significant differences between the populations of each node are identified as features with discriminatory power and are thus desirable for use in the classification of the nodes.

Hypothesis tests are a set of statistical tests which give a probability, or p-value, of some observation given a set of conditions or assumptions. In general, p-values test a "null hypothesis" (H_0). The p-value gives a probability of observing the data (or more extreme) if H_0 is assumed to be true. It makes sense, then, that a low p-value indicates we should be rather suspicious of the null hypothesis, while a high p-value indicates that H_0 is at least plausible. A commonly accepted significance level is $\alpha = 0.05$, which means that we will reject the truth of the null hypothesis at $p < \alpha$. In this section we will discuss some relevant hypothesis tests. We'll begin with the independent samples t-test, then move on to the Kolmogorov-Smirnov (KS) test.

The independent samples t-test is used for testing the difference between the mean of two populations (assuming they are approximately normally distributed). It is perhaps the most commonly used test for evaluating the difference between two populations. The null hypothesis is generally $\mu_1 = \mu_2$ (the means of the two populations are equal). It uses the test statistic shown in equation 3.9, where X_k is sample k, s^2_{pooled} is the pooled variance estimate, and N_k is the number of data points in sample k. Pooled variance is found by equation 3.10, where s^2_k is the variance of sample k and $df_k = N_k - 1$ is the number of degrees of freedom. Assuming the two means are equal, this test statistic follows what is known as a t distribution with $df_{total} = df_1 + df_2$ degrees of freedom. Thus, the value of t can be turned into a p-value based on where it falls within its t distribution. This p-value represents the probability of observing the found difference between the two means or more extreme (larger

difference), assuming that the means are actually equal.

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s^2_{pooled}}{N_1} + \frac{s^2_{pooled}}{N_2}}} \tag{3.9}$$

$$s^2_{pooled} = \frac{df_1 s_1^2 + df_2 s_2^2}{df_{total}} \tag{3.10}$$

The KS test is a test for differences between two statistical distributions. It makes no assumption on the distribution for either population, which makes it useful for populations with unknown distribution. It works using the test statistic shown in equation 3.11, which is the maximum distance between two cdfs weighted by the square root of the number of samples n. This statistic follows a Kolmogorov distribution, which has the cdf shown in equation 3.12 [48]. This value is turned into a p-value by evaluating the Kolmogorov distribution. Details for this can be found in [49].

$$D = \sqrt{n}\ sup_{x \in \mathbb{R}} |F_2(x) - F_1(x)| \tag{3.11}$$

$$P[D \leq t] \rightarrow H(t) = 1 - 2\sum_{i=0}^{\infty} (-1)^{i-1} e^{-2i^2 t} \tag{3.12}$$

3.2 Noise Models

This section will extend the statistical discussion to discuss stochastic processes. We will discuss some models for how values in a population are generated, with an eye towards specifically how noise may be modeled as a stochastic process. We will then mention some existing mathematical models related to noise in semiconductor devices.

3.2.1 Stochastic Models

There are many different stochastic process theories which have been developed that apply well to random noise. Understanding these processes and their models will give good insight for how to approach feature extraction. Some highlights of stochastic processes that have been used to describe noise include the Markov process, martingale, Poisson process, and Gaussian process. We will start by discussing the Markov property and Martingales, then move on to Poisson and Gaussian processes. As it will be seen, these processes are all quite related.

A Markov process is any process that has the Markovian property, which is shown in equation

3.13. In words, it essentially says that future states depend only on the current state. This is taken a step further with martingales, which place restrictions on the expected value of future values. The martingale, submartingale, and supermartingale are defined in equation 3.14, where, in all cases, the expected value is assumed to be finite [50]. A good example of a process that can have all of these properties is the random walk, which has a huge number of applications from noise [51, 52] to neural activity [53]. This process is often explained by comparing it to a drunk person trying to walk. Simplifying to one dimension, we can think of a drunk person trying to walk forward. In his or her drunkenness, he or she may take a step either forward or backward. In the case that each of these is equally likely, it is called a zero-mean random walk (which is also a martingale). A graph of this is shown in Figure 3.1.

$$P[x_n = j | X_{n-1} = i, X_{n-2} = k, ..., X_0 = l] = P[X_n = j | X_{n-1} = i] \tag{3.13}$$

$$E[x_n | X_{n-1} = i, X_{n-2} = k, ..., X_0 = l] = X_{n-1} \quad \forall \quad n \geq 2 \tag{3.14a}$$
$$E[x_n | X_{n-1} = i, X_{n-2} = k, ..., X_0 = l] \leq X_{n-1} \quad \forall \quad n \geq 2 \tag{3.14b}$$
$$E[x_n | X_{n-1} = i, X_{n-2} = k, ..., X_0 = l] \geq X_{n-1} \quad \forall \quad n \geq 2 \tag{3.14c}$$

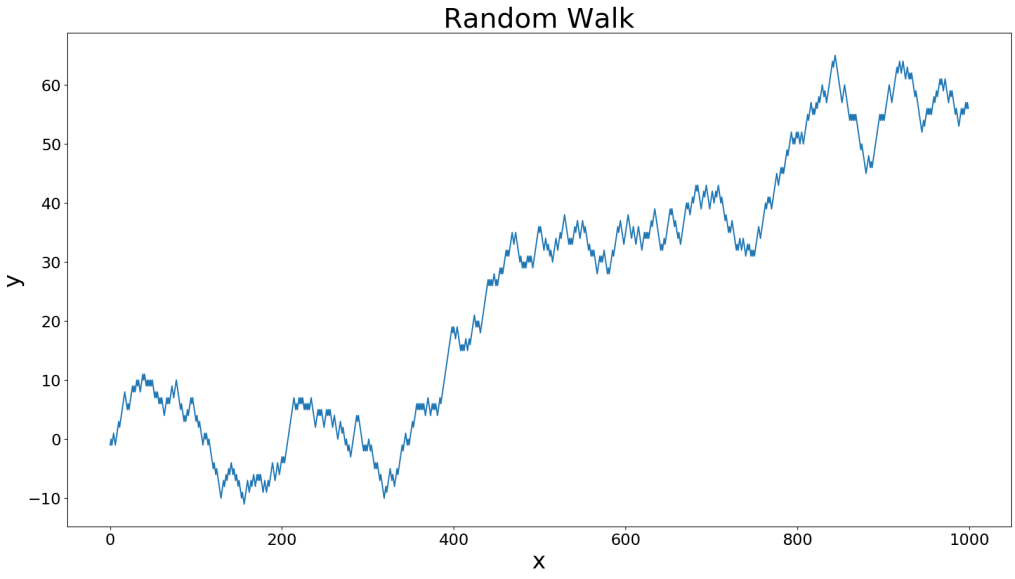


Figure 3.1: An example of a zero-mean random walk.

A Poisson process is any process which follows 5 basic assumptions. For each $t \geq 0$, where N_t is an integer-valued random variable which can be thought of as representing a number of arrivals, we have:

1. $N_0 = 0$ (start with no arrivals)
2. $s < t \Rightarrow N_s$ and $N_t - N_s$ are independent (arrivals in disjoint time periods are independent)
3. N_s and N_{t-s} are identically distributed (number of arrivals depends only on period length)
4. $lim_{t\to 0}\frac{P[N_t=1]}{t} = \lambda$ (arrival probability is proportional to period length if the length is small)
5. $lim_{t\to 0}\frac{P[N_t>1]}{t} = 0$ (no simultaneous arrivals)

Any such process will have an N_t that follows the Poisson distribution (for any given t) with pmf shown in equation 3.15 [46]. As can be seen from property 3, any Poisson process is an example of a supermartingale since future arrivals don't depend on past arrivals and N_t can obviously only get larger as t increases. The parameter lambda is generally given by a rate of arrival multiplied by the length of time, and N_t will represent the distribution for the number of arrivals in that time frame. Noise can be modeled to be the result of a Poisson process since it is the result of discrete electron "arrivals" [52, 51, 53, 54, 55].

$$P[N_t = n] = e^{-\lambda t}\frac{(\lambda t)^n}{n!} \tag{3.15}$$

The Gaussian process is perhaps the most common process observed in nature, thanks to the central limit theorem, as described earlier. Since observed noise is generally the result of many atomic-level events coming together, it is well-modeled by a Gaussian [47]. This also follows from noise's interpretation as a Poisson process since a Poisson distribution is well approximated by a Gaussian for large λ (ie, when the rate of arrival is large and N_t models a sum of large numbers of independent arrivals over a given time period). The Gaussian distribution is presented in equation 3.8 [46]. The Gaussian property of noise is used by many important applications, such as the Kalman filter's Gaussian assumption [56].

3.2.2 Physical Origins

Electrical noise occurs in part due to the discrete and probabilistic nature of charge and matter at small scales. Observed noise is the result of averages over the randomness of a large number of particles. Interference from electric fields both within electrical devices and from outside sources also plays a role in contributing to noise. There are three main types of electrical noise - thermal noise, shot noise, and flicker noise. In general, all three types can be modeled as Gaussian processes. This section will give only a brief overview of each of these

three types of noise. For more detailed explanations, the reader is encouraged to explore [47].

Thermal noise is the result of random thermal motion, and can be understood by treating the particles as an ideal gas. Any two-terminal linear electrical circuit with a purely resistive impedance will display thermal noise which depends only on resistance and temperature. For measurable frequencies and normal temperature ranges, the spectral content of current fluctuations due to thermal noise can be accurately represented by equation 3.16, where T is temperature, k is Boltzmann's constant, and R is its resistance. Since Boltzmann's constant is so small, $(1.38 * 10^{-23})$, this will likely be only a very small amount of any measured noise [47].

$$S_{th}(f) = \frac{2kT}{R} \tag{3.16}$$

Shot noise can be present in any device which contains some potential barrier. Noise is created whenever an electron gains sufficient energy to cross this barrier. The stochastic model for shot noise current is given by equation 3.17, where q is the charge of an electron, $N(t)$ is the number of electrons which have crossed a potential barrier at time t (with crossing times t_i) as modeled by a Poisson process, and δ is the Dirac-delta function. The spectral density of the current may be time-varying and is given by equation 3.18, where $V_T = kT/q$. [47].

$$Q(t) = qN(T) \tag{3.17a}$$

$$I(t) = \frac{dQ(t)}{dt} = \sum_i q\delta(t - t_i) \tag{3.17b}$$

$$S_s(t, f) = qI_s(e^{V/V_T} + 1) \tag{3.18}$$

Flicker noise refers to excess noise observed at low frequencies that can't be explained by thermal or shot noise. It is observed in many different phenomena aside from just electrical noise. Since it is often inversely proportional to frequency, flicker noise is sometimes referred to as $1/f$ noise. The characteristics of flicker noise often differ from device to device, even for two of the same type of devices from the same die. Various theories have been proposed for flicker noise in many different electronic components, each resulting in slightly different models. From experimental work, the time-invariant spectral density of flicker noise is often modeled as shown in equation 3.19, where K is a device-specific constant, a is a constant in $[0.5, 2]$, and b is a constant approximately equal to 1 [47].

$$S_{1/f}(f) = K\frac{I^a}{f^b} \tag{3.19}$$

3.3 Frequency Analysis

As shown above in section 3.2.2, each of the types of electrical noise has an associated frequency domain representation which depends on some device-specific parameters. Since measuring these parameters to the necessary resolution would be extremely difficult if not impossible and, in the case of flicker noise, the exact physical origins of these parameters are unknown, it makes sense to look for differences in the frequency estimates of the measured noise to assist with identifying device-related differences between the generated noises. This section will first discuss the Fourier transform. Building upon the Fourier transform are many different spectral estimation techniques that are designed to give better estimates of the spectral content of a signal based on certain assumptions. Here, we will discuss two of these - Welch's estimation using averaging and a technique which uses autoregressive model fitting.

3.3.1 Fourier Transform

The Fourier transform, shown in equation 3.20, is one of the most basic techniques for frequency analysis. It uses the complex exponential to split up a signal into a summation of sines and cosines of different frequencies. In practice, we use a finitized and discretized version of this transform shown in equation 3.21) [57] (more specifically, we use the Fast Fourier Transform algorithm [58] for computations). The effects of discretizing and finitizing this transform are well-studied. Taking a finite time segment for a signal is the same as multiplying the infinite signal by a rectangular window function defined on $t \in [-T/2, T/2]$. It can be shown that multiplication in the time domain is equivalent to convolution in the frequency domain, so the result of an FFT will give the true frequency representation of our signal convolved with the frequency representation of the window. This creates leakage in the result of the FFT, which essentially means that any frequencies present in the signal become spread out through neighboring frequencies in the FFT according to the FFT of the window. For the rectangular window, this results in leakage in the form of the sinc function as shown by its FFT in Figure 3.2. To help mitigate these effects, one can choose a different window for the data, such as Chebyshev, Hamming, or Hann. For example, a pure sine wave of frequency 2 Hz contains only one frequency. However, cutting it off in a finite rectangular window (that is not a multiple of its period in length), gives significant leakage in the FFT. This effect is shown in Figure 3.4, where, for the purposes of this example, we contrast the use of no window (a rectangular window) with that of a Chebyshev window (whose FFT is shown in Figure 3.3) to show how this can significantly improve our frequency estimate. The Chebyshev window is chosen for use in this thesis based on its combination of low side lobe level while maintaining a relatively thin main lobe. Since frequency magnitudes will be looked at individually, it is especially important to eliminate the bias in estimates due to spectral leakage. Thus, the Chebyshev window was parameterized to give 90 db of side lobe

attenuation ($\alpha = 4.5$). Further investigation of windows can be found in [57].

$$X(\omega) = \int_{-\infty}^{\infty} X(t)e^{-j\omega t}dt \tag{3.20}$$

$$X(\omega) = \sum_{n=-N/2}^{N/2} X(nT)e^{-j\omega nT} \tag{3.21}$$

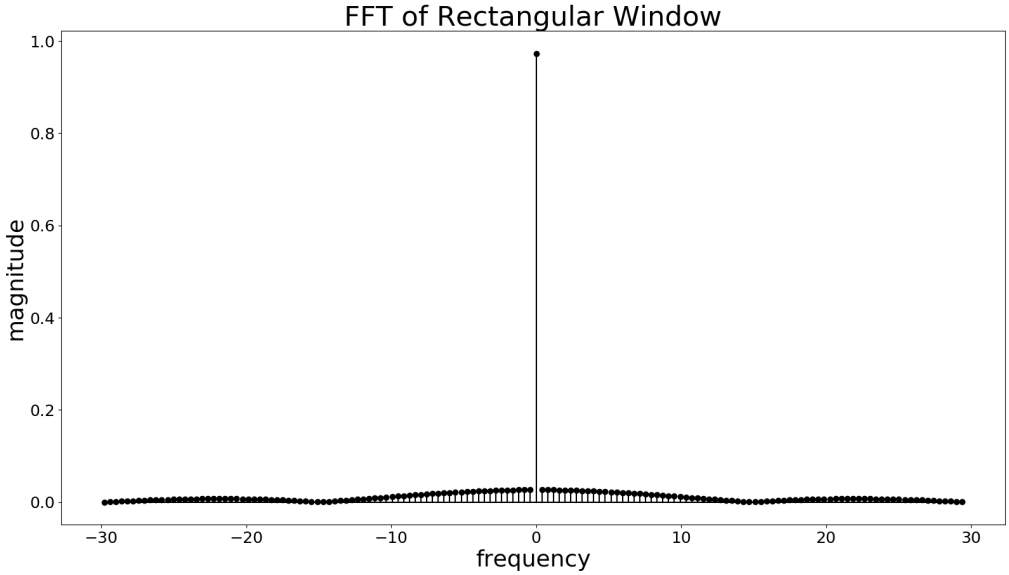


Figure 3.2: The FFT of a rectangular window.

3.3.2 Welch's Method

Another common method for estimating spectral density is Welch's method. This method is an attempt to compensate for randomness when only one signal is present. This can often give improved estimates over a simple FFT and is a logical starting point for spectral estimation without requiring known signal parameters. Welch's method works by assuming stationarity of the signal and breaking the data up into K possibly overlapping segments of length L. These segments are windowed, and the Fourier of each segment is taken as normal. The spectral estimate is obtained by taking the average value of the squared magnitude for each frequency. This process is shown in equations 3.22-3.25, where X is the signal, X_k is the kth segment, W is the window, I is the periodogram (squared-magnitude of the FFT) for each segment, and P is the spectral estimate of the signal [59]. For comparison, Welch's

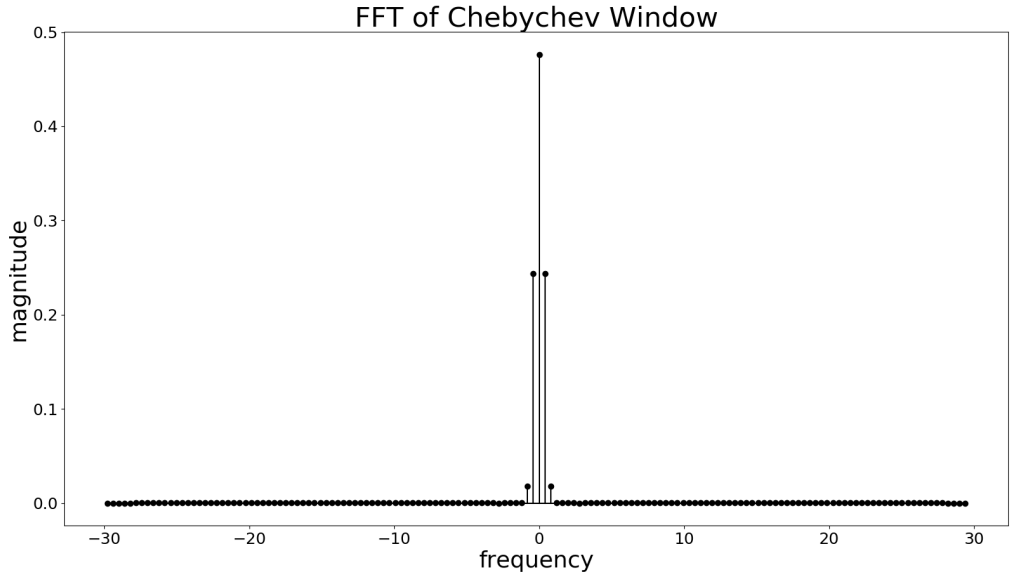


Figure 3.3: The FFT of a Chebyshev window.

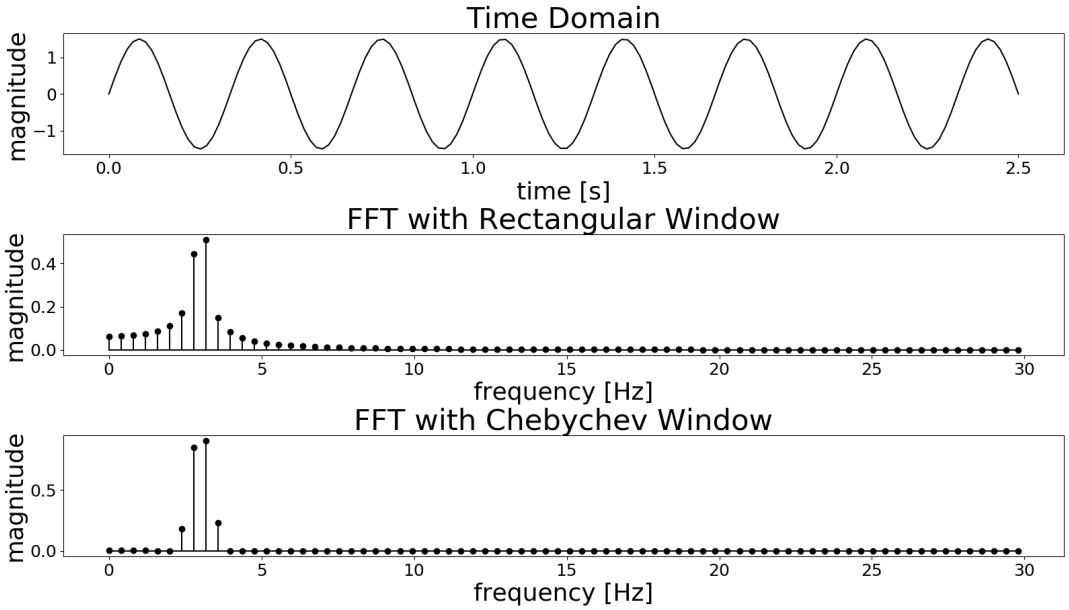


Figure 3.4: A 3 Hz sine wave (top), its FFT with a rectangular window (middle), and its FFT with a Chebyshev window.

method is applied to the same 3 Hz sine wave from before, and the result is shown in Figure

3.5. As can be seen, one disadvantage of this approach is the loss of frequency resolution from breaking the signal up into multiple segments.

$$X_k(\omega) = \sum_{n=-N/2}^{N/2} X(nT)W(nT)e^{-j\omega nT}dt \tag{3.22}$$

$$I_k(\omega) = \frac{L}{U}|X_k(\omega)|^2 \tag{3.23}$$

$$U = \frac{1}{L}\sum_{n=-N/2}^{N/2} W(nT)^2 \tag{3.24}$$

$$P(\omega) = \frac{1}{K}\sum_{k=1}^{K} I_k(\omega) \tag{3.25}$$

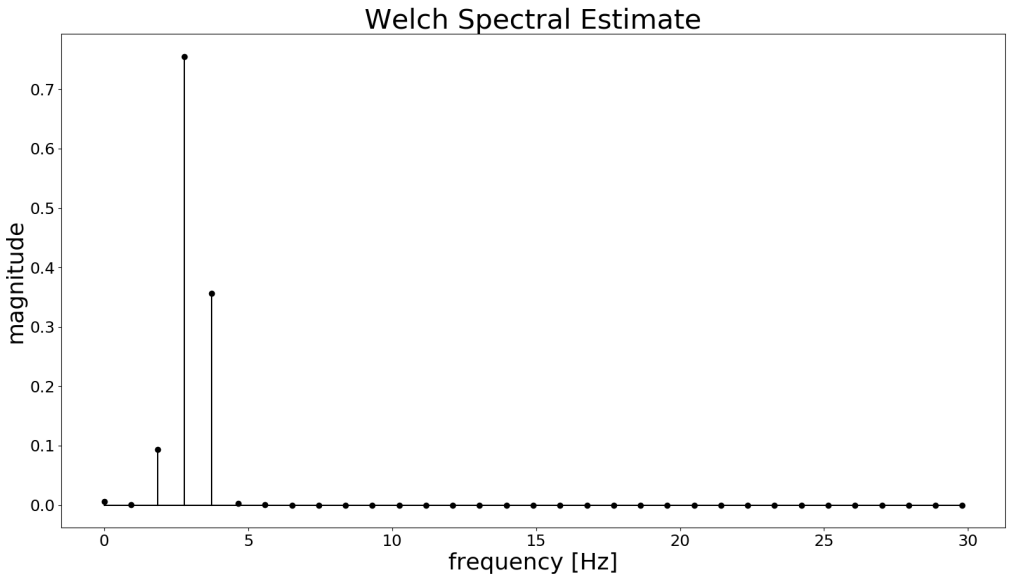


Figure 3.5: The Welch power spectral density estimate of the 3 Hz sine wave.

3.3.3 Autoregressive Techniques

Spectral estimation can also be performed by constructing an autoregressive representation for the signal. Autoregressive models construct signals based on previous values. In other words, given a certain number of previous values (from the order of the model), a linear

prediction, or regression, is used to generate the next value. For such signals, autoregressive techniques can give good spectral estimates. A downside of this technique is it requires a priori knowledge of a parameter for the signal - the autoregressive order. However, there are ways for estimating this, such as in [60]. An IIR filter driven by a serially independent noise sequence would produce an autoregressive signal. Then, since analog filters, often included on CAN bus lines and transceivers, as well as any data acquisition system, are generally IIR filters, this model makes sense for use in estimating the spectral content of the measured noise on the CAN bus.

The autoregressive spectral technique is constructed under the assumption of a non-deterministic, weakly stationary time series $X(n)$. Under this assumption, $X(n)$ can be represented by the $(M+1)^{th}$ order autoregressive model in equation 3.26, where ε is white noise. The characteristic equation of this form is given in equation 3.27. To compute the power spectral density, we estimate the coefficients using the covariance method, which uses a least-squares method to solve equation 3.28. The variance - found with equation 3.29, where C_{XX} is the auto-covariance function - is also used to scale the spectral estimate. The power spectral density estimate is then given by equation 3.30[1] [61]. For reference, the 3^{rd} order autoregressive PSD estimate of the 3 Hz sine wave is shown in Figure 3.6. With this method, we don't lose any frequency resolution. It is worth noting that this signal does not meet our assumptions for the construction of this estimate; thus, any comparison with the previous methods should be taken with a grain of salt.

$$X(n) = \varepsilon(n) + \sum_{m=1}^{M} a_m X(n-m) \tag{3.26}$$

$$0 = 1 - \sum_{m=1}^{M} a_m z^m \tag{3.27}$$

$$C_{XX}(l) = \sum_{m=1}^{M} \hat{a_m} C_{XX}(l-m) \tag{3.28}$$

$$S^2(M) = C_{XX}(0) - \sum_{m=1}^{M} \hat{a_m} C_{XX}(m) \tag{3.29}$$

$$\hat{p_{xx}}(f) = \frac{S^2(M)}{|1 - \sum_{m=1}^{M} \hat{a_m} e^{-i2\pi f m}|^2} \tag{3.30}$$

[1] This is an estimate of the autospectrum of $X(n)$, $p_{xx} = \sum_{l=-\infty}^{\infty} C_{XX}(l) e^{-i2\pi f l}$.

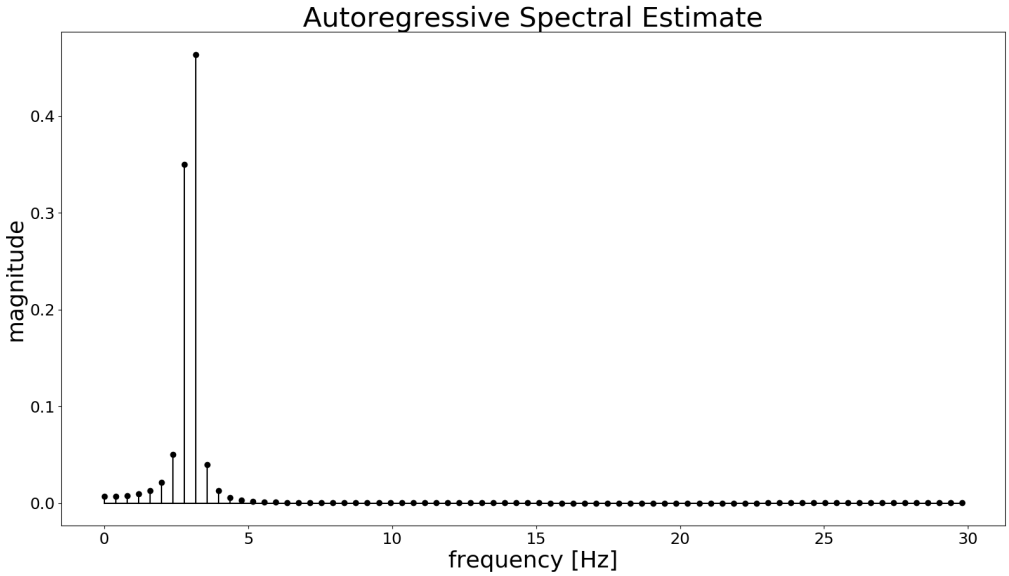


Figure 3.6: The autoregressive power spectral density estimate of the 3 Hz sine wave.

3.4 Wavelet Analysis

Wavelets originated as an extension of the Fourier Transform to finite, orthonormal basis functions. The original wavelet, Morlet's wavelet, was a continuous wavelet transform which was made to be an alternative to the short-time Fourier transform (STFT) for time-frequency analysis. Morlet wanted precise time resolution for high-frequency signals, as well as precise frequency resolution for low-frequency signals. Due to inputs from researchers in various fields from quantum mechanics to electrical engineering, wavelets have since grown and evolved to encompass many more applications. For example, the discrete wavelet transform (used in this thesis)[2] is often used to perform subband FIR filtering [62].

Wavelets have been shown to be a good feature to extract for time series classification [63]. Due to their finite nature, wavelets are able to provide local signal information that would be missed by the previously discussed frequency analysis techniques. Additionally, this makes wavelets much more useful for approximating choppy data with sharp discontinuities (eg, noise) than a Fourier transform [64]. For these reasons, wavelets are chosen to complement the frequency analysis to provide more complete statistical representations of the data.

The discrete wavelet transform was created by Ingrid Daubechies in 1988. It is made up

[2]It is worth noting that the discrete wavelet transform is not the same as a discretized continuous wavelet transform. Continuous wavelet transforms may be calculated discretely. There is actually a significant difference between the analysis for continuous wavelet transforms and discrete wavelet transforms.

of two functions - a scale function and a wavelet function, sometimes called the father and mother wavelets, respectively. Each is convolved with the data to generate $\frac{N}{2}$ coefficients (where N is the number of data points) since the convolution is done by sliding with a factor of two (in other words, every other value is skipped). The scale function gives what is known as approximation coefficients, which can be thought of as a lower dimensional approximation of the original data, while the wavelet function gives detail coefficients, which contain the rest of the information from the original signal that is not contained in the approximation coefficients. If desired, this process can then be repeated on each new set of approximation coefficients until there are not enough coefficients remaining to perform another transform. This repetition is known as a multiresolution analysis, with each successive transform giving another layer of decomposition [65].

A wavelet must be chosen for the analysis. Since noise is generally sharp rather than smooth, smoother wavelets with more vanishing moments[3] are discarded, eliminating symlets. Additionally, since both the approximation and detail coefficients will be used, it is believed that it is unnecessary to have vanishing moments from both the mother and father wavelets, eliminating coiflets. Of the most commonly used discrete wavelets, this leaves Daubechies 4 tap wavelet (db2 for its 2 vanishing moments) from Ingrid Daubechies' original discrete wavelets as an appealing choice.

Daubechies 4 tap wavelet transform is given by the orthonormal basis in its scaling and wavelet coefficients in equations 3.31 and 3.32. The associated multiresolution analysis is given by equations 3.33 and 3.34, where the first subscript denotes the decomposition level (so ϕ_{0k} is the signal and ϕ_{1k} is the first decomposition, etc.) and the second subscript denotes the k^{th} value of the associated signal or decomposition. These functions are shown graphically in Figure 3.7 [65]. Additionally, the frequency response function of the scale and wavelet functions are shown in Figure 3.8.

$$h(0) = \frac{1 \mp \sqrt{3}}{4\sqrt{2}} \tag{3.31a}$$

$$h(1) = \frac{3 \mp \sqrt{3}}{4\sqrt{2}} \tag{3.31b}$$

$$h(2) = \frac{3 \pm \sqrt{3}}{4\sqrt{2}} \tag{3.31c}$$

$$h(3) = \frac{1 \pm \sqrt{3}}{4\sqrt{2}} \tag{3.31d}$$

$$g(n) = (-1)^n h(-n+1) \tag{3.32}$$

[3]The m^{th} moment of a function is defined as $\int x^m f(x)dx$. If a function has k vanishing moments, then $\int x^m f(x)dx$, $m = 0, 1, ..., k-1$.

$$\phi_{jk} = \sum_n h(n-2k)\phi_{j-1n} \tag{3.33}$$

$$\psi_{jk} = \sum_n g(n-2k)\phi_{j-1n} \tag{3.34}$$

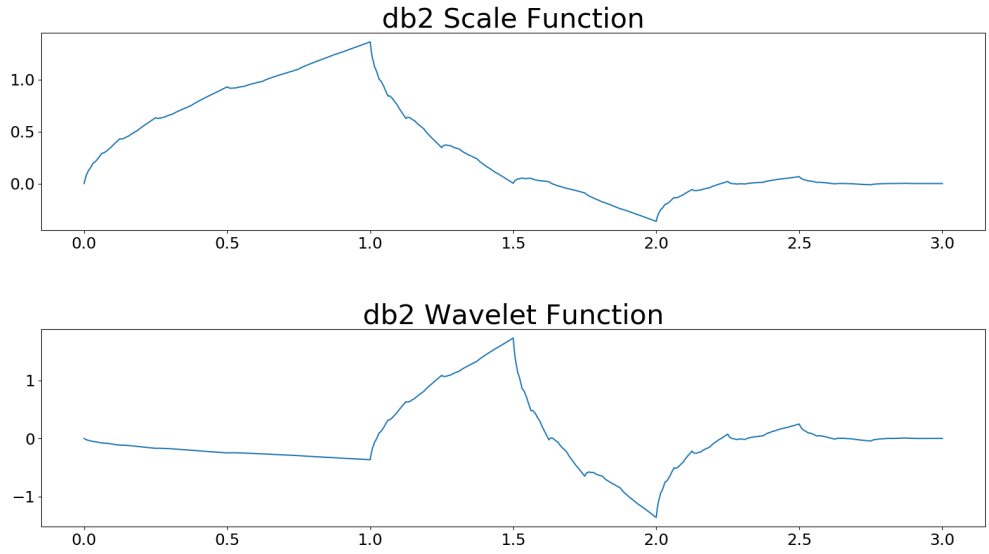


Figure 3.7: The db2 scale and wavelet functions.

3.5 Additional Feature Extraction

In addition to frequency and wavelet analyses, some additional features have been chosen for extraction. These include the percentage of positive derivative terms, sum of the absolute value of the derivative, variance, autocovariance, time-reversal asymmetry statistic, and timing information. Many of these features have been chosen based on the work by Fulcher and Jones in [66] and [67]. Of these features, this section will cover those that have not been discussed up to this point (and are not self-explanatory): time-reversal asymmetry statistic and timing information.

Many different time-reversal statistics exist in literature, including [67] and [68]. In general, time-reversal amounts to a sort of skewness[4] measure at a certain lag. In keeping with

[4]Skewness is defined as the third central moment of a function: $skew = \int_{-\infty}^{\infty}(x-\mu)^3 f(x)dx$, where μ is the expected value of x.

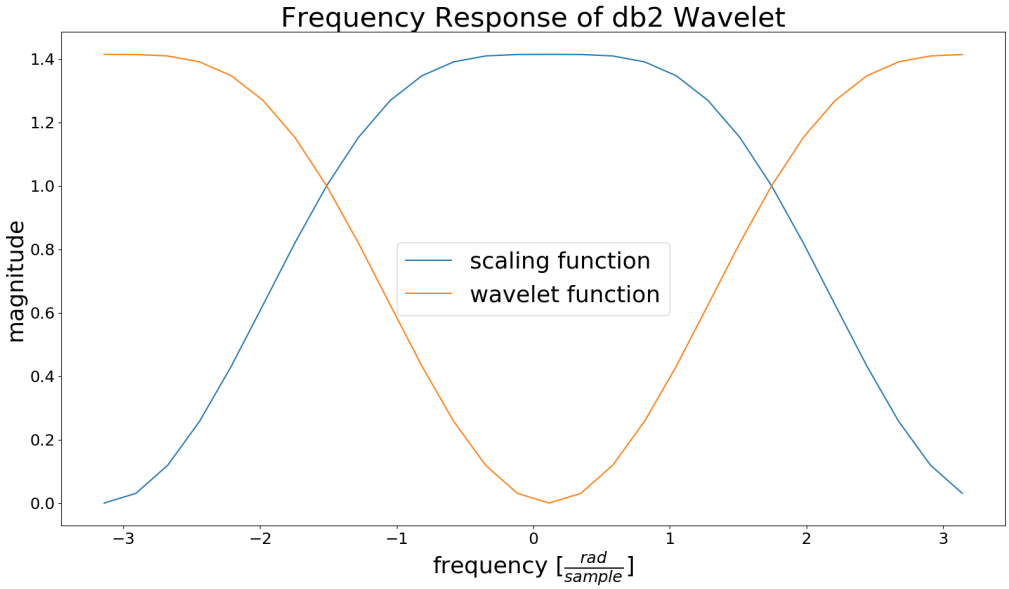


Figure 3.8: The frequency response of the db2 scale and wavelet functions.

the definition of the moment coefficient of skewness γ (equation 3.35)[69], this thesis uses the time-reversal asymmetry statistic displayed in equation 3.36, where τ is the lag [67].

$$\gamma = \mathbb{E}[(\frac{X-\mu}{\sigma})^3] = \frac{\mathbb{E}[(X-\mu)^3]}{(\mathbb{E}[(X-\mu)^2])^{3/2}} = \frac{\sum_{i=1}^{N}(x-\bar{x})^3}{(\sum_{i=1}^{N}(x-\bar{x})^2)^{3/2}} \tag{3.35}$$

$$t_{rev}(\tau) = \frac{\sum_{i=1}^{N-\tau}(x_{t+\tau}-x_t)^3}{(\sum_{i=1}^{N-\tau}(x_{t+\tau}-x_t)^2)^{3/2}} \tag{3.36}$$

Timing features are of interest because each node on a CAN bus will have its own oscillator which controls the timing of the signals it transmits. While the CAN protocol, of course, implements synchronization protocols on each bit so that nodes don't drift completely out of sync, it is believed that small variations in the true frequency of each oscillator will result in slightly different timing characteristics for messages from each node. To capture these differences, three different measures are used in this thesis: average bit length and average rising and falling edge derivatives. The average bit length is taken by summing the total number of points at a high or low bit value and dividing by the total number of bits in the message. The average rising edge derivative is calculated by taking the time and voltage difference between the first and last points on a rising edge to find a derivative and taking the average value of each of these for a given message (the falling edge derivative is calculated similarly). Note that the derivative is used rather than the rising or falling time to account for the finite sampling of the signal. With limited time resolution, it can't be

guaranteed where on an edge the samples will fall. If a sample falls at the very beginning of an edge, more of the edge will be captured (over more time) than if the first sample falls slightly later on the edge. The derivative takes the location of the sample on the edge into account by also measuring the voltage change between the first and last samples of an edge. This is illustrated in Figure 3.9. As it can be seen, due to the discrete sampling, different edge times would be calculated for each sample. To correct for that, the derivative is used, which is more stable to different sampling points.

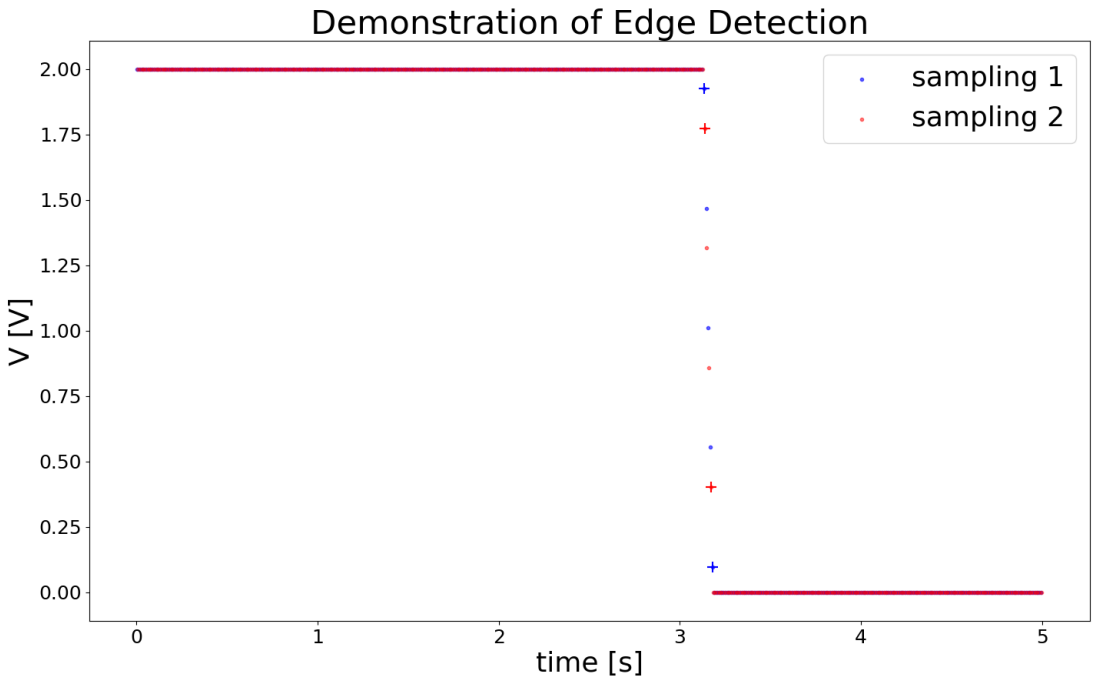


Figure 3.9: This figure shows two identical square waves sampled at different points. The points marked with + indicate the first and last edge points selected for the signal.

3.6 Classification

Once features have been extracted, classification must still be performed. To do this, we will rely on some learning algorithms to learn the differences between the populations of features for each node directly from collected data. There are many different algorithms that can be used to do this, as well as several things to be aware of when constructing and evaluating a learning algorithm. In this section, we will discuss a way of splitting the data into separate data sets, issues with overfitting, two powerful learning algorithms which will be used in

this thesis (support vector machines and multilayer perceptron neural networks), and the importance of selecting good features to learn from.

3.6.1 Data Sets

The most important thing for any machine learning algorithm is, of course, the data. There are a few things to consider with respect to datasets. The first one is to make sure that enough data is obtained to represent all possible cases. An algorithm can only learn what the data presents, and if the data is skewed to represent only a subset of possible cases, then so will be the learned function. Once the proper data has been obtained, it must then be broken up into different datasets. Generally, training, cross validation, and testing datasets are made with a 60-20-20 spread (60% of the data in the training set, and 20% of the data in each the cross-validation and testing sets). It is important to use three datasets rather than two because many algorithms have parameters that need to be tuned. If parameters are tuned using the training and testing datasets, they may slightly "overfit" the testing dataset, which will cause the algorithm to perform better on the testing dataset than can be reasonably expected for additional examples. Thus, it is important to use a cross validation data set to tune any parameters [70].

3.6.2 Bias and Variance

It can be shown that the error in any machine learning algorithm can be broken down into two parts - bias and variance. Bias in a learning algorithm represents an approximation error due to its inability to properly learn the appropriate data model. Bias occurs when an algorithm is not powerful enough to learn the true model of the data. This results in poor performance on both testing and training data. On the other hand, variance occurs due to the inadequacies of the data from estimation error in measurements. Variance is observed in an algorithm when a learned function is more complicated than the underlying model for the data and thus includes both the noise and the "true" data in its model. This leads to problems generalizing to additional data examples outside of the training set [70] [71]. For example, data was generated from a fifth-order polynomial with added Gaussian noise for $x \in [0, 3]$. Three models were fit to a random subset of 60% of these data points using a simple linear regression - a first-order model (a line), a fifth-order model, and a fifteenth-order model. As can be seen in Figure 3.10, the fifth-order model fits the data well in both the training and testing datasets; the fifteenth-order model fits the training data well, but does not generalize as well to the testing data; and the first-order model fails to fit either the training or testing data. This is a good example of the problems of bias and variance. Too weak of a learning function (the first-order estimate) gives a biased model that will not fit the data properly; while too powerful of a learning function (the fifth-order model) gives a model with high variance with respect to changes in the input data that will fit the testing

data extremely well, but will fail to generalize to additional examples.

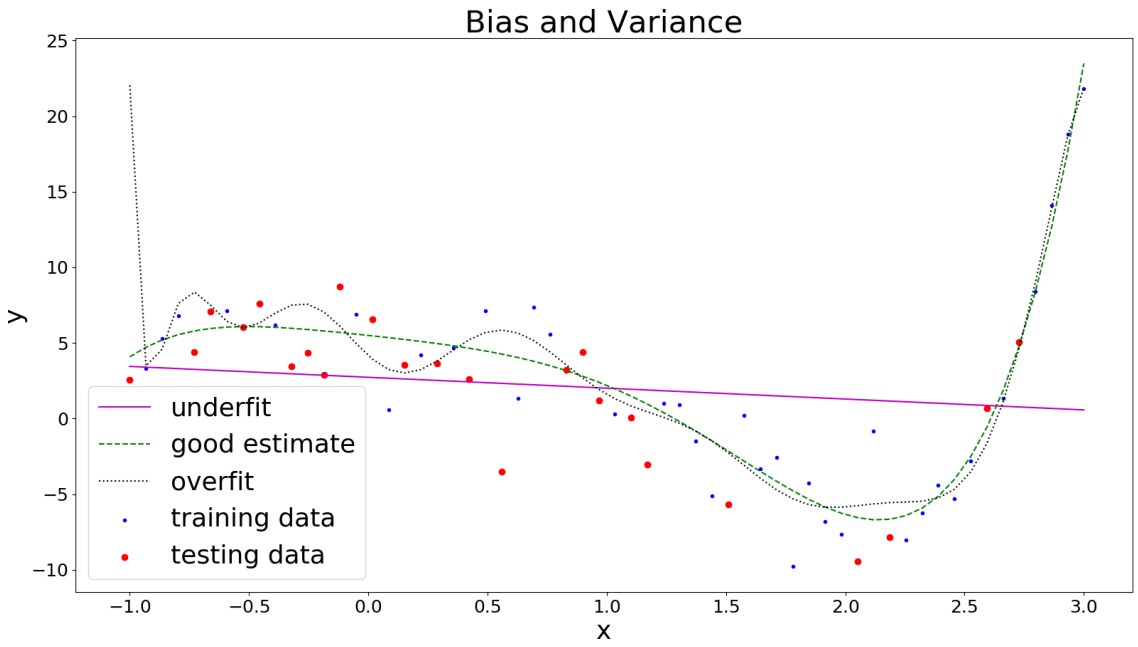


Figure 3.10: This plot gives an example of models with high bias and high variance, as well as a properly chosen model.

When dealing with high dimensional data that is difficult to visualize, picking the exact proper model for an algorithm can be an extremely difficult problem. To deal with the potential of overfitting, a few methods have been developed, such as regularization, weight elimination, and dropout. The idea is if several models fit the data approximately equally well, the simplest one will usually generalize the best. Regularization accomplishes this through an additional term in the cost function to keep the value of model weights low, as shown in equation 3.37, where w is a vector or matrix of weights [72]. Weight elimination, shown in equation 3.38, works similarly, but with the addition of a w_0, which can be selected as a sort of cutoff. For $w_i >> 0$, the value of the fraction approaches 1, while for $w_i << 0$, the value approaches 0 [73]. For neural networks, a method known as dropout is often used. This is done by randomly dropping a neuron in the network during training time with a specified probability p. This acts as an approximation for training 2^n different neural networks (where n is the number of neurons in the network). At testing time, a pseudo-average of each of these 2^n networks is performed by multiplying all outgoing weights of a neuron by its probability of being retained [74]. To demonstrate how these methods help prevent overfitting, Figure 3.11 shows the same plot from the previous paragraph, but with regularization performed on the fifteenth-order fit. It can be seen that this model no longer overfits the

data. In fact, this model matches the "proper" fifth-order model very closely. Similar ideas apply to classification learning, where an algorithm is learning decision boundaries from the data rather than fitting a curve to it.

$$J(\theta) = cost + \lambda \sum_{i \in \mathcal{W}} w_i \tag{3.37}$$

$$J(\theta) = cost + \lambda \sum_{i \in \mathcal{W}} \frac{w_i^2 / w_0^2}{1 + w_i^2 / w_0^2} \tag{3.38}$$

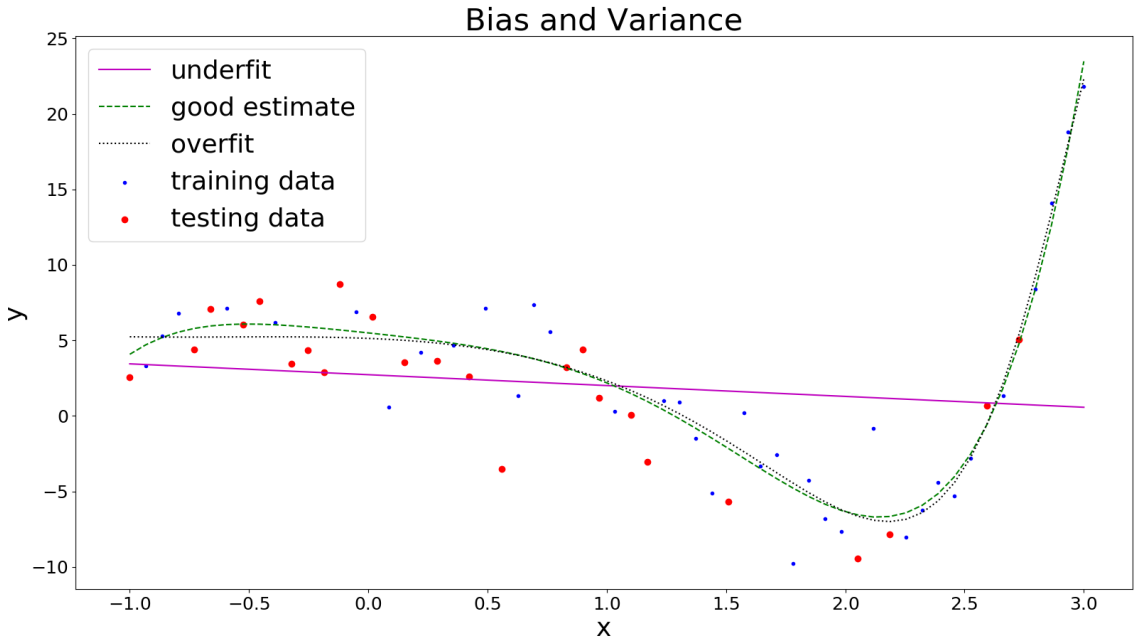


Figure 3.11: This plot gives an example how regularization can prevent overfitting.

3.6.3 Learning Algorithms

There are many different learning algorithms that allow for classification of data. The highlights include algorithms such as logistic regression, decision tree methods, support vector machines (SVM), and artificial neural networks (ANN). Support vector machines and neural networks are directly applicable to this thesis and will be discussed in the most detail, while logistic regression is discussed since it is fundamental to both SVMs and ANNs.

Logistic regression is a linear classifier which makes its decision based on a linearly weighted

sum of the inputs as shown in equation 3.39. It takes this sum and models the probabilities of possible outcomes using a logistic function, as shown in equation 3.40. The vector of weights θ is learned by minimizing a cost function. The most common cost function for classification is the log-loss cost function, shown in 3.41. A commonly used optimization method for minimizing the cost function is gradient descent. This algorithm involves computing the gradient $\frac{\partial J(\theta)}{\partial \theta}$ with each iteration, and attempting to step downward until a minimum has been reached. For a more complete explanation of gradient descent, see [75]. Logistic regression is the simplest and most fundamental of these algorithms, and, as it will be seen, most other algorithms build on these ideas.

$$z(x) = \theta^T x \tag{3.39}$$

$$h(x) = \frac{1}{1 + e^{-z}} \tag{3.40}$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} [y(log(h(x))) - (1 - y) log(1 - h(x))] \tag{3.41}$$

Support vector machines are designed to perform a sort of regression to select a decision boundary that maximizes the space between classes. They are also guaranteed to produce a convex cost function so that the global optimum is always acheived, making them a popular choice in machine learning. They accomplish nonlinear learning through the use of kernel functions, such as polynomials or a Gaussian function. The kernel function is used to compute new features by comparing the data point with landmarks that have been placed at each training example. These new features are used by the regression to find the decision boundary[5]. This is shown for Gaussian and polynomial kernels in equations 3.42a and 3.42b [76, 70, 77, 78].

$$f_j = e^{-\frac{||x - l||^2}{2\sigma^2}} \tag{3.42a}$$

$$f_j = (x\, l + c)^p \tag{3.42b}$$

Neural networks are another algorithm capable of learning nonlinear functions. Neural networks can come in many different sizes and architectures, each with its own advantages and disadvantages. This thesis will focus specifically on the multilayer perceptron (MLP) architecture. A neural network classifier estimates the posterior probability that an observation belongs to a certain class based on a series of nodes connected in a layer structure, where each node in one layer connects to each node in the next layer. An example of this structure is shown in Figure 3.12. It has been shown that neural networks with an arbitrary

[5]Note that a support vector machine with no kernel function (sometimes called a linear kernel) is equivalent to a logistic regression.

bounded and nonconstant activation function are universal approximators, and with a sufficiently smooth activation function, they are capable of arbitrary accuracy in approximating a function (assuming the network is sufficiently large) [79, 80]. In general, deep neural networks can learn functions with higher complexity than shallow ones with the same number of hidden units [81]. However, it is important to keep in mind that learning overly complex functions will cause the learning algorithm to overfit the training data, resulting in a high variance model and a loss of generality to additional examples. Thus, it is important not to build too deep a network (it's also more computationally expensive). On the flip side, less powerful learners are more likely to underfit the data, resulting in a high bias model and poor overall performance [71, 82]. Unfortunately, there is no one neural network architecture that can be easily selected and applied to any data set. Instead, learning algorithms must be selected and tuned based on the needs of individual datasets. A common thing to do is to experimentally search the space of learning architectures to select what works best.

Each node in an MLP performs a sort of logistic regression on its inputs with some nonlinear activation function applied to the result of a weighted sum of its inputs. In fact, the logistic function from equation 3.40, also known as the sigmoid, is a common choice for the activation function, along with the rectified linear unit shown in equation 3.43. For classification, we have a slightly different case for the output nodes, where the softmax activation function is generally used to predict $P[y = j|x]$ given K inputs, as shown in equation 3.44. The weights are learned to minimize error on the training examples using a method known as backpropagation, which tries to assign blame for wrong predictions to intermediate weights by backpropagating the error through the neural network structure. The error assigned to the outer layer L is shown equation 3.45a, while the error for an earlier layer is shown in equation 3.45b. Here, a_j represents the "activation" (output) of the j^{th} neuron, the superscript $^{(l)}$ represents the l^{th} layer, and the matrix of learned weights mapping the l^{th} layer to the $(l+1)^{th}$ layer is represented by $\theta^{(l)}$. The matrix $\theta^{(l)}$ is s_{l+1}x(s_l+1), where s_l is the number of neurons in layer l (the $+1$ in $s_l + 1$ is for the bias term in each layer - essentially, the constant term in each layer's "regression"). These are related to the cost function gradient for use in gradient descent by equation 3.46 [70].

$$a(z) = \begin{cases} 0 & : z \leq 0 \\ z & : z > 0 \end{cases} \tag{3.43}$$

$$P[y = j|x] = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{3.44}$$

$$\delta^{(L)} = a^{(L)} - y^{(L)} \tag{3.45a}$$

$$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} (a^{(3)}(1 - a^{(3)})) \tag{3.45b}$$

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^{l}} = a_j^{(l)} \delta_i^{(l+1)} \tag{3.46}$$

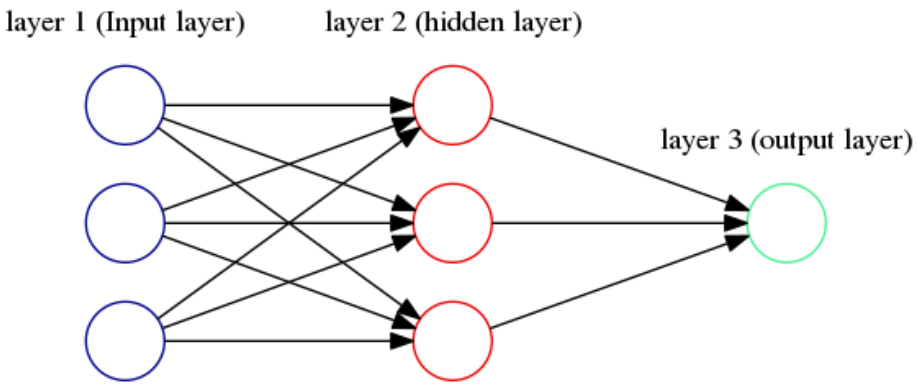


Figure 3.12: This shows an example of an MLP architecture neural network.

Many machine learning algorithms are built for classifying binary cases (eg, SVM). To overcome this, the strategy one versus rest is commonly used. The one versus rest strategy employs one classifier per class, which fits that class against all the other classes. The class with the highest probability over all other classes is chosen as the predicted class. Advantages of this method include computational efficiency (only n_c classifiers are needed) and interpretability (it is possible to gain knowledge about any class by inspecting its corresponding classifier) [83, 70].

3.7 Feature Selection

No machine learning algorithm can work without good features providing the necessary information for classification. However, even if a dataset contains good features, it is important to do some analysis to ensure that all features passed to a learning algorithm are relevant[6]. Passing extraneous features to an algorithm increases computational complexity and the likelihood of getting stuck in a local optimum rather than the global optimum. Additionally, it has been shown that using only a small subset of relevant features significantly reduces the necessary number of data points to guarantee good generalization for the learned function [84]. Of the used features, it is often beneficial to avoid repeating information with two highly similar features, especially for neural network classifiers [71].

There are three different types of feature selection algorithms - wrapper, embedded, and

[6]A feature x_i is often described as a strongly relevant feature if two examples with non-zero probability from different classes differ only in their assignment to x_i. A feature x_i is referred to as weakly relevant if it is possible to remove a subset of features such that x_i becomes strongly relevant.

filter methods. Wrapper methods search the feature space by repeatedly training and testing the algorithm on different subsets of features. Depending on the size of the feature space, there are different approaches that can be taken. In general, testing all possible combinations of features would be intractable since that would be 2^f possible combinations (for f different features). Thus, heuristic methods such as greedy algorithms[7] must be used. An example would be a forward search (start with 0 features and add a feature each time) where an added feature is kept if it increases the algorithm's performance and discarded otherwise. Since this method tests the features directly on the algorithm, it often gives the best performance for any specific algorithm; however, it is also the most computationally intensive since the full algorithm is trained and tested at each step. Embedded feature selection embeds the feature selection in the learning algorithm itself. For example, weight pruning can be done to essentially eliminate features that are given sufficiently low weights by the learning algorithm. The third method, filtering, is generally the cheapest method computationally. Filter methods apply some sort of statistical test or criteria to the features. These methods are more general than wrapper or embedded methods, so they generally give slightly worse performance for a specific learning algorithm. Thus, they are generally done to provide a feature ranking or subset to be passed to a wrapper or embedded method to reduce the search space and computational demand [84].

[7] A greedy algorithm attempts to search a space by making a locally optimal decision at each step.

Chapter 4

Process

4.1 Equipment and Setup

The idea behind this thesis is to identify the node that is sending a message and compare this with known information about which node(s) should be sending this message (ie, is it that ECUs job to control this feature or provide this information). Since the purpose of the message is contained in the message identifier, an experiment was conducted to see if a node could be recognized from the way it sends a specific identifier. This way, the underlying message content of the signals being compared is always the same, and it can be determined if the proper node is sending a message from the content in the message identifier.

For this thesis, a 3 node CAN bus constructed from Maxon EPOS2 50/5 positioning controllers (part number 347717) was used to generate data for testing. This bus uses the CANopen protocol, so each node has a configurable four-bit identifier associated with it that is part of the CAN message identifier. For the data collection, data relevant to each node sending the identifier of interest was collected one at a time. In other words, at any point of data collection, one node on the bus had the given identifier, and all messages of the same type with the same full CAN message identifier were kept for processing. The measurements were made using the external CAN connector holes that are on each node.

A Picoscope 5442A from Picotech was used to collect data on the bus from both the CANH and CANL using two (simultaneous) channels. The wiring diagram of this set-up is shown in Figure 4.1. This figure shows the Picoscope attached to an external CAN connector on node 1, where it was plugged in for most of the data collection. However, as will be described in section 4.2, some data was also collected with the Picoscope plugged into the available CAN connector on node 3. Additionally, between some collections, the Picoscope was unplugged and plugged back in to the CAN connector, but the same P6060 probes, which came with the Picoscope, were used for all data collection. The wiring of the CAN lines was left alone,

and the cables experienced minimal, if any, jostling or movement. The cable between nodes 1 and 2 is of length 37.5 cm, and the cable between nodes 2 and 3 is 41 cm. A picture of the CAN bus, with connected probes, is shown in Figure 4.2.

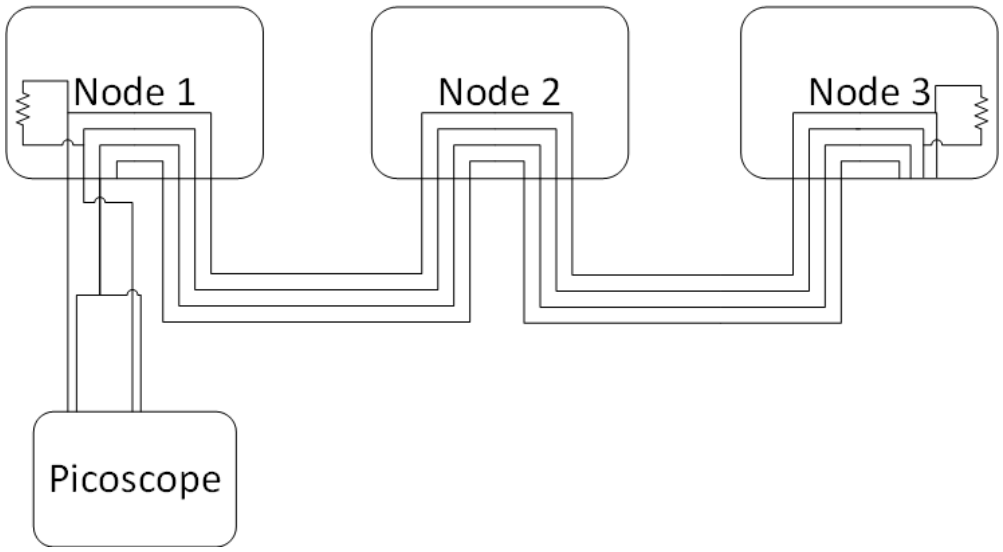


Figure 4.1: A schematic showing the wiring diagram and data collection for most of the data. However, part of the testing set was measured from node 3 instead of node 1.

4.2 Data Collection

To perform the classification, data must be collected to train and evaluate the classification algorithm. To do this, data must be collected in some sort of controlled environment where it is known for sure which node is sending at any given time. For this thesis, 4650 unique messages were initially captured and used (from the CAN connector on node 1), with exactly a third coming from each node. The data was collected at 125 MHz with 15 bits of resolution between ± 5 V. This was done over multiple instances in the span of one month to ensure a more robust data set and that any time-varying elements of the noise would be captured in the data. The data was split up in a roughly 80-20 spread (3690 for testing and cross-validation, and 960 for testing) such that an even spread of data for each node was maintained for each data set. After another three months had transpired, 1950 additional data points (650 from each node) were collected by plugging in to the CAN connector on on node 3. These data points were added in to the test data set. A small section of 11 bits of each of these messages (in this case, the message identifier) was selected for use in the algorithm. The underlying digital content of those 11 bits was the same for all 6600 messages; thus, we ensure that any differences we detect are a result of differences between the nodes and not differences between the message.

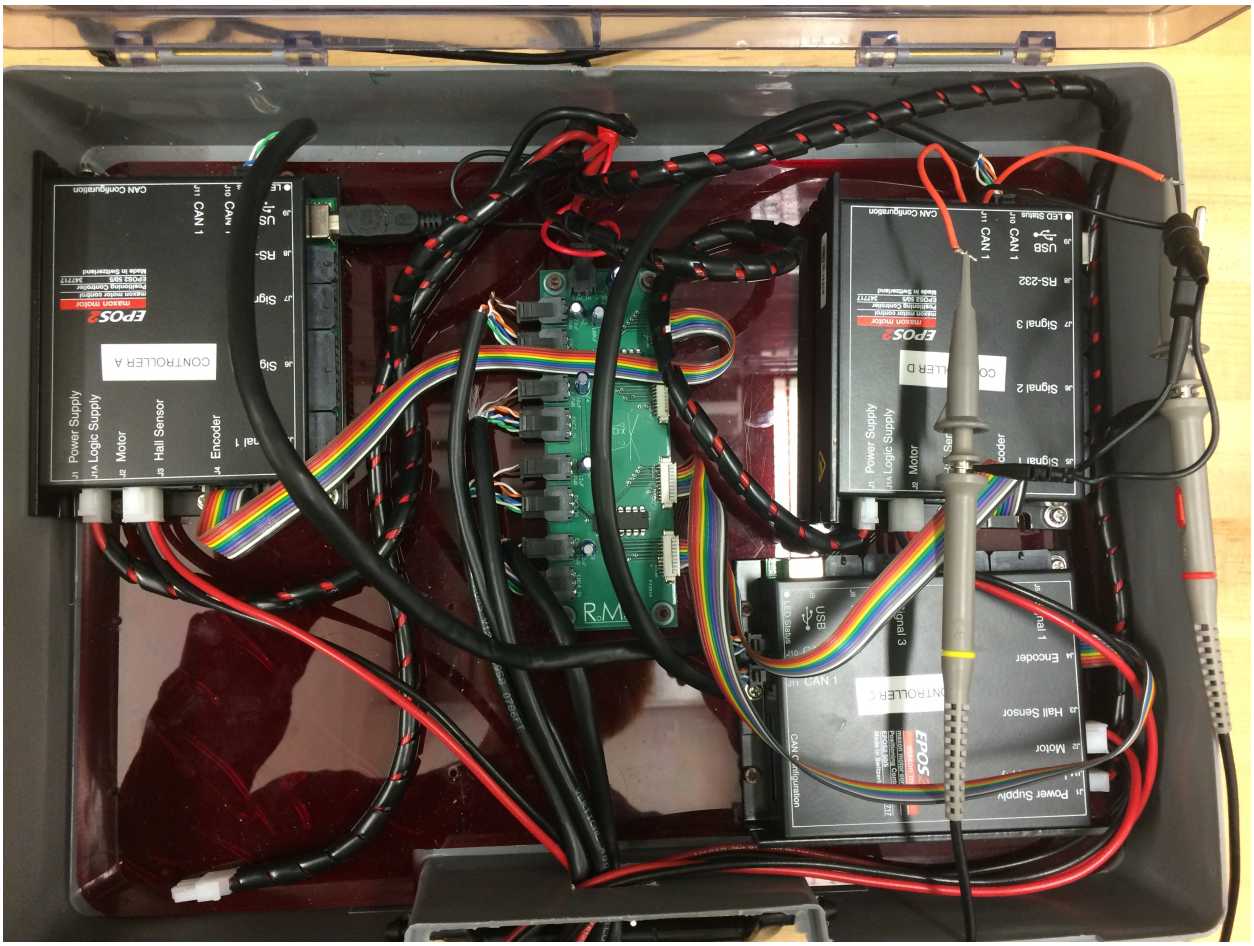


Figure 4.2: A picture of the CAN bus is shown here. The Picoscope probes in this picture are attached to node 3 in the top right.

4.3 Preprocessing and Feature Extractions

Once the data has been collected, preprocessing must be applied to the data before classification can be performed. This section will go through all of the preprocessing that is done on the data before extracting any features, as well as the features that were extracted.

The first step in the preprocessing is to read the bits of the message to find the appropriate section to use for the classification. Additionally, for a real-time implementation, this would also be necessary for identifying the message and determining expected sources. Once this has been identified, features that require the original signal are extracted. These are the average bit length, average rising edge derivative, and average falling edge derivative. To extract the average bit length, the signal is broken up according to where bit edges are. Then, the time length of the flat areas are divided by the theoretical bit times for 1 Mbit/s

($1\,\mu s$) to get a 'time per bit'. These values are then averaged to get the overall average bit length for the signal. For the rising and falling edge derivatives, the first and last values detected to be a part of the edge are used. The derivative is taken to be the voltage difference over the time difference for these two points ($\frac{\Delta V}{\Delta t}$). This is done separately for each line.

The next step is to separate the signal into 2 components - a pure signal component and a noise component. To do this, the CANH and CANL lines are tracked using a simple Schmitt trigger technique, and a noiseless copy of each signal is constructed using the theoretical voltage values for high and low bits on each line (CANH and CANL). For the bit edges, the noiseless signal simply follows the voltages exactly since it is difficult to split this up into signal and noise components. Once this noiseless copy of the signal has been constructed, the final 'signal' and 'noiseless signal' are defined to be the difference between the CANH and CANL lines in order to take advantage of the common mode rejection of the CAN bus architecture. The noise component is then found to be the difference of the noisy and noiseless signals. An example of the resulting noise component is shown in Figure 4.3.

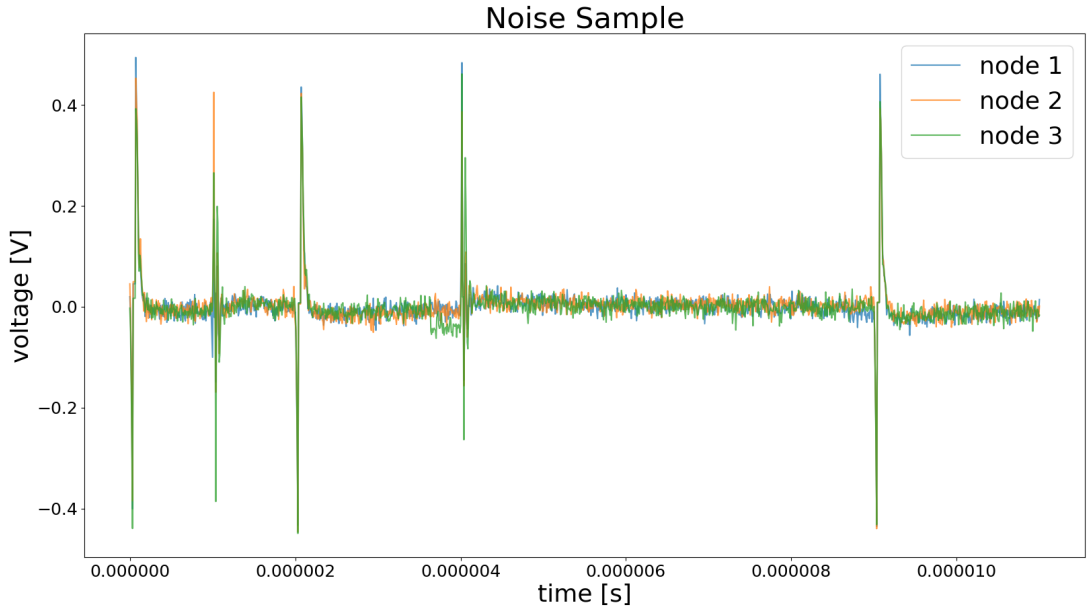


Figure 4.3: An example of the noise measured in one of the CAN bus nodes used for this thesis.

With the noise component separated, the rest of the features can be extracted from the noise. The entirety of the extracted features are summarized with brief descriptions in Table 4.1. These features and reasons for choosing them were discussed in Chapter 3. The order for the autoregressive model was determined by fitting models of order 1 through 25

to the samples in the testing data set, and the best order was selected based on Akaike's information criterion.

4.4 Feature Selection

For the feature selection, a combination of filter and wrapper methods were used. The features were ranked based on a custom scoring method, and then a simple forward search was performed using the top performing features.

The feature ranking was performed based on performance from two statistical tests: the independent samples t-test, and the Kolmogorov-Smirnov test. The independent samples t-test is chosen because it is an often used location test with good statistical power[1] for normally distributed data and is the preferred test for analysis of variance. The Kolmogorov-Smirnov test was chosen because it makes virtually no assumptions, meaning that it should be reliable for almost any data, and the author believes that its test statistic is a crucial requirement for data classification since a large difference between two cdfs indicates good separability of the data, which makes classification easier. Since it is desired that each class population be different from all other populations, a multiple comparison methods is used, and p-values from different tests are summed since each value is already on the same scale ($p \in [0, 1]$). Furthermore, since consistent classification performance requires consistent data, the data set is split in half, and each class is also compared with itself. The multiple comparison score is found according to equation 4.1, where $t_{i,j}$ is the p-value from a t-test comparing class i with class j, and $k_{i,j}$ is the p-value from the same Kolmogorov-Smirnov test. Thus, a high score represents a feature with a population that is both consistent within its class and different between classes. However, since many machine learning algorithms, including SVMs, naturally perform binary classification and are typically extended to multiclass classification through one-versus-rest methods, a one-versus-rest scoring system was also developed, and its scoring function is shown in equation 4.2, where t_{i,i^C} represents the p-value from a t-test between the data for class i and all data not in class i. Additionally, since highly correlated features likely don't provide significant new information, the feature ranking is kept from becoming clogged up with similar features by implementing a correlation check. After each feature is scored, the correlation matrix for the feature matrix is calculated. When two features are found to have correlation $\rho > 0.9$, the higher-scoring feature is kept, and the score of the lower-scoring feature is set to 0.

$$score = \frac{\sum_{i=1}^{n}(t_{i,i} + k_{i,i})}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(t_{i,j} + k_{i,j})} \tag{4.1}$$

[1]Statistical power represents the probability of correctly rejecting the null hypothesis when the alternative hypothesis is true.

FFT	This is a frequency estimate made purely by taking the FFT of the data with a Chebyshev window ($\alpha = 4.5$)
Welch Spectral Density Estimate	This is a frequency estimate made using Welch's method with a Hamming window.
Autoregressive spectral estimate	An estimate of the spectral content of a signal done by constructing a 12^{th} order autoregressive model. This method assumes a nondeterministic, weakly stationary signal.
db2 detail coefficients	The detail coefficients from a single layer discrete wavelet decomposition using Debauchies 4 tap wavelet. The detail coefficients come from the convolution of the wavelet function with the data.
db2 approximation coefficients	The approximation coefficients from a single layer discrete wavelet decomposition using Debauchies 4 tap wavelet. The approximation coefficients come from the convolution of the scale function with the data.
time-reversal asymmetry statistic	The time-reversal asymmetry statistics for lags 1-5 are extracted. This measure is similar to a skewness measure for the lagged differential signal.
autocovariance	The autocovariance of the data for all lags is calculated.
average rising edge derivative	The average derivative of a rising edge in the signal.
average falling edge derivative	The average derivative of a falling edge in the signal.
average bit length	The average length of 1 bit of a signal in the message.
noise variance	The variance of the noise estimate for the signal.
absolute sum of difference	The sum of the absolute value of the difference between successive noise values.
percentage of increasing difference	The percentage of positive values in the vector of differences between successive noise values.

Table 4.1: This table includes all the features extracted for analysis by the algorithm.

$$score = \frac{\sum_{i=1}^{n}(t_{i,i} + k_{i,i})}{\sum_{i=1}^{n}(t_{i,i^C} + k_{i,i^C})} \tag{4.2}$$

With this feature ranking, simple forward searches are used to perform the final feature selection for each machine learning algorithm. Each algorithm is initially run only on the highest performing feature, and then features are added one at a time in order of performance, and the algorithm is again run on each new feature set. A five-fold cross validation scheme is used, and the algorithm's performance is measured according to its average performance on the cross-validation dataset, as measured by the log-loss cost function. The first feature is selected if it can provide greater than 33% classification (in other words, better classification than random guessing), and subsequent features are selected if the loss function decreases. Selected features are kept in the feature set, while features not meeting this criteria are discarded from the selection process. This is iteratively done to test the top 100 scroring features from each algorithm. First, the top 100 features from the multiple comparison method are used (in order from highest to lowest scoring), since this method gives the features which are most different between each class. Then, the top 100 features from the one-versus-rest scoring method are passed to the search since this method is more likely to give features which have some similar classes, but at least one class that is significantly different from the other two (which will help with classification of data coming from said class).

4.5 Classification

There are many algorithms which can be used to perform the classification, as discussed in Chapter 3. This thesis picks two of the more popular and powerful algorithms, which are the support vector machine (SVM) - as implemented in scikit-learn [85] - and multilayer perceptron (MLP) neural network - as implemented in Keras [86]. The SVM is chosen because it guarantees to find the globally optimum solution according to its parameters and it is resistant to overfitting, while the MLP is chosen for its ability to learn more complex functions. Both of these algorithms are tested (including separate feature selection), and the results can be compared.

Each algorithm has some hyperparameters that must be tuned. This tuning is done initially with the ten highest ranking features from the feature filtering method to get approximate solution, and another grid search is performed on the final feature set to ensure best performance. To tune these parameters, a grid search with five-fold cross validation is implemented using scikit-learn's API [85]. For the SVM, the parameters tuned include the error penalty C for the cost function (essentially an inverse regularization coefficient) and the kernel. Additionally, since some kernels themselves have hyperparameters (the degree of the polynomial and $\gamma = \frac{1}{2\sigma^2}$ for the gaussian kernel), these are included in the search. The values searched are shown in table 4.2. For the MLP, the optimization algorithm, amount of

C	0.25, 0.5, 1.0, 1.5, 2.0, 2.5, 3.3, 5.0
kernel function	polynomial and gaussian
polynomial degree	2, 3, 4, 5
γ	0.025, 0.05, 0.075, 0.10, 0.15, 0.20, 0.25, 0.33, 0.4, 0.5, 0.67, 0.75, 0.9, 1.0

Table 4.2: A table containing the values used for the grid search to determine the best values for the hyperparameters of the SVM.

neurons in layers 1 and 2	(64, 32), (32, 32), (64, 64), (32, 64), (48, 32), (48, 48), (32, 48)
optimization algorithm	stochastic gradient descent, adam, adamax, adagrad, adadelta, nadam, RMSprop
dropout	0.4, 0.5, 0.6
batch size	8, 16, 32
training epochs	10, 25, 50

Table 4.3: A table containing the values used for the grid search to determine the best values for the hyperparameters of the MLP.

dropout, activation function, number of neurons per layer, number of training epochs, and batch size are are included in the grid search. The values for these are shown in table 4.3.

Chapter 5

Results

5.1 Feature Extraction

In this section, the examples of measured noise shown in Figure 4.3, will be used to show example results of each feature extraction outlined in section 4.3 will be shown.

The autocovariance of the signals are claculated and plotted in Figure 5.1. It does seem to smooth ot the noise some, but differences between the nodes are not obvious, and no features from the autocovariance are chosen.

The Chebyshev-windowed FFT is taken and shown in Figure 5.2. As it can be seen, there is quite a bit of noise in the estimate. Thus, while it looks by eye like there may be some points of differences in the estimate, it is desirable to find an estimate with less noise.

Welch's spectral estimate is calculated and shown in Figure 5.3. Again, the estimate is quite noisy. From this, it was hypothesized that, in order to get an estimate which was less noisy, a different approach would need to be taken.

The autoregressive spectral estimate is then calculated. The autoregressive order is found by fitting autoregressive models of order $1 - 25$ to each of the training samples, and the best fit is determined using Akaike's information criteria. The average value of these best fit orders is taken, and found to be 12. The resulting spectral estimate is shown in Figure 5.4, and the autoregressive model fits for each node are overlayed with the noise data in Figures 5.5, 5.6, and 5.7 to demonstrate model fit. It can be seen that it follows a similar underlying pattern as the FFT, but it provides a significantly smoother representation since it is based only on the coefficients of a 12^{th} order autoregressive model.

The wavelet transformations are calculated using Daubechies 4 tap wavelet. The resulting

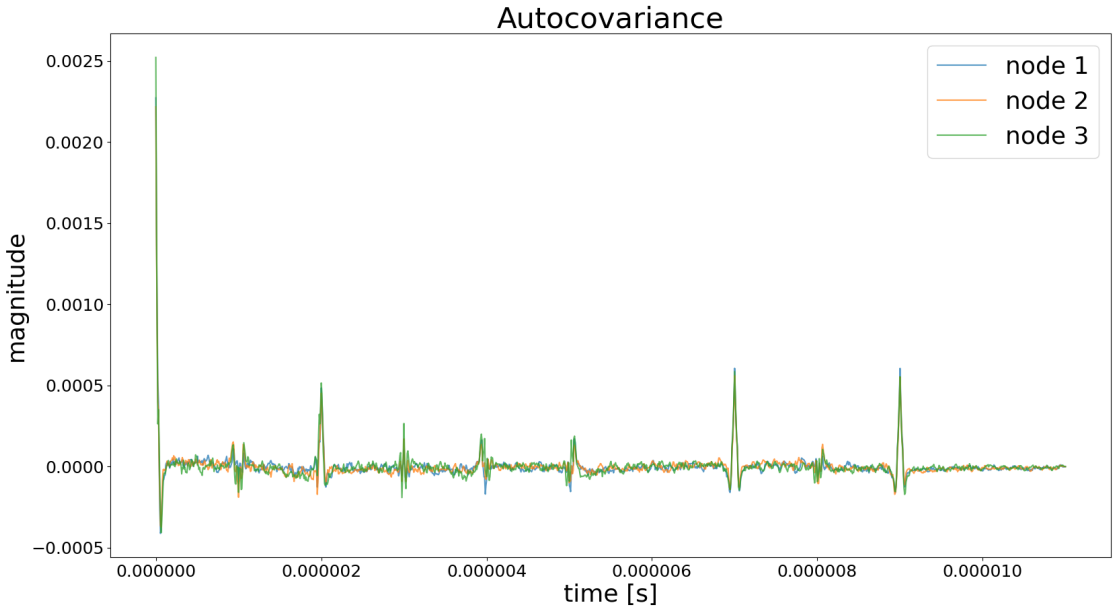


Figure 5.1: This plot shows an example of the calculated frequency spectrum from the Chebyshev-windowed FFT.

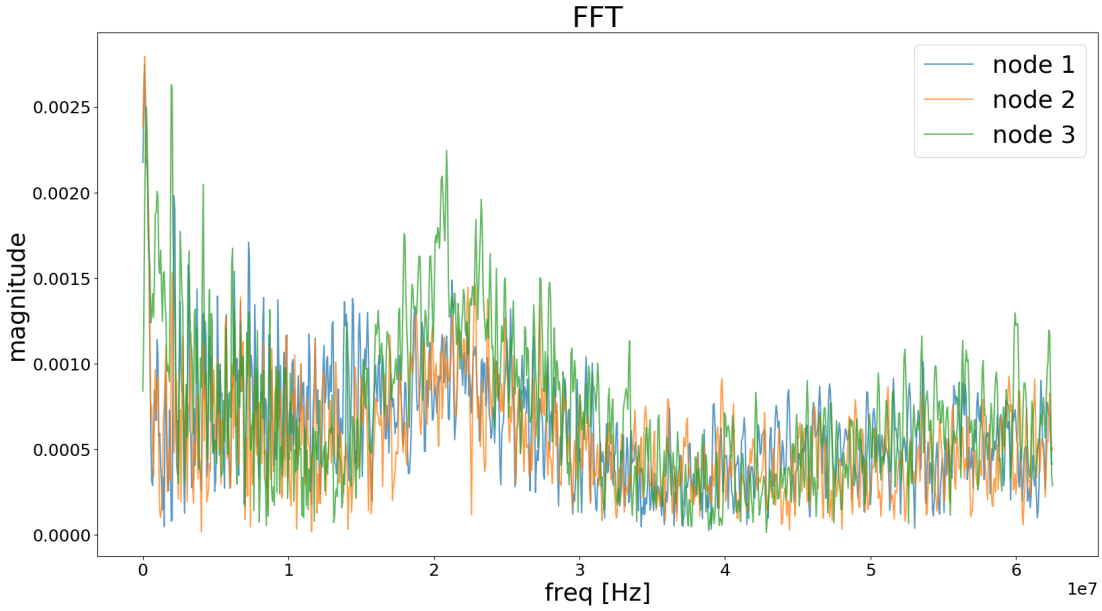


Figure 5.2: This plot shows an example of the calculated frequency spectrum from the Chebyshev-windowed FFT.

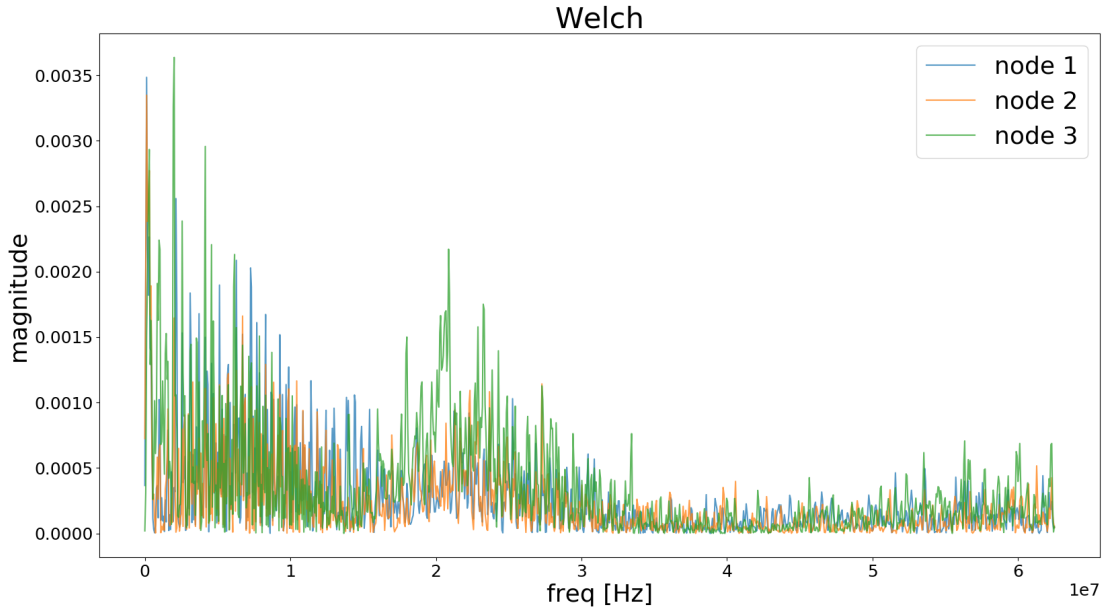


Figure 5.3: This plot shows an example of the calculated Welch estimate of the frequency spectrum.

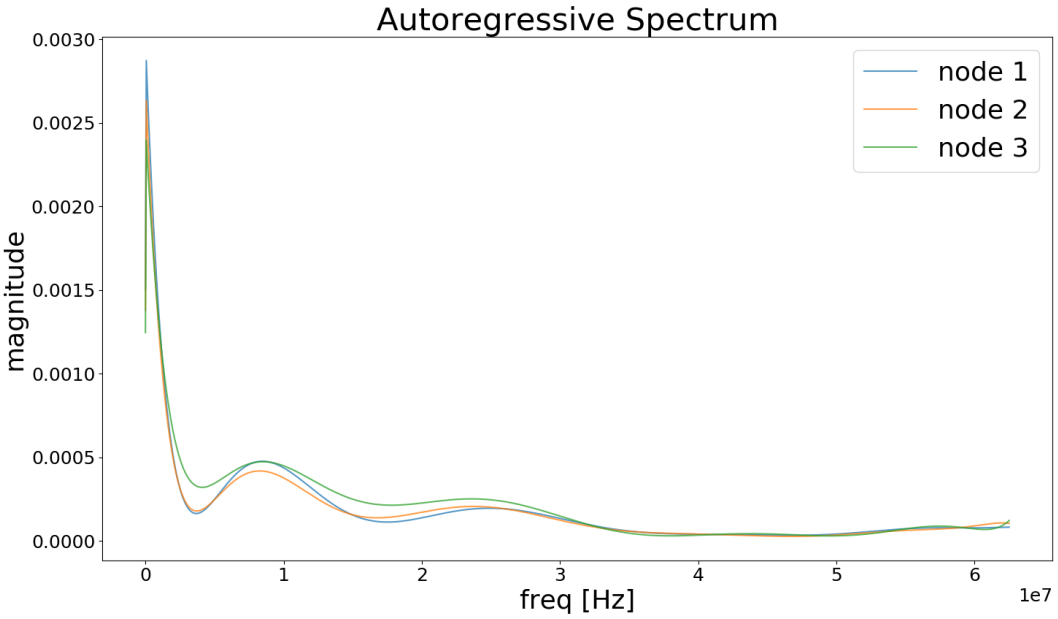


Figure 5.4: This plot shows an example of the calculated frequency spectrum from autoregressive model.

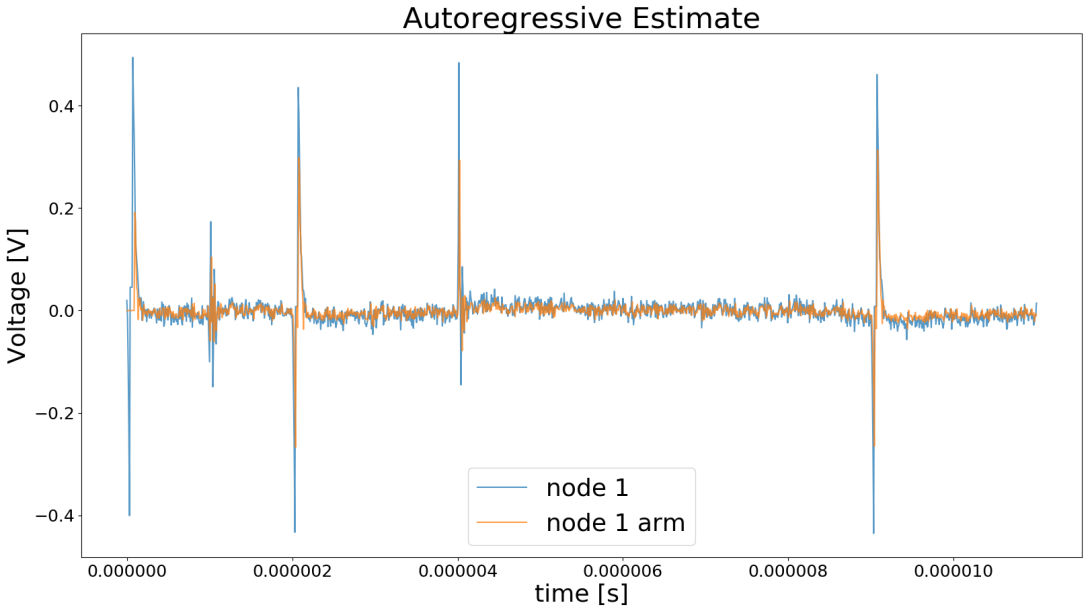


Figure 5.5: This plot shows the autoregressive fit for the noise sample for node 1.

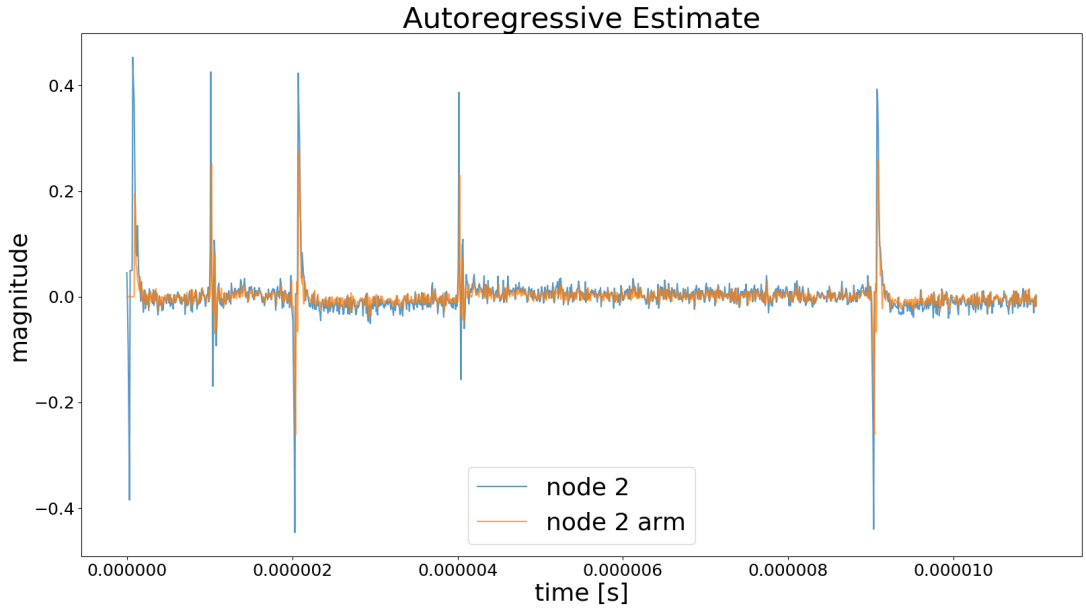


Figure 5.6: This plot shows the autoregressive fit for the noise sample for node 2.

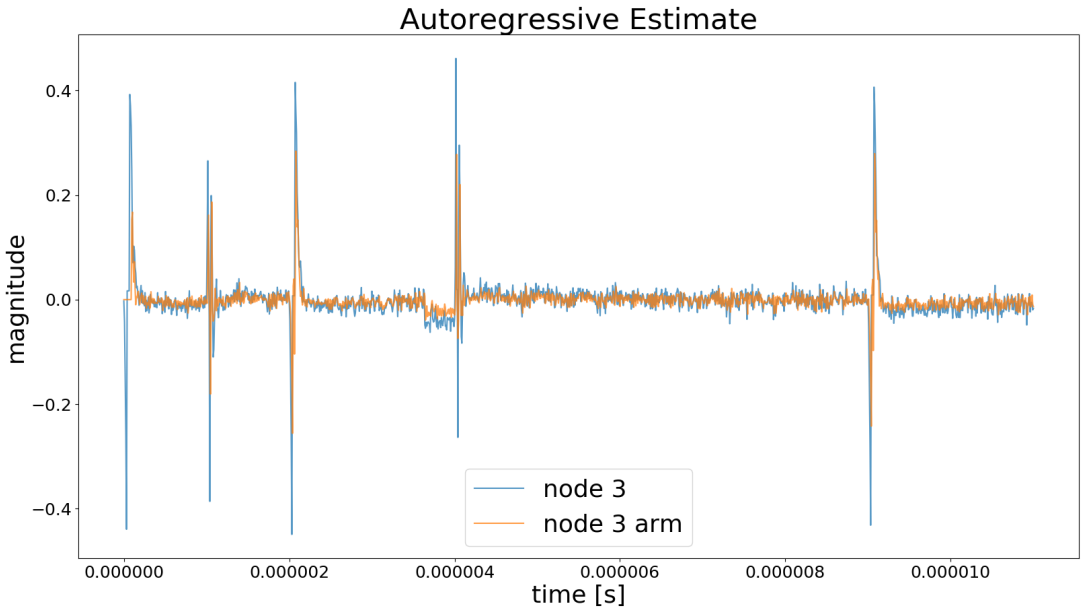


Figure 5.7: This plot shows the autoregressive fit for the noise sample for node 3.

coefficients are plotted across the data's timespan in Figures 5.8 ans 5.9. While differences are difficult to see on the graph, we will see in the next section, that there are some local differences between nodes.

The time-reversal asymmetry statistics for lags 1-5 are calculated and plotted in Figure 5.10. It can be seen that, at least for this example, there are differences between the nodes. While it's not chosen by either algorithm, it does show up in the the output of the feature filter.

The additional features calculated include average rising edge derivative, average falling edge derivative, average bit length, noise variance, absolute sum of difference, and percentage of increasing difference and are shown in Table 5.1. The only one of these feaures that is chosen is the average rising edge derivative.

5.2 Feature Selection

For the feature selection, 18 total features were selected for the MLP and 9 features were selected for the SVM. These features are listed along with their scores from the feature rankings in Table 5.2. Histograms for each of these features are plotted in Appendix A.

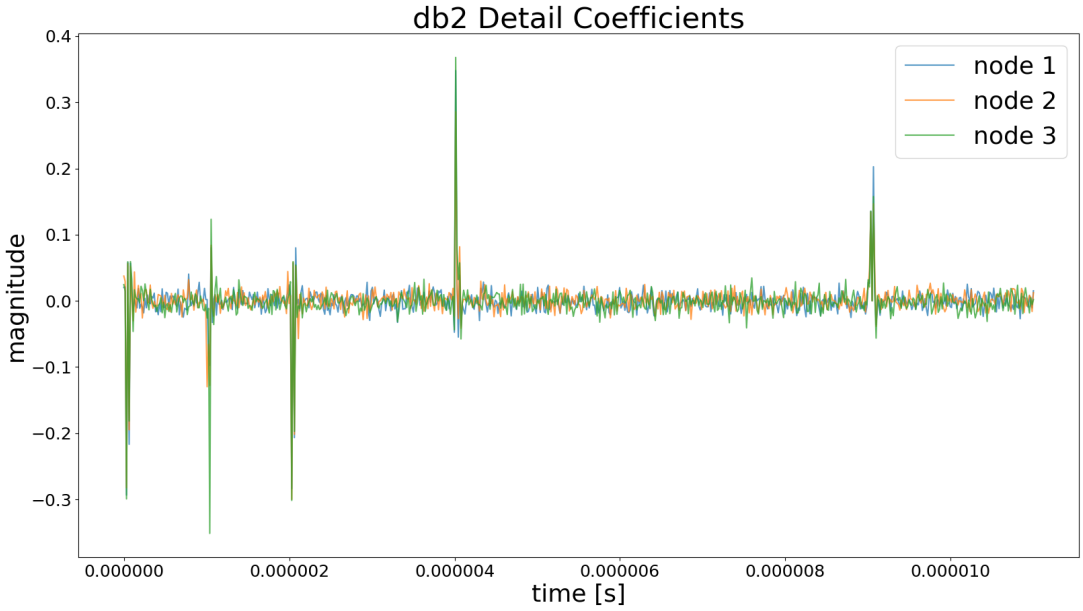


Figure 5.8: This plot shows the calculated detail coefficients for the sample noise.

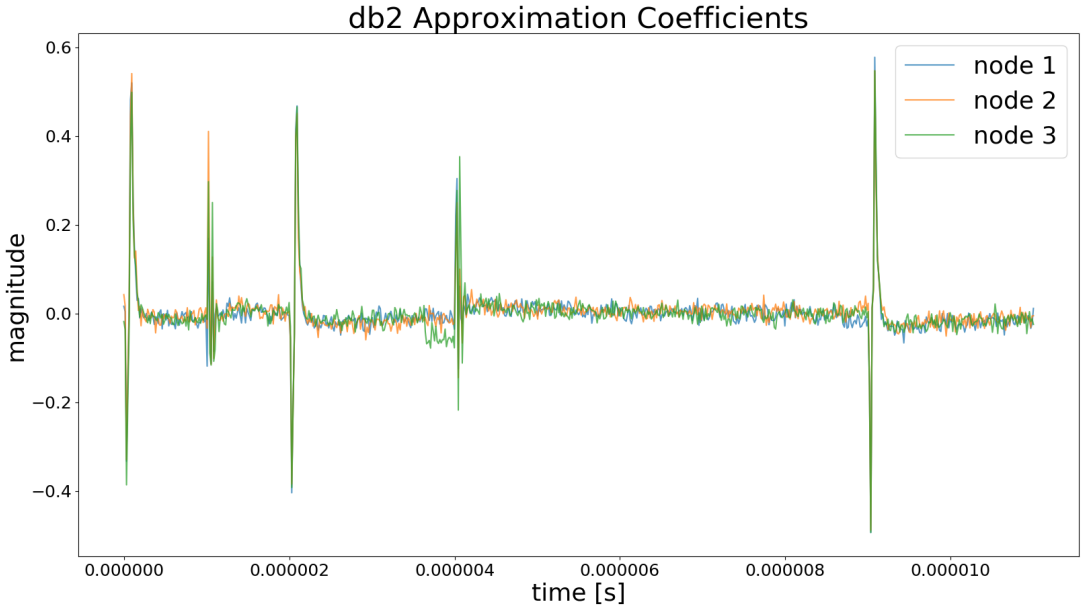


Figure 5.9: This plot shows the calculated approximation coefficients for the sample noise.

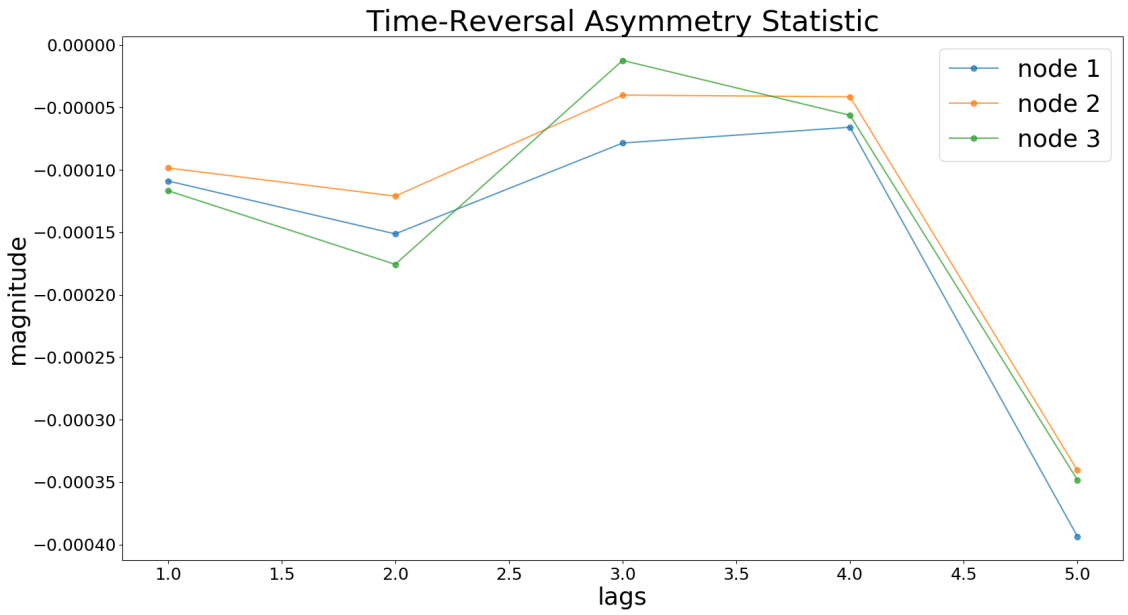


Figure 5.10: This plot shows the time-reversal asymmetry statistics for the first five lags.

Feature	Node 1	Node 2	Node 3
average rising edge derivative	3.2543e+07	3.3738e+07	3.8457e+07
average falling edge derivative	-1.3386e+08	-1.4648e+08	-2.1341e+08
average bit length	1.0034e-06	1.0036e-06	1.0016e-06
noise variance	2.2743e-03	2.2188e-03	2.5241e-03
absolute sum of difference	2.3731e+01	2.4080e+01	2.6140e+01
percentage of increasing difference	6.8800e+02	6.6900e+02	6.6000e+02

Table 5.1: This table shows features extracted from the example noise samples.

Feature	Algorithms	MC Ranking	OVR Ranking
Autoregressive PSD 14.28 MHz	SVM and MLP	1	4301
db2 approximation coefficient for $t = 4.096\,\mu s$	SVM and MLP	3	353
Autoregressive PSD 17.15 MHz	MLP	4	4682
FFT magnitude for 12.08 MHz	MLP	9	730
FFT magnitude for 13.06 MHz	MLP	10	634
FFT magnitude for 11.05 MHz	MLP	13	736
FFT magnitude for 12.51 MHz	SVM and MLP	16	174
FFT magnitude for 18.01 MHz	MLP	17	4401
db2 approximation coefficient for $t = 1.088\,\mu s$	SVM and MLP	4405	4
FFT magnitude for 11.96 MHz	MLP	613	8
FFT magnitude for 1.831 MHz	MLP	238	10
FFT magnitude for 10.99 MHz	MLP	1120	12
db2 approximation coefficient for $t = 2.240\,\mu s$	MLP	3987	18
FFT magnitude for 2.075 MHz	MLP	118	21
FFT magnitude for 11.90 MHz	MLP	182	22
FFT magnitude for 13.37 MHz	MLP	524	23
FFT magnitude for 10.31 MHz	MLP	678	37
FFT magnitude for 11.84 MHz	MLP	448	39
Time-reversal asymmetry statistic for lag 4	SVM	175	1
FFT magnitude for 2.197 MHz	SVM	212	6
FFT magnitude for 12.82 MHz	SVM	522	18
FFT magnitude for 21.06 MHz	SVM	408	84

Table 5.2: This table includes all the features used by either algorithm. The ranking by each filter algorithm is also shown (MC Ranking stands for multiple comparison ranking and OVR Ranking stands for one-versus-rest ranking).

5.3 Classification

The best accuracy achieved by the MLP is 99.9% accuracy on the test data set, while the SVM maxes out at 99.8%. Graphs of the classification accuracy versus number of features are shown in Figures 5.12 and 5.11 for the MLP and SVM, respectively, and the confusion matrices for each are shown in Tables 5.3 and 5.4. The parameters used for each of these models are the parameters found to be best from the grid search mentioned in Chapter 4. For the SVM, these are: a gaussian kernel function with $\gamma = 0.075$ and a penalty parameter of $C = 5$ for the error term in the optimization function. For the MLP, these are: 3 layers (plus the input layer) with 64, 32, and 3 (for three classes) neurons, respectively; the Adam optimization method [87]; a dropout of 0.5 for the first 2 layers; the rectified linear activation function for the first two layers (with softmax activation for the classification layer); 50 epochs for training; and a batch size of 32.

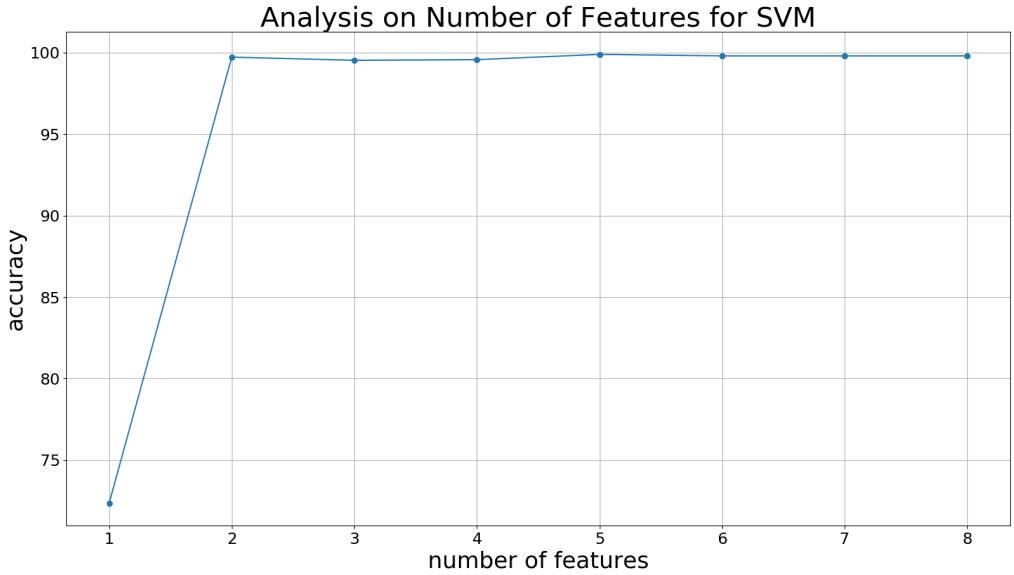


Figure 5.11: A plot showing how the accuracy of the SVM changes as features are added (only including those features chosen as the final feature set for the SVM).

However, as it can be seen from the above plots, classification accuracies in the high 90% are acheived by both algorithms after only 2 features. Thus, if desired, the algorithms could be constructed using only 2 features to save on computation. To find the best set of two features, an exhaustive search was done for each algorithm over all combinations of 2 features from the features in Table 5.2. The feature set with the lowest log loss from a five-fold cross validation was selected. This resulted in a reasonably wide spread of log loss values, including some pretty high losses. The best combination of 2 features ended up being the

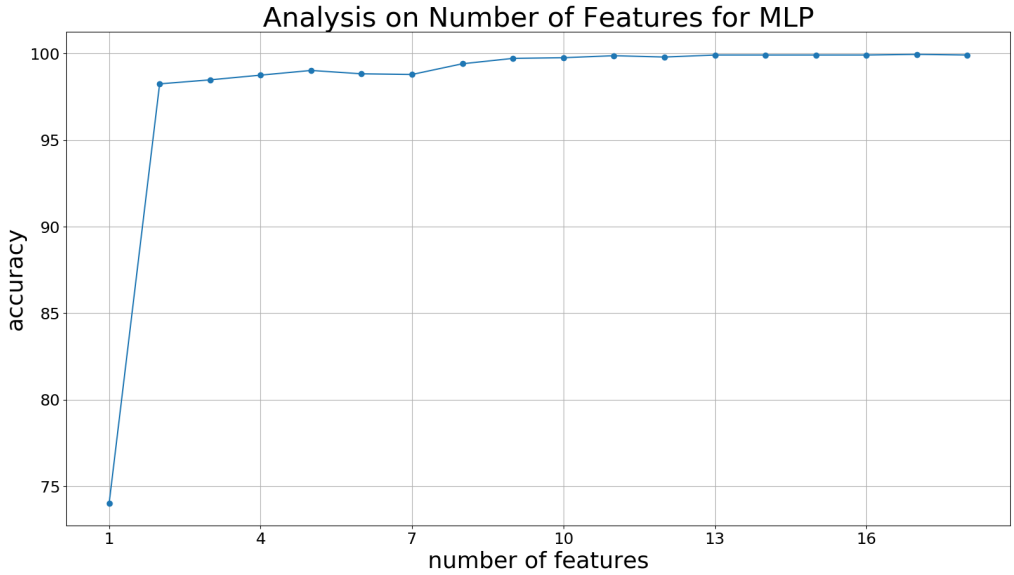


Figure 5.12: A plot showing how the accuracy of the MLP changes as features are added (only including those features chosen as the final feature set for the MLP).

		Predicted			Total
		Node 1	Node 2	Node 3	
Actual	Node 1	860	0	0	860
	Node 2	0	855	5	860
	Node 3	0	0	860	860

Table 5.3: Confusion matrix for the best SVM configuration

		Predicted			Total
		Node 1	Node 2	Node 3	
Actual	Node 1	860	0	0	860
	Node 2	0	859	1	860
	Node 3	0	1	859	860

Table 5.4: Confusion matrix for the best MLP configuration

same for both algorithms - the AR Spectral Magnitude at 14.28 MHz and the approximation coefficient from the db2 wavelet transform corresponding to t=4.096 μs from the convolution. In this case, the SVM achieved the best accuracy on the test data set, at 99.2%, while

the MLP only reached 98.3%.

5.4 Additional Testing

In order to investigate a potential cause for the discrepancy between the characteristics of the signals for each node, data was taken from a different point on the bus, testing for any dependency for measurement location. However, the relevant features showed no verifiable change according to statistical tests. In fact, the best MLP still achieved 99.9% accuracy on this data set, and the SVM still achieved 99.8% accuracy, illustrating that this change did not noticeably affect the performance of the classification. Despite this, it is important to realize that the CANH and CANL lines are not perfect conductors, but have impedances which must be taken into account when designing a system. Therefore, it is best practice to ensure that training data is sampled from the same point on the bus as the final implementation.

Chapter 6

Discussion and Conclusions

6.1 Discussion of practical application

6.1.1 False Alarm Probability

Since 100% identification accuracy is not quite achieved in this paper, and it can certainly never be guaranteed for new data, the possibility of misclassifications must be taken into account. These misclassifications give the potential for false alarm intrusion detections. For the sake of this discussion, we will consider an algorithm with 99.7% accuracy over the long run. While this seems like an extremely high accuracy, at this accuracy we surpass 50% probability of a misidentification after only 167 messages, which can be sent in a small amount of time on the CAN bus. Therefore, the node identification would likely need to be combined with an additional layer. Here, we will propose a strikes-based approach, which keeps a memory of a certain number of previous messages and doesn't raise a flag until a certain number of strikes have been reached within its memory.

It has been found that 0.5 milliseconds is a reasonable average spacing between messages [22]. Thus, we might expect around 2,000 messages per second in a CAN bus. We must now decide a number of messages to keep in memory based on an average span of time that is acceptable to allow a potential hacker to control the bus and a percentage of allowable perceived 'rogue' messages within that span of time. The tradeoff here is that as we decrease the number of messages, we must increase the percentage of allowable 'rogue' messages to prevent the false alarm probability from increasing. To select these parameters, we refer to the binomial distribution. Assuming the correctness of each message is independent, we can model the number of misidentifications k in a span of n messages with a binomial distribution. Then, the false alarm probability will be $1 - P[k \leq nq]$ where q is the percentage of allowable rogue messages (so nq is the number of rogue messages for a block of n messages). Since $P[k \leq nq]$ is the cdf of the binomial, this can be calculated using the regularized

incomplete beta function [88]. This calculation was performed using mpmath for python in order to achieve the best possible floating point precision [89]. A plot showing the tradeoff between time and percentage is shown in Figure 6.1. The false alarm probability $1e-15$ is used to estimate about one billion driving-years (one person driving for one year's worth of time would be one driving-year) before one false alarm can be expected. This number is believed to be sufficient for an extremely sparse occurrence of false alarms. At this false alarm rate, we would have to allow up to about 10% of messages to be marked 'rogue' in a half second interval before we would raise a flag. Alternatively, if we accept more frequent false alarms - say, one every 100,000 driving-years, we decrease the false alarm probability to $1e-11$. However, it can be seen that we do not get much of a decrease in the allowed rogue messages for the significance of the increase in false alarms. At $0.5\,s$ long message blocks (1000 messages per block), we would only decrease the allowable rogue messages by 0.4%, or 4 messages, and at $0.25\,s$ long message blocks (500 messages per block), we would only decrease the allowable rogue messages by 0.8%, or, again, 4 messages.

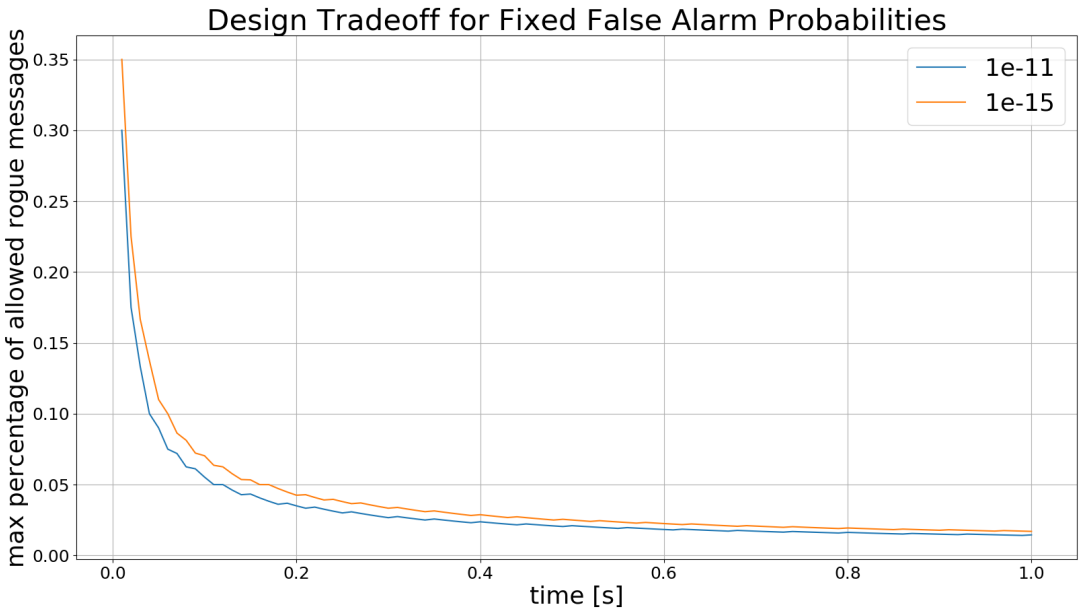


Figure 6.1: This plot shows how the maximum allowable messages marked as rogue varies with average time length based on constant false alarm probabilities of $1e-15$ and $1e-11$.

6.1.2 Computational Complexity of Online Implementation

It is important to consider the computational complexity of this algorithm in order to assess its real-world usefulness. In this section, we will discuss computational complexities

for individual features, as well as an overall computational complexity. For the sake of this discussion, we will assume the use of an MLP neural network since that achieved slightly higher maximum performance in this thesis.

Considering all the features used by the MLP gives 18 features, including magnitudes from an FFT, autoregressive power spectral density estimate, discrete wavelet transform, and time-reversal asymmetry statistic. The typical computational complexity of a FFT is $O(NlogN)$; plus, one $O(N)$ computation is required to apply the time-domain window). The full discrete wavelet transform is $O(WN)$, where W is the number of coefficients for the wavelet or scale function. However, since we again require only a subset of the transform, we can reduce this to $O(W)$ computations for each coefficient. Additionally, the db2 wavelet transform uses only four coefficients each for its scale and wavelet functions, making each of these computations quite small. This leaves us with the calculation for autoregressive PSD estimate, which is the most computationally demanding computation in this group. To solve this, a number of steps must be taken. First, the autocovariance must be found, which is an $O(N^2)$ computation. However, we can cheat with this computation. Using the property that convolution in time domain is multiplication in the frequency domain, we can find the autocovariance using an FFT, which requires 2 $O(NlogN)$ calculations. Next, the autoregressive coefficients must be found, which is an $O(M^2N)$ computation, where M is the number of autoregressive coefficients. An additional $O(NlogN)$ computation must be made to find the PSD estimate based on these parameters. The dominant computation here is $O(M^2N)$, so, technically, this full algorithm is $O(M^2N)$; however, for an embedded, real-time application, these other steps cannot be ignored. Finally, the 3-layer feedforward MLP neural network requires three computations of $O(K_0K_1)$, $O(K_1K_2)$, and $O(K_2K_3)$ complexity, where K_l denotes the number of neurons in layer l (and K_0 is the number of features).

If we use all features, the total computation required includes 4 $O(NlogN)$, 1 $O(M^2N)$, 3 $O(W)$, and 3 $O(K_lK_{l-1})$ computations. This will be dominated by the $O(M^2N)$ computation, which, using the data vectors of $N = 1,376$ and AR order $M = 12$ used in this thesis, requires $198,144$ computations. Continuing this, using the time complexities (for $K_0 = 18$, $K_1 = 64$, $K_2 = 32$, and $K_3 = 3$) results in a lower bound estimate of $261,996$ calculations. The vast majority of this $(198,144)$ comes from the $O(M^2N)$ calculation for the autoregressive model. Of course, it is also possible to reduce computational time by using only a subset of these features, such as the top performing set of 2 features, to reduce the computational load.

6.1.3 Hardware Required for Online Implementation

In order to implement this solution, a few additional pieces of hardware would need to be added to a CAN bus node. These include an ADC to make analog voltage measurements for measuring the noise and some sort of processor to be able to handle the feature extraction

and classification (the initial feature selection and training of the algorithm can be done offline and on separate computers). This section will discuss some suggestions for the selection of such hardware.

The ADC will need to be selected based on both sample rate as well as resolution. The sample rate used in this thesis is 125 MHz. The largest frequency directly used in this thesis is the feature containing the power spectral density at 43.58 MHz, so, the frequency could potentially be decresed to around 90 MHz, but this is not recommended since sample rate has a more complicated relationship with the outputs of other feature extraction techniques, such as the wavelet transformation. Thus, a minimum of 125 MHz is suggested. Additionally, the ADC should be at least 14 bits to give the equivalent of the ADC used in this thesis (15 bits over $(-5, 5)\,V$, or, basically, 14 bits over $(0, 5)\,V$ since negative voltage is never used). However, more bits can only assist in obtaining the best data, and it is recommended to get an ADC with as many bits as possible. Additionally, since transition information is not used in the final results of this thesis, there has been some discussion about optimizing the range of the ADC to only contain the areas around the high and low voltage values. This could be done by using two separate ADCs with tight ranges set around the high and low voltages, and a voltage-controlled switch could be used to control which ADC is read by the processor. This could be a good topic for future work.

The processing power to analyze a 125 MHz data stream in real-time is significant. The potential for the use of a DSP chip will be discussed here, but an application-specific integrated circuit (ASIC) or field-programmable gate array (FPGA) could also be designed to achieve high-performance parallel processing. For determining the necessary specifications of a DSP, we will analyze the number of operations necessary for a full 11-bit section of data (since the section used for classification in this thesis is 11 bits long). We will assert that, if implemented on the identifier frame in a CAN message, the processing must be done by the end of the data frame, which is traditionally 8 bytes long in car communications, giving approximately $64\,\mu s$ for the calculations (assuming a $1 Mbit/s$ bitrate and no stuffed bits, which, if present, would increase the allowable time). Based on our lower bound computational estimate from 6.1.2, allowing for that we ignored some constant factors, and building in some additional room for overhead, we will require the capability for at least $600,000$ calculations in this $53\,\mu s$ time span. As an example, the TMS320C6678 from TI [90], available for roughly $190, has 8 cores running at 1.25 GHz, making it capable of $640,000$ operations during this time frame and thus meeting this criteria. If necessary, the allowed processing time could be extended to include the 16-bit CRC frame, giving an additional $16\,\mu s$ for the calculation. For comparison, in this amount of time, the TMS320C6678 is capable of $800,000$ operations during this time frame.

6.1.4 Portability

This thesis has demonstrated results for only one CAN bus. However, it is believed that it will be portable to other CAN buses because of the amount of features extracted. While all the same features will likely not be significant for all CAN buses, it is believed that all will have some set of features which can be used to perform a classification, and this algorithm is set up so that a large number of features can be extracted, and the best will be chosen based on the data. Additionally, if desired, more features can easily be added without making significant changes to the overall algorithm. Additional wavelets and PSD estimates are obvious potential candidates for addition, as well as making use of existing feature extraction libraries such as tsfresh [91] and the highly comparitive time series analysis introduced in [66] and [67].

6.2 Review

6.2.1 Summary

As cars become more complex, they are including more and more ECUs with much more power and connectedness with the world. While this makes for a publicly better product, it also comes with the need for improved security. It has been demostrated that cars can be hacked into remotely and, once control of the CAN bus has been achieved, a car can be controlled to act as desired or display false warnings to the driver.

This thesis presents a method for assisting in intrusion detection on a CAN bus. We have presented a method of "fingerprinting" the transceivers on the CAN bus, which provides an assurance for which node is sending a message. By identifying the nodes on the CAN bus, it becomes easier to evaluate traffic on the bus and detect suspicious activity.

6.2.2 Accomplishments

The classification of 3 nodes on a CAN bus was tested on an independent data set, and accuracy as high as 99.9% using as little as 13 features with an MLP neural network. Additionally, a potential path for avoiding false alarms has been given. This thesis provides a good proof of concept of the proposed algorithm, and will hopefully be an impetus for future work in this area.

6.2.3 Future Work

To further verify this algorithm, it is necessary to implement it on more CAN buses to prove that it is effective. This includes testing on CAN buses with more than 3 nodes since cars will typically have many more ECUs. If it is necessary to improve performance, more features can be added, as discussed in 6.1.4 or an ADC with higher resolution and/or sampling rate can be used.

It would also be helpful for more testing to be done in which certain parameters related to the CAN bus were varied to see how much each parameter effects the measurements being made and subsequent classification. For example, changing the cable lengths on the CAN bus and taking more data would test how much of the captured effect is coming from transmission line effect, and artificially removing the large spikes in the noise measurements would test how much of the measured effect is a result of the asynchronicity between the switching transistors driving the CANL and CANH lines in the node transceiver.

An additional area of great interest would be to improve the measurement of timing characteristics on the CAN bus. It is known and well understood that any oscillator on a CAN bus node will have manufacturing tolerances around a certain frequency. Such tolerances should lead to differences between the internal frequencies of each node and result in a useful figure for classification. This could be done by using a time-to-digital converter to obtain better time resolution.

Once this algorithm has been verified to work on several CAN buses, a true online implementation must be developed and tested. This requires selecting the proper hardware componenets, developing the integration of these components with the existing components on a CAN bus, writing the feature extraction and classification to work in real time, and developing the protocol for when to become suspicious and what to do if it is believed that an attack is occurring. One way to do this would be to develop a way to integrate the knowledge provided by the node identification into existing intrusion detection systems.

6.2.4 Final Remarks

This thesis presents a promising method for identifying devices on a communication bus. While this work focused mainly on the CAN bus, such identification has obvious security benefits for any communication bus. Knowing the origin of a message can identify both the addition of new units to the bus, as well as compromised units that are attempting to take control of functionality for malicious purposes.

Bibliography

[1] Charlie Miller and Chris Valasek. *Adventures in Automotive Networks and Control Units*. Tech. rep. IOActive, 2014. URL: https://ioactive.com/pdfs/IOActive_Adventures_in_Automotive_Networks_and_Control_Units.pdf (visited on 10/20/2017).

[2] *CANHack.org - Index page*. URL: https://canhack.org/board/ (visited on 12/02/2017).

[3] Craig Smith. *Car Hacking 101: Tools of the Trade — Make:* Apr. 2016. URL: https://makezine.com/2016/04/08/car-hacking-tools-trade/ (visited on 11/01/2017).

[4] Ariel Nuñez. *An Introduction to the CAN Bus: How to Programmatically Control a Car*. June 2017. URL: https://news.voyage.auto/an-introduction-to-the-can-bus-how-to-programmatically-control-a-car-f1b18be4f377 (visited on 11/01/2017).

[5] Andy Greenburg. *A $60 Gadget That Makes Car Hacking Far Easier*. Mar. 2015. URL: https://www.wired.com/2015/03/60-gadget-thatll-make-car-hacking-easier-ever/ (visited on 11/01/2017).

[6] Kristoffer Smith. *a complete guide to hacking your vehicle bus on the cheap & easy - part 1 (hardware interface)*. Mar. 2013. URL: https://theksmith.com/software/hack-vehicle-bus-cheap-easy-part-1/ (visited on 11/01/2017).

[7] Eric Evenchick. *CAN Hacking: Introductions*. Oct. 2013. URL: https://hackaday.com/2013/10/21/can-hacking-introductions/ (visited on 11/01/2017).

[8] *Hack Your Vehicle CAN-BUS With Arduino and Seeed CAN-BUS Shield*. URL: http://www.instructables.com/id/Hack-your-vehicle-CAN-BUS-with-Arduino-and-Seeed-C/ (visited on 11/01/2017).

[9] Pierluigi Paganini. *Car Hacking: You Cannot Have Safety without Security*. Feb. 2014. URL: http://resources.infosecinstitute.com/car-hacking-safety-without-security/ (visited on 11/01/2017).

[10] Chad Gibbons. *Chad Gibbons' Blog*. URL: http://chadgibbons.com/ (visited on 11/01/2017).

[11] *Car Hacker's Handbook by Craig Smith – CanBusHack*. URL: http://canbushack.com/product/car-hackers-handbook-by-craig-smith/ (visited on 11/01/2017).

[12] *Blog – CanBusHack*. URL: http://canbushack.com/blog/ (visited on 11/01/2017).

[13] Edward Markey. *S.1806 - 114th Congress (2015-2016): SPY Car Act of 2015*. eng. webpage. July 2015. URL: https://www.congress.gov/bill/114th-congress/senate-bill/1806/all-info (visited on 10/11/2017).

[14] Stephen Checkoway et al. "Comprehensive Experimental Analyses of Automotive Attack Surfaces." In: *USENIX Security Symposium*. San Francisco, 2011. URL: http://static.usenix.org/events/sec11/tech/full_papers/Checkoway.pdf (visited on 09/08/2017).

[15] Charlie Miller and Chris Valasek. "A survey of remote automotive attack surfaces". In: *Black Hat USA* 2014 (2014).

[16] Charlie Miller and Chris Valasek. "Remote exploitation of an unaltered passenger vehicle". In: *Black Hat USA* 2015 (2015). URL: https://securityzap.com/files/Remote%20Car%20Hacking.pdf (visited on 09/08/2017).

[17] Charlie Miller and Chris Valasek. *CAN Message Injection*. June 2016. URL: http://illmatics.com/can%20message%20injection.pdf (visited on 10/23/2017).

[18] Rebecca Bace and Peter M. Mell. "Intrusion Detection Systems". en. In: *Special Publication (NIST SP) - 800-31* (Nov. 2001). DOI: 151244. URL: https://www.nist.gov/publications/intrusion-detection-systems (visited on 10/19/2017).

[19] Hung-Jen Liao et al. "Intrusion detection system: A comprehensive review". In: *Journal of Network and Computer Applications* 36.1 (Jan. 2013), pp. 16–24. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2012.09.004. URL: http://www.sciencedirect.com/science/article/pii/S1084804512001944 (visited on 10/11/2017).

[20] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures". In: *Reliability Engineering & System Safety*. Special Issue on Safecomp 2008 96.1 (Jan. 2011), pp. 11–25. ISSN: 0951-8320. DOI: 10.1016/j.ress.2010.06.026. URL: http://www.sciencedirect.com/science/article/pii/S0951832010001602 (visited on 10/11/2017).

[21] A. Taylor, N. Japkowicz, and S. Leblanc. "Frequency-based anomaly detection for the automotive CAN bus". In: *2015 World Congress on Industrial Control Systems Security (WCICSS)*. Dec. 2015, pp. 45–49. DOI: 10.1109/WCICSS.2015.7420322.

[22] H. M. Song, H. R. Kim, and H. K. Kim. "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network". In: *2016 International Conference on Information Networking (ICOIN)*. Jan. 2016, pp. 63–68. DOI: 10.1109/ICOIN.2016.7427089.

[23] Marko Wolf and Timo Gendrullis. "Design, Implementation, and Evaluation of a Vehicular Hardware Security Module." In: *ICISC*. Springer, 2011, pp. 302–318.

[24] M. Müter and N. Asaj. "Entropy-based anomaly detection for in-vehicle networks". In: *2011 IEEE Intelligent Vehicles Symposium (IV)*. June 2011, pp. 1110–1115. DOI: 10.1109/IVS.2011.5940552.

[25] Chouchang Yang and Alanson P. Sample. "EM-ID: Tag-less identification of electrical devices via electromagnetic emissions". In: *RFID (RFID), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1–8.

[26] *CAN in Automation (CiA): History of the CAN technology*. URL: https://can-cia.org/can-knowledge/can/can-history/ (visited on 11/01/2017).

[27] *CAN in Automation - Mercedes W140: First car with CAN*. Mar. 2016. URL: https://can-newsletter.org/engineering/applications/160322_25th-anniversary-mercedes-w140-first-car-with-can (visited on 11/01/2017).

[28] *ISO 11898-1:2015 - Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*. URL: https://www.iso.org/standard/63648.html (visited on 11/07/2017).

[29] *ISO 11898-2:2016 - Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit*. URL: https://www.iso.org/standard/67244.html (visited on 11/07/2017).

[30] *ISO 11898-3:2006 - Road vehicles – Controller area network (CAN) – Part 3: Low-speed, fault-tolerant, medium-dependent interface*. URL: https://www.iso.org/standard/36055.html (visited on 11/07/2017).

[31] *ISO 16845-1:2016 - Road vehicles – Controller area network (CAN) conformance test plan – Part 1: Data link layer and physical signalling*. URL: https://www.iso.org/standard/59166.html (visited on 11/07/2017).

[32] *ISO 16845-2:2014 - Road vehicles – Controller area network (CAN) conformance test plan – Part 2: High-speed medium access unit with selective wake-up functionality*. URL: https://www.iso.org/standard/59504.html (visited on 11/07/2017).

[33] *ISO 15765-1:2011 - Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 1: General information and use case definition*. URL: https://www.iso.org/standard/54498.html (visited on 11/07/2017).

[34] *ISO 15765-2:2016 - Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services*. URL: https://www.iso.org/standard/66574.html (visited on 11/07/2017).

[35] *ISO 15765-4:2016 - Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 4: Requirements for emissions-related systems*. URL: https://www.iso.org/standard/67245.html (visited on 11/07/2017).

[36] *J2284/3A: High-Speed CAN (HSC) for Vehicle Applications at 500 KBPS - SAE International*. URL: http://standards.sae.org/j2284/3_201611/ (visited on 11/07/2017).

[37] *Sae Mobilus*. URL: http://standards.sae.org/j2284/5_201609/ (visited on 11/07/2017).

[38] *High-Speed CAN (HSC) for Vehicle Applications at 500 kbps with CAN FD Data at 2 Mbps (J2284/4 Ground Vehicle Standard) - SAE Mobilus*. URL: http://standards.sae.org/j2284/4_201606/ (visited on 11/07/2017).

[39] *High-Speed CAN (HSC) for Vehicle Applications at 500 KBPS (J2284/3 Ground Vehicle Standard) - SAE Mobilus*. URL: http://standards.sae.org/j2284/3_201611/ (visited on 11/07/2017).

[40] *High Speed CAN (HSC) for Vehicle Applications at 250 kbps (J2284/2 Ground Vehicle Standard) - SAE Mobilus*. URL: http://standards.sae.org/j2284/2_201611/ (visited on 11/07/2017).

[41] *High Speed CAN (HSC) for Vehicle Applications at 125 kbps (J2284/1 Ground Vehicle Standard) - SAE Mobilus*. URL: http://standards.sae.org/j2284/1_201611/ (visited on 11/07/2017).

[42] *J2411: Single Wire Can Network for Vehicle Applications - SAE International*. URL: http://standards.sae.org/j2411_200002/ (visited on 11/07/2017).

[43] Steve Corrigan. *Introduction to the Controller Area Network*. Aug. 2016. URL: http://www.ti.com/lit/an/sloa101b/sloa101b.pdf (visited on 10/07/2017).

[44] Steve Corrigan. *Controller Area Network Physical Layer Requirements*. Jan. 2008. URL: http://www.ti.com/lit/an/slla270/slla270.pdf (visited on 10/07/2017).

[45] Marco Di Natale. *Understanding and using the Controller Area Network*. Oct. 2008. URL: http://inst.cs.berkeley.edu/~ee249/fa08/Lectures/handout_canbus2.pdf (visited on 10/24/2017).

[46] George Casella and Roger Berger. *Statistical Inference*. Second Edition. Duxbury Advanced Series. Duxbury, 2002.

[47] Alper Demir and Alberto Sangiovanni-Vincentelli. *Analysis and Simulation of Noise in Nonlinear Electronic Circuits and Systems*. DOI: 10.1007/978-1-4615-6063-0. Boston, MA: Springer US, 1998. ISBN: 978-1-4613-7777-1 978-1-4615-6063-0. URL: http://link.springer.com/10.1007/978-1-4615-6063-0 (visited on 09/22/2017).

[48] Dmitry Panchenko. *Kolmogorov-Smirnov Test*. 2006. URL: https://ocw.mit.edu/courses/mathematics/18-443-statistics-for-applications-fall-2006/lecture-notes/lecture14.pdf.

[49] George Marsaglia, Wai Wan Tsang, and Jingbo Wang. "Evaluating Kolmogorov's Distribution". In: *Journal of Statistical Software* 8.18 (2003), pp. 1–4. ISSN: 1548-7660. DOI: 10.18637/jss.v008.i18. URL: https://www.jstatsoft.org/article/view/v008i18 (visited on 09/23/2017).

[50] Robert Gallager. *Stochastic Processes: Theory for Applications*. 1st ed. Cambridge University Press, 2014. ISBN: 978-1-107-03975-9. URL: https://www.scribd.com/doc/152258575/Stochastic-Processes-Theory-for-Applications-by-Robert-g-Gallager (visited on 10/02/2017).

[51] V. Kontorevich and V. Lyandres. "Impulsive noise: A nontraditional approach". In: *Signal Processing* 51.2 (June 1996), pp. 121–132. ISSN: 0165-1684. DOI: 10.1016/0165-1684(96)00036-9. URL: http://www.sciencedirect.com/science/article/pii/0165168496000369 (visited on 09/30/2017).

[52] Gregory Kotler et al. "On the Markov model of shot noise". In: *Signal Processing* (1999), pp. 79–88.

[53] P. Blanchard et al. "Stochastic dynamical aspects of neuronal activity". en. In: *Journal of Mathematical Biology* 31.2 (Jan. 1993), pp. 189–198. ISSN: 0303-6812, 1432-1416. DOI: 10.1007/BF00171226. URL: https://link.springer.com/article/10.1007/BF00171226 (visited on 09/30/2017).

[54] A. Papoulis. "High Density Shot Noise and Gaussianity". In: *Journal of Applied Probability* 8.1 (1971), pp. 118–127. ISSN: 0021-9002. DOI: 10.2307/3211842. URL: http://www.jstor.org/stable/3211842 (visited on 09/30/2017).

[55] E. N. Gilbert and H. O. Pollak. "Amplitude distribution of shot noise". In: *The Bell System Technical Journal* 39.2 (Mar. 1960), pp. 333–350. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1960.tb01603.x.

[56] Rudolph Emil Kalman. "A new approach to linear filtering and prediction problems". In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45. URL: https://fluidsengineering.asmedigitalcollection.asme.org/pdfaccess.ashx?resourceid=4506257&pdfsource=13 (visited on 10/01/2017).

[57] Frederic Harris. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform". In: *Proceedings of the IEEE* 66.1 (Jan. 1978), pp. 51–83.

[58] James W. Cooley and John W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series". In: *Mathematics of Computation* 19.90 (Apr. 1965), p. 297. ISSN: 00255718. DOI: 10.2307/2003354. URL: http://www.jstor.org/stable/2003354?origin=crossref (visited on 09/20/2017).

[59] P. Welch. "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms". In: *IEEE Transactions on Audio and Electroacoustics* 15.2 (June 1967), pp. 70–73. ISSN: 0018-9278. DOI: 10.1109/TAU.1967.1161901.

[60] Hirotugu Akaike. "Fitting autoregressive models for prediction". In: *Annals of the institute of Statistical Mathematics* 21.1 (1969), pp. 243–247.

[61] Hirotugu Akaike. "Power spectrum estimation through autoregressive model fitting". In: *Annals of the Institute of Statistical Mathematics* 21.1 (Dec. 1969), pp. 407–419.

[62] I. Daubechies. "Where do wavelets come from? A personal point of view". In: *Proceedings of the IEEE* 84.4 (Apr. 1996), pp. 510–513. ISSN: 0018-9219. DOI: 10.1109/5.488696.

[63] Fabian Mörchen. *Time series feature extraction for data mining using DWT and DFT*. Tech. rep. 2003.

[64] A. Graps. "An introduction to wavelets". In: *IEEE Computational Science and Engineering* 2.2 (1995), pp. 50–61. ISSN: 1070-9924. DOI: 10.1109/99.388960.

[65] Ingrid Daubechies. "Orthonormal bases of compactly supported wavelets". In: *Communications on pure and applied mathematics* 41.7 (1988), pp. 909–996. URL: http://onlinelibrary.wiley.com/doi/10.1002/cpa.3160410705/full (visited on 09/27/2017).

[66] Ben D. Fulcher, Max A. Little, and Nick S. Jones. "Highly comparative time-series analysis: the empirical structure of time series and their methods". en. In: *Journal of The Royal Society Interface* 10.83 (June 2013), p. 20130048. ISSN: 1742-5689, 1742-5662. DOI: 10.1098/rsif.2013.0048. URL: http://rsif.royalsocietypublishing.org/content/10/83/20130048 (visited on 10/08/2017).

[67] B. D. Fulcher and N. S. Jones. "Highly Comparative Feature-Based Time-Series Classification". In: *IEEE Transactions on Knowledge and Data Engineering* 26.12 (Dec. 2014), pp. 3026–3037. ISSN: 1041-4347. DOI: 10.1109/TKDE.2014.2316504.

[68] Thomas Schreiber and Andreas Schmitz. "Surrogate time series". In: *Physica D: Nonlinear Phenomena* 142.3 (Aug. 2000), pp. 346–382. ISSN: 0167-2789. DOI: 10.1016/S0167-2789(00)00043-9. URL: http://www.sciencedirect.com/science/article/pii/S0167278900000439 (visited on 10/08/2017).

[69] D. N. Joanes and C. A. Gill. "Comparing Measures of Sample Skewness and Kurtosis". In: *Journal of the Royal Statistical Society. Series D (The Statistician)* 47.1 (1998), pp. 183–189. ISSN: 0039-0526. URL: http://www.jstor.org/stable/2988433 (visited on 10/08/2017).

[70] Andrew Ng. *Machine Learning*. Lecture. Coursera.com, 2017. URL: https://www.coursera.org/learn/machine-learning?authMode=login (visited on 10/01/2017).

[71] G. P. Zhang. "Neural networks for classification: a survey". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 30.4 (Nov. 2000), pp. 451–462. ISSN: 1094-6977. DOI: 10.1109/5326.897072.

[72] Anders Krogh and John A. Hertz. "A Simple Weight Decay Can Improve Generalization". In: *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann, 1992, pp. 950–957.

[73] Andreas Weigend, Bernardo Huberman, and David Rumelhart. "Generalization by Weight-Elimination with Application to Forecasting". In: *Int. J. Neural Syst.* 1 (Jan. 1990), pp. 193–209. DOI: 10.1142/S0129065790000102.

[74] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958. URL: http://jmlr.csail.mit.edu/papers/v15/srivastava14a.html (visited on 10/01/2017).

[75] Jonathan Richard Shewchuk et al. *An introduction to the conjugate gradient method without the agonizing pain*. Carnegie-Mellon University. Department of Computer Science, 1994. URL: ftp://ftp.unicauca.edu.co/Facultades/.FIET_serepiteencuentasyocupaespa
DEIC/docs/Materias/computacion%20inteligente/parte%20II/semana12/gradient/painless-conjugate-gradient.pdf (visited on 10/02/2017).

[76] Chih-Jen Lin. "Probability estimates for multi-class classification by pairwise coupling". In: *The Journal of Machine Learning Research* (Aug. 2004). URL: http://www.academia.edu/2834474/Probability_estimates_for_multi-class_classification_by_pairwise_coupling (visited on 09/29/2017).

[77] Johan AK Suykens and Joos Vandewalle. "Least squares support vector machine classifiers". In: *Neural processing letters* 9.3 (1999), pp. 293–300. URL: http://www.springerlink.com/index/n75178640w32646j.pdf (visited on 09/29/2017).

[78] Jason Weston. *Support Vector Machine Tutuorial*. URL: http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf (visited on 09/30/2017).

[79] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (Jan. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T (visited on 09/27/2017).

[80] "Learning Polynomials with Neural Networks". eng. In: *Proceedings of the 31 st International Conference on Machine Learning*. Ed. by Alexandr Andoni et al. Vol. 32. OCLC: 179807959. Beijing, China: ACM Press, 2014. ISBN: 978-1-58113-960-0.

[81] M. Bianchini and F. Scarselli. "On the Complexity of Neural Network Classifiers: A Comparison Between Shallow and Deep Architectures". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.8 (Aug. 2014), pp. 1553–1565. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2013.2293637.

[82] D. Hunter et al. "Selection of Proper Neural Network Sizes and Architectures: A Comparative Study". In: *IEEE Transactions on Industrial Informatics* 8.2 (May 2012), pp. 228–240. ISSN: 1551-3203. DOI: 10.1109/TII.2012.2187914.

[83] Chistopher Bishop. *Pattern Recognition and Machine Learning*. Springer US, 2006.

[84] Avrim L. Blum and Pat Langley. "Selection of relevant features and examples in machine learning". In: *Artificial Intelligence*. Relevance 97.1 (Dec. 1997), pp. 245–271. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(97)00063-5. URL: http://www.sciencedirect.com/science/article/pii/S0004370297000635 (visited on 09/28/2017).

[85] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[86] Francois Chollet et al. *Keras*. GitHub, 2017. URL: `https://github.com/fchollet/keras`.

[87] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv: 1412.6980. URL: `http://arxiv.org/abs/1412.6980` (visited on 11/05/2017).

[88] George Proctor Wadsworth and Joseph G. Bryan. *Introduction to Probability and Random Variables*. en. Google-Books-ID: NNtQAAAAMAAJ. McGraw-Hill, 1960.

[89] Fredrik Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.0.0)*. Sept. 2017. URL: `http://mpmath.org`.

[90] *TMS320C6678*. Nov. 2014. URL: `http://www.ti.com/lit/ds/symlink/tms320c6678.pdf` (visited on 11/05/2017).

[91] Maximillian Christ et al. *TSFRESH*. GitHub, 2017. URL: `https://github.com/blue-yonder/tsfresh`.

Appendix A

Features

A.1 MLP Features

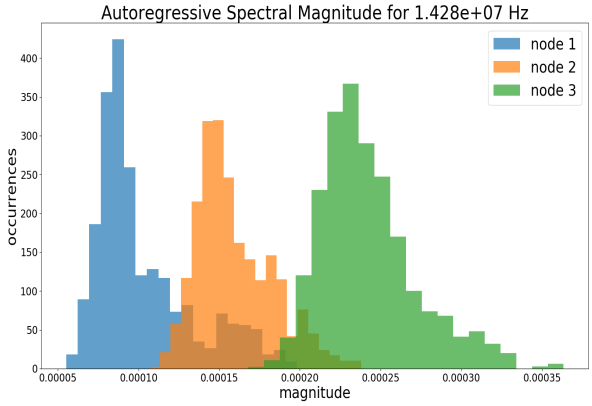


Figure A.1: The overlayed histogram for all nodes.

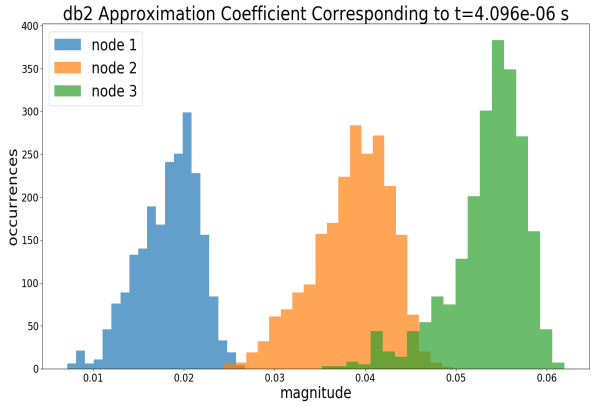


Figure A.2: The overlayed histogram for all nodes.

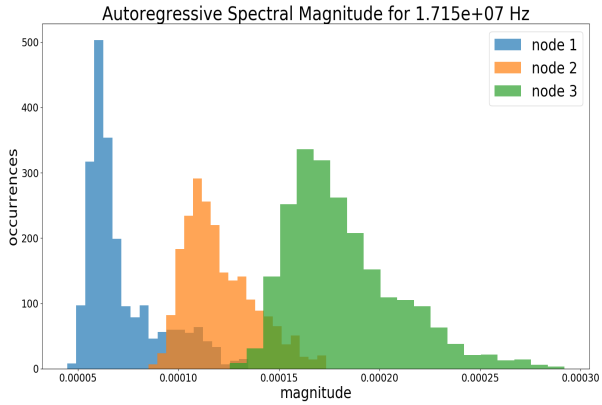


Figure A.3: The overlayed histogram for all nodes.

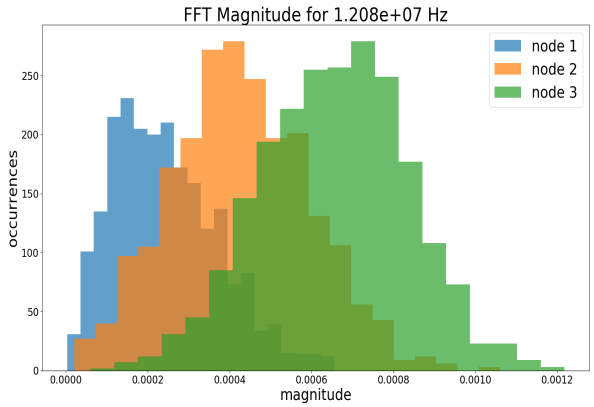


Figure A.4: The overlayed histogram for all nodes.

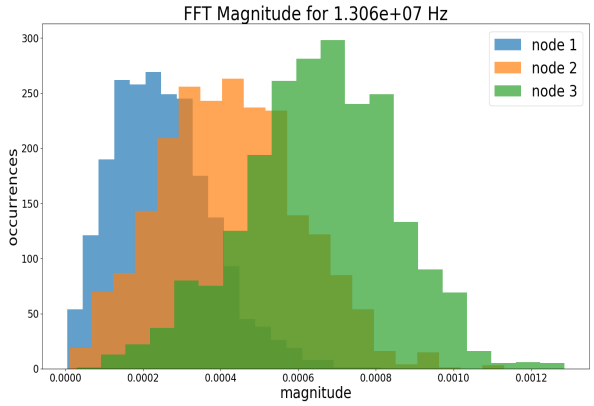


Figure A.5: The overlayed histogram for all nodes.

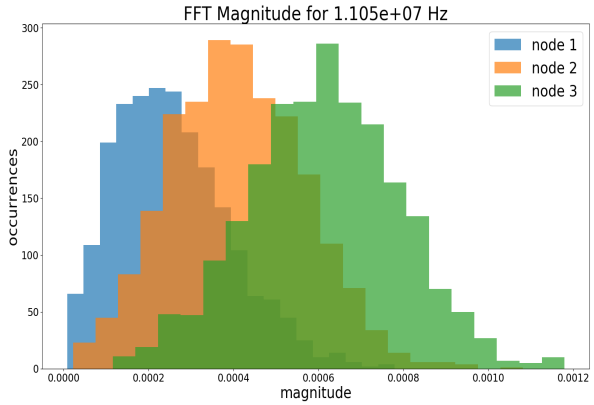


Figure A.6: The overlayed histogram for all nodes.

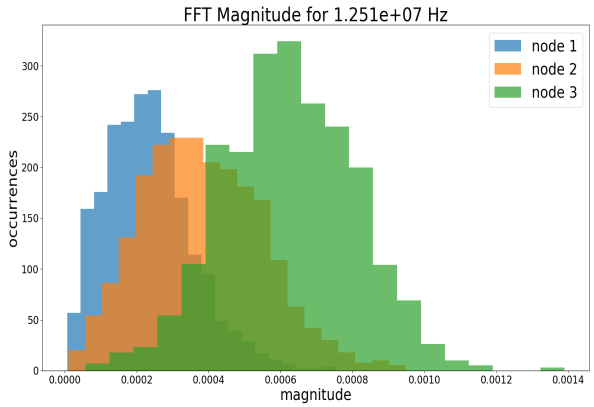


Figure A.7: The overlayed histogram for all nodes.

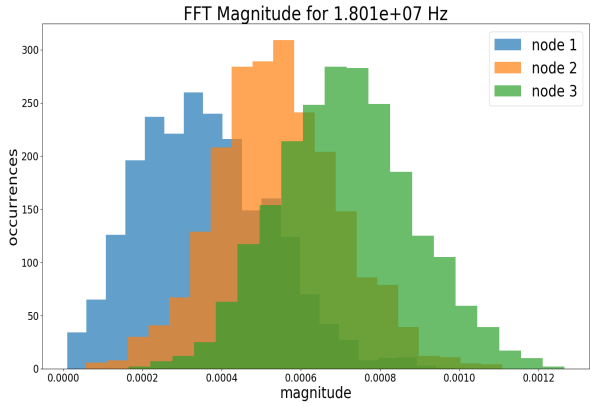


Figure A.8: The overlayed histogram for all nodes.

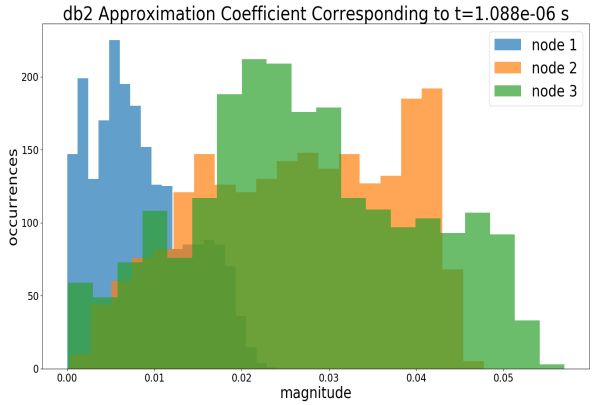


Figure A.9: The overlayed histogram for all nodes.

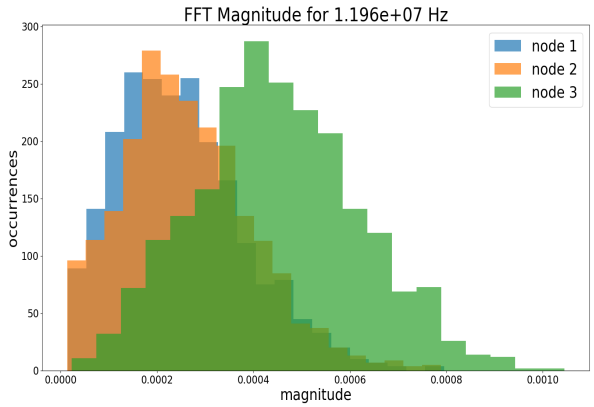


Figure A.10: The overlayed histogram for all nodes.

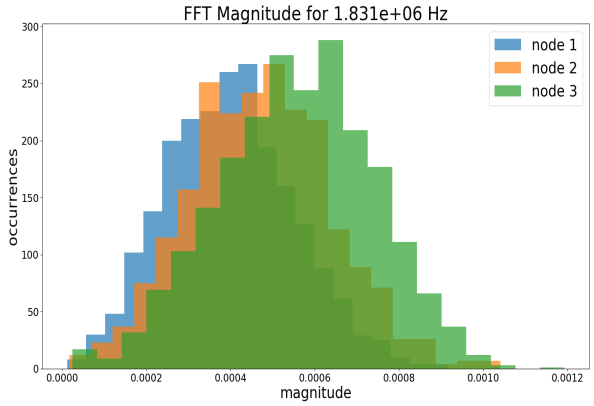


Figure A.11: The overlayed histogram for all nodes.

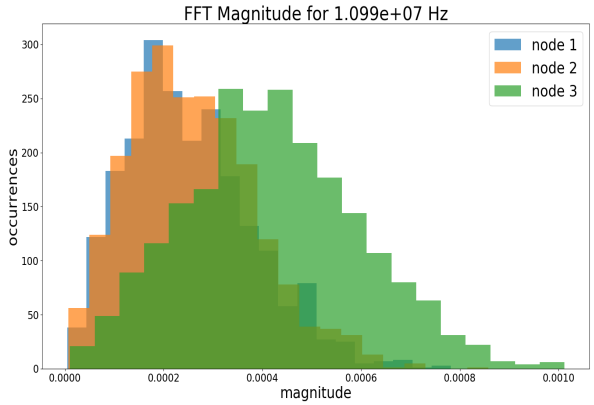


Figure A.12: The overlayed histogram for all nodes.

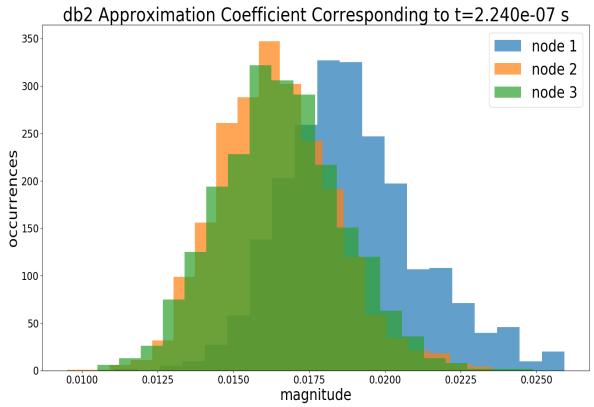


Figure A.13: The overlayed histogram for all nodes.

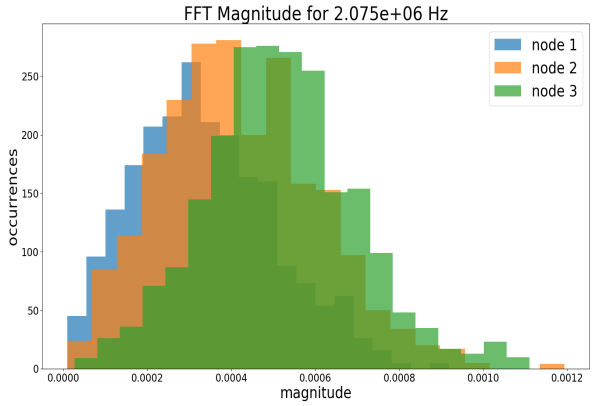


Figure A.14: The overlayed histogram for all nodes.

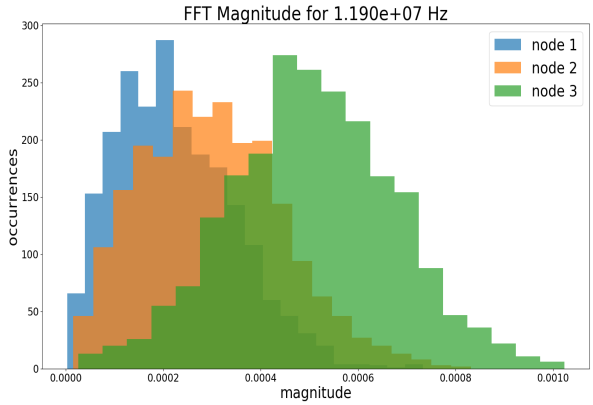


Figure A.15: The overlayed histogram for all nodes.

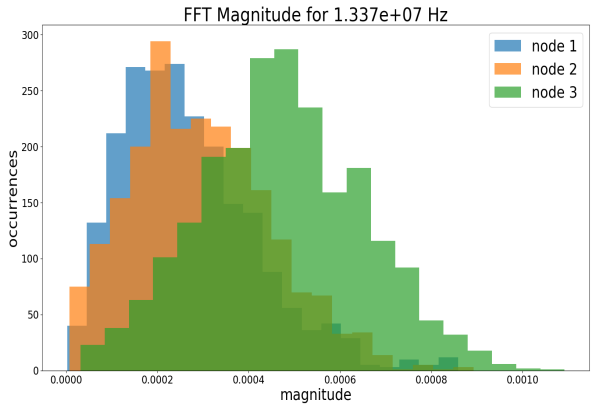


Figure A.16: The overlayed histogram for all nodes.

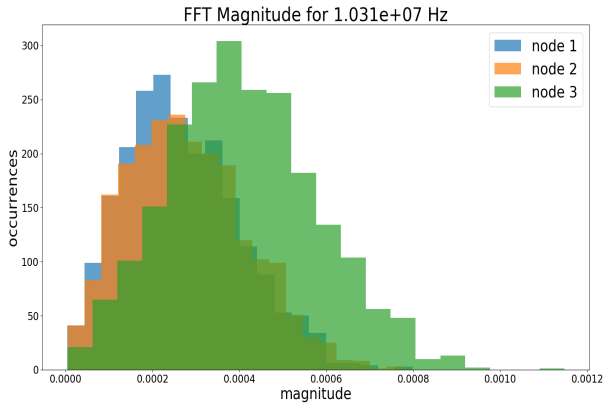


Figure A.17: The overlayed histogram for all nodes.

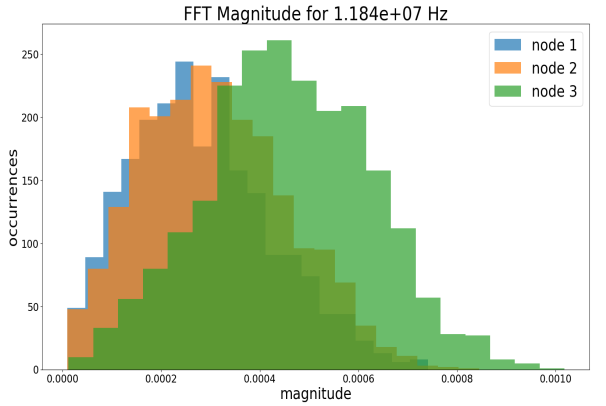


Figure A.18: The overlayed histogram for all nodes.

A.2 SVM Features

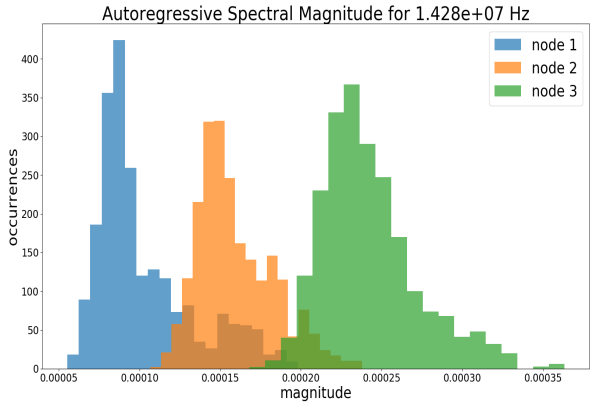


Figure A.19: The overlayed histogram for all nodes.

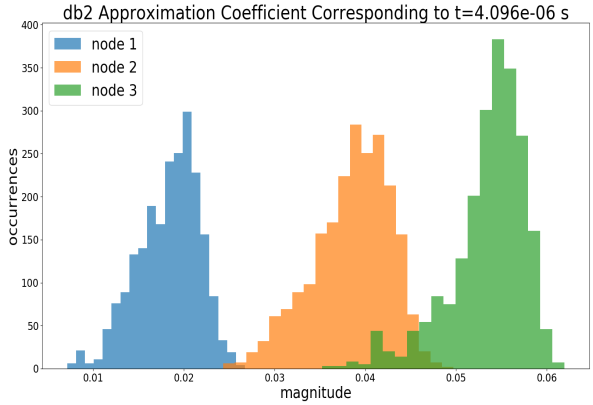


Figure A.20: The overlayed histogram for all nodes.

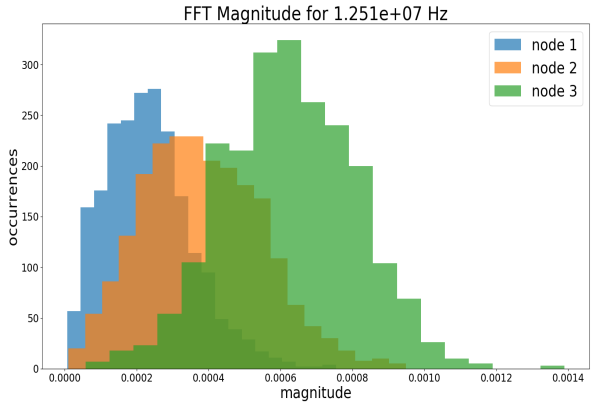


Figure A.21: The overlayed histogram for all nodes.

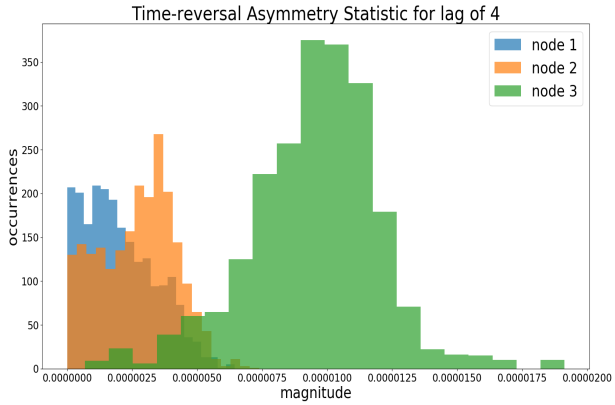


Figure A.22: The overlayed histogram for all nodes.

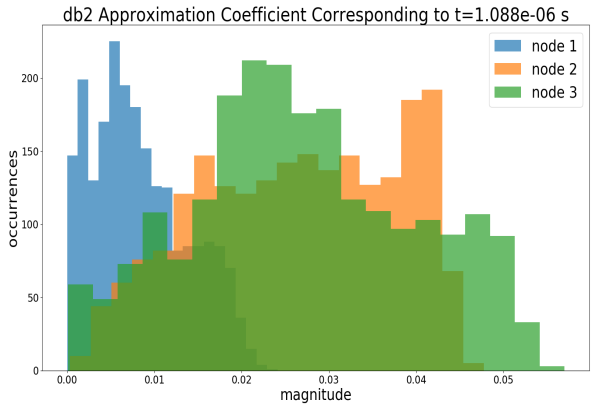


Figure A.23: The overlayed histogram for all nodes.

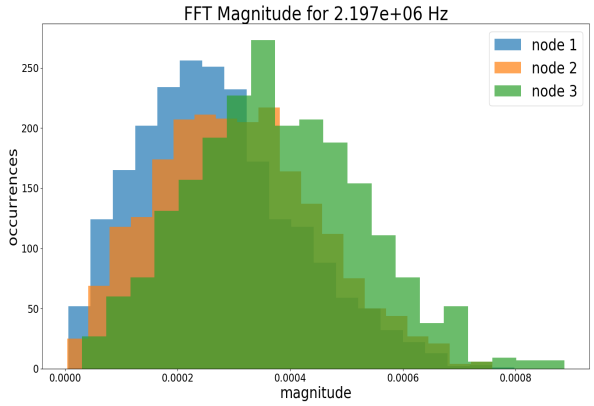


Figure A.24: The overlayed histogram for all nodes.

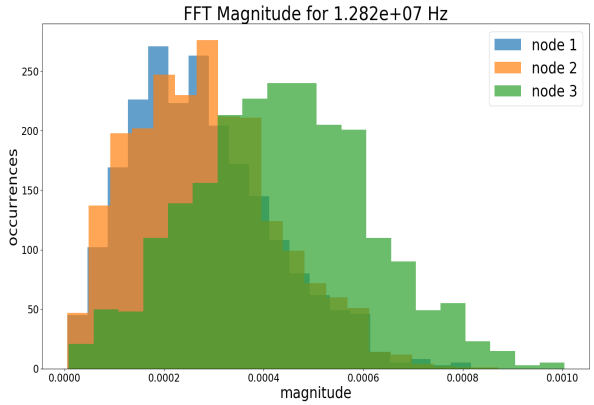


Figure A.25: The overlayed histogram for all nodes.

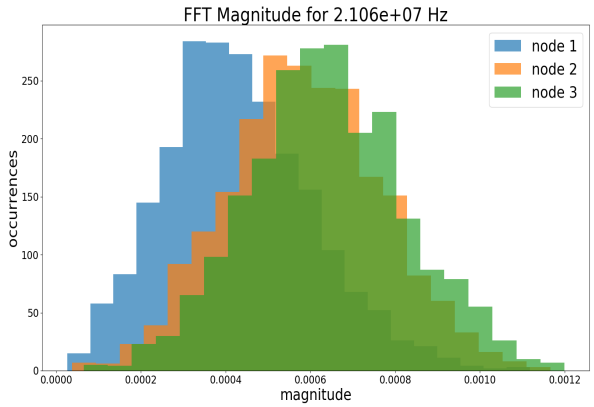


Figure A.26: The overlayed histogram for all nodes.