Shape detection of physical objects with Intel 5300 and the 802.11n CSI Tool

Andrew Lopez

Follow this and additional works at: https://lib.dr.iastate.edu/creativecomponents

Part of the Digital Communications and Networking Commons

Recommended Citation
Lopez, Andrew, "Shape detection of physical objects with Intel 5300 and the 802.11n CSI Tool" (2020). Creative Components. 526.
https://lib.dr.iastate.edu/creativecomponents/526

This Creative Component is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Creative Components by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Shape detection of physical objects with Intel 5300 and the 802.11n CSI Tool

by

Andrew Thomas Lopez

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Daji Qiao, Major Professor

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2020

Copyright © Andrew Thomas Lopez, 2020. All rights reserved.
DEDICATION

I would like to dedicate this thesis to my wife Nissa. She has always supported and encouraged me in my educational endeavors and all other areas of my life. Without her sacrifice, taking care of our newborn and toddler, the many long nights and busy weekends working on this project would not have been possible.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>NOMENCLATURE</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>viii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>ix</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION</td>
<td>10</td>
</tr>
<tr>
<td>Overview</td>
<td>10</td>
</tr>
<tr>
<td>Contributions</td>
<td>11</td>
</tr>
<tr>
<td>Organization</td>
<td>11</td>
</tr>
<tr>
<td>CHAPTER 2. CSI FUNDAMENTALS</td>
<td>12</td>
</tr>
<tr>
<td>Introduction to CSI</td>
<td>12</td>
</tr>
<tr>
<td>Understanding MIMO</td>
<td>13</td>
</tr>
<tr>
<td>Data Transmission Quality</td>
<td>14</td>
</tr>
<tr>
<td>CHAPTER 3. EXPERIMENT DESIGN</td>
<td>16</td>
</tr>
<tr>
<td>CSI Platforms</td>
<td>16</td>
</tr>
<tr>
<td>Tool Installation</td>
<td>17</td>
</tr>
<tr>
<td>Tool Operation</td>
<td>19</td>
</tr>
<tr>
<td>Experiments</td>
<td>25</td>
</tr>
<tr>
<td>Hardware</td>
<td>27</td>
</tr>
<tr>
<td>CHAPTER 4. DATA COLLECTION</td>
<td>32</td>
</tr>
<tr>
<td>Data Sets</td>
<td>32</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>35</td>
</tr>
<tr>
<td>CHAPTER 5. DATA ANALYSIS</td>
<td>37</td>
</tr>
<tr>
<td>Pre-Analysis Setup</td>
<td>37</td>
</tr>
<tr>
<td>Analysis</td>
<td>38</td>
</tr>
<tr>
<td>Results</td>
<td>41</td>
</tr>
<tr>
<td>CHAPTER 6. CONCLUSION</td>
<td>43</td>
</tr>
<tr>
<td>Summary</td>
<td>43</td>
</tr>
<tr>
<td>Lessons Learned</td>
<td>43</td>
</tr>
<tr>
<td>Future Work</td>
<td>44</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>46</td>
</tr>
<tr>
<td>APPENDIX A. USER’S GUIDE</td>
<td>48</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1. Plot of CSI Signal to Noise Ratio .......................................................... 13
Figure 2. Network Controller ................................................................................. 19
Figure 3. Wlan0 Device ......................................................................................... 21
Figure 4. Wlan0 Link Status ................................................................................... 22
Figure 5. IP Address .............................................................................................. 22
Figure 6. CSI Logging Command ......................................................................... 23
Figure 7. Ping Access Point .................................................................................. 23
Figure 8. Capturing CSI ......................................................................................... 24
Figure 9. CSI Data Verification ............................................................................. 24
Figure 10. Reflector Angle 15 Degrees ................................................................. 26
Figure 11. Mirrored L Shape ................................................................................. 27
Figure 12. Shuttle XPC System .............................................................................. 28
Figure 13. Reflector Panels ................................................................................... 28
Figure 14. Netgear N600 ..................................................................................... 29
Figure 15. Test Environment ................................................................................. 31
Figure 16. CSI Event Signatures ......................................................................... 34
Figure 17. CSI in MATLAB Workspace ............................................................... 37
Figure 18. First CSI Entry .................................................................................... 37
Figure 19. Plot of Key Factor Analysis .................................................................. 39
Figure 20. TR2B and TR1A Shape Detection Comparison ...................................... 42
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1. Object Angles</td>
<td>26</td>
</tr>
<tr>
<td>Table 2. Object Shapes</td>
<td>27</td>
</tr>
<tr>
<td>Table 3. Experiment Events</td>
<td>29</td>
</tr>
<tr>
<td>Table 4. Masters Guide for Experiment 1</td>
<td>33</td>
</tr>
<tr>
<td>Table 5. Masters Guide for Experiment 2</td>
<td>34</td>
</tr>
<tr>
<td>Table 6. Key Factor Analysis Results</td>
<td>40</td>
</tr>
</tbody>
</table>
### NOMENCLATURE

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSI</td>
<td>Channel State Information</td>
</tr>
<tr>
<td>MIMO</td>
<td>Multiple-Input and Multiple-Output</td>
</tr>
<tr>
<td>SISO</td>
<td>Single-Input and Single-Output</td>
</tr>
<tr>
<td>LOS</td>
<td>Line of Sight</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal to Noise Ratio</td>
</tr>
<tr>
<td>OFDM</td>
<td>Orthogonal Frequency Division Multiplexing</td>
</tr>
<tr>
<td>AP</td>
<td>Access Point</td>
</tr>
<tr>
<td>WPA</td>
<td>Wi-Fi Protected Access</td>
</tr>
<tr>
<td>WEP</td>
<td>Wired Equivalent Privacy</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Interface Card</td>
</tr>
<tr>
<td>SSID</td>
<td>Service Set Identifier</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I want to express my gratitude and thanks to those who have assisted me in conducting the research for this thesis during my graduate studies. I would like to thank my Major Professor Dr. Daji Qiao for all of his guidance, motivation, and support throughout this research. I would also like to thank my Uncle Russell for his expert guidance on proper antenna placement and the reflector design. My Aunt Tammie provided an immense amount of help proofreading this paper and I could not have done it without her. Next, I would like to thank my coworkers for their expert troubleshooting support on wireless performance and hardware configurations. In addition, I would also like to thank my friends, colleagues, and the department faculty and staff for making my time at Iowa State University a wonderful experience. Lastly, I would like to thank my dad, who taught me to always work hard and achieve my goals.
ABSTRACT

This thesis presents a detailed look at channel state information (CSI), an efficient approach to the shape detection of physical objects, and observations on how environment noise affects CSI. CSI describes the communication link between a transmitter and receiver through the properties of multiple channels. Once analyzed, these channels or subcarriers describe how the signal travels from the transmitting device to the receiving device. While this information was intended to help increase signal quality and strength of a communication link, many other applications have been suggested.

The proposed application in this paper provides a way with minimal resources to capture and utilize CSI for shape detection. Instead of relying on gestures or movements, I focus on the experiment setup used to detect object shapes and their respective CSI signature. Hence, if I can determine the shape an object should be, then I can detect when that shape changes and capture specific events. In addition, I utilize Wi-Fi as a common source available to provide CSI in many real-world applications without the need for additional resources. Experimentally, I demonstrate how to build a system capable of capturing CSI, how to use CSI logging tools, how environment noise affects CSI, and an approach to detect shape changes.
CHAPTER 1. INTRODUCTION

Overview

Radio Frequency (RF) signals are transmitted and captured by millions of devices. Wi-Fi, a well-known RF frequency standard, continues to increase its presence all around the globe. Wi-Fi signals contain information known as channel state information (CSI). CSI provides channel properties of a communication link between the transmitter and receiver devices with the intention of determining the expected performance of a wireless signal[1]. If an object is located between a transmitting location and a receiving location, the transmitted Wi-Fi signal reflections are different compared to a clear path [2]. Even if an object is not directly in the line of sight (LOS) between the transmitting and receiving device, I can still detect a change in the Wi-Fi signal’s path. This is due to the multiple-input and multiple-output standard, or MIMO. MIMO utilizes multiple antennas for transmitting and receiving to allow multiple signals to reach their destination via more than one path [3]. The resulting data contains CSI for each transmit and receive antenna pair at each subcarrier frequency [4].

In addition to transferring information between devices, RF signals can be utilized for other applications. There are many published research papers on CSI-based sensing. However, there is a lack of research that provides a step-by-step guide to obtaining CSI and a simple approach to detecting shape change in objects. I propose one such path and an application for the shape detection of physical objects. Consider this practical example: Imagine an old deteriorating bridge. One wishes to detect the movement of this bridge in the event the bridge becomes structurally compromised. Normally this would be an expensive endeavor involving cameras, sensors, and other devices consuming large amounts of energy. Consider another possibility that requires minimal resources and provides an early warning of a structural integrity failure. I can
capture RF signals already being transmitted in the environment and monitor the bridge by analyzing the signal’s CSI. To do this, I build a target reflector and place it on the bridge. I then monitor shape change in the reflector. This allows high visibility of the bridge as it changes the propagation of signals being captured by the receiver. As the bridge changes shape or moves, it would in turn move the reflector and be detected.

Using minimal resources, this application provides a unique solution to shape detection. Careful placement of the receiver and reflector would allow use of transmissions from nearby sources to monitor an object and detect specific changes.

**Contributions**

The contributions of this paper include:

- Proposing a simple application for object shape detection
- Demonstrating experimentally how to detect shapes in CSI
- Demonstrating experimentally how to detect environment noise
- Developing MATLAB code to parse the CSI and generate shape detection plots

**Organization**

The rest of this thesis is organized as follows. In Chapter 2, I provide a detailed introduction to CSI fundamentals. In Chapter 3, I discuss platforms for CSI and provide details on the experimental design. In Chapter 4, I cover data collection and discuss technical issues encountered. In Chapter 5, I provide basic observations and discussion on the data collected. I show figures of the results and explain how they support the hypothesis. Finally, the thesis is concluded in Chapter 6 with a summary, lessons learned, and possible future directions.

Appendix A includes a User’s Guide for the 802.11n CSI Tool. Appendix B includes MATLAB code used to parse and analyze the CSI data.
CHAPTER 2. CSI FUNDAMENTALS

Introduction to CSI

Channel State Information (CSI) is the information generated by wireless signals between a transmitting device and receiving device. CSI property values are derived from the physical layer and based on the path the signal travels [5]. For example, an omnidirectional antenna will transmit a wireless signal equally in all directions. The signal path is altered when there are objects in the environment in proximity to the either the transmitter or receiver. The wireless signals can be reflected off multiple surfaces and still reach the target receiver. The CSI captured by the receiver contains a specific signature that reflects the path taken to reach the receiver. This information can be analyzed to characterize events and detect shapes.

The CSI signature is typically measured by amplitude, phase angles, and the calculated signal to noise ratio (SNR). Originally, the intention of CSI was to determine the expected performance of a wireless signal. However, one or more of these properties can be used to determine if an object has moved and changed shape. Movement of the transmitter, receiver, and nearby objects are other possible events detectable by CSI [6]. Figure 1 shows a plot of SNR [dB] on the y-axis and time on the x-axis for 30 subcarrier frequencies. The large SNR fluctuations show detectable changes tied to events occurring in the environment. In this study, these events correspond to a mix of large events, such as the garage door opening, to smaller events, such as changing the shape of an object.
By increasing the number of transmit and receive antennas, an increase in subcarrier frequency groups (SFG) occurs. Each subcarrier, in its respective SFG, represents a different frequency and provides an additional source of CSI information. The additional CSI data captured is useful in detecting more features from an environment, because it increases the scope of the environment observed. CSI is very sensitive to changes in the environment and therefore can generate a lot of unwanted noise [7]. The noise makes it difficult to detect small events due to their minimal impact to the environment. Therefore, in some cases, it is necessary to filter out the noise to detect smaller actions [7]. One of the main goals of this study is to show proof of concept for shape detection. Hence, I focus on larger event footprints that do not require noise filtering to detect.

**Understanding MIMO**

The ability to capture large amounts of CSI from multiple subcarriers provides a higher success rate of detecting specific events and eliminates the need for detecting objects. This method of communication is known as multiple-input and multiple-out or MIMO. MIMO is a communications standard used to increase throughput and enhance performance of wireless signals by allowing multipath propagation of wireless signals. In other words, this technology
was created to allow for an increase in the number of antennas a device could support for simultaneous data transmission. In addition, MIMO helps to reduce signal interference.

There are three main advantages a system using MIMO has over Single in, Single out (SISO). First, the signal strength can be improved without a clear line of sight [8]. This is due to MIMO being able to use transmissions that have been reflected. In this study, this is particularly useful as it allows for unique CSI signatures from reflections off an object. Second, the amount of bandwidth generated from multiple redundant data streams will increase the capability of detecting events [8]. The more transmit and receive antenna pairs available, the more data streams can be transmitted simultaneously. Third, multiple data streams also increase quality of the signal which provides an increase in capability when detecting objects outside LOS [8]. In this study, this aides in detecting CSI on subcarriers that are reflected off an object. This is important since placing objects within LOS will not always be a practical option in real world scenarios. Wireless devices that use IEEE 802.11n/ac standards typically use MIMO [7]. In this study I utilize the 802.11n wireless networking standard.

Mathematically, a MIMO system can be expressed as: \( Y = Hx + n \)

Where \( y \) is the receiving vector, \( x \) is the transmitting vector, \( H \) is the channel matrix containing complex values of CSI, and \( n \) is the vector representing environmental background noise [3]. Using MIMO, a CSI matrix is created by transmitting data to an access point. The access point, in turn, constructs the CSI information and returns the information to the client where it is extracted.

**Data Transmission Quality**

The underlying reason behind CSI’s use is data transmission quality. The 802.11n standard transmits data using Orthogonal Frequency Division Multiplexing (OFDM). OFDM provides modulated data across multiple subcarriers using multiple antennas on the transmit and
receiver side. The quality of data transmitted between the transmitter and receiver is affected by many factors such as scattering, fading, and signal loss over distance. To measure the quality, I use CSI because it can measure channel properties on each individual subcarrier across multiple antennas [9].

The CSI matrix or channel matrix, $H$, contains complex numbers that describe the amplitude and phase angles of the wireless link [9]. Using MIMO, a CSI matrix is created by transmitting data to the receiver. The subcarriers scattering across different paths generate a unique CSI signature described by their amplitude and phase values. CSI values are determined for each subcarrier. The receiver, in turn, constructs the CSI matrix data and returns the information to the transmitter where it is extracted.
CHAPTER 3. EXPERIMENT DESIGN

CSI Platforms

CSI is collected and used for many different sensing applications. The software platforms commonly used to capture CSI are the Linux 802.11n CSI Tool and the Atheros CSI Tool [10]. Both tools are similar in that they capture CSI from a network interface card (NIC) and use open source wireless drivers. There are a few major differences, however. The Linux 802.11n Tool uses the Intel Wi-Fi 5300 NIC and iwlwifi wireless driver [11]. A limitation of the 802.11n tool is the modified firmware used by the Intel 5300 NIC being closed source. Another caveat is the Intel card being limited to capturing thirty subcarrier groups of CSI [11]. Note that thirty subcarrier groups across multiple antennas provide a substantial amount of CSI. The subcarriers captured are spread evenly across either fifty-six subcarriers for the 20Mhz channel or one hundred fourteen subcarriers for the 40Mhz channel [11].

In comparison, the Atheros CSI Tool uses ath9k, an open source Linux kernel driver that claims to support all Atheros 802.11n chipsets [12]. Although ath9k supports a wider range of wireless devices, the limitation of using a NIC is still in place. There are options for USB wireless devices and embedded wireless, however locating these can be challenging. Next, the tool functions are open source and designed solely through software[12]. Hence, there are no modifications to the firmware needed and users can modify the software tool to suit their needs [12]. One major advantage of the Atheros CSI Tool is the detailed instructions for installation and use provided on the website. Lastly, the tool can capture all 56 subcarriers for the 20Mhz channel or 114 subcarriers for the 40Mhz channel [12].

At the time of writing this report, a third CSI platform known as the ESP32 CSI tool was introduced. The tools previously mentioned require hardware to support a NIC card and
additional hardware to transmit. The ESP32 is a standalone all-in-one microcontroller found on microcontroller boards. The ESP32 can be programmed to operate as an access point or active wireless client and collects CSI data using onboard storage. The ESP32 tool can capture up to 64 subcarriers although due to bandwidth limitations for processing the CSI in real time it is recommended to reduce the capture size to 32 subcarriers or less [10].

As mentioned before, the ESP32 tool was not known at the time of selecting a CSI Tool for this project. Therefore, the choice of CSI platforms was largely personal preference between the Linux 802.11n CSI Tool and Atheros CSI Tool. The Linux 802.11n CSI Tool was chosen for its many available resources, stable track record, straightforward hardware selection, and quick processing MATLAB utilities. An 802.11n CSI Tool Users Guide was developed in support of this study to provide future researchers with detailed instructions on installation, operation, and troubleshooting (see Appendix A).

**Tool Installation**

This section provides a brief overview of the 802.11n CSI Tool installation. Refer to the “GETTING STARTED” section of the User’s Guide for more details (see Appendix A). To install the Linux 802.11n CSI Tool, I used a small form factor PC with an empty PCI-e card slot located on the motherboard to install the IWL 5300 WIFI module. The 802.11n CSI Tool also requires a specific Linux kernel version and a supported Linux distribution installed. In addition, I installed a modified firmware and custom open source wireless drivers to effectively operate the IWL 5300 for CSI logging. Lastly, I selected a mode to run the tool and collect CSI data on appropriate sources. A description of the different modes is given in the next section. Alongside this study and the User’s Guide previously mentioned, I developed and created instructional videos to allow future researchers a quick start on CSI data collection. The videos are accessible via links provided in this study.
The 802.11n CSI Tool relies on a closed source custom firmware image that only works on the IWL 5300 module. The interface on the IWL5300 is Mini PCI-e. Mini PCI-e slots can be found on some mini-ITX motherboards, but PCI-e slots are much more common. Therefore, I used an adapter to install the IWL 5300 into a PCI-e slot. An instructional video set was created for future researchers to perform the installation with ease. Navigate to the Instructional Videos folder and follow “Step 1. mini PCI-E to PCI-E Installation” for the adapter installation. Then proceed to install the PCI-e card using “Step 2. PCI-E Computer Installation” as a guide (see Appendix A).

The supported operating systems are Ubuntu 12.04 and Ubuntu 14.04. Installing the latest supported version of Ubuntu provides the latest version of supported drivers and firmware. This is recommended to increase chances of compatibility with the user’s computer hardware. The operating system used in this project was Ubuntu 14.04.5 - x86_64 (Trusty Tahr). Navigate to the Instructional Videos folder and follow “Step 4. Ubuntu Baseline” to configure Ubuntu and install the Linux kernel 4.2 (see Appendix A).

The official installation instructions can be found on the Linux 802.11n CSI Tool Website, however the section regarding driver modifications can be challenging to follow. Therefore, it is recommended to watch the installation video as a supplement to the website instructions. Navigate to the Instructional Videos folder and follow “Step 5. CSI Tool Installation.” The installation guide introduces an alternative method for determining the correct kernel tag that allows the installer to determine the tag name precisely. It also shows a number of useful integrity checks to verify the installation steps. After following the installation instructions, the 802.11n CSI Tool is ready to use (see Appendix A).
Tool Operation

The 802.11n CSI Tool has four modes of operation: Client, AP, IBSS (ad-hoc), and Monitor. In Client mode, the captured CSI packet is transmitted from the access point to the connected Intel 5300 NIC. The packet can be sent either directly to the client’s address or as a broadcast. In AP mode, the captured CSI packet is transmitted by a connected client. In independent basic service set or IBSS (ad-hoc) mode, the packet is transmitted by another station directly connected to the client. In Monitor mode, the CSI packet is modified to have the same fixed address for the sender and receiver. The Client mode of operation was selected for this study to provide stability and consistency in capturing CSI (see Appendix A).

Operating the CSI Tool is divided into three steps: starting the CSI logger, connecting to an access point, and generating CSI data. An overview of the instructions to operate the CSI logging tool is given below. Refer to the “USING 802.11n CSI TOOL” section of the User’s Guide for more details (see Appendix A).

Verify that the 802.11n driver is unloaded by logging into Ubuntu and opening a terminal window. Inside the terminal window, type `lspci -k | grep -i network` and press the enter key. A list of Network Controllers recognized by Ubuntu will display in the terminal output. An example of the expected terminal output is shown below.

![Network Controller](image)

Figure 2. Network Controller

Look for the Network Controller, “Intel Corporation Ultimate N Wi-Fi Link 5300.” Verify there is no mention of “Kernel driver in use” below the Network Controller. If it appears the default driver is loaded, then attempt to unload the default driver before loading the custom
Unload the driver by typing `sudo modprobe -r iwlwifi mac80211` and pressing the enter key. The next step is to load the custom driver by typing `sudo modprobe iwlwifi connector_log=0x1` and pressing the enter key. This will load the correct driver with CSI logging enabled.

The Intel 5300 custom firmware is now loaded. The next step is connecting to an access point. There are many access points that support the 802.11n standard. Select one and follow the manufacturer’s guide to access the configuration webpage. Verify the access point has encryption turned off. Encryption options typically available are WPA, WPA2, and WEP. A simple access point solution is to use the hotspot feature on a cellphone with no password. However, this may affect performance. Navigate to the Instructional Videos folder and follow “Step 6. CSI Tool Operation” for instructions on creating a mobile hotspot (see Appendix A). I caution use of a mobile hotspot if using a sample rate faster than 1 packet per second. In testing, I discovered packet drops and eventually a connection drop would occur with this configuration. For this reason, I switched to a different access point mentioned in the next section, “EXPERIMENTS.” Note that using a different access point does not change the configuration setup.

Once the access point is setup, connect to the AP from the client by following the next set of instructions. Bring up the IWL 5300 NIC by opening a terminal, typing `sudo ip link set wlan0 up`, and pressing the enter key. If the client system has multiple wireless cards installed, then `wlan0` may be the wrong wireless device identifier for the IWL 5300 NIC. Make sure to replace all commands with `wlan0` using the correct identifier. Next, type `ifconfig` and press the enter key. Verify the wlan device is listed in the terminal output. A figure showing a similar output to what is expected is given below.
Now the receiver is ready to connect to the access point (AP). First, scan for access points by typing `sudo iw dev wlan0 scan | grep -i ssid_name` where the “ssid_name” is the access point service set identifier (SSID) and press the enter key. SSID is often commonly referred to as the Wi-Fi name. I configured the access points Wi-Fi name to be “AndroidAP.” The Wi-Fi name of the access point is displayed in the command output. If nothing is output, recheck the Wi-Fi name spelling and verify the access point is visible. Then rescan for the access point. Another option is to remove the `| grep -i ssid_name` part of the command to view all visible access points. After verifying the access point is visible, type `sudo iw dev wlan0 connect ssid_name` and press the enter key to connect. Make sure to replace the ssid_name portion of the command with the correct SSID! Wait up to 30 seconds for the connection to complete.

To verify the connection status for the wlan0 device, type `sudo iw dev wlan0 link` and press the enter key. Verify the connection by observing the correct SSID of the access point displayed in the terminal output. For example, if the SSID is “AndroidAP” then the command output should show something like the image in the figure below.
The next step is to request an IP address from the access point. An alternate option is to create a static IP address. Creating a static IP address is not covered but there are many tutorials available on the web. Type `sudo dhclient wlan0` and press the enter key to request an IP address. Then type `ifconfig` and press the enter key to verify wlan0 has an IP address. Note the figure below with the yellow arrow highlighting the IP address given to wlan0.

```
Figure 5. IP Address
```

Now I am ready to start the 802.11n CSI Tool and capture CSI data. To generate CSI data, I will need two terminals open. Terminal one will be used to run the CSI data logging application. Terminal two will be used to trigger a response from the access point. The access point will then send packets with CSI information to be extracted by the receiver system. The first step is to setup the CSI logging tool. Open two terminals, then in terminal one, type `sudo linux-80211n-csitool-supplementary/netlink/log_to_file csi.dat` and press the enter key. The command in terminal one is shown in the figure below. Make sure to rename the “csi.dat”
portion of the command to a unique name so as not to overwrite CSI files on subsequent data collections.

![CSI Logging Command](image.png)

**Figure 6. CSI Logging Command**

Leave terminal one running the CSI logging tool and in terminal two, type `ifconfig` and press the enter key. Use wlan0’s IP address to determine the access point’s IP address. Look at the IP address obtained from the access point and determine the access point’s IP address. For example, if given an IP address of 192.168.43.110 then the access point’s IP address is likely 192.168.43.1. Notate the access point’s IP address and in terminal two, type `ping xxx.xxx.xxx.xxx` and press the enter key. In this case, I typed `ping 192.168.43.1` and pressed the enter key to ping the access point. Both terminals with each command in their respective terminal is shown in the figure below.

![Ping Access Point](image.png)

**Figure 7. Ping Access Point**

In terminal 1, verified that the CSI Tool is logging CSI data. Several lines of received `xxx bytes:id: xx val: x seq: x clen: xxx` should be visible. Verify the expected terminal output with the figure below.
To stop logging CSI data, press Ctrl + C in each terminal to terminate CSI data collection. In terminal one, type `ls -l | grep -i csi.dat` and press the enter key. Make sure to replace “csi.dat” with the filename used. Verify the CSI file contains data in the terminal output by observing the file size. The figure below points out the location of the file size from the command output.

At this point CSI data is being generated and captured (see Appendix A).
Experiments

Prior to performing the experiments in this study, I performed several steps. First, I designed and built a reflector to act as the object. Second, I ran the 802.11n tool through stress testing. This involved testing packet rates against the stability of the CSI logging tool, running the tool for prolonged periods of time, and analyzing the data to determine if the reflector was detected. Third, I tested the tool in several locations to select a controllable environment that would yield the best results for capturing stable background noise (no fluctuations). A stable baseline noise was needed to detect object shapes with a high measure of success.

Two experiments were designed for this project, each with a separate goal. Experiment one’s goal was to capture CSI data capable of identifying the current angle of an object. Experiment two’s goal was to capture CSI data capable of identifying the shape of an object. Due to the subtle changes in experiment one and the increased complexity of the goal, the analysis needed is out of scope of this paper. Experiment one was specifically performed for future research and development of machine learning algorithms. Both experiments were conducted over the course of several weeks in a 24’ by 26’ garage. To develop a noise baseline, background CSI was collected prior to each data collection. All changes to the environment were scripted and timed during each experiment. A list of the events is provided later in this section followed by more details regarding the testing environment in the next section. Lastly, the experiment was performed within a static environment during the same time each day to limit the chance of background noise changes between experiments and produce comparable results.

In experiments one’s setup, there are 10 experimental settings shown in the table below. Each setting configuration dictates the transmitter, receiver, and the center of the object remain
in a fixed position. The only change is the angle of the reflector. When positioning the reflector, it should rotate around the center point of its bottom edge.

Table 1. Object Angles

<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angle of Object</td>
<td>0°</td>
<td>+15°</td>
<td>+30°</td>
<td>+45°</td>
<td>+60°</td>
<td>+90°</td>
<td>-15°</td>
<td>-30°</td>
<td>-45°</td>
<td>-60°</td>
</tr>
</tbody>
</table>

A visual aide to “Index 2,” showing the object’s angle in reference to the transmitter and receiver, is shown in the figure below. The transmitter and receiver were placed six feet apart. The object’s center was placed three feet from the transmitter and receiver along the x-axis and three feet along the y-axis.

![Figure 10. Reflector Angle 15 Degrees](image)

For experiment two’s setup, there are three experimental settings shown in the table below. Each setting configuration dictates the transmitter, receiver, and the center of the object remain in a fixed position once placed. The only change is the shape of the reflector. When positioning the reflector, it should rotate around the center point of its bottom edge. Row two in the table indicates the different shapes for each setting configuration. Row one indicates a
reference name to the object shape types that is used in the rest of the paper.

<table>
<thead>
<tr>
<th>Shape Type</th>
<th>Shape 1</th>
<th>Shape 2</th>
<th>Shape 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Mirrored “L”</td>
<td>“L”</td>
<td></td>
</tr>
</tbody>
</table>

A visual aide of the experiment setup showing the reflector as “Shape 2” is shown in the figure below. Note that the “Open” shape type is the reflector without being folded. The transmitter and receiver were placed six feet apart. The object’s center was placed three feet from the transmitter and receiver along the x-axis and three feet along the y-axis. The placement selection for experiment one and two yielded the best results for CSI capture during preliminary testing.

Figure 11. Mirrored L Shape

**Hardware**

The receiver used in this study was a custom-built Shuttle XPC. The relevant internal hardware is a customized network interface card (NIC). The customized NIC is a combination of
a Mini PCI-e to PCI-e Adapter with the IWL5300n module (see Appendix A). It contains three external dipole antennas installed in fixed positions throughout each experiment.

Imagery of the receiver is shown in the figure below.

![Image of Shuttle XPC System]

**Figure 12. Shuttle XPC System**

In terms of reflectors, the larger the shape of the reflector the easier it is to detect. The object used for detection was a custom built 4’x4’ square reflector panel with a wood frame. A solid reflective surface was installed on one side of the wood frame. A reflective surface is defined as anything that conducts electricity. In this study, I chose aluminum sheeting because of its good conductive rating and low cost. The reflector was separated into two 2’x4’ panels and connected with metal hinges down the middle. This allowed the reflector to fold upon itself and be configured for different shapes. Imagery of the reflector is shown in the figure below.

![Image of Reflectors]

**Figure 13. Reflector Panels**
The transmitter or access point used was a Netgear N600 router. The N600 supports the IEEE 802.11 b/g/n standards for 2.4Ghz frequency band and the IEEE 802.11 a/n standards for 5.0Ghz frequency band. In addition, it contains two internal transmitter and receiver pairs. In this study the N600 was set to the manufacturers default settings and then configured as an open access point. The 2.4Ghz frequency band was solely used to ensure compatibility and stability with the CSI logging tool. Imagery of the Netgear N600 router is shown in the figure below.

![Netgear N600](image)

Figure 14. Netgear N600

For each experiment, data was collected at 50 samples per second to provide a higher data sample set during analysis. This was accomplished by setting the ping interval to 20ms. The ping command used to accomplish this task was `sudo ping -D -r -v -i .02 xxx.xxx.x.x` where the “x” represents the IP address of the access point. A list of events was selected and incorporated into each experiment to determine how well different scaled events could be detected with minimal analysis. The events and their description are provided in the table below.

<table>
<thead>
<tr>
<th>Event</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflector</td>
<td>Move reflector into position</td>
</tr>
<tr>
<td>Garage Door</td>
<td>Close the garage door</td>
</tr>
<tr>
<td>Garage Door</td>
<td>Open the garage door</td>
</tr>
<tr>
<td>Vehicle</td>
<td>Park vehicle in garage</td>
</tr>
<tr>
<td>Walking</td>
<td>Walk around experiment</td>
</tr>
</tbody>
</table>
As stated earlier, the experiments were conducted in the two-car garage shown in the figure below. One vehicle was permanently removed from the garage during the experiments. The other, a 2011 Honda CRV, was used in the “Vehicle” event during the experiments. All components were placed carefully per the design in the previous section. The distances between the transmitter, receiver, and reflector were recorded for each data collection. Concrete blocks were placed to hold the reflector in its designated position and to prevent other components from moving easily during measurement activities and events. The garage floor was marked to allow for accurate repetitive placement of the reflector and to designate walking patterns. The vehicle parking was aligned within a specific range by marking the adjacent wall for the driver to stop next to when it reached the driver’s side mirror. The only components moved during the experiment were specific to each event. Hence, the reflector was positioned during the reflector event and the garage door was moved from closed to open and open to closed during its respective events.

The transmitter and receiver were placed on matching stools at six feet apart. The height measured for the center antenna on the IWL5300 was 2’1.5”. The height measured for the top of the N600 was 2’2.5”. The placement was intended to center the components in height to the reflector’s center to provide good signal reflections and avoid signal integrity issues. A total of eight concrete blocks were used to hold the reflector and secure the placement of the receiver and transmitter. The receiver was remotely controlled via secure shell (SSH). The physical connection used was an Ethernet cable connected to the Shuttle XPC onboard NIC and routed to a laptop. The laptop was connected to the receiver throughout the entire experiment in order to make periodic visual integrity checks on the CSI logging tool. The heating element used was a 50K-80K BTU convection propane tower portable heater. I used the heating element to pre-heat
the garage to a nominal temperature prior to the experiment. In most circumstances this would not be necessary. However, in this study, I encountered connection issues discussed in the TROUBLESHOOTING section of Chapter 4.

Figure 15. Test Environment
CHAPTER 4. DATA COLLECTION

Data Sets

CSI data sets were collected for each experiment based upon the number of configurations in each experiment. Dry runs were conducted prior to each experiment to verify the CSI data captured contained valid data. Following the instructions in the “TOOL OPERATION” section of Chapter 3, two terminals were used to remotely control the CSI data collection process. Terminal one was used to start the logger and provide visual checks for CSI data captures throughout the experiment. Terminal two was used to control the CSI capture rate and sample size. This was accomplished by pinging the access point at a predetermined packet interval. Each terminal output was logged to text files, and a readme file with the experiment details was generated.

As mentioned earlier, experiment one consisted of ten data sets. One data set generated for each designated angle in which the reflector was configured. Experiment two consisted of three data sets. One data set for each designed shape. A master guide table was generated for each experiment to enact each event in a specific order with specific time periods. The master guide was used as a template to record event times for all data collections in each experiment.

The master guide for experiment 1 is shown in the table below. Data collected for experiment 1 was recorded in 30-minute blocks. Main events were set in 15-minute increments except for the garage door and vehicle events in the third 30-minute segment. Those events were set in shorter increments to reduce the temperature drop in the garage once the garage door was opened. The first 30-minute segment was used to collect background noise and place the reflector in its unique configuration. The second 30-minute segment was used to generate a unique CSI signature through the specific event of walking at a measured pace along a
designated path around the experiment. The third 30-minute segment was used to generate three unique CSI signatures from parking a 2011 Toyota CRV in the garage. The three events to accomplish this task were opening the garage, parking the vehicle, and closing the garage. Lastly, walking around the experiment was performed again to note any detectable changes in its signature after the vehicle was added to the background environment.

Table 4. Masters Guide for Experiment 1

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Event</th>
<th>Temp</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/9/2020</td>
<td>19:30:00</td>
<td>Heat Garage</td>
<td>record temp, warm up</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:00:00</td>
<td>Start Data Recording</td>
<td>record temp, photo doc, take measurements</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:02:00</td>
<td>Garage Exit</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:15:00</td>
<td>Place Reflector</td>
<td>record temp</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:17:00</td>
<td>Garage Exit</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:30:00</td>
<td>Start Data Recording</td>
<td>record temp, photo doc, warm garage (optional)</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:45:00</td>
<td>Start Data Recording</td>
<td>record temp, photo doc, take measurements</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:00:00</td>
<td>Garage Exit</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:05:00</td>
<td>Walk Around</td>
<td>walk around experiment (path is marked)</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:10:00</td>
<td>Garage Exit</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:15:00</td>
<td>Stop Data Recording</td>
<td>record temp, photo doc, take measurements</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:30:00</td>
<td>Start Data Recording</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:32:00</td>
<td>Garage Door</td>
<td>record temp before opening garage door</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:35:00</td>
<td>Car Enters</td>
<td>record temp after parking car</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:37:00</td>
<td>Garage Door</td>
<td>record temp after closing garage door</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:40:00</td>
<td>Heat Garage</td>
<td>turn on heater, record temp</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:45:00</td>
<td>Walk Around</td>
<td>walk around experiment (path is marked)</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:47:00</td>
<td>Garage Exit</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>22:00:00</td>
<td>Stop Data Recording</td>
<td>record temp, photo doc, warm garage (optional)</td>
<td></td>
</tr>
</tbody>
</table>

The master guide for experiment 2 is shown in the next table below. The data was recorded in 120-minute blocks with the total duration of CSI data collection ranging from 90 to 120 minutes. Main events were set in 30-minute increments except for the events occurring between opening the garage door and walking around the experiment. Those events range from 5 to 15-minute increments to limit exposure to outside temperatures. The first 30-minute segment was used to collect background noise. The second 30-minute segment was used to generate a
unique CSI signature created after placing the reflector. The third 30-minute segment is similar to experiment one. I sought to generate multiple CSI signatures through the events of parking a vehicle in the garage. Lastly, the fourth segment was used to walk around the experiment.

Table 5. Masters Guide for Experiment 2

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Event</th>
<th>Temp</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/9/2020</td>
<td>19:30:00</td>
<td>Heat Garage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:00:00</td>
<td>Start Data Recording</td>
<td>record temp, photo doc, take measurements</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:02:00</td>
<td>Garage Exit</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:30:00</td>
<td>Place Reflector</td>
<td>record temp</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>20:32:00</td>
<td>Garage Exit</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:00:00</td>
<td>Garage Door</td>
<td>record temp before opening garage door</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:05:00</td>
<td>Car Enters</td>
<td>record temp after parking car</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:10:00</td>
<td>Garage Door</td>
<td>record temp after closing garage door</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:15:00</td>
<td>Garage Exit</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:30:00</td>
<td>Walk Around</td>
<td>walk around experiment (path is marked)</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>21:32:00</td>
<td>Garage Exit</td>
<td>check CSI tool before exiting</td>
<td></td>
</tr>
<tr>
<td>1/9/2020</td>
<td>22:00:00</td>
<td>Stop Data Recording</td>
<td>validate CSI data, backup data, photo doc</td>
<td></td>
</tr>
</tbody>
</table>

A plot showing the CSI signatures of different events from experiment 2 is shown in the figure below. This data set was collected from antenna receiver one, subcarrier nine. Details on the specific events I was able to detect through a simplified process of principal component analysis will be covered in the next chapter.

Figure 16. CSI Event Signatures
The recorded events for each data set can be accessed here: Event Logs

The data sets for each experiment and their respective support files can be access here: Data Sets

**Troubleshooting**

During the data collection of experiment two, a connectivity issue was discovered. The wireless connection between the transmitter and receiver would occasionally drop. In some cases, I was unable to re-establish the connection to the access point. To resolve the issue, I rebooted the receiver system and the access point as well. However, this would not do because it disrupted the data collection during the experiment. I discovered a large number of connection drops occurred during the events associated with parking the vehicle. A temperature gauge was placed on top of the receiver to measure the temperature of the garage. It was noted that the temperature in the garage would drop by 10 degrees or more while the garage door was left open. This was due to extremely cold weather. Before conducting any experiments in the garage, several tests were performed indoors without any connection drops. Therefore, the equipment was moved back inside and run through several more endurance tests to determine root cause. Unfortunately, the results yielded no connection drops while testing inside. Hence, nothing out of the ordinary was discovered. I determined the main differences between the indoor and outdoor environments were temperature, the environment background, and the events used during the experiment. Next, I proceeded to systemically apply tweaks to resolve the issue.

First, the garage was heated, and the temperature raised to be within range of nominal indoor temperatures. This was done to make sure cold temperatures were not adversely affecting the hardware. In most cases, the heating element was used prior to the experiment. However, in a few instances, the outside temperature caused the garage temperature to drop below the acceptable threshold. This always occurred during the events with the garage door open.
Therefore, the heating element was run during the experiment in these instances. Temperatures were recorded during every event to determine if colder temperatures were having a negative effect on the hardware. The results showed a significant reduction in connection loss with most drops occurring when the temperature was lower than 50 degrees Fahrenheit. However, there were still occasional connection drops. This led me to believe that more than one factor was causing connection loss.

Second, the sample rate was reduced from 100 samples per second to 50 samples per second to relieve strain on the receiver. During testing, it was discovered that drops were less likely to occur with a slower sample rate. In addition, the CSI logging tool combined with logging the terminal output and the operating system’s background processes caused a strain on the receiver’s resources. By reducing the samples per second, I noticed an increase in system performance and connection stability.

Lastly, after completing experiment two, during which connection drops were still occurring, the recording time was reduced to 30-minute blocks for experiment 1. Note that experiment two was run before experiment one. Experiment one was conducted with the new event timing schedule, the nominal temperature adjustment, and the reduced sample rate. The results yielded no more connection drops. This led me to believe there was a culmination of issues that led to the connection drop. When each of the three solutions were attempted individually, connection drops would occur. It was only after combining the three adjustments that the issue was fully resolved.
CHAPTER 5. DATA ANALYSIS

Pre-Analysis Setup

To analyze CSI data, MATLAB or Octave must be installed. In this study, MATLAB was used exclusively for analysis. To setup the MATLAB environment, follow the instructions given in the ANALYZING CSI DATA section of the User’s Guide (see Appendix A). After loading the “.dat” CSI log file into MATLAB, locate the loaded data in MATLAB’s workspace. By choosing the variable name “csi_trace” it should look similar to the figure below.

![Figure 17. CSI in MATLAB Workspace](image)

To view the first entry, type `csi_entry = csi_trace{1}` in the MATLAB “Command Window” and press the enter key. The structure of the CSI entry expected is shown in the figure below. A breakdown on each variable can be found in the INSPECTING CSI section of the User’s Guide (see Appendix A).

![Figure 18. First CSI Entry](image)
The information used for analysis is gathered from the “csi” variable in the “csi_entry” structure. The “csi” variable contains the raw CSI data. The CSI data is represented in a 3-D matrix of the MIMO channel for the connection in the format, “TX x RX x SF.” The first dimension, “TX,” represents the number of transmit antennas. The second dimension, “RX,” represents the number of receiving antennas per transmitter. Lastly, the third dimension, “SF,” represents the number of subcarrier frequencies collected per receiver. At this stage, the CSI values are based on Intel’s internal reference level [13]. To normalize the CSI, I used the “get_scaled_csi” function. Next, I created a MATLAB script to parse the data and graph the subcarrier frequencies of each transmitter and receiver pair. To incorporate time, I loaded a timing file based on the timestamped logs of the CSI logging tool’s terminal output. A commented version of MATLAB script labeled, “CSI_Plotter,” is provided to help automate plotting the SNR in MATLAB (see Appendix B).

**Analysis**

In this section, I focus solely on experiment 2. Note that I collected CSI data from two transmitters and the MATLAB code used for the analysis was based on that assumption. Before starting with PCA, I assessed the data and removed any outliers. This was where I saw spikes in the data that did not correspond to any event or a section that did not detect any events. Next, I assessed the data using Principal Component Analysis (PCA) and determined which set of subcarrier frequencies provided the most valuable information. In this study, I incorporated PCA through feature elimination. The feature elimination process involved an inspection of the subcarrier frequencies to determine which features are prevalent in the majority of subcarriers. I narrowed down the features to three significant events. The three events chosen were reflector placement, garage door opening, and garage door closing. The garage door events were selected
because they were the easiest events to detect. However, since the goal was to determine if I could detect a shape change in the reflector, the focus was on the reflector event.

Using MATLAB, I ran the CSI_Plotter script to plot the subcarrier frequencies in the format SNR [dB] over Time (seconds) across all the subcarriers. Next, I analyzed each subcarrier frequency individually and began feature extraction. This was accomplished by grading three key factors among each subcarrier: observable events, jitter, and event change. The observable events were the events I scripted for the experiment along with their approximate times. It should be noted that the event times were not exact but within two minutes of the actual event. This was due to time needed for various tasks associated with each event. Depending on the event, the tasks may have included entering the testing area, recording the environment temperature, verifying the CSI logging tool performance, walking to different locations, and positioning the reflector. The jitter was the variation of SNR associated with the data collection values when no events were occurring. Lastly, the event change refers to the level of change in the SNR between events. Again, I specifically focused on the reflector placement event since the goals are directly associated with it. A subcarrier frequency plot of experiment 2 showing the key factors I look for is shown below. It should be noted that the majority of scripted events were detectable.

Figure 19. Plot of Key Factor Analysis
Using the analysis criteria above, I analyzed the plots of each transmitter/receiver pair and narrowed the select group of subcarriers for deeper analysis. The analysis results are shown in the table below.

Table 6. Key Factor Analysis Results

<table>
<thead>
<tr>
<th>Transmitter</th>
<th>Receiver</th>
<th>Observable Events</th>
<th>Jitter</th>
<th>Event Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>4</td>
<td>Large</td>
<td>Small</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>4</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>4</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>4</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>4</td>
<td>Low</td>
<td>Large</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>4</td>
<td>Low</td>
<td>Medium</td>
</tr>
</tbody>
</table>

I completed feature elimination by selecting the subcarrier frequency group (SFG) with the best results. First, I determined the number of observable events for each subcarrier frequency. Second, I determined the level of jitter by “Low” being less than two decibels (dB), “Medium” being between two and four dB, and “High” being greater than four dB. I determined the level of event change by “Small” being less than one dB, “Medium” being between one and four dB, and “Large” being greater than four dB. The threshold values were determined through experimentation, looking at large amounts of data, and assessing the best performance. All experiments were both complex and unique so I would encourage anyone to re-evaluate these metrics for their test. From the table above, I found T2RB to be the optimal choice. The main difference being that the other transmitter receiver pairs had smaller event changes.

To detect shape change in the reflector, I analyzed the TR2B groups SNR plot and determined the margin by which the reflector placement was detectable. I calculated this in two steps. The first step was to calculate the difference in the average SNR across the TR2B subcarrier frequencies surrounding each event. Specifically, I calculated the average SNR across the subcarrier frequencies during the portion of time no reflector was present and after placing
the reflector. Then I calculated the difference between those values. The final step was to perform the first step’s calculations for each reflector shape and compare the reflector’s change in shape by the difference in SNR. Hence, by detecting a change in shape, I calculated the events SNR values and classified the reflector’s shape.

**Results**

The shape detection results for TR2B’s subcarrier frequencies are shown in the figure below. I found a clear indication of shape change in the reflector through the SNR representation. In this study, by following the process of subcarrier frequency group selection, the results were repeatable with minimal change in SNR variance.

Recall that TR1A displayed undesirable jitter and a smaller reflector event change. To witness the impact of selecting the least desirable SFG, see the TR1A SNR plot. Notice the difference in SNR change between reflector shapes one and three to be approximately 0.21 dB. In a static environment, this may be acceptable, but for real world applications, it is preferable to have a larger margin of variance. The larger margin provides a clear indication of a change in the shape of an object and robustness in noisier environments.
There are many studies that could provide methods of data smoothing and complex machine learning algorithms. These methods could contribute to future research of this study; however, they were beyond the scope of this paper. The focus of my analysis in this study was to provide a detailed guide for future research into a complex subject of detecting shapes with CSI. Through shape detection analysis, I have contributed to CSI related research.
CHAPTER 6. CONCLUSION

Summary

In conclusion, this study has taken a detailed look at channel state information and how signal paths are affected by physical objects and the surrounding environment. I have shown how to collect and analyze CSI in order to detect specific events. I have shown that with carefully structured experiments and specific event selection it is possible to use CSI to detect changes in object shapes with simple analysis techniques. During this journey, I meticulously detailed every step, and in doing so, provided a path for future researchers to follow. With this knowledge, others can use this study as a guide to further research into CSI experimentation and analysis.

Lessons Learned

The amount of data collected by the 802.11n CSI Tool and other CSI platforms is extremely large. There are thirty subcarrier frequencies per receiver and multiple receivers per transmitter. In this study, I analyzed two transmitters, each with three receivers. That resulted in one hundred eighty subcarriers. In experiment two, I collected data for up to two hours per configuration. At a packet rate of fifty packets per second for two hours, that resulted in a total of three hundred sixty thousand packets per subcarrier frequency! In order to perform shape change detection and event detection with a high measure of success, all subcarrier frequencies had to be analyzed. This amounted to a very time-consuming process. I used MATLAB to parse the data and auto generate plots. This helped to reduce the workload. However, using machine learning algorithms to automate outlier removal and PCA analysis would have been immensely helpful.

Early on during the testing phase, it was discovered the 802.11n CSI Tool did not provide valid time. The timing information in each CSI packet did not log actual time. Instead, it represented the Intel 5300’s clock value and reset to zero every 4300 seconds [13]. This was
unacceptable since I needed timing to determine the CSI signatures for specific events. To resolve this, I used a terminal application for Windows called “MobaXterm.” This tool had a feature allowing me to record and timestamp all terminal activity into a text file. When the 802.11n CSI Tool captured a CSI packet it would output a notification to the terminal. Using the timestamped notifications, I was able to determine the time for each CSI packet capture with millisecond precision and pull that information into MATLAB for analysis. Another method of capturing time would be to modify the tool directly. While running the CSI logging tool, a time file would be generated and local time, or a duration from the start time, would be logged for every packet sent.

**Future Work**

The MATLAB scripting created for this work still requires a large amount of user interaction. Further development in automating the process to detect environment changes and specific object shapes would help in producing faster feedback for the user. There are software applications available to collect CSI live. A database categorizing CSI signatures for unique events could be implemented with a CSI live tool. Once an event is detected, it is logged and sent to an application like the MATLAB script I used for processing. This would alleviate some processing power currently needed to sift through the large amounts of CSI data by discarding sections where no event occurs. Another option would be to use the process above as an event detection system in real world applications.

One more area for future work would involve the implementation of a directional antenna. Using a standard dipole antenna would require a short distance of separation between the transmitter and receiver to work. In a real-world application that requires longer distance than standard Wi-Fi range, a directional antenna would be necessary for both the transmitter and receiver. Directional antennas have a narrow beam which are required at longer distances.
Narrow beams provide more precision. Further testing would be needed to determine the effects on CSI capturing and shape detection performance.

Lastly, the size of the reflector and its distance between the transmitter and receiver impacted the level of change in the SNR. To fully understand the capability of detecting the shapes of various objects. It is important to develop criteria for shape, size, and placement. Further experiments in both static and dynamic environments would need to be conducted with different object sizes, shapes, and distances.
REFERENCES


APPENDIX A. USER’S GUIDE
User’s Guide

802.11n CSI Tool
Ath10k Driver

for Academic Use Only

Prepared by Andrew Lopez

August, 2019
## Revision Sheet

<table>
<thead>
<tr>
<th>Release No.</th>
<th>Date</th>
<th>Revision Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev. 0</td>
<td>08/04/19</td>
<td>User’s Manual Created</td>
</tr>
<tr>
<td>Rev. 1</td>
<td>04/17/20</td>
<td>Formatting</td>
</tr>
</tbody>
</table>
USER'S MANUAL

TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERAL INFORMATION</td>
<td>A1</td>
</tr>
<tr>
<td>SYSTEM SUMMARY</td>
<td>A2</td>
</tr>
<tr>
<td>GETTING STARTED</td>
<td>A3</td>
</tr>
<tr>
<td>USING 802.11n CSI TOOL</td>
<td>A4</td>
</tr>
<tr>
<td>ANALYZING CSI DATA</td>
<td>A5</td>
</tr>
<tr>
<td>ATH10K QUICKGUIDE</td>
<td>A6</td>
</tr>
</tbody>
</table>
TABLE OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Ubuntu Releases</td>
<td>A3-1</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Don't Upgrade</td>
<td>A3-2</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Declined Upgrade</td>
<td>A3-2</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Driver Modification Instructions</td>
<td>A3-3</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Network Controller</td>
<td>A4-1</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Wlan0 Device</td>
<td>A4-2</td>
</tr>
<tr>
<td>Figure 7</td>
<td>IW Status</td>
<td>A4-3</td>
</tr>
<tr>
<td>Figure 8</td>
<td>IP Address</td>
<td>A4-3</td>
</tr>
<tr>
<td>Figure 9</td>
<td>CSI Logging Command</td>
<td>A4-4</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Ping Access Point</td>
<td>A4-4</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Capturing CSI</td>
<td>A4-5</td>
</tr>
<tr>
<td>Figure 12</td>
<td>CSI Data</td>
<td>A4-5</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Add To Path</td>
<td>A5-1</td>
</tr>
<tr>
<td>Figure 14</td>
<td>csi_trace</td>
<td>A5-2</td>
</tr>
<tr>
<td>Figure 15</td>
<td>csi_entry</td>
<td>A5-3</td>
</tr>
<tr>
<td>Figure 16</td>
<td>SNR Plot</td>
<td>A5-5</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Network Controllers</td>
<td>A6-1</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Device Name</td>
<td>A6-1</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Ethtool</td>
<td>A6-2</td>
</tr>
</tbody>
</table>
STATEMENT OF PURPOSE

The Purpose of this document is to provide the user with an approach to building a system capable of utilizing the 802.11n CSI Tool. The user will understand the operational capability of the IWL5300 module, the CSI Tool, CSI data analysis, and additional sources capable of effectively generating CSI data.

CSI OVERVIEW

The CSI Tool was built on the Intel Wi-Fi Wireless Link 5300 802.11n MIMO radios. The tool uses proprietary firmware and open source Linux wireless drivers. The IWL5300 provides 802.11n channel state information in a format that reports the channel matrices for 30 subcarrier groups. This amounts to about one group for every two subcarriers at 20 MHz or one in four at 40 MHz. Each channel matrix entry is a complex number, with signed 8-bit resolution for the real and imaginary parts. The complex numbers specify the gain and phase of the signal path between a single transmit-receive antenna pair.¹

With CSI, it is possible to see the extent of frequency-selective fading. Weak, deeply faded subcarriers require the transmitter to expend more power to deliver the same performance. This requires a higher RSSI to achieve optimal performance.¹

ACRONYMS AND DEFINITIONS

CSI – Channel State Information (CSI) describes how a signal propagates from the transmitter to the receiver and represents the combined effect of, for example, scattering, fading, and power decay with distance.

MIMO – Multiple-Input Multiple-Output (MIMO) is a wireless technology that uses multiple transmitters and receivers to transfer more data at the same time

RSSI – Received Signal Strength Indicator (RSSI) is an estimated measure of power level that a RF client device is receiving from an access point or router.
SYSTEM CONFIGURATION

Users will need a computer with an empty PCI-e card slot located on the motherboard to install the IWL 5300 WIFI module. To utilize the 802.11n CSI tool, the user will install a specific Linux kernel version and the recommended Linux distribution. In addition, the user will install a modified firmware and custom open source wireless drivers. The user will run the tool in different modes and collect CSI data on appropriate sources.

BILL OF MATERIALS (BOM)

Custom PC – Computer with at least one empty PCI-e slot on the motherboard and capable of running the i386 or x86_64 architecture.

IWL5300 Wi-Fi Module - IEEE 802.11a/b/g/Draft-N1 wireless network adapter that operates in both the 2.4 GHz and 5.0 GHz spectra. Available in both PCI-e* Mini Card and Half Mini Card form factor. Delivers up to 450 Mbps of bandwidth.

Compex WLE600VX Wi-Fi Module - Qualcomm-Atheros QCA9882 Device. Dual Band 2×2 MIMO 802.11ac Wave 1 Module

Mini PCI-e to PCI-e Adapter – PCI-e adapter for mini PCI-e Wi-Fi modules

2x Screws - #3-48 ¼" – Secures the Wi-Fi Module to the PCI-e adapter

2x Nuts - #3-48 ¼" – Secures the Wi-Fi Module to the PCI-e adapter

2x Washers - #3-48 ¼" – Secures the Wi-Fi Module to the PCI-e adapter

Flathead Screwdriver – Used to remove metal enclosure

Phillips Screwdriver – Used remove bracket from adapter board and secure IWL 5300

Electrical Tape – Protects antenna wiring from metal enclosure

Diagonal Pliers – Used to open metal enclosure
IWL5300 INSTALLATION

Before proceeding with this section, gather all materials from the BOM in section 2.2. The 802.11n CSI Tool relies on a custom firmware image that only works on the IWL5300. Mini PCI-e slots can be found on some mini-ITX motherboards, but PCI-e slots are much more common. Therefore, I used an adapter to install the IWL 5300 into a PCI-e slot. To install the IWL 5300 PCI-e adapter, navigate to the Instructional Videos folder and follow “Step 1. mini PCI-E to PCI-E Installation.” Then proceed to install the PCI-e card using “Step 2. PCI-E Computer Installation” as a guide.

INSTALLING UBUNTU

There are several methods of installing Ubuntu. I will cover two of the most common methods: Installing from a live cd or a USB stick. There are many tutorials available online but I recommend using the official Ubuntu Tutorial as a detailed reference to this guide.

1. Preparing the ISO

Navigate to the old Ubuntu releases and scroll down to find Ubuntu 12.04 or Ubuntu 14.04

*Note the image used with this guide was Ubuntu 14.04.5 - x86_64 (Trusty Tahr)*

Follow this link to directly download Ubuntu 12.04: Ubuntu 12.04.4 Release

Follow this link to directly download Ubuntu 14.04: Ubuntu 14.04.5 Release

Select the link under “Desktop Image” that best matches the computer architecture.

To create a live CD, burn the ISO image to disc

There is free image burning software available. Windows users can download imgburn
To create a USB stick, follow the tutorial: How to Create Ubuntu Live USB in Windows

2. Installing Ubuntu

Once completed, the operating system will prompt to upgrade to a newer version of Ubuntu.

![Don't Upgrade](image)

*Figure 22: Don’t Upgrade*

Select the “Don’t Upgrade” option as shown in the figure above. The upgrade prompt may occur periodically, and it is important to decline the upgrade. Doing this maintains a compatible operating system for the 802.11n CSI Tool.

After the upgrade prompt, there will be an information prompt. An image of the information prompt is shown in the figure below.

![Declined Upgrade](image)

*Figure 23: Declined Upgrade*

Select “OK” to continue.

SETTING UP UBUNTU

The 802.11n CSI Tool only supports Linux operating systems with a Linux kernel version between 3.2 and 4.2. This corresponds to Ubuntu 12.04 through Ubuntu 14.04. Since I wanted to use the latest drivers and firmware, I proceeded to install the latest supported version of Ubuntu.

Navigate to the Instructional Videos folder and follow “Step 4. Ubuntu Baseline” to configure Ubuntu and install the Linux kernel 4.2.
INSTALLING 802.11n CSI TOOL

The official installation instructions can be found on the Linux 802.11n CSI Tool Website. In addition, there is a video installation guide available to supplement the website instructions. It is recommended to watch the video while following the official installation instructions.

The video guide deviates slightly from the official instructions due to an issue with proper tag name identification. The majority of the content in this section was retrieved from “https://dhalperi.github.io/linux-80211n-csitool/installation.html.”

```
# Modify the line above with your Ubuntu kernel tag. First, determine your full kernel
# version by reading /proc/version_signature; then, look up the Ubuntu kernel tag at:
# https://people.canonical.com/~kernel/info/kernel-version-map.html

. /etc/lsb-release

# Setup local development environment

git remote add ubuntu git://kernel.ubuntu.com/ubuntu/ubuntu-$[DISTRIB_CODENAME].git

git pull --no-edit ubuntu $[UBUNTU_KERNEL_TAG]
```

Figure 24: Driver Modification Instructions

The link shown in the figure above, http://people.canonical.com/~kernel/info/kernel-version-map.html, is provided by the official instructions to look up Ubuntu kernel tags. However, at the time of writing this guide, the link does not contain a complete list of tags. In addition, some of the tag names are also mislabeled. There is an alternative method for determining the correct kernel tag, shown in the CSI Tool Installation video, that allows the installer to determine the tag name precisely.

Navigate to the Instructional Videos folder and follow “Step 5. CSI Tool Installation”.
USING 802.11n CSI TOOL
USING 802.11n CSI TOOL

Before attempting to use the tool for the first time, it is important to reboot the computer following the installation. After rebooting the system, follow the instructions below.

1. **Verify that the 802.11n driver is unloaded**
   
   Login to Ubuntu
   
   Open a terminal window
   
   Type `lspci -k | grep -i network -A 2` and press Enter.

   a list of network controllers recognized by Ubuntu will display in the terminal output.

   ![Network Controller](image)

   *Figure 25: Network Controller*

   Look for the network controller, “Intel Corporation Ultimate N Wi-Fi Link 5300.” Verify there is no mention of, “Kernel driver in use” below the network controller.

2. **Unload the driver if needed**
   
   Unload the driver by typing `sudo modprobe -r iwlwifi mac80211` and pressing Enter.

   If the error message, "FATAL: Module iwlwifi is in use” appears, type `sudo modprobe -r iwldvm iwlwifi mac80211` and press Enter.

3. **Load the custom driver**
   
   Type `sudo modprobe iwlwifi connector_log=0x1` and press Enter.

   *This will reload the driver with CSI logging enabled.*

   The operating system is now ready to collect CSI data. The next step is to configure an operating mode for the CSI logging tool.

   There are four modes: Client, AP, IBSS (ad-hoc), and Monitor.³

   In Client Mode, the packet is transmitted by the connected AP. The destination is either the local address or a broadcast address.³

   In AP Mode, the packet is transmitted by a connected client³

   In IBSS (ad-hoc) Mode, the packet is transmitted by another station in the connected IBSS. The destination is either the local address or broadcast address.³

   In Monitor Mode, the packet is addressed both to and from a hardcoded, fixed address.³

   This guide will only demonstrate Client Mode.
SETTING UP AN ACCESS POINT

Setting up an access point is specific to the hardware being utilized. If using a Raspberry Pi 3, then it is recommended to follow the guide on the SparkFun Website. Another option, is to build an access point on a Linux machine using the Hostapd application. Hostapd requires hardware capable of operating in “AP Mode.” If using an android cellphone to create a hotspot, then navigate to the Instructional Videos folder and follow “Step 6. CSI Tool Operation.” Make sure the access point is configured without a password. In other words, do not use encryption on the access point or the CSI tool will not work.

CONNECTING TO AN ACCESS POINT

Once the access point is turned on, navigate back to the computer system running the CSI tool.

1. **Bring up the wireless card**
   Open a terminal, type `sudo ip link set wlan0 up` and press Enter.
   
   *This will bring up the wireless card. If running multiple wireless network cards, then wlan0 may need a different value (wlan1 or wlan2 for example). Remember to replace all subsequent commands containing wlan0 with the correct value.*

   Type `ifconfig` and press Enter.
   
   *Verify the wlan device is visible.*

   ![Wlan0 Device](image)

   **Figure 26: Wlan0 Device**

2. **Connect to the access point (AP)**
   Scan for an access point by typing, `sudo iw dev wlan0 scan | grep -i ssid_name` where the “ssid_name” is the access point SSID setup to send CSI data and press Enter.
   
   *The SSID of the access point should be visible in the output of the command. If nothing is output, recheck the SSID and verify the access point is visible. Then rescan for the access point.*

   Type `sudo iw dev wlan0 connect ssid_name` and press Enter
   
   *Make sure to replace ssid_name with the correct SSID! This will connect the client system to the access point. Up to 30 seconds may elapse before the connection is made.*

   Type `sudo iw dev wlan0 link` and press Enter.
   
   *This will verify the connection status for the wlan0 device*
Verify the connection to the access point by observing “Connected” in the terminal output. Use the figure below as an example.

![IW Status](image)

**Figure 27: IW Status**

3. **Obtain an IP Address**
   
   Type `sudo dhclient wlan0` and press Enter.
   
   Type `ifconfig` and press Enter to verify wlan0 has an IP address.

![IP Address](image)

**Figure 28: IP Address**

**GENERATING CSI DATA**

To generate CSI data I will need two terminals. **Terminal one** will be used to run the CSI data logging application. **Terminal two** will be used to trigger a response from the access point. The access point will then send packets with CSI information to capture. This type of CSI data collecting is known as “Client Mode.”

1. **Setup CSI data logger**
   
   Open two terminals
   
   In terminal one, type `sudo linux-80211n-csitool-supplementary/netlink/log_to_file csi.dat` and press Enter.
   
   *Rename the “csi.dat” file to any name so as not to overwrite CSI files on subsequent data collects.*
4.0 Using The 802.11n CSI Tool

2. Ping the access point (AP)

Leave terminal one running the CSI logging tool.

In terminal two, type `ifconfig` and press Enter.

Use wlan0’s IP address to determine the access point’s IP address.

Observe the IP address obtained from the access point and determine the access point’s IP address. For example, an IP address of 192.168.43.110 would imply the access point’s IP address is likely 192.168.43.1.

In terminal two, type `ping “xxx.xxx.xx.xxx”` (access point’s IP address) and press Enter.

*From the previous example the command would be ping 192.168.43.1*

3. Verify CSI data logging

In terminal one, verify that the CSI tool is logging CSI data.

*There should be several lines of “received xxx bytes: id: xx val: x seq: x clen: xxx” being generated. An example of CSI data being generated is shown below.*
4. Using The 802.11n CSI Tool

4. Stop logging CSI data

Press Ctrl + C in each terminal to terminate the CSI data collection.

In terminal one, type `ls -l | grep -i csi.dat` and press Enter.

_Make sure to replace “csi.dat” with the filename used to log CSI data!

Verify the CSI file contains data by checking the file size. An image showing where to look on the command output is shown below.

![Figure 32: CSI Data](image-url)

At this point, CSI data has been generated and logged.
ANALYZING CSI DATA
5.0 Analyzing CSI Data

ANALYZING CSI DATA

This section provides a starting point for the user to analyze CSI data. The user will need MATLAB or Octave installed. In this guide, MATLAB will be used exclusively. The content in this section was retrieved from, “https://dhalperi.github.io/linux-80211n-csitool/installation.html.”

PARSING A CSI TRACE FILE

1. Setting up MATLAB Environment

Navigate to the Linux 802.11n CSI Supplementary GitHub repository. Click the “clone or download” button and select “Download Zip.”

Unzip the contents to the desired path choice.

Optional: To clone, in a Linux terminal type `git clone https://github.com/dhalperi/linux-80211n-csitool-supplementary.git` and press Enter. In Windows, download and install the GitHub software.

Launch the MATLAB Application.

In the MATLAB “Current Folder” Window, Navigate to “linux-80211n-csitool-supplementary-master.”

The folder downloaded from the Linux 802.11n CSI Supplementary GitHub repo.

Right Click the folder and select Add to Path -> Selected Folders and Subfolders.

![Figure 33: Add to Path]
2. Load CSI into MATLAB

Navigate to the MATLAB “Command Window.”

Type `linux-80211n-csitoosl-supplementary-master\matlab\` and press Enter.

*This command will change to the MATLAB directory in the CSI Tool supplementary material folder.*

Type `csi_trace = read_bf_file('csi.dat')` and press Enter.

*This command will read in the CSI trace file. Change “csi.dat” to the name of the desired CSI log file. Make sure csi.dat is located in the CSI Tool supplementary material folder. Otherwise, the absolute path must be used.*

Verify the csi_trace data was loaded into MATLAB’s “Workspace.”

![Workspace](image)

*Figure 34: csi_trace*

INSPECTING CSI

1. Review CSI fields

`csi_trace` is a 1x123 cell array, which holds 123 structs. This contains the CSI information for 123 received packets. Proceed to inspect one of the entries.

In the MATLAB “Command Window,” type `csi_entry = csi_trace{1}` and press Enter.
5.0 Analyzing CSI Data

A description of all the fields in the “csi_entry” structure is given below.

- **timestamp_low** is the low 32 bits of the NIC’s 1 MHz clock. It wraps about every 4300 seconds, or 72 minutes.\(^3\)
- **bfee_count** is simply a count of the total number of beamforming measurements that have been recorded by the driver and sent to userspace. The netlink channel between the kernel and userspace is lossy, so these can be used to detect measurements that were dropped in this pipe.\(^3\)
- **Nrx** represents the number of antennas used to receive the packet by this NIC, and **Ntx** represents the number of space/time streams transmitted. In this case, the sender sent a dual-stream packet and the receiver used all 3 antennas to receive it.\(^3\)
- **rssi_a**, **rssi_b**, and **rssi_c** corresponds to RSSI measured by the receiving NIC at the input to each antenna port. This measurement is made during the packet preamble. This value is in \(\text{dB}\) relative to an internal reference; to get the received signal strength in \(\text{dBm}\) I must combine it with the Automatic Gain Control (AGC) setting (**agc**) in \(\text{dB}\) and subtract off a magic constant.\(^3\)
- **perm** tells us how the NIC permuted the signals from the 3 receive antennas into the 3 RF chains that process the measurements. The sample value of [2 1 3] implies that Antenna B was sent to RF Chain A, Antenna A to Chain B, and Antenna C to Chain C. This operation is performed by an antenna selection module in the NIC and generally corresponds to ordering the antennas in decreasing order of RSSI.\(^3\)
5.0 Analyzing CSI Data

- **rate** is the rate at which the packet was sent, in the same format as the rate_n_flags defined above. Note that the antenna bits are omitted, as there is no way for the receiver to know which transmit antennas were used.\(^3\)
- **csi** is the CSI itself, normalized to an internal reference. It is a \(N_{tx} \times N_{rx} \times 30\) 3-D matrix where the third dimension is across 30 subcarriers in the OFDM channel. For a 20 MHz-wide channel, these correspond to about half the OFDM subcarriers, and for a 40 MHz-wide channel, this is about one in every 4 subcarriers. Which subcarriers were measured is defined by the IEEE 802.11n-2009 standard (in Table 7-25f on page 50).\(^3\)

Now I need to compute the CSI in absolute units, rather than Intel's internal reference level. I need to combine the RSSI and AGC values together to get RSS in dBm and include noise to get SNR. If there is no noise, I instead use a hard-coded noise floor of -92 dBm. I use the script “get_scaled_csi.m” to do this:\(^3\)

2. Normalizing the CSI
   In the MATLAB “Command Window,” type `csi = get_scaled_csi(csi_entry);` and press Enter.

   *The CSI is now a 1\(\times\)3\(\times\)30 matrix that represents the MIMO channel state for this link. Its units are in linear—i.e., not dB—voltage space.*\(^3\)

   The CSI is now normalized and provides unit noise.

PLOTTING SNR

1. Plot SNR in MATLAB
   To analyze the three different spatial paths on the measured link I can plot the SNR.

   In the MATLAB “Command Window”:

   Type `plot(db(abs(squeeze(csi)).'))` and press Enter.

   `squeeze()` turns `csi` into a 3\(\times\)30 matrix by removing the first singleton dimension. `db()` converts from linear (voltage) space into logarithmic (base-10, power) space. `abs` converts each complex number into its magnitude. The .’ operator transposes the squeezed CSI from 3\(\times\)30 matrix into a 30\(\times\)3 matrix and does not complement the complex numbers.\(^3\)

   Type `legend('RX Antenna A', 'RX Antenna B', 'RX Antenna C', 'Location', 'SouthEast')`; and press Enter.

   Type `xlabel('Subcarrier index')`; and press Enter.

   Type `ylabel('SNR [dB]')`; and press Enter.
The resulting SNR plot is shown below.

![SNR Plot](image)

**Figure 36: SNR Plot**

**COMPUTING EFFECTIVE SNR VALUES**

To conclude this section, I compute the Effective SNR from the CSI matrices. To do so, use the “get_eff_SNRs” script, which takes as input a CSI matrix and returns a 7×4 matrix of effective SNR values in linear (power) space.³

1. **Compute SNR in MATLAB**

   In the MATLAB “Command Window,” type `db(get_eff_SNRs(csi), 'pow')` and press Enter.

   *Observe the desired output format below.*

   ```matlab
   ans =
   
   22.3345   22.4192   23.0303   24.5096
   -156.5356 -156.5356 -156.5356 -156.5356
   -156.5356 -156.5356 -156.5356 -156.5356
   -156.5356 -156.5356 -156.5356 -156.5356
   -156.5356 -156.5356 -156.5356 -156.5356
   -156.5356 -156.5356 -156.5356 -156.5356
   -156.5356 -156.5356 -156.5356 -156.5356
   ```
The four columns correspond to the effective SNR using the four 802.11 modulation schemes. The modulation schemes are BPSK, QPSK, 16QAM, and 64QAM. The 7 rows correspond to the seven possible antenna selections. This amounts to three antennas and one, two, or three spatial streams. In particular, the first three rows correspond to single-stream transmissions with antenna A, B, or C. The next three rows correspond to dual-stream transmissions with antennas AB, AC, or BC. The last row corresponds to a three stream transmission using all antennas.\(^3\)

Note that this is a 1×3 link, so the only valid antenna configuration is SIMO with the single transmit antenna I measured. The other 6 rows correspond to a very small SNR, i.e., a large, negative dB.\(^3\)
ATH10K QUICKGUIDE
ATH10K SUMMARY

This section will cover how to verify a Wi-Fi module is using the ath10k open source driver, how to update the firmware, and how to acquire the source code. Before continuing with this section, make sure the hardware installed is an ath10k supported device. The ath10k driver supports Qualcomm Atheros 802.11ac, QCA98xx hw2.0, and QCA6174 based devices. Navigate to the Instructional Videos folder and watch “Step 1. mini PCI-E to PCI-E Installation” and “Step 2. PCI-E Computer Installation” for installation guidance.

DRIVER VERIFICATION

1. List the Network Controllers Ubuntu has identified
   Type `lspci | grep -i network` and press Enter.

   This lists all PCI devices connected to the system and identifies any network controllers. This will indicate whether Ubuntu successfully identified the wifi module.

   ![Network Controllers](image)

2. Determine the device name for the desired Wi-Fi module
   The wifi module installed in this system is the Compex WLE600VX. This is a Qualcomm Atheros QCA9882 device. Supported devices are listed on the ath10k wiki page. The image above gives us confidence that Ubuntu has selected the correct driver.

   Type `ifconfig -a` and press Enter.

   ![Device Name](image)

3. Confirm the driver is ath10k
   Type `ethtool -i wlan1` and press Enter.
The driver being used by wlan1 is “ath10k_pci.” This also lists the driver version and firmware version.

**UPDATING FIRMWARE**

1. **Compare firmware versions**  
   Using information from section 1, check the current firmware version. Compare the Wi-Fi module firmware version with the latest firmware available. At the time of writing this guide, the latest QCA9882 firmware is version 10.2.4.70.70.

2. **Acquire the latest firmware version**

3. **Copy the file to the ath10k firmware directory**  
   Type `cp firmware-2 /lib/firmware/ath10k/QCA988X/hw2.0/firmware-2.bin` and press Enter.

   *Note that “firmware-2” is an example name. The firmware name will vary. It is important to rename or remove the existing firmware after copying the latest firmware.*

4. **Reload the ath10k driver (or restart the system)**  
   Type `sudo modprobe -r ath10k_pci` and press Enter.

   Type `sudo modprobe ath10k_pci` and press Enter

   Verify the latest firmware version is in use:

   Type `ethtool -i wlan1` and press Enter.

   The ath10k driver should now have the latest firmware installed.
ACQUIRING SOURCE CODE

1. **Reviewing source code**


2. **Cloning the source code tree**

   Type `git clone git://git.kernel.org/pub/scm/linux/kernel/git/kvalo/ath.git` and press Enter.

   *This will clone the tree and allow the user to browse the source code. The ath10k driver is in the directory, “drivers/net/wireless/ath/ath10k.”*
REFERENCES

APPENDIX B. MATLAB CODE
%CSI_Plotter.m
%Developed by Andrew Lopez

%Load Data

% Modify path for linux-80211n-csitool-supplementary-master location

cd 'input\path\to\linux-80211n-csitool-supplementary-master\here';

% Modify path for desired .dat file to analyze

csi_trace = read_bf_file('input\path\to\dat\file\here');

% This variable is used for the loop iteration counter
sz = size(csi_trace);

% Set Variables

% Transmitter tracking variables for each packet


% Unknown tracking variable

rx_low = 0;

% Packet counter

count = 0;

% CSI placeholder for transmitter 1

d = 0;

% CSI placeholder for transmitter 2

e = 0;

Counters for NIC permuted signals for the 3 receive antennas

p = 0;

p_diff = 0;

% Main loop for process CSI data

for n = 1:length(csi_trace)

% Output CSI packet# being processed

fprintf('CSI Loop Iteration: %i out of %i\n', n, sz(1))


if perm(1) == 3 && perm(2) == 1 && perm(3) == 2
\( p = p + 1; \)

else
\( p\_\text{diff} = p\_\text{diff} + 1; \)
end

\% One Transmitter
\ifcheck(1) == 1
\tx1 = tx1 + 1;
\% SNR
\a = csi;
\a = squeeze(a);
\a = a.\';
\angle_a = \angle(a);
\a = abs(a);
\a = db(a);
\fi

\% Average the 50 packets per second i.e. convert to seconds
\ifcount < 50.0
\d = a + d;
\e = b + e;
\else
\d = d / 50.0;
\e = e / 50.0;
\fi

\if n == 51
\% initial subcarrier set
\rx1(:,1) = d(:,1);
\rx2(:,1) = d(:,2);
\rx3(:,1) = d(:,3);
\ry1(:,1) = e(:,1);
\ry2(:,1) = e(:,2);
\ry3(:,1) = e(:,3);
\end
\fi
\if n > 51
\% combine subcarriers for each receiver
\rx1 = horzcat(rx1,d(:,1));
\rx2 = horzcat(rx2,d(:,2));
\rx3 = horzcat(rx3,d(:,3));
\ry1 = horzcat(ry1,e(:,1));
\ry2 = horzcat(ry2,e(:,2));
\ry3 = horzcat(ry3,e(:,3));
\end
\fi
\% reset parameters
\count = 0;
\d = 0;
\e = 0;
end
count = count + 1;
end

% 2 Transmitters
if check(1) == 2
tx2 = tx2 + 1;
% SNR
a = csi(1,:,:);
b = csi(2,:,:);
a = squeeze(a);
a = a.:
angle_a = angle(a);
a = abs(a);
a = db(a);
b = squeeze(b);
b = b.:
angle_b = angle(b);
b = abs(b);
b = db(b);

% Average the 50 packets per second i.e. convert to seconds
if count < 50.0
d = a + d;
e = b + e;
else
d = d / 50.0;
e = e / 50.0;
if n == 51
% initial subcarrier set
rx1(:,1) = d(:,1);
rx2(:,1) = d(:,2);
rx3(:,1) = d(:,3);
ry1(:,1) = e(:,1);
ry2(:,1) = e(:,2);
ry3(:,1) = e(:,3);
end
if n > 51
% combine subcarriers for each receiver
rx1 = horzcat(rx1,d(:,1));
rx2 = horzcat(rx2,d(:,2));
rx3 = horzcat(rx3,d(:,3));
ry1 = horzcat(ry1,e(:,1));
ry2 = horzcat(ry2,e(:,2));
ry3 = horzcat(ry3,e(:,3));
end
%reset parameters
count = 0;
d = 0;
e = 0;
end
count = count + 1;
end

%Track number of packets without any transmitter information
if check(1) == 0
    tx0 = tx0 + 1;
end
%Track number of packets with 3 Transmitters
%If count is significant then implement parsing into code
if check(1) == 3
    tx3 = tx3 + 1;
    break
end
%Unknown value detected
if check(2) ~= 3
    rx_low = rx_low + 1;
end
end

count = 0;
t1 = 0;
t2 = 0;
t3 = 0;
t4 = 0;

%Depending on time file, you will need to adjust the origin
variable to set duration count to start a time 0 seconds
%shape 1
%origin = 75857;
%shape 2
%origin = 68362;
%shape 3
%origin = 76737;
%shape 4
%origin = 777875
%You must load correct time file before proceeding!
%Time Duration Loop
for i = 1:length(time)
    fprintf('Time Loop Iteration: %i out of %i \n', i, length(time))
    t1=time(i);
    [Y, M, D, H, MN, S] = datevec(t1);
    t1 = H*3600+MN*60+S;
    if count < 50
        t2 = t1 + t2;
    else
        % Convert to seconds
        x = round(t2 / 50);
        if i == 51
            t3 = x;
        end
        if i > 51
            t3 = horzcat(t3, x);
            t4 = horzcat(t4, x - origin);
        end
        count = 0;
        t1 = 0;
        t2 = 0;
    end
    count = count + 1;
end

%Run these loops individually to plot SNR plots for each Subcarrier Group
%Uncomment the savefig command after each loop to save figures
figure(1)
hold on
for j = 1:30
    plot(t4, rx1(j,:))
end
hold off
%savefig('CSI_All_Subcarriers_rx_a_1.fig')

figure(2)
hold on
for j = 1:30
    plot(t4, rx2(j,:))
end
hold off
%savefig('CSI_All_Subcarriers_rx_a_2.fig')

figure(3)
hold on
    for j = 1:30
        plot(t4, rx3(j,:))
    end
hold off
%savefig('CSI_All_Subcarriers_rx_a_3.fig')

figure(4)
hold on
    for j = 1:30
        plot(t4, ry1(j,:))
    end
hold off
%savefig('CSI_All_Subcarriers_rx_b_1.fig')

figure(5)
hold on
    for j = 1:30
        plot(t4, ry2(j,:))
    end
hold off
%savefig('CSI_All_Subcarriers_rx_b_2.fig')

figure(6)
hold on
    for j = 1:30
        plot(t4, ry3(j,:))
    end
hold off
%savefig('CSI_All_Subcarriers_rx_b_3.fig')

close all
    for j = 1:30
        figure(j);
        plot(t4.rx1(j,:))
        ylim([0 20])
        %savefig(sprintf('CSI_rx1_Subcarrier%d.fig',j));
    end
close all
    for j = 1:30
        figure(j);
        plot(t4.rx2(j,:))
        ylim([15 35])
        %savefig(sprintf('CSI_rx2_Subcarrier%d.fig',j));
    end
close all
for j = 1:30
    figure(j);
    plot(t4.rx3(j,:));
    ylim([15 30])
    %savefig(sprintf('CSI_rx3_Subcarrier%d.fig',j));
end
close all
for j = 1:30
    figure(j);
    plot(t4.ry1(j,:));
    ylim([25 35])
    %savefig(sprintf('CSI_ry1_Subcarrier%d.fig',j));
end
close all
for j = 1:30
    figure(j);
    plot(t4.ry2(j,:));
    ylim([10 30])
    %savefig(sprintf('CSI_ry2_Subcarrier%d.fig',j));
end
close all
for j = 1:30
    figure(j);
    plot(t4.ry3(j,:));
    ylim([23 33])
    %savefig(sprintf('CSI_ry3_Subcarrier%d.fig',j));
end
close all
%CSI_Analysis.m
%Developed by Andrew Lopez

%CSI_Plotter must be run before using this script

%Baseline Event
for i = 50:1700  %Must determine window for event!
    if(~isinf(ry2(1,i)))
        e1_sub1(i-49) = ry2(1,i);
    end
    if(~isinf(ry2(2,i)))
        e1_sub2(i-49) = ry2(2,i);
    end
    if(~isinf(ry2(3,i)))
        e1_sub3(i-49) = ry2(3,i);
    end
    if(~isinf(ry2(4,i)))
        e1_sub4(i-49) = ry2(4,i);
    end
    if(~isinf(ry2(5,i)))
        e1_sub5(i-49) = ry2(5,i);
    end
    if(~isinf(ry2(6,i)))
        e1_sub6(i-49) = ry2(6,i);
    end
    if(~isinf(ry2(7,i)))
        e1_sub7(i-49) = ry2(7,i);
    end
    if(~isinf(ry2(8,i)))
        e1_sub8(i-49) = ry2(8,i);
    end
    if(~isinf(ry2(9,i)))
        e1_sub9(i-49) = ry2(9,i);
    end
    if(~isinf(ry2(10,i)))
        e1_sub10(i-49) = ry2(10,i);
    end
    if(~isinf(ry2(11,i)))
        e1_sub11(i-49) = ry2(11,i);
    end
    if(~isinf(ry2(12,i)))
        e1_sub12(i-49) = ry2(12,i);
    end
    if(~isinf(ry2(13,i)))
        e1_sub13(i-49) = ry2(13,i);
    end
end
if(~isinf(ry2(14,i)))
    e1_sub14(i-49) = ry2(14,i);
end
if(~isinf(ry2(15,i)))
    e1_sub15(i-49) = ry2(15,i);
end
if(~isinf(ry2(16,i)))
    e1_sub16(i-49) = ry2(16,i);
end
if(~isinf(ry2(17,i)))
    e1_sub17(i-49) = ry2(17,i);
end
if(~isinf(ry2(18,i)))
    e1_sub18(i-49) = ry2(18,i);
end
if(~isinf(ry2(19,i)))
    e1_sub19(i-49) = ry2(19,i);
end
if(~isinf(ry2(20,i)))
    e1_sub20(i-49) = ry2(20,i);
end
if(~isinf(ry2(21,i)))
    e1_sub21(i-49) = ry2(21,i);
end
if(~isinf(ry2(22,i)))
    e1_sub22(i-49) = ry2(22,i);
end
if(~isinf(ry2(23,i)))
    e1_sub23(i-49) = ry2(23,i);
end
if(~isinf(ry2(24,i)))
    e1_sub24(i-49) = ry2(24,i);
end
if(~isinf(ry2(25,i)))
    e1_sub25(i-49) = ry2(25,i);
end
if(~isinf(ry2(26,i)))
    e1_sub26(i-49) = ry2(26,i);
end
if(~isinf(ry2(27,i)))
    e1_sub27(i-49) = ry2(27,i);
end
if(~isinf(ry2(28,i)))
    e1_sub28(i-49) = ry2(28,i);
end
if(~isinf(ry2(29,i)))

e1_sub29(i-49) = ry2(29,i);
end
if(~isinf(ry2(30,i)))
e1_sub30(i-49) = ry2(30,i);
end
end

e1_sub_mean(1) = mean(e1_sub1);
e1_sub_mean(2) = mean(e1_sub2);
e1_sub_mean(3) = mean(e1_sub3);
e1_sub_mean(4) = mean(e1_sub4);
e1_sub_mean(5) = mean(e1_sub5);
e1_sub_mean(6) = mean(e1_sub6);
e1_sub_mean(7) = mean(e1_sub7);
e1_sub_mean(8) = mean(e1_sub8);
e1_sub_mean(9) = mean(e1_sub9);
e1_sub_mean(10) = mean(e1_sub10);
e1_sub_mean(11) = mean(e1_sub11);
e1_sub_mean(12) = mean(e1_sub12);
e1_sub_mean(13) = mean(e1_sub13);
e1_sub_mean(14) = mean(e1_sub14);
e1_sub_mean(15) = mean(e1_sub15);
e1_sub_mean(16) = mean(e1_sub16);
e1_sub_mean(17) = mean(e1_sub17);
e1_sub_mean(18) = mean(e1_sub18);
e1_sub_mean(19) = mean(e1_sub19);
e1_sub_mean(20) = mean(e1_sub20);
e1_sub_mean(21) = mean(e1_sub21);
e1_sub_mean(22) = mean(e1_sub22);
e1_sub_mean(23) = mean(e1_sub23);
e1_sub_mean(24) = mean(e1_sub24);
e1_sub_mean(25) = mean(e1_sub25);
e1_sub_mean(26) = mean(e1_sub26);
e1_sub_mean(27) = mean(e1_sub27);
e1_sub_mean(28) = mean(e1_sub28);
e1_sub_mean(29) = mean(e1_sub29);
e1_sub_mean(30) = mean(e1_sub30);

%Reflector Event
start = 1849;
for i = 1900:3500 %Must determine window for event!
if(~isinf(ry2(1,i)))
e2_sub1(i-start) = ry2(1,i);
end
if(~isinf(ry2(2,i)))
e2_sub2(i-start) = ry2(2,i);
end
if(~isinf(ry2(3,i)))
e2_sub3(i-start) = ry2(3,i);
end
if(~isinf(ry2(4,i)))
e2_sub4(i-start) = ry2(4,i);
end
if(~isinf(ry2(5,i)))
e2_sub5(i-start) = ry2(5,i);
end
if(~isinf(ry2(6,i)))
e2_sub6(i-start) = ry2(6,i);
end
if(~isinf(ry2(7,i)))
e2_sub7(i-start) = ry2(7,i);
end
if(~isinf(ry2(8,i)))
e2_sub8(i-start) = ry2(8,i);
end
if(~isinf(ry2(9,i)))
e2_sub9(i-start) = ry2(9,i);
end
if(~isinf(ry2(10,i)))
e2_sub10(i-start) = ry2(10,i);
end
if(~isinf(ry2(11,i)))
e2_sub11(i-start) = ry2(11,i);
end
if(~isinf(ry2(12,i)))
e2_sub12(i-start) = ry2(12,i);
end
if(~isinf(ry2(13,i)))
e2_sub13(i-start) = ry2(13,i);
end
if(~isinf(ry2(14,i)))
e2_sub14(i-start) = ry2(14,i);
end
if(~isinf(ry2(15,i)))
e2_sub15(i-start) = ry2(15,i);
end
if(~isinf(ry2(16,i)))
e2_sub16(i-start) = ry2(16,i);
end
if(~isinf(ry2(17,i)))
e2_sub17(i-start) = ry2(17,i);
end
if(~isinf(ry2(18,i)))
e2_sub18(i-start) = ry2(18,i);
end
if(~isinf(ry2(19,i)))
e2_sub19(i-start) = ry2(19,i);
end
if(~isinf(ry2(20,i)))
e2_sub20(i-start) = ry2(20,i);
end
if(~isinf(ry2(21,i)))
e2_sub21(i-start) = ry2(21,i);
end
if(~isinf(ry2(22,i)))
e2_sub22(i-start) = ry2(22,i);
end
if(~isinf(ry2(23,i)))
e2_sub23(i-start) = ry2(23,i);
end
if(~isinf(ry2(24,i)))
e2_sub24(i-start) = ry2(24,i);
end
if(~isinf(ry2(25,i)))
e2_sub25(i-start) = ry2(25,i);
end
if(~isinf(ry2(26,i)))
e2_sub26(i-start) = ry2(26,i);
end
if(~isinf(ry2(27,i)))
e2_sub27(i-start) = ry2(27,i);
end
if(~isinf(ry2(28,i)))
e2_sub28(i-start) = ry2(28,i);
end
if(~isinf(ry2(29,i)))
e2_sub29(i-start) = ry2(29,i);
end
if(~isinf(ry2(30,i)))
e2_sub30(i-start) = ry2(30,i);
end
end

% combine mean of each subcarrier into a vector
e2_sub_mean(1) = mean(e2_sub1);
e2_sub_mean(2) = mean(e2_sub2);
e2_sub_mean(3) = mean(e2_sub3);
e2_sub_mean(4) = mean(e2_sub4);
\begin{verbatim}
e2_sub_mean(5) = mean(e2_sub5);
e2_sub_mean(6) = mean(e2_sub6);
e2_sub_mean(7) = mean(e2_sub7);
e2_sub_mean(8) = mean(e2_sub8);
e2_sub_mean(9) = mean(e2_sub9);
e2_sub_mean(10) = mean(e2_sub10);
e2_sub_mean(11) = mean(e2_sub11);
e2_sub_mean(12) = mean(e2_sub12);
e2_sub_mean(13) = mean(e2_sub13);
e2_sub_mean(14) = mean(e2_sub14);
e2_sub_mean(15) = mean(e2_sub15);
e2_sub_mean(16) = mean(e2_sub16);
e2_sub_mean(17) = mean(e2_sub17);
e2_sub_mean(18) = mean(e2_sub18);
e2_sub_mean(19) = mean(e2_sub19);
e2_sub_mean(20) = mean(e2_sub20);
e2_sub_mean(21) = mean(e2_sub21);
e2_sub_mean(22) = mean(e2_sub22);
e2_sub_mean(23) = mean(e2_sub23);
e2_sub_mean(24) = mean(e2_sub24);
e2_sub_mean(25) = mean(e2_sub25);
e2_sub_mean(26) = mean(e2_sub26);
e2_sub_mean(27) = mean(e2_sub27);
e2_sub_mean(28) = mean(e2_sub28);
e2_sub_mean(29) = mean(e2_sub29);
e2_sub_mean(30) = mean(e2_sub30);

%Event Detection Change (Baseline to Shape)
%Calculate difference between events
for i = 1:30
    diff_sub(i) = abs(e2_sub_mean(i) - e1_sub_mean(i));
end
\end{verbatim}