2019-12-01

Exploring the Efficiency of Software-Defined Radios in 3D Heat Mapping

Andrew Scott Thomas
Brigham Young University

Exploring the Efficiency of Software-Defined Radios in 3D Heat Mapping

Andrew Scott Thomas

A dissertation/thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Justin Giboney, Chair
Willie Harrison
Derek Hansen

School of Technology

Brigham Young University

ABSTRACT

Exploring the Efficiency of Software-Defined
Radios in 3D Heat Mapping

Andrew Scott Thomas
School of Technology, BYU
Master of Science

A common method of connecting to the internet is a wireless network. These networks can be monitored to discover the area of their coverage, but commercial receivers don't always provide the most accurate results. A software-defined radio was programmed to sniff wireless signals and tested against a commercial receiver and the results were compared. The results suggest that the software-defined radio performs at least as well as the commercial receiver in distance measurements and significantly better in samples taken per minute. It was determined that the software-defined radio is a viable replacement for a commercial receiver in 3D heat mapping.

Keywords: software-defined radio, WiFi, security, heat map

ACKNOWLEDGEMENTS

I would like to thank everyone who has helped me and pushed me to finish my research and thesis. I appreciate my parents for letting me use their house as a testing grounds for everything I did and for supporting me through the whole process. I would also like to thank Dale Rowe who chaired my committee until his departure from BYU and helped me discover this area of research. I also thank Justin Giboney, my committee chair, who encouraged me and helped me see this research through to the end.

I appreciate the help of all my friends and coworkers who kept me motivated and working toward the end goal. I'm grateful for everyone who reviewed my writing, helped me troubleshoot, and gave me the best possible chance of success.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Background

The first IEEE standard for wireless networks was released in 1997 and has continued to grow and be revised over the twenty years since its inception. The first iteration of that standard, the 802.11 wireless standard, was relatively slow compared to speeds we are able to achieve today, but it laid the foundation for a large shift from wired to wireless networks in many organizations (IEEE, 1997). The flexibility of such a network is beneficial for many reasons, but it is also a weakness that can be exploited.

Being able to access a wireless network outside of a building is one method of attack that a malicious party can leverage. If the network extends to the parking lot or the street, an attacker doesn't need physical access to the building to be able to attempt to compromise the network and subsequently the hosts on that network. Attack vectors introduced by this vulnerability include man-in-the-middle (MITM) attacks and denial-of-service (DOS) attacks (Elhamahmy & Sobh, 2011; Li, Koutsopoulos, & Poovendran, 2010). Being able to regulate and understand the perimeter of the wireless network is an important step in mitigating that risk (Issac, Jacob, & Mohammed, 2005).

A method of mapping the network accessibility and signal strength outside of a building in order to visualize the perimeter of the network is commonly used to defend against these attacks. This method of network analysis, called "heat mapping" or "war driving" provides

valuable information about a wireless network's strength outside of the building, and thus how far away an attacker would need to be to gain access. There have been various methods for doing war driving, including pushing a laptop on a cart and measuring the network strength or driving around the building in a car measuring network strength, but most only analyze the ground-level strength of the network (Hurley, Thornton, & Puchol, 2004).

Wireless networks are broadcast in three dimensions (3D), so if an attacker is unable to access the network near the ground, it may still be possible to find an access point above ground level through new technology such as a drone. A drone with the proper equipment would be able to land on a balcony or roof or simply hover around the building and relay the signal to the attacker on the ground. To prevent this, an organization needs to be able to create a 3D heat map that covers every surface of the building, not just the ground level.

The most effective method to create a 3D heat map employs a drone to circle a building and map the network at a given distance away (Pack, 2014). This is a promising method, but it can be improved. One improvement that can be made is the use of a software-defined radio (SDR) to implement the IEEE 802.11 standard instead of using a conventional WiFi receiver. This method can increase efficiency by being able to read packets across a network without connecting where a normal receiver would traditionally need to conduct a complete handshake or do a scan of available networks. To increase efficiency, the SDR can also be programmed to simply read partial packets to find the network identifier without wasting time on decoding the entire packet. By taking advantage of this technology, wireless heat mapping in three dimensions can be created at faster speeds and greater distances.

1.2 Research Objectives

Since access to a wireless network is one method for an attacker to access a system, organizations that host such networks need an efficient way to monitor the reach of their networks. With the recent advent of affordable drone technology, a scanning platform must be developed with a small enough form factor that it can be mounted on a drone to map not only signal strength on the ground-level, but also signal strength across every face of a building. Although technologies do exist that are able to create a 3D heat map of a building, improvement on the effectiveness of these technologies is possible to increase the capture speed and capture distance. This new design focuses on improving two main features: collection speed of the receiver and the distance at which samples can still be reliably collected.

These two aspects can be summarized by the following research question:

Research Question 1: What advantages does a software-defined radio have over a commercial WiFi receiver?

1.3 Hypotheses

The purpose of the experiments will be to test the following hypotheses:

- Hypothesis 1: An implementation of the IEEE 802.11 standard on an SDR will be more capable of interpreting WiFi signals at greater distances than a conventional WiFi receiver.
- Hypothesis 2: An implementation of the IEEE 802.11 standard on an SDR will collect samples at a greater speed than a conventional WiFi receiver.

1.4 Definitions

Access Point (AP) – The device from which a wireless signal originates.

Attacker – Any party, an individual or organization, that attempts to access a network with malicious intent.

Countermeasure Evaluation – A determination of how to mitigate risks.

Drone – An unmanned aerial vehicle whose movements are often autonomous. See also UAV.

Heat Map – A geographic visualization of density or change in factors such as population and weather. In the context of this study, it refers to WiFi signal level.

IEEE – Institute of Electrical and Electronics Engineers.

IEEE 802.11 Standard – The de facto standard for WiFi communication

Packet – A networking term used to describe the finite-length data being transferred over wired or wireless networks.

Red Team – A team of security professionals hired by an organization to examine weaknesses within their network. These professionals attempt to access the network in the same manner as an attacker.

Service Set Identifier (SSID) – The name of a WiFi network

Signal Level – Also known as signal strength. This refers to the strength of the signal above the "noise," which represents the point where the signal is too low to be read.

Site Survey – See Heat Map

Software-Defined Radio (SDR) – A programmable device capable of transmitting and receiving wireless signals.

UAV – Unmanned Aerial Vehicle. An aircraft that can operate without a pilot on board.

WiFi Hotspot – See Access Point

WiFi Receiver – A device that is able to read the signal originating from a WiFi hotspot.

Wireless Network (or WiFi) – A method of connecting a host to the internet that does not require a wired connection.

Wireless Local Area Network (WLAN) – A wireless network within a small area

1.5 Methodology

Data will be collected on a given wireless network using two methods: the first will use a commercial WiFi receiver, a Raspberry Pi 4, and the second will use an implementation of the IEEE 802.11 protocol on a Nuand BladeRF SDR. These were chosen due to their small form factor and availability. The SDR was chosen because it is one of a few that will reach the 2.4 GHz range. Both platforms will be subjected to tests that will examine two important attributes: collection speed (how many samples can be collected per second) and distance (how far from the receiver will the receiver be able to collect packets).

1.5.1 Assessments

The speed assessment will test the number of samples that can be collected every minute by each platform. This will be tested by setting the platform in a stationary position and

collecting samples over a set period of time. The average samples per minute will then be calculated from this data.

The distance measurement will simply measure the greatest distance at which the system is able to accurately capture WiFi packets. Accurate capture will be based on a noise floor calculated for each platform. Once the signal level decreases below this noise floor, the platform will be considered unable to capture packets.

A rubric of efficacy will be created by the researcher to evaluate the results of these experiments. The two platforms will be evaluated based on this rubric and a recommendation will be made for the most effective platform.

There will be two situations in which the speed and distance measurements will be taken; the first will be a controlled environment with the access point in a known location, and the second will be in a real-world situation.

1.5.2 Controlled Environment

The control constitutes a WiFi hotspot being placed in a known location in a field or other open area. This hotspot will send out a consistent signal that will be received by the two platforms. For the first test, each platform will be placed at known distances away from the AP with a line-of-sight view and several measurements will be taken. The average signal level at each distance will be taken and plotted. For the second test, the platforms will be placed near the AP and will collect samples for a set amount of time. The average samples per second will be measured from the samples taken.

1.5.3 Real-World Environment

The real-world application constitutes measuring the signal level of a WiFi network outside of a building. With permission from the correct administrators and officials, the speed and distance measurements were again evaluated and compared to the developed rubric. All heat maps and results were provided to the correct administrators in the building used with recommendations for which was deemed most accurate.

1.6 Assumptions and Delimitations

1.6.1 Assumptions

The assumptions made at the beginning of the experiment are set out as follows:

Assumption 1: The conventional receiver (raspberry pi) functions properly and consistently.

Assumption 2: The proposed platform (SDR receiver) functions in accordance with the IEEE 802.11 standards for WLAN networks.

Assumption 3: The wireless networks examined will offer a consistent output that will allow both platforms to be accurately compared, despite not being tested simultaneously.

1.6.2 Delimitations

There are certain delimitations that are outside the scope of this project.

1: Alternative Communication Channels. The proposed platform can certainly be modified to examine other channels of communication such as GSM and Bluetooth, but these are considered out of the scope of this project. Also out of scope are higher-frequency WiFi channels such as the 3.7 GHz and 5 GHz bands that are available. This is

due largely to hardware restrictions. While it would be interesting to examine these channels, this project focuses on the IEEE 802.11 standard operating at 2.4 GHz.

2: Penetration Testing. The proposed platform can be modified to allow sending WiFi packets as well as receiving them, but for this experiment only reception will be considered. Thus, the viability of this tool as a Red Team attack vector will not be considered.

3: Countermeasure Evaluation. The experiments run in this thesis will give important information to the entities being tested, but it will not suggest mitigation procedures for dealing with vulnerabilities. The purpose of this experiment will be to show those vulnerabilities and allow the separate entities to decide how to mitigate their risks.

2 LITERATURE REVIEW

2.1 The IEEE 802.11 Family

As wireless communications have grown, the Institute of Electrical and Electronics Engineers (IEEE) has developed standards for WiFi communications. These standards describe methods of interpreting electromagnetic signals so that data can be passed wirelessly and are grouped under the 802.11 family (IEEE, 1997). The initial launch of the 802.11 standard was in 1997 with what is now referred to as the legacy protocol, 802.11-1997. This protocol implemented Direct-Sequence Spread Spectrum (DSSS) and Frequency-Hopping Spread Spectrum (FHSS) techniques for modulation that would result in a throughput of about 1-2 Mbit/s at 2.4 GHz (IEEE, 1997). Two years later both the 802.11a and 802.11b protocols were introduced. The first, 802.11a, introduced both the 3.7 GHz licensed band and the 5 GHz band as well as Orthogonal Frequency-Division Multiplexing (OFDM), but these higher frequencies are more prone to fading and thus have a lower range than their 2.4 GHz counterpart.

For this reason the 802.11b standard (and subsequently the g and n standards) were widely accepted and implemented. The current standard in the 2.4 GHz band is 802.11n, which implements a Multi-Input Multi-Output OFDM (MIMO-OFDM) modulation for high speeds (reaching 150 Mbit/s) and very high throughput (IEEE, 2009). This is also rivaled by the 802.11ac standard that operates in the 5 GHz band reaching theoretical speeds of 866.7 Mbit/s

(IEEE, 2013). Over the past 20 years this standard has defined wireless communications and is a large influence on our lives.

While wireless networks are convenient and advantageous, they are also problematic. Since standard is wireless, the signal must be broadcasted. This allows anyone with the proper technology to access the raw signal. There are structures within the design of the 802.11 protocol that offer some security, and there are many other protocols that attempt to protect the digital data being transferred, but WiFi sniffing is an inherent vulnerability that will be present as long as the technology remains wireless.

2.2 Methods of Attacking Wireless Networks

Several authors have examined attack vectors for wireless networks. One such method is a jamming attack. In this type of attack, the malicious part attempts to disrupt as many communication channels as possible (Li et al., 2010). This attack is not meant to steal data but to disrupt normal traffic flows in an attempt to prevent users from using the network. This is a type of denial of service (DOS) attack that is relatively easy to carry out.

More advanced methods include eavesdropping on a network and cracking its WEP keys (Yuan, Matthews, Wright, Xu, & Yu, 2010). This attack vector involves discovering a network, sniffing packets being passed across the network, and using those packets to crack the WEP key. This attack allows all packets being passed to be decrypted. This attack allows the malicious party to steal information instead of simply preventing information from being passed.

A third method of attacking a wireless network is a Man in the Middle (MITM) attack (Yuan et al., 2010). The most common method to perform this attack is to poison the ARP cache on the access point. Poisoning implies tricking the access point into thinking the malicious

computer is a different (trusted) computer. The malicious computer can then intercept packets, modify them, and send them to their destination.

2.3 Securing Wireless Networks

Securing a wireless network is not a simple task. There are as many solutions as there are attack vectors, and there isn't a single solution that will mitigate all the risks. For example, a method of preventing a jamming attack is a monitoring tool that assesses energy limitations and performance specifications for nodes on the network (Li et al., 2010). This mitigation would not stop a MITM attack. For MITM attacks, static ARP entries within the network could be a solution (Yuan et al., 2010).

One method of assessing the threats to a wireless network is a process called war driving. Many studies show this as the primary source of information when beginning attacks on wireless networks (Issac et al., 2005; Li et al., 2010; Yuan et al., 2010). War driving involves driving a laptop (or similar reception platform) around in search of wireless networks. Once a network is discovered, an attack can be mounted in an attempt to compromise the network. A case study in Malaysia used this method to find and capture packets from a wide variety of wireless networks (Issac et al., 2005). This study found that many APs weren't using any sort of encryption, and even when encryption was being used, a lot of useful information was still being passed in plain text. The study offers 16 security measures for securing wireless networks. Number nine suggests physically limiting the reach of the wireless network. Site surveys aim at understanding the reach of the wireless network around a building and using this information to better limit the reach of that network.

2.4 3D Site Surveys

A site survey, or an analysis of a wireless network, is a way of understanding usage properties of the network as well as its range. These surveys can also be used to understand risks by mapping the area of coverage of a network outside a building as well as finding rogue access points that may offer a point of entry for wireless attacks. These surveys can be completed by various methods, but often they are a simple two-dimensional (2D) scan done by a laptop being pushed around the building on a cart (Hills & Schiegel, 2004). There have been studies that have also suggested methods for three-dimensional heat mapping using new drone technology that offer a more complete understanding of the network (Pack, 2014).

Many site surveys offer data limited to a 2D field (the sidewalk around a building, for example) or take too much time to be feasible (Pack, 2014). This means that an organization is unlikely to do extensive surveys and thus they will not be aware of any vulnerabilities that may exist due to this oversight. In order to make site surveys feasible, they need to be low-cost and accurate. The most effective method found has been the implementation of drone technologies to conduct a 3D heat mapping of the outside of a building (Pack, 2014).

For these 3D surveys to be effective, the receiver needs to be small enough to be carried by a drone but powerful enough to accurately detect and map the signal strength. The method used in Pack's research used a Raspberry Pi to achieve this goal, but an improvement on this method could use a software-defined radio to capture the WiFi signal. The small form factor of many SDRs as well as their flexibility in programming make them an ideal test platform.

2.5 SDR-Based 802.11 Wireless Communication Module

As the IEEE 802.11 standard has been developed, many applications have been built and tested, including the use of a Software Defined Radio (SDR) as a reception platform. In one such experiment, long-range communications were established using the SDR platform where the IEEE 802.11a/g standard was implemented (Guerra, Anand, & Knightly, 2014). This type of platform offers many advantages, including a dynamic and highly customizable platform ideal for a development situation. It also offers a flexible environment that allows the developer to adapt to many different situations. This may also be a downside as improper programming may lead to performance loss, and size constraints may restrict the extent to which the platform will be effective. It is also limited by the bandwidth allowed on the platform. Many SDRs have a very limited band in which they were designed to operate and may not cover both the 2.4 GHz and 5 GHz bands in which WiFi signals operate, such as the first generation of the Nuand BladeRF used in this research. This can be overcome either by choosing a band to examine, buying separate platforms for each band, or finding a newer platform that will cover all the bands in the required spectrum

2.6 Difficulties of Wireless Communications

There are several difficulties in the physical properties of electromagnetic signals that lead to difficulties in their reception and subsequent interpretation into logical information (as a string of meaningful bits). The first of these properties, attenuation, can destroy a signal completely if the signal-to-noise ratio gets too low, meaning the signal isn't distinguishable from the static naturally in the air. When a signal is seen, it can be interpreted, but the techniques used in interpretation require a certain input level in order to function properly (Rice, 2008). This problem is solved with automatic gain control (AGC) that reads the input level of a signal and

iteratively regulates the signal strength (Rosu, n.d.). An AGC loop generally consists of a variable gain amplifier (VGA) whose gain is established by a feedback loop with error correction (Whitlow, n.d.). This sort of feedback loop is designed to minimize the error, or the difference between the incoming signal level (after amplification) and the expected signal level. The second physical property that causes difficulty is the carrier phase error of the signal. Carrier phase error is a phenomenon in which an electromagnetic wave expands or contracts in time (meaning it is received at a slightly different frequency than it was transmitted at) (Rice, 2008). The problem that arises is that the carrier frequency upon arrival is not the frequency expected by the receiver. To overcome this, a phase-lock loop (PLL) can be implemented that tracks the incoming frequency of the carrier and changes the interpreter accordingly. Unfortunately the receiver is also unaware of the ideal time to sample a signal, so a PLL is usually integrated into a timing-synchronization loop that will extrapolate from the signal the ideal sample time (Rice, 2008). These two tools allow the receiver to accurately sample the incoming electromagnetic signal and pass the symbols to the interpreter where they will be translated into a series of bits.

This interpreter may be software-based or hardware-based. A software-based interpreter will be the basis of this research as it is more flexible and easier for development. Hardware-based interpreters, such as one built on an FPGA, offer more speed while functioning, but they require more time for development. Where software allows for quick debugging and testing, FPGA development consists of running simulations and loading the image onto the hardware. These tend to add an additional time cost to development, particularly if the image contains errors and the process must be repeated.

3 METHODOLOGY

A description of the preparation, instrumentation tests, and flight assessment plans are given. Flight plans are laid out and analysis methods are established.

3.1 Equipment Preparation

The wireless collection platforms (the raspberry pi and the SDR) will be tested in a stationary position to assess consistent reporting of results. These will then be mounted and tested in the context of the full drone system to examine if there is any interference within the system that would influence the results.

3.2 SDR Functionality Tests

Most of the equipment to be used is commercially available, thus it does not need extensive testing. In order to verify functionality, the SDR, which runs on a non-commercial WiFi receiver program, is tested according to the experiment that follows.

The WiFi receiver program on the SDR must be able to receive, decode, and interpret WiFi packets. The program was written according to the IEEE 802.11 standard, and thus was tested against that standard. A wireless access point (WAP) was set up that continuously sent packets. The SDR was then introduced and began capturing packets. When the SDR was able to

capture and interpret these packets accurately, it was considered adequate for further testing on the drone platform.

3.3 Speed Test

The speed test measured the speed at which each platform was able to capture packets, measured in packets captured per minute. This was done with a WiFi transmitter sending a constant signal at a known location. Each platform was placed a yard away from the transmitter and set to capture packets for an hour. The pi was tested for the first hour and the SDR for the second hour. An overall average of packets per minute was calculated at the end of each test. In addition to this average, each hour was divided into five minute intervals and an average was taken for each 5 minute interval. The overall averages were compared to determine which platform was able to collect more samples per minute. Each platform was also evaluated individually to determine the variance of the data. This was used to determine the consistency of a given platform.

3.4 Distance Test

The distance test measured the farthest distance at which each of the receivers provided accurate results. A WiFi transmitter was placed in a known location and each of the receivers collected results at 15 yard intervals away from the transmitter. The receivers remained within the line of sight of the transmitter to reduce interference. This test was measured over a period of 5 minutes at each interval and an average signal level for the time period was calculated. A receiver was deemed ineffective at a certain distance when the signal level fell below an acceptable threshold, or noise floor.

3.4.1 Noise Floor

The noise floor was determined as the signal level at which a receiver was able to connect to the AP. This measure does not consider the maximum distance that a platform can see the existence of a network (but not connect to it), only the distance at which a connection can occur. The measurement is conducted in this way due to the differences between the platforms. The pi, which was able to actively scan for networks, could see the existence of a network far outside of the range of being able to connect to and communicate with the network. The SDR was listening for actual communication on the network, so when it passed the range of being able to interpret those communications, samples were unable to be taken. The noise floor measurement is a method of more accurately comparing the two platforms.

3.5 Field Test

A flat plane (a parking lot) with no obstructions was used in this test. A WiFi transmitter was placed at the edge of the plane transmitting a constant signal and the speed test was conducted. The results of this test provided an understanding of the capabilities of the WiFi receiver platforms.

The distances outlined in the distance test were measured and marked. The distance test was then conducted and the results recorded. The results of this test provided an understanding of how well each platform responded at each of the distances outlined.

3.6 Site Test

The site test implemented both the speed and distance tests in a real-world environment. The tests were conducted on a live network and their results compared. The distance test did not

attempt to determine the maximum distance at which the signal could be read, but to show the use of each platform in the real world. Some anecdotal assertions can be made from the distance test, however no other conclusions were drawn.

3.7 Performance Criteria

The platforms were evaluated based on two different measurements: functional distance from the transmitter and average samples per minute. Functional distance refers to the distance at which the signal level of the platform is above the noise threshold. Average samples per minute refers to the number of samples that each platform can take in one minute.

3.8 Hardware Setup

To best simulate the environment of a drone, the platforms were integrated into an existing drone platform and tested with the drone equipment. No flight tests were conducted, but the drone power supply was used and the drone's equipment was powered on during the tests. Figure 1 shows the overall setup of the drone environment.

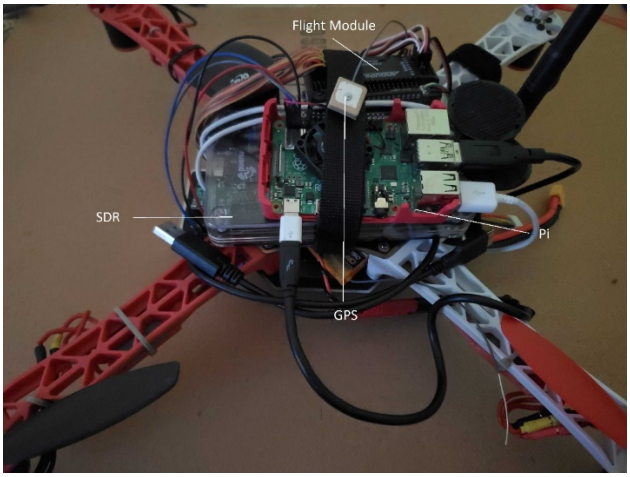

Figure 1: Drone Environment

4 IMPLEMENTATION

4.1 Equipment

The following equipment was used.

Table 1: Equipment Used

Type	Model
GPS	Goouuu Tech GT-U7 (NEO-6M)
Power Module	3DR C/I Power Monitor
Battery	Lectron Pro 5200mAh 3 cell Lithium Polymer
WiFi Receiver	Raspberry Pi 4 Model B
Software-Defined Radio	Nuand BladeRF (x40)
SDR Antenna	Nuand Tri-Band Antenna (5dBi)

4.1.1 Variables

Each of these platforms are somewhat modular and reliant on a number of variables for their outcome. Environmental variables as well as differences between hardware versions can cause a change in the results. As such, it was important to control as many of these variables as possible. Changes in environmental variables such as temperature were accounted for by testing each platform in the same area on the same day with as little time in between as possible. Simultaneous tests were not possible due to hardware restrictions. The hardware for each platform was also controlled to be as similar as possible. Both platforms used the same power

supply, GPS, barometer, and processor. The only difference in hardware was the antenna. These differences are discussed in section 4.1.2.

4.1.2 Antennas

The WiFi receiver contains an on-board antenna. In order to minimize form factor, this antenna was used instead of an external antenna and to provide a basic scenario of how a scan might be conducted (minimizing the number of external parts). For a similar reason, a generic 3-band antenna was used with the SDR. This is a generic "starter" antenna, meaning it is inexpensive and will cover a wide range of frequencies. The antennas used are obviously different between the two platforms, and it was expected that the SDR antenna would perform better than the pi antenna. There are several external antennas that can be purchased for the pi that would improve performance, however none were chosen for the purpose of this research. Instead, this research focuses on comparisons between the most basic (not optimized) models of each platform.

4.2 Initial SDR Development

To facilitate development of the SDR data collection script, the development was done on a virtual machine (VM) running Ubuntu 18.04 Desktop. Appendix A gives the setup of the virtual machine. The data collection script was based on the Gnu Radio development platform. The script was based largely off an example provided in the gr-ieee802-11 module, with some modifications to allow for the use of the BladeRF platform. Since this was a desktop environment, the GUI provided a simple development interface and allowed for visual representations of the data that wouldn't be available in headless mode. Once it was determined

that the module was working correctly, the automatically-generated python script was taken and modified to be used in headless mode.

4.3 Raspberry Pi Development

4.3.1 Raspbian Development

Due to its similarities to Ubuntu as well as being built for the Raspberry Pi, Raspbian was initially chosen as the operating system for the WiFi receiver. The same setup method used on the VM was attempted on this distro as well, but the packages needed (found in Appendix A) were not compatible with ARM processors. For this reason, Raspbian was discarded.

4.3.2 Ubuntu 18.04 for Raspberry Pi Development

In an attempt to create an environment more closely akin to the VM environment used to develop the data collection script, a version of Ubuntu 18.04 compatible with the Raspberry Pi 4 was found and installed in place of Raspbian. The setup used for the VM was again attempted and it was found that the packages would again not install due to the Pi's ARM processor. For this reason, the next attempt was based on the binary package of Gnu Radio available for install on the Pi. All other packages needed to be built from source and compatible with Gnu Radio 3.8, which is the packet manager's version. When this again encountered errors, Gnu Radio was uninstalled and the latest version (3.9) was built from source. All extra packages that depended on the version of Gnu Radio were also uninstalled and the Gnu Radio 3.9-compatible versions were built on the pi. Some code modifications were made to make it compatible with the ARM processor. This install was completed successfully and the python script developed on the VM was transferred to the Pi.

The GnuRadio program automatically generates a python script when a flowchart is created. This script was taken and modified to run without any of its GUI components. It was initially developed to output a constellation plot of the signal as well as an FFT plot of the raw signal. During development, the constellation plot gave a graphical representation of the symbols being accepted by the SDR. Similarly, the FFT plot showed the frequency band being received. These elements were removed from the final script as there was no need for any visual indicators after the script was verified to be working. Since the script was originally built on Gnu Radio 3.7, another small modification was also needed to convert to version 3.9. This involved converting the moving_average function implemented in version 3.7 of the gr-ieee802-11 module to the moving_average function that came native with Gnu Radio (the function was removed in gr-ieee802-11 3.9 in favor of the native Gnu Radio function). This modification allowed the script to run correctly as it had on the VM.

4.4 Data Collection Platform

The drone environment was used to determine if the SDR platform would perform appropriately while surrounded by the equipment on the drone. This platform included GPS and barometer modules along with flight controller and telemetry modules that could potentially cause interference. This environment was also used to show that the SDR was small enough to be mounted on a drone, and to determine if power issues would occur when the SDR was powered on using the drone power source.

4.5 Data Collection Scripts

There were two separate data collection scrips written: one for controlling the SDR and one for the conventional WiFi scanning method. Each script had a systemd service that could be

enabled so the script would run on boot. The service for the method being tested was enabled before the test began and stopped at the end of the test. Both scripts can be found in Appendix B

4.5.1 SDR Script

The script that controlled the SDR had two distinct parts: the collection of data from the SDR and the collection of data from the GPS. The script controlling the SDR was largely generated by Gnu Radio with modifications to remove the GUI components. The code consists of a class that connects signal processing blocks together in a manner that allows for the capture of WiFi signals and stores them in a pcap (network capture) file readable in Wireshark and similar programs. Due to the nature of the capture, these packets also contain a Radio Tap header which includes a measurement of signal strength. This measurement was used to determine the effectiveness of this platform. The name of this file was log[number]_wifi_[timestamp][operating frequency].pcap. The [number] section was used to avoid files with the same name. If a file already existed with a certain [number], it was incremented until the file name was unique. This check was implemented due to the pi's lack of an internal clock. When the pi was running without an active network connection, stopping and restarting the pi would cause clock skew that sometimes resulted in data points that seemed to be taken at the same time. This implementation allowed for multiple data points to be taken and given unique file names no matter the circumstance. The timestamp was in the format MM_DD_YY_12:34:56, varying according to the day and the time of day according to the time saved by the pi. The operating frequency was the frequency of the channel being scanned by the SDR, for example 2412000000 (or 2.412 GHz, WiFi channel 1). The SDR did not have scanning functionality, so a single channel was set at the beginning of the experiment based on the frequency of the WiFi being tested against.

The second part of the script controlled the collection of GPS data. Due to the nature of the program, the GPS data couldn't be introduced directly into the pcap file, so a new file was created to hold the GPS data. It was named exactly like the corresponding pcap file but with a .log extension instead. This allowed for direct correlation between the network data and the GPS data. Both files included timestamps of when the data was taken, so these timestamps were used to determine what network data corresponded to which GPS measurements. The GPS data was output with a timestamp, the latitude, the longitude, and the altitude.

The two parts worked on two different threads, so data from the SDR was being continuously collected in parallel with the GPS data being collected.

4.5.2 Pi Script

The script used on the Pi platform was designed to collect WiFi data directly from the Raspberry Pi's WiFi interface. This was done in a simple loop that scanned for all available access points (APs), parsed and formatted the information, acquired and formatted GPS and barometer information, and then wrote all the information to a log file in CSV format. This allowed for simple parsing of all the data in a single location and didn't require two separate files as in the SDR script. The filename for these logs was in the format log[number]_[timestamp].log. These values are used in the same manner as described in the SDR script. The notable difference is the lack of a frequency component. Since the pi can scan all APs on all frequencies, the frequency component of the filename was not necessary and was thus excluded.

This script was largely taken from Scott Pack's thesis research, but some modifications were made for this use case. In particular, the code was modified to allow for the geolocation data to not be logged when a flightless test was being conducted.

4.6 Analysis Scripts

There were three scripts developed to facilitate analysis of the data. The first script was developed to examine the raw pcap file created by the SDR and create a log file with a more useful format, including timestamps for the packets and relative signal strength. The second script was developed to divide the formatted data output by the first script into correct time divisions and take the average signal strength over the given division. Both scripts were used for the SDR. The third script was created to divide the data collected by the pi into time divisions and take the average signal strength for the given time division. Minor modifications to each of these scripts were made between the distance tests and the speed tests.

4.6.1 SDR Processing Script

To prepare the data captured by the SDR for analysis, a processing script was run on the capture (pcap) file. This script scraped the relevant information from the file, namely the SSID, timestamp, and signal strength of each packet, and then wrote that information to a new file: converted_data.log. Each line represented a different packet. This file was then able to be analyzed by the SDR analysis script.

4.6.2 SDR Analysis Script – Distance Test

A script was developed to take the average signal strength given by the SDR at a certain distance away from the transmitter. This script divided the packets by a pre-recorded time stamp

and added together all the signal strengths for each time period. A count was also kept of how many packets were processed within that time period in order to accurately represent the average. At the end of the script, a calculation was made for each distance represented by a certain time period and an average was given.

4.6.3 SDR Analysis Script – Speed Test

A script was also developed to determine the number of samples taken during the speed test. The SDR processing script was run on the network capture file to prepare for this script. The analysis script divided the pcap file into 5-minute intervals and counted the number of packets (or samples) were in each interval. This count was then divided by 5 to determine the number of samples per minute. The final time slot was disregarded as it was under 5 minutes. A total count was also kept and the total time was measured. The total count was divided by the total time to find an overall average of samples per minute across the whole hour.

4.6.4 Pi Analysis Script – Distance Test

The SDR analysis script for the distance test was modified to accept the log file output by the pi's data collection script. The log file was parsed according to time measurements, and average signal strengths were calculated for each time measurement as with the SDR analysis script. Since the pi collected information on multiple SSIDs, this script was modified to search for the correct SSID within the log file before processing the lines found within the log. A slight change in variable parsing and indexing was also necessary to correctly identify the signal level within the log.

4.6.5 Pi Analysis Script – Speed Test

This script was a modification of the SDR analysis script for the speed test. Since this was simply counting the number of samples taken within a time frame, the only modification needed was to search for the correct SSID in the line of the log file being imported.

5 RESULTS AND ANALYSIS

5.1 Rubric of Efficacy

In an attempt to compare the two platforms, the following criteria will be used to determine effectiveness of the platforms: functional distance and average samples per minute the platform can take.

5.1.1 Functional Distance

The functional distance is considered the maximum distance at which the platform can receive samples above the noise level, which is the distance at which each platform is able to receive packets. Each platform has its own measure of this due to the difference in measurement for the noise floor. Both noise measurements were taken in dBm, which is a mathematical comparison of the received signal power level compared to 1 mW. The SDR defines the noise floor as 0dBm, and any signals below this point were not interpreted. The pi does not specify the noise floor, but from the data collected, the noise floor can be seen at approximately -70dBm. The pi was able to see the AP at signal levels lower than this, however connections to the AP at this noise level or lower were unsuccessful. For the purposes of this research, these values will be used to determine the functional distance of the platform. The platform with the longest functional distance will be considered the most effective for this measurement.

5.1.2 Average Samples per Minute

The average samples per minute refers to how many measurements the platform can take in a minute without any constraints such as GPS measurements. This measurement attempts to find the maximum possible speed that measurements can be taken. A direct comparison can be made between the two platforms, and the platform with the highest average per minute is considered the most effective.

5.2 Field Test Results

The field tests were taken under a controlled environment with a WiFi router placed in a known location. Measurements taken were based on this single access point (AP).

5.2.1 Distance Test

The following results come from the flightless distance test. Initially the flightless distance test was attempted using a WiFi hotspot from a cell phone. Due to the weak signal level and poor results, this attempt was discarded. The second attempt used a WiFi router set up in a parking lot and with measurements being taken every 15 yards. This test ended at 150 yards. For each transition between 15-yard increments, the instruments were powered off and moved to create a separate log file for each distance. Instrumentation error caused these numbers to be unusable. The third attempt was done in the same environment as the second, but instead of powering off the instrumentation with each move, GPS measurements were taken to augment the physical measurement taken. This allowed the instruments to remain on during the transition between stages which reduced the possibility of error.

Measurements were taken at 15 yard increments for 5 minutes each. Average signal levels were taken at each stage as shown in table 2. Figures 2 and 3 show these numbers graphically compared to the noise floor.

Table 2: Distance Test Comparison – Field Test

Distance	SDR (dBm)	Pi (dBm)
5 yards (baseline)	0	-45
15 yards	9	-53
30 yards	16	-74
45 yards	8	-73
60 yards	0	-72
75 yards	0	-75
90 yards	0	-79
105 yards	0	-76
120 yards	0	-75
135 yards	0	-78
150 yards	0	-75

The data in table 2 shows that both platforms were able to see the signal with varying degrees of success. Notably the SDR very clearly either received a packet or it didn't. At longer distances the SDR was completely blind to the signal. On the other hand, the pi was able to see the hotspot at all distances, but the signal from 30 yards and onward was very faint. The pi did have clear signals at 5 and 15 yards.

Of interest are the first two tests for the SDR. The expectation was that the signal strength would be strongest at 5 and 15 yards and fall off after that. The lack of packets at 5 yards is most likely due to the positioning of the antenna with respect to the hotspot. The hotspot was located in an elevated location to offer better visibility at longer distances, but the SDR (and pi) were placed on the ground for each test. The antenna was situated vertically to allow better reception at longer distances, but at these short distances it was not ideal for reception.

The SDR also lost all reception after 45 yards while the pi reported being able to at least see the network being tested. Both platforms had reached their noise floor at that point, and it is likely that the SDR was not able to process the signal among the noise. More processing and likely upgraded equipment would be necessary to enable the SDR to extract information from below its noise floor.

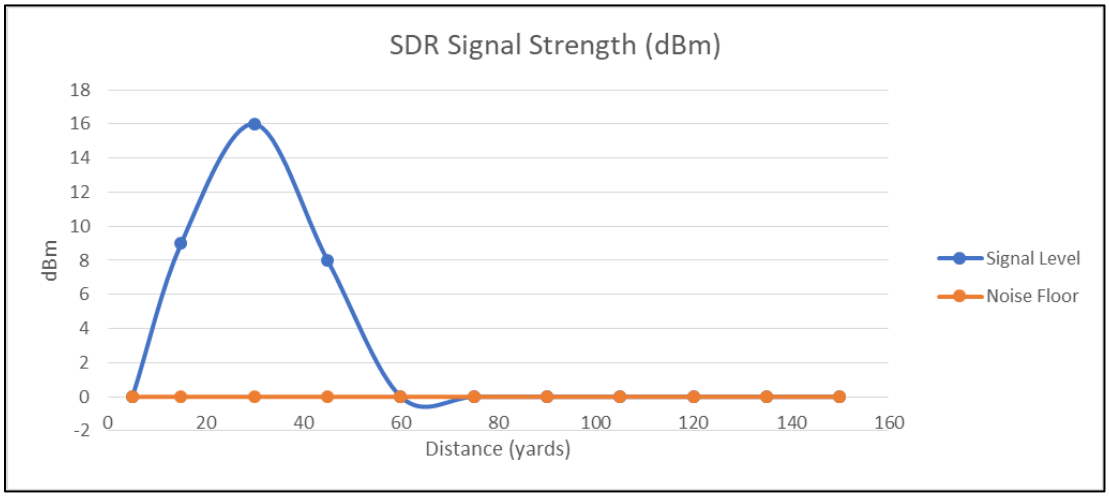

Figure 2: SDR Signal Strength - Field Test

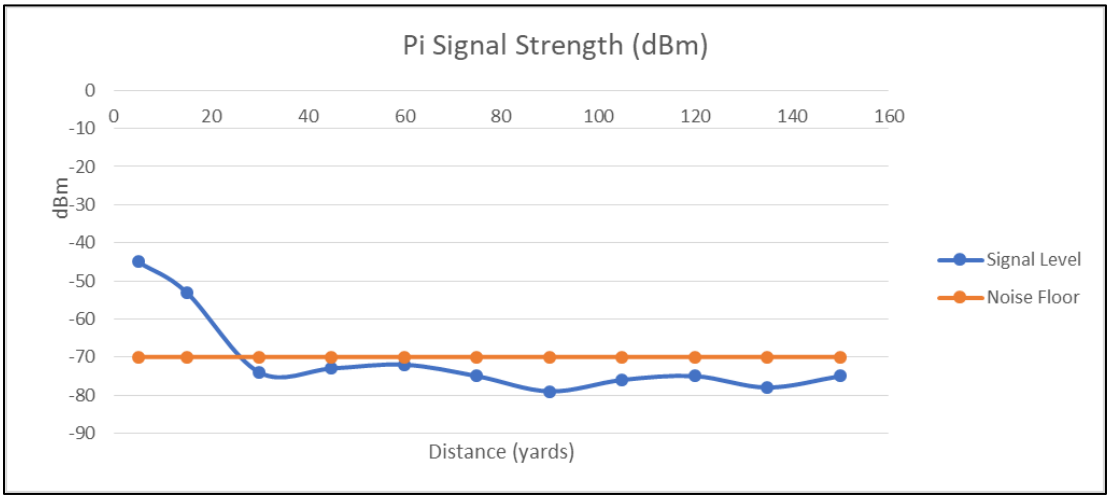

Figure 3: Pi Signal Strength - Field Test

The results of this test suggest that the SDR is no worse than the pi at detecting the distance at which the network is accessible. Since the pi was able to detect the signal at larger distances, even if it is unable to connect, it has an advantage in certain situations. However, when testing the limits of a connection or when being able to capture packets is required, the pi and the SDR have similar functional distances. Since the main reason for using an SDR is to test connection limits and perform packet capture, it is determined to be a viable platform.

5.2.2 Speed Test Results

The AP was set up on the same plane as the collection platform and was constantly sending and receiving data. Each collection platform was set up one yard away with GPS measurements turned off, and the script was set to collect data for an hour. The pi collected data for the first hour and the SDR collected data for the second hour. After each platform collected data, the hour was divided into 5-minute intervals, and an average number of samples per minute was taken for each 5-minute interval. An overall average was also taken for the full hour. Figures 4 and 5 show the results of this test for the SDR and the pi, respectively.

The overall average for the pi was 15.7 samples per minute, and the overall average for the SDR was 507 samples per minute. The SDR outperformed the pi by a factor of 32.3 according to raw averages. This is a significant difference between the two platforms, however the pi offered more consistency in its sampling. The variance of the pi for this test was 0.021 where the variance of the SDR was 507.5. This difference in variance is caused by the scanning method employed by each platform. The pi performs an active scan that will produce results as long as the AP is in range. The SDR requires network traffic to be captures as it flows between the AP and some endpoint. If the traffic spikes or lags, this will be reflected in the number of

samples per minute. For most active APs, the amount of traffic that can be detected by the SDR will outweigh the difference in variance, in which case the SDR will be the superior platform for speed, however the SDR will be inferior when testing inactive APs or APs that do not regularly see much traffic.

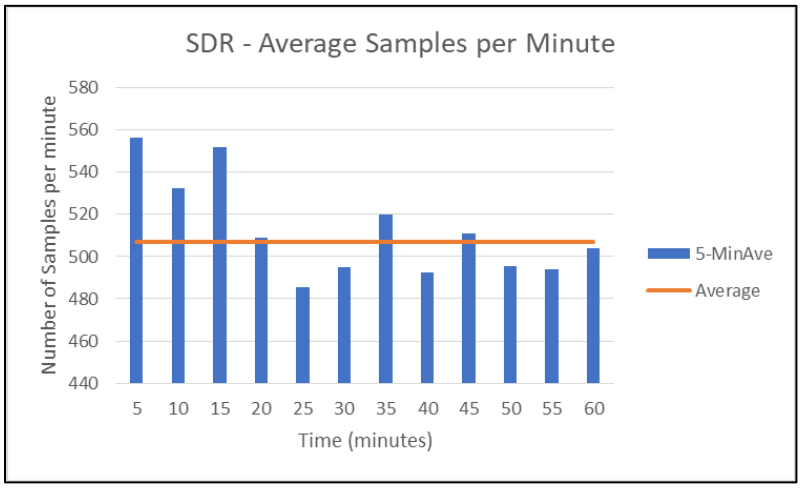

Figure 4: SDR Average Samples per Minute – Field Test

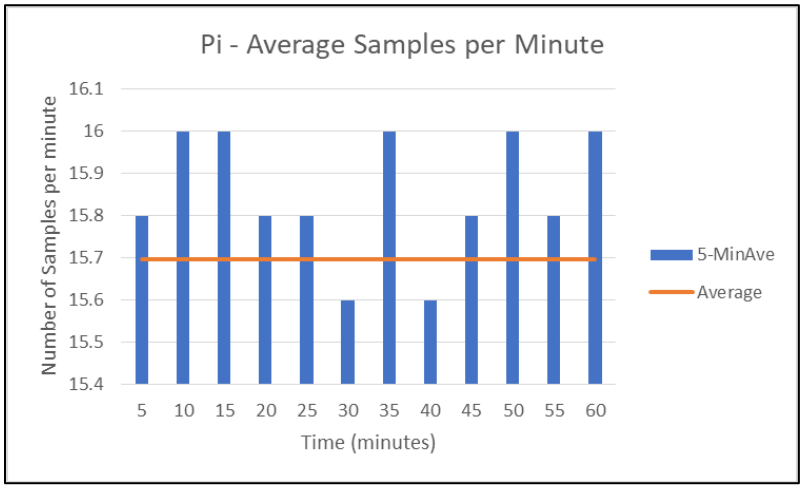

Figure 5: Pi Average Samples per Minute – Field Test

5.3 Site Test Results

The site test involved an active and utilized AP within a home. With the owner's permission, the distance and speed tests were again conducted. While there were several visible APs in the area, the tests were based solely on the owner's SSID.

5.3.1 Distance Test Results

The distances taken for this test were based on safe locations outside the house (namely sidewalks to avoid the road). The first distance was 11 yards away from the AP, the second distance was 19 yards, and the final distance was 36 yards. Table 2 shows the average signal levels for both the SDR and the Pi at each of these distances.

Table 3: Distance Test Comparison – Site Test

Distance (yards)	SDR (dBm)	Pi (dBm)
11	9	-60
19	8	-66
36	4	-59

Neither platform reached the noise floor in either experiment, but where the SDR had a predictably decreasing trend, the pi exhibited a flat behavior that suggests a minimal signal. It was confirmed that the pi was collecting data properly, and the evidence suggests that the pi was detecting a somewhat minimal signal. The behavior experienced here is similar to the behavior in the field test after 30 yards. The interference from walls and other obstacles would cause much quicker attenuation in the signal, and this is likely the reason for the flatness of the pi signal received. The SDR also saw effects of this attenuation. Compared to the field test, the signal was significantly attenuated, however the signal received by the SDR did more accurately show the

signal strength decreasing with greater distance. For this test, the SDR appears to perform a more accurate measurement of the expected signal strength of the network. This data is not sufficient to conclude that the SDR is better or worse than the pi, however it does show that the SDR functioned as expected in a real-world scenario.

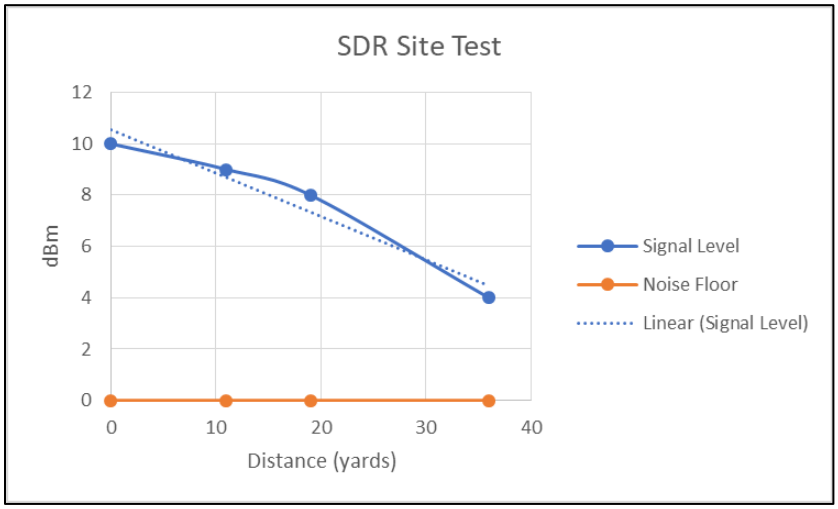

Figure 6: SDR Signal Strength – Site Test

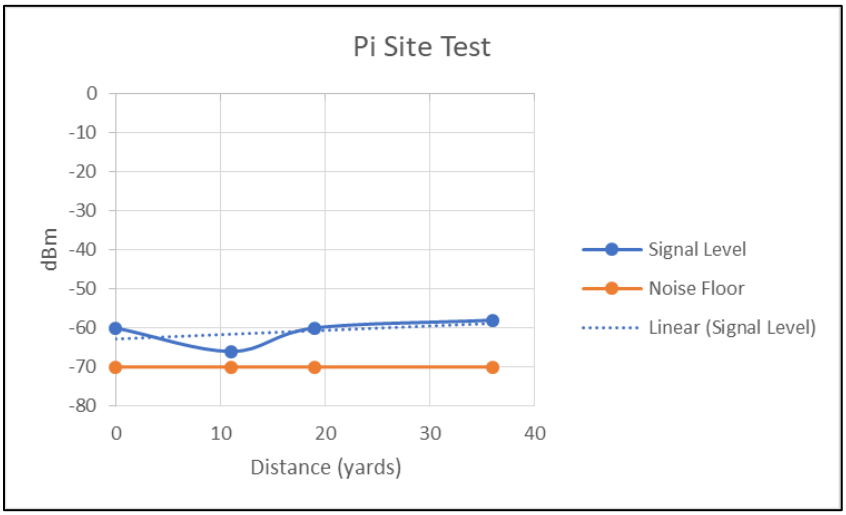

Figure 7: Pi Signal Strength – Site Test

5.3.2 Speed Test Results

The on-site speed test did not edit the collection script to remove the GPS timing. In this case, the pi slightly outperformed the SDR with an average of 12.5 samples per minute, shown in figure 9, compared to the SDR's 10.7 samples per minute, shown in figure 8. The pi slightly underperformed compared to the speed test conducted in the controlled environment by three samples per minute. This is to be expected. By comparison, the SDR only functioned at 2% efficiency compared to the controlled environment. This drastic decrease in efficiency is unexpected. A likely explanation for this is a lack of network activity during the test. The test was re-run at a time of higher network activity. This test showed an average of 55.6 samples per minute, as shown in figure 10. While this is still significantly lower than the field test, it does outperform the pi by a factor of 4.45. It also highlights a difference between the two collection platforms. The pi is a stable platform because it actively scans the APs available to extract the data necessary for the experiment. The SDR, being a passive sniffer, requires network traffic to be sent or broadcast from the transmitter. If there is a network available with no network traffic being passed, the SDR will be unable to detect that network's traffic. When traffic is available, the SDR continues to outperform the pi by a significant margin, despite the decrease in sampling caused by the GPS.

It is important to note that figures 8 and 10 show the exact same test being run at two different times. The data shown in figure 8 was collected at a time of very low network activity on the network being tested, while the data in figure 10 was collected at a time of higher network activity on that network.

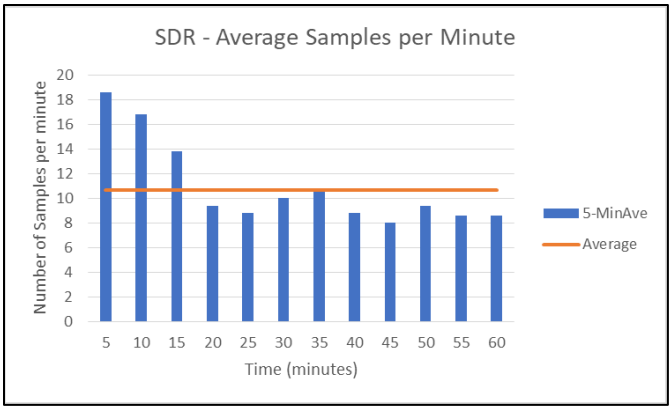

Figure 8: SDR Average Samples per Minute – Site Test

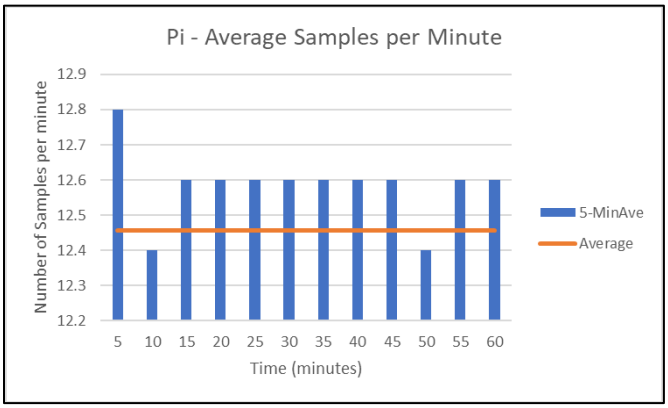

Figure 9: Pi Average Samples per Minute – Site Test

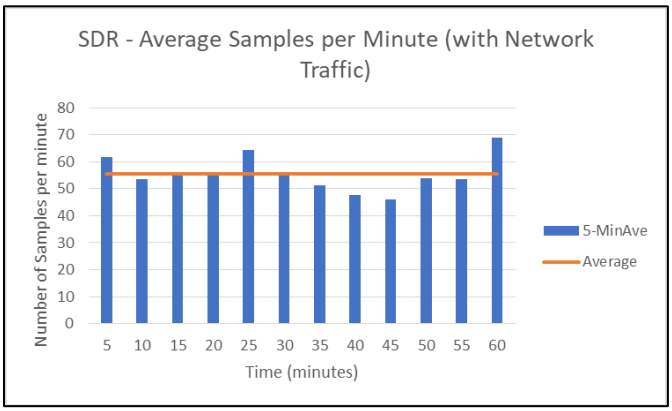

Figure 10: SDR Average Samples per Minute (Network Traffic) – Site Test

5.4 Analysis of Research Question 1

Research Question 1: What advantages does a software-defined radio have over a commercial WiFi receiver?

The results of the speed test taken show that the SDR outperforms the pi by a factor or 32 in a controlled environment. The SDR is able to outperform the pi to this level for one major reason: the pi is designed as an active scanner while the SDR is designed to passively scan the frequency. The pi requires an active scan of all frequencies for all available access points within range. This requires time and resources that greatly limit the number of measurements that can be taken in a minute. The SDR is able to passively pick up packets being sent on a certain channel or frequency. This means that this iteration does not scan all frequencies, but it is able to listen to a single frequency channel and capture samples at a much higher rate than the pi. It was also shown that this can be a weakness of the SDR. When network activity is low, the SDR will potentially collect less data than the pi. However, in most scenarios, the SDR will outperform the pi in speed.

The distance scenario was less conclusive. The pi and the SDR performed similarly at similar distances, but there were clear advantages and weaknesses in the two platforms. The pi was able to see, but not connect to, the AP at longer distances. It did pass the noise floor at nearly the same time as the SDR passed its noise floor, meaning their functional distances were similar. For this reason, the advantages of the SDR are situational. It performs similar to or better than the pi in the cases it was designed for, but the pi performs better than the SDR outside of those cases.

5.5 Analysis of Hypothesis 1

Hypothesis 1: An implementation of the IEEE 802.11 standard on an SDR will be more capable of interpreting WiFi signals at greater distances than a conventional WiFi receiver.

The distance test showed that the functional distance of the SDR was about 15 yards greater than the functional distance of the pi. This is a fairly insignificant difference, and the only conclusion that can be reasonably drawn is that the SDR is no worse than the pi in functional distance. This does concede, however, that the SDR is a viable platform for this type of test.

5.6 Analysis of Hypothesis 2

An implementation of the IEEE 802.11 standard on an SDR will collect samples at a greater speed than a conventional WiFi receiver.

Given that the SDR was superior to the pi in both speed measurements, this hypothesis is true. The caveat for this assessment is the situation in which network traffic is limited. Since most organizations performing assessments will be working with active networks during business hours, this caveat is noted but deemed inconsequential.

5.7 Further Observations

Observations not strictly relating to the original research question and hypotheses are considered and assessed.

5.7.1 Complexity

The SDR as a platform is superior to the pi in its flexibility and potential, however it requires more effort on the part of the user. It requires an understanding of signal processing that

most users do not have. While it would require a lot of time and expertise to develop an SDR platform such as the one used in this research, the benefits of the platform outweigh the costs. The code provided in Appendix B is a good starting point for this development, but the code provided is still in a basic form that required further development for more advanced cases.

5.7.2 Unknown Networks

While developing the SDR platform, previously invisible networks appeared on the test scans. Much of the development took place in two locations (work and home). One location (the work network) had no known wireless networks on the 2.4 GHz range, and the other location (the home network) had a 2.4 GHz network on a known channel. Testing on the home network provided expected results, however when scanning the work network, two previously unknown wireless networks appeared. Both networks found had counterparts in the 5 GHz band, however these were in the 2.4 GHz band and did not appear to be accessible by a normal computer. Much of the information gathered about this network seemed to point to it being a hidden network used for printers and other similar devices. Further investigation is necessary to determine the exact nature of these networks and their accessibility.

5.7.3 Weaknesses of the SDR

During testing of the SDR platform, many strengths were found as well as some weaknesses. The most apparent weakness in the SDR platform is its use as a passive sniffer. Where the pi is able to do active scanning in searching for a network, the SDR is currently unable to do so. For this reason, if a network is not active (if devices are not sending packets), the SDR will not see the network. Modification to the code could allow the SDR to do active reconnaissance, however that was beyond the scope of the research.

5.7.4 Using the SDR in the Drone Environment

While not a primary focus of this research, the testing showed that the SDR was able to run while mounted on the drone. This was an expected outcome, but further validates the assertion that the SDR is a viable platform for 3D heat mapping.

5.7.5 Summary of Advantages and Disadvantages

Table 4 describes the advantages and disadvantages of each platform for certain characteristics and conditions. This is not an exhaustive list, but it is a summary of some of the important findings of this research.

Table 4: Advantages and Disadvantages of the Platforms

	SDR		Pi	
	Advantage	Disadvantage	Advantage	Disadvantage
Scanning	Passive - faster and able to sniff traffic	Passive - if there is no traffic, the sniffer will not detect a network	Active - can see a network even when there is no traffic	Active - slower, only attempts to see the network, doesn't sniff traffic
Weight		Requires the pi for processing, heavier	Does not require external modules (lightest possible version)	
Data	Data output in pcap format, viewable in Wireshark	Quick parsing of data is more difficult, if a packet is malformed the data may not be viewable in wireshark	Human-readable data output in a log file	Must understand the order of the data to read (data is unlabeled)
Speed	Very fast (number of samples is the number of packets sniffed)	If there are no packets, it doesn't function	Stable, speeds don't change much between tests	Much slower than the SDR
Distance	Functions at similar distances to the Pi	No samples collected at longer distances when packets drop below the noise floor	Can scan and see WiFi networks at longer distances, connections can be established at similar distances to the SDR	May still see an AP at long distances but be unable to connect (potential false positive)

6 CONCLUSIONS AND FUTURE RESEARCH

6.1 Conclusion

The research suggests that the SDR platform is a viable platform for mapping WiFi signal strength. Difficulties in development and lack of knowledge base inhibit the number of users able to implement such a platform, however those who are able to build a similar platform can find improved results over conventional systems. These improvements largely relate to the speed of collection seen on the SDR platform. This increased collection speed could translate into more time-efficient creation of heat maps. Combined with an equally time-efficient geolocation method, a heat map could potentially be created four times faster than with previous methods, assuming the findings from the site test hold true.

Due to the passive nature of the SDR, an active wireless network is necessary for testing. This is the biggest advantage of the SDR as well as its biggest disadvantage. The passive nature is the reason for the increase in speed, but it also limits the situations in which it can function. This disadvantage can be avoided by conducting assessments with this platform during normal business hours using a network that is known to be active.

6.2 Future Work

The following items can be investigated to improve the system or expand on the work found in this research.

Expanded Spectrum Surveys: While this survey only covered the 2.4 GHz WiFi spectrum, future surveys could expand to the 5 GHz band, Bluetooth, or GSM. The SDR could be easily reprogrammed to accept the 5 GHz band, and modules exist for capturing Bluetooth and other data.

Penetration Testing Assessment: The principles of cybersecurity call researchers to develop better tools under the assumption that malicious parties are also developing better strategies to compromise people and systems. This research has shown that the SDR platform can be more effective than other tools in analyzing wireless networks. It was developed as a reception platform, but some modifications could allow for its use as a vulnerability assessment and penetration testing tool. This could involve more advanced sniffing, Man-in-the-Middle attacks, and network spoofing.

Countermeasure Evaluation: The assessment in this research could easily be used to develop countermeasure evaluations within an organization, including how to best reduce WiFi leakage and how to detect attacks against their network.

Specialized Equipment: This research showed that the SDR platform is viable in the use case. Further improvements can be made including the use of a directional antenna to survey at a greater distance, FPGA-based signal processing for speed, and other specialized modifications aimed at improving performance.

FPGA Programming: The SDR receiver was implemented mainly in software and required a significant amount of processing power. If the majority of the processing can be done within an FPGA array such as the one available on the bladeRF SDR used, the speed and accuracy of the data collected could be greatly increased.

Active Network Tracking: The SDR is a passive device and thus can only sniff available traffic. To make the type of scanning outlined in this thesis more practical and be able to detect live but inactive networks, an active scanning module could be added.

Network Monitoring: The SDR receiver can be modified to track the activity on a network. This could simply measure the number of packets being sent across the network, or it could be used to look at different types of packets being sent and their contents.

REFERENCES

Elhamahmy, M., & Sobh, T. (2011). Preventing Information Leakage Caused by War Driving Attacks in Wi-Fi Networks. *International Conference on Aerospace Sciences and Aviation Technology*, *14*(AEROSPACE SCIENCES), 1–9. https://doi.org/10.21608/asat.2011.23405

Guerra, R. E., Anand, N., & Knightly, E. W. (2014). *Demo: An Open-Source Develpment Platform for Long-Range UHF-Connected WiFi Hotspots*. 275–278. MobiCom.

Hills, A., & Schiegel, J. (2004). Rollabout: A Wireless Design Tool. *IEEE Communications Magazine*, 132–138.

Hurley, C., Thornton, F., & Puchol, M. (2004). *WarDriving: Drive, Detect, Defend - A Guide to Wireless Security* (R. Rogers, Ed.). Syngress Publishing, Inc.

IEEE. (1997). Telecommunications and Information Exchange Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE-SA Standards Board*.

IEEE. (2009). Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE-SA Standards Board*.

IEEE. (2013). Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE-SA Standards Board*.

Issac, B., Jacob, S. M., & Mohammed, L. A. (2005). The Art of War Driving and Security Threats - A Malaysian Case Study. *2005 13th IEEE International Conference on Networks Jointly Held with the 2005 7th IEEE Malaysia International Conference on Communications, Proceedings*, *1*, 124–129. https://doi.org/10.1109/ICON.2005.1635452

Li, M., Koutsopoulos, I., & Poovendran, R. (2010). Optimal Jamming Attack Strategies and Network Defense Policies in Wireless Sensor Networks. *IEEE Transactions on Mobile Computing*, *9*(8), 1119–1133. https://doi.org/10.1109/TMC.2010.75

Pack, S. J. (2014). *Multi-Rotor—Aided Three-Dimensional 802.11 Wireless Heat Mapping*. Retrieved from http://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=5014&context=etd

Rice, M. (2008). *Digital Communications: A Discrete-Time Approach*.

Rosu, I. (accessed 2019). Automatic Gain Control (AGC) in Receivers. Retrieved from http://www.qsl.net/va3iul/Files/Automatic_Gain_Control.pdf

Whitlow, D. (accessed 2019). Design and Operation of Automatic Gain Control Loops for Receivers in Modern Communications Systems. Retrieved from http://www.analog.com/media/en/trainingseminars/tutorials/42575412022953450461111812375Design_and_Operation_of_AGC_Loops.pdf

Yuan, X., Matthews, D., Wright, O., Xu, J., & Yu, H. (2010). Laboratory Exercises for Wireless Network Attacks and Defenses. *Proc. of the 14th Colloquium for Information Systems Security Education. Baltimore, Maryland, USA*, (336).

APPENDIX A. VM AND RASPBERRY PI SETUP

VM Setup:

Anything with a gray background is a command

```
Install VM OS and update to latest version (installed Ubuntu Desktop 16.04,
upgraded to 18.04)


pip install --upgrade git+https://github.com/gnuradio/pybombs.git
pybombs auto-config
pybombs recipes add-defaults
mkdir ~/pybombs/
pybombs prefix init ~/pybombs/bladeRF -a bladeRF -R gnuradio-default
(if getting uhd errors: sudo apt install python-requests python-mako python-
setuptools)
pybombs -p bladeRF install bladeRF gr-iqbal gr-osmosdr gqrx

Edit normal user (sdr) and root user .bashrc file to be able to run GRC:

#!/bin/bash

# Add GNU Radio binaries to the search path
GNURADIO_PATH=/home/sdr/pybombs/bladeRF # THIS DEPENDS ON WHERE PYBOMBS IS
INSTALLED
export PATH=$PATH:$GNURADIO_PATH/bin

# Add GNU Radio python libraries to python search path
if [ $PYTHONPATH ]; then
        export PYTHONPATH=$PYTHONPATH:$GNURADIO_PATH/lib/python2.7/dist-
packages
else
        export PYTHONPATH=$GNURADIO_PATH/lib/python2.7/dist-packages
fi

Add a file: /etc/ld.so.conf.d/gnuradio.conf
In that file:

/home/sdr/pybombs/bladeRF/lib

That should just be a single line in the file

Run:
sudo ldconfig -v | grep gnuradio
```

```
source ~/.bashrc
```
First one checks that things were set up properly, second one updates the path

Next step is to update udev to be able to run the SDR
https://github.com/Nuand/bladeRF/tree/master/host/misc/udev
Add those three files into /etc/udev/rules.d (change @BLADERF_GROUP@ to plugdev)
make sure they're all 644 for permissions
run: `sudo udevadm control --reload-rules && sudo udevadm trigger` (unplug and replug the SDR)

For Constellation (and other things that require OpenGL):

```
sudo pip install pyopengl
```

Installing the WiFi-specific libraries:
```
pybombs install gr-foo
pybombs install gr-ieee802-11
```

In .bashrc (for root) add the following line at the end:
export QT_X11_NO_MITSHM=1

TROUBLESHOOTING:
If it doesn't register the bladeRF, try changing the VM USB port driver to 3 instead of 2

Resources:
https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux
http://pyopengl.sourceforge.net/documentation/installation.html
http://www.codebind.com/linux-tutorials/install-opengl-ubuntu-linux/

Raspberry Pi Setup:

Install Ubuntu 18.04 on an SD card (image and setup instructions available at https://jamesachambers.com/raspberry-pi-4-ubuntu-server-desktop-18-04-3-image-unofficial/)

On the image, GnuRadio v3.9 was built from source.

The following packages were also built from source to fit GnuRadio v3.9: osmocom, gr-foo, and gr-ieee802_11. Version 3.9 of these packages was not the maintainers' stable version, but it was necessary to ensure it built properly.

The GPS used was USB-capable, so no extra configuration was needed to ensure compatibility.

The barometer required the Adafruit BMP280 library. This can be easily downloaded, but it must be added to the $PYTHONPATH variable. If it is not added to $PYTHONPATH, all development must happen within the same directory.

The data collection scripts were transferred to the pi and functionality was tested before testing began.

APPENDIX B. DATA COLLECTION SCRIPTS

SDR Data Collection Script:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
# GNU Radio Python Flow Graph
# Title: Wifi Rx
# GNU Radio version: 3.7.13.4
##################################################

from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import fft
from gnuradio import filter
from gnuradio import gr
from gnuradio.eng_option import eng_option
from gnuradio.fft import window
from gnuradio.filter import firdes
from gnuradio.qtgui import Range, RangeWidget
from optparse import OptionParser
import foo
import ieee802_11
import osmosdr
import sys
import time
```

```
from datetime import datetime
import adafruit_bmp280
import board, busio
from pynmea import nmea
import os
import serial


print("\033[1;32;40m[+]\033[1;37;40m Giving the GPS time to
acquire...\033[0;39;40m")
time.sleep(6)
print("\033[1;32;40m[+]\033[1;37;40m GPS acquisition time
elapsed\033[0;39;40m")


now = datetime.now()
flghtless = True


# Set up the barometer (if attached)
try:
  i2c = busio.I2C(board.SCL, board.SDA)
  sensor = adafruit_bmp280.Adafruit_BMP280_I2C(i2c)
  sensor.sea_level_pressure = 1013.25
  start_alt = sensor.altitude
  barometer = True
except Exception as e:
  print("\033[1;31;40m[-] No barometer found:\033[0;39;40m")
  print("\033[1;31;40m[-]\033[1;95;40m " + str(e) + "\033[0;39;40m")
  barometer = False


# Find the correct filename
g_datetime = now.strftime("%m_%d_%Y_%H%M%S_")
g_freq = 2417000000
log_path = '/var/log/geolocate/test/'
filenum = 0
```

```
fileset = False
while not fileset:
  filenum = filenum + 1
  log_name = "log" + str(filenum) + "_wifi_%s%d.pcap"%(g_datetime,g_freq)
  gps_log_name = "log" + str(filenum) + "_wifi_%s%d.log"%(g_datetime,g_freq)
  if not os.path.isfile(log_path + log_name):
    fileset = True

f = open(log_path + gps_log_name,'w+')
f.write("\n")
f.flush()
os.fsync(f)

# Find the correct serial port for the GPS
filenum = 0
fileset = False
ACM_path = '/dev/ttyACM'
while not fileset:
  if not os.path.exists(ACM_path + str(filenum)):
    filenum += 1
  else:
    ACM_path = ACM_path + str(filenum)
    fileset = True
  if filenum > 5:
    break

# Set up the GPS object
try:
  gps = serial.Serial(ACM_path,9600,timeout=1)
  gpgga = nmea.GPGGA()
  gps_conn = True
except Exception as e:
```

```
  print("\033[1;31;40m[-]\033[1;37;40m No GPS found:\033[0;39;40m")
  print("\033[1;31;40m[-]\033[1;95;40m " + str(e) + "\033[0;39;40m")
  gps_conn = False

class wifi_rx(gr.top_block):

    def __init__(self):
        gr.top_block.__init__(self, "Wifi Rx")

        ##################################################
        # Variables
        ##################################################
        self.window_size = window_size = 48
        self.sync_length = sync_length = 320
        self.samp_rate = samp_rate = 20e6
        self.lo_offset = lo_offset = 0
        self.gain = gain = 0.75
        self.freq = freq = g_freq
        self.chan_est = chan_est = 0
        self.datetime = g_datetime #now.strftime("%m_%d_%Y_%H%M%S_")
        self.log_path = log_path
        self.log_name = log_name

        ##################################################
        # Blocks
        ##################################################
        self.ieee802_11_sync_short_0 = ieee802_11.sync_short(0.56, 2, False,
False)
        self.ieee802_11_sync_long_0 = ieee802_11.sync_long(sync_length,
False, False)
        self.ieee802_11_parse_mac_0 = ieee802_11.parse_mac(False, True)
        self.ieee802_11_moving_average_xx_1 =
blocks.moving_average_ff(window_size + 16,1,4000)
```

```
        self.ieee802_11_moving_average_xx_0 =
blocks.moving_average_cc(window_size,1,4000)

        self.ieee802_11_frame_equalizer_0 =
ieee802_11.frame_equalizer(chan_est, freq, samp_rate, False, False)

        self.ieee802_11_decode_mac_0 = ieee802_11.decode_mac(False, False)

        self.foo_wireshark_connector_0 = foo.wireshark_connector(127, False)

        self.fft_vxx_0 = fft.fft_vcc(64, True, (window.rectangular(64)),
True, 1)

        self.dc_blocker_xx_0 = filter.dc_blocker_cc(32, True)

        self.blocks_stream_to_vector_0 =
blocks.stream_to_vector(gr.sizeof_gr_complex*1, 64)

        self.blocks_multiply_xx_0 = blocks.multiply_vcc(1)

        self.blocks_file_sink_0 = blocks.file_sink(gr.sizeof_char*1,
self.log_path + self.log_name, True) # '/var/log/geolocate/field-distance-
2/wifi_%s%d.pcap'%(self.datetime,self.freq), True)

        self.blocks_file_sink_0.set_unbuffered(True)

        self.blocks_divide_xx_0 = blocks.divide_ff(1)

        self.blocks_delay_0_0 = blocks.delay(gr.sizeof_gr_complex*1, 16)

        self.blocks_delay_0 = blocks.delay(gr.sizeof_gr_complex*1,
sync_length)

        self.blocks_conjugate_cc_0 = blocks.conjugate_cc()

        self.blocks_complex_to_mag_squared_0 =
blocks.complex_to_mag_squared(1)

        self.blocks_complex_to_mag_0 = blocks.complex_to_mag(1)

        self.bladeRF_Source = osmosdr.source( args="numchan=" + str(1) + " "
+ "fpga='/home/ubuntu/hostedx40-latest.rbf'" )

        self.bladeRF_Source.set_sample_rate(samp_rate)

        self.bladeRF_Source.set_center_freq(freq, 0)

        self.bladeRF_Source.set_freq_corr(0, 0)

        self.bladeRF_Source.set_dc_offset_mode(2, 0)

        self.bladeRF_Source.set_iq_balance_mode(0, 0)

        self.bladeRF_Source.set_gain_mode(False, 0)

        self.bladeRF_Source.set_gain(10, 0)

        self.bladeRF_Source.set_if_gain(20, 0)

        self.bladeRF_Source.set_bb_gain(20, 0)

        self.bladeRF_Source.set_antenna('', 0)
```

```
        self.bladeRF_Source.set_bandwidth(samp_rate, 0)


        ##################################################

        # Connections

        ##################################################

        self.msg_connect((self.ieee802_11_decode_mac_0, 'out'),
(self.foo_wireshark_connector_0, 'in'))

        self.msg_connect((self.ieee802_11_decode_mac_0, 'out'),
(self.ieee802_11_parse_mac_0, 'in'))

        self.connect((self.bladeRF_Source, 0), (self.dc_blocker_xx_0, 0))

        self.connect((self.blocks_complex_to_mag_0, 0),
(self.blocks_divide_xx_0, 0))

        self.connect((self.blocks_complex_to_mag_squared_0, 0),
(self.ieee802_11_moving_average_xx_1, 0))

        self.connect((self.blocks_conjugate_cc_0, 0),
(self.blocks_multiply_xx_0, 1))

        self.connect((self.blocks_delay_0, 0), (self.ieee802_11_sync_long_0,
1))

        self.connect((self.blocks_delay_0_0, 0), (self.blocks_conjugate_cc_0,
0))

        self.connect((self.blocks_delay_0_0, 0),
(self.ieee802_11_sync_short_0, 0))

        self.connect((self.blocks_divide_xx_0, 0),
(self.ieee802_11_sync_short_0, 2))

        self.connect((self.blocks_multiply_xx_0, 0),
(self.ieee802_11_moving_average_xx_0, 0))

        self.connect((self.blocks_stream_to_vector_0, 0), (self.fft_vxx_0,
0))

        self.connect((self.dc_blocker_xx_0, 0),
(self.blocks_complex_to_mag_squared_0, 0))

        self.connect((self.dc_blocker_xx_0, 0), (self.blocks_delay_0_0, 0))

        self.connect((self.dc_blocker_xx_0, 0), (self.blocks_multiply_xx_0,
0))

        self.connect((self.fft_vxx_0, 0), (self.ieee802_11_frame_equalizer_0,
0))

        self.connect((self.foo_wireshark_connector_0, 0),
(self.blocks_file_sink_0, 0))
```

```
        self.connect((self.ieee802_11_frame_equalizer_0, 0),
(self.ieee802_11_decode_mac_0, 0))

        self.connect((self.ieee802_11_moving_average_xx_0, 0),
(self.blocks_complex_to_mag_0, 0))

        self.connect((self.ieee802_11_moving_average_xx_0, 0),
(self.ieee802_11_sync_short_0, 1))

        self.connect((self.ieee802_11_moving_average_xx_1, 0),
(self.blocks_divide_xx_0, 1))

        self.connect((self.ieee802_11_sync_long_0, 0),
(self.blocks_stream_to_vector_0, 0))

        self.connect((self.ieee802_11_sync_short_0, 0), (self.blocks_delay_0,
0))

        self.connect((self.ieee802_11_sync_short_0, 0),
(self.ieee802_11_sync_long_0, 0))


    def get_window_size(self):

        return self.window_size


    def set_window_size(self, window_size):

        self.window_size = window_size

        self.ieee802_11_moving_average_xx_1.set_length(self.window_size + 16)

        self.ieee802_11_moving_average_xx_0.set_length(self.window_size)


    def get_sync_length(self):

        return self.sync_length


    def set_sync_length(self, sync_length):

        self.sync_length = sync_length

        self.blocks_delay_0.set_dly(self.sync_length)


    def get_samp_rate(self):

        return self.samp_rate


    def set_samp_rate(self, samp_rate):

        self.samp_rate = samp_rate
```

```
        self._samp_rate_callback(self.samp_rate)
        self.ieee802_11_frame_equalizer_0.set_bandwidth(self.samp_rate)
        self.bladeRF_Source.set_sample_rate(self.samp_rate)
        self.bladeRF_Source.set_bandwidth(self.samp_rate, 0)

    def get_lo_offset(self):
        return self.lo_offset

    def set_lo_offset(self, lo_offset):
        self.lo_offset = lo_offset
        self._lo_offset_callback(self.lo_offset)

    def get_gain(self):
        return self.gain

    def set_gain(self, gain):
        self.gain = gain

    def get_freq(self):
        return self.freq

    def set_freq(self, freq):
        self.freq = freq
        self._freq_callback(self.freq)
        self.ieee802_11_frame_equalizer_0.set_frequency(self.freq)
        self.bladeRF_Source.set_center_freq(self.freq, 0)

    def get_chan_est(self):
        return self.chan_est

    def set_chan_est(self, chan_est):
        self.chan_est = chan_est
```

```
        self._chan_est_callback(self.chan_est)
        self.ieee802_11_frame_equalizer_0.set_algorithm(self.chan_est)


def main(top_block_cls=wifi_rx, options=None):

    tb = top_block_cls()
    tb.start()

    def quitting():
        tb.stop()
        tb.wait()
    try:
        while 1:
            try:
                location = datetime.now().strftime('Time:%H:%M:%S')
                if gps_conn:
                    data = gps.readline()
                else:
                    data = b""
                if 'GPGGA' in data.decode():
                    #parse into a GPGGA object
                    print("\033[1;32;40m[+]\033[1;37;40m Found the
GPS\033[0;39;40m")
                    print("\033[1;32;40m[*] " + data.decode() +
"\033[0;39;40m")
                    gpgga.parse(data.decode())
                    lat = float(gpgga.latitude)
                    long = float(gpgga.longitude)

                    #shift the decimal point and convert long & lat into
decimal degrees
                    lat = ((lat-(lat%100))/100)+(lat%100)/60
```

```
                    long = ((long-(long%100))/100)+(long%100)/60
                    if gpgga.lon_direction == 'W':
                        long*=-1
                    if gpgga.lat_direction == 'S':
                        lat*=-1
                    location += ",lat:" + str(lat) + ",long:" + str(long)

                    #Grab the barometer altitude, subtract the starting
altitude to get relative to ground
                    if barometer:
                        altitude = sensor.altitude - start_alt
                        location += ",alt:" + str(altitude)

                elif barometer:
                    altitude = sensor.altitude - start_alt
                    location += ",alt:" + str(altitude)
                else:
                    print("\033[1;35;40m[*]\033[1;95;40m No GPS
Data\033[0;39;40m")
                    time.sleep(1)
                print("\033[1;32;40m[+]\033[1;37;40m Writing to
log:\033[0;39;40m")
                f.write(location)
                f.write('\n')
                f.flush()
                os.fsync(f)
                print("\033[1;32;40m[+]\033[1;36;40m " + location +
"\033[0;39;40m")
            except KeyboardInterrupt:
                print("\033[1;34;40m[*]\033[1;37;40m
Quitting...\033[0;39;40m")
                break
            except Exception as e:
                #No GPS attached or no signal or some such problem.
```

```
                print("\033[1;31;40m[-]\033[1;37;40m GPS
error:\033[0;39;40m")
                print("\033[1;31;40m[-]\033[1;95;40m " + str(e) +
"\033[0;39;40m")
                time.sleep(1)
                continue
    except:
        quitting()
    finally:
        quitting()



if __name__ == '__main__':
    main()
```

Pi Wifi Data Collection Script:

```
import serial
import re
import subprocess
import time
from decimal import *
from pynmea import nmea
import adafruit_bmp280
import board, busio
import os.path
import os
from threading import Event
from datetime import datetime

print("\033[1;32;40m[+]\033[1;37;40m Giving the GPS time to
acquire...\033[0;39;40m")
time.sleep(6)
```

```
print("\033[1;32;40m[+]\033[1;37;40m GPS acquisition time
elapsed\033[0;39;40m")


now = datetime.now()

timestamp = now.strftime("_%m_%d_%Y_%H%M%S")


stopper = Event()

file_path = '/var/log/geolocate/test/'

flightless = True


try:
  bmp = busio.I2C(board.SCL, board.SDA)
  sensor = adafruit_bmp280.Adafruit_BMP280_I2C(bmp)
  sensor.sea_level_pressure = 1013.25
  start_alt = sensor.altitude
  print("\033[1;32;40m[+]\033[1;37;40m Barometer Found (I2C)\033[0;39;40m")
  barometer = True
except Exception as e:
  print("\033[1;31;40m[-] No barometer found in I2C:\033[0;39;40m")
  print("\033[1;31;40m[-]\033[1;95;40m " + str(e) + "\033[0;39;40m")
  barometer = False


if not barometer:
  try:
    import digitalio
    spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
    cs = digitalio.DigitalInOut(board.D5)
    sensor = adafruit_bmp280.Adafruit_BMP280_SPI(spi, cs)
    start_alt = sensor.altitude
    print("\033[1;32;40m[+]\033[1;37;40m Barometer Found (SPI)\033[0;39;40m")
    barometer = True
  except Exception as e:
```

```
    print("\033[1;31;40m[-] No barometer found in SPI:\033[0;39;40m")
    print("\033[1;31;40m[-]\033[1;95;40m " + str(e) + "\033[0;39;40m")
    barometer = False

# Identify the next unused log file.
filenum = 0
fileset = False
while not fileset:
  filenum = filenum + 1
  filename = "log" + str(filenum) + timestamp + ".log"
  if not os.path.isfile(file_path + filename):
    fileset = True

filenum = 0
fileset = False
ACM_path = '/dev/ttyACM'
while not fileset:
  if not os.path.exists(ACM_path + str(filenum)):
    filenum += 1
  else:
    ACM_path = ACM_path + str(filenum)
    fileset = True
  if filenum > 5:
    break

f = open(file_path + filename,'w+')
f.write("\n")
f.flush()
os.fsync(f)

# Set up the output lists
addresses = []
```

```
channels = []
qualities = []
signal_levels = []
essids = []
getcontext().prec = 10
getcontext().rounding = ROUND_FLOOR

# Set up the GPS object
try:
  gps = serial.Serial(ACM_path,9600,timeout=1)
  gpgga = nmea.GPGGA()
  print("\033[1;32;40m[+]\033[1;37;40m GPS Found\033[0;39;40m")
  gps_conn = True
except Exception as e:
  print("\033[1;31;40m[-]\033[1;37;40m No GPS found:\033[0;39;40m")
  print("\033[1;31;40m[-]\033[1;95;40m " + str(e) + "\033[0;39;40m")
  gps_conn = False

# Start data collection
while not stopper.is_set():
  location = ""

  del addresses[:]
  del channels[:]
  del qualities[:]
  del signal_levels[:]
  del essids[:]
  gotAPs = False
  while not gotAPs and not stopper.is_set():
    try:
      output =
subprocess.check_output(['iwlist','wlan0','scan']).decode('UTF-8')
```

```
      output = output.splitlines()
      gotAPs = True
    except KeyboardInterrupt:
      print("\033[1;34;40m[*]\033[1;37;40m Quitting...\033[0;39;40m")
      stopper.set()
      # Sometimes the wireless doesn't work for a sec, just keep trying.
      break
    except Exception as e:
      print("\033[1;31;40m[*]\033[1;37;40m Error in the WiFi
scan:\033[0;39;40m")
      print("\033[1;31;40m[-]\033[1;95;40m " + str(e) + "\033[0;39;40m")
      continue

  gotAP = False
  for line in output:
    try:
      address = re.search('Address: (..:..:..:..:..:..)',line)
      if address is not None:
        addresses.append(address.group(1))
        gotAP = True

      channel = re.search('Channel:(.*)',line)
      if channel is not None:
        channels.append(channel.group(1))

      quality = re.search('Quality=(.*?)\/70',line)
      if quality is not None:
        qualities.append(quality.group(1))

      signal_level = re.search('Signal level=(.*?) dBm',line)
      if signal_level is not None:
        signal_levels.append(signal_level.group(1))
```

```
      essid = re.search('ESSID:\"(.*?)\"',line)
      if essid is not None:
        essids.append(essid.group(1))
    except KeyboardInterrupt:
      print("\033[1;34;40m[*]\033[1;37;40m Quitting...\033[0;39;40m")
      stopper.set()
      break
    except:
      print("\033[1;31;40m[-]\033[1;37;40m An error occured while examining
the address\033[0;39;40m")
      continue


  located = False
  gps.flushInput()
  gps.flushOutput()
  if gotAP:
    while (not located) and not stopper.is_set():
      try:
        location = ""
        if gps_conn:
          data = gps.readline()
        else:
          data = b""
        if 'GPGGA' in data.decode():
          # parse into a GPGGA object
          print("\033[1;32;40m[+]\033[1;37;40m Found the GPS\033[0;39;40m")
          print("\033[1;32;40m[*] " + data.decode() + "\033[0;39;40m")
          gpgga.parse(data.decode())
          lat = float(gpgga.latitude)
          long = float(gpgga.longitude)
          # shift the decimal point and convert long & lat into decimal
degrees
```

```
          lat = ((lat-(lat%100))/100)+(lat%100)/60
          long = ((long-(long%100))/100)+(long%100)/60
          if gpgga.lon_direction == 'W':
            long*=-1
          if gpgga.lat_direction == 'S':
            lat*=-1

          location = str(lat) + "," + str(long)
          # Grab the barometer altitude, subtract the starting altitude to
get relative to ground
          if barometer:
            altitude = sensor.altitude - start_alt
            location += ",alt:" + str(altitude)

          located = True
        elif barometer:
          altitude = sensor.altitude - start_alt
          location += ",alt:" + str(altitude)
          located = True
        else:
          location += "none"
          print("\033[1;35;40m[*]\033[1;95;40m No GPS or Barometer
Data\033[0;39;40m")
          time.sleep(1)
          if flightless:
            located = True
      except KeyboardInterrupt:
        print("\033[1;34;40m[*]\033[1;37;40m Quitting...\033[0;39;40m")
        stopper.set()
        break
      except Exception as e:
        # No GPS attached or no signal or some such problem.
        print("\033[1;31;40m[-]\033[1;37;40m GPS error:\033[0;39;40m")
```

```
        print("\033[1;31;40m[-]\033[1;95;40m " + str(e) + "\033[0;39;40m")
        time.sleep(1)
        continue
  for idx, val in enumerate(addresses):
    try:
      print("\033[1;32;40m[+]\033[1;37;40m Writing to log:\033[0;39;40m")
      f.write(datetime.now().strftime('Time:%H:%M:%S') + "," + essids[idx] +
"," + addresses[idx] + "," + channels[idx] + "," + qualities[idx] + "," +
signal_levels[idx] + "," + location)
      f.write('\n')
      f.flush()
      os.fsync(f)
      print("\033[1;32;40m[+]\033[1;36;40m " +
datetime.now().strftime('Time:%H:%M:%S') + ',' + essids[idx] + "," +
addresses[idx] + "," + channels[idx] + "," + qualities[idx] + "," +
signal_levels[idx] + "," + location + "\033[0;39;40m")
    except KeyboardInterrupt:
      print("\033[1;34;40m[*]\033[1;37;40m Quitting...\033[0;39;40m")
      stopper.set()
      break
    except:
      continue
  f.flush()
```

APPENDIX C. ANALYSIS SCRIPTS

SDR Data Processing Script:

(Process pcap and output relevant information to a file)

```
import dpkt
from scapy.utils import RawPcapReader
from scapy.layers.l2 import Ether
from scapy.layers.inet import IP, TCP
import time
import binascii
import datetime
import pytz

filename = 'log1_wifi_10_26_2019_141307_2452000000.pcap'

def process_pcap(file_name):
    local_format = "%H:%M:%S"
    print_file = 'converted_data.log'
    f = open(print_file, 'w+')

    for (pkt_data, pkt_metadata,) in RawPcapReader(file_name):
        count += 1
        utc_time = datetime.datetime.utcfromtimestamp(pkt_metadata[0])
        utc_zone = pytz.utc
        utc_time = utc_zone.localize(utc_time)
        pytz.timezone('America/Denver')
```

```
        local_time = utc_time.astimezone(pytz.timezone('America/Denver'))

        try:
          tap = dpkt.radiotap.Radiotap(pkt_data)
          t_len = binascii.hexlify(pkt_data[2:3])
          t_len = int(t_len, 16)
          ieee80211Frame = pkt_data[t_len:]
          wlan = dpkt.ieee80211.IEEE80211(ieee80211Frame)
          wlan.unpack_ies(pkt_data)
          wlan.unpack(ieee80211Frame)
          sig_str = tap.ant_sig.db
          ssid = wlan.ies[0].info

          f.write(ssid.decode() + "," + str(sig_str) + "," +
local_time.strftime(local_format) + '\n')
        except KeyboardInterrupt as e:
          print(e)

process_pcap(filename)
```

SDR Data Analysis Script – Distance Test:

Analyze the log file and find the SSID, Signal Strength, and Timestamp

```
f = open('converted_data.log', 'r')

strengths = [0]*12
count = [0]*12
averages = [0]*12

for x in f:
  [ssid, sig_str, ts] = x.split(',')
```

```
[h, m, s] = ts.split(':')
time = int(h)*60 + int(m)
if time < 846:
  strengths[0] += int(sig_str)
  count[0] += 1
elif time < 852:
  strengths[1] += int(sig_str)
  count[1] += 1
elif time < 858:
  strengths[2] += int(sig_str)
  count[2] += 1
elif time < 863:
  strengths[3] += int(sig_str)
  count[3] += 1
elif time < 868:
  strengths[4] += int(sig_str)
  count[4] += 1
elif time < 874:
  strengths[5] += int(sig_str)
  count[5] += 1
elif time < 880:
  strengths[6] += int(sig_str)
  count[6] += 1
elif time < 886:
  strengths[7] += int(sig_str)
  count[7] += 1
elif time < 891:
  strengths[8] += int(sig_str)
  count[8] += 1
elif time < 897:
  strengths[9] += int(sig_str)
  count[9] += 1
```

```
  elif time < 903:
    strengths[10] += int(sig_str)
    count[10] += 1
  else:
    strengths[11] += int(sig_str)
    count[11] += 1
print(strengths)
print(count)
for i in range(len(strengths)):
  if count[i] == 0:
    pass
  else:
    averages[i] = strengths[i]/count[i]
    print("Average for split %d: %d" %(i+1,averages[i]))
```

Pi Data Analysis Script – Distance Test:

Analyze the log file and find the SSID, Signal Strength, and Timestamp

```
f = open('log1_10_26_2019_134002.log', 'r')

averages = [0]*12
count = [0]*12

for x in f:
  tmp = x.split(',')
  if len(tmp) < 2:
    continue
  if tmp[1] == "SDR Thesis":
    temp = tmp[0].split(':')
    spot = int(temp[1])*60 + int(temp[2])
    if spot < 826:
```

```
  averages[0] += int(tmp[5])
  count[0] += 1
elif (spot) < 832:
  averages[1] += int(tmp[5])
  count[1] += 1
elif (spot) < 837:
  averages[2] += int(tmp[5])
  count[2] += 1
elif (spot) < 843:
  averages[3] += int(tmp[5])
  count[3] += 1
elif (spot) < 848:
  averages[4] += int(tmp[5])
  count[4] += 1
elif (spot) < 854:
  averages[5] += int(tmp[5])
  count[5] += 1
elif (spot) < 859:
  averages[6] += int(tmp[5])
  count[6] += 1
elif (spot) < 865:
  averages[7] += int(tmp[5])
  count[7] += 1
elif (spot) < 870:
  averages[8] += int(tmp[5])
  count[8] += 1
elif (spot) < 876:
  averages[9] += int(tmp[5])
  count[9] += 1
elif (spot) < 881:
  averages[10] += int(tmp[5])
  count[10] += 1
```

```
    else:
      averages[11] += int(tmp[5])
      count[11] += 1
print(averages)
print(count)
for i in range(len(averages)):
  print("Average for time slot %d: %d"% (i,averages[i]/count[i]))
```

Data Analysis Script – Speed Test:

Count the number of samples per time period

```
f = open('converted_data.log', 'r')

total_count = 0 # Total number of samples taken
time_stamp = 0 # Index of the 5-minute segment taken
step_t = 0 # Time of the beginning of the step (in seconds)
time = 0 # Current time
fives = [0] # number of packets per 5-minute time stamp
start_t = ''
stop_t = ''

for x in f:
  tmp = x.split(',')
  if total_count == 0:
    temp = tmp[0].split(':')
    start_t = int(temp[1])*60*60 + int(temp[2])*60 + int(temp[3])
    step_t = start_t
  if int(tmp[1]) > 0: # for SDR analysis
#  if tmp[1] == "SDR Thesis": # for pi analysis
    total_count += 1
    temp = tmp[0].split(':')
    time = int(temp[1])*60*60 + int(temp[2])*60 + int(temp[3])
```

```
#    if time < (current_t + 300):
    fives[time_stamp] += 1
    if time > (step_t + 300):
      step_t = time
      time_stamp += 1
      fives.append(0)
    stop_t = time


total_time = stop_t - start_t


total_time_h = total_time/3600
total_time_m = (total_time % 3600)/60
total_time_s = (total_time % 3600) % 60


print("Total time: %d:0%d:%d" %(total_time_h, total_time_m, total_time_s))
print("Total number of samples: %d" % total_count)
print("Average samples per minute: %f" %
((float(total_count)/float(total_time))*60))
for i in range(len(fives)):
  print("Number of samples for time slot %d: %d - Average per
minute: %#f" %(i+1, fives[i], float(fives[i])/5.0))
```