Software Defined Radio Localization Using 802.11-style Communications

A Major Qualifying Project Report
Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the
Degree of Bachelor of Science
in Electrical and Computer Engineering
by

Ryan Dobbins

Saul Garcia

Brian Shaw

Sponsoring Organization:
United States Naval Research Laboratory

Project Advisor:

Professor Alexander Wyglinski

Abstract

This major qualifying project implements a simple indoor localization system using software defined radio. Both time of arrival and received signal strength methods are used by an array of wireless receivers to trilaterate a cooperative transmitter. The implemented system builds upon an IEEE 802.11b-like communications platform implemented in GNU Radio. Our results indicate substantial room for improvement, particularly in the acquisition of time data. This project contributes a starting point for ongoing research in indoor localization, both through our literature review and system implementation.

Acknowledgements

We wish to thank Mr. Douglas Geiger and the United States Naval Research Laboratory at Washington DC for offering us the opportunity to complete this sponsored research and providing technical and financial support, as well as Professor Alexander Wyglinski for overseeing and advising the project and giving us guidance and confidence throughout our work. Additionally, we wish to thank Devin Kelly, Yuan Shi, Ramsey Abouzahra and the rest of the students in the Wireless Innovation Laboratory at Worcester Polytechnic Institute for assisting with laboratory equipment and offering suggestions for improvement.

Authorship

This project is the work of Ryan Dobbins, Saul Garcia, and Brian Shaw. All team members contributed equally.

Table of Contents

List of Figures

List of Tables

List of Equations

Glossary

802.11: See **IEEE 802.11**.

AoA: Angle of Arrival.

Base Station: A stationary node with a known location.

BBN: Raytheon BBN Technologies. See also: BBN80211.

BBN80211: A set of GNU Radio blocks that partially implement IEEE 802.11b.

BS: See **Base Station**

Data Fusion: The process of determining the position of a mobile station by assimilating distance measurements and known error.

DP: Direct Path, a straight-line route between a transmitter and receiver. This route may or may not be obstructed and may or may not be detectable.

DSP: Digital Signal Processing. Some articles use this abbreviation for a digital signal processor, a data processing device used to perform DSP.

DTDoA: Differential Time Difference of Arrival, a method for determining the distance between two nodes by measuring the round-trip time of a packet.

FCC: Federal Communications Commission, a regulatory body in the United States that is responsible for the wireless spectrum.

Flight Time: See **Time of Flight**.

FPGA: Field Programmable Gate Array, an integrated circuit implementing programmable digital logic.

FTW: Forschungszentrum Telekommunikation Wien, or the Telecommunications Research Center of Vienna, Austria.

GUI: Graphical User Interface used by a program to communicate with a human.

Handshake: A bidirectional correspondence between two nodes on a network. Often used to establish connections, achieve synchronization, or authenticate a user.

IEEE: Institute of Electrical and Electronics Engineers.

IEEE 802.11: The first IEEE 802.11x standard. This term is often used interchangeably with **IEEE 802.11x**.

IEEE 802.11x: A family of standards pertaining to wireless local area networks.

LAN: Local Area Network.

Localization: The process of determining the location of a node in a wireless network.

LoS: Line of sight.

Multilateration: See **localization**.

Mobile Station: A portable or moving node. The location of the Mobile Station must be found through localization.

MS: See **Mobile Station**

NLoS: Non-line-of-sight

Node: One device in a network.

NRL: United States Naval Research Laboratories, our project sponsor.

PPLD: Post Processing Localization Database, software we developed to perform data quality analysis and visualize errors.

RF: Radio Frequency.

RSS: Received Signal Strength, a class of localization techniques that use RSSI to calculate the distance a packet has travelled.

RTLS: Real Time Localization Solver, software we developed to perform real-time localization and visualize the locations of mobile transmitters.

RTT: Round Trip Time, the Time of Flight for one packet that is sent from one node to another and then returned.

RSSI: Received Signal Strength Indication, the measured magnitude of the received RF signal.

Round-Trip Time: The combined flight time of two packets used in a handshake.

SDR: Software Defined Radio.

Sensor: A device that collects data about its environment. In the context of our project, a sensor is a base station that collects data for localizing mobile stations.

SNR: Signal to Noise Ratio, the ratio of strength of the desired signal to the strength of the noise present in the environment. This value is often expressed logarithmically using decibels.

Synchronization: A state where two nodes or modules share the same system time and clock, often by knowing the exact time and frequency offset between them.

Time of Flight: The period of time required for a packet to travel between a transmitter and a receiver, as measured using timestamps of the time it was sent and the time it was received. This contains the time required for the first bit of the packet to propagate through the environment, the analog RF components of the transmitter and receiver, as well as error from various sources.

ToA: Time of Arrival, a method for determining the distance between a transmitter and a receiver using the flight time of a packet. Requires synchronization between all nodes in the network.

ToF: See **Time of Flight**

TDoA: Time Difference of Arrival, a method for determining the distance between a mobile station and nearby synchronized base stations.

UDP: Undetected Direct Path, a condition where packets propagating through the direct path between the transmitter and receiver are undetectable but other paths are able to deliver packets to the receiver.

USRP: Universal Software Radio Peripheral, a SDR platform from Ettus Research. Predecessor to the USRP2.

USRP2: Universal Software Radio Peripheral 2, the Ettus Research SDR platform used for our project.

UWB: Ultra Wide Band. This refers to a method of communications where a very high frequency carrier is employed along with multi-GHz bandwidth or frequency spreading.

WPI: Worcester Polytechnic Institute.

Executive Summary

Decades of research have been devoted to wireless localization projects. The success of Global Positioning Systems and cellular phone localization has enhanced outdoor applications such as emergency call localization, vehicular navigation, and the recovery of stolen articles. However, cost-effective indoor localization remains an area of ongoing research. Indoor localization requires a very high level of precision and accuracy but is stymied by exceptionally challenging wireless channel conditions.

Software Defined Radio has revolutionized the communications industry by providing unprecedented levels of flexibility. SDRs implement nearly all radio functionality in programmable components such as field programmable gate arrays (FPGA) and computers, allowing them to be reconfigured without any changes to the radio hardware. This flexibility makes SDR popular for applications such as rapid prototyping, scientific experimentation, limited-production devices and cognitive radio. One example of a low cost software defined radio platform is Ettus Research's USRP2.

The United States Naval Research Laboratory is developing an indoor localization system for improving the security of their secure research facilities. Our goal was to create a low-cost indoor localization testbed as a first step towards developing this system. This testbed is able to localize wireless transmitters using an array of three stationary USRP2 sensors positioned at known locations. Additional tools were developed for assessing the quality of this localization data, and will be used by the NRL as they continue to improve the accuracy of their results in increasingly challenging environments. Time of Arrival and Received Signal Strength Indication data is utilized by our system.

Our system utilizes a GNU Radio platform along with a partial implementation of IEEE 802.11 developed by Raytheon BBN Technologies. This system is capable of localizing cooperative devices transmitting in the 2.4GHz frequency band. The platform was modified as and enhanced with additional features. The full sensor platform configures itself using saved parameters, collects experimental data in a computer-readable format, and makes the collected data available to real-time localization software. Real-time localization software receives this data and uses it to display the computed location of the transmitter on a map of the environment. This real-time software is intended to be used as the basis for a final security product, while the stored results and data analysis software provide the NRL with essential tools for visualizing errors and developing improved methods of counteracting them.

1 Introduction

Localization has become part of everyday life through the use of global positioning system (GPS) devices, smartphones, and other technologies. These systems rely on signals from sources such as satellites, cellular towers, and wireless access points to locate the user. Many systems use localization information to provide directions or coordinates related to the user's current location. However, most commercially deployed systems have limited accuracy and precision in indoor environments due to interference, path loss, and other variables. A robust indoor localization system would be very beneficial in providing high accuracy positioning and correction for the unpredictability of the indoor environment. Applications for such a system include pedestrian navigation, locating firefighters and other personnel within a building, and the detection and isolation of possible threats to the building's security.

There are several techniques for localization, using data such as the time when signals arrived and received signal strength along with an understanding of how this data changes as mobile devices move throughout the environment. Although the free space behavior of electromagnetic waves is known to be predictable, localization techniques must account for environment-specific errors introduced during indoor operation in order to achieve a usable level of accuracy. These errors come from physical phenomena such as spatial and temporal fading, path loss, multipath, and interference from other wireless activity [40]. While outdoor systems available today are sufficient for the applications they serve, indoor environments demand much more precise position estimates while introducing unpredictable sources of error that outdoor applications are not forced to consider.

Numerous indoor localization techniques require a process known as fingerprinting to be performed before they can accurately localize mobile devices. This process is time-consuming, costly, and must be repeated for each new environment or when the environment changes. These constraints greatly limit the number and types of applications that can use indoor localization. As a result, a robust method requiring no or minimal knowledge of the area is sorely needed and could prove to be very valuable. A considerable body of research exists discussing the theoretical foundations of this problem and possible solutions but fewer focus on low-cost implementations.

This project proposes a solution for such an indoor localization system using a combination of Received Signal Strength Indication (RSSI) and Time of Arrival (ToA) based methods over an IEEE 802.11-like wireless communication system. Three USRP2 software defined radios with known locations create a sensor network able to localize an unknown wireless transmitter employing the WiFi standard, which is implemented using another USRP2 software defined radio. This localization setup is shown in Figure 1.

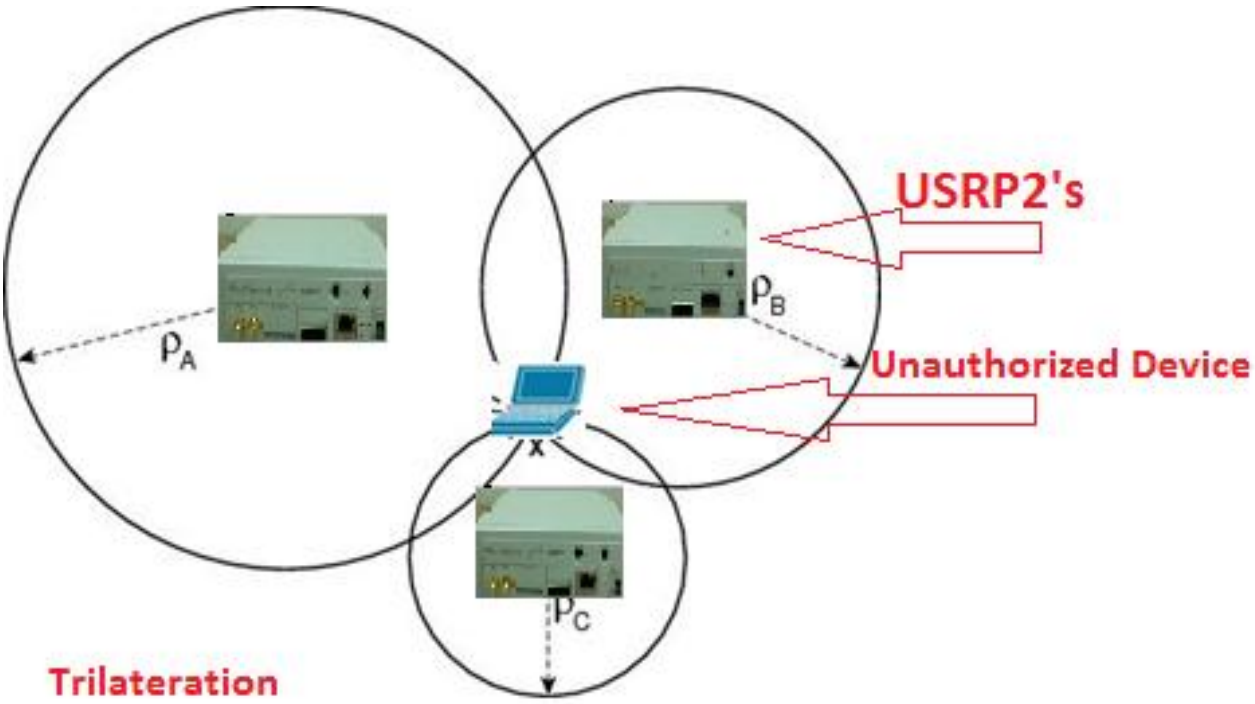


Figure 1: Localization system setup with three USRP2 sensors and a transmitter

Using ToA or RSSI methods, the distance between each of the sensors and the transmitter can be estimated. The distances along with the coordinates of the sensors are used in a simple trilateration algorithm to calculate the coordinates of the transmitter. Additional software processing compensates for the unpredictable indoor environment by calculating the error contributed by various sources and using the results to correct incoming data. The proposed system is also designed to be a starting point for future research that will continue to improve localization performance, add the ability to localize uncooperative transmitter, and prepare the system for deployment in a security application.

This document provides an introduction to the broad concepts required to understand the proposed wireless localization system and demonstrates the capabilities and limitations of the proposed system in its current state. Chapter 2 provides an explanation of key topics related to wireless localization and the platform utilized for this system. Focusing on the system design,

Chapter 3 presents the specific goals and requirements met by the proposed system, the components that constitute the proposed system, and the design decisions that guided system development. Discussed in Chapter 4 is the system implementation and the environments and methodologies used for deploying and testing this system. Results are presented in Chapter 5, explaining both the data collected and an analysis of the data's significance. Finally, the concluding Chapter 6 summarizes the results of this project, presents conclusions, and offers recommendations for future work in the field of indoor localization.

2 Wireless Networks and Localization Overview

This section introduces the concepts of wireless networks and localization, as well as several specific localization techniques. Known sources of error that arise from indoor wireless channels and equipment limitations are also explained and discussed.

Location information is invaluable for countless applications of wireless technology. Cellular operators use location information to find the source of emergency calls and manage infrastructure resources, while cellular devices and GPS receivers provide this information to location-aware applications such as navigation tools. Location information is also essential for wirelessly tracking objects such as vehicles and valuable goods [1]. Wireless localization's success in outdoor environments has led to considerable research into indoor localization. On June 14 of 2010, WPI hosted an International Workshop on Opportunistic RF Localization for next generation wireless devices, where many important figures, such as the CTO of Verizon Wireless, and the CEO of Skyhook Wireless, gathered to speak about location based mobile applications. This is sometimes referred to as geolocation. Some notable speeches at the convention were given by the Department of Homeland Security ("Localization Requirements for Homeland Security Applications"), QUALCOMM ("Technology Challenges and Opportunities in Indoor Location"), the Verizon Wireless Standards Group (“"The Role of Standards in Localization and Progress Made”) and many others researchers at the University of Massachusetts Lowell, and the Massachusetts Institute of Technology. These presentations reviewed the challenges associated with indoor localization [2].

There are multiple challenging indoor conditions that make localization more than just an elementary research topic. However these challenges are documented well on the web making it easy to learn, such as effects of scatter which take into account multipath fading (signals that

reflect off objects in the environment and arrive at the receiver with some latency), path delays (time it takes for a transmission to be received), Doppler spread (provides insight on angle of arrival (AoA)), non-line-of-sight (NLOS) conditions (obstacles blocking the direct path to a signal), noise introduced by the channel, and more [3]. Due to the difficulty of overcoming challenging indoor channel conditions using low-cost hardware, indoor localization remains a research field.

Mobile wireless devices can be localized in several different ways. In applications where the mobile device needs to know its own location, the mobile user can rely on existing wireless emitters at known locations or deploy their own localization infrastructure. Global Positioning System (GPS) uses this type of application, employing a combined approach where many independent mobile users localize themselves using emissions from cooperative satellites deployed by the United States Department of Defense [4]. Other applications are driven by the infrastructure maintainer's need to localize mobile devices. One example of this application is Enhanced 911, a cellular localization application where cell phone providers provide the location of customers placing emergency calls [1]. In order for a device to use radio frequency (RF) signals for localization, there needs to be a set of signals that can be used to determine the location of the mobile device. These signals are provided by permanent transmission infrastructures and other deployable devices. There are three major methods for providing an infrastructure for trilateration.

One solution is to develop a custom-designed infrastructure that can have any characteristics and be deployed as needed. However, this flexibility comes at a great expense in that the infrastructure must be specially designed and produced. This cost in time and money is beyond the scope of this project.

Another common solution is to use existing RF signals that are produced for other purposes for localization. Cellular phone signals and wireless Internet are commonly used for communications and are already widely deployed, and as the transmitters are often in fixed locations these signals can be localized without any additional infrastructure design and deployment costs. However, the localization system is then constrained by the properties of the signal used. For example, WiFi localization is forced to use direct-sequence spread spectrum (DSSS) on the 2.4GHz band to remain compatibility.

A third solution is to develop the infrastructure dynamically through an ad-hoc network. By attempting to localize several devices using each other, relative positions can be obtained without installing a permanent infrastructure. This has advantages in hostile environments where fixed infrastructures may be impractical, such as a coal mine, while preserving much of the design flexibility of a fixed infrastructure.

Other techniques may be classified as combinations of the above methods. For example, GPS implements a fixed satellite infrastructure, but allows these signals to be treated as an existing infrastructure by any number of mobile receivers. Ad-hoc networks can be used to extend the coverage of other techniques, by positioning nodes relative to known transmitters when available and relative to each other when unavailable. Furthermore, many systems incorporate RF localization with other technologies such as accelerometers and acoustic signals. Those approaches are able to provide improved accuracy due to the additional data input. Since our project uses a USRP2 and not a fully-functional robot, other sensors are not available.

2.1 Localization Techniques

There are many different techniques through which a position can be found relative to other known positions. These can be split up mainly into the categories of time based and signal

strength based. Each method has its pros and cons, and all require the use of multiple sensors with known locations.

2.1.1 Time of Arrival

Time of arrival (ToA), uses the travel time from the transmitter to the receiver, or time-of-flight (ToF), to measure the distance between the two. In order to properly localize with ToA, there must be at least three sensors. When the distances from three different sensors are known, the location can be found at the intersection of the three circles created around each sensor with the radius being the distance calculated. Imperfect measurements create a region of uncertainty between each of the sensors in which the transmitter might be contained. shown in Figure 2

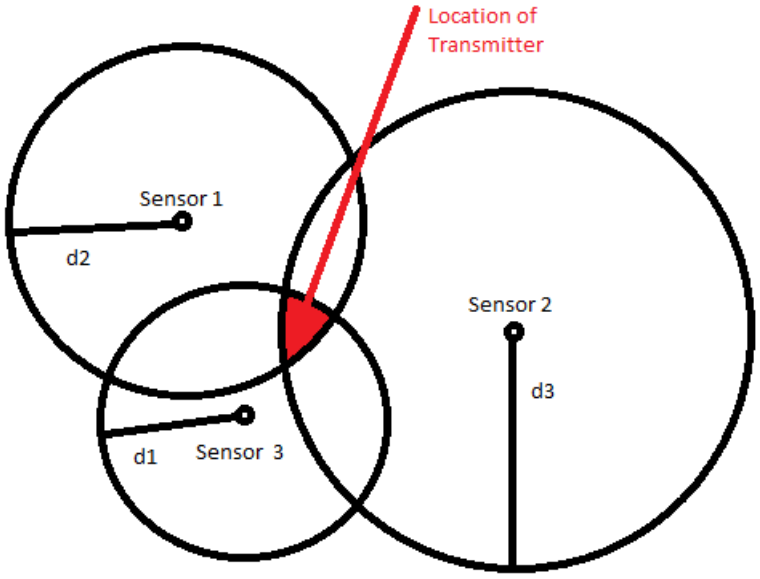


Figure 2: Time of Arrival

Since ToA relies on the difference between the time of arrival and time of departure, all receivers and transmitters must be synchronized so there is no error in the difference due to clock

offsets. This may prove to be a problem, especially considering the high speed at which the signals travel. Also, as with any time sensitive systems, there is also the possibility of significant hardware delays that must be accounted for to calculate the correct distances.

2.1.2 Time Difference of Arrival

Time difference of arrival (TDoA), uses multilateration, or hyperbolic positioning, to locate the intruder. It is very similar to ToA in that it uses the travel time from the transmitter to the receiver in order to measure distances. Instead of using the travel time from each receiver to find the distance between the transmitter and receiver, the difference in travel times from each sensor are used to find the distance between each sensor. This results in several hyperbolas to which the intersection of is the location of the transmitter [7]. Similar to ToA or any other time-based methods, synchronicity must exist in order for different time measurements to be accurate. However, since TDoA does not use the distance between the transmitter and the receiver, the transmitter is not required to be in sync with the sensor. Synchronicity is only required between all sensors since the calculation is based on their time/distance difference.

2.1.3 Received Signal Strength

The received signal strength (RSS) from a transmitter may be used to estimate the distance using a method known as received signal strength indication (RSSI). This method typically requires much prior work to be done before localization can take place. Using the RSS from the transmitter, the distance between it and the receiver can be estimated using existing data of RSSs found in different areas.

Although this method can prove to be suitable indoors, there is much room for error and it is very much an estimate in that there are many factors that affect the RSS. Path loss modeling can be used based on the area, but there is still a large margin of error and unpredictability. A

simple log-linear path loss model that can calculate the distance between the transmitter and receiver is presented in Equation 1.

$$\boldsymbol{RSSI = 10\alpha \log(d)}$$

$$\boldsymbol{d = 10^{(RSSI - RSSI_{calibration})(-10\alpha)} + d_{calibration}}$$

Equation 1: Distance calculation using log-linear path loss model

This equation can estimate the distance from a signal, given that d is the distance between the transmitter and receiver, $RSSI$ is the received signal strength measurement, $RSSI_{calibration}$ is the RSSI offset, α is the path loss gradient of the environment, and $d_{calibration}$ is the distance offset.

2.2 Calculating Distance using Timestamps

Every ToA/TDoA approach to wireless localization relies on determining the time of flight of a packet. Three different approaches to this problem have been identified in the literature.

2.2.1 Synchronized BS and MS, one-packet ToA

This approach uses a single packet sent from the BS to the MS containing the time it was transmitted, relying on BS-MS synchronization to eliminate clock-related drift (citation needed). Since the receiving MS knows when the packet arrived and that it is synchronized with the BS, the distance travelled can be calculated using the following formula:

$$\boldsymbol{d_{toa} = c * t_{toa}}$$

Equation 2: Distance calculation from time difference [26]

This simple equation is used to calculate distance based on the time it took a signal to go from the transmitter to the receiver when d_{toa} is the distance between the transmitters and receiver, c is the speed of light (approximately $3*10^6$ m/s), and t_{toa} is the time difference.

2.2.2 Synchronous BS, asynchronous MS, one-packet TDoA

This approach uses a single packet sent from the BS to the MS, but does not assume that synchronization has been established. By mathematically estimating the time offset between the BS clock and the MS clock, all of the benefits of a single-packet solution can be preserved without requiring BS-MS synchronization. Reference [10] describes a probability-based algorithm that can be used to implement TDoA distance measurements using this approach.

2.2.3 Differential Time Difference of Arrival (DTDoA)

This approach mathematically eliminates time offsets between the BS and MS by having both nodes send a packet to one another. Since the same time offset is added to the measured time of one packet and subtracted from the other, the correct time of flight can be calculated without any synchronization between the MS and the BS. Since the time offset can also be solved for using this data, this approach permits synchronization to be established using the localization packets [11]. This provides a distinct benefit to other services and protocols relying on the BS and MS. However, this method requires the MS and the BS to work together. In security-related applications where one device is passively attempting to determine the location of the other, this method is not realistic.

It is helpful to have similar BS and MS devices when using this approach. This is because this method, in its simplest form, assumes that the path travelled by each packet is the same. Our project uses identical USRP2 devices for both the BS and the MS. 802.11 devices are unlicensed,

and therefore subject to strict FCC regulations (in the United States) concerning the transmitted power. This consideration has resulted in the deployment of 802.11 systems where both the BS and MS are similar in design, also validating the assumption of this approach.

The PinPoint localization system developed at the University of Maryland is one example of this approach. PinPoint polls the BS and collects timestamps from both the packet sent and the packet received. By combining the time values as shown, this method is able to calculate and therefore overcome local clock drift and skew [11], as shown in Equation 3 and Figure 3.

$$\textit{Time of Flight} = (\tau_{B1} - \tau_{A1} + \tau_{A2} - \tau_{B2}) / 2$$

Equation 3: Time of flight calculation from PinPoint

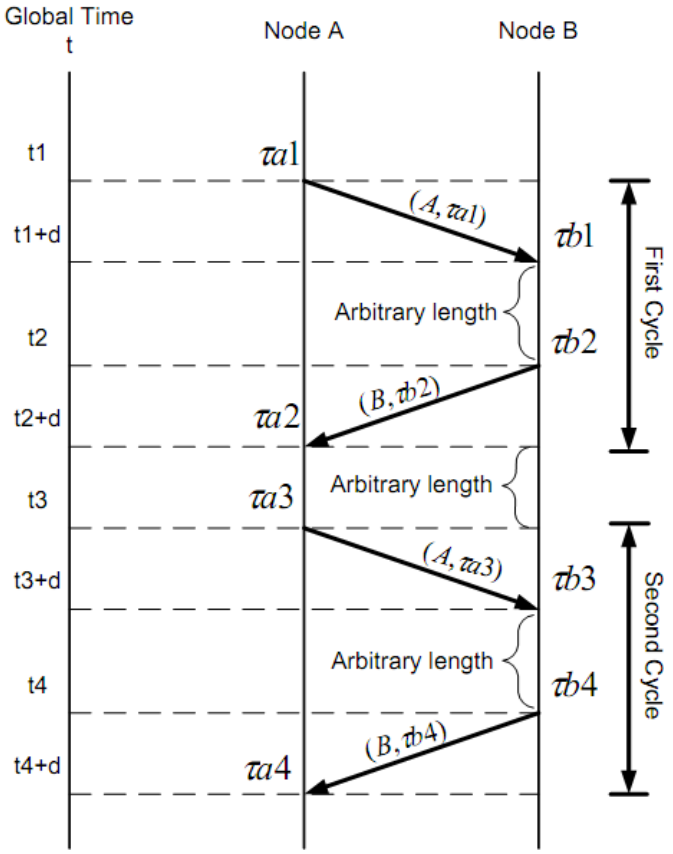


Figure 3: PinPoint DTDOA scheme [11].

τ_A represents a time value measured using Node A's local clock

τ_B represents a time value measured using Node B's local clock

2.2.4 Synchronization

ToA requires the clocks to be synchronized between each of the base stations. A scheme must be employed to ensure that any given pair of nodes can be synchronized and remain synchronized. One method of establishing synchronization is a polling method based on the DTDoA approach [11]. However, a more commonly used method of establishing synchronization is to exploit the preamble and training sequence found in 802.11 packets. Wang et. al. discuss a method for obtaining synchronization from the 802.11 packets [12].

2.3 Software Defined Radio Platforms

Traditional radios often consist of a super-heterodyne or integrated circuit transceiver implemented using dedicated hardware. While this type of implementation became ubiquitous as consumer devices from televisions to mobile phones proliferated, new hardware had to be developed for each platform that was created [13]. Software defined radio is a different approach. While both methods can implement the same transmitter and receiver design, software radios implement most stages in a digital form using programmable devices such as digital signal processing (DSP) and FPGAs. A common software defined radio design is to implement high-frequency RF signal processing in analog hardware and all intermediate frequency and baseband processing in software. This architecture is more flexible than using dedicated-purpose hardware and requires fewer parts but is bound by the reliability and security of the software. Additionally, software-defined radio uses more power than comparable specialized hardware [14]. Figure 4 offers a comparison between an analog radio and this software-defined radio architecture.

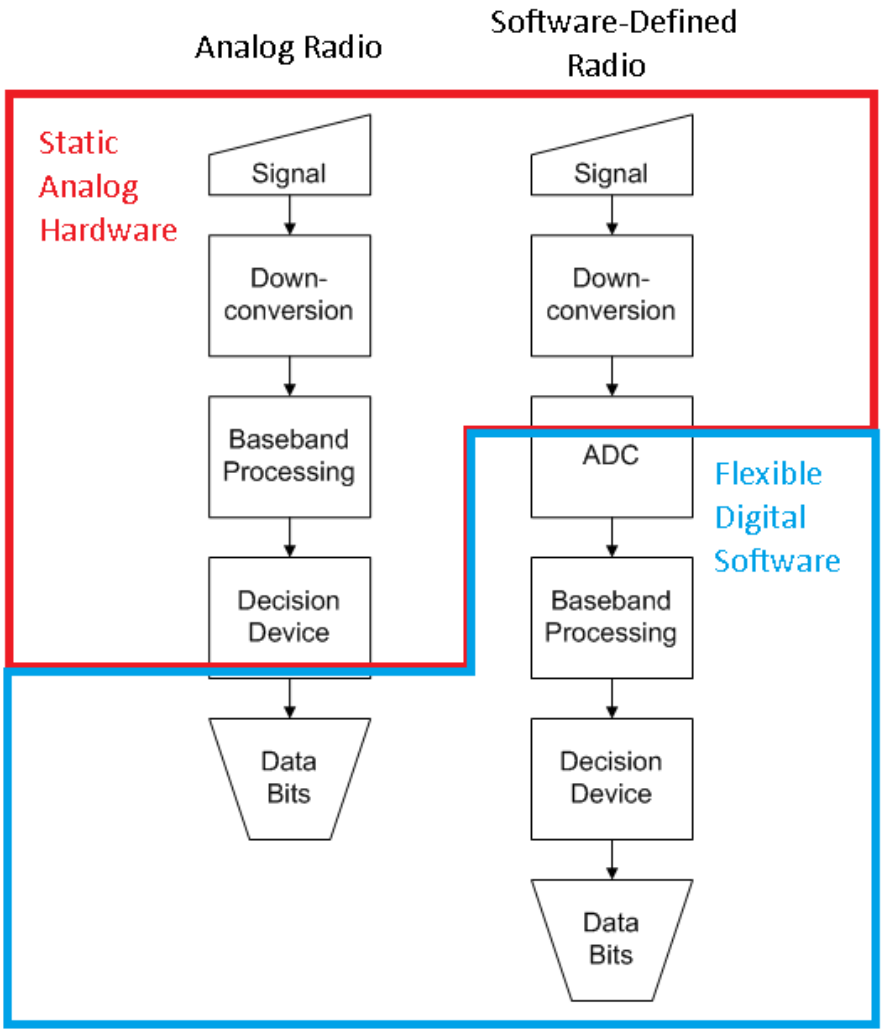


Figure 4: Diagram comparing SDR with analog radio [13]

Note that while Figure 4 demonstrates a receiver, both transmitters and receivers can be built in either fashion. Numerous significant flexibility benefits are obtained from using a software approach. First, the time required to deploy new products and product upgrades is greatly reduced by the near-elimination of new hardware development. In fact, already-deployed software radio devices can simply be reprogrammed to take advantage of emerging standards and features that were not available when the radio hardware was deployed [14] [15]. Additionally, new products can be created without any need to design and manufacture new hardware designs [16]. Interoperability between otherwise incompatible radio standards, such as

those used by different types of emergency responders, is another important benefit provided by software-defined radio [17]. Furthermore, software-defined radio is a key enabling technology for cognitive radio and dynamic spectrum access, emerging disciplines that utilize self-reconfiguring radios to adapt to channel conditions and unutilized spectrum [14] [17].

2.3.1 The Universal Software Radio Peripheral

The Universal Software Radio Peripheral, or USRP, is a low-cost software defined radio platform produced by Ettus Research. The device consists of a USB 2.0 host computer interface, an Altera Cyclone FPGA, and compatibility with a wide variety of plug-and-play high-frequency RF modules. Four input channels and four output channels are provided along with MIMO capability, supporting up to two full transceivers [18]. Up to 8MHz of signal bandwidth can be used on each channel [13]. Ettus Research also produces and sells USRP-compatible RF daughtercards that provide the USRP a range of high-frequency options. At time of writing, Ettus Research sells RF daughtercards to support frequencies from DC to 2.9GHz as well as the 4.9-5.9GHz band [19]. Third-party manufacturers such as Agile SDR Solutions also provide USRP-compatible daughtercards with frequency support up to 4.4GHz, though Agile's product has only become available recently [20].

The USRP and its Ettus Research daughterboards are fully open source hardware and can be supported entirely with open-source software. Full schematics for each of these components can be found on the Ettus Research website [19]. When the USRP is used with GNU Radio, discussed in Section 2.4.3, the resulting software defined radio is completely open source. Other software environments are also available for use with the USRP, including LabVIEW and Simulink. Simulink is discussed in Section 2.4.4, while LabVIEW is undergoing beta testing at the time of writing. All USRP hardware is sold as test equipment. This approach allows

maximum flexibility, but holds the end user responsible for complying with wireless spectrum regulations [19].

2.3.2 The USRP2

The USRP2 is another low-cost software defined radio platform produced by Ettus Research. The device has a very similar architecture as the USRP, though with different components capable of providing higher performance. The device consists of a Gigabit Ethernet host computer interface, a Xilinx Spartan FPGA, and compatibility with all USRP RF modules. Two input channels and two output channels are provided, and MIMO capability is supported when multiple USRP2s are connected together. Only one full transceiver is supported, but up to 50MHz of signal bandwidth can be used due to the 100MS/s ADC sampling rate [21]. The USRP2 is fully supported by GNU Radio and Simulink, which are discussed in Section 2.4.3 and Section 2.4.4 respectively.

At time of writing, the USRP2 is being discontinued in favor of the USRP N200 and USRP N210. These radios offer the same daughtercard and transceiver capabilities along with a 100MS/s ADC sampling rate, but have different Xilinx FPGAs and several more subtle differences. Ettus Research has stated that the USRP N200-series is code-compatible with the USRP2, allowing code written for the USRP2 to be used seamlessly with a newer radio [19].

2.3.3 GNU Radio

GNU Radio is a free, open-source software environment for the implementation of low-cost software defined radio systems. The GNU Radio project was created so the general public could experiment with radio hardware and the wireless spectrum, and does so by providing free software to complement low-cost hardware [20]. The project is used by a variety of academic, government, and commercial researchers as well as some amateur radio enthusiasts, and supports

both simulation environments and Ettus Research software-defined radio hardware products [20] [18][21].

As of GNU Radio 3.2.2 and 3.3.0, the software provides a framework for combining signal processing "blocks" combined together into "flowgraphs." Each block, written in the C++ programming language for performance reasons, implements a single data processing step such as matched filtering [13] [14]. GNU Radio provides numerous commonly-used blocks, though users are encouraged to create their own when a block they need is unavailable [20]. Each block executes in a separate operating system thread, allowing for pipelined processing that efficiently exploits multicore computer architectures [23]. Flowgraphs are typically written in the Python programming language and represent a data stream between the radio hardware and the user's choice of a data source or output. Additionally, flowgraphs can be produced automatically through a graphical user interface (GUI) tool called GNU Radio Companion (GRC). This implementation permits easy radio system design and development without compromising runtime performance [13] [20].

While GNU Radio is fully open source, has a strong user base, and is a well-established software-defined radio platform, the code remains in active development and is generally lacking in documentation. Significant bugs are routinely discovered and fixed, while backwards compatibility is often broken as the software attempts to benefit from the latest developments in the many libraries it depends on. Documentation consists of several tutorials, an API, email archives and code comments while end-user support is frequently performed by volunteers [20]. GNU Radio addresses the host computer software used with a USRP or USRP2, and when combined with either Ettus Research product a complete software-defined radio is formed.

2.3.4 Simulink Communications Blockset

As our project was under way, a proprietary alternative to GNU Radio became available. The MathWorks chose to extend their Matlab and Simulink software with a real-time communications processing toolbox, including USRP2 hardware blocks [21]. Simulink has long been established as a simulation toolbox, and can be used to implement flowgraphs in a visual form. This functionality is similar to GNU Radio Companion, except fully integrated with the Simulink product line. This provides easy access to, for example, common digital signal processing and logic functions. Additionally, this software interfaces with the well-established software package Matlab. Matlab focuses on offline vector and matrix manipulation as well as data visualization [22]. At the time of writing, numerous limitations still exist, such as comparatively weak USRP2 support and an implementation that was poorly optimized for real-time data processing on multicore processors. The authors were exposed to this product and its limitations during an academic course on software defined radio during which the MathWorks gathered feedback about its performance. It is expected that many of these issues will be resolved in future releases.

2.4 IEEE 802.11

IEEE 802.11 is a wireless standard utilized by a wide variety of devices to provide a network connection. The standard provides several physical layer specifications and one medium access control (MAC) protocol [33]. Different variants of the 802.11 standard have been used for different applications, including mainstream WiFi network devices. These variants rely on different physical layer transmission schemes and carrier frequencies, and as a result are not always interoperable. Table 1 shows a selection of IEEE 802.11 variants used by both consumer and more specialized applications, describing their transmission frequency band, transmission

scheme, and compatibility. All standards shown in the same vertical column are backwards compatible with each other. Standards are presented from top to bottom in chronological order of release, from oldest to newest.

Table 1: Comparison of selected 802.11x protocols [33]

802.11 (original) FHSS and DSSS 2.4GHz and others	802.11a OFDM 5.8GHz	802.11b DSSS 2.4GHz
	802.11g OFDM (DSSS preamble) 2.4 GHz	
	802.11p (draft) OFDM 5.9GHz	802.11n (DSSS preamble, various schemes)

Three IEEE 802.11 implementations currently exist for the USRP and USRP2. Each is described in the following subsections.

2.4.1 BBN80211b

BBN Technologies, now a subsidiary of Raytheon, developed a partial implementation of IEEE 802.11b for the USRP as part of the ADROIT project. ADROIT's objective is to create a cognitive wireless network capable of reconfiguring the radios according to network needs and channel conditions. The primary objective of BBN in implementing IEEE 802.11 over a software-defined radio was to create a radio designed from the ground up to be cognitively controlled [23].

The ADROIT project developed a wide variety of software packages, many of which were publicly released under open-source licenses. In addition to network management and cognition software, a functional IEEE 802.11b transmitter and receiver were produced for the USRP platform as well as improvements for GNU Radio 3.1.1 [24]. The receiver is able to completely decode IEEE 802.11b packets transmitted at a 1Mbps data rate and partly decode packets sent at a 2Mbps data rate [24].

BBN80211 is not without limitations. No automatic gain control is implemented, making the system's performance sensitive to the user-specified gain. Additionally, to overcome the bandwidth limitations of USB 2.0, the receiver subsamples incoming signals before delivering them to the host computer [24]. Subsequent contributors to the BBN80211 codebase have ported it to the USRP2 platform and GNU Radio 3.2.2, circumventing the limitations of USB 2.0 by using Gigabit Ethernet as the interface with the host computer. Examination of the BBN80211 source code indicates a lack of both frequency and phase correction [25]. These limitations reduce the robustness of BBN80211b to adverse channel conditions.

2.4.2 SPAN 80211b Receiver

Mohammad Firooz and Neal Patwari of the Signal Processing Across Networks (SPAN) Laboratory at the University of Utah developed an IEEE 802.11b receiver for the USRP. This receiver project was motivated by one of the primary limitations of BBN80211b at the time. Since BBN80211b was deliberately downsampling the signal and reducing it to an 8-bit format for transmission over the USB 2.0 host computer connection, both data rate and SNR were being compromised. This issue was resolved by implementing the OFDM de-spreading operation on the Altera FPGA located within the USRP itself. The result is a USRP2 receiver based on BBN's work but fully capable of receiving IEEE 802.11b packets transmitted at 2Mbps [26].

2.4.3 FTW 802.11p Encoder and Transmitter

Another significant software-defined radio implementation of IEEE 802.11 is an 802.11a/g/p transmitter developed by researchers at Forschungszentrum Telekommunikation Wien (Telecommunications Research Center of Vienna) and the University of Salento. This transmitter is compatible with GNU Radio 3.2.2, employs the USRP2 platform and was fully functional as of February 2010 [24]. The code is open source and can be found on CGRAN (the

Comprehensive GNU Radio Archive Network [24]. Although designed to create IEEE 802.11p frames for USRP2 transmission, both IEEE 802.11p and the encoder use the same physical layer as IEEE 802.11a and IEEE 802.11g [24]. At time of writing, no corresponding 802.11p receiver has been developed for a software-defined radio platform.

2.5 Challenges in Flight Time Calculation

Numerous sources of error exist for the calculated flight time. The resolution of the timestamps limits the precision of the distance measurements, while a delay is introduced by the hardware required to process the arriving signal regardless of the system implementation [9]. Multipath conditions result in time-delayed copies of the packet arriving at the receiver. Large errors can be introduced when the line-of-sight packet is undetectable or obscured by multipath copies, while smaller yet still significant errors result when low bandwidth conditions coalesce multiple copies into a single RSSI peak [6].

2.5.1 Resolution of the Timestamps

There are various techniques for determining the time of arrival of a signal in these kinds of systems, depending on the level of accuracy needed versus the cost of the fully implemented system. A purely software-based approach is to detect when the first decoded bit arrives. The precision of such a system is limited to the data rate of the connection, as it is uncertain when the asynchronous signal arrived during the time between each bit [7]. Commercially available 802.11 chipsets are limited to 1 microsecond of accuracy at the MAC layer, due to the 1Mbps data rate used by the 802.11 preamble [8]. However, this level of precision is insufficient for indoor environments. Each bit arrives once every microsecond.

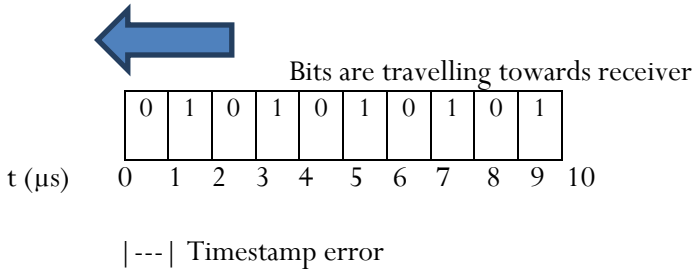


Figure 5: Arrival of bits showing possible timestamp error

RF signals in free space propagate at the speed of light, approximately $3*10^8$ m/s . Using this estimate and Equation 4, we can calculate the approximate distance travelled by the packet in 1µs.

$$\boldsymbol{d = c * (10^{-6} s)}$$

$$\boldsymbol{d = 300\ m}$$

Equation 4: Distance calculation in 1 microsecond

A 300-meter (approximately 1000 feet) error is unacceptable for indoor localization, as it is longer than many buildings. Software-based approaches also encounter issues if the time-measuring software is run on the same machine as other programs. The processing time on systems where the processing hardware is shared with other applications is uncertain, so an unknown delay is added to the time measurement [9]. For these reasons, indoor localization requires that all some form of hardware-based approach to timestamping. It is important to ensure that the timestamp resolution is as high as possible, to minimize the error introduced. If the packet detection technique is implemented on the FPGA, the measurement uncertainty is theoretically limited by the USRP2's hardware capabilities. Since the sampling rate of the USRP2's ADC is 100Msamples/second [21], the timestamp resolution is effectively limited by a 100MHz clock.

Efforts should be undertaken to improve the resolution of the timestamp beyond the USRP2's sampling rate in order to achieve maximum performance. This is because the time period between samples is 10ns, and by Equation A will result in an uncertainty of approximately 3.3 meters (10 feet). Two general approaches to this problem have been identified in the literature.

Table 2: Comparison of analog and digital packet arrival detection

Approach	Benefits	Costs
Analog, asynchronous arrival detection.	Uncertainty can be improved to sub-nanosecond levels using methods described in [7].	Requires an analog packet detector that bypasses the baseband ADC.
Digital approach using quantized data.	No analog baseband feed or analog detector design required.	Complex data processing needed to improve resolution. Unable to achieve sub-nanosecond resolution

Using an asynchronous analog approach eliminates the limitations of the ADC. One possible implementation of the analog approach is to use an analog comparator to implement the Threshold RSSI technique. The binary output of the comparator could then be fed into an asynchronous digital input and used to provide a binary timestamp signal. Analog implementations of more advanced techniques can also be used so long as they are able to act asynchronously on the signal as it arrives. The arrival time of this asynchronous timestamp signal can be measured using any of the methods discussed in [7] with sub-nanosecond accuracy even on FPGAs with a 100MHz clock.

However, the costs associated with an analog implementation are too high. The limitations associated with Threshold RSSI, as discussed in Section 2.5.1, are too great for reliable use in an 802.11-like wireless localization system. Most importantly, modifying the analog hardware of the USRP2 or its daughterboards is not an option for this project. This is due to the cost of the equipment and the fact our devices are shared with other students and researchers at WPI.

2.5.2 Hardware Delay

Even under perfectly synchronized conditions with strong line of sight, the time measured for a packet to travel from one node to another does not truly reflect the distance

between the nodes. This is because there is a delay at both the transmitter and receiver. This delay originates in the device hardware, since it takes time for a packet to propagate from the end of an antenna to the timestamping module. Additionally, the timestamp determination logic implemented in the FPGA introduces measurement and calculation delays. However, this hardware delay is constant and therefore easily compensated for [9]. Reference [11] recommends tuning each mobile and base station by asking them to transmit a packet to themselves when each device is powered on. The flight time of the self-calibration packet incorporates both transmitter and receiver delay, but virtually no travel distance. We have observed that the transmit antenna and receive antenna of our USRP2 devices are approximately one inch apart, corresponding to a flight time of approximately 0.1ns. This error in the hardware delay calculation is one to two orders of magnitude less than the desired accuracy of our system.

2.5.3 Multipath Rays

When the first bit of a packet is arriving at the receiver, numerous multipath "rays" arrive carrying the packet. These rays appear as peaks in the received signal strength, as seen in [12]. The first of these rays is the direct path, a line-of-sight path going directly from the transmitter to the receiver. In non-line-of-sight conditions, a secondary ray produced by multipath may be much stronger than the direct path ray. In some instances, this direct path component may be undetectable. In any event, it is important that the time the first ray arrives is precisely documented to ensure the measured flight time is actually the time that the packet was in transit between the two nodes.

As the first detectable ray of a wireless signal bit arrives at the receiver, the amplitude of the detected signal rises above a predetermined detection threshold. This rise is followed by a peak and then both the other multipath rays and other bits. The peak of this signal is considered

the exact time that the ray arrived. However, the signal has yet to be decoded into meaningful data. If a packet was not successfully received, it is likely that noise triggered the timestamp module and so the timestamp must be discarded. Detection techniques differ in how they determine when the first ray arrived and how they ensure that the ray begins to a valid transmission.

2.5.4 Bandwidth limitations

The bandwidth of the received signal limits any algorithm's ability to determine when the direct path arrived. When multiple multipath rays arrive very close together in a low bandwidth signal, they combine into one perceived peak. This means that the detected peak does not accurately represent a distinct direct path ray, introducing an error [6]. The correlation algorithms examined in [1], including TD-MUSIC, consider limited bandwidth situations. The performance of these algorithms degrades as the bandwidth decreases though peaks are still noticeable.

2.5.5 Undetected Direct Path

In a Non-Line-of-Sight (NLoS) channel where the direct path exists but is below the detection threshold of the receiver. This results in a large error in the distance estimate, as the distance traveled by the packet is not the distance between the transmitter and the receiver. Much research has been conducted on detecting and eliminating this error. This research is discussed further in Section 2.7 Undetected Direct Paths, as these techniques are often implemented after distances have been calculated and the sensor is attempting to resolve measurements from multiple transmitter units.

2.6 Methods of Detecting Arriving Packets

Every ToA/TDoA approach to wireless localization relies on determining the time of flight of a packet transmitted between the transmitter and receiver. This is accomplished by knowing when the packet was transmitted as well as when it was received and removing any known errors. Having an accurate time of flight allows the distance to be calculated, as electromagnetic signals travel at the known speed of light. Much research has focused on achieving very precise estimates of the arrival time, using a variety of hardware and software based approaches. A hardware implementation is needed to keep measurement error small or constant, indicating that the chosen approach is to be implemented on the USRP2's FPGA.

2.6.1 Peak Detection Using RSSI

When no packets are present in the medium, the magnitude of the measured signal solely depends on environmental noise. The presence of a packet increases the RSSI [35]. The simplest method for detecting the arrival of a packet is to compare the received signal strength with a known threshold and signal the packet's arrival once the threshold is exceeded. This method has the additional advantage of being easy to implement using an analog comparator, producing an asynchronous digital signal. The asynchronous signal can then be used as an input to the high-resolution timestamping methods discussed in [7], making sub-nanosecond resolution possible. However, it is important to choose an appropriate threshold when using this method. The noise power is generally unknown, unpredictable, and subject to change. Path loss also affects the RSSI [35]. IEEE 802.11 implements power control [33], which means that the transmitter output power may change over time as well.

According to [6], the peak of the received signal energy indicates the correct time of arrival for the first symbol of the packet. This approach is not dependent on the magnitude of the

total received power [35]. The rate of change in the RSSI can be used to help determine when a peak is detected and the timestamp should be recorded. To reduce the system's sensitivity to noise, digital implementations can employ a double sliding window approach. Two values for the RSSI are calculated by taking the sums of two adjacent time intervals. As new samples arrive, old values are discarded. This is implemented using Equation 5.

$$m_{n+1} = m_n + {r_{n+1}}^2 - {r_{n-L+1}}^2$$

Equation 5: RSSI [22]

This equation uses m to indicate the output value of the sliding window, r_n to indicate an individual RSS measurement, and L is the length of the sliding window.

Larger values of L improve noise resilience, at the expense of a longer algorithm-induced time delay and increased processing requirements. Since the delay is constant it does not present a significant issue, but both sliding windows need to be able to update once every clock cycle. Since a larger sliding window requires more memory [35], the size of the sliding window is limited by the capabilities of the FPGA hardware.

2.6.2 Matched Filter Correlation

Matched filtering is a well-known technique for determining when a signal arrived, and is commonly used in noisy environments [30]. Given a known pattern within the transmitted signal, a matched filter can be created by reversing the pattern in time and multiplying it by the arriving signal. A fixed time delay equal to the length of the pattern being correlated must be introduced to ensure the filter is causal [30]. The matched filter can be represented with Equation 6.

$$h(t) = A * s(-t - \tau)$$

Equation 6: Matched filter

This equation gives the result of a matched filter when $s(t)$ is the known waveform, τ is the length of the known waveform in the time domain, and A is a filter gain.

The output of the filter, h(t)*s(t), is a correlation function that maximizes the SNR of its output. The peak values of this correlation function indicate the strongest matches between the received signal and the anticipated signal. When a strong correlation appears, a timestamp can be created. If the matched filter method is used, care must be taken to select an appropriate known waveform that will indicate the arrival of the packet. IEEE 802.11 provides a known preamble at the beginning of every packet consisting of "scrambled ones" [33], allowing matched filters to be created using part of or the entire known signal.

2.6.3 MUSIC-based Correlation Algorithms

An algorithm known as MUSIC was developed as an improved correlation method for determining the time of arrival. Reference [39] presents an improved version of this algorithm known as Root-MUSIC in the context of an 802.11a/g system. Reference [1] presents and compares three very similar algorithms known as TD-MUSIC, FD-EigenValue, and FD-MUSIC. Due to the similarity of these algorithms, TD-MUSIC is presented below. This algorithm was shown to have the best performance of the three variants discussed.

TD-MUSIC creates a pseudospectrum similar in appearance to a Fourier transform that may be used to isolate individual peaks when determining the time at which the direct path arrived. The pseudospectrum $F(\tau)$ is generated using the following steps:

1) Formulate an autocorrelation matrix for the received signal.
2) Perform Eigenvalue decomposition on the autocorrelation matrix.

3) Estimate the signal subspace dimensions. This is crucial to the performance of the TD-MUSIC algorithm [1].
4) Separate the signal subspace using the magnitudes of the Eigen values.
5) Pass the new signal subspaces through the function defined in Equation 7.

$$F_{TD-MUSIC}(\tau) = \frac{1}{s(n-\tau)^T U_N U_N{}^T s(n-\tau)}$$

Equation 7: TD-MUSIC algorithm

Many of these quantities represent standard quantities in digital signal processing. τ is the time offset between the current value in the sample and the first value of the sample, while $s(n-\tau)$ is the generalized signal vector. U_N represents the noise subspace.

2.7 Undetected Direct Paths

When there is no line of sight, the direct path ray may be undetectable. In this case, the distance calculated is the total distance that particular ray travelled and not the distance between the MS and BS. This creates a large range calculation error, biasing the distance estimate and posing a critical problem for localization [27]. This can be the result of two different conditions. One situation is that a large metal object is blocking the direct path between a transmitter and receiver that are otherwise close to one another. Examples cited in the literature include elevators and the RF-isolated chambers found in some research laboratories [6] but large metal filing cabinets can have a similar effect. A second possible situation is that the transmitter and receiver are further away and the direct path is undetectable due to path loss. Another source of UDP-related bias results from the DP signal propagating at a lower speed through some of the objects that are found along the direct path.

There are many different approaches used to take UDP conditions into account.

1) Accept the error.

One approach is to assume that the MS has line of sight with all the BS units, and that the direct path is detectable. If these assumptions hold, there is no UDP and no compensation is necessary. The advantage of this approach is that no algorithms need to be implemented for compensating for the UDP. However these assumptions do not hold if UDP conditions exist, which is common in indoor environments [6] [28].

2) Improve transmitter coverage and discard the UDP measurements.

In situations where the LOS distance measurement cannot be reconstructed, the distance measurement can be detected as an outlier and discarded. If more than the minimum number of neighbors are present (3 for 2D space, 4 for 3D space) the redundant data can be used to replace the outlier in triangulation. In order to discard a measurement as an outlier, some algorithm needs to determine which if any data values need to be discarded. Reference [38] presents three different methods of identifying LOS and NLOS channels, although only one is viable for an unknown indoor environment. This method relies on the kurtosis (peakiness) of the multipath channel. The multipath channel for the data is found by the equalizer in 802.11. This kurtosis, along with the kurtosis of known IEEE path-loss models, is then modeled using a log-normal distribution with the following probability density function in Equation 8.

$$\boldsymbol{p(k) = \frac{1}{k\sqrt{2\pi\sigma_k}} \exp\left(-\frac{(\ln(k) - \mu_k)^2}{2{\sigma_k}^2}\right)}$$

Equation 8: Probability density function of kurtosis

In this density function, $\mu_{k\text{is}}$ the mean of ln(k) and $\sigma_{k\text{is}}$ the standard deviation of ln(k). If the probability density function (PDF) matches that of a LoS path-loss model, the

signal is deemed LOS. If it matches the NLoS path-loss model, the signal is deemed NLoS. [38] mentions that this holds true of cellular systems in home, office, and industrial indoor environments but not in outdoor environments. Since that paper focuses on outdoor environments, it does not discuss this technique further.

Additional BS units can be provided to eliminate severe UDP conditions. BS units can be located on the opposite side of large metal objects to provide detectable direct paths at more locations. Reference [6] performed an experiment in which this technique effectively eliminates large ranging errors caused by UDP conditions. This experiment took place in the same environment as shown in [7], with two additional LoS BS units. While this method improves localization accuracy when the BS locations are well-chosen, all costs associated with installing and maintaining additional BS units are increased.

3) Rely on distance calculations performed by neighboring nodes.

In an ad-hoc network, each node computes its location relative to its surrounding nodes. This information is sometimes shared between nodes to provide synchronized coordinate systems and improved localization data. If a neighboring node was able to calculate the distance more accurately, its measurements can be relied upon instead of the MS's own measurements. Reference [20] proposes such a system, but does not produce or test a prototype. Reference [6] also discusses this technique. This method is directed towards ad-hoc networks where multiple mobile users are present. Since our test system assumes that only one MS is present, this technique is not immediately useful but can be added to our system in the future should multiple mobile user capability be implemented.

4) Compensate for the UDP.

Several different methods exist for compensating for the undetected direct path. This is considered a major research area, and is the topic of numerous background materials. Selections of the published algorithms that are suitable for ToA localization in an unknown environment are described below.

2.7.1 Exploit frequency, temporal, or spatial diversity

Shadow fading in communications systems can be compensated for by repeating the signal multiple times. This can be done by broadcasting it simultaneously over multiple frequencies, repeating it at a different point in time, or by using Multiple Input Multiple Output (MIMO) spatial multiplexing techniques [6]. IEEE 802.11 preambles use DSSS, which provides frequency diversity that can potentially be used to help correct for UDP conditions. However, it does not provide any other form of diversity [33]. Communications applications use this diversity to overcome shadow fading, a similar phenomenon where the transmitted signal momentarily degrades. However, [6] points out that many of the techniques that have been developed to overcome shadow fading do not consider the complexity of radio propagation in an indoor environment. This remains an area for research.

2.7.2 Use AoA to exploit a non-direct path for localization

It is possible to calculate the distance of a non-direct path and use it for localization when additional components have been added to the MS. Reference [6] shows how a signal that has been reflected once off a wall can be reverse ray-traced and used to calculate the distance between BS and MS. This method requires a known direction of travel, as well as a precise sectored antenna capable of finding the AoA. Since the USRP2 lacks a sectored antenna capable of finding AoA, this method is not feasible for our project.

2.7.3 Maintain a history of the MS's measurement noise, and use it to reconstruct LOS

Numerous techniques exist for correcting the NLOS error given knowledge of the environment. For example, [28] is able to combine knowledge of the environmental path loss with the signal strength of the direct path ray to determine the distance. Our project assumes that the environment has not been mapped; therefore this approach is not useable. Reference [27] is able to use a statistical method to remove identified NLoS biases from distance measurement data. However, this statistical approach assumes that the standard measurement error exceeds the NLoS error and that a known bound for the NLoS error can be established. These assumptions are valid for the cellular system described [27], but not for an indoor localization system [6].

2.7.4 Comparison of UDP compensation techniques

Table 3 provides a comparison of the discussed UDP compensation techniques that are possible to implement with our system.

Table 3: Comparison of UDP compensation techniques

Method	Benefits	Costs
Accept the error	Simplest to implement No additional hardware	Large errors under UDP conditions
Improve transmitter coverage, rely on other measurements	Simple to implement Eliminates most UDP error	Requires additional BS units
Compensate for UDP using frequency diversity	No additional hardware	Research required Limited by system bandwidth

2.8 Data Fusion

After the distance between the intruder and each sensor have been established, the localization system must then determine the location of the intruder on a Cartesian coordinate system. This coordinate system contains the known coordinates of each sensor, and can be displayed to the user using a GUI. Time-based protocols provide measurements of the distance

between each sensor and the intruder, as discussed in Section 2.2. Since our project necessitates a time-based approach, the chosen data fusion algorithm must use these measurements.

Due to the sources of error discussed in Section 2.4, each distance measurement is usually too large. As seen below in Figure 6, there is no single point where all of the distance measurements can agree the intruder is at. Various data fusion algorithms have been developed to resolve these measurements, determine the most likely location of the intruder, and minimize the localization error. These calculations can be processed on any node that knows all the distance measurements.

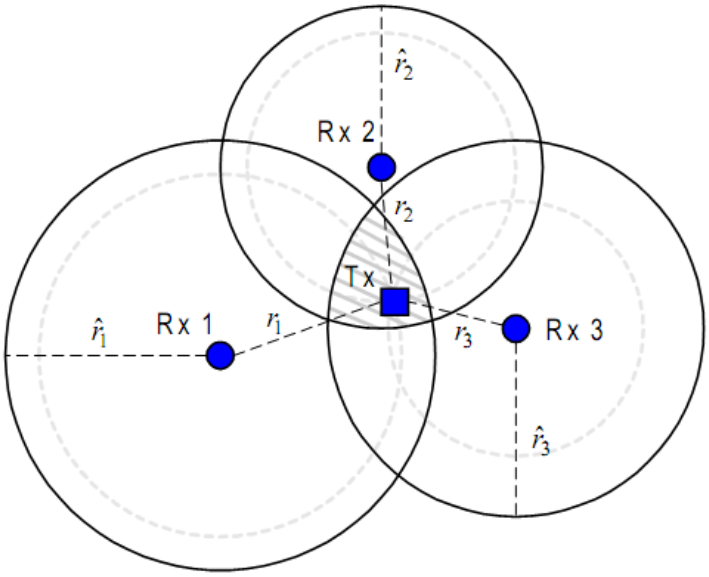


Figure 6: How Range Error Affects Triangulation [31]

Figure 6 shows a mobile station, labeled "Tx," being localized with respect to three BS units labeled "Rx 1" through "Rx 3." $\check{r}_1$ through $\check{r}_3$ are the distances measured between the MS and each BS, while r_1 through r_3 are the distances between these nodes after the data fusion algorithm has determined the location of the MS. This example illustrates a moderately good case, as there is a substantial overestimate of each distance but the distances are still resolvable.

Different data fusion algorithms have different ways of deciding where within the center region the MS actually is and if any of the distance measurements are too inaccurate to be utilized.

Each of the listed algorithms uses the geometry of the nodes in relation to one another. Statistical approaches are generally associated with techniques where the environment is mapped, whereas our project does not permit prior mapping of the environment.

2.8.1 Least-Squares Error Minimization

The Least-Squares method is a generic mathematical process for approximating the solution to a system of equations where more equations are provided than variables. It attempts to minimize the error by which each equation deviates from the approximated result. Least-squares algorithms are typically implemented in matrix form, to permit an arbitrary number of equations to be solved.

One variant of this technique is the weighted least-squares method. Each distance measurement is given a weighting based on the receiver's level of confidence in it. When weighted least-squares is used for localization, the BS closest to the MS is typically given the strongest weight. This is because ranging errors, such as those related to multipath, frequently increase the measured distance (citation needed). The MS also receives a much stronger signal from a nearby tower, especially in the context of a cellular network.

[32] provides three variants of the least-squares method, using a series of linear equations in matrix form to minimize the distance estimation error for each input. These linear equations are approximations of nonlinear systems, and can be used to minimize the error associated with each distance measurement. The error ε is calculated using the equation below [19]:

$$\underline{\epsilon} = \underline{\delta} - 2R_s\underline{d} - 2S\underline{x}_s$$

where

$$\underline{\delta} \triangleq \begin{bmatrix} R_2^2 - d_{21}^2 \\ R_3^2 - d_{31}^2 \\ \vdots \\ R_N^2 - d_{N1}^2 \end{bmatrix}, \qquad \underline{d} \triangleq \begin{bmatrix} d_{21} \\ d_{31} \\ \vdots \\ d_{N1} \end{bmatrix},$$

$$S \triangleq \begin{bmatrix} x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix}.$$

This equation error can then be used in one of the many methods proposed by [32], some of which are referred to in Section 2.8.2. Weights can be applied to each of the measurements without much difficulty using this, but tuning may be required to find appropriate weights. Reference [29] references additional least-squares algorithms that use a two-step iterative process to solve similar equations.

2.9 Summary

This section contained information regarding the concepts of wireless networks. We introduced the components and processes that make up a wireless infrastructure and reviewed the concepts behind time calculation, packet protocols, data fusion algorithms and many others. This source is intended to provide readers with information that help them to understand the concepts of wireless networks, localization, and of the methods of localizing needed to create a wireless localization system. The next section is a combination of several concepts that help to form the design of our wireless communication system.

3 System Design

This section introduces the goal and motivation of our project. Within this chapter we introduce our major objectives and our design requirements. This is followed by a design overview of our system. We then illustrate the integration of our major components and walk through the fundamental procedures required for the development of our localization system. Finally, we take a look at the data analysis tools used to track erroneous conditions, and learn about the error correction methods we applied to enhance our final results.

3.1 Project Motivation and Goal

Localization can be used to detect emitting transmitters that should not be within a network. Therefore, there is a need to overcome important security challenges. For instance, some facilities may be off-limits to most electronics devices, such as unauthorized portable computers and cellular phones. This policy is in place because unapproved devices are typically capable of recording images and audio, and potentially could capture sensitive information. These security breaches are often caused by legitimate laboratory employees and visitors who themselves are allowed within these secure confines, but forget to leave unauthorized devices such as their personal cell phones outside the secure areas [36].

A full resolution to this problem is a difficult and involved task which requires a functional, though not necessarily reliable or secure, wireless localization system incorporating one or more transmitters and multiple receivers. For simplicity in system implementation, we are permitted to assume that the transmitter will be cooperative and able to assist in localization. Our goal is to develop a cyber-physical system consisting of a sensor network, an unauthorized transmitter, and processing software that can determine the transmitter's location and relay this information to the sensor network operator.

This project utilizes open-source software defined radios because of the many strengths behind a coding approach [36]. Software defined radio, as discussed in Section 3.6, implements nearly all radio functionality in software. This approach allows a single radio hardware design to serve many purposes, and facilitates both in-place upgrades of deployed systems and cognitive approaches where radios optimize their configuration using feedback from the environment [16]. When this software and accompanying hardware is fully open source, every piece of functionality can be closely examined for issues such as bugs, limitations, processing delay, and security vulnerabilities. These benefits both facilitate system development and add value to the resulting localization system.

3.2 Objectives

Considering our project's motivation as well as guidance from our sponsor, we created a list of the major objectives for our project. These objectives are presented below:

- Collect time and signal strength data from a transmitting software defined radio using an IEEE 802.11-like protocol with a group of software defined radio sensors.
- Use the collected data to calculate the transmitter's position.
- Display the transmitter's position relative to other objects in the environment.
- Provide tools for calculating and visualizing measurement errors so the system may continue to be improved.

Each of these objectives is fulfilled by one or more pieces of software. Data collection occurs at each sensor and is integrated with the IEEE 802.11b-like receiver. This process is discussed further in Section 3.8. Position calculation is performed in two pieces of software, the real-time localization solver (RTLS) and the post processing localization database (PPLD). The RTLS also displays the transmitter's position within the environment in real time, while the

PPLD is a post-processing tool for calculating and visualizing measurement errors. These software packages are discussed in Section 3.9 and Section 3.10, respectively.

3.3 Design Requirements

In order to complete the project's objectives, we compiled a list of the subtasks that our design must perform in order to meet the objectives. Namely,

- Establish wireless communication between a transmitter and receiver.
- Create a packet protocol for communicating localization information.
- Achieve time synchronization between the sensor nodes.
- Collect and store localization data.
- Understand, calculate, and compensate for known sources of measurement error.
- Find and implement algorithms for finding the distance between each sensor and the transmitter using the collected data.
- Find and implement an algorithm for determining the position of the transmitter given the calculated distances.
- Present the calculated results in a user-friendly format.

The first requirement is to establish wireless communication. At minimum, the system must be capable of successfully communicating a data packet from a transmitter to a compatible receiver. This packet may be any size that is appropriate for our needs, but all of the data transmitted must be received correctly. Keeping the end goal of localizing consumer devices in mind, we were guided towards IEEE 802.11-like communications protocols. After considering the complexity of IEEE 802.11, we opted to reuse and modify existing IEEE 802.11-like

transmitter and receiver implementations instead of creating our own. The full decision-making process is illustrated in Section 3.7.1.

The second objective is to establish a packet protocol for communicating localization information from the transmitter to the receiver. Since RSSI data is provided by the original BBN 802.11b receiver, w determined that the only two pieces of information that needed to be conveyed in the payload are the transmitter's MAC address and the time at which the packet was sent. Both fields are represented in transmitted packets as plain ASCII text. The inherent redundancy of this format is used to perform error detection. A more detailed discussion of this protocol can be found in Section 3.7.2.

The third requirement is sensor synchronization. All of the TDoA algorithms we found in our research require that each sensor shares a common time reference, thereby requiring sensor synchronization. Other algorithms sometimes avoid this requirement, but we determined that they were not appropriate for our application. Sensor synchronization is handled by the Network Time Protocol (NTP) and the Network Time Protocol Daemon (NTPD), which rely on the host's internet connection to adjust the system clock to the correct time. As a result, no synchronization data is carried in the packet. The details of and rationale behind this protocol are discussed in further detail in Section 4.3.

The fourth requirement is to collect and store localization data. Data collection is handled at each sensor by modifications made to existing receiver code. This software extracts the MAC address the time of transmission from the contents of each received packet. Received signal strength indications and times of arrival are collected from the receiver. This information, along with the receiver's location within the environment, is both logged to disk in a computer-readable format and transmitted across a network for real-time processing.

Our fifth requirement is to implement distance calculators using the available ToA and RSSI data. We chose the simple linear and log-linear mathematical techniques discussed in Section 3.8 to perform this task and implemented each in both the RTLS and the PPLD. Both formulas convert the measured data to distance values indicating how far the transmitter is from each particular sensor. Some calibration data is needed beforehand, but this information is easily obtained from simple tests involving one transmitter and one receiver in the deployment environment.

Our sixth requirement is to understand, calculate, and compensate for known sources of measurement error. Each sensor reads in a set of stored environmental parameters from our data files managed by the PPLD, while real-time error correction occurs in the RTLS. Additionally, the PPLD is responsible for visualizing the errors that were not removed during processing so improvements can be made to the processing software.

Our seventh requirement is to implement position finding algorithms. A single position finding algorithm is used for both ToA-based and RSSI-based distance measurements. This algorithm was chosen for its simplicity, and is implemented in both the PPLD and the RTLS.

Our eighth and final requirement is to present the results in a user-friendly form. This is important since the end user must know at a glance where the transmitter is located and how reliable the position estimates are. The RTLS is primarily responsible for visualizing the transmitter's location in the environment, and does so by overlaying the transmitter's position on a user-provided map. Additional information is provided when the user requests it. The PPLD contributes to the fulfillment of this requirement by generating plots and graphs of measurement errors.

3.4 Design Overview

Our design is implemented by four pieces of software that work together to fulfill the project objectives. These components are:

Table 4: Software Components and their Roles

Component	Purpose
Transmitter	Simulate unauthorized users; provide MAC address and transmit time
Sensor (3 needed)	Receive transmissions; capture data for storage and network transmission
RTLS	Real-time data processing; display unauthorized user's location
PPLD	Data quality analysis; enable user-friendly transmitter and sensor configuration

The localization techniques we selected govern the configuration of our transmitter and sensors. First, a distance measurement from each sensor to the transmitter is calculated using TDoA and log-linear path loss. Then, a geometric position calculation uses the distance measurements to find the transmitter's location. This process is discussed in greater detail in Section 3.8. Since three or more sensors are required to perform two-dimensional localization, three or more sensors must be placed at known locations. One additional sensor is used as a mobile transmitter and positioned in several different test locations. Once set up, the transmitter broadcasts packets into the wireless channel at a known frequency. These packets are compliant with the message exchange protocol detailed in Section 3.6.2. Each sensor then receives a copy of this packet and measures its time of arrival and received signal strength indication. Upon collecting this data, each sensor then simultaneously writes a copy of the information to disk and delivers it to the RTLS over a network connection as described in Section 3.7. The configuration of the transmitter is controlled by data files produced by the PPLD. This approach allows for environment-specific settings to be loaded in a user-friendly manner, and is described in greater

detail in Section 3.10. Figure 7 shows this system in a flow graph format, demonstrating how data moves between each node and software package.

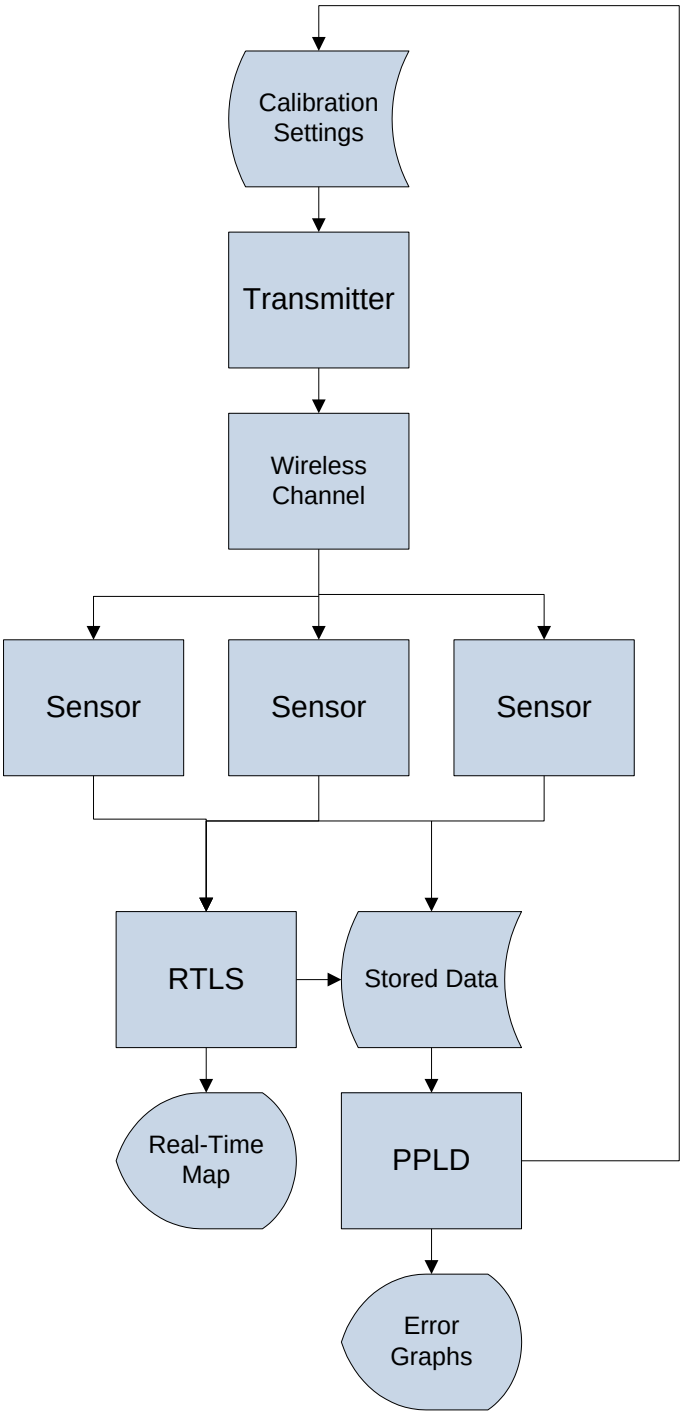


Figure 7: Full system architectural flow graph

The configuration of our data processing and visualization software was governed by the needs of end users at the NRL. Three approaches to presenting localization and data quality information became apparent: post-processing in a data analysis program such as Microsoft Excel or Matlab, real-time output from a custom-implemented program to a text-only terminal, and real-time visualization through a custom-designed program with a graphical user interface. A comparison of each of these approaches is shown in Figure 8.

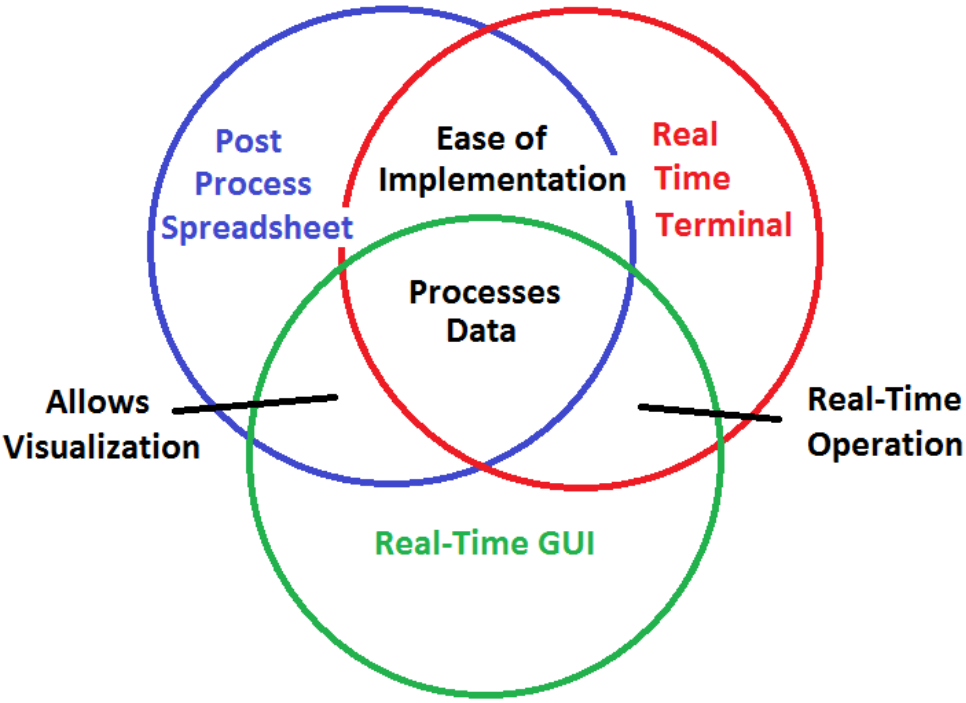


Figure 8: Comparison of localization software realizations

Our final system utilizes all three of these approaches, each for different purposes. Post processing is utilized for our data quality analysis so that system developers can quickly and easily visualize sources of error. This approach manifests itself in the PPLD and is well suited to the needs of system development. System development requires rapid, straightforward implementation of new error correction and visualization tools that present considerable challenges for software that has been custom-implemented in a low-level language. Additionally,

system developers can afford to stop the localization infrastructure and gather stored data files. In contrast, end users view any interruption in system availability as a window of opportunity for security threats to go undetected. For this reason it became clear that end users need a real-time system. We chose to use a GUI as it is a very visual format, allowing security personnel to know where unauthorized users are intuitively, without having to translate numbers into positions within a building. However, a real-time terminal output is provided for debugging the RTLS. This approach is used because text is more conducive to system error logging and a terminal output is immune to problems with the GUI display.

We have developed a higher level system design, made up of five components and several requirements. First we developed the data component which is responsible for storing information via multiple CSV files. Information collected includes raw project data, localization test results, sensor locations, frequency offsets, and calibration data. The remaining four components interact with each other in order to provide the data component with information.

Second we developed our data gathering component. This component consists of the actual communication via the 802.11BBN code, and an error correcting module. Calibration data and frequency offset CSV's provide our modified 802.11BBN code with enhancements to provide initial average error correction to our system. The error correction module uses theoretical lookups and range based analysis of incoming data to categorize the inputs and then add the most likely error to the data.

Third is the localization component that contains a variety of modules. Received in real-time is the project data with a mode identifier, which decides between using RSSI or ToA based distance calculations. After the distance is calculated, we use an aggregator that groups the distances by incoming MAC address. Each MAC address uses the sensor location CSV to reveal

its actual XY coordinates. Another mode is then triggered that distinguishes between using our Python localization or our C localization. The final results are logged and sent to our GUI.

The GUI component is rather small, however helps tremendously. This component contains the actual real time localization mapping and tracking. It also contains a test bench that can manipulate the data within the PPLD to test the effect of different variable parameters such as time difference, or the path loss gradient.

The final component consists of our processing and reporting which contain the PPLD as well as the Excel Localization Test Bench (ELTB). These are quality analysis tools that use our raw project data to ultimately determine the best calibration parameters for our system. In addition, both tools provide detail analysis of the environment in well-organized test environment reports. In the following sections, we go into the procedural details of our system. Figure 9 shows our design components and the relationships between them.

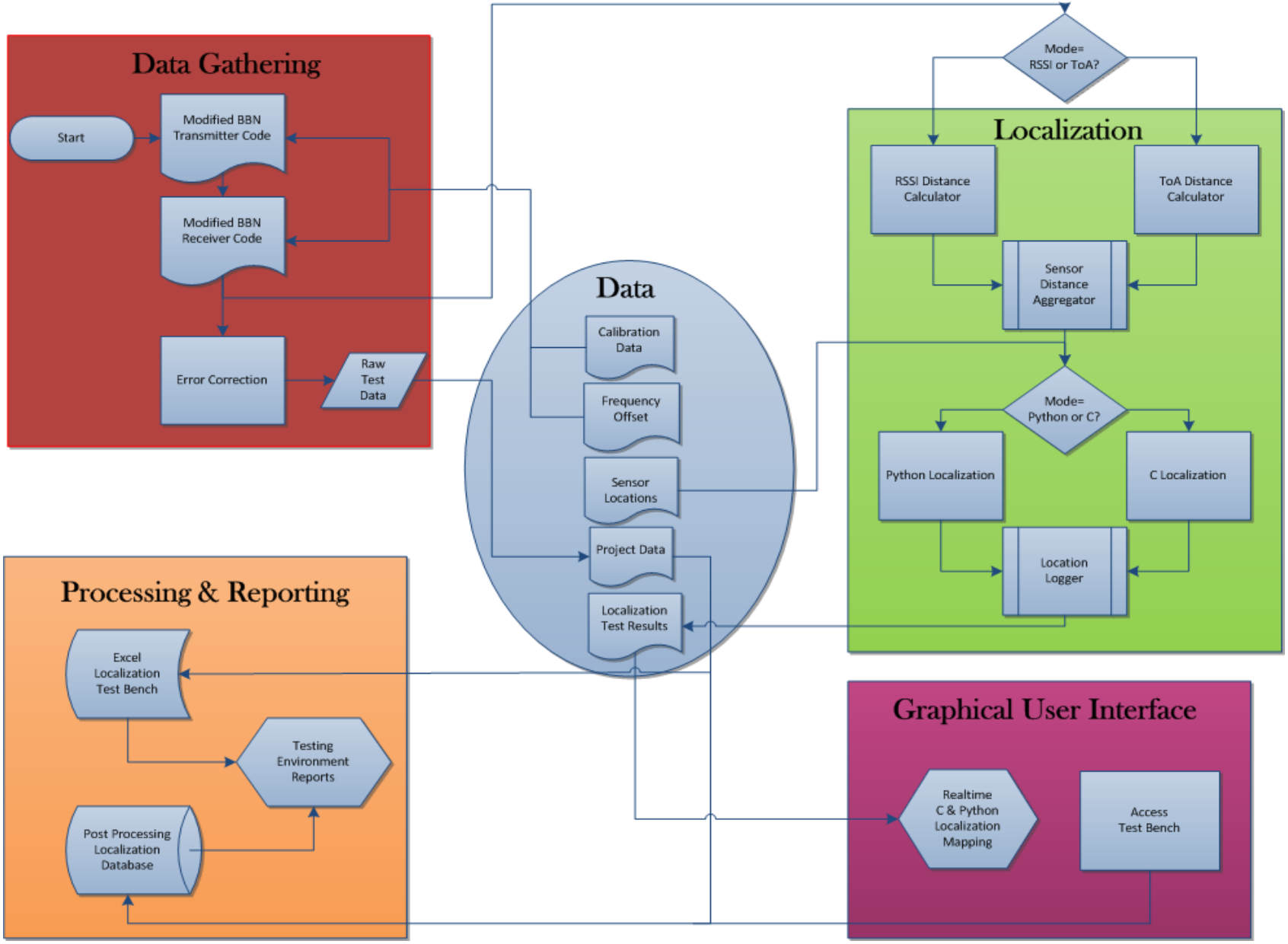


Figure 9: System functionality diagram

3.5 Mathematical Localization Techniques

Many localization techniques exist, but most of these are presented in mathematical form in research papers. In order to perform localization with real data, our chosen techniques must be translated into software. This placed a serious constraint on our chosen techniques, as there is a direct relationship between algorithmic complexity and the time required to produce a software implementation. This was especially challenging as many algorithms are presented in mathematics that are appropriate for the graduate level, often in matrix form so they may scale to arbitrary numbers of nodes.

Many RSSI-based distance calculation algorithms rely on coverage maps of the environment or reverse ray-tracing techniques. While it is possible to obtain a coverage map, a full sensor deployment would be required to update the coverage map every time RF-opaque objects are moved within the area being sensed. This would add considerable complexity to everyday tasks such as moving a filing cabinet, and in practice is unlikely to be implemented as routine procedure. Reverse ray tracing is at the time of writing an active research field. Both the complexity and equipment requirements of reverse ray tracing place it firmly beyond the scope of our project.

Accordingly, we chose to use simplistic and easily implemented methods for our solver software. We decided on the following simple formulas in to find the (x,y) coordinates of the intruder based on the coordinates of the sensors and the calculated distances.

$$x = \frac{r_1^2 - r_2^2 + x_2^2}{2x_2}$$

$$y = \frac{r_1^2 - r_2^2 + x_3^2 + y_3^2}{2y_3} - \frac{x_3 x}{y_3}$$

Equation 9: Localization algorithms to calculate coordinates of intruder

These algorithms will return the coordinates of the intruder where r is the calculated distance from the specified sensor to the intruder, x is the x-coordinate of the specified sensor, and y is the y-coordinate of the specified sensor.

Calculations need three or more distances to determine position. Therefore additional sensors along with some modifications can add new nodes to our calculations thus enhancing accuracy of the system. Future work ought to consider the benefits of combining our research with that of more theory-oriented projects, so the contributions of the latter may be fully realized in the context of our system.

3.5.1 Distance Calculation

The flight of a packet is used to determine the distance between transmitters and receivers. This relies on a time of arrival type approach to determine distance. We will implement our Packet based communication protocol on our system in attempt to synchronize the base stations and mobile station. In order to do so we will have our receiver search the air for any incoming signals, or in other words our receiver will be polling and when a packet is acknowledged the mobile station begins synchronization with the base station.

The receiver will need to receive packets from at least three of the four base stations in order to implement the location algorithm that applies trilateration to find position. The additional base station, if picked up, will add additional data, which should help the accuracy of our measurements. This will be one of the many things we will look into as we begin implementation.

3.5.2 Position Calculation

Determining the position of the unauthorized device is possible once we have determined the three distances from our sensors. The algorithm we have chosen uses the equations for a circle, as well as the distances between each sensor. Using the three equations for each sensor, it is then just a matter of some simple algebra to solve for the two unknowns – the x and y coordinates of the intruder. More on this can be found in Appendix A.

The collected distance values, along with the three sensor positional coordinates, are used to calculate the location of the intruder. All results are stored in a CSV file for later processing via the PPLD. There, we can utilize a test bench, to review and enhance position estimates. Further research will be to experiment with the implementation of different localization methods within our software packages, and compare the results of each method.

3.6 Software-Defined Radio Platform Selection

As discussed in Section 3.1, a software-defined radio environment was needed for implementing our localization and satisfying the project goals. At the time our project was commencing, GNU Radio was the only readily available open-source option for our radio software platform. Since any homemade equivalent would require extensive software development, GNU Radio was selected for use as the basis of our project. A more in-depth discussion of GNU Radio can be found in Section 2.4.3. In retrospect, GNU Radio and more particularly the BBN80211 communications system we used with it proved to be unreliable and poorly documented. Future work that does not require an open-source platform could potentially benefit from utilizing a competing platform, as a commercially-supported competitor was released while this project was underway. At the time of writing, the MathWorks has produced a

functional alternative called the Communications System Toolbox that is fully integrated with their Matlab and Simulink products. However, no 802.11-like communications platforms are known to have been developed for this software environment. This product is discussed further in Section 2.4.4.

When choosing a software-defined radio to work with, we found our choices were rather limited. The radio hardware that was already available to us consisted of USRP2s and USRP software-defined radios, all of which are shared by a large group of students and researchers. The purchase of other radio platforms was beyond the means of our budget, while the development of a custom hardware solution was outside the scope of our project and our skills. Therefore, our choice of radio hardware was limited to USRPs and USRP2s. Table 5 provides a comparison of the two radios, focusing on features that are relevant to this localization project.

Table 5: Comparison of Available SDR Platforms

Feature	**USRP**	**USRP2**
Available Radios	4	14
Receiver Sampling Rate	64 MS/s [18]	100 MS/s [21]
Interface Required	USB 2.0 [18]	Gigabit Ethernet [21]
GNU Radio version*	3.1.1	3.2.2

*The version of GNU Radio required for running BBN80211. Both radios work properly with newer versions of GNU Radio, but BBN80211 currently does not. See Section 2.5 for a more thorough discussion of the BBN80211 platform.

Radio availability was our most important consideration. Conducting this project required two to four radios at any given time. With other students and researchers using the radios from time to time, sufficient availability was not guaranteed. Our budget was not sufficient to purchase multiple additional radios to supplement this supply. The substantially larger supply of USRP2s was a significant factor in weighing which platform to use. Additionally, the USRP2's

higher sampling rate allows higher resolution timestamping. This improvement is even more noticeable when examining the samples at the software host, as Gigabit Ethernet provides much higher data rates than USB 2.0. A heterogeneous system using both radios was considered but encountered problems due to the disparate versions of GNU Radio. As mentioned in Section 2.5, BBN80211 has separate branches to support each radio. The USRP2 branch of BBN80211 has been updated to support GNU Radio 3.2.2, whereas the USRP branch of BBN80211 is currently incompatible with any version newer than GNU Radio 3.1.2. This situation was discovered by the authors while experimenting with the radio platforms. Unfortunately, GNU Radio 3.1.2 has a known issue with Python 2.6 (http://vps.gnuradio.org/redmine/issues/show/312) that was fixed in later releases, but was found to impair installation on our Ubuntu 10.10 platform. Downgrading to an older version of Python or Ubuntu was not viable on shared laboratory computers and porting BBN80211 to a newer version of GNU Radio was not considered an efficient use of project time. For these reasons, we selected the USRP2 for use as our radio platform.

Our project's goal is to build a localization system using specially configured USRP2 transmitters and receivers, while a subsidiary goal is to provide a foundation for future work that will be undertaken to achieve our sponsor's mission. Since the timestamps associated with localization can easily be carried by a 1Mbps or faster communications system, we will attempt to keep our communications protocol implementation as simple as possible without considering the costs in terms of data rates.

3.7 Real-time Communications

In order to replicate unauthorized users employing IEEE 802.11, an IEEE 802.11-like communications protocol was needed. The transmitter provides a MAC address to identify itself as well as a time of transmission, information that must be relayed to the sensor via a communications system and packet protocol. This section details the selection of our communications platform and packet protocol.

3.7.1 Communications System

In order to reduce the implementation time for our project, we opted to utilize an existing IEEE 802.11-like communications platform. Although our project requirements specified that our system needs to be compatible with consumer WiFi devices, they do not require a specific protocol. Additionally, as discussed in Section 3.7.2, our packet protocol does not demand high data rates. These considerations offer a high degree of flexibility in choosing an IEEE 802.11-like platform.

The process of choosing a communications platform is intertwined with the process of choosing a localization algorithm. Any form of localization requires a receiver of some form. Algorithms that require highly precise time of transmit information necessitate specially configured transmitters and receivers, as this information is not customarily provided by IEEE 802.11 (cite IEEE 802.11, see references). Additional benefits to having an SDR transmitter is that we can directly control the time and contents of each transmission, whereas a commercial wireless card integrated into a laptop would be less predictable unless custom drivers were created. Since many wireless drivers and wireless cards are proprietary, reverse engineering one would be difficult at best and result in an implementation specific to hardware that might be discontinued at any time.

Only one of the available IEEE 802.11 implementations discussed in Section 2.5 contains both a transmitter and a receiver and is compatible with the USRP2. This implementation is the USRP2 port of BBN802.11. Testing this system showed that it can properly decode the headers of IEEE 802.11 packets being transmitted across WPI's campus wireless network, though the contents were unreadable as they were encrypted and often transmitted at unsupported IEEE 802.11g/n data rates. We elected to use the 1Mbps data rate setting to ensure robustness, as higher data rates were not required for our message exchange protocol.

Compatibility issues have arisen between newer releases of GNU Radio and the BBN codebase. We were unable to get the code functioning with version 3.3.0 of GNU Radio, but succeeded in making it work with version 3.2.2 and GCC 4.4.5 with only minor modifications. Although we modified the BBN802.11 code during this project so it would log packet data, as discussed in Section 3.8, the underlying communications system is functionally unchanged.

3.7.2 Message Exchange Protocol

Implementation of a message exchange protocol is essential for enabling appropriate data collection. In order for received signal data to be collected at each sensor, some form of packet must be transmitted by the unauthorized user. However, a cooperative transmitter could provide additional benefits by simplifying or improving the robustness of our localization algorithm. Choosing our DToA approach requires the transmitter to produce a time of transmission but avoids the need for bidirectional communications. Additionally, the transmitter needs to provide some form of identifying information to distinguish it from other signal sources found in an indoor environment. Since IEEE 802.11 uses MAC addresses to identify senders, the MAC address format was selected as a form of transmitter identification [33].

Having determined that the data the transmitter needs to provide is its MAC address and the time of transmission, the next step is to design a message exchange protocol to ensure this information is correctly received and understood at each sensor. The design goals for this protocol were as follows:

- Ability to carry MAC addresses and timestamps
- Size of the packet in bytes
- Robustness against bit errors introduced by the channel
- Implementation complexity

Two options presented themselves for each item of data into each packet. The data could be sent in a serialized network-byte-order form, or converted to human-readable ASCII before transmission. The advantages of transmitting in raw ASCII format include ease of debugging and the ability to use sanity checks on each decoded character to detect bit errors. Only characters that could represent hexadecimal numbers may appear in ASCII-format MAC addresses, while any number may appear in a standard MAC address. The advantages of transmitting the raw data are a smaller packet size from elimination of redundancy and similarity with real network protocols such as IEEE 802.11. Disadvantages include reduced ease of debugging and the need for separate error detection or correction.

Three potential protocols were considered, although other protocols could be developed. Some of these protocols require the MAC addresses and timestamps to be implemented in numerical form, while others require them in ASCII form. Numerical MAC addresses require 6 bytes while ASCII MAC addresses that include the colons require 17 bytes. Numerical timestamps require at least 4 bytes to represent the seconds that have passed since the Unix epoch, and at least 4 more bytes to represent the nanoseconds. However, numerical timestamps

that allow for fractions of nanoseconds, and years greater than 2038, are implemented using 64-bit fields, and use as much as 16 bytes total. Our ASCII timestamps print the time at which our experiments took place and require 20 bytes for single-nanosecond precision, including the decimal point.

1) IEEE 802.11 Header with Timestamp

Table 6: IEEE 802.11-based Message Exchange Protocol

Frame Control (2 bytes)	Duration / ID (2 bytes)	Address 1 (6 bytes)	Address 2 (6bytes)	Address 3 (6bytes)	Sequence Control (2 bytes)	Address 4 (6 bytes)	Numerical Timestamp (16 bytes)	CRC (4 bytes)

This protocol has the advantages of being IEEE 802.11 compliant and incorporating error checking, but is very large given the limited quantity of data being transmitted. Due to the frequency drift endemic to USRP2s and the lack of automatic frequency offset compensation in BBN80211, the reliable and complete transmission of a 46-byte header at a low data rate could not be guaranteed.

2) Bare-Minimum Packet Size

Cutting out data fields not required by the project greatly reduces the probability of a bit error occurring in the data fields but does not eliminate it completely. The following protocol also reduces the size of the timestamp, reducing future-proofing in the interest of saving space:

Table 7: Minimal Message Exchange Protocol Format

Numerical MAC Address (6 bytes)	Abbreviated Timestamp (8 bytes)	CRC (4 bytes)

This protocol, though more efficient, still requires the use of a CRC and makes debugging complex. Text files created with unmodified numerical data are unreadable in many text editors. This situation complicated the process of verifying that BBN80211 was functioning

properly. As a result, we decided to use ASCII formatted data fields despite their size penalty. This also allowed us to move away from CRC error checking and towards a technique which could be easily verified by human programmers. The resulting packet format is shown in Table 8.

Table 8: Final Message Exchange Protocol Format

MAC address (17 bytes) ASCII	Timestamp (20 bytes) ASCII

Though this format is nearly as large as the original IEEE 802.11-based protocol, it greatly simplifies the process of determining whether or not BBN802.11 and the error checking features are functioning correctly. However, ASCII-based error correction is not as robust as CRC checks and consumes considerably more space in the packet. Future work may benefit from reconsidering other protocol formats after the reliability of BBN80211 has been improved.

The selected design encodes both the MAC address and timestamp as plain ASCII text. The MAC address aa:bb:cc:dd:ee:ff is represented in the packet as AA:BB:CC:DD:EE:FF, although the protocol is not case sensitive. When bit errors occur in the MAC address, there is a high probability that the results will not form a valid ASCII MAC address. When ASCII text such as the letter ‘Q’ that cannot be used to represent a hexadecimal value is found in the MAC address field, the sensor knows that bit errors occurred and drops the packet. This technique proved to be very helpful when using BBN80211, as our packet protocol does not implement explicit error detection or correction. The time of transmission timestamp is also represented as ASCII text. The timestamp itself appears as a floating point number of seconds that have passed since the Unix epoch, with 9 decimal places and the decimal point included. This format consumes a total of 20 bytes and is trivially created by Python-based software. Error checking is

also performed on the timestamp, and the packet is dropped if invalid ASCII characters appear or the decimal point is missing.

3.8 Data Collection

Our data collection system consists of a sensor node equipped with a modified version of the BBN80211 receiver. Each sensor is capable of reading packets sent using our packet protocol and collecting data both from the arriving signal and the packet contents. Localization data is collected within the BBN receiver. At first, we modified the receiver callback function where packet contents are made available to the end users of BBN80211 to print out the calculated RSSI value as well as the time when the function is called. This performed as expected, though there was considerable room for improvement. Since the callback function is the very last functional block to process the received packet data, it is the most sensitive to variable processing delays. Additionally, the RSSI values only showed two significant digits of precision, resulting in very little variation in the measurements.

Table 9: Comparison of Localization Data Collection Point

	PLCP block	Callback block
Language	C++	Python
Easier to program for	C programmers	Object oriented programmers
RSSI precision	6 significant figures	2 significant figures
Bit count	Readily available	Not readily available
Runtime performance	Fast	Moderate

Due to this need for improvement, we chose to move our data collection and timestamping to a lower level component of BBN80211. Our final design modifies Physical Layer Convergence Protocol (PLCP) block, as this is the first block in the receiver flowgraph where all of the essential data is present and has been decoded. The PLCP block's internal RSSI

data has six significant figures, a considerable improvement over the two significant figures provided in the callback function. Another advantage of modifying the PLCP block is that it is implemented in C++, allowing for higher runtime performance.

Table 10 shows the pieces of data that are extracted from the PLCP block.

Table 10: Data gathered by the sensors

Data	Extracted From:
Transmitter MAC address	Packet contents
Time of Transmission	Packet contents
Time of Arrival	System clock
Received Signal Strength Indication	PLCP block internal data
Sensor location	Sensor parameters

These data fields are used by the RTLS and PPLD. Each sensor is responsible for ensuring that the data is accessible to these localization tools, so C++ code is provided for simultaneously transmitting the data over a network and logging it to disk. Both computer-readable and human-readable files are created. Blocking I/O operations take a very long period of time relative to most other tasks a computer performs. It is essential to ensure that this constraint does not interfere with the PLCP layer's processing of arriving USRP2 samples. Therefore, two additional threads of execution are created in addition to the PLCP block so network transmission, file operations, and PLCP processing can occur simultaneously. The design decisions leading to this implementation are presented in more detail in Section 3.8.4.

3.8.1 Sensor and Transmitter Configuration

The transmitter and sensors are run by executing a Python script that configures and executes a GNU Radio flow graph. When this code is run, program parameters are read in from a

series of five data files. These files are labeled calibration data, frequency offset, sensor locations, project data, and localization test results. Each computer file is created and managed by the PPLD, allowing the user to easily save settings for each test configuration or environment the system is operated in. Table 11 describes the contents of each computer file.

Table 11: Transmitter and Sensor Configuration Parameters

Computer File Name	**Description**
Calibration Data	Contains ToA and RSS calibration data grouped by environment ID.
Frequency Offset	Contains BSSID's along with individual frequency offsets.
Sensor Locations	Contains Sensor XYZ locations grouped by BSSID's.
Project Data	Contains raw data from every test ever conducted.
Localization Test Results	Tracks XYZ location of an unauthorized device by project ID.

Our design uses these computer files to provide an intuitive, PPLD-driven method of configuring the sensors. As a result, an iterative improvement process is facilitated as shown in Figure 10.

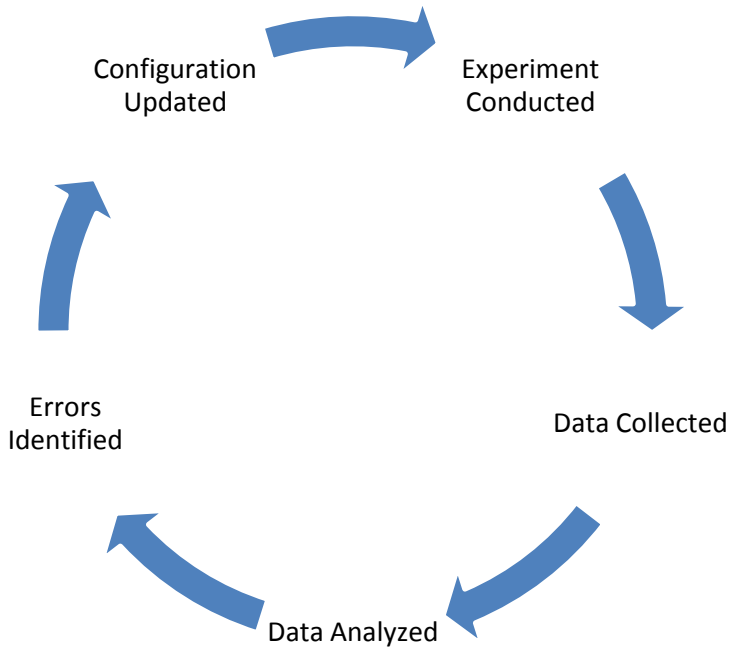


Figure 10: The iterative design process enabled by the PPLD

This integration between the transmitter and sensor parameters and our data quality analysis tools adds value to the project helping future work focus on the localization data rather than the nuances of the underlying software packages.

3.8.2 Time Synchronization

In order to use timestamps for localization, the clocks used for timestamping must share a common reference. Unsynchronized clocks will drift due to small but significant errors in the oscillator clock frequency. Accurate timestamps must be sent so we can calculate the distance correctly. In order to establish time synchronization between our units we decided to look into Network Time Protocols so that we can use a server's atomic clock, similar to what GPS currently does in order to sync computers.

One potential approach is to use a DTDoA communication protocol between sensor nodes to establish and update the clock offset between each node. DTDoA, as discussed in Section 2.3.3, achieves this by sending a packet round-trip between each pair of sensors. The difference in the recorded flight time for each trip is used to calculate the offset. This approach has the advantage that synchronization can be achieved with a level of accuracy limited only by the radio hardware and timestamp calculation methods employed. However, there is a notable disadvantage in that the channel being sensed must be used for overhead packets. Since our radio platform only provides half-duplex capability, each sensor will be inoperable while transmitting. These considerations, combined with the enormous difficulty BBN80211 has in successfully transmitting packets, strongly encouraged us to seek an alternative solution.

Since the BBN code struggles to provide successful communications, we opted to use an approach to time synchronization that does not rely on successful full-duplex communication. As each sensor is in a fixed location indoors, it is not unreasonable to expect that they will each have a connection to a static network such as a wired LAN. An existing protocol called Network Time Protocol (NTP) is used to synchronize servers to atomic clocks over such a network. NTP customarily checks the atomic clock's time once per day and corrects the server's time accordingly. This, by itself, is not enough to achieve an appropriate level of synchronization as the computer's clock may drift considerably over the course of a day. Linux offers another tool for improving this synchronization called Network Time Protocol Daemon (NTPD). The NTPD is a daemon that runs in the background and continuously corrects this drift in the clock. This behavior is precisely what is needed for ensuring time synchronization across all three sensors.

This discovery proffers two new approaches to time synchronization. Firstly, we could employ NTP and NTPD directly as implemented in Ubuntu Linux. Secondly, we could examine

the open-source codebases for these features and use them as a guide for implementing our own time synchronization method. Further examination concluded that both features are highly complicated—for example, NTPD must ensure that other processes that rely on the system clock are not affected by corrections to the system time. Therefore we concluded that it would be more appropriate to use these protocols as-is, rather than budgeting large quantities of time to improving these already-functional protocols. Given the importance of time synchronization to time-based localization, this remains a potential source of improvements to our system.

3.8.3 Data Format

Localization data is stored in a computer-readable file, utilizing the Comma-Separated Values (CSV) file format. This format is simple and uses a combination of commas, semicolons, and newlines to separate different pieces of data within a text file. The simplicity of this format has led to it being compatible not only with the code we have implemented but programs such as Microsoft Excel. As a result, CSV files are used not only for configuring the transmitter but for storing the localization data. This configuration was described in more detail in Section 3.8.1.

During the data collection process, two logged copies of the data are created. One copy is in CSV format and is intended for use with the PPLD. The other is a text (.txt) format intended to be easily read by humans. The PPLD field names of the raw data which print from the PLCP layer can be found in Table 12.

Table 12: Raw data fields

Field Names	Field Descriptions
Mac_Address	Base Station Identifier
Time_Sent_sec	Time Sent (seconds)
Time_Sent_ns	Time Sent (nanoseconds)
Time_Rcvd_sec	Time Received (seconds)
Time_Rcvd_ns	Time Received (nanoseconds)
RSSI_dbm	Received Signal Strength Indicator
Sensor_X_location	X coordinate on provided map
Sensor_Y_location	Y coordinate on provided map
Sensor_Z_location	Z coordinate on provided map

The MAC address is a C++ class encapsulating an integer, and is used to separately identify multiple transmitters as well as environmental packets that aren't being localized. It also provides functionality for checking incoming packets for bit errors as described in Section 3.7.2. The time of transmission and time received are represented using two 64-bit data values, and correspond to the time that has passed since the Unix epoch. By representing the number of nanoseconds as a 64-bit floating point value, fractions of nanoseconds can easily be stored and mathematical calculations are able to avoid losing precision. This ability to retain full precision during calculations outweighs the performance advantages of using a smaller fixed-point integer value. The RSSI is a 32-bit floating point number using the BBN code's own RSSI format, and is not calibrated to any particular reference value. Localization software is responsible for performing this calibration using known information about the transmitter or prior test results. The location of the sensor consists of floating point coordinates representing the sensor's position, in meters, relative to an arbitrary point (0, 0). The Z value is reserved for future use with three-dimensional localization algorithms, but can also be used to represent a known difference between the transmitter's elevation and the sensor's elevation.

3.8.4 Efficiently Logging Data in a Real-Time Environment

One fundamental implementation challenge, given our sensor's need for real-time performance, is effectively handling some form of data output. Each available technique for implementing our data processing requires a different kind of output, as shown below:

Table 13: I/O requirements

Approach	I/O operations needed
Display all data using a terminal	Print to terminal (may be redirected to a file output)
Write data to file	File output
Transmit data over a network	Network output

None of these options are able to guarantee that long delays are avoided. Even the fastest hard drives are many orders of magnitude slower than a computer processor, while networks potentially introduce even longer delays. This delay is highly variable and depends on whether or not the output resource is available when the I/O function call occurs. As there are no high-performance options here, we opted to find an alternate solution to the real-time I/O problem and choose an approach based on other considerations. Terminal output was designated for problem reporting and debugging, as information logged there is most easily accessible for troubleshooting and diagnostics. File I/O was selected as it allows data processing to occur outside of real time, whereas network I/O could be used in the future for forwarding data to a centralized collection point for real-time processing.

Several approaches were considered and compared to handle writing raw sensor data to log files.

- Synchronous blocking I/O in the primary execution thread.
- Synchronous blocking I/O in a secondary helper thread.

- Asynchronous non-blocking I/O using *asio.*

The performance constraints on blocking I/O are unacceptably time-consuming for implementation in the primary execution thread. We assumed that the PLCP block already demands a lot of resources, so the combined load is assumed to be outside real-time operation. Blocking I/O in a helper thread allows the host OS to schedule file output in parallel with PLCP execution, but only permits one such operation to happen at the same time. This problem could be partially alleviated by using multiple helper threads.

Asynchronous I/O using an API, such as *asio*, introduces a different set of tradeoffs. The underlying implementation can vary, as can its performance characteristics. Advanced API implementations, such as OS-level support for this feature, offer potential performance benefits that would be time-consuming or impossible to realize with a handmade solution. However, we were not able to locate an API with our logger's desired behavior: a function that is called once, takes ownership of a copy of the packet data, and writes it to disk without any further complications. *Asio* and the related *boost::asio* APIs require a callback function that is used once the output operation completes. Considering that callback function would, at most, delete extraneous copies of the data we felt the signaling overhead introduced by this approach was at best undesirable.

Additionally, the underlying implementation of an asynchronous I/O API could potentially reduce performance. Every thread that is created incurs some overhead, while the benefits that are realized from multithreading vary based on the design of and load on the computer's processor. By opening Ubuntu's System Monitor while the BBN receiver is running, we determined that GNU Radio creates large numbers of threads and is capable of fully utilizing dual and quad-core processors. Any additional threads that wish to execute must share processor

cores with GNU Radio, thereby introducing more overhead as the OS performs context switches. Thread-intensive asynchronous I/O implementations potentially compound this problem, especially on computers with fewer processor cores or that cannot benefit from features such as hyperthreading. For these two reasons, we elected to create a custom I/O solution. The option we chose is to have a dedicated logging thread serially output the packet data using blocking I/O. Although only one thread is currently used to handle output to both log files, the system could easily be reconfigured to use separate threads for each desired file.

3.9 Real-Time Processing and Display

After our data has been successfully collected, we process it with an automated localization solver to produce transmitter-to-sensor distances, a position estimate for the transmitter, as well as an output for the end user. To do this in real time, we created a software package known as the Real Time Localization Solver, or RTLS. The RTLS is an example of and foundation for a security-oriented localization system targeted at end users. It contains real-time localization data processing as well as a continuously updating map of the environment. This map contains the locations of the sensors and any unauthorized transmitters that have been detected. Menus are provided as part of a complete user interface for configuring the real-time localization system. A picture of this interface is shown in Figure 11.

Figure 11: RTLS graphical user interface

The blue triangles represent the sensors while the red circle represents the unauthorized device. The yellow dot in the center of the circle indicates that the unauthorized device is stationary. If the device was moving, the simulator would show an arrow indicating the current direction of travel. A map of a sample environment is provided and overlaid with the sensors and unauthorized devices. Note that a "Help" dropdown is provided within the menu at the top left. One of the items in this menu is a "What Next?" command, which can be used to determine what action should be performed next to configure and run the simulation.

The coordinate system used during localization is hidden from the viewer, though the point (0, 0) is currently designated to be the top left corner of the map. Sensors are depicted as blue triangles while unauthorized transmitters are depicted as a red circles with an arrow indicating direction of travel. The unauthorized user appears to move with time and its path is tracked on the screen.

3.9.1 Real-Time Simulator

A corresponding simulator was created to test and demonstrate the RTLS. This simulator is provided as a development tool for testing the solver. Simulated sensor data has a known, controllable quantity of error whereas actual sensor data is subject to many sometimes unknown sources of error. A similar user interface showing a map of the environment is used in the simulator. As seen in Figure 12, the user interface is nearly identical to that of the RTLS. This similarityThis approach makes comparing the RTLS results to the simulated environment exceptionally straightforward.

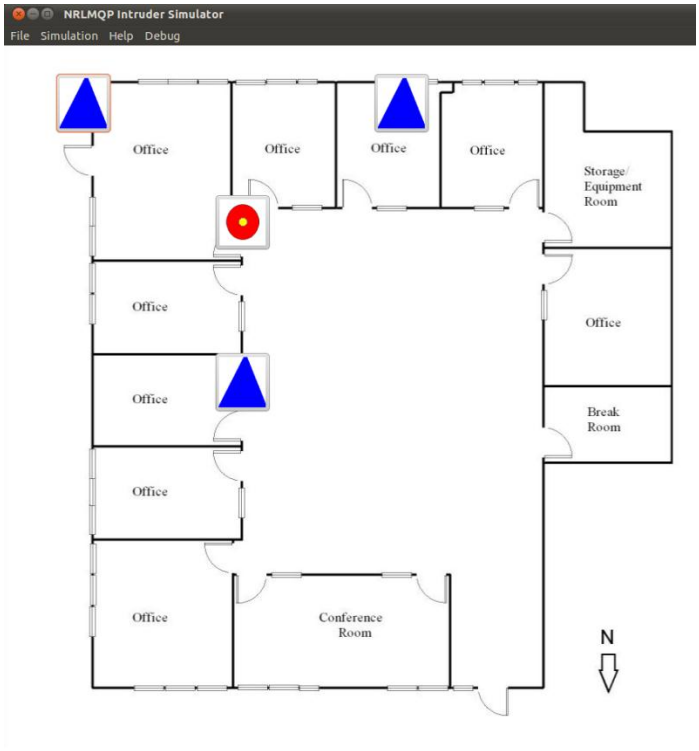


Figure 12: Real-Time Simulator GUI.
Note its many similarities with the real-time solver.

3.9.2 RTLS Specifications, Features and Structure

The RTLS and its simulator are written in C++, using wxWidgets 2.8 to provide GUI functionality. It has been developed for Ubuntu 10.10 64-bit, though nearly all of the code is system-independent and easily portable to other major desktop platforms. A script called "makenmove" is provided to compile both the RTLS and its simulator, automatically producing executable binaries for both components. This script also prepares the user's copy of BBN80211 for integration with the sensor code, though the user then has to configure and install BBN80211 as specified by that software package's README. When each program is launched, the user is presented with the main GUI screen. Unfamiliar users are advised to use the built-in Help, particularly the "What Next?" feature.

Our source code is divided up into five major groups, as described below. Header files are kept separate from implementations to reduce the clutter in each directory. Additional folders are provided as necessary for images such as the environment map and object files used in compilation.

- "Sensor" files are exclusively used with BBN80211 at each RTLS-compatible sensor.
- "Simulator" files are exclusive to the provided simulator.
- "Solver" files are exclusive to the RTLS.
- "Shared" files are common to both the RTLS and the simulator.
- "Common" files are used by all three components.

This architecture organizes the source code in a manageable fashion. Each major class receives its own header and implementation file, while relatively small and closely related classes often share files. The name of each file reflects the class or classes contained within as well as the functional purpose that it serves. Copies of BBN80211 files are kept in the sensor directory only if they have been modified.

Figure 13 shows how data flows into and through the RTLS.

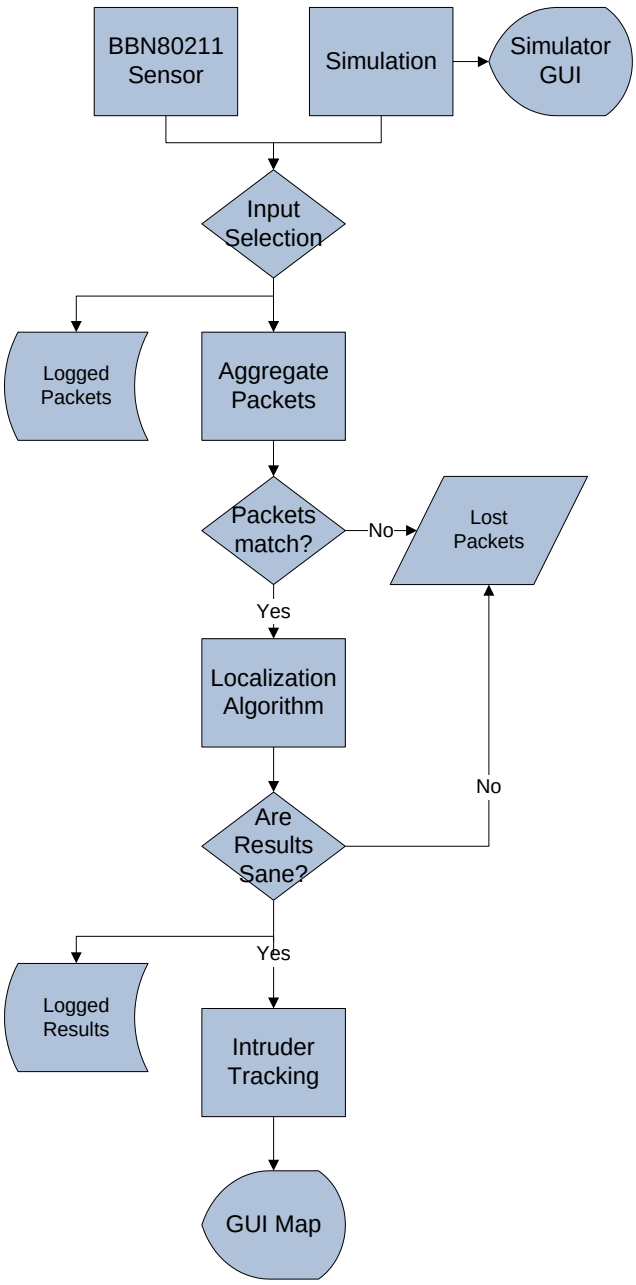


Figure 13: RTLS Flow Chart

Data is fed into the RTLS from the sensor or the simulator. The sensor code extracts packet information from the BBN80211 PLCP layer, while the simulator periodically generates packets using user-specified locations for the sensors and unauthorized users. Users are able to switch between simulated data and sensor data through a socket-based interface. Each sensor data stream is handled by a TCP connection, transferring a standard "packet" data structure as a

serialized array. Each sensor forms a connection with the RTLS when initialized, while the simulator creates three such connections when told to by the user. This modular system places the RTLS in an independent process that can be run on any machine with a network connection to the incoming data stream, regardless of whether or not that machine is being used as a sensor or simulator. TCP provides a lossless and easy to troubleshoot connection, though a different network protocol could be considered for the future.

We also considered using the wireless network set up by the USRP2s to transfer data from each sensor to the RTLS, but determined that BBN80211 isn't a simple or reliable enough network interface for this to be reasonable. These reliability concerns stem from the fact that BBN 80211 lacks automatic frequency offset compensation, resulting in bit errors and lost transmissions when the USRP2 clock frequencies periodically drift. A socket-based approach also provides considerable future-proofing, as the interface between modules is simple and easy to replicate. Future work could implement software for playing back saved experimental data to the RTLS or use BBN80211 as a LAN without modifying the RTLS so long as they use the same socket-based interface.

Upon arrival, the packet data is simultaneously written to disk and placed into a buffer until a timeout occurs. The buffer is structured as a heap with the oldest packet at the root. Packets are ordered using the time at which they arrived at the sensor and timeouts occur one second after the time of arrival. This method allows for proper aggregation regardless of network delay or the order in which the packets arrived. Since transmissions arrive at each sensor within tens of nanoseconds of each other, it is assumed that all of the packets generated by each transmission will expire simultaneously. Packets that share the same MAC address, are generated

at different sensor locations, and arrived within 100 nanoseconds of each other are considered matches.

The localization algorithm is a modular component consisting of a "framework" and one or more "models." Frameworks are C++ wrapper classes that encapsulate a particular solver architecture, while models are individual localization algorithm components. The current RTLS implements one framework called FiveStepFramework, which subdivides localization into the following tasks:

1) Calculate distance using the transmit and receive timestamps.
2) Calculate distance using RSSI, a known RSSI at some known distance and a path loss model.
3) Find location of the unauthorized device using time-based distance calculations.
4) Find location of the unauthorized device using RSSI distance calculations.
5) Select one of the two positions for output.

Each framework is required to have a function called "localize()", which accepts a measurement containing a group of aggregated packets and adds both a calculated position and an uncertainty to the measurement. Future work is encouraged to use this simple interface for implementing error correction, as well as more advanced localization algorithms drawn from cutting-edge research in wireless localization theory.

For each step required by our chosen framework, we implement both a model specifying what is required to complete that step and an example algorithm for performing it. Time-based distance calculations are handled by a class called LightspeedToA, which produces distance information.

The time offset is a constant, user-specified value for the radio hardware processing delay. Since all of the sensors are USRP2s, it can be assumed that this processing delay will be almost identical for each sensor. The speed of light is also user-specified, though the speed of light in free space is used by default.

A single position calculator called SimpleGeometricLocalizer is employed for both types of position calculations. This class implements several sanity checks to ensure that erroneous results are discarded. Although this algorithm does not produce an uncertainty estimate, the model provided to encapsulate position calculation does. Future work could easily incorporate uncertainty estimation, as all of the functionality needed to process and display it already exists in the RTLS.

With two position data streams available, a technique must be employed to select which data stream is more accurate. FiveStepFramework implements a sanity check to ensure useable data is being passed along. If neither position calculation succeeded in producing valid results, a non-fatal error is thrown to inform the user of this occurrence. If one or both sets of calculations returned erroneous or nonexistent results, the valid data stream is used for the unauthorized device's location. If both data streams contain valid data, a selection algorithm is called to figure out which data set to use. A simplistic algorithm is provided in the form of a class known as ShortestDistanceChooser. ShortestDistanceChooser will return whichever position estimate has the least distance to any one sensor. This is intended to avoid large position errors that place the unauthorized device far away from the sensor grid but provide little utility when both position estimates are reasonably accurate. The results that are kept are written to disk and passed along to the intruder tracker.

The intruder tracker manages the intruder objects that indicate where unauthorized devices are located in the environment. It treats the localization measurements as periodic updates, determines whether the measurements' MAC addresses correspond to newly identified unauthorized devices or those that are already in the environment. If an unauthorized device has already been found and an update for it is received, the intruder tracker can update the GUI with its current direction of travel. If an existing device is no longer detected, it will continue to appear for a user-specified length of time or until the same device reappears. This "coasting" behavior is intended to help with the localization of transmitters that are temporarily silent, but may result in out-of-date information being displayed. The coasting period is user-specified due to this tradeoff.

3.9.3 RTLS Calibration

The RTLS can be easily calibrated using simulated or sensor data. This calibration process is necessary for determining the RSSI at a known distance and the processing delay encountered by time of arrival measurements. Calibration is performed by providing the solver with the real location of an unauthorized user, referred to in the dialog as an "Intruder" for conciseness and consistency with the source code, as well as a MAC address for identifying that user. This dialog is shown in Figure 14 below.

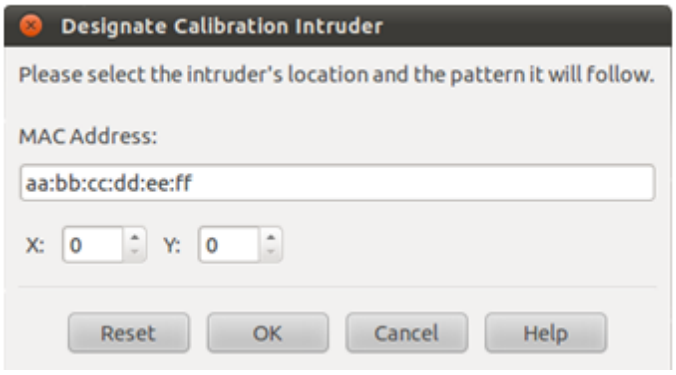


Figure 14: RTLS calibration dialog

A different approach that was considered for addressing calibration was to flag arriving data packets as "calibration packets" before they were transmitted to the RTLS. The user would configure each sensor or the simulator to mark outbound packets for calibration before each experiment that is performed for calibration, providing both the real distance between the transmitter and sensor as well as a "flag" indicating this data is present. This approach was not chosen because it is more complicated both to implement and for the user, but would not actually improve calibration results. The chosen approach is very simple to use and to implement, but adds a small amount of processing overhead to packets being processed by the RTLS. This overhead occurs because each arriving packet is checked against the list of calibration MAC addresses and "flagged" if it originated from one of them.

3.9.4 Design Decisions

When we identified a need for a GUI-based localization solver capable of visualizing the environment, we discussed various possible specifications for the RTLS. The software needed to be capable of receiving data from both USRP2-based sensors and a simulator, displaying the information received to a user interface, and provide a map of the environment overlaid with the positions of both our sensors and the unauthorized user(s) being localized. Real-time operation was considered highly desirable. These requirements were then translated into a series of design decisions and eventually the finished RTLS.

First, we chose to develop the RTLS for the Ubuntu Linux platform. This was primarily motivated by the fact that our USRP2-based sensors operate on Linux and porting the necessary sensor code to another platform is unlikely to be a trivial task. While GNU Radio has been tested on other platforms, BBN80211 has not as of the time of writing. Effectively and reliably porting both codebases to a non-Linux environment is a delicate task that would take time away from

more important parts of our project. At minimum, the interface between the RTLS and BBN80211 needed to be implemented in Linux. Another advantage of the Linux platform is that it can be installed on virtually any computer without the need for software licenses. For these reasons, we elected to build the entire RTLS for the Linux platform. Ubuntu 10.10 was selected as the Linux distribution used for development due to its relatively high compatibility with GNU Radio and existing deployment on the available computers.

At first, we did not realize how profoundly our choice of a GUI platform would impact the structure and content of the RTLS' code. We began writing RTLS code in a mixture of C and C++, choosing to remain with familiar C programming while occasionally exploiting a more advanced C++ feature. After some initial testing, we began to incorporate the GUI and then realized that our code would need to be restructured. Accordingly, our design decisions are presented in the order in which they were made rather than in the order they should have been made.

We reassessed our choice of programming language upon realizing the significance of our GUI platform. Several programming languages were considered for the RTLS. Programmer familiarity and C compatibility were important concerns as the time and effort required to learn a new programming language or programming paradigm can be non-trivial. At this point, we already had some investment in C/C++ but were willing to work in a compatible environment if the benefits outweighed the costs. We also considered performance, scalability, implementation time, and compatibility with our Linux platform. Ubuntu Linux supports most popular programming languages, as do most mature desktop platforms.

GNU Radio is implemented in a combination of Python and C++, object-oriented languages that are within relatively easy reach of a C programmer. As a result, our comparison focused on C, C++, and Python as candidate programming languages.

Table 14: Comparison of RTLS Programming Languages

Language	**C**	**C++**	**Python**
Our Familiarity	Excellent	Good	Fair
C/C++ compatibility	Excellent	Excellent	Fair*
GUI platform compatibility	Limited**	Good	Good
Runtime Performance	Excellent	Excellent	Fair
Built-In Features	Fair	Excellent	Excellent
Ubuntu support	Excellent	Excellent	Excellent
Scalability	Fair	Excellent	Good
Implementation time	Poor	Fair	Good
Educational value	N/A	Excellent	Good

*Python code can treat C and C++ code as custom-implemented modules. This can be done by manually writing Python-compliant C/C++ code or using a software package called SWIG. We tested SWIG, and while it functioned correctly it proved to be weakly documented and had a noticeable learning curve.

**C code can be used within C++ applications, meaning a nominally C++ codebase can almost entirely be written as though it were C code.

We chose C++ as it offers the best combination of all these considerations. Python uses a different syntax and structure than C and therefore requires special techniques to reuse existing code, while extensive use of C would greatly limit our ability to employ time-saving features and

advanced user interface libraries. Our limited familiarity with the object-oriented programming paradigm was compensated for by the significant improvement in scalability as well as the learning experience that would be provided. Transitioning from C to C++ could occur gradually and coincide with productive software development. Additionally, writing the software in a language known for good runtime performance makes the RTLS easier to transform into a final product. While performance was not critical for our prototype, runtime performance is directly related to the ability for a final product to handle large and active environments. Future work is likely to scale the RTLS beyond three sensors and one unauthorized user. This would generate much more data to be processed, making runtime performance and algorithmic efficiency important future-proofing considerations.

As the project continued to progress, our code evolved from a very C-like style to a predominantly object oriented approach. Using C++'s classes to encapsulate functionality proved very beneficial. This change provided many advantages especially as the code grew and reached many thousands of lines of source code., as C++'s classes make it much easier to encapsulate functionality. This is what is referred to by scalability and directly affects implementation time. Additionally, C++ provides built-in support for variable-size arrays, queues, and heaps. Learning to use these features proved enormously beneficial. The final RTLS makes extensive use of C++ features and the process of transitioning to C++ from C proved to be a valuable educational experience.

As we were selecting a programming language to use, we were concurrently searching for a GUI platform to use for the RTLS. This critical software dependency had to be chosen carefully, as further investigation showed how the API selected dictates the syntax and structure of all of our GUI-related code. The API chosen had to be compatible with a Linux platform,

readily available, and capable of producing GPL-compliant software. GPL-compliance is critical as GNU Radio and BBN80211 are both licensed under the GPL, a license for open-source software that is inherited by all derivative works. Additionally, NRL specified that the results of our project be freely available under the GPL. Simplicity, user-friendliness, documentation and cross-platform compatibility were also desired features. C, C++, or Python compatibility needed to be present and mature although any combination of the three could be supported. C++ compatibility in particular became increasingly desirable as we weighed the available programming languages.

There was considerable discrepancy as to whether the RTLS has more in common with real-time games or traditional user interfaces. Video game oriented libraries such as Allegro and ClanLib offer extensive support for constantly updating interactive environments, but little or no support for GUI features such as menus and message boxes. GUI libraries such as wxWidgets offer extensive GUI support but are typically not designed to support frequently updating on-screen objects.

We chose to use a GUI library instead of a game library, reasoning that it would be easier to make real-time objects in a GUI than menus in a video game. In contrast, no clear and simple method of creating menus presented itself when looking at freely-available Linux-compatible video game libraries. Implementation time was a key consideration, as no team member had extensive experience with any particular game or GUI library and therefore considerable time would be spent learning the basic API.

Of the GUI libraries, wxWidgets emerged as a strong candidate for many reasons. It provides highly used C++ and Python support, meaning that we would not be substantially limiting our choice of a programming language. Very importantly, most wxWidgets code is open

source and licensed under a variant of the LGPL that permits commercial applications. The few exceptions to this rule were not relevant to our application, while all of the wxWidgets sample code provided could be freely inter-mixed with original RTLS code. When examining the available documentation, we determined that wxWidgets has adequate though not spectacular online and print documentation including tutorials and examples for getting started. Indeed, wxWidgets serves as a wrapper for several other potential choices, using GTK+ and QT if they are native to the current platform while transparently incorporating Windows and Mac natives on those platforms. Other GUI libraries, such as Visual Basic, don't support Linux and introduce licensing difficulties open source alternatives lack.

wxWidgets was selected as our choice of a GUI library as it incorporated all of our desired features in a single GUI library. Implementing the GUI simply required us to modify freely available example code and integrate it with features of the localization code we had already developed. Just as importantly, wxWidgets provides many core features in a relatively easy to use format. wxWidgets implements cross-platform threads, sockets, and events. Other libraries such as Boost also provide these features but would increase both the compile time and learning curve of the RTLS. While wxWidgets' threads were not substantially different from pthreads, the event-driven paradigm proved helpful even when writing non-graphical functionality. wxWidgets' implementation of sockets proved to be far easier to use and make robust than Berkeley sockets, so we migrated the RTLS to that paradigm. However, we reserved our prior Berkeley socket client implementation for the USRP2 sensor code as BBN80211 does not use a wxWidgets GUI.

In retrospect, wxWidgets and C++ proved difficult to work with due to manual memory management and many other nuances. A different approach, such as Microsoft Visual Basic,

may have been a more time-conscious choice. While cross-platform and Linux compatibility is helpful, our decision to use sockets as an interface between the sensor and RTLS permitted us to run the RTLS independently on a different computer. This permitted us to use any OS for the RTLS, though Linux would still be preferred for other reasons. However, enduring these difficulties did add value to the RTLS. A finished product capable of meeting the demands of a deployment environment can reuse any of the existing code without compromising platform compatibility, licensing or performance considerations.

3.10 Data Quality Analysis Tools

The processing and reporting component of our system design, shown in Figure 15, contains our data quality analysis tools. These tools are also known as the Excel Localization Test Bench (ELTB) and the Post Processing Localization Database (PPLD). Both of these tools automatically generate useful reporting documents by collecting data from the project CSV. The data is processed using mathematics and creates testing environment reports containing tables and graphs. Both tools have their own unique benefits and help to expand upon each other.

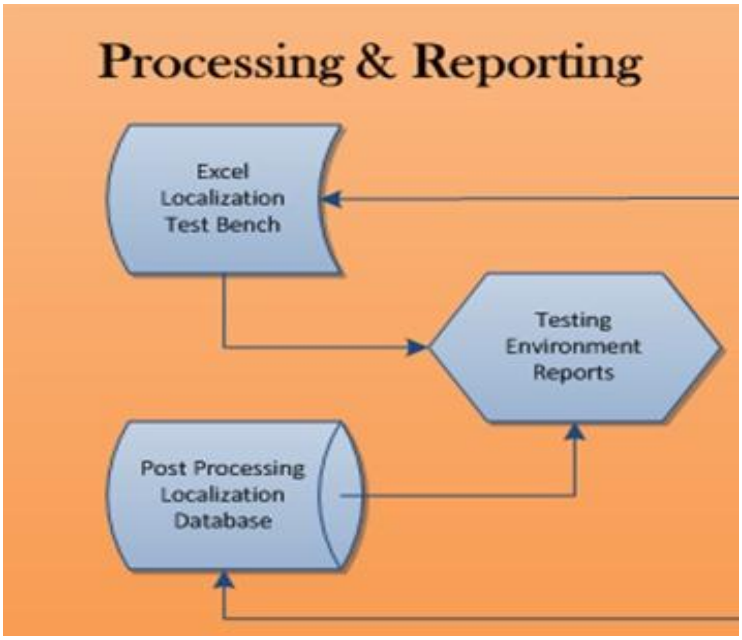


Figure 15: Enlarged Flow Diagram of Processing and Reporting

The ELTB allows us to establish quick and easy calibration. It takes the raw project data and produces both RSSI and ToA reports based on errors in the system. It also suggests the best initial parameters that can be used to enhance the system. Input fields allow you to change parameters and possibly enhance the results. The ELTB is quick and easy and only requires that you import calibration data into the results sheet of the ELTB.

Below in Figure 16 is an example of the most basic concept of ToA Correction via the ELTB. The test is a calibration test at one meter, with our initial time difference and deviation set to zero. The ELTB gives results revealing the average time difference of the test. The captions explain the process we take to find the best calibration parameters.

ELTB ToA Test Bench for Calibration at 1 meter

Before Calibration:

TEST BENCH							Test Distance(Input 1
MAC Address	TimeSent(s)	TimeSent(ns)	TimeArvl(s)	TimeArvl(ns)	TimeDifference	Distance	Distance Error
00:50:c2:85:38:0	1300158927	0.091641903	1300158927	0.094419	0.002777100	83255.35	-83254.34729
00:50:c2:85:38:0	1300158928	0.493355989	1300158928	0.49649692	0.003140927	94162.61	-94161.61458
00:50:c2:85:38:0	1300158928	0.296859026	1300158928	0.300282	0.003422975	102618.2	-102617.221
00:50:c2:85:38:0	1300158931	0.997720003	1300158932	0.00090289	0.003182888	95420.58	-95419.58393
00:50:c2:85:38:0	1300158945	0.182945967	1300158945	0.18613505	0.003189087	95606.41	-95605.40955

2. Distance(m): 94212.64

Average TOA Results				
Avg_Time_Dif	Avg_Distance	Avg_Dist_Err	Avg_timing offset	SD
0.003142595	94212.6353	94212.63527	-0.000314256	0
Output	Output	Output	Output	Intput

1. Notice that the initial timing offset is 0.

2. The Distance above is the average distance and it is incorrect. Should = 1.

3. Our results reveal our offset. Next, we shall enter it in below

Initial Variables	
Timing Offset	0
Avg_Timing offset	0

After ToA Calibration:

TEST BENCH							Test Distance(Input 1
MAC Address	TimeSent(s)	TimeSent(ns)	TimeArvl(s)	TimeArvl(ns)	TimeDifference	Distance	Distance Error
00:50:c2:85:38:0	1300158927	0.091641903	1300158927	0.094419	-0.000365461	-10956.26	10957.25886
00:50:c2:85:38:0	1300158928	0.493355989	1300158928	0.49649692	-0.000001634	-48.99156	49.99156471
00:50:c2:85:38:0	1300158928	0.296859026	1300158928	0.300282	0.000280414	8406.615	-8405.614858
00:50:c2:85:38:0	1300158931	0.997720003	1300158932	0.00090289	0.000040327	1208.978	-1207.977782
00:50:c2:85:38:0	1300158945	0.182945967	1300158945	0.18613505	0.000046526	1394.803	-1393.8034

7. Distance(m): 1.029123

Average TOA Results				
Avg_Time_Dif	Avg_Distance	Avg_Dist_Err	Avg_timing offset	SD
3.43279E-08	1.02912322	1.029123219	0.000000000	0
Output	Output	Output	Output	Intput

4. We have changed our Initial Variables to the suggested.

5. The deviation is the error of the system.

6. Timing Offset is miniscule.

7. Distance corrected.

Initial Variables	
Timing Difference	-0.003142595
Avg_Timing_Dev	0.000000034

Figure 16: ELTB test bench for calibration at 1 meter

We change the initial time difference to the value given by the ELTB. This enhances our results, however does not fix it precisely. The time deviation input can be set by the user and should ultimately fix the final average distance to what it should be. In our case, the average time

difference is .003142595 seconds for our one meter results. By conducting these experiments within the ELTB, we can organize test data by distance in order to plot system errors. The concepts introduced when creating the ELTB have found its way into the PPLD to provide as a foundation for error correction.

The PPLD has been modified to ds the exact same calculations, however they are done automatically. In addition, because the database collects and stores multiple projects, the averages used for calibration are of greater precision. The PPLD also monitors the average errors associated with RSSI and ToA values and filter the errors so that when an incorrect value is passed to our system the PPLD swaps the value with the correct one based on the likeliness it has occurred. Therefore for an environment with lots of calibration and project data, is likely to have better results. This can easily be tested by using the Access Test Bench GUI.

The Access Test Bench Graphical User Interface is designed to test the effect of different parameters, as we have been done preciously in the ELTB. The parameters include time offset, time deviation, pathloss gradient, sensor location coordinates and more. By changing these parameters we are able to monitor the affect they had on the errors of our system.

This is a fundamental learning technique that can be applied to all SDR localization systems. Studies on localization systems require that you observe the environment and that you conduct multiple tests with varying test conditions in order to determine the best initial parameters. To do so accurately you need a lot of data, a lot of tests, and lots of results. By doing so you can learn to adapt and manipulate your system to obtain the correct results.

Other data used by the PPLD is obtained from the Localization Test Results computer file. This allows the database to pull up localization results by project identification and then plot them sequentially on a map. Overall we can produce more sophisticated results such as the time

offsets specific to MAC address, average environment errors for RSSI and ToA, environment specific calibrations, and localization result comparisons.

Post-Processing teaches us many things about the environment and the components we have developed allow us to capture and display our results in a way that can be easily understood by anyone. The complete SDR Localization design had been developed so that any SDR hardware can be connected and utilize the benefits of the system. Therefore this modular design would also work on successors to the USRP2 such as the E100 and the N210. The task of designing this interweaving system was not novel due to the multiple relationships needed for the system design and PPLD.

3.10.1 Managing Test Data

In order to successfully localize an unauthorized user we need to adapt our system to the environment we are testing, which requires calibration. We start by initializing variables associated with RSSI and ToA distance calculations. Once we have learned about the test environment we can set timing offsets, deviations and gradients in order to reduce any large error in the position estimates produced.

The ELTB and PPLD, introduced in Section 3.4, manage the test environment. The ELTB can be utilized for pure calibration using two USRP2's in a test environment. In the ELTB a user is allowed to traverse through four sheets of information containing imported results, a test bench for ToA enhancing, a test bench for RSSI enhancing, and a sheet that uses our localization algorithm to determine the MS's position and displays results. The other tool we created is the Post Processing Localization Database, which maintains a history of all test results while suggesting calibration data based on environmental characteristics. In the next section we go into the organization of our quality assurance tools.

3.10.2 Excel Localization Test Bench

The Excel Localization Test Bench was created in order to experiment and manipulate our results primarily for calibration between two sensors in the environment. Using Excel was a preliminary processing scheme that we decided to use until a method of logging real time data was established. This takes in results in the form of a CSV file, created by our logger files, which were developed in C++ and Python. The excel file contains multiple sheets that compute results based on the preceding sheet labeled “RESULTS”. The “RESULTS” tab is where we import calibration test results via the importing CSV process in Microsoft Excel.

This test bench also automatically manipulates the results in order to display characteristics of the environment such as average path loss gradient and average latency of our system. The different sheets are the RESULTS, TEST_BENCH_TOA, TEST_BENCH_RSSI, and LOCALIZATION. On the sheets are cells that correspond to initialization inputs along with suggested outputs. Below is a description and sample image the actual sheets and fields associated with them.

RESULTS

The RESULTS tab is strictly for importing new data. This updates every sheet with the appropriate fields and calculations.

TEST_BENCH_TOA

For our TOA calculations we needed to be able to determine our average timing difference. We also need to be able compensate for the timing deviation which is a more precise offset added to help calibration. Below are the contents of TEST_BENCH_TOA.

TEST_BENCH_RSSI

We need three variables for RSSI calculations. The variables are the alpha that pertains to the path loss gradient of the environment, the standard deviation of our RSSI results, and the transmit power at 1 meter. When testing two USRP2s we can find average values of RSSI, pathloss, and distance along with a new standard deviation to help calibrate our system. Below are the contents of TEST_BENCH_RSSI.

LOCALIZATION

Although not a functionality of the original design, localization has found its way into the ELTB. In order to accomplish these tasks the system required that excel distinguished between a calibration test and a regular test. The additional steps introduce a variable that requires the user to designate what type of test is being conducted allowing you to classify the results and successfully distinguish between the two.

Localization requires five inputs and should return an XY location that pertains to the position of a user. The inputs are the average distance measurements at each of the three sensors and the XY location at each of the nodes. The distance measurements are either from ToA results or RSSI results so there are two different localization results for each test.

3.10.3 Post Processing Localization Database

The use of databases has become extremely common in the design of major projects due to the excellent capabilities it provides a user when storing and information querying. Because of the nature of our project there are multiple initial conditions that require calibration based on the propagation of the environment. The Post Processing Localization Database allows us to conduct various different projects in multiple different test environments and review the results of each

project. The PPLD additionally observes the nature of these probabilistic signals and averages the environmental conditions in order to enhance calibration settings as each test gets added to the PPLD.

One of the more important design features is the automated data population of each test as they are conducted. It allows the database to update our calibration settings automatically. This is due to a behind the scenes interaction between the database and CSV files.. In this project we identify groups of information by project and environment. This becomes very powerful, as multiple CSV's are stored on our computers in order to provide our design with bi-directionality capabilities. Recall that the CSV's initial responsibility was to provide our python test modules with the proper calibration settings. Meanwhile our database serves as an auto-updating calculator that our modified 80211BBN code can call upon during runtime in order to retrieve updated values.

One of the main concerns brought to our attention was the security of this information. Seeing as this information may contain sensitive results due to the constant updating of calibration data we do not want unauthorized users to alter any of the data. This calls for a security protocol that only allows access to authorized users. Microsoft Access is capable of adding security to our project and it also can create a new table containing authorized users in the process. This not only allows us to add security to our system but we can track who has conducted test or modified any data. We can also create multiple bios for our staff members. Multiple groups of people may need to access the data but not necessarily make changes sp we decided to add different levels of access such that each user may contain a varying level of read and write privileges. It is clear that organizing test data and creating relationships allows us to introduce a multitude of functionality to our system.

The database relies heavily on tables that organize data but requires advanced preparation before attempting to build. The database is made up of six tables. These tables are the PPLD Master Table, Input Controller, Environment Table, Mac Table, BSSID Frequency Offset, and the RSS Error Table. Each Table is connected via some identifier in order to establish a relationship between data. This becomes helpful because we can identify the fields that affect each other, for example, if we received an administrator id of 1, we could use that information to figure out who used the database from the employee table. It also allows us to safe guard unnecessary personal information when transmitting data such as a person's name. Table 15 provides a brief description of each of the major tables.

Table 15: PPLD table descriptions

Name of Table	**Description of Table**
PPLD_Master_Table	This table contains all imported data associated with any projects.
Input_Controller	This table contains the control functions that used to alter initializations such as time offset or test location (used when calibrating).
Environment_Table	This contains a list of environments along with their pathloss gradient's, and any associated standard.
Mac_Table	This table contains a list of all Mac Address previously entered in the system as well as its XYZ location.
BSSID_Frequency_Offset	This table contains a list of base station id's allotted to each Mac Address along with the frequency offsets of each.
Employee_Table	This table is used to lookup test administrators via admin_id.
Master_Distance_Calculator	Calculates predicted distancesand assigns error identification.
Calibration_RSS_ToA	Error table used to track average distance error by actual test result.

Figure 17 shows the complete PPLD database architecture.

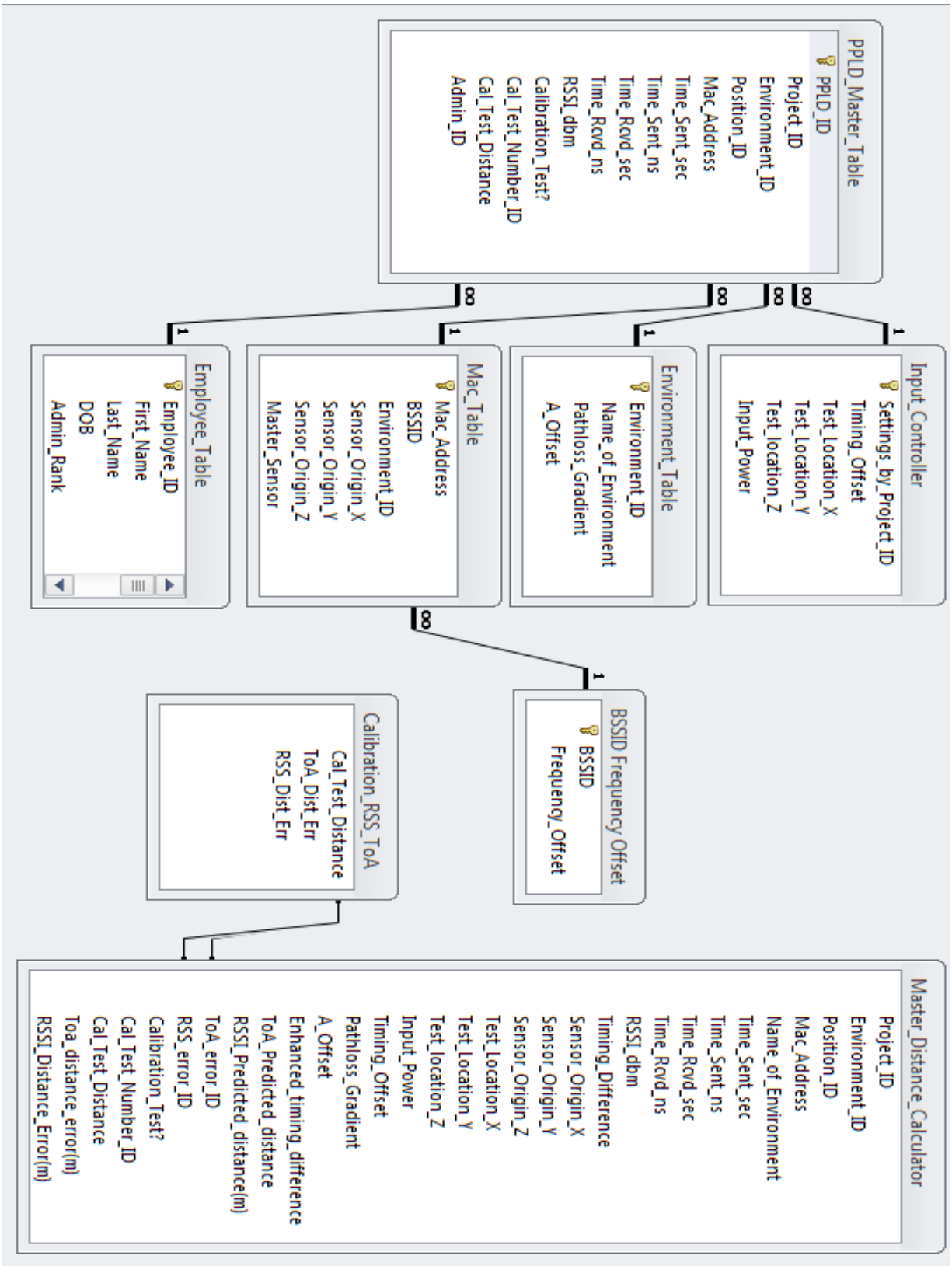


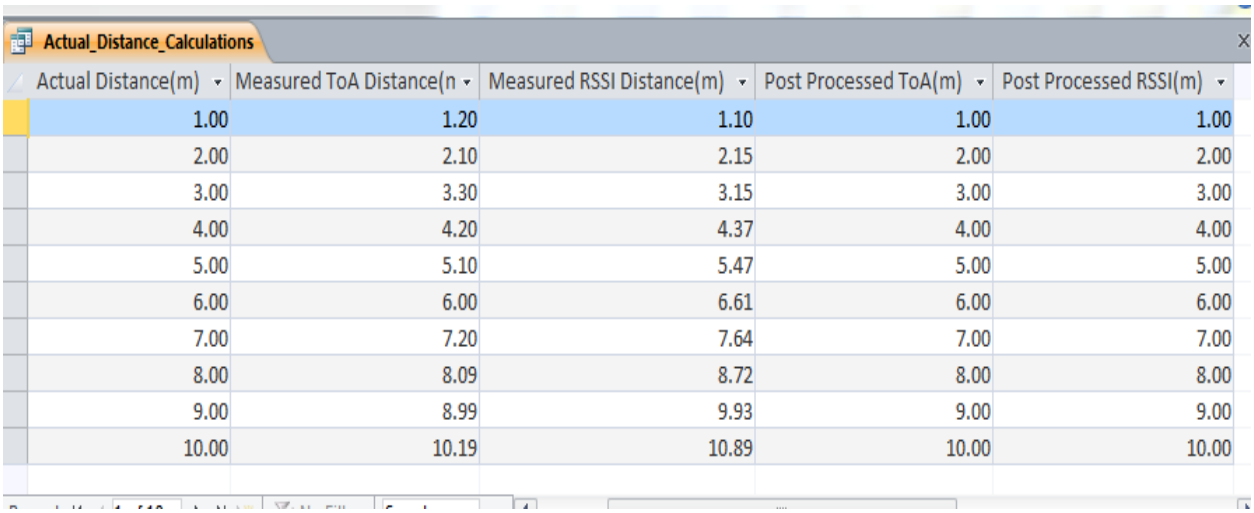
Figure 17: Relationships of PPLD parameters

In general, calculation based queries are used when multiple inputs are needed to reach a desired result such as to perform mathematical algorithms. Form based queries are prepared queries containing the data a form will need. For instance, if you were doing calibration initialization you would only need to see the fields that relate to calibration. A cross tab requirement is created when a report requires us to analyze the change between two different fields. An example would be, tracking monthly shipping cost by shipping categories, which is more sophisticated than just analyzing total monthly shipping cost. Report based queries are used for reporting such as graphs, finalized tables or cross tab queries. Table 16 contains a list of our queries followed by their desired purpose.

Table 16: List of queries and descriptions

Query Name	Query Type	Description
Actual Distance Calculations	Calc. Based/ Repor.t Based	Compared Measured and Post Processed Distances
Actual Time Delay Calculations	Calc. Based	Lookup query to compare Time Delays
Calibration RSS ToA	Calc. Based	Contains Calibration results
Employee Project Diagnosis	Form. Based	Used for Employee Forms
Auto Correct Distance Filter	Calc. Based	Corrects Distances based on environment &error
Distance vs RSSI Measurement	Report. Based	Distance vs. RSSI chart
Distance vs Toffset Comparison	Report. Based	Compares various Time delays
Mac Specific Time Statistics	Report. Based	Average Time delay by Mac
Pathloss interpretation	Report. Based	Slope and Y intercept of RSSI pathloss
ToA/RSSI Average distance Error	Report. Based	Average Errors associated with distances
Validation Select Master Sensor	Form. Based	Master Sensor Identifier Validation
Localization	Cross Tab.	Sequential Non Real Time localization Calculation

As an example, we show the result of the Actual Distance Calculation Query in Figure 18. This displays the predicted distances of both RSSI and ToA for 1 to 10 meters, as well as the post processed distances which reveal the correct enhanced distances.



Actual_Distance_Calculations

Actual Distance(m)	Measured ToA Distance(n	Measured RSSI Distance(m)	Post Processed ToA(m)	Post Processed RSSI(m)
1.00	1.20	1.10	1.00	1.00
2.00	2.10	2.15	2.00	2.00
3.00	3.30	3.15	3.00	3.00
4.00	4.20	4.37	4.00	4.00
5.00	5.10	5.47	5.00	5.00
6.00	6.00	6.61	6.00	6.00
7.00	7.20	7.64	7.00	7.00
8.00	8.09	8.72	8.00	8.00
9.00	8.99	9.93	9.00	9.00
10.00	10.19	10.89	10.00	10.00

Figure 18: Actual distance calculations of PPLD

With the fundamentals established the next step was creating a graphical user interface. The production of the graphical user interface is done within Microsoft Access using a combination of VBA and SQL code. This allows us to view input forms, data oriented reports, graphs of data and many other useful tools that reveal information about our project.

The creation of this database gives our SDR project flexibility. Not only can we manipulate and expand upon the type of data the database receives, we also have limitless possibilities to manipulate and enhance our design. The PPLD is interchangeable when it comes to SDRs. As long as the imported data to the PPLD is consistent with that of the Master Table, and initializations such as sensor locations are set, the PPLD will provide as test bench and report analysis tool for many future SDR projects to come.

3.10.4 Summary of ELTB and PPLD

The goal of the ELTB and PPLD was to provide post processing abilities that enhance our overall localization system design. They are also to help determine optimal calibration

settings as well as provide us with a report that shows the path of an unauthorized device. The two creations interact with our data differently providing us with multiple testing options and opportunities.

The ELTB allows us to determine important calibration information quickly while at the same time allowing us to manipulate the results to see how inputs may affect them. With a few tweaks we are also able to group the imported file and create a table containing the XY coordinates of an unauthorized user and plot the path taken. The PPLD allows us to implement the exact same functionality of the ELTB plus much more. With the PPLD we are not limited by the design of an excel spreadsheet so we can always expand our tables to collect additional fields. The PPLD also allows us to choose from advanced queries, forms, and reports that have been originally created whereas the ELTB is more of just a reporting tool that utilizes form input fields. The tradeoff is that the ELTB is straightforward and takes little time understanding, whereas the PPLD covers more ground, therefore takes more time to understands but provides us with the ability to update multiple computer readable files that can interact with our modified 80211BBN code.

3.11 Summary

The capabilities of software defined radio were employed to track unauthorized transmitters within a network of stationary sensors. Both time of arrival and received signal strength are employed to calculate the distances to the mobile node, while our localization algorithm is able to calculate the position of the node and provide a level of confidence in the calculation's accuracy. An existing 802.11b implementation within GNU Radio was extended to meet our design specifications, with new functionality being implemented to collect data and allow for easier parameter configuration.

4 System Deployment and Testing

To test the performance of our system, we deployed it in a variety of locations. These tests were to provide examples of the system's data output as well as to evaluate the accuracy. Preliminary testing allowed some offsets to be found and accounted for in later testing. Large scale testing used four USRP2s and was the most complete method of testing the system as a whole.

4.1 Test Locations

Each particular test environment may alter the propagation of waves through the air. In order to test our system under these varying conditions, we conducted tests in several locations, each with different properties. These locations included on and off campus housing as well as several locations on campus. Table 17 compares the properties of each test site.

Table 17: Comparison of testing sites

	Off-Campus Apartment	East Hall Apartment	Atwater Kent Labs	Bowling Center
Size	Small	Small	Medium	Large
Obstructions	Some	Some	Many	None
Other Wireless Activity	Medium	Medium	High	Low
Signal Strength	Medium	Medium	Low	High

By testing in these locations, we subject our system to different types of conflicts such as non-line-of-sight, physical and wireless interference, multipath, and low signal strength, each of which could affect the performance of the system.

4.2 Sensor and Transmitter Configuration

Although the location of each sensor should not affect the performance of the system, we conducted testing with a variety of arrangements. In general, it is best to have the sensors spread out equally in somewhat of a triangle. This provides a route from the transmitter in all directions, as well as the easiest integration with our localization algorithm.

4.2.1 Frequency Offset

Each USRP2 has a unique and sometimes varying frequency offset for both its transmitter and receiver. In order for a pair of two USRP2s to communicate successfully, the offset must be compensated for. To do this, we simply transmitted a signal at a known frequency from one USRP2. On another, we can look at the FFT to realize what the offset is. We used a frequency of 2.49 GHz for all testing. Figure 19 shows an example of this procedure.

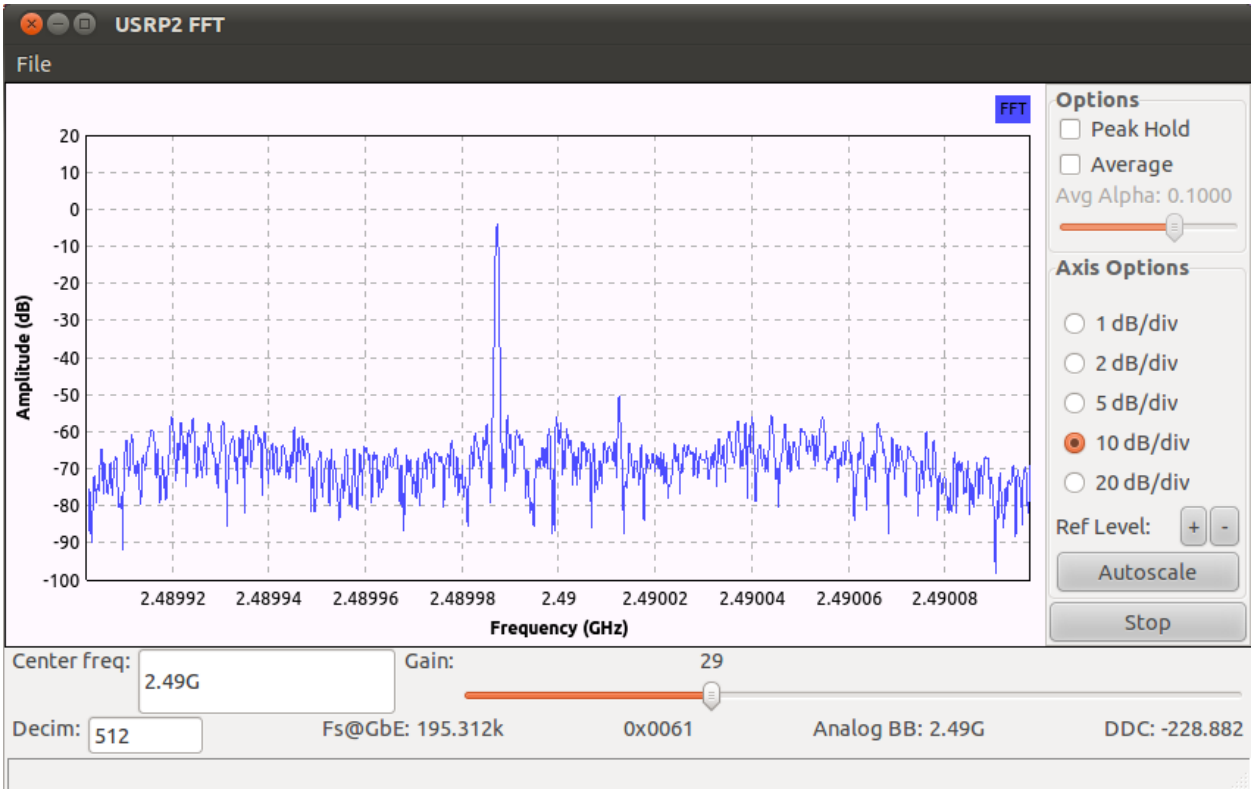


Figure 19: FFT for frequency offset detection

As shown in the figure, the signal is not being received at the frequency of 2.49 GHz that it was transmitted at. Due to a frequency offset in the receiver, it appears to be receiving at roughly 2.489985 GHz. Since our testing involves one transmitter and three receivers, it is easiest to transmit at a known frequency and adjust the receiving frequency to compensate for the offset of each USRP2. For the USRP2 used to obtain the figure, it would have to receive at 2.489985 GHz in order to properly receive the data transmitted at 2.49 GHz. This process can be repeated for each of the sensors to find at what frequency the receivers must be set to. The results of one of these tests are shown in Table 18.

Table 18: Frequency offset test results

Sensor	Offset
EDU-003	-11 kHz
EDU-008	-32 kHz
EDU-009	-17 kHz

These results were used during our testing that allowed us to implement communication between the intruder and sensors. Using these offsets and a 2.49 GHz transmit frequency, these sensors need to be set to receive at 2.489989, 2.489968, and 2.489983 GHz respectively. Although these offsets worked for the duration of our test, they can vary over long periods of time and are specific to each set of radios used. Until an automatic frequency compensation mechanism is implemented for BBN80211, the offset must be determined before each test.

Although using these offsets allowed us to communicate with all sensors simultaneously, there were times when the receive frequency needed to be adjusted. If a sensor stopped receiving data consistently, we can simply look at the FFT to confirm the frequency offset and adjust accordingly.

4.2.2 Gain

Due to the variable nature of our testing, the gain of each sensor is also something to keep an eye on as conditions change. If one sensor is far away or behind walls, then it will need to have a higher gain applied to it than a sensor that is close or in the line of sight of the intruder.

Similar to the frequency offset, this can also be addressed by looking at the FFT of each sensor. As the gain is changed in the FFT window, the received waveform will generally increase or decrease in amplitude. If it is too low, then the receiver might not be able to hear the transmission clearly enough to decode it. If it is too high, then any noise will also be amplified to

the point that it can interfere with the data being received. Figure 20 shows a waveform that has too much gain applied to it.

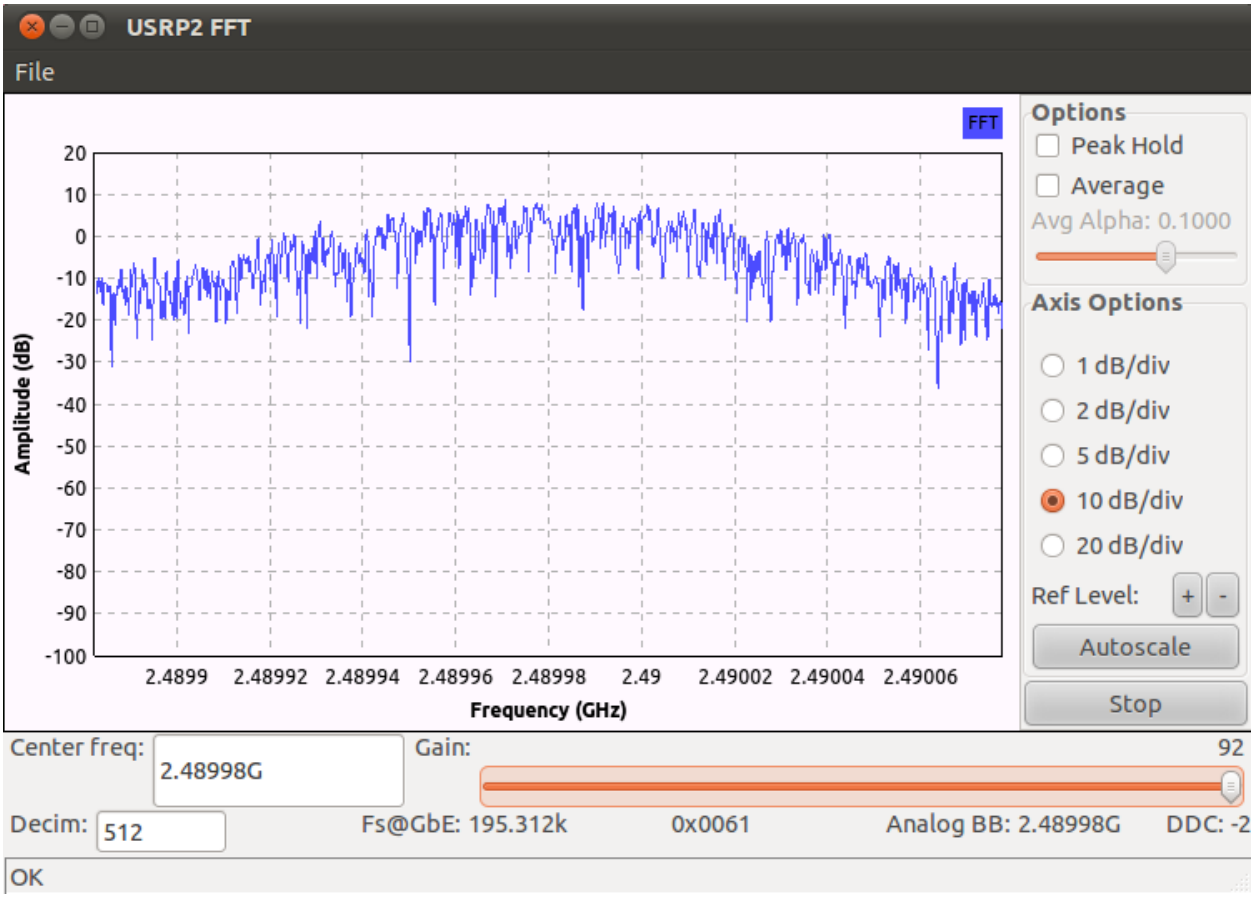


Figure 20: FFT showing effect of too much gain

Also like the frequency offset, the gain may need to be adjusted throughout the course of testing. Since the tests were performed with the transmitter at different locations, the gain needed to be considered for each of these locations. Using the FFT and some preliminary testing, we found that a gain of 29 consistently received data. During testing, each sensor started with a gain of 29. If any sensor was failing to receive data, the gain would be increased until it was able to receive the transmission successfully and consistently. This could also be verified at any time by again looking at the FFT of the receiver.

4.3 Time Synchronization

In order to send accurate timestamps so we can calculate distance we must be synchronized to each other's clocks, otherwise we are completely wrong in our calculations. We must also take into account drift, which occurs in the system as well. In order to establish time synchronization between our units we decided to look into Network Time Protocols so that we can use a server's atomic clock, similar to what GPS currently does in order to sync computers.

We set up NTPD on the computers and the results were successful. The protocol not only syncs the time to the server but it determines our computers offset to the server and has an algorithm that over time tracks the drift. This shows how to track and adapt our system based on incoming data that is similar to the research conducted in cognitive radio.

The benefits from synchronization via NTPD is that our system times are synced so we can have greater accuracy when calculating distance and we don't have to establish bidirectional communication for the purpose of round trip time synchronization. After looking into the NTPD documentation we may be able to develop a system that establishes time synchronization however NTPD does the job we need in order to conduct more reliable experiments so we utilized it.

4.4 Packet Protocol

Our packet protocol helps us determine how to decode data correctly at the receiver. We have developed multiple stages of packet protocol testing. One is conducted in Python while the other is conducted in C++. The python testing is the preliminary stage that we use to verify that our design will work. This section pertains only for the testing in Python.

Looking into the BBN code we have acknowledged that packet length, RSSI value, Source ID, time since the receiver was turned on and data rate is initially passed in. The packet

contents also contained a payload, which is where we decided to pass the additional information we needed. The additional information we transmitted was the MAC address and a timestamp at the point of transmission. Once the information is successfully passed into the receiver we must parse the message so that we can utilize each piece of data accordingly. Figure 21 shows both the transmitter and receiver message being processed.

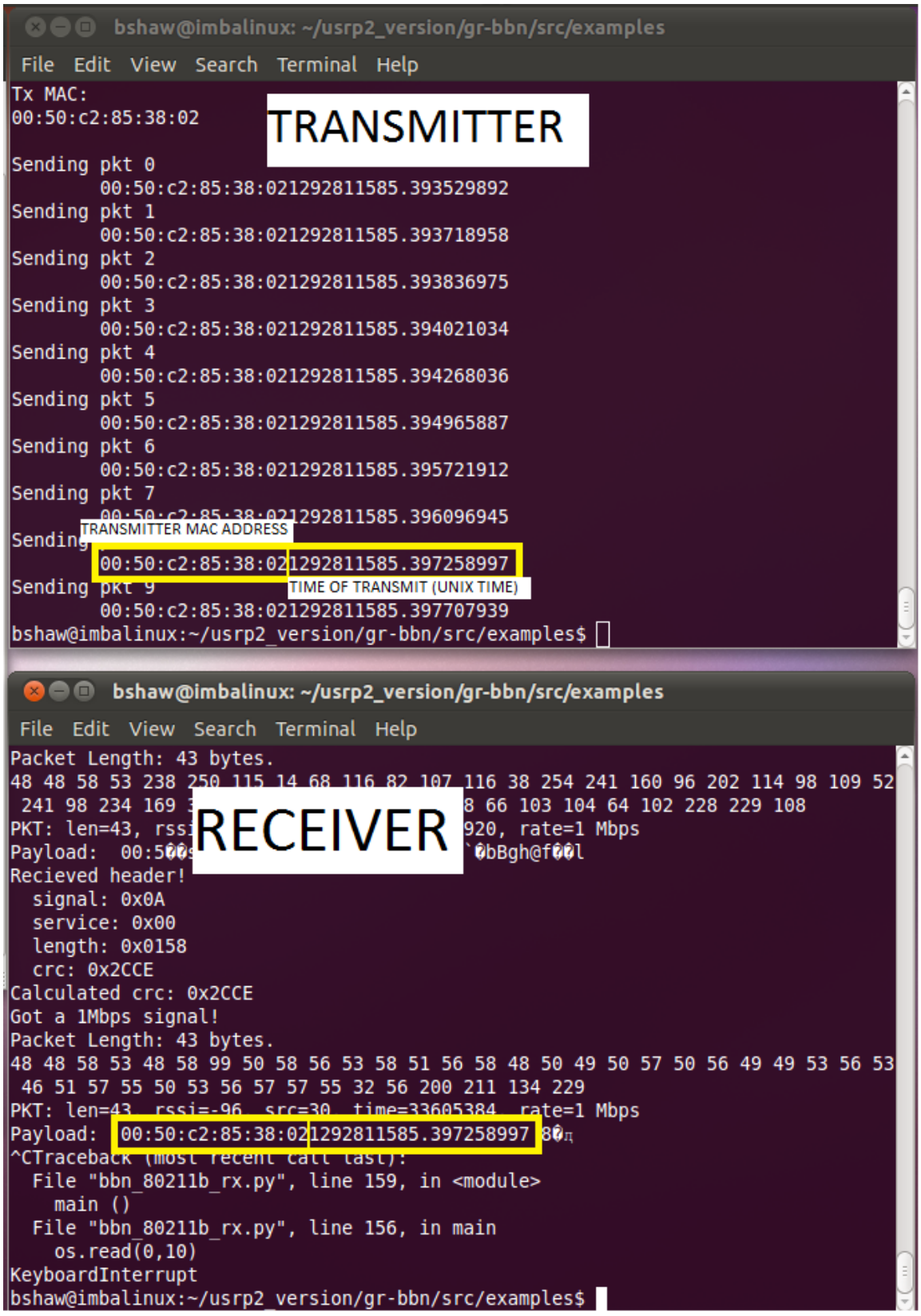


Figure 21: Packet protocol test

5 Results of Data Analysis

This chapter will discuss the results from our research and testing and present an analysis of the data. Also included is a brief discussion on what the data means and an explanation of the errors encountered.

We ran several different tests to get a variety of data for our system. First, we ran some preliminary tests between only two USRP2s to check for any consistency or correlation. Using this data, we were able to put it through an extensive database solver that calculates useful data such as averages, standard deviations, and offsets. We also ran a large scale test using four USRP2s in an attempt to localize one of them.

5.1 Preliminary Tests

A test consisting of only two USRP2s (one transmitter and one receiver) was conducted at varying distances apart from each other. The distance between the two can be calculated by simply multiplying the difference in the timestamps by the speed of light. However, this is a very sensitive method and does not account for any delays in propagation or hardware.

Results from this test showed that the time difference was not linear with distance. This is partially due to the fact that both of the timestamps are not placed at the correct time. The receiver timestamp is placed when the entire packet has been received, causing a large delay while the data comes in and is processed. An alternate solution to this problem could be to place a timestamp the instant an increase in energy is seen at the receiver. After it has been confirmed that a packet was successfully received, the timestamp can then be associated with that data and logged.

5.1.1 Time of Arrival Testing

This test consisted of getting a large amount of data at several different distances between a pair of USRP2s. Several graphs present the data gathered during this test.

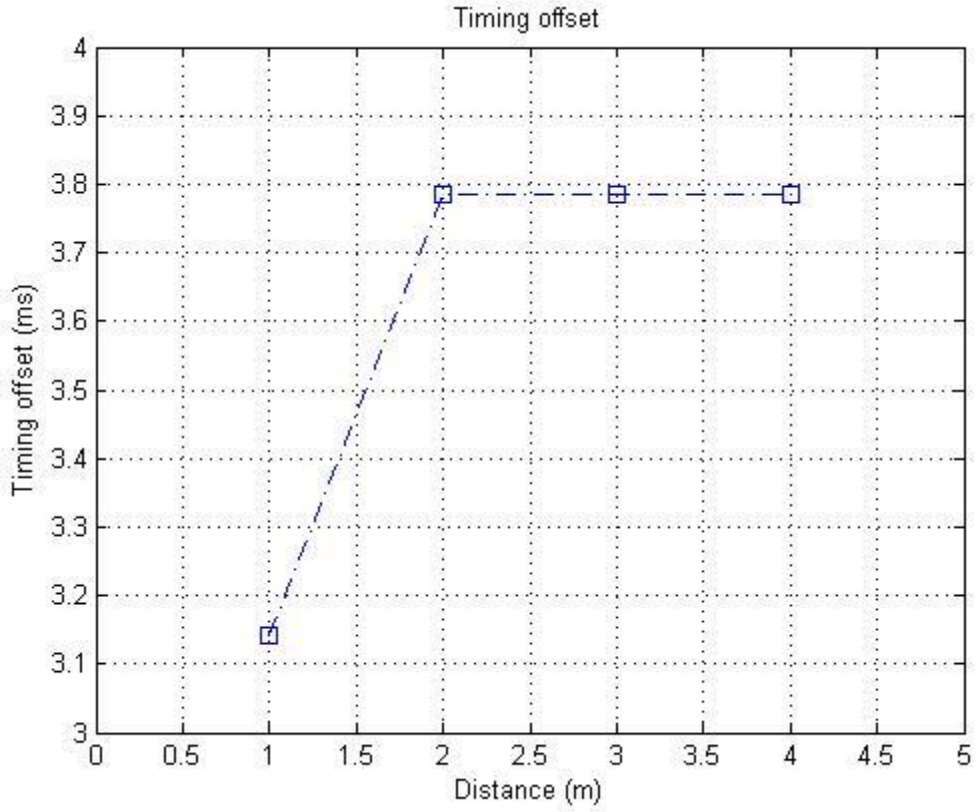


Figure 22: Average timing offsets from data

Figure 22 shows the average timing offsets at each distance. These were found by calculating the actual time difference and comparing it with the theoretical time difference, which is known since the distance in between them is known. As shown in the figure, the offset is relatively constant for all distances, which shows that it is mainly caused by hardware delays, or the fact that the receiver timestamps are placed too late. The slightly different offset at 1 meter can be attributed to the fact that it is a very small distance for a very high speed wave to travel.

Using the same data, the average distance can be found from the average time differences. These raw results are shown in Figure 23.

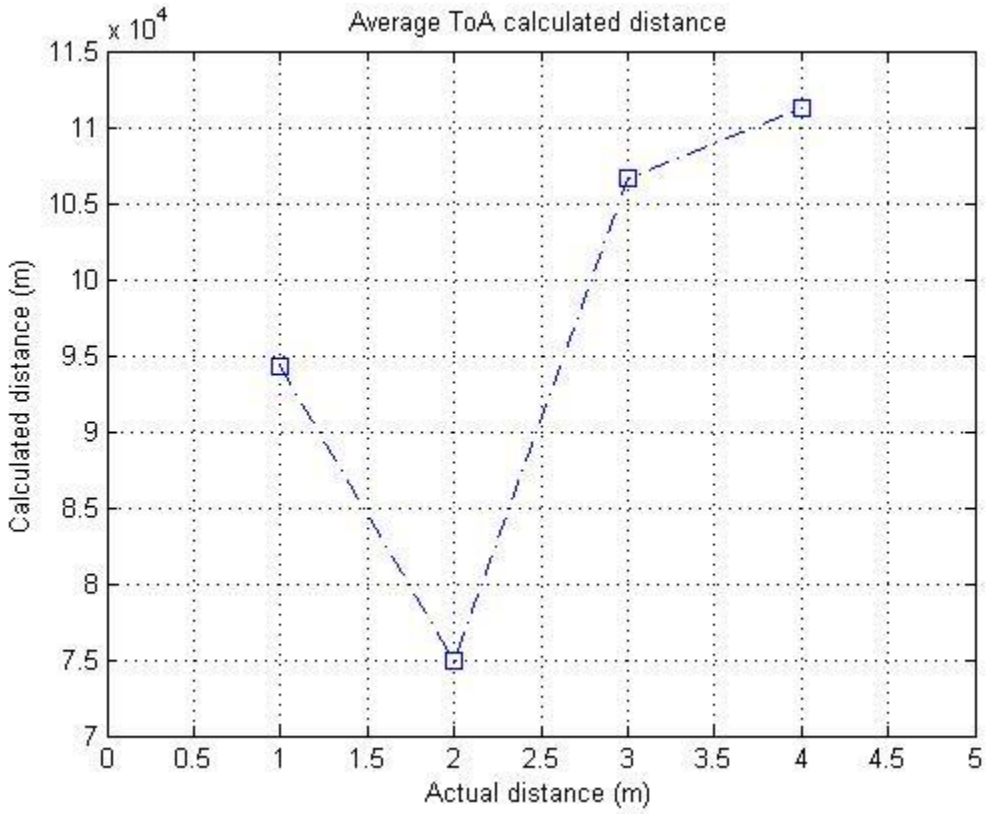


Figure 23: Raw distance results

As shown in the figure, the results from the raw values are not very accurate. However, after putting the data through the calculations to find the offsets, we are able to vastly improve the results, as shown in Figure 24.

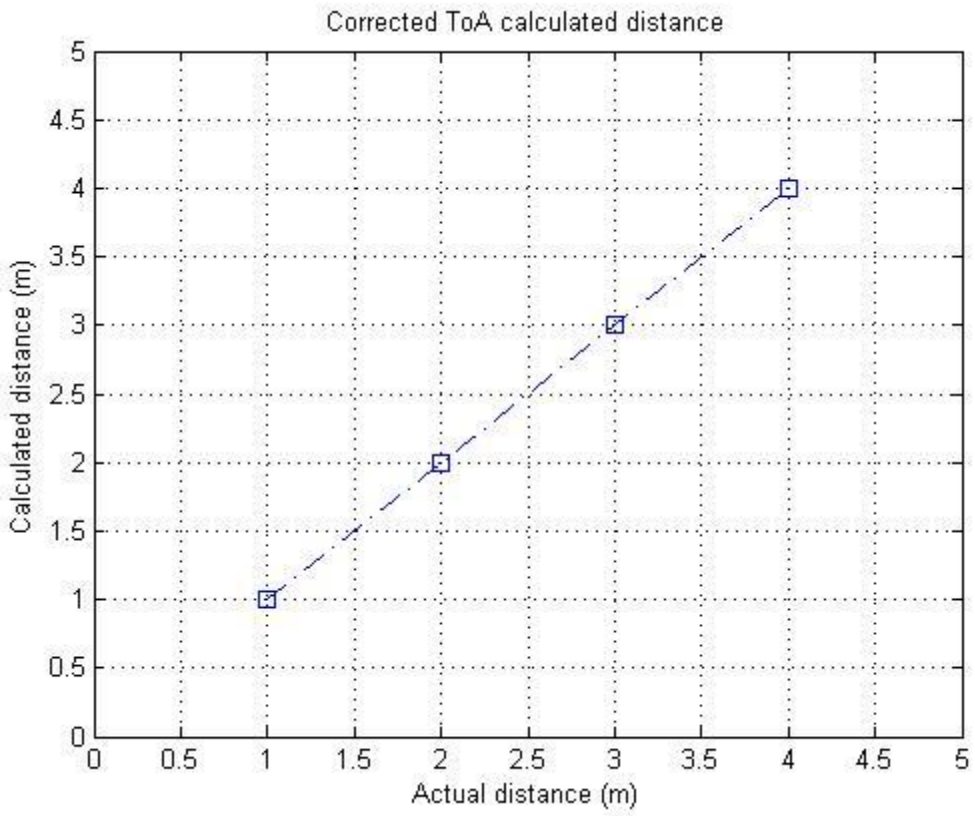


Figure 24: Corrected distance results

The offsets appear to be correcting for the delays in the system and resulting in very accurate calculations.

5.1.2 RSSI Testing

We also ran the same data through the solver using the RSSI from each of the trials. These results are presented below.

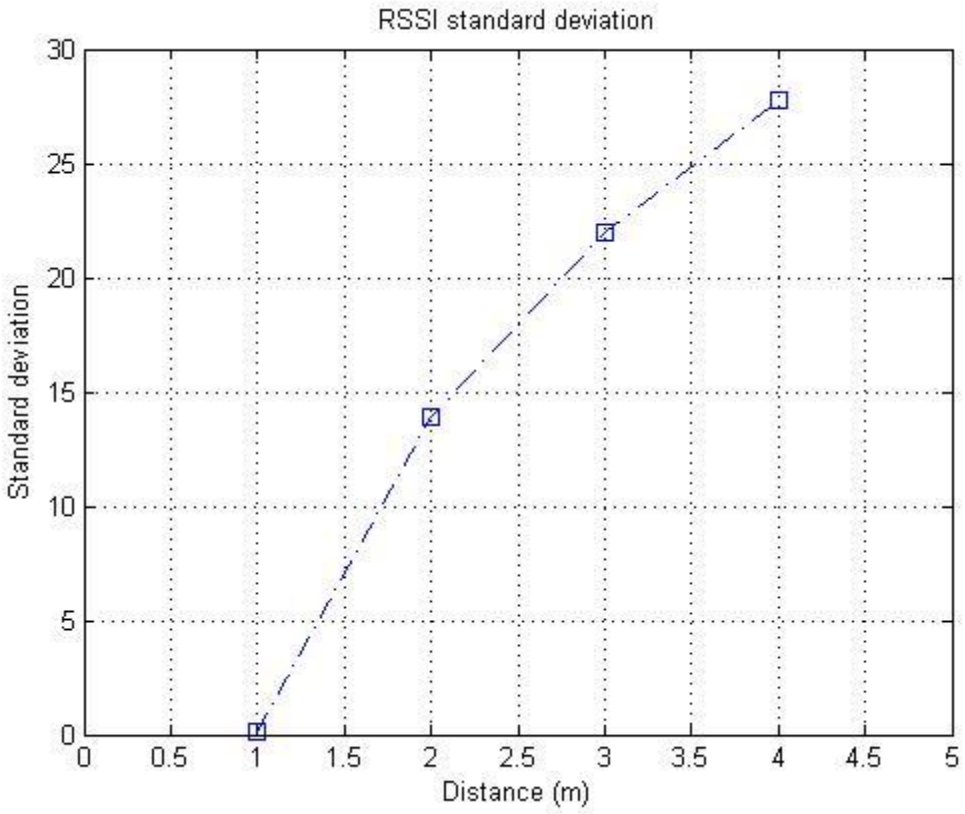


Figure 25: RSSI standard deviation

Figure 25 shows the standard deviation of the RSSI values at each distance during the test. As you can see, the environment becomes much more unpredictable, causing unreliable results. The raw results from the RSSI test are shown in Figure 26.

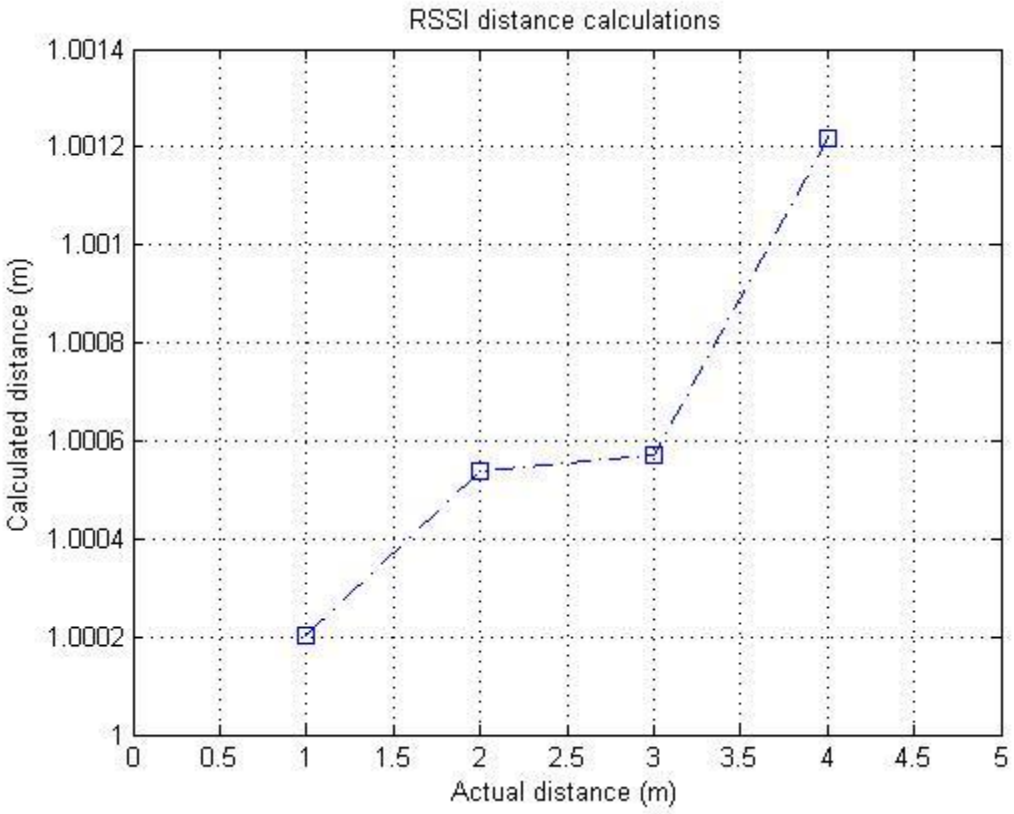


Figure 26: Raw RSSI distance results

The raw RSSI distance results were not very accurate since the RSSI values did not vary very much from test to test. There is, however, a clear correlation of increasing RSSI calculated distance as the actual distance increases. Again using some calculated offsets and initializations, we can adjust for the changing environment and varying RSSI values. These results are shown in Figure 27.

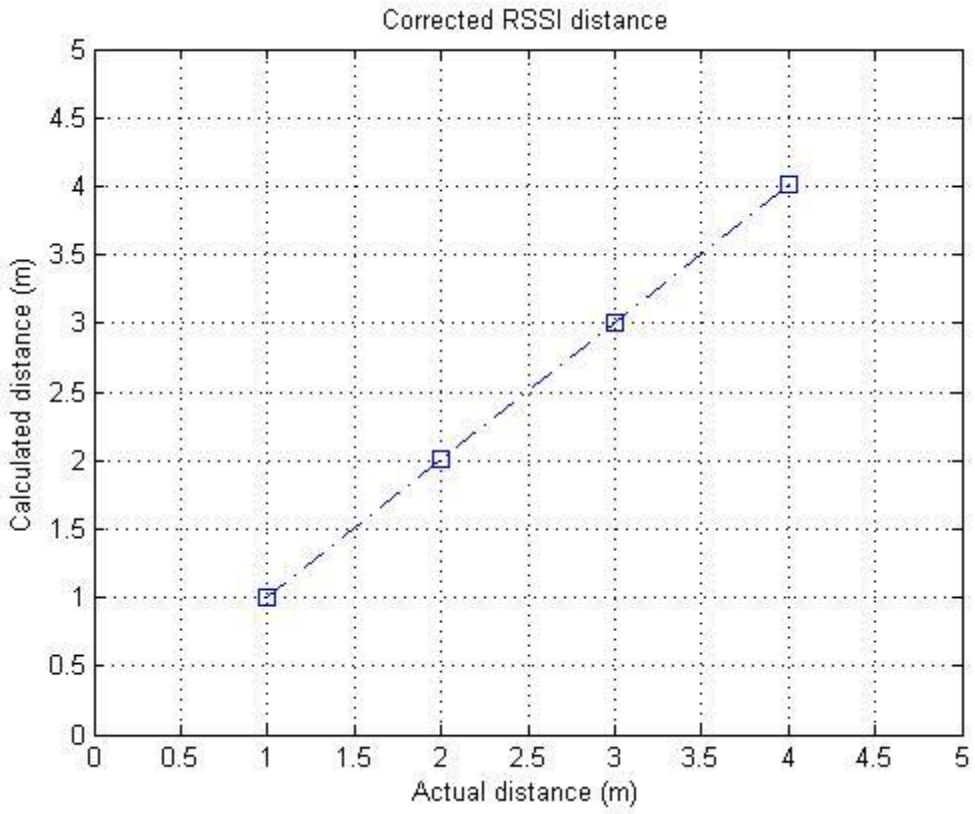


Figure 27: Corrected RSSI distance results

The figure clearly shows that the initializations improved the RSSI distance results to the correct values.

5.2 Test Results

Now that our packets were being decoded and parsed correctly we are able to add formulas to our python code at the receiver in order to calculate distance measurements. These formulas convert distance to time and are also used to calculate the offset in time needed to help the system robustness. We also print out the data as its modulated so that we can track the process on the screen. Below in Figure 28 is an example of a Python test result for one packet.

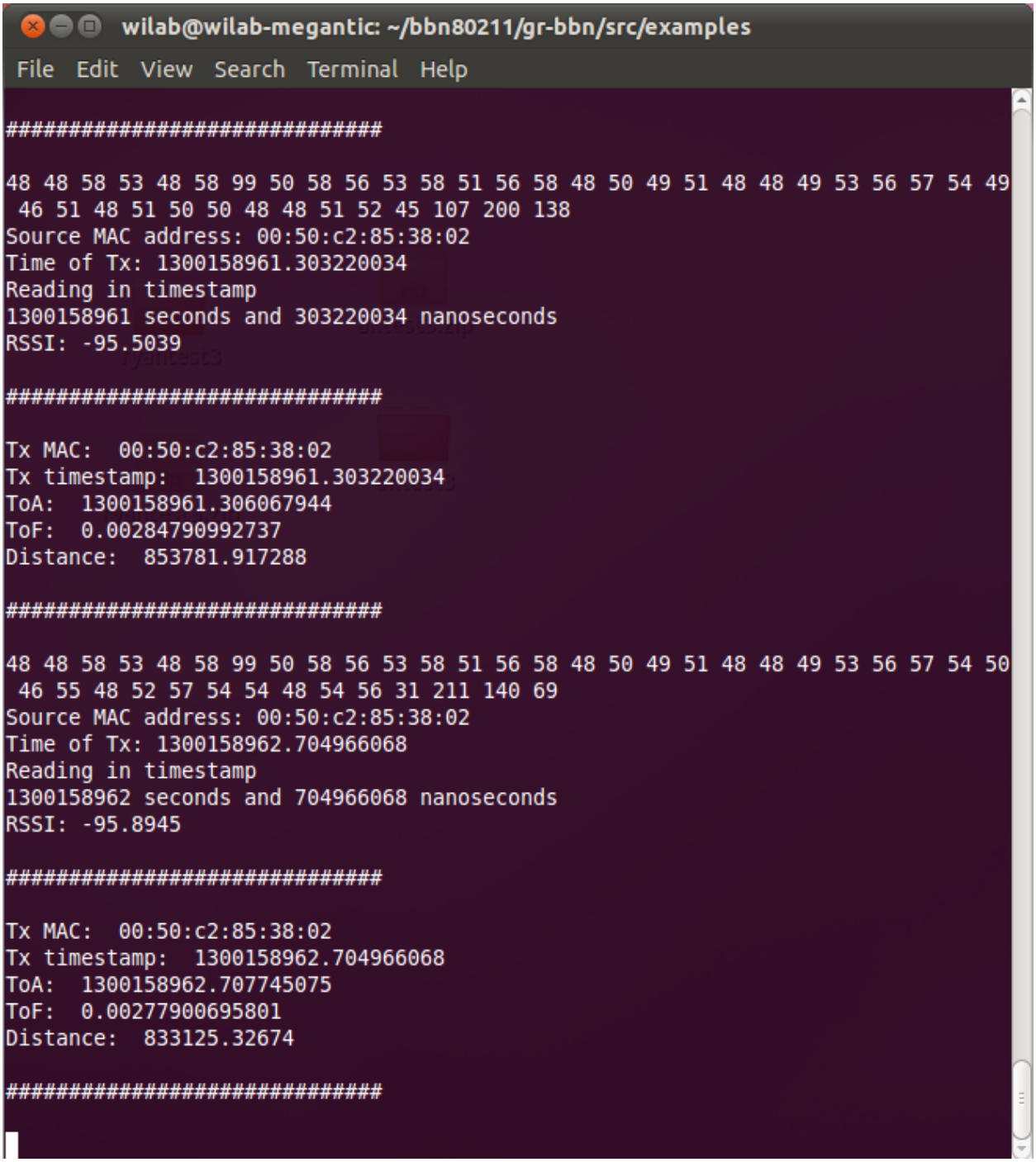


Figure 28: Python test receiver screenshot

Our results are initially inaccurate because we were not working in an ideal environment, and the system is also not perfect in terms of hardware delays and placing timestamps. Since the system suffers from large time delays due to misplaced timestamps, hardware delay, and low

precision, we must increase the accuracy of our system outside of the system. This will be done after gathering the results during which initial values, offsets, and standard deviations will increase the accuracy.

5.3 Localization Test Results

During testing, it became clear that there was significant clock drift between the transmitter and receiver. By finding the slope of a line of best fit to the raw data, shown in Figure 29, we are able to compensate for the drift.

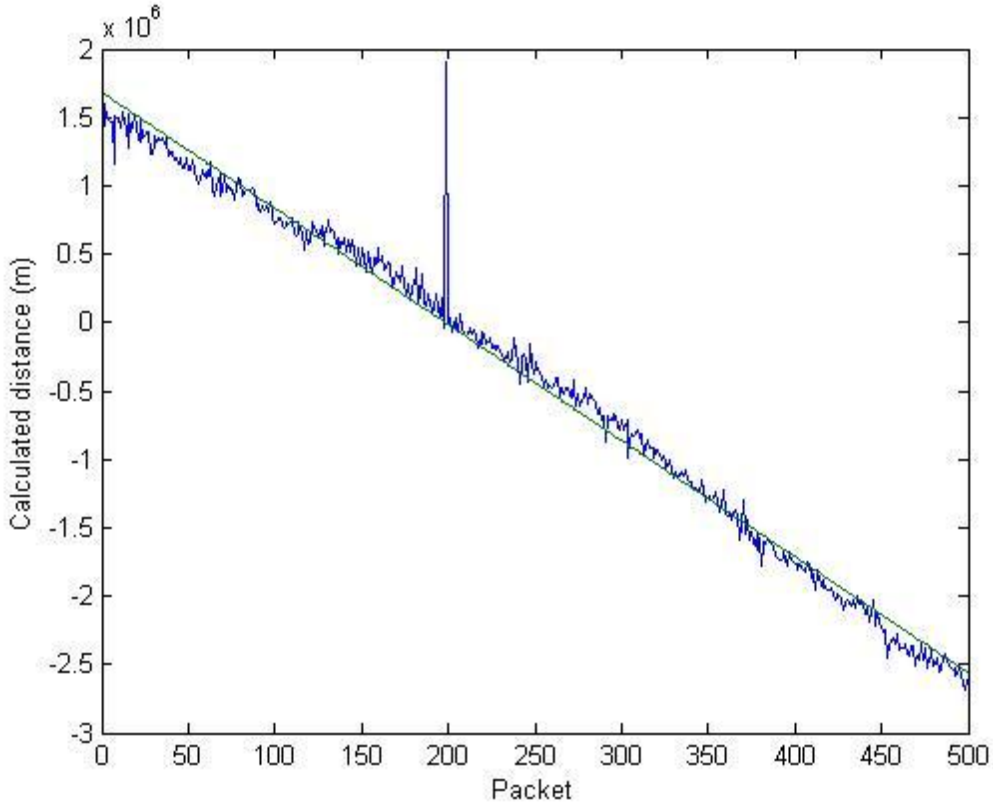


Figure 29: The raw data with best-fit line, clearly showing clock drift

To correct the clock drift, the raw data is added to a function of the slope of the line of best fit as shown in Equation 10.

$$newdata = rawdata + slope * index$$

Equation 10: Multiplying raw data by slope of beste fit line to normalize

The raw data is added to the slope of the best fit line multiplied by the packet number. The result of this is new data being approximately constant. However, the data is still very inaccurate due to a large timing offset. An average offset can be calculated and subtracted from the data in order to increase accuracy. The result of this is shown in Figure 30.

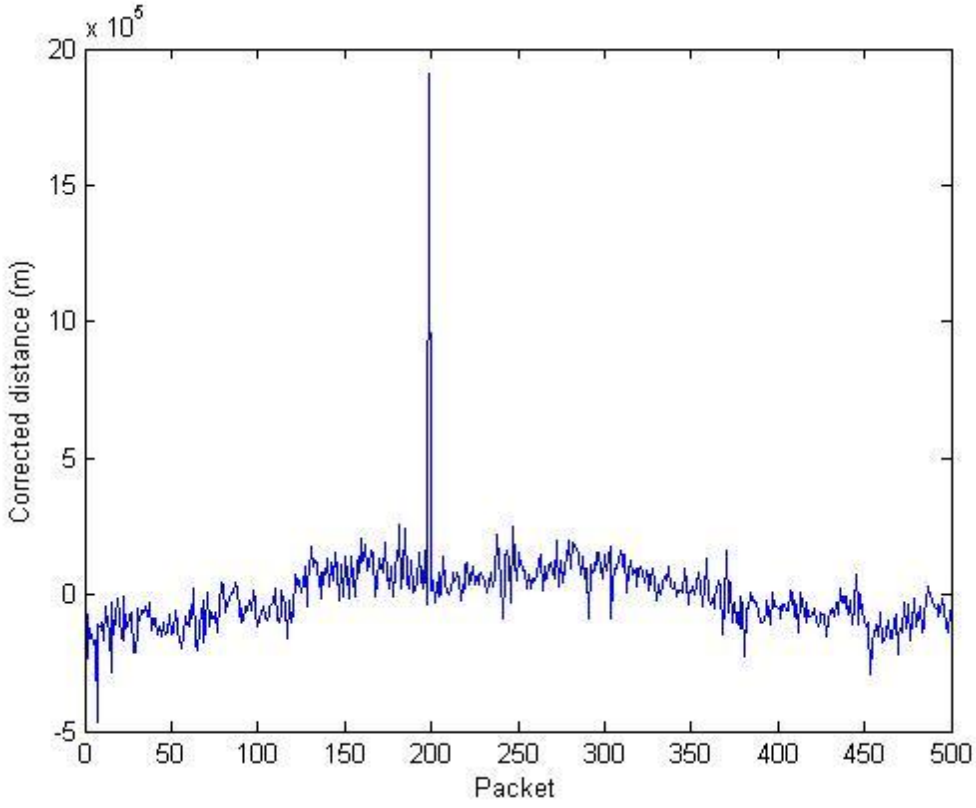


Figure 30: Normalized data to correct for clock drift to be made nearly constant

Now that the data has been corrected for clock drift and timing offset, each data point for individual packets remains to be possible very inaccurate, but the average of a large sample of data will produce a better result. After this is repeated for each of the three sensors, we have

three calculated distances, along with the known coordinates of the sensors. Plugging these values into the localization algorithm yields the calculated coordinates of the transmitter. Figure 31 shows the location of each sensor (circle), the calculated radius from each sensor, the calculated location of the transmitter (+), and the actual location of the transmitter (*).

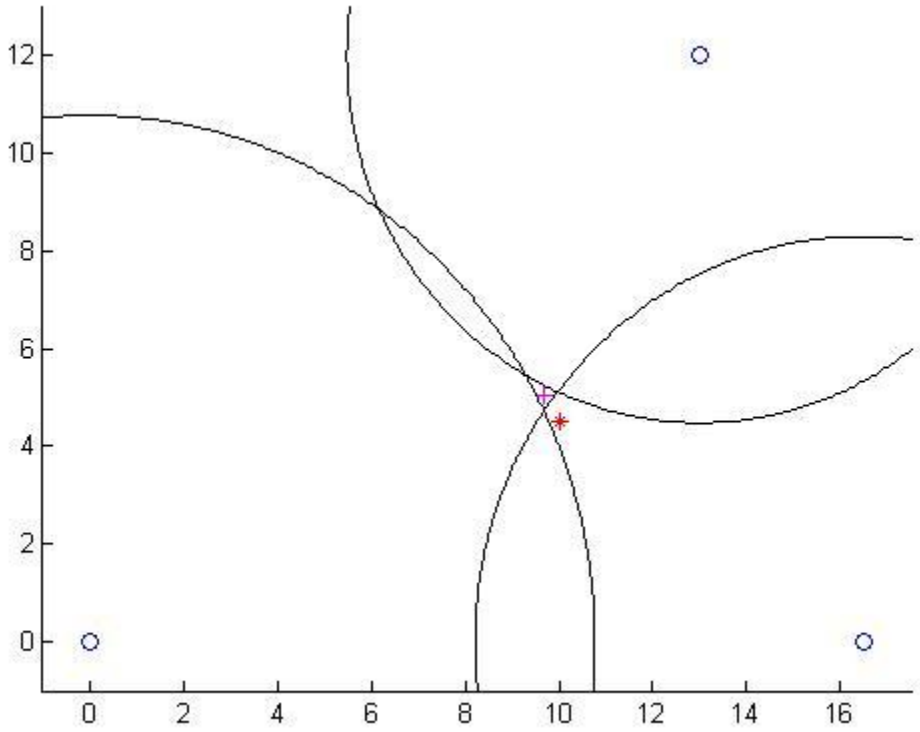


Figure 31: Localization test results showing sensor location and known and estimated transmitter locations

5.4 Summary

Although our raw data was not very accurate, we were still able to use our custom database and solver to set offsets and initializations to correct for any delays and other errors. However inaccurate our data may be, combined with our research and testing, we are able to come to several conclusions on how to advance in this field, particularly in the pursuit of accurate high-precision timestamp data.

6 Conclusions and Recommendations

In conclusion, we designed a wireless localization solution for indoor environments using both ToA and RSSI propagation methods. Initial measurements proved to be very inaccurate, however through post processing of the raw test data, and through classifying the environments through unique identifiers, we were able to learn about the propagation characteristics of our environment and create a trained database to update calibration parameters within our modified code that significantly improved the result of localization in the test environment. The system was fully compatible with Ettus Research's USRP2's, which through adding modular coding modifications to the BBN80211 code we could provide a foundation for localization on SDR platforms.

6.1 Conclusions

Through our research, testing, data gathering, and analysis, we have come to a few main conclusions on the topic of our project.

- Nanosecond-precise timestamps are difficult to achieve
    - Our system was simply unable to provide a reliable timestamp with the precision required for accurate distance calculations.
- Time synchronization is difficult to achieve
    - Even the slightest bit of clock offset between any two sensors will result in a noticeable error in the distance calculations.
- Inserting a timestamp into outbound packets is a problem
    - This adds considerable delay between when the timestamp is created and when the packet is sent, while not accurately representing IEEE 802.11 packets.
- Time of arrival localization methods are very sensitive to error

    - As with the timestamps and synchronization, any time based method of calculating distance is very sensitive when dealing with transmissions traveling at the speed of light.
- RSSI does a considerably better, though modest, job of localizing transmitter nodes once the RSSI at a known location has been calculated.
    - An improved system might want to exploit the sensors' stationary locations to continuously generate path-loss maps for higher accuracy.

6.2 Recommendations

In order to produce an accurate and reliable indoor localization system, several main components must be addressed and improved. First, the sensitivity of timestamps proves to be the biggest obstacle in obtaining accurate distance measurements. Due to the high speed that wireless signals travel at, even a very small timing offset will cause a very large distance error. This is also a problem when considering the precision of timestamps. It is very difficult to achieve timestamps that provide the precision to accurately localize. Because of these issues, a sample counting method would be a major improvement over the timestamping method.

Synchronization is also very important when considering such a time-sensitive system. Again, it is very difficult to achieve synchronization between the transmitter and each of the sensors in order to avoid significant distance errors. Sample counting, along with the establishment of a point-to-point mesh network, could make synchronization much easier. This way, the sensors could be synchronized with each other through their own mesh network, as opposed to using some external network clock.

There is also an option for a more involved and complex localization algorithm. Our system only used a direct calculation for each individual packet. A complex algorithm that considered several packets at a time could provide more accuracy. Increasing the number of sensors in the network could also provide more data, giving a better estimate of distances and location.

6.3 Topics for Future Work

Future work could implement software for playing back saved experimental data to the RTLS or use BBN80211 as a LAN without modifying the RTLS so long as they use the same socket-based interface.

In order for our system to achieve the NRL's objectives, more research and system development is needed. A full system would employ more reliable, possibly interchangeable localization solver tools that draw from advanced research and correct for additional non-ideal conditions such as undetected direct paths. Better synchronization between the sensors would need to be implemented. Our project began work on a real-time GUI that overlays results onto a map of the environment; we advocate that future work complete that secondary project and subsequent work improve it and make that GUI ready for field deployment. In order for our system to be truly suitable for the NRL's application, the localization technique would need to be modified to handle uncooperative transmitters that do not insert any timestamp data into outbound packets. This would also improve timing accuracy, as one major source of error is the delay between timestamp insertion and packet transmission. Finally, additional and likely classified work would need to be conducted to ensure that the final system is completely secure and ready for deployment in a security setting.

Research using SDR platforms has increased dramatically in the last decade, and the full potential of these systems has yet to be fully exploited. With nearly all of the radio's parameters accessible to programmers, SDR enables both rapid prototyping as well as adaptive and cognitive approaches. Since a single SDR can be used for many different applications, SDR can potentially reduce costs through large-scale mass production. Localization remains a pivotal research challenge, remaining a high priority for countless applications. Ours is but one of many challenges that would benefit from reliable, cost-effective indoor localization. As the research becomes more mature and advances in materials and mass production lower costs, SDR and localization both have bright futures.

References

[1] Caffrey, J., Stuber, G., 1998, "Subscriber Localization in CDMA Cellular," IEEE Transaction on Vehicular Technology, Vol. 47 No. 2.

[2] Dorsey, M., 2010, "Geolocation, next phase of the socialmedia revolution, focus of June 14 workshop at WPI," http://www.eurekalert.org/pub_releases/2010-06/wpi-gnp060810.php

[3] Linnartz, J., 1995, "Scatter Function," Wireless Communication, http://www.wirelesscommunication.nl/reference/chaptr03/fading/scatter.htm

[4] Garmin, 2011, "What is GPS?" http://www8.garmin.com/aboutGPS/

[5] G. M. R. I. Godaliyadda and H. K. Garg, "Versatile Algorithms For Accurate Indoor Geolocation," National University of Singapore, Pulau Bukom, Singapore, 2009.

[6] K. Pahlavan *et. al.*, "Indoor Geolocation In The Absence Of Direct Path," *IEEE Wireless Communications*, Dec 2006.

[7] R. Exel and P. Loschmidt, "High Accurate Timestamping by Phase and Frequency Estimation", in *ISPCS 2009 International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication* , Brescia, Italy, 2009, pp. 126-131.

[8] D. Geiger, "High Resolution Time Difference of Arrival Using Timestamps for Localization in 802.11b/g Wireless," United States Naval Research Laboratory, Washington, DC., USA, 2010.

[9] P. Loschmidt, R. Exel, A. Nagy, and G. Gaderer, "Limits of Synchronization Accuracy Using Hardware Support in IEEE 1588," in *Int. IEEE Symposium on Precision Clock Synchronization for Measurement, Control, and Communication*, Ann Arbor, MI., USA, 2008, pp. 12-16.

[10] F. Gustafsson and F. Gunnarsson, "Positioning Using Time-Difference of Arrival Measurements," Linköping University, Linköping, Sweden, Oct., 2002.

[11] M. Youssef *et. al.*, "PinPoint: An Asynchronous Time-Based Location Determination System," in *MobiSys 2006*, Uppsala, Sweden, 2006, pp. 165-176.

[12] K. Wang, M. Faulkner, J. Singh, and I. Tolochko, "Timing Synchronization for 802.11a WLANs under Multipath Channels," Victoria University Centre for Telecommunications and Microelectronics, Melbourne, Australia, 2003.

[13] D. Valerio, "Open Source Software-Defined Radio: A survey on GNUradio and its applications," Forschungszentrum Telekommunikation Wien, Vienna, Austria, Rep. *FTW-TR-2008-002*, 2008.

[14] Scaperoth, D., 2005, "Cognitive Software Defined Radio"

[15] Slah, A., 2002, "An Introduction to Software Radio," http://www.signallake.com/innovation/SWRprimer.pdf

[16] Wireless Innovation Forum, 2011, "Benefits of SDR," http://www.wirelessinnovation.org/page/Benefits_of_SDR

[17] Cook, P., 2007, "Introduction to Software Defined Radio," http://www.npstc.org/meetings/IO-Cook-0802145%20NPSTCa.pdf

[18] Ettus Research. (2009, Nov. 18). *Universal Software Radio Peripheral: The Foundation for Complete Software Radio Systems* [Online]. Available: http://www.ettus.com/downloads/ettus_ds_usrp_v7.pdf

[19] Ettus Research, http://www.ettus.com/

[20] "Introducing Wideband RF Transceiver Board U-RFX," http://osdir.com/ml/discuss-gnuradio-gnu/2011-02/msg00017.html

[21] Ettus Research. (2009, Nov. 18). *USRP2: The Next Generation of Software Radio Systems* [Online]. Available: http://www.ettus.com/downloads/ettus_ds_usrp2_v5.pdf

[22] GNU Radio, 2009, "Welcome to GNU Radio," http://gnuradio.org/redmine/wiki/gnuradio

[23] "GNU Radio and multi core processors," http://www.ruby-forum.com/topic/217088

[24] P. Fuxjäger et. al., "IEEE 802.11p Transmission Using GNURadio" in the *6th Kalrsruhe Workshop on Software Radios,* Karlsruhe, Baden-Württemburg , Germany, 2010, pp. 83-86.

[25] The Comprehensive GNU Radio Archive Network, "FTW IEEE802.11a/g/p OFDM Frame Encoder", accessed October 10, 2010 at https://www.cgran.org/wiki/ftw80211ofdmtx

[26] S. Ĉapkun, M. Hamdi, and J. Hubaux, "GPS-free positioning in mobile Ad-Hoc networks," in Proc. of the 34th Hawaii International Conf. on System Sciences, Jan 2001, pp. 1-10.

[27] M. P. Wylie and J. Holtzman, "The Non-Line of Sight Problem in Mobile Location Estimation," Rutgers University Wireless Information Network Lab., Piscataway, NJ., USA., pp. 827-831.

[28] B. Alavi and K. Pahlavan, "Analysis of Undetected Direct Path in Time of Arrival Based UWB Indoor Geolocation," in *62nd Semiannual IEEE Vehicular Technology Conference,* Dallas, Texas, USA, Sep 2005.

[29] P. Sadhukhan and P. K. Das, "MGALE: A Modified Geometry-Assisted Location Estimation Algorithm Reducing Location Estimation Error in 2D Case under NLOS Environments," in *Mobile Entity Localization and Tracking in GPS-Less Environments*, Orlando, Fl., USA, 2009, pp. 1-18

[30] J. Veen and P.C.J.M. van der Wielen, "The Application of Matched Filters to PD Detection and Localization," *IEEE Electrical Insulation Magazine*, vol. *19*, no. *5*, pp. *20-26*, Sep-Oct 2003.

[31] K. Pahlavan, X. Li, M. Ylianttila, R. Chana, and M. Latva-aho, "An Overview of Wireless Indoor Geolocation Techniques and Systems," in *Mobile and Wireless Communications Networks IFIP-TC6/European Commission Networking 2000 International Workshop*, Paris, France, May 2000

[32] J. O. Smith and J. S. Abel, "Closed-Form Least-Squares Source Location Estimation from Range-Difference Measurements," *IEEE Trans. on Acous., Speech, Signal Process.*, vol. *35*, pp. 1661-1669, Dec 1987.

[33] *IEEE Standard for Information Technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Standard 802.11, 2007.

[34] G. Fischer, B. Dietrich, and F. Winkler, "Bluetooth Indoor Localization System," in *Proceedings of the 1st Workshop on Positioning, Navigation, and Communication*, Hannover, Germany, 2004, pp. 147-156.

[35] J. Heiskala and J. Terry, "Synchronization," in OFDM Wireless LANs: A Theoretical and Practical Guide. USA: Sams Publishing, 2002, ch. *2*, pp. 49-56.

[36] A. Wyglinski and D. Geiger, private communication, Sept. 2010

[37] The Comprehensive GNU Radio Archive Network, "BBN 802.11 receiver," accessed October 10, 2010 at https://www.cgran.org/wiki/BBN80211.

[38] R. I. Reza, “Data Fusion For Improved TOA/TDOA Position Determination in Wireless Systems,” M.S. thesis, Dept. Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, 2000.

[39] İ. Guvenc, C. Chong, F. Watanabe, and H. Inamura, “NLOS Identification and Weighted Least-Squares Localization for UWB Systems Using Multipath Channel Statistics,” EURASIP Journal on Advances in Signal Processing, vol. 2008. DOI: 10.1155/2008/271984

[40] Langer, P., “Localization system of an autonomous mobile device,” 2010

Appendix A – Localization Math

$$r_1{}^2 = x^2 + y^2$$

$$r_2{}^2 = (x - x_2)^2 + y^2$$

$$r_3{}^2 = (x - x_3)^2 + (y - y_3)^2$$

$$r_1{}^2 - r_2{}^2 = (x^2 + y^2) - ((x - x_2)^2 + y^2)$$

$$= x^2 - (x - x_2)^2$$

$$= x^2 - (x^2 - 2x_2x + x_2{}^2)$$

$$= x^2 - x^2 + 2x_2x - x_2{}^2$$

$$= 2x_2x - x_2{}^2$$

$$r_1{}^2 - r_2{}^2 + x_2{}^2 = 2x_2x$$

$$r_1{}^2 = \left(\frac{r_1{}^2 - r_2{}^2 + x_2{}^2}{2x_2}\right)^2 + y^2$$

$$r_1{}^2 - x^2 = y^2$$

$$\boldsymbol{x = \frac{r_1{}^2 - r_2{}^2 + x_2{}^2}{2x_2}}$$

$$\boldsymbol{y = \frac{r_1{}^2 - r_3{}^2 + x_3{}^2 + y_3{}^2}{2y_3} - \frac{x_3 x}{y_3}}$$