DESIGN AND IMPLEMENTATION OF A DISTRIBUTED TDOA-BASED GEOLOCATION SYSTEM USING OSSIE AND LOW-COST USRP BOARDS

Michael S. Meuleners

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

Master of Science
In
Electrical Engineering

Jeffrey H. Reed, Chair
Carl B. Dietrich
Luiz A. DaSilva

May 2, 2012
Ballston, VA

Keywords: SDR, OSSIE, TDOA, Geolocation

DESIGN AND IMPLEMENTATION OF A DISTRIBUTED TDOA-BASED GEOLOCATION SYSTEM USING OSSIE AND LOW-COST USRP BOARDS

Michael S. Meuleners

ABSTRACT

The Software Communications Architecture (SCA) specification defines a framework that allows modular software components to be developed and assembled to build larger radio applications. The specification allows for these components to be distributed among a set of computing hardware and to be connected by standard interfaces. This research aims to build a spatially distributed SCA application for the Open Source SCA Implementation: Embedded (OSSIE) implementation using low-cost Universal Software Radio Peripheral (USRP) hardware. The system collects signals from multiple spatially distributed collection devices and use those signals to compute precision estimates for the location of emitters using time difference of arrival (TDOA) computations. Several OSSIE components and tools are developed to support this research. Results are presented showing the capabilities of the geolocation system.

Acknowledgements

I would like to thank the faculty of Wireless@VT, especially my advisors Dr. Jeff Reed, Dr. Carl Dietrich, and Dr. Luiz DaSilva for their support in completing this thesis research. I would also like to thank contributors to both the OSSIE project at Virginia Tech and the GNU Radio project for providing such useful tools that helped greatly in the completion of this research.

I would like to thank my parents for allowing me to leave radio hardware at their house for an extended period of time to assist in collecting data for this research.

Finally, I would like to thank my beautiful wife, Melissa, for her patience and love. I spent many hours working on this research while also working a full-time job, leaving little time for us. I am truly thankful for her patience.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADC	Analog to Digital Converter
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
CF	SCA Core Framework
CORBA	Common Object Request Broker Architecture
DAC	Digital to Analog Converter
DAS	Device Assembly Sequence
DCD	Device Configuration Descriptor
DMD	Domain Manager Configuration Descriptor
DPD	Device Package Descriptor
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
GPS	Global Positioning System
GPSDO	GPS Disciplined Oscillator
GUI	Graphical User Interface
GWT	Google Web Toolkit
IDL	Interface Definition Language
IF	Intermediate Frequency
JAMA	Java Matrix Package
JPEO	Joint Program Executive Office
JTRS	Joint Tactical Radio System
OE	SCA Operating Environment
ORB	Object Request Broker
OS	Operating System
OSSIE	Open-Source SCA Implementation: Embedded

PDF	Probability Density Function
PGA	Programmable Gain Amplifier
PRF	Properties Descriptor
RBDS	Radio Broadcast Data System
RDOA	Range Difference of Arrival
RF	Radio Frequency
RX	Receive
SAD	Software Assembly Descriptor
SCA	Software Communications Architecture
SCD	Software Component Descriptor
SDR	Software-Defined Radio
SNR	Signal-to-Noise Ratio
SOI	Signal of Interest
SPD	Software Package Descriptor
TDOA	Time Difference of Arrival
TOA	Time of Arrival
TX	Transmit
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
VPN	Virtual Private Network
XML	Extensible Markup Language

1 Introduction

This research aims to build a spatially distributed radio system to perform Time Difference of Arrival (TDOA) based geolocation of a specified signal of interest utilizing the Internet as a network to connect Software Defined Radios (SDR). This section provides background material on SDR, summarizes contributions of the thesis, and provides an overview of the rest of the document.

1.1 Background

In the past, radio systems were largely developed for a specific purpose. A radio might be defined to support the 802.11(b) standard and implemented in an application-specific integrated circuit (ASIC) or other piece of purpose-specific hardware. If a new standard was defined, such as 802.11(g), a new piece of hardware is required to implement that standard. SDR defines a class of radios that are largely implemented in, or controlled by software [1]. The definition of SDR is a large topic of discussion in the radio community, and can have many different definitions depending on who is asked. While one might consider any hardware aside from a digitizer to be contraband in the world of SDR, another may allow for a complex configurable RF front-end prior to the digitizer. For the purposes of this research, an SDR is defined as "a radio that is substantially defined in software and whose physical layer behavior can be significantly altered through changes to its software," as defined by [1]. This definition is important because it allows for a radio system to be built with maximum flexibility through use of easily reprogrammable general-purpose processors (GPP), while also creating a realistic design without the requirement for extremely high-speed digitization hardware. One radio that meets this definition is the Universal Software Radio Peripheral (USRP) [2]. This piece of hardware is a modular design incorporating a digitizer and a set of pluggable RF front-ends of varying capability and complexity. The flexibility gained with an SDR is not without its drawbacks. With more flexible processing come possible increases in required processing power. This deficiency has become less important recently with increased performance and lower cost of commodity computer hardware.

In an SDR application, once a hardware solution has been chosen, software must be written to control that hardware and utilize the signals that have been digitized. Devices such as the USRP provide a set of libraries for controlling the radio and sending and receiving signals. Software can then be written to take those signals and apply any processing required. Several frameworks have been defined to facilitate building SDR applications. A software framework is "a universal, reusable software platform used to develop applications, products and solutions," as defined by [3]. A software framework is important for developers who wish to concentrate on building software signal processing components and applications and allow the framework to provide a means to connect those components. Some frameworks also provide libraries with useful collection of signal processing components and utilities to aid in software development.

Several frameworks exist or are being developed. GNU Radio is a popular tool for hobbyists and researchers to use when prototyping software defined signal processing systems. GNU Radio provides a simple framework for connecting software components and building SDR applications [4]. Another framework that is available is the Software Communications Architecture specification, which was defined for the Joint Tactical Radio System (JTRS) by the Joint Program Executive Office (JPEO) [5]. JTRS is a radio system that was designed to be implemented by the next generation radio used by the US Military [6]. The SCA specification has several implementations, including several commercial and open-source implementations. The SCA specification is built upon the Common Object Request Broker Architecture (CORBA), which provides support for distributed and multiplatform computing [5]. The ability do define a distributed application is important not only because it allows distribution of processing requirements over a set of hardware, but also because it allows for those components to be distributed over a wide area network so that some components can be placed as close as possible to their corresponding RF hardware. The signal can then be processed accordingly, possibly resulting in a smaller amount of data being transmitted over a slow or unreliable network.

The increasing flexibility of radio systems through implementation in software allows for radio systems to change quicker today than ever possible in the past. A single radio device such as a cell phone or tablet may be able to support many different communications standards with a single RF/Digital front-end. A handset could be built to support both GSM and CDMA, both utilizing the same RF front-end, with the rest of the physical layer being implemented in software. These possibilities allow for standards to be created and updated faster, introducing many more types of signals into the already crowded world of radio communications.

In many cases, it is useful to have the ability to locate an RF signal. This can be useful for mapping the location of a set of transmitters in an area of interest, or to locate a radio of a user who is in distress. Because of the increasing variability and complexity of radio systems, an SDR is a great platform for signal collection for performing geolocation. The flexibility of the systems allows for a wide variety of signals to be collected, both in frequency and in signal type. Since most of the system can be defined or controlled by software, it is possible to take advantage of having access to the signal prior to and after demodulation and processing. There are a number of challenges with generating a usable location estimate from an SDR, especially a low-cost device such as a USRP. For this research, TDOA geolocation was considered. Precision and accuracy in time tagging of the data are very important for TDOA geolocation. Signal-to-noise ratio (SNR), bandwidth, and integration time are also important factors, and directly affect the error in the TDOA measurement [7].

1.2 Contributions

Several contributions will be made to the OSSIE framework as a result of this research.

- OSSIE Bug fixes – Several bugs were found in the OSSIE framework precluding the deployment of multiple nodes utilizing the USRP board.
- OSSIE CORBA Event Service implementation – The OSSIE Core Framework was modified to allow CORBA Event Channels to be connected to user developed software components. This allowed for easier development of de-coupled command and control for OSSIE software applications.

- OSSIE Java Support – Several OSSIE software components were built for the Java programming language. This includes a Java WavLoader, for loading and controlling OSSIE applications from Java. Support for Java OSSIE *Resources* was added to allow for developers to build software components using the Java programming language. Several OSSIE *Ports* were also implemented in java to facilitate connecting Java OSSIE *Resources* with other *Resources*.

The OSSIE TDOA geolocation application developed for this research will also be made available for anyone interested in continuing this research.

1.3 Thesis Organization

This thesis starts in section 2 by introducing the technologies and key concepts that are important for designing and building distributed SDR applications implemented in OSSIE. The Software Communications Architecture (SCA) framework and OSSIE implementation of the SCA are introduced. In addition, the hardware platform utilized for this research is introduced, as are concepts related to using that hardware to achieve the goal of this research. In section 3 the components and software built for this research are described. Components for collecting the signal, correlating the signal to generate TDOA measurements, and using those measurements to locate an emitter were presented. These concepts are then combined to build an OSSIE Application. The results gained from this exercise are then presented in section 4 and conclusions are drawn and future work is suggested in section 5. Important code implementations are included in the appendices.

2 Contributing Technologies and Concepts

Many software and hardware components were utilized in the execution of this research. The following technologies and concepts were considered vital in completing the research presented in this paper. SDR is introduced, including the frameworks utilized for completing this research. The signal of interest used in this research is then introduced. Finally, the collection platform used by this research is introduced. Because this research focuses on receiving and processing a signal, signal reception will be the focus of this discussion.

2.1 Software Defined Radio

SDR has been used to describe many different types of systems. In its ideal form, it is nothing more than a digitizer connected to an antenna, with all other components implemented in software. This is very convenient in theory, but at least for the moment, it is not practical in reality. There are many reasons for this, but mainly it is a result of limitations and cost of hardware. An ideal hardware system for software radio would be required to cover the range of all possible signals that could possibly be collected with that device [1]. These signals can range from DC all the way up to many gigahertz in frequency. According to Nyquist, an analog to digital converter to cover this range would be required to sample at least twice the highest bandwidth of the collected signal. Not only is a digitizer of this caliber very expensive, if not impossible to acquire, but once captured, the signal would be sampled at such a high rate that it may be impossible to process in real time. Another important factor is dynamic range, because it defines the highest and lowest power signals that can be digitized [1]. For an ideal SDR to operate, it would be required to capture the lowest and highest power signals in its coverage range. This may result in reduced dynamic range and increased quantization error for lower power signals in a band where higher power signals exist. For these reasons, a typical software defined radio employs a number of signal conditioning steps prior to digitization. These steps can include [1]

- Signal amplification/attenuation, to scale the analog signal to match the capabilities of the digitizer,

- Filtering, to reject out of band signals to prevent aliasing during digitization,
- Down conversion, to shift the frequency range of the desired signal down to something that the digitizer can sample. This can be to an intermediate frequency (IF) or directly to baseband with possibly several stages in-between.

The USRP SDR system used by this research employs all of these components [4]. For this research the definition of an SDR is a radio system where much of the system is implemented or controlled by software. It is a system that is capable of multiple modes of operation because its input and output behavior is determined by software.

2.2 Software Communications Architecture

The Software Communications Architecture (SCA) defines a portable, open software framework for SDR applications and components that allows for SDR applications to be built with a re-usable, common set of components [5].

According to the Software Communications Architecture Specification document [5], the SCA has been designed to:

i. "Provide for portability of applications software between different SCA implementations,
ii. Leverage commercial standards to reduce development cost,
iii. Reduce software development time through the ability to reuse design modules,
iv. Build on evolving commercial frameworks and architectures."

These requirements were achieved by defining a standards-based, modular platform, ensuring that components developed on different platforms and programming languages can interoperate together without a great deal of time spent facilitating that interoperability by the developer [5].

2.2.1 Common Object Request Broker Architecture

The Common Object Request Broker Architecture (CORBA) is an Object Management Group (OMG) middleware standard that allows for applications written in different programming languages and for different platforms to interact together executing on distributed processors and platforms [5]. It defines an Interface Definition Language (IDL) to define interfaces between components. Software components developed for use with CORBA implement these interfaces, and the Object Request Broker (ORB) provides the plumbing to connect these interfaces and pass information between software components [5].

2.2.2 SCA Core Framework

The Core Framework (CF) defines a set of CORBA interfaces defined for different components in an SCA system. The Framework Control Interfaces include the *DomainManager*, *DeviceManager, Application,* and *ApplicationFactory* interfaces. These components are responsible for installing, managing, and uninstalling software from the system [5]. The Framework Services interfaces include the *File*, *FileSystem, and FileManager* interfaces. These interfaces provide access to the file system and other services [5]. The Devices interfaces include *Device*, *LoadableDevice*, *ExecutableDevice*, and *AggregateDevice*. These interfaces provide an API for interaction with devices [5]. Finally, the CF provides a set of Base Application Interfaces, including *Port, LifeCycle*, *TestableObject*, *PropertySet, PortSupplier, ResourceFactory,* and *Resource*, that provide an API for system software components, including *Resources* and *Devices* [5]. The *Resource* interface is important for this research because it is the base class for every software component written to support this research including the *USRP_TIME Device* and each of the signal processing components. An implementation of the SCA must provide implementations of most of these interfaces [5].

2.2.3 SCA XML Configuration Files

Every component in an SCA system has a set of one or more configuration XML files, whose purpose is to describe the capabilities and requirements of each component [5]. These files are:

- The Software Package Descriptor (SPD) describes the implementation of a software component,
- The Software Component Descriptor (SCD) describes a software component, including what interface *Ports* the component shall *Use* or *Provide*,
- The Software Assembly Descriptor (SAD) describes the components and connections which make up an SCA waveform, as well as waveform-specific property values,
- The Properties Descriptor (PRF) describes a set of properties that a software component has, including its default values,
- The Device Package Descriptor (DPD) describes a *Device*, including its make and model,
- The Device Configuration Descriptor (DCD) identifies the *Devices* associated with an instance of a *DeviceManager*,
- The Domain Manager Configuration Descriptor (DMD) describes the configuration for the *DomainManager*.

2.2.4 SCA Operating Environment

The SCA Operating Environment (OE) is the set of services that an SCA implementation provides to users of the framework. The OE includes the POSIX Operating System (OS), CORBA Middleware, CORBA Naming Service, CORBA Log Service, and CORBA Event Service [5].

2.2.5 Applications

Each *Application* (Waveform) that can be installed in an SCA system implements the *Application* interface. Each *Application* has one or more *Resources* that act together to perform a function. The *Application* has a SAD file to describe its *Resources*, their configured properties and the *Port* interconnections [5].

2.2.6 Resources

Each software component that is to be used in an *Application* implements the *Resource* interface. This includes both *Device* and signal processing *Resources*. Each *Resource*

has a PRF, a SCD, and a SPD file, to describe its properties, its capabilities, and its implementations [5].

2.2.7 Ports

The SCA defines a *Port* interface for data flow between *Resources*. The *Port* defines an API for connecting and disconnecting ports. A software component can both provide ports and use ports. The provider of a *Port* provides an implementation of that *Port*. The user of a *Port* can call the provided methods associated with the port. For this reason, the SCA specification refers to the *Ports* differently depending on what side of the port the component sits. If the component sits on the user side, it is known as a *Uses Port*. If it sits on the provider side, it is known as a *Provides Port* [5].

2.2.8 CORBA Event Service

The SCA specification requires that each implementation implement the CORBA Event Service as part of its OE. The implementation of the CORBA Event Service must implement both the *PushSupplier* and *PushConsumer* interfaces defined in the *CosEventComm* module. The SCA uses the CORBA Event Service by allowing a set of event channels to be set up to allow decoupled connectivity between software components. The SCA Specification indicates two channels that must be defined, the Incoming Domain Management Channel (IDM_Channel) and the Outgoing Domain Management Channel (ODM_Channel). Other event channels may be created and connected to software components as seen fit [5].

2.3 OSSIE

Open Source SCA Implementation: Embedded (OSSIE) is an implementation of the Software Communications Architecture (SCA) specification developed by Dr. Max Robert and a team of Dr. Jeff Reed's students at Virginia Tech [8]. OSSIE is a free and open source SCA implementation that was built initially because of high costs associated with commercial implementations of the SCA. OSSIE has seen continued development by students at Virginia Tech and has received interest from industry [8].

While OSSIE implements a large portion of the SCA specification, it is missing some important features. OSSIE does not support automatic allocation of resources. For this reason, OSSIE defines an additional XML file, the DAS (Device Assembly Sequence) file, for each application that assigns each software component to the *Device* where it will be executed. OSSIE also lacks the ability to allow more than one of the same *Device* nodes to be used within one *DomainManager* at the same time. The most important missing feature that was required in this research was support for the CORBA Event Service for software components to allow decoupled command and control of an *Application*.

OSSIE provides a limited set of included *Port* implementations for connecting software components. It provides a set of generic data *Ports* for streaming signal data, and a second set of the same *Ports* allowing metadata to be transmitted. In general, an OSSIE *Uses Port* sends data across an interface and an OSSIE *Provides Port* receives that data. For the purposes of this research, where *Ports* are used or built, the terminology will be used in the same manner.

2.4 GNU Radio

GNU Radio is another open source framework for building experimental SDR applications. While GNU Radio does not implement a standard like OSSIE, it also allows a developer to build up signal-processing applications using modular components. GNU Radio connects and configures components in a Python application known as a "flow graph." Components can be written in several languages, including C++ and Python [4]. This allows for flexible applications that can be modified and quickly executed. GNU Radio also provides several out-of-the-box tools that can be used to visually build a flow graph and other tools that can be used analyze spectrum and verify that the radio is working. For this research, this was the part of GNU Radio that was most helpful. The FFT tool and waterfall tool were used to investigate what signals were available and to choose the proper signals for use in this research. Also used was libusrp, a library included with GNU Radio providing an API to interface with the USRP Radio.

This API allows parameters to be set that control the operation of the USRP and its daughterboards. The API also allows data to be streamed to and from the USRP [4].

2.5 FM Radio

For this research, VHF FM Radio broadcast signals were chosen as the signal of interest (SOI) for collection and location determination. This signal was chosen because it is consistently available and easy to collect. In a standard analog FM radio signal, the entire signal data is contained within an FM modulated signal. Once demodulated, the one-sided spectrum of an FM signal is approximately 100kHz wide. Figure 1 shows the spectrum of the FM signal after demodulation. In the audible frequency range, the signal contains the mono audio channel (left + right) between 30hz and 15kHz. At 19kHz, there is a pilot tone indicating a stereo audio channel (left – right) is centered at twice this frequency, 38kHz. The FM signal may optionally contain several auxiliary channels between 53kHz and 99kHz. Radio Broadcast Data System (RBDS), if present, is centered at 57kHz. Centered at 67.65kHz is DirectBand, or a subcarrier containing secondary content (SCA). A second secondary content subcarrier can sometimes be found at 92kHz [9]. When demodulating the audio for mono audio playback, the demodulated signal can be filtered to reject anything above 15kHz, but for the purposes of this research, the time-varying components above 15kHz were kept to aid in the correlation phase by keeping as much time-varying information as possible in the signal to maximize the bandwidth of the signal.

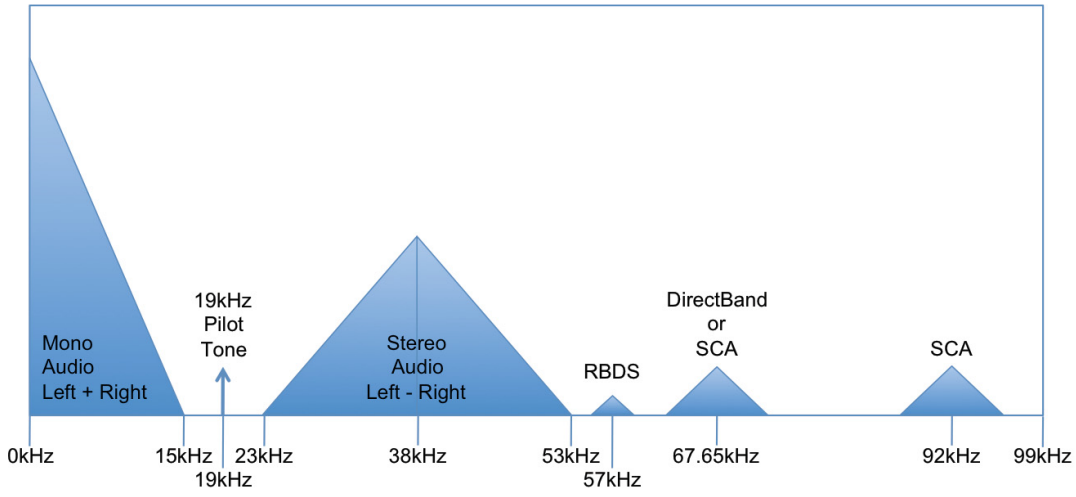

Figure 1 FM radio spectrum

Figure 2 shows an actual averaged FFT captured using an application built with the GNU Radio Companion to demodulate the FM Radio signal and generate an FFT.

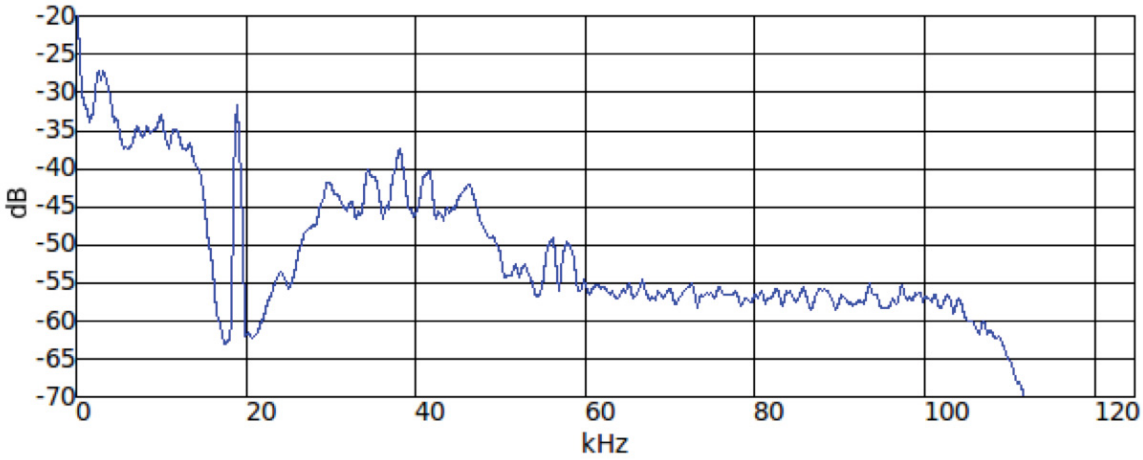

Figure 2 FM signal averaged FFT after demodulation

2.6 Universal Software Radio Peripheral

The hardware platform chosen for this research is the original Universal Software Radio Peripheral (USRP), also known as the USRP1, since newer USRP boards have since been released [4]. The USRP is an open source experimental SDR radio developed by Matt Ettus of Ettus Research [4]. The USRP was chosen for its low-cost and for its availability for this research. The USRP provides capability for both receiving and transmitting digital signals. It is built with a modular design, allowing different receive (RX) and transmit (TX) front-ends to be installed on its four daughterboard slots. The USRP has four A/D converters for signal digitization, four D/A converters for signal transmission, an Altera FPGA chip for digital processing, and a Cypress FX2 USB interface chip to provide a high speed USB2 interface to the host PC [4].

Two RX daughterboard slots are provided. Each slot provides two high-speed 12-bit A/D converters capable of digitizing analog signals at 64M samples per second. Prior to each A/D converter is a programmable gain amplifier (PGA) that allows boosting the analog signal up to 20dB prior to digitization to make best use of the A/D dynamic range. Depending on the RX daughterboard used, the signal is either digitized as a complex signal or a real signal and may be either at baseband, an intermediate frequency (IF), or at the original frequency. After digitization, the A/D is received by the digital inputs on the

Altera FPGA chip. The FPGA is loaded with an image that provides several signal conditioning steps prior to the data being streamed across the USB2 interface to the PC. The signal is first multiplied by a constant frequency signal to shift the signal to baseband, and then the signal is decimated. The in-phase (I) and quadrature (Q) samples are then interleaved and transmitted over the USB2 bus to the PC [4].

The USRP board also provides two TX chains for transmitting signals. The TX process is much the same as the RX process, except in reverse. The interleaved I and Q samples are received by the FPGA chip over the USB bus then interpolated and up-converted to IF. Two high-speed 14-bit D/A converters are provided for each TX chain, capable of converting signals at 128M samples per second. After up-conversion to IF, the digital signal is passed through the D/A converter and run through a PGA providing up to 20dB gain. The analog signal is then delivered to the TX daughterboard through the provided connector where it can be mixed and filtered to the proper band [4].

The behavior of the daughterboards and the FPGA components for both the TX and RX chains are completely programmable over the USB bus.

The RF daughterboard chosen for signal reception in this project is the WBX board. The WBX is a transceiver, having a RX/TX capability from 50-2200MHz [10]. It provides on-board analog filtering and mixing to reject out of band interference before digitization. Its mixers down-convert the signal to IF and also separate the incoming signal into I and Q components such that the signal is digitized as a complex signal. The WBX was chosen for its wide range and coverage around the FM radio band.

The USRP board has a number of limitations, which are important for solving a geolocation problem. First, in order to provide an accurate timestamp for each sample recorded, each radio system must have access to a very accurate time source. If a host computer could provide an accurate time source, the signal could be time-tagged on the host. The arrival time of each packet on the PC is not deterministic due to the method in how the data is transmitted from the FPGA to the PC. The data is loaded into a buffer on

the FPGA and transmitted via the USB interface when possible. Because of this the jitter in arrival time of data on the host is too great. The USB bus is also too slow to transmit data to the PC at the full rate of 64MS/s for maximum timing resolution. The signal must be decimated prior to transfer, so not only is the jitter too great, but the time resolution is also too great to generate an accurate timestamp. The USB2 bus can handle a maximum of 480 Mbit/sec. At 64MS/sec, the USRP transmits

$$64\frac{Msample}{\text{sec}}32\frac{bits}{sample}=2048\frac{Mbit}{\text{sec}} \tag{1}$$

which is well above the theoretical maximum for the USB interface. Decimating by 8 brings the rate down to a more manageable 256Mbit/sec. If accurate timing could be attained using a PC at this rate, corresponding to a sampling rate of 8MS/sec, the time resolution would be

$$\frac{1}{8\frac{Msample}{\text{sec}}}=1.25\cdot10^{-7}s \tag{2}$$

during which time a collected signal travels

$$1.25\cdot10^{-7}s\cdot299792458\frac{m}{s}=37.5m \tag{3}$$

which might be acceptable for some targets and requirements for geolocation accuracy but not for others. Therefore, a method of time-tagging the samples coming off of the USRP board is used and is described in section 3.1.2.

Another problem with the USRP board is that there is some tolerance in the oscillator clock rate, such that a signal collected at two separate USRP boards will not be sampled at precisely the same rate. Because the signals are not collected at the same rate, correlating the data collected from these sources will not be possible to a high degree of precision. An arbitrary ratio resampler is described in section 3.4.5 to compensate for this issue.

3 Design and Implementation

This research required many building blocks to assemble a fully working TDOA geolocation system. The network and hardware infrastructures are introduced. This is followed by a discussion of hardware modifications required. The OSSIE framework modifications are then described. Finally, the OSSIE waveforms and components are described.

3.1 Hardware

3.1.1 Distributed Hardware

Two OSSIE nodes were built, one placed in Haymarket, VA and the other placed in Ashburn, VA. Each OSSIE node contained a single GPP and a single USRP board. The USRP boards were connected to 75-ohm antennas through the RG-6 wiring already present in each house, via a 50/75-ohm converter, to avoid any unnecessary interference. A Trimble Thunderbolt GPS Disciplined Oscillator (GPSDO) and antenna were installed in each house, also connected to their antennas through the house RG-6 wiring. The GPP *Devices* were connected to the LAN in either house and the house LANs were linked via a VPN connection installed for this purpose, as seen in Figure 3.

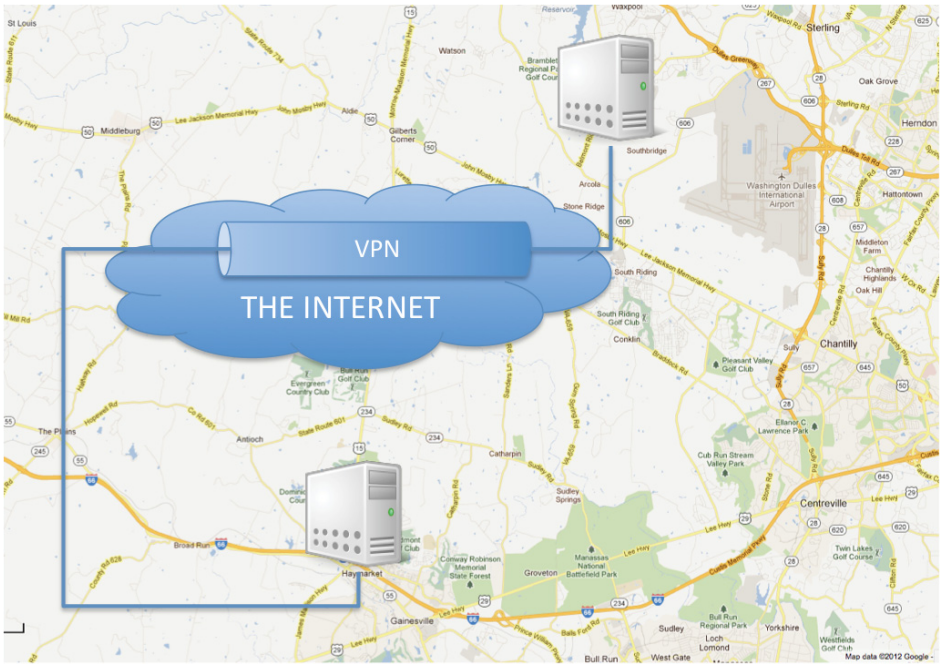

Figure 3 VPN between Ashburn and Haymarket

3.1.2 USRP Precision Timing

The USRP board in its default configuration serializes only signal data over the USB bus. Because the research presented requires precision time tagging to facilitate accurate TDOA measurements, an accurate time source had to be connected to the USRP board. Earlier in the development of the GNU Radio project, there was an effort to build an FPGA firmware version that transmitted metadata in-band with the signal data over USB. This endeavor was abandoned, but the Verilog codebase still exists [11]. The Verilog code maintains an internal clock and tags each packet sent to the host with a timestamp in samples. Because the USRP board could not be modified physically for this research, there is no way to coherently lock the on-board oscillator to an outside reference. The on-board oscillator cannot be relied upon to deliver a precise 64MS/s clock. There is a possibility, however, to connect an accurate 1PPS reference to a high speed input on a Basic RX daughter board on the unused side of the USRP board. The input can then be used to reset the clock of the USRP board at each 1PPS signal. The received signal is later resampled based on the number of clocks per second that were detected, such that the sampling rate on the data from each USRP matches. The signals are still not coherently sampled, but they should be aligned to within the period of one sample, which was deemed accurate enough for the purposes of this research. The Verilog code that was added/modified to perform this clock reset is shown in Figure 4. Figure 5 shows a picture of the USRP board with a 1PPS signal wired in.

```
wire ts_reset_in;
reg ts_reset_a;
reg ts_reset_b;

assign ts_reset_in = ~io_rx_b[15];

initial ts_reset_a = 0;
initial ts_reset_b = 0;

always @ (posedge clk64) begin
  ts_reset_a <= ts_reset_in;
  ts_reset_b <= ts_reset_a;
  if (ts_reset_a & ~ts_reset_b)
        timestamp_counter = 32'd0;
  else if (tx_dsp_reset | rx_dsp_reset)
        timestamp_counter <= 32'd0;
  else
        timestamp_counter <= timestamp_counter + 32'd1;
end
```

Figure 4 Verilog change to reset timestamp on USRP

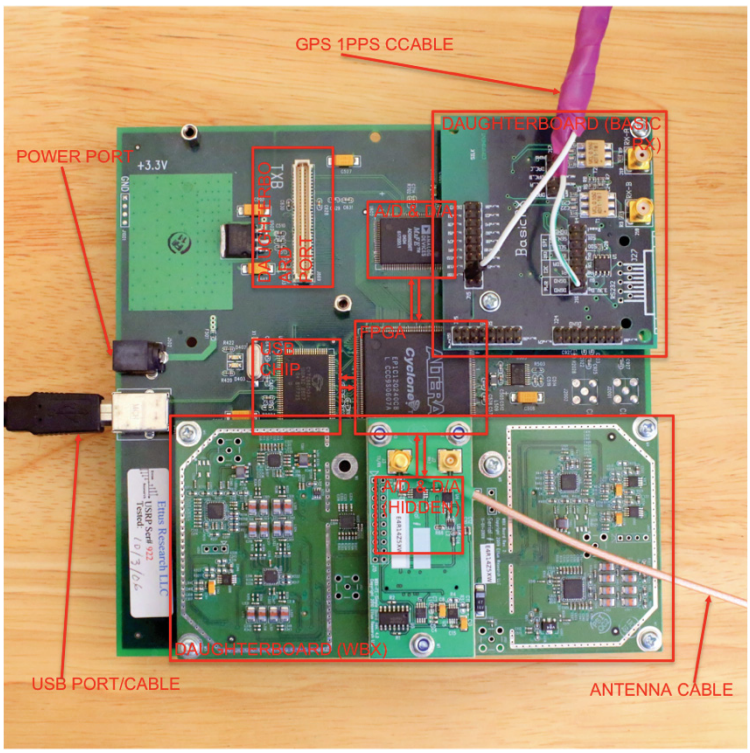

Figure 5 USRP board

3.1.3 Trimble Thunderbolt GPS Disciplined Oscillators

Trimble Thunderbolt GPS Disciplined Oscillators (GPSDO) were installed alongside each USRP board to provide the precise 1PPS signal. The signal provided by the Thunderbolt is output between 0 and 5V, where there is a precise rise in signal from 0 to 5V at the beginning of each second. Because the signal ranges from 0 to 5V, this signal would damage the USRP board if connected to the input. A voltage divider was built up to drop the voltage down to 3.125V, well within the 3.3V allowed by the USRPs high-speed inputs. A pull-down resistor with capacitor was also added to filter out high frequency transients in the input. This is the same voltage divider used on the USRP N210 board [12]. Figure 6 shows the schematic of the implemented voltage divider.

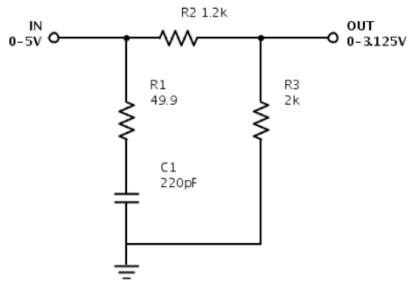

Figure 6 Voltage divider for 1PPS input on USRP

The Thunderbolt also has available a serial output that transmits a number of useful pieces of information. The most useful for this research are the time packets that indicate the current time and are broadcast at the beginning of every second. Figure 7 shows the structure of each packet coming off of the GPSDO [13].

Figure 7 Trimble Thunderbolt packet structure

Figure 8 shows the layout of the <DATA STRING> element for the timestamp packet [13]. Each row represents 32 bits, and the rows are consecutively transmitted. All fields shown of greater than one bit are encoded as unsigned integers and are carried in network byte order, where the most significant byte is carried first [13]. The timestamp defined by the packet indicates the second at which the last 1PPS signal was received. Because the Linux system could not keep reliable time, this timestamp was used as an accurate time reference for the system.

Figure 8 Timestamp packet

3.2 OSSIE Frameworks Modifications

Several modifications were made to the OSSIE framework to support this research. A set of interfaces was added to OSSIE to support distributed data transfer. The OSSIE Event channel was enabled and implemented for application components. Finally, a set of Java OSSIE Software Components was built.

3.2.1 Thesis Interfaces

To overcome the speed and latency issues found in section 3.3.1 with the OSSIE test waveform using the OSSIE-provided interfaces, a new type of *Port* that is buffered on both ends was defined in order to overcome these issues and still have the ability to reliably transmit a number of sequential packets. For this research, a new version of the *standardInterfaces complexShort* interface was created that has a buffer on both the *Uses* and *Provides* side of the connection. On the *Uses* side, packets are accumulated in a buffer until they can be sent and then buffered again on the *Provides* side. This type of interface can only be used in between a set of components that is not transmitting at all times, otherwise the buffer on the *Uses* side of the *Port* will be guaranteed to fill up if the amount of data exceeds the throughput available. It is particularly useful for transmitting snapshots of continuous data. A *TriggerControl* interface was also built to allow one component to request a snapshot of data from another component. These *Ports* were placed in a new OSSIE package called *thesisInterfaces.* The standard OSSIE *standardInterfaces Ports* were also modified to allow each *Port* to optionally transmit metadata.

3.2.2 OSSIE Event Channel Implementation

The SCA specification calls for the OE to provide an implementation of the CORBA Event Service that implements the *PushSupplier* and *PushConsumer* interfaces from the *CosEventComm* module [5]. There is such an implementation defined for use with omniORB, and that implementation is called omniEvents. In looking at the OSSIE source code, it appears as though at some point there was an intention to implement this functionality, as some of this functionality was present, but commented out. The Incoming Domain Management Channel (IDM_Channel) and the Outgoing Domain

Management Channel (ODM_Channel) were implemented partially, but capability to connect SCA *Resources* to event channels was not fully implemented.

For this research, the CORBA Event Service was enabled in OSSIE and functionality was implemented that allows connecting software components to event channels. A software component can utilize the event channel by connecting a *Uses Port* on that component to an event channel, whether or not the *Port* is publishing or subscribing to events. The *ApplicationFactory* implementation was modified such that when an *Application* requires an event channel to be connected to a component's *Uses Port*, it passes a reference to the specified event channel into the *connectPort* method of the component's *Uses Port*. The component subsequently connects a publisher or subscriber to the connection and allows the messages to be sent to or received from that event channel. This functionality was useful in this research as it allowed the command and control GUI to be decoupled from the waveform code by allowing both the waveform components and the GUI application to independently publish and subscribe to the same CORBA Event Channels. These Event Channels can be connected in the software assembly descriptor XML file using the same <connectinterface> tags used to connect other *Uses* and *Provides Ports* [5]. Figure 9 shows an example connection used in this research to connect the *Correlator* component's *EVENTS_IN Port* to the *THESIS_EVENTS* event channel.

```
<connectinterface id="DCE:89dbcca3-1ea9-11e1-b0a0-000c297957ca">
        <usesport>
          <usesidentifier>EVENTS_IN</usesidentifier>
          <findby>
            <namingservice name="Correlator_1"/>
          </findby>
        </usesport>
        <findby>
          <domainfinder type="eventchannel" name="THESIS_EVENTS"/>
        </findby>
      </connectinterface>
```

Figure 9 Event Channel Port connection

The *CosEventComm* interface indicates that events passed on the event channel must be the CORBA Any type, which means that any CORBA data type can be used as an event on an event channel [14]. For this research, several event types were defined and used, including a *TdoaEvent* that is sent whenever a TDOA measurement is made. A *TriggerCollect* event is defined to allow a trigger to be sent from the control application. A *TuneRequestEvent* is also defined to allow tuning of the USRP during operations. The CORBA IDL definition of these events is illustrated in Figure 10.

```
struct TdoaEventAvg {
float freq;
string id;
string col1id;
float col1lat;
float col1lon;
string col2id;
float col2lat;
float col2lon;
unsigned long num_collects;
long tdoa_avg;
float tdoa_std;
};
```

```
struct TuneRequestEvent {
string tuner_id;
float freq;
short decim;
};

struct TriggerCollect {
float start_freq;
float freq_interval;
short delay;
unsigned long num_channels;
unsigned long repetitions;
unsigned long duration;
};
```

```
struct TdoaEvent {
float freq;
string id;
string col1id;
float col1lat;
float col1lon;
string col2id;
float col2lat;
float col2lon;
long tdoa;
};
```

Figure 10 Thesis events

Decoupling the software components means that components do not have to care or know if other components are interacting on its event channels. If a component is generating status, it does not care if anyone is consuming that status. In the same way, if a component is listening for command and control events, it does not care who sends the event or if they are not online at any given point in time. This is useful in a large system where it may be unknown how many waveforms are online at a given point in time, where single or multiple points of command and control are desired.

3.2.3 Java OSSIE Support

OSSIE supports building signal processing components in two programming languages, the first being C++ [15]. C++ is a statically typed, multi-level programming language that combines the features of an Object-Oriented programming language with similar syntax to and interoperability with C [16]. This allows usage of many existing legacy signal-processing libraries. C++ is also a native, compiled language that is compiled to machine language before execution. This allows for very fast, efficient code that is

optimized for the operating system and hardware. Because of these traits, C++ is a great language for writing signal processing components for a SDR system.

Python, the other supported language, is a higher-level programming language than C++ and is an interpreted language. Python's emphasis is an ability to write powerful code that is very readable [17]. Python also provides a very large standard library that provides many functions, including graphical user interface (GUI) functionality [17]. Python is often used as a glue language, connecting software from different languages. Python, being an interpreted language, will not run as fast as compiled C++, so it is best left for less intensive usage in an application like OSSIE. Some possible usages are GUIs and application control software [17].

When choosing which programming language to use for this research, the choice was clear for the signal processing components in the system (C++) because of performance reasons and availability of signal processing libraries. For the user interface and web application portions, Java was the chosen language. This was done partially because of the researcher's experience with Java, but also because of the large popularity of the Java language both in user interface (UI) development and in web application development [18]. It was also done as an exercise in learning how to interact with the SCA framework from a new programming language. Java has built-in CORBA support, which is interoperable with omniOrb's naming service, making it a great choice for interacting with OSSIE [19]. Because OSSIE does not support Java, new software had to be written to allow development in Java. Several types of OSSIE software were implemented in Java for this research, a Java OSSIE *Resource* implementation, a Java WavLoader, and several *Port* implementations.

The SCA *Resource* interface and its parent interfaces, *LifeCyle*, *TestableObject*, *PropertySet*, and *PortSupplier* were implemented in Java. The Java *Resource* class is implemented in a very similar way to how the *Resource* class is implemented in C++, but extra care was made to define the software in such a way that minimal code had to be written for each software component that was built. The *Resource* class thus

implemented the property configure and query methods, to allow configuring of parameters on the *Resource*. The Java *Resource* class also implements a port management system allowing the child class to simply add and retrieve *Ports* as necessary. Because of this, when the core framework requests a *Port* with the *getPort* method on the *Resource*, the child class no longer needs to respond to this message because it is already taken care of by the provided implementation of *Resource*. Another convenience added into the system is a static method *runApplication,* intended to start the application from its main method, given parameters required to connect to the CORBA Naming Service by the *ApplicationFactory*. All threading and thread control is also built into the Java *Resource* class. Because of these added convenience methods, a component need only implement the process method, a constructor that defines and adds *Ports* to the *Resource*, and a main method to call *runApplication*.

In addition to the *Resource* class, several *Port* interfaces were implemented in Java. Two existing *Ports* from the OSSIE standard interfaces, *complexShort_p*, a *Provides Port* and *complexShort_u*, a *Uses Port*, were implemented. A *Port* from *thesisInterfaces* was also implemented, *TriggerControl_u*, a *Uses Port*. Finally, implementations of publisher and subscriber *Ports* were implemented in Java for interoperability with the CORBA Event Service.

A graphical OSSIE component was built with the capability to trigger a TDOA collection to be made using the *TriggerControl Port*. This component was later deprecated in favor of a component that would use the *THESIS_EVENTS* event channel to send trigger events to the TDOA collection waveform using the *TriggerCollect* event. This decoupled connection allows the same trigger event to trigger collections in multiple simultaneously deployed waveforms. The GUI interface for both applications can be seen in Figure 11. The GUI can trigger a set of TDOA measurements to be made, allowing the user to set the frequency channels to collect, the number of TDOA samples to take at each channel, and the integration time in samples for each collection.

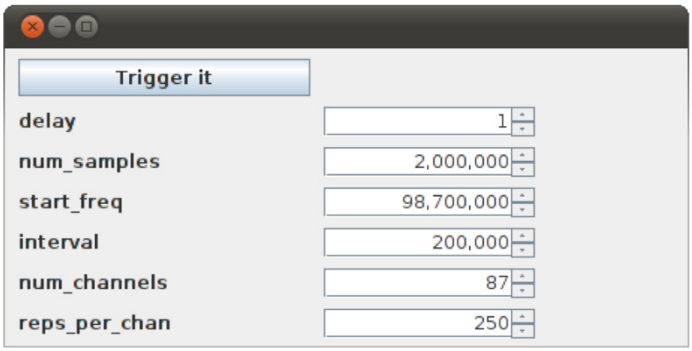

Figure 11 Command and control GUI for TDOA waveform

The other java application that was built was the Java OSSIE WavLoader application. This application provides a Java GUI allowing a user to install/uninstall and start/stop one or more waveforms. The Java OSSIE WavLoader is able to load and run multiple waveforms on the same OSSIE nodes. It also allows a user to inspect the current parameters on a loaded waveform. It can also be used as a library to programmatically control an OSSIE Domain, including starting and stopping waveforms and configuring properties of a loaded waveform. The application is shown running multiple copies of the *ossie_demo* waveform in Figure 12.

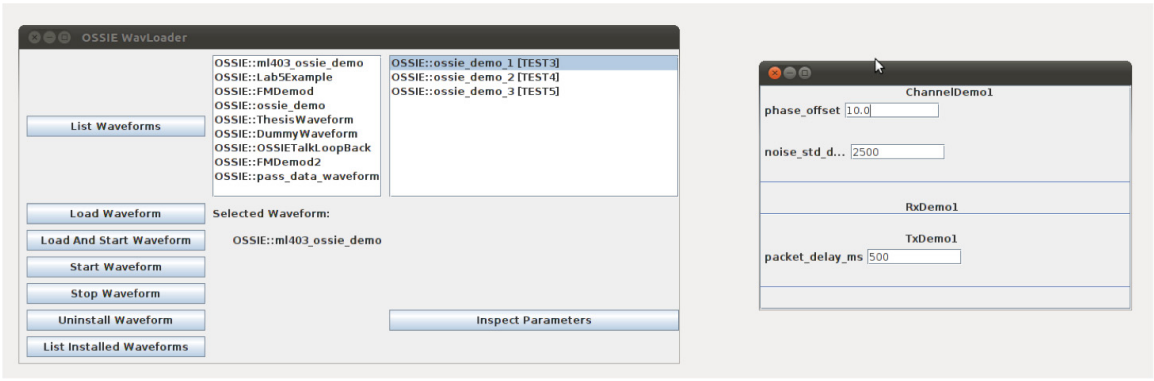

Figure 12 WavLoader Java application

3.3 OSSIE Waveforms

Two OSSIE waveforms were built. The first waveform was built to test the data transfer between the distributed hardware. The second waveform was the full TDOA collection waveform built for this research.

3.3.1 Initial Distributed Test Waveform

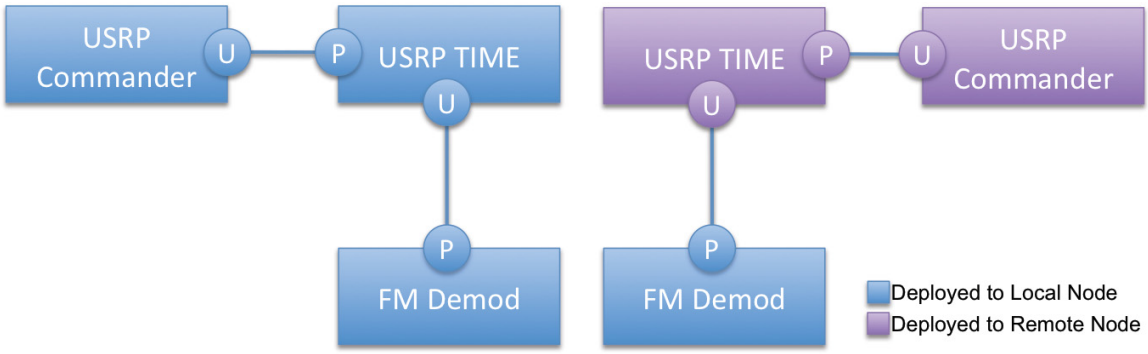

Figure 13 Initial OSSIE waveform

An initial OSSIE waveform with two USRP boards was built, as seen in Figure 13. The waveform was built simply to test whether it was realistic to transmit the required data over the VPN link. As it turned out, the speed of the network over the VPN was not fast enough to keep up with the data rate coming off of the USRP boards, not even at the maximum USRP decimation of 256 [4]. Because the packet data was buffered only at the *Provides* side of the *Port*, a bottleneck was encountered at the *Uses* side of the *Port* and packets were lost. This was a problem for a TDOA computation because the correlation procedure requires that there are no missing samples in the signal data. The *Throttler* component was built to solve this problem, and is presented in section 3.4.3.

3.3.2 Final TDOA Waveform

A final TDOA waveform was built encompassing all of the components described in section 3.4 as seen in Figure 14. Once the waveform is installed and run, it sits idle waiting for a *TriggerCollect* event to be delivered into the *Correlator* to begin the collection, demodulation, resampling, and correlation process. The *TriggerCollect* event specifies which time in the future the collection should occur, what frequency to collect on, and how many samples to collect. It also allows the requester to indicate how many trials should be run at each frequency and how many frequencies should be tested, along with the interval between frequencies. In the case of this research, the *TriggerCollect* event was sent from the decoupled Java command and control GUI.

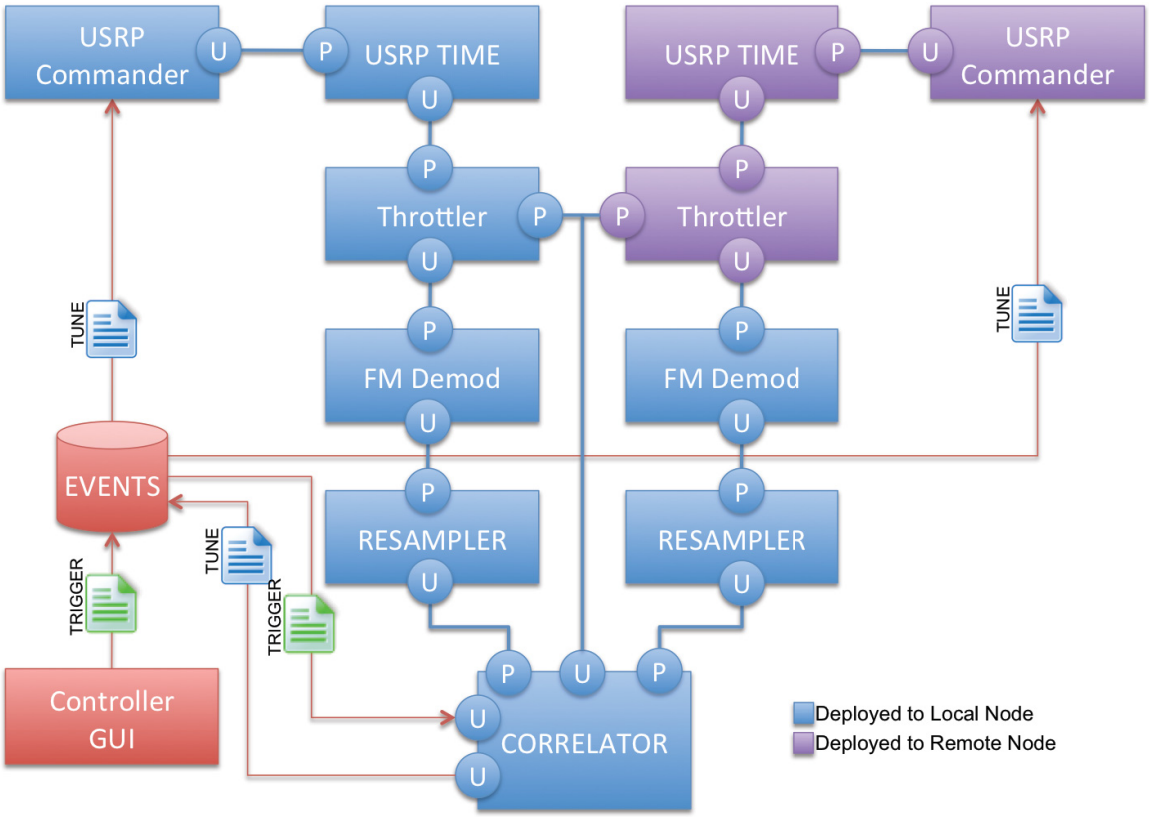

Figure 14 Final TDOA waveform

3.4 OSSIE Components

Several OSSIE Components were built to collect, transmit, demodulate, resample, and correlate signal data to produce TDOA samples.

3.4.1 USRP_TIME Component

Because the data coming off of the USRP board contains header data in-band with the signal data, a new USRP OSSIE component had to be built to decode this packet data. After receiving packet data from the libusrp library, the data is inserted into a circular buffer of 512 byte packets. The packets are read from the circular buffer and processed in sequence to extract the metadata in the header.

Each packet contains 8 bytes of header data and 504 bytes of sampled signal data. Each sample contains two 16 bit signed integer values comprising the I and Q components of

the digitized signal. The header contains 4 bytes of bit-packed header data and 4 bytes containing a 32-bit unsigned timestamp. Figure 15 illustrates the USRP in-band packet structure. The first 32 bits of the header include several bit-packed fields including the payload length, which is always 504 bytes. The timestamp follows in the next 32 bits and contains the number of samples (at 64MS/s, regardless of decimation rate) elapsed since the last 1PPS signal from the GPSDO. All fields of greater than one bit are encoded as unsigned integers and are carried in network byte order.

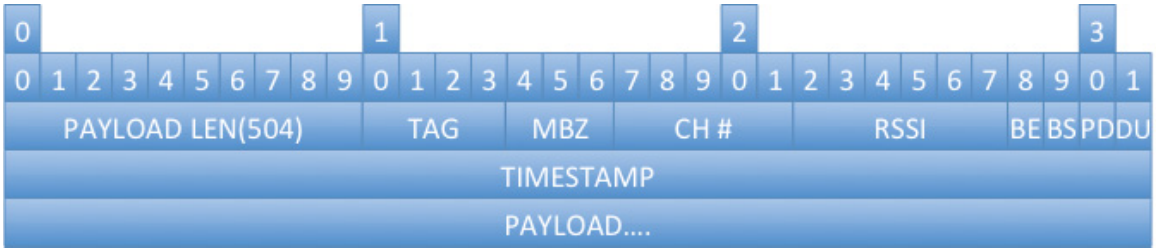

Figure 15 USRP in-band packet

When a packet is received, the timestamp is read from the packet. When a counter recycle is detected, the number of samples since the last recycle is computed and is considered to be the new sampling rate. The packet data is then pushed out onto a *Uses standardInterfaces complexShort Port* with metadata. The metadata is populated with the timestamp of the first sample in each packet and the USRP sampling rate, both in samples. The USRP decimation ratio is also included in the metadata.

3.4.2 USRP_Commander Component

The OSSIE *USRP_Commander* component was modified to take in events from the CORBA Event Service, specifically a *TuneRequestEvent*. This event instructs the *USRP_Commander* to change its parameters, specifically decimation factor and carrier frequency.

3.4.3 Throttler Component

The *Throttler* component is an OSSIE component used to take snapshots of collected data when requested. It utilizes a *Uses complexShort Port* from *thesisInterfaces* to allow buffering of data on the slow side of a wide area network. It has a *Provides thesisInterfaces TriggerControl Port* to allow other components to trigger snapshot data

collection. When this trigger is received, it indicates the amount of data that should be collected, and at what time the collection should start. After sending out the snapshot data, the *Throttler* then returns to its default state of sending no data.

3.4.4 FM Demodulator Component

The *WFMDemod* component included with OSSIE was used for the demodulation task. This component implements a simple differentiator to demodulate the incoming signal. Because the amplitude data for the FM signal is encoded in the signal as a changing frequency, all that is required to demodulate the signal is computing the instantaneous frequency at any given point in time [20]. This can be accomplished by computing the derivative of the signal's instantaneous phase. The phase can be computed by taking the arctangent of the ratio of the quadrature signal to the in-phase signal. The derivative of this signal results in the instantaneous frequency [20]:

$$\Delta\theta(n) = \frac{i(n)\frac{d[q(n)]}{dn} - q(n)\frac{d[i(n)]}{dn}}{i^2(n) + q^2(n)} \tag{4}$$

This equation can be implemented in software using tapped-delay line Finite Impulse Response (FIR) differentiating filters as shown in Figure 16, where the result is scaled by

$$\frac{f_s}{2\pi} \tag{5}$$

to compute the instantaneous frequency, and f_s is the sampling rate [20].

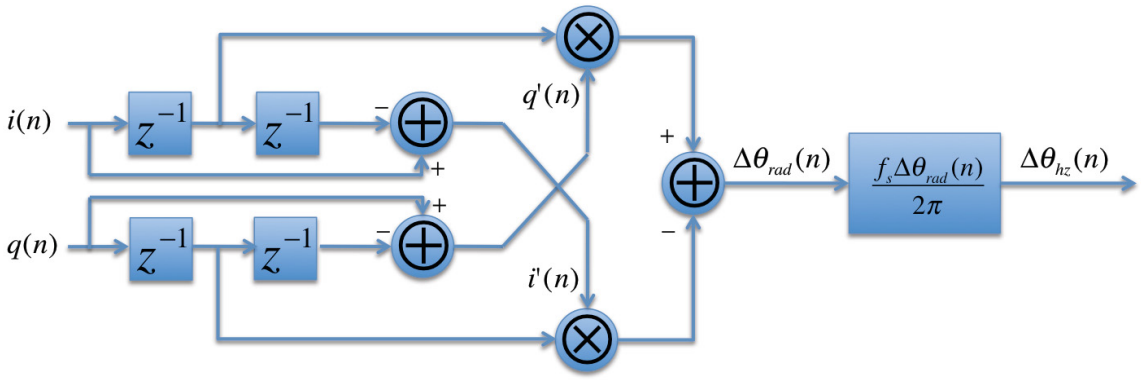

Figure 16 FM demodulator, adapted from [20]

3.4.5 Arbitrary Ratio Resampler Component

Because the USRP board does not have its oscillator locked on to any reliable reference, the rate at which the signal is sampled is not precisely 64MS/s, nor does it remain consistent over time. It is, however, relatively consistent from one second to the next. Because of this, the sampling rate of the signal can be calculated by counting the number of samples generated in a one-second period. Using the calculated sampling rate, the collected signal can be resampled to an accurate 64MS/s.

A traditional rational (fractional) resampler can achieve non-integer resampling ratios by first interpolating by an integer ratio, M, and then decimating by an integer ratio, N, producing an effective M/N resampling ratio [21]. The signal is first zero-padded with M-1 samples for each input sample, then low pass filtered to remove spectral images, and finally down-sampled by removing every N-1 samples. This type of resampling wastes computing cycles both during up-sampling, where multiply-by-zero operations are computed when low-pass filtering, and during down-sampling, where newly computed samples are thrown away. A polyphase filter can avoid this problem by building the resampler in a way that it skips these unnecessary operations. In a polyphase filter, the low-pass filter designed for the rational resampler can be broken up into M partitions of length L/M, where each partition computes a single output sample [21]. Because of zero padding, the only filter coefficients that contribute to each output sample are those where $c = iM + p$, where c is the coefficient number in the original filter, i is the index within the partitioned filter, L is the filter length, M is the number of partitions, and p is the current partition number. Iterating from $i = 0$ to $i = L/M - 1$ will produce the filter coefficients for each partition. The partitioned filter can be used in the same way as the original filter, except the signal need not be zero padded. After each output sample is calculated, the filter partition is incremented and the next output sample is calculated. After all the filter partitions are used, the input sample is incremented and the process starts over again, as illustrated in Figure 17.

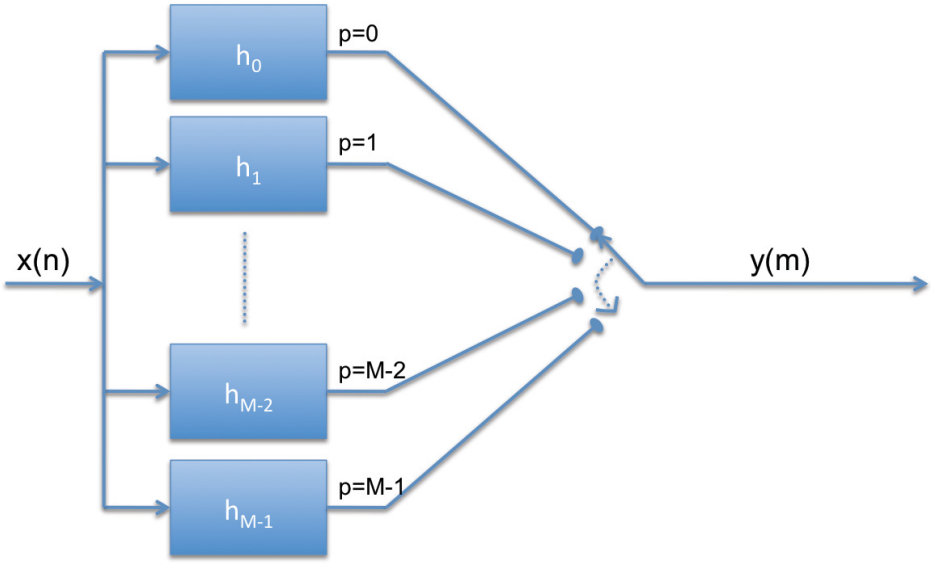

Figure 17 Polyphase resampler

The polyphase filter discussed thus far implements only the interpolation portion of the rational resampler. If a rational (fractional) ratio is needed, the partition can be incremented by N instead of one, where N is the decimation factor [21]. The resulting resampling ratio will be M/N. In any case the low-pass filter should be designed to operate at the interpolated rate, M*(initial sampling rate), and to filter appropriately for the decimated rate, M*(initial sampling rate)/N.

For a typical resampling rate used in this research, where the incoming sampling rate might be 63,984,333S/s, an interpolation factor of 256*64,000,000 and decimation factor of 63,9843,33 are required to both return the signal to its original sampling rate (after decimation by 256 on the USRP) and to correct the error in the sampling rate. A polyphase filter for this ratio would require M=1.6384E10 partitions and a correspondingly large low pass filter length, L. Because of the size, this filter would be impractical to implement. The polyphase concept can be taken a step further to design a resampler with ratio M/N, or any irrational ratio, without this complexity. If the partitions are incremented at a non-integer rate, such a large filter is no longer necessary. The problem with this approach is that, when incremented at this rate, partitions are needed between actual partitions defined in the polyphase filter. These intermediate filters can be handled in one of three ways [21]:

- The nearest neighboring filter partition can be chosen,

- A linear interpolation can be performed between the two filters,
- A more sophisticated curve fitting interpolation method can be used to interpolate between the two filters (known as a "Farrow" filter).

The first option, choosing the nearest neighbor, has a great advantage in that it is very simple. Some noise will be injected into the signal in the appearance of spectral imaging, because the resultant sample is chosen that should have occurred to the right or left of the desired sample. This has a similar effect on a signal as a zero-order-hold, because of the re-use of output samples [21]. These images should be tolerable by designing a filter such that the stop band has sufficient attenuation so that these images do not contribute to the resulting signal. Because the filter is in effect an interpolation followed by a decimation, the filter should, however, be designed such that the stop band has a continuous roll off rather that an equiripple design to avoid the zero-order hold images being folded back into the passband and accumulating during the effective decimation [21].

According to Harris [21], to ensure that the maximum amplitude of any residual spectra caused by the usage of the nearest-neighbor approach is smaller than any imaging injected by quantization error, the number of phases must meet the following requirement

$$N > 2^{(b-1)} \tag{6}$$

where b is the number of bits in the digitizer and N is the number of phases in the polyphase resampler. For the USRP board, where the number of bits in the ADC is 12, this would require 2048 phases in the resampler. For this research, to simplify the design of the filter, the number of phases was held back to 64, and, instead, the filter was designed to attenuate the signal further to compensate for the spectral artifacts. Since the filter used has continuous roll-off in its stop-band, the energy contained in these images should not accumulate substantially when folded back into the pass-band.

The other two options, whereby a more sophisticated interpolation is performed on the partitioned filters before resampling, require a partial recomputation of the filter phase prior to each filter computation. This was considered too great a computational requirement to be considered for this research.

An octave application was built to experiment with different filter parameters. A Kaiser windowed FIR filter was designed utilizing the Octave window-based FIR filter design tools. The desire was to keep all images attenuated to at least 50dB below the filter pass-band. A Kaiser window was selected and designed to generate a stop-band attenuation of -100dB to also attenuate the spectral imaging produced by the zero-order hold effect. A test filter was designed with 64 phases and a resampling ratio of 256*64,000,000/63,9843,33=256.06. A signal with two tones was passed through that filter and the resulting spectrum is presented in Figure 18. Because the resampling ratio is non-integer and some phases of the filter are used multiple times, you can see zero-order hold induced images present in the spectrum, though they are all kept under -50dB. The inset in Figure 18 also shows a magnification of the plot showing the main signal components. Figure 19 shows the zero-order hold effect on the time-domain signal. The figure shows samples being re-used and forming a "stair-step" effect. In-fact, the average number of re-used samples is roughly four, since the resampling rate is roughly four times the number of phases in the designed filter.

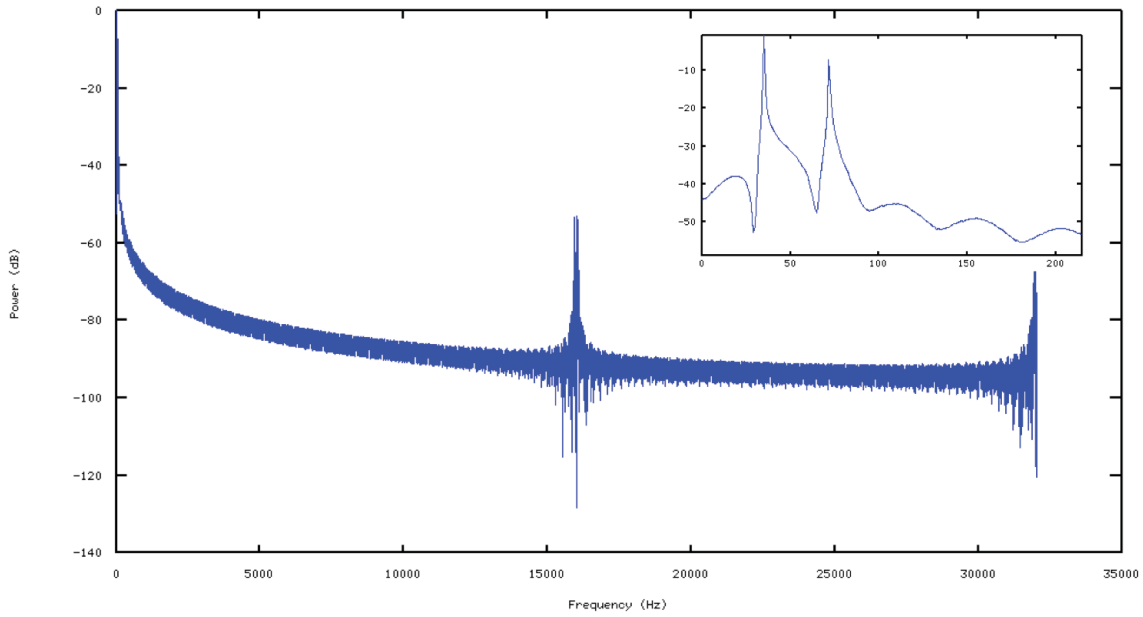

Figure 18 Plot for polyphase resampler implemented in Octave

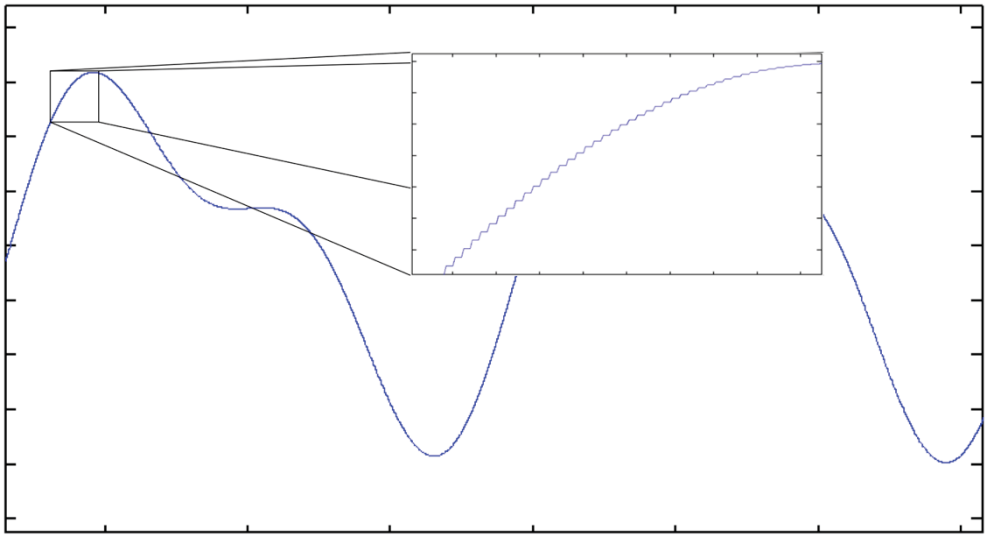

Figure 19 Effects of nearest-neighbor interpolation on signal

An OSSIE Component was also built as a port from this Octave code, but was later abandoned in favor of using the polyphase resampler that is part of the LiquidDSP library [22]. The LiquidDSP resampler was utilized in a new OSSIE *Resampler* component.

3.4.6 Correlator Component

The *Correlator* component controls the process of collecting and correlating a signal from a pair of receivers. When a *TriggerCollect* event is received by the *Correlator* component, the process of collecting and correlating a pair of signals is started. The *TriggerCollect* event indicates what channels should be collected, how long the snapshots should be, and the number of TDOA samples that should be made per channel. After receiving the event, the *Correlator* sends a *TuneRequestEvent* to tune each collector, then a trigger is sent through the *TriggerControl Port* to start a snapshot collect from both *Throttler* components. The *Correlator* then buffers data from each collector until the proper number of samples is received. The signals are then aligned in time and correlated. The maximum correlation is detected, and used as a TDOA sample. After the correlation is computed, the *Correlator* writes the TDOA sample, buffered signal data, and the correlation plot to disk for analysis. A *TdoaEvent* is then published to the

THESIS_EVENTS event channel for each TDOA sample generated. This process is repeated until all TDOA samples are collected for all channels requested.

The fast correlation algorithm takes advantage of the fact that convolution in the time domain translates to multiplication in the frequency domain.

$$a(t) * b(t) \Leftrightarrow A(\omega) \cdot B(\omega) \tag{7}$$

The demodulated FM data that is being correlated translates into a complex FFT, and, as such, the multiplication for each element is accomplished by:

$$C(\omega)_{real} = A(\omega)_{real} B(\omega)_{real} - A(\omega)_{imag} B(\omega)_{imag} \tag{8}$$

$$C(\omega)_{imag} = A(\omega)_{real} B(\omega)_{imag} + B(\omega)_{real} A(\omega)_{imag} \tag{9}$$

Once the product is computed, an inverse FFT is performed on the result, the magnitude of the resulting signal is computed, and the peak is detected to determine the TDOA.

3.5 TDOA Geolocation

Two methods of geolocating an emitter using TDOA samples are considered. The first method utilizes an iterative Taylor series approximation method [23]. The second method is the closed form spherical intersection method [24]. These methods are presented and the error in the measurements is translated to error in position estimation of the emitter.

3.5.1 Iterative Taylor Series Approximation Method

The TDOA samples generated in the *Correlator* component can be converted into range-difference of arrival (RDOA) samples and used to solve for the location of the unknown emitter. An RDOA value is simply a TDOA value converted from time in samples to distance in meters. The RDOA is computed,

$$TDOA_{\text{sec}} = \frac{TDOA_{samples}}{f_s} \tag{10}$$

$$RDOA_m = TDOA_{\text{sec}} \cdot c \tag{11}$$

where c is the speed of light in meters per second and f_s is the sampling rate of the collector.

The emitter is located in three-dimensional space. Because each TDOA measurement requires the difference of arrival at two collectors, four collectors and three TDOA measurements are required to compute the three-dimensional location of the emitter. Converting the location of the emitter and the collectors from Geodetic (Latitude, Longitude, Altitude) to Cartesian (x, y, z) coordinates, the following RDOA equations are formulated using the Pythagorean theorem,

$$\begin{aligned} R_{21} &= f_1(x,y,z) = \sqrt{(x-x_2)^2+(y-y_2)^2+(z-z_2)^2} - \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2} \\ R_{31} &= f_2(x,y,z) = \sqrt{(x-x_3)^2+(y-y_3)^2+(z-z_3)^2} - \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2} \\ &\vdots \\ R_{n1} &= f_n(x,y,z) = \sqrt{(x-x_n)^2+(y-y_n)^2+(z-z_n)^2} - \sqrt{(x-x_1)^2+(y-y_1)^2+(z-z_1)^2} \end{aligned} \tag{12}$$

where x, y, and z are the location of the emitter in Cartesian coordinates, x_i, y_i, and z_i are the location of the i[th] collector, and R_{ij} is the RDOA measurement between the i[th] and j[th] collector. Each equation defines a hyperbolic curve with the collectors as points of focus [23]. Alternatively, if the emitter can be considered to be on the surface of the earth, only three collectors are needed, the third equation in (12) with an equation describing an oblate spheroid that the earth closely resembles [25],

$$1 = f_3(x,y,z) = \frac{x^2+y^2}{a^2} + \frac{z^2}{c^2} \tag{13}$$

where

$$a = EARTH\_EQUITORIAL\_RADIUS \tag{14}$$

$$c = a(1 - EARTH\_FLATTENING\_FACTOR) \tag{15}$$

These equations are non-linear and can be difficult to solve with a closed form solution, even if each measurement is made without error. The system of equations can be made approximately linear by using the Taylor series expansion centered on an initial estimate for the location of the emitter, neglecting all terms other than the first order [23],

$$R_{i1} - f_i(x_g, z_g, z_g) = \frac{\partial f_i}{\partial x}(x - x_g) + \frac{\partial f_i}{\partial y}(y - y_g)\frac{\partial f_i}{\partial z}(z - z_g) \tag{16}$$

where x_g, y_g, and z_g are the location of the initial guess. In matrix form this becomes,

$$\Delta M = J\Delta X \tag{17}$$

where the Jacobian, J , is the set of partial derivatives forming the first order term of the Taylor series expansion,

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x} & \frac{\partial f_n}{\partial y} & \frac{\partial f_n}{\partial z} \end{bmatrix} \tag{18}$$

and ΔM is the difference between the measured value and the value at the initial guess,

$$\Delta M = \begin{bmatrix} R_{21} - f_1\left(x_g, z_g, z_g\right) \\ R_{31} - f_2\left(x_g, z_g, z_g\right) \\ \vdots \\ R_{n1} - f_n\left(x_g, z_g, z_g\right) \end{bmatrix} \tag{19}$$

and ΔX is the delta between the initial guess and the solution to the Taylor series approximation,

$$\Delta X = \begin{bmatrix} x - x_g \\ y - y_g \\ z - z_g \end{bmatrix} \tag{20}$$

Because this system can be over-determined if there are more equations than unknowns (n > 3) and the measurements are not precise, it cannot be solved exactly, and there will be some error in these equations. To account for error in the system, equation (17) can be re-organized as,

$$\varepsilon = J\Delta X - \Delta M \tag{21}$$

where ε is the error.

A least-squared minimization can be used to minimize the sum of the squares of the error,

$$\min\left|\left|J\Delta X - \Delta M\right|\right|^2 \tag{22}$$

This can be accomplished by multiplying both sides of equation (17) by J^T ,

$$J^T \Delta M = J^T J \Delta X \tag{23}$$

then solving for ΔX [23],

$$\Delta X = (J^T J)^{-1} J^T \Delta M \tag{24}$$

If a confidence can be placed on each measurement, a weighting matrix may be employed to place an appropriate weighting value on each measurement,

$$\Delta X = (J^T W J)^{-1} J^T W \Delta M \tag{25}$$

This weighting matrix W, is simply a diagonal matrix of weights,

$$W = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_n \end{bmatrix} \tag{26}$$

where w_i is the weight for each measurement. If the variance is known for each measurement, the inverse of the diagonal matrix of variances from each measurement can be used as the weighting matrix.

In either case, ΔX can then be added to the initial guess, and used as a subsequent guess to approach closer to the solution,

$$X_{G,new} = \Delta X + X_{G,old} \tag{27}$$

until the absolute value of ΔX is smaller than some minimum error, err_{min}.

$$|\Delta X| < err_{\min} \tag{28}$$

The main problem with this technique is that it can be rather difficult to choose the initial location. If the initial location is improperly chosen, the iteration can quickly diverge due to the highly imperfect approximation using only the first order part of the Taylor series approximation. Large steps in each increment can easily jump right over the actual emitter location and possibly never approach the emitter location. Figure 20 shows a converging iteration sequence where the algorithm gradually approaches the target. The figure also shows a situation where the algorithm diverges, oscillating around the earth, not finding the emitter.

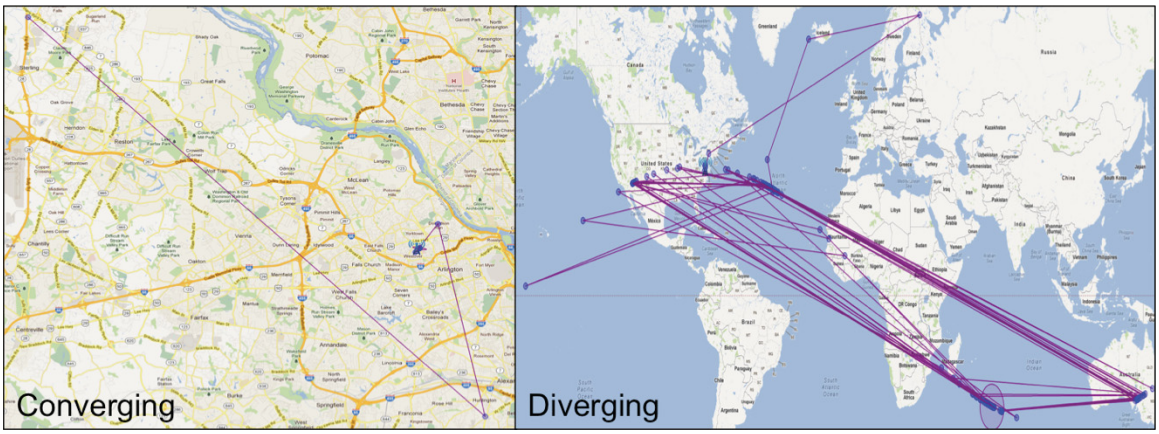

Figure 20 Iterative method converging and diverging

3.5.2 Spherical Intersection Method

There are also several methods for approximating a closed-form solution for the location of the emitter. One considered here is the spherical intersection method [24]. This method says that the location of the emitter can be found by finding the intersection of the spheres inscribed by the range from each collector to the emitter. Given that one of the collectors is considered to be the origin of the system, the range from each collector to the emitter is [24]

$$\begin{aligned} D_1 &= D_e \\ D_2 &= D_e + R_{21} \\ &\vdots \\ D_n &= D_e + R_{n1} \end{aligned} \tag{29}$$

where D_i is the distance from collector i to the emitter , D_e is the distance from the origin (in the case of this problem, collector 1) to the emitter, and R_{i1} is the RDOA between collector i and collector 1 (the origin). D_2 through D_n are the set of equations that are used to solve for the location of the emitter. Using the distance from the emitter to collector i, and the Pythagorean theorem, these equations become,

$$\left(D_e + R_{i1}\right)^2 = \left(x_i - x\right)^2 + \left(y_i - y\right)^2 + \left(z_i - z\right)^2 \tag{30}$$

expanding, and simplifying, this becomes [24],

$$D_e^2 + D_e R_{i1} + R_{i1}^2 = x_i^2 + y_i^2 + z_i^2 + x^2 + y^2 + z^2 - 2x_i x - 2y_i y - 2z_i z \tag{31}$$

$$D_e^2 + 2D_e R_{i1} + R_{i1}^2 = D_i^2 + D_e^2 - 2X_i^T X \tag{32}$$

$$0 = D_i^2 - R_{i1}^2 - 2D_e R_{i1} - 2X_i^T X \tag{33}$$

and in matrix form,

$$0 = \delta - 2D_e \underline{R} - 2SX \tag{34}$$

where

$$\delta = \begin{bmatrix} D_2^2 - R_{21}^2 \\ D_3^2 - R_{31}^2 \\ \vdots \\ D_n^2 - R_{n1}^2 \end{bmatrix} \tag{35}$$

$$S = \begin{bmatrix} x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \tag{36}$$

$$\underline{R} = \begin{bmatrix} R_{21} \\ R_{31} \\ \vdots \\ R_{n1} \end{bmatrix} \tag{37}$$

If the system is over-determined system, there will be some error,

$$\varepsilon = \delta - 2D_e \underline{R} - 2SX \tag{38}$$

This error can be minimized by using a least square's method, minimizing the sum of the squares,

$$\min \left|\left| \delta - 2D_e \underline{R} - 2SX \right|\right|^2 \tag{39}$$

This can be accomplished by reorganizing the equation [24],

$$\delta - 2D_e \underline{R} = 2SX \tag{40}$$

multiplying both sides by S^T,

$$S^T \left(\delta - 2D_e \underline{R} \right) = 2S^T SX \tag{41}$$

and solving for X yields,

$$X = \frac{1}{2} \left(S^T S \right)^{-1} S^T \left(\delta - 2D_e \underline{R} \right) \tag{42}$$

Equation (42) can be inserted into the Pythagorean equation for the range of the emitter from the origin [24],

$$D_e^2 = X^T X \tag{43}$$

expanding this equation results in [24],

$$aD_e^2 + bD_e + c = 0 \tag{44}$$

where

$$a = 4 - 4D_e^T \left(\left(S^T S \right)^{-1} S^T \right)^T \left(S^T S \right)^{-1} S^T D_e \tag{45}$$

$$b = 4D_e^T \left(\left(S^T S \right)^{-1} S^T \right)^T \left(S^T S \right)^{-1} S^T \delta \tag{46}$$

$$c = -\delta^T \left(\left(S^T S \right)^{-1} S^T \right)^T \left(S^T S \right)^{-1} S^T \delta \tag{47}$$

and

$$D_e = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{48}$$

yields an approximation for D_e. Plugging D_e into (42) leads to a linear closed-form solution for X [24].

This method results in a large error in location estimation for any error in range difference measurements if the collectors are not located near the emitter [24]. Therefore, this closed-form method is to be used as an initial guess to the iterative method.

3.5.3 Error

The error in the TDOA measurements can be transformed into a corresponding error in the estimated position of the emitter. The variations in each TDOA measurement made can be described by a random variable. A single random vector can represent the random variables for a set of measurements [26],

$$T = \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_n \end{bmatrix} \tag{49}$$

and

$$\mu_T = E(T) \tag{50}$$

is the mean vector for T and

$$\Sigma_T = E\left[(T-\mu_T)(T-\mu_T)^T\right] \tag{51}$$

is the covariance matrix for T. Expanding the equation,

$$\Sigma_T = \begin{bmatrix} \delta_1^2 & \delta_{12} & \cdots & \delta_{1n} \\ \delta_{21} & \delta_2^2 & \cdots & \delta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n1} & \delta_{n2} & \cdots & \delta_n^2 \end{bmatrix} \tag{52}$$

where δ_n^2 is the variance of the nth measurement, and δ_{nm} is the covariance between n and m [26],

$$\delta_{nm} = \rho\delta_n\delta_m \tag{53}$$

and ρ is the correlation coefficient,

$$\rho = \frac{\delta_{nm}}{\delta_n\delta_m} \tag{54}$$

Assuming that all of the random variables defining the TDOA measurements are independent, and therefore uncorrelated, $\rho = 0$, and the covariance terms disappear and a purely diagonal matrix results [26],

$$\Sigma_T = \begin{bmatrix} \delta_1^2 & 0 & \cdots & 0 \\ 0 & \delta_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_n^2 \end{bmatrix} \tag{55}$$

This covariance matrix can be translated from TDOA measurements to Cartesian (x,y,z) coordinates by taking advantage of the law of propagation of uncertainty [27],

$$\Sigma_{XYZ} = E\left(E_{XYZ}E_{XYZ}^{T}\right) \tag{56}$$

$$\Sigma_{XYZ} = E\left(\left(\left(J^{T}J\right)^{-1}J^{T}E_{TDOA}\right)\left(\left(J^{T}J\right)^{-1}J^{T}E_{TDOA}\right)^{T}\right) \tag{57}$$

$$\Sigma_{XYZ} = E\left(\left(J^{T}J\right)^{-1}J^{T}E_{TDOA}E_{TDOA}^{T}J\left(J^{T}J\right)^{-1}\right) \tag{58}$$

$$\Sigma_{XYZ} = \left(J^{T}J\right)^{-1}J^{T}E\left(E_{TDOA}E_{TDOA}^{T}\right)J\left(J^{T}J\right)^{-1} \tag{59}$$

$$\Sigma_{XYZ} = \left(J^{T}J\right)^{-1}J^{T}\Sigma_{TDOA}J\left(J^{T}J\right)^{-1} \tag{60}$$

Finally, Σ_{XYZ} can be rotated into the local east-north-up (ENU) coordinate system by again using the laws of propagation of uncertainty. A simple rotation can be performed, with the following rotation matrix, given geodetic latitude and geodetic longitude [25],

$$R = \begin{bmatrix} -\sin(lon) & \cos(lon) & 0 \\ -\sin(lat)\cos(lon) & -\sin(lat)\sin(lon) & \cos(lat) \\ \cos(lat)\cos(lon) & \cos(lat)\sin(lon) & \sin(lat) \end{bmatrix} \tag{61}$$

Thus, the Σ_{ENU} can be calculated,

$$\Sigma_{ENU} = E\left(E_{ENU}E_{ENU}^{T}\right) \tag{62}$$

$$\Sigma_{ENU} = E\left(\left(RE_{XYZ}\right)\left(RE_{XYZ}\right)^{T}\right) \tag{63}$$

$$\Sigma_{ENU} = RE\left(E_{XYZ}E_{XYZ}^{T}\right)R^{T} \tag{64}$$

$$\Sigma_{ENU} = R\Sigma_{XYZ}R^{T} \tag{65}$$

resulting in a covariance matrix in the local coordinate system, ENU, in meters at the location of the solution calculated in the solution step.

The eigenvectors and eigenvalues of the covariance matrix can then be used to generate an error ellipsoid (3D) and error ellipse (2D) to represent the error in the local coordinate system [26],

$$\Lambda = eigvec(\Sigma_{ENU}) \tag{66}$$

$$\nu = eigval(\Sigma_{ENU}) \tag{67}$$

where Λ is a matrix containing the eigenvectors of $\sum_{ENU}$ in its columns and v is a vector containing the eigenvalues of $\sum_{ENU}$. The square roots of the eigenvalues are then proportional to the lengths of the axis of the ellipsoid and the eigenvectors point in the direction of the axis [26],

$$axis_n = \sqrt{K\lambda_n} \tag{68}$$

where K is described by a chi-squared random variable with n degrees of freedom

$$\chi_n^2 \tag{69}$$

and n is the number of Gaussian random variables in the random vector. The chi-squared random variable has a probability density function (PDF) [26],

$$f_n(x) = \frac{1}{2^{\frac{n}{2}}\Gamma\left(\frac{n}{2}\right)} x^{\frac{n}{2}-1} e^{-\frac{x}{2}} \tag{70}$$

where the probability of containment is equal to the integral of the PDF from zero to K,

$$P = \int_0^K f_n(x)dx \tag{71}$$

If a two-dimensional error ellipse is desired, the covariance matrix tangent to the surface of the earth at the emitter must be generated. This covariance matrix is simply the first two rows/columns in the ENU covariance matrix.

For this research, the TDOA error was calculated by first making many TDOA measurements then calculating the variance of these measurements. The resulting data generally resembled a Gaussian distribution. Any error due to differing signal delay in the collection systems is considered system bias and needs to be removed from the system before performing the geolocation computations to produce the most accurate prediction possible. The delays induced by the USRP or software processing components such as the resampler are ignored in this research because they do not affect the TDOA result, as they are the same in each collection path.

3.6 Web Application

For this research, a unique method of displaying and analyzing results was envisioned and built. A Java web application was built using Google Web Toolkit (GWT) [28] and a Tomcat web application server. GWT allows a developer who is familiar with Java to build an entire web application with the Java programming language, allowing both the server-side and client-side to be written in Java [28]. The client-side is compiled into JavaScript at build-time and is embedded into a web page to dynamically build the user interface at runtime. The web application provides the capability for a user to load and analyze a set of TDOA data and its corresponding geolocations. The web application has a set of visualization tools including a data grid, map plot, signal plot, and histogram plots. A system diagram is shown in Figure 21.

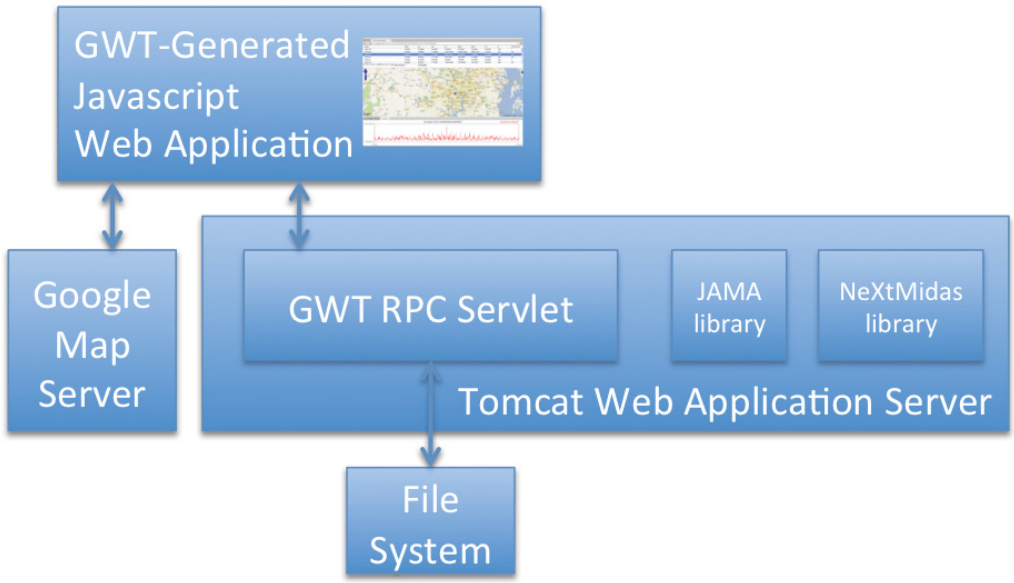

Figure 21 Web application system diagram

When the web application starts up, the TDOA datasets are loaded into memory. The TDOA measurements for each FM radio station are loaded and statistics (mean and standard deviation) are computed. The location estimate and error ellipse are then produced using the algorithms presented in section 3.5. When a user loads a dataset, a list of FM radio stations is listed. Each FM radio station is displayed along with its predicted location and error ellipse size and orientation. If a user selects an FM radio station, the

data associated with that FM radio station is loaded. The map is updated to show the actual location of the emitter and the predicted location of the emitter and the measurements' error ellipse. The location of each collector used in the geolocation is also plotted. In another list, the set of TDOA pairs used to locate the emitter is listed. If one of these pairs is selected, a histogram is loaded showing the distribution of TDOA measurements made for that pair. The signal data explorer plots an example the collected signal and correlation for the pair of collectors. The web application server uses the Java Matrix Package (JAMA) [29] to do matrix math and NeXtMidas [30] for position translations.

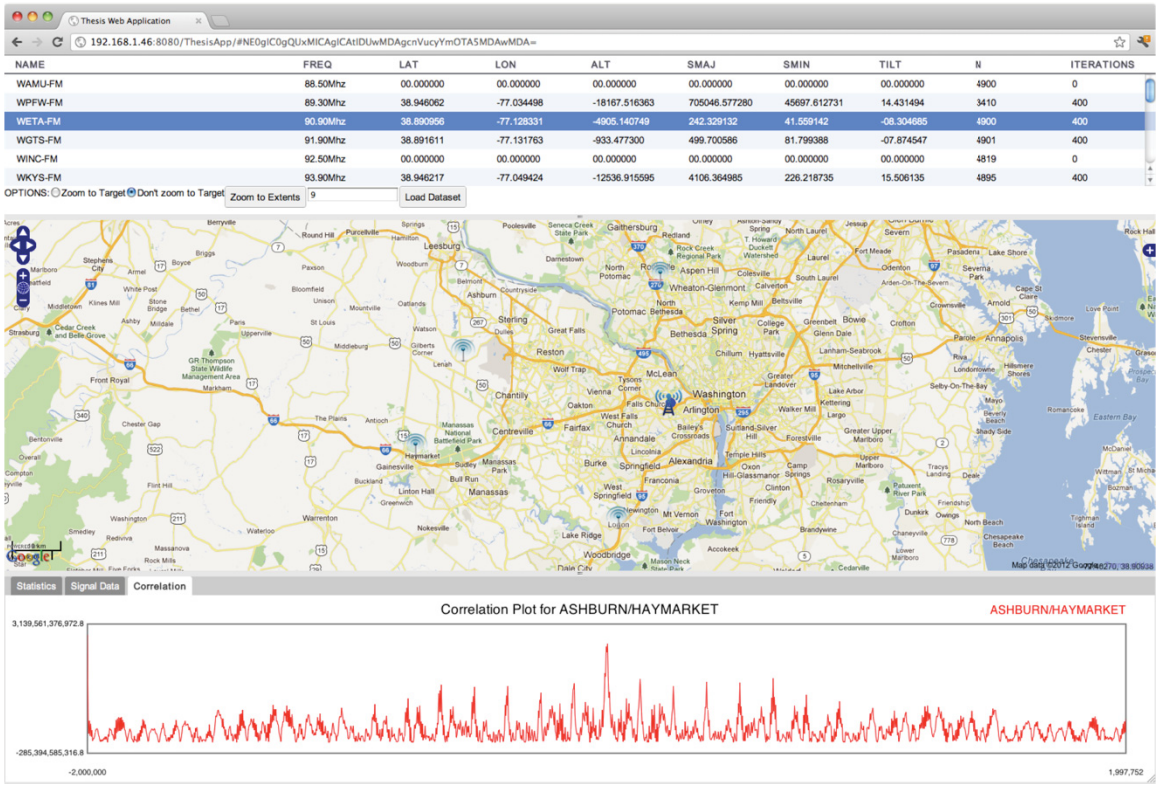

Figure 22 Web application GUI

Figure 22 shows the thesis web application GUI built for this research. On the top of the screen is the data grid showing the list of FM stations collected. The section immediately below this is a set of user controls. The user can set an option as to whether the map plot will zoom to the target or remain at the current zoom position after each FM Radio station selection. A "Zoom To Extents" button is available that will zoom the map to display the region of consideration for this research. The "Load Dataset" button loads a

popup where the user may select a dataset to load. Below the user inputs, a map is available to plot any map-related data. Below the map is a set of tabs that contain the pair statistics and the signal data explorer plots.

The web application is fully configurable with an XML file. The XML file defines the locations of each collector and emitter. The XML file also defines any bias present in the measurements for each collector so that bias can be removed prior to position estimation. The simulated error standard deviation for each collected signal at the real collectors is included for use when all the collectors are simulated. An example XML file is shown in Figure 23.

```
<XML>
<STATIONS>
  <!-- USE THE POSITIONS OF THE TRANSMITTERS TO SIMULATE A THIRD AND FOURTH COLLECTOR -->
  <STATION FREQ="88500000" NAME="WAMU-FM" LAT="38.93611" LON="-77.09250" ALT="223" SIM_STD="4.586"/>
  <STATION FREQ="90900000" NAME="WETA-FM" LAT="38.89167" LON="-77.13194" ALT="252" SIM_STD="5.05"/>
  <STATION FREQ="91900000" NAME="WGTS-FM" LAT="38.89167" LON="-77.13194" ALT="252" SIM_STD="10.231"/>
  <STATION FREQ="93900000" NAME="WKYS-FM" LAT="38.94000" LON="-77.08167" ALT="286" SIM_STD="21.222"/>
  <STATION FREQ="94700000" NAME="WIAD-FM" LAT="38.96361" LON="-77.10500" ALT="312" SIM_STD="2.344"/>
  <STATION FREQ="97100000" NAME="WASH-FM" LAT="38.95028" LON="-77.07972" ALT="315" SIM_STD="62.827"/>
</STATIONS>
<RECEIVERS>
  <RECEIVER NAME="ASHBURN" LAT="38.968886" LON="-77.523353" ALT="94" SIMULATED="NO" BIAS="0"/>
  <RECEIVER NAME="HAYMARKET" LAT="38.829525" LON="-77.614909" ALT="103" SIMULATED="NO" BIAS="45"/>
  <RECEIVER NAME="ROCKVILLE" LAT="39.191039" LON="-76.839861" ALT="115" SIMULATED="YES" BIAS="0"/>
  <RECEIVER NAME="LORTON"    LAT="38.591039" LON="-76.839861" ALT="100" SIMULATED="YES" BIAS="0"/>
  <RECEIVER NAME="FREDERICK" LAT="39.426389" LON="-77.420278" ALT="200" SIMULATED="YES" BIAS="0"/>
</RECEIVERS>
</XML>
```

Figure 23 Web application configuration XML

3.6.1 Signal Data Explorer

The Signal Data Explorer is the software component used to plot signal data and correlation data in the web application. It was built to allow interaction with a large set of sampled data. Many plotting applications are available for HTML/JavaScript applications, but none could be found that allowed interacting with a server-side dataset with 2-4+ million data points. All data points could not be transmitted or plotted at once due to both limitations in the Internet connection speed with clients and the computational load of attempting to plot all data points at once. The Signal Data Explorer defines a server interface that asynchronously makes requests to the server for more data as the user zooms deeper into a signal data plot.

4 Experimentation

4.1 Procedure

Two OSSIE *DeviceManager* nodes, each with a *USRP_TIME Device* and a *GPP* device were initialized, along with a single *DomainManager* instance. The TDOA waveform described in section 3.3.2 was built, installed, and started. For each FM radio station considered, 4,000,000 up-sampled and demodulated samples were buffered and correlated to generate TDOA measurements. 5000 TDOA samples were made for each station. The TDOA samples taken were plotted as a histogram in the web application and those channels that produced Gaussian-shaped histograms were considered for this research. Two additional simulated collectors were simulated for each station and used to generate the two additional TDOA measurements required to solve the system of TDOA equations. Finally, the TDOA data was loaded and analyzed using the web application, where estimates for the location of each emitter were generated and analyzed.

4.1.1 Signal Collection

A signal was collected with both *USRP_TIME Devices* at each center frequency considered, and 4,000,000 samples were collected at each distributed collector. An example of one of these collections is shown in Figure 24 where the 91.9MHz channel captured from the Ashburn collector is shown in red and the capture from the Haymarket collector is shown in blue. Correlation of the signals is clear in this plot.

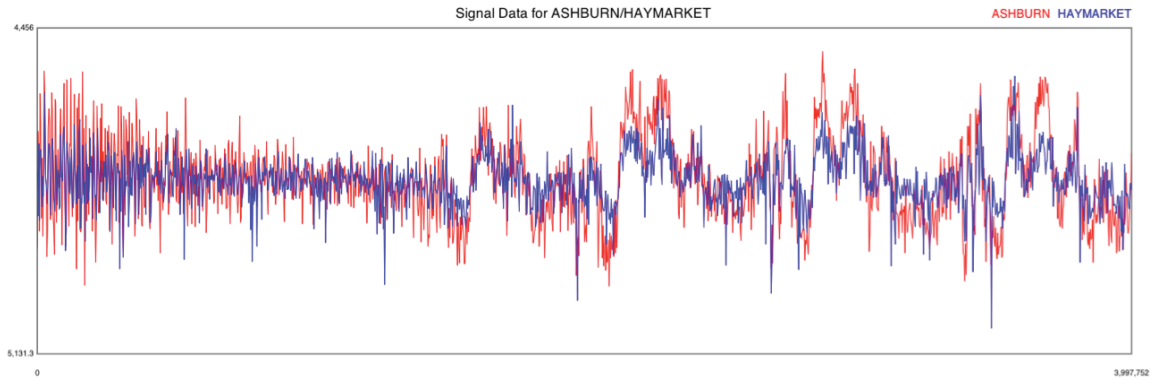

Figure 24 Sample time plot for 91.9MHz

4.1.2 Signal Correlation

After the signals were collected and aligned in time, the signals were then correlated. The maximum value was found and recorded as the TDOA measurement. In Figure 25, a sample correlation plot can be seen. In this plot, a peak was detected at -1589 samples, indicating that the signal was received by the Haymarket collector 1589 samples after the Ashburn collector.

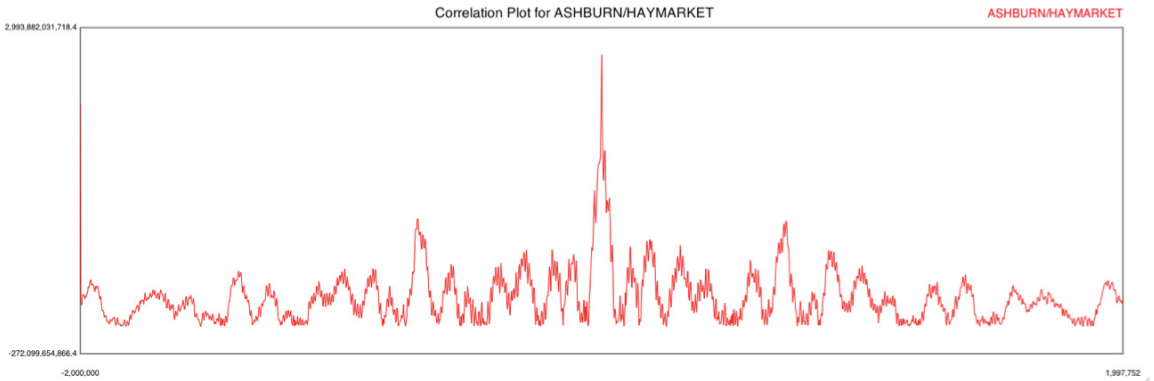

Figure 25 Sample correlation plot for 91.9MHz

5000 TDOA samples were made at 91.9MHz, and the mean TDOA value was determined to be -1586.06 samples with a variance of 104.67 samples. Figure 26 illustrates the location of each collector along with the location of the 91.9MHz radio tower, illustrating the TDOA measurement associated with each pair of collectors.

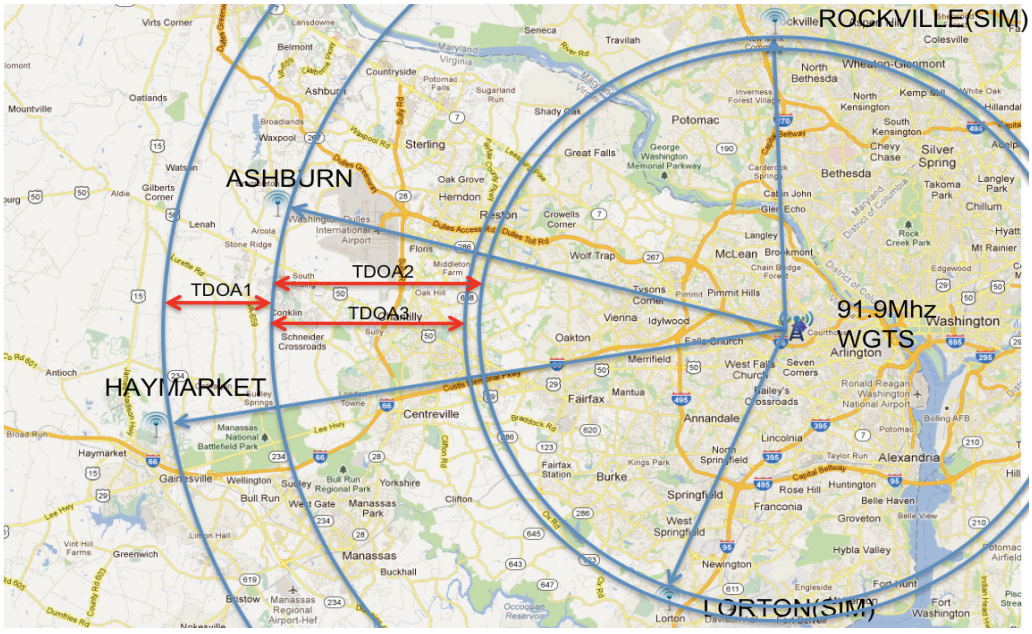

Figure 26 Plot showing collectors and emitter with TDOA measurements

4.1.3 Simulating Collectors

Because the system of equations to solve for the location of the emitter has three unknowns (x, y, z), at least three TDOA measurements are required to solve for the location of the emitter. Two additional TDOA measurements were simulated for this research, utilizing two simulated collectors and one of the real collectors. The location of each emitter was known, thus the distance from each emitter to each collector could be calculated. The simulated mean TDOA can be calculated by finding the difference between the distances to the emitter from each collector in each pair, as shown before in equation (12). It can be shown that the variance of the TDOA measurement is given by [7],

$$\delta_{TDOA}^2 \approx \frac{1}{\beta^2 BT\gamma} \tag{72}$$

where β is the frequency in radians, B is the noise bandwidth at the collector inputs, T is the integration time, and γ is the SNR. The value of γ can then be found from [7],

$$\gamma = \frac{2\gamma_1\gamma_2}{\gamma_1 + \gamma_2 + 1} \tag{73}$$

where γ_1 and γ_2 are the SNRs at either collector. Because bandwidth, frequency, and integration time are assumed to be equal at all collectors, they can be ignored and,

$$\delta^2 \propto \frac{1}{\gamma} \tag{74}$$

Since the signal strength is inversely proportional to the square of the distance, substituting $\frac{1}{r_1^2}$ for γ_1 and $\frac{1}{r_2^2}$ for γ_2 in (73) and (74) yields

$$\delta^2 \propto \frac{2}{r_2^2 + r_1^2 + r_2^2 r_1^2} \tag{75}$$

Knowing the TDOA variance of one pair of receivers (that are r_{1_1} and r_{2_1} meters from the emitter), δ_1^2, the TDOA variance of another pair of receivers (that are r_{1_2} and r_{2_2} meters from the same emitter), δ_2^2, can be approximated,

$$\delta_2^2 \approx \delta_1^2 \frac{r_{2_2}^2 + r_{1_2}^2 + r_{2_2}^2 r_{1_2}^2}{r_{2_1}^2 + r_{1_1}^2 + r_{2_1}^2 r_{1_1}^2} \tag{76}$$

This variance can then be used for the simulated collectors. Simulated TDOA measurements were made using Java's Gaussian Random Number Generator and used for the simulated collectors. Figure 27 shows a sample Histogram for both a real (91.9MHz with Haymarket and Ashburn) and simulated (91.9MHz with Lorton and Ashburn) TDOA Measurements. 5000 samples were taken for each histogram.

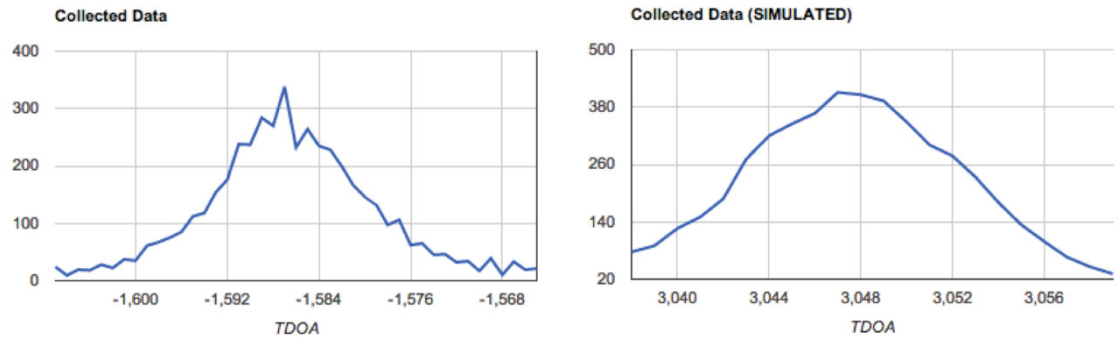

Figure 27 Histogram for real (Haymarket/Ashburn) and simulated (Lorton/Ashburn) TDOA measurements for 91.9MHz

4.1.4 Simulated Collector Arrangement

In [31] it is shown that a balanced placement of collectors around an emitter will produce the best possible TDOA position estimation, minimizing the effects of error in each measurement. For these results, the simulated collectors were placed in two configurations. The first configuration was an off-balanced configuration that maximized the effect of the real collectors. The second configuration was a balanced configuration that should produce more precise position estimations. Figure 28 shows a comparison of the unbalanced and balanced collector placements for locating the 91.9MHz emitter. In the unbalanced configuration, the simulated collectors are concentrated on the left-hand side of the circle around the emitter. In the balanced arrangement, the simulated collectors are placed in a balanced arrangement, making a triangle centered at the emitter. The TDOA equations intersection with the earth is shown in red. The balanced arrangement has more evenly distributed intersections, resulting in a tighter estimate for the emitter location given error in the measurements, as seen in the inset error ellipses.

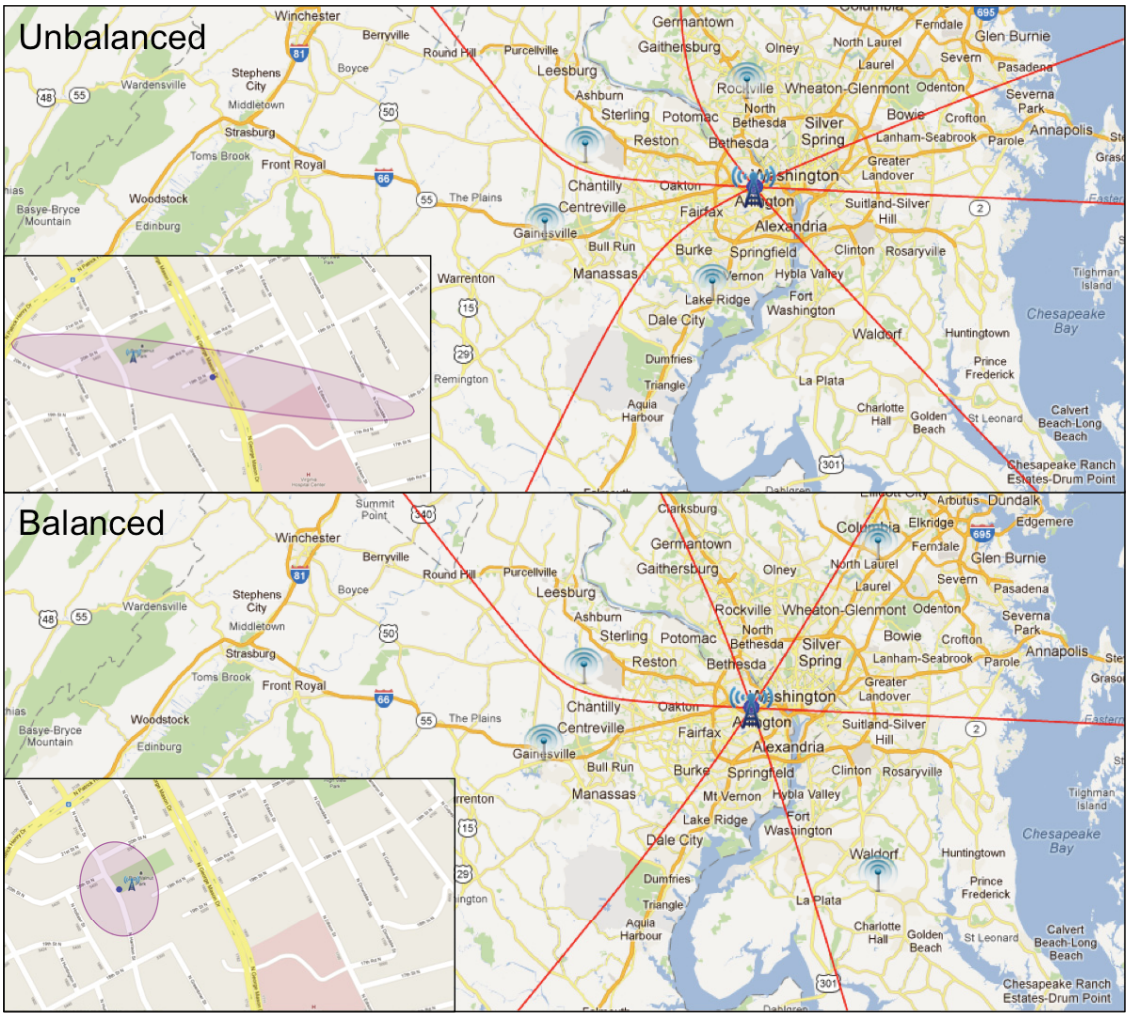

Figure 28 Balanced and unbalanced collector placements

4.1.5 Solving for the Location of the Emitter

The spherical intersection method was used to find an initial guess for the location of the emitter. With the initial guess, the iterative method was used to solve for the final predicted position of the emitters. The covariance matrix for each set of measurements was then transformed to the local coordinate system and used to generate a 95% error ellipse. The geolocation algorithm was implemented in the web application to allow quick reconfiguration of the application for differing locations of simulated collectors.

4.1.6 Analyzing the Results

The web application was used to request and analyze results. The application was opened in the Chrome web browser and each dataset was requested. The data was

verified by loading the data associated with each FM radio station in each dataset (theoretical, unbalanced, and balanced). The actual location of each emitter was plotted, along with the predicted location for each emitter and the 95% error ellipse. The TDOA measurements for each pair of collectors were loaded, their histogram plotted, and statistics analyzed for each FM radio station in each dataset.

4.2 Results

Three sets of results are presented. Theoretically perfect data is used to test the position estimation algorithm. This is followed by position estimations using real and simulated data with the simulated collectors placed in both unbalanced and balanced configurations.

4.2.1 Theoretical Data

The TDOA geolocation algorithm was first tested with theoretical data to ensure that the algorithm worked properly. Theoretical TDOA measurements were generated by first calculating the distance between the actual position of the emitter and the actual location of each collector. The TDOA was then determined by finding the difference between these two distances. These predicted measurements were used to solve for the location of the emitter assuming no error. The results from this experiment are shown in Table 1. The actual locations of the emitters are shown in Table 2. Latitude and longitude are in degrees and altitude is in meters. The results matched very closely with some error in the altitude, resulting from floating-point rounding error.

Table 1 TDOA results with theoretical data

STATION	FREQUENCY	LATITUDE	LONGITUDE	ALTITUDE
WAMU-FM	88.50MHz	38.936111	-77.092499	248.565
WETA-FM	90.90MHz	38.891664	-77.131938	45.299
WGTS-FM	91.90MHz	38.891664	-77.131938	45.299
WKYS-FM	93.90MHz	38.939996	-77.081672	185.223
WIAD-FM	94.70MHz	38.963611	-77.104998	336.588
WASH-FM	97.10MHz	38.950281	-77.079719	327.560

Table 2 Actual location of emitters

STATION	FREQUENCY	LATITUDE	LONGITUDE	ALTITUDE
WAMU-FM	88.50MHz	38.936110	-77.092500	223
WETA-FM	90.90MHz	38.891670	-77.131940	252
WGTS-FM	91.90MHz	38.891670	-77.131940	252
WKYS-FM	93.90MHz	38.940000	-77.081670	286
WIAD-FM	94.70MHz	38.963610	-77.105000	312
WASH-FM	97.10MHz	38.950280	-77.079720	315

4.2.2 Unbalanced Configuration

Position estimations were first calculated with an unbalanced collector arrangement seen in Figure 28. Table 3 shows the mean TDOA measurements and their standard deviation for the 5000 TDOA samples made for both the real and simulated pairs. The mean and standard deviation are both in USRP samples. In each case, the histogram was found to have a Gaussian distribution.

Table 3 TDOA statistics for unbalanced collector placement

STATION	FREQUENCY	COL 1	COL 2	TDOA MEAN	TDOA STD	SIMULATED
WAMU-FM	88.50MHz	ASHBURN	HAYMARKET	-1985.236	4.586	N
		ASHBURN	LORTON	2322.565	2.614	Y
		ASHBURN	ROCKVILLE	4365.900	1.691	Y
WETA-FM	90.90MHz	ASHBURN	HAYMARKET	-1569.462	5.050	N
		ASHBURN	LORTON	3048.071	2.436	Y
		ASHBURN	ROCKVILLE	2914.009	2.518	Y
WGTS-FM	91.90MHz	ASHBURN	HAYMARKET	-1586.060	10.231	N
		ASHBURN	LORTON	3048.124	5.010	Y
		ASHBURN	ROCKVILLE	2913.875	5.152	Y
WKYS-FM	93.90MHz	ASHBURN	HAYMARKET	-1965.269	21.222	N
		ASHBURN	LORTON	2340.867	12.170	Y
		ASHBURN	ROCKVILLE	4581.956	7.397	Y
WIAD-FM	94.70MHz	ASHBURN	HAYMARKET	-2220.577	2.344	N
		ASHBURN	LORTON	1550.633	1.449	Y
		ASHBURN	ROCKVILLE	4786.493	0.702	Y
WASH-FM	97.10MHz	ASHBURN	HAYMARKET	-2077.486	62.827	N
		ASHBURN	LORTON	2126.476	36.799	Y
		ASHBURN	ROCKVILLE	4815.395	20.690	Y

The mean and standard deviation for each TDOA pair were used to solve for the location and 95% error ellipse for each FM radio station, shown in Table 4. The altitude, semi major, and semi minor axis are in meters. The tilt is in degrees.

Table 4 Geolocation solutions for unbalanced collector placement

STATION	FREQUENCY	LATITUDE	LONGITUDE	ALTITUDE	SEMI MAJOR	SEMI MINOR	TILT
WAMU-FM	88.50MHz	38.936526	-77.089678	-3581.827	577.637	23.322	13.778
WETA-FM	90.90MHz	38.890611	-77.126387	-6154.107	232.973	21.491	-10.642
WGTS-FM	91.90MHz	38.891227	-77.129824	-3702.751	460.512	42.588	-10.554
WKYS-FM	93.90MHz	38.947708	-77.041903	-14074.380	4099.278	132.865	14.920
WIAD-FM	94.70MHz	38.964158	-77.103263	-2652.102	535.919	13.743	25.453
WASH-FM	97.10MHz	38.954907	-77.061014	-9035.710	14650.950	375.645	18.790

These results show a precise geolocation estimate despite the unbalanced collector placement as seen in Figure 28. The unbalanced collector arrangement generally made good predictions for the location of the emitter in the earth-tangent direction (latitude and longitude), but resulted in a large miss on the altitude. The 95% error ellipse in the east and north directions remained relatively small, being larger in the axis corresponding to the measurements made from the real collectors.

4.2.3 Balanced Configuration

Position estimates were then computed with a more balanced collector arrangement seen in Figure 28. Table 5 shows the mean TDOA measurements and their standard deviation for the 5000 samples made. The mean and standard deviation are both in USRP samples. In each case, the histogram was found to represent a Gaussian distribution.

Table 5 TDOA statistics for balanced collector placement

STATION	FREQUENCY	COL 1	COL 2	TDOA MEAN	TDOA STD	SIMULATED
WAMU-FM	88.50MHz	ASHBURN	HAYMARKET	-1985.236	4.586	N
		ASHBURN	LORTON	-1415.046	4.302	Y
		ASHBURN	ROCKVILLE	375.863	3.494	Y
WETA-FM	90.90MHz	ASHBURN	HAYMARKET	-1569.462	5.050	N
		ASHBURN	LORTON	-1479.430	5.036	Y
		ASHBURN	ROCKVILLE	-1442.158	4.980	Y
WGTS-FM	91.90MHz	ASHBURN	HAYMARKET	-1586.060	10.231	N
		ASHBURN	LORTON	-1479.238	10.113	Y
		ASHBURN	ROCKVILLE	-1442.159	10.095	Y
WKYS-FM	93.90MHz	ASHBURN	HAYMARKET	-1965.269	21.222	N
		ASHBURN	LORTON	-1206.691	19.161	Y
		ASHBURN	ROCKVILLE	761.771	15.728	Y
WIAD-FM	94.70MHz	ASHBURN	HAYMARKET	-2220.577	2.344	N
		ASHBURN	LORTON	-2365.560	2.364	Y
		ASHBURN	ROCKVILLE	458.203	1.707	Y
WASH-FM	97.10MHz	ASHBURN	HAYMARKET	-2077.486	62.827	N
		ASHBURN	LORTON	-1385.267	58.524	Y
		ASHBURN	ROCKVILLE	995.672	44.239	Y

The mean and standard deviation for each TDOA pair were used to solve for the location and 95% ellipse for each FM radio station, shown in Table 6. The altitude, semi major, and semi minor axis are in meters. The tilt is in degrees.

Table 6 Geolocation solutions for balanced collector placement

STATION	FREQUENCY	LATITUDE	LONGITUDE	ALTITUDE	SEMI MAJOR	SEMI MINOR	TILT
WAMU-FM	88.50MHz	38.936187	-77.092530	-3195.813	65.781	20.080	-83.371
WETA-FM	90.90MHz	38.891488	-77.132798	-6990.493	52.998	42.478	-72.860
WGTS-FM	91.90MHz	38.891550	-77.132262	-4239.078	105.951	84.771	-73.780
WKYS-FM	93.90MHz	38.941739	-77.081587	-11145.248	332.864	94.072	-89.946
WIAD-FM	94.70MHz	38.963650	-77.105025	-1849.543	46.270	9.773	-78.884
WASH-FM	97.10MHz	38.951006	-77.079636	-6711.528	1104.310	270.775	269.360

These results show an even more precise geolocation estimate than with the unbalanced collector placement. Predictions were generally good with smaller, more rounded 95% error ellipses. Again, a large error in altitude is present.

4.2.4 Overall Results

Overall, the results were quite good. The balanced collector placement resulted in better estimates for the emitter location than the unbalanced placement, while both resulted in what appears to be large errors in the altitude direction. In the theoretical case, floating point error introduced enough error into the system to cause relatively large deviations in the altitude estimates. The results above only showed the ellipse cross-section at the surface of the earth. Three-dimensional ellipsoids were generated to show the error in the vertical direction. The 95% error ellipsoids for the unbalanced collector placement can be seen in Table 7 and for the balanced collector placement in Table 8. This explains the much larger error seen for the vertical direction and indicates more ambiguity in the vertical direction resulting from the system of TDOA equations.

Table 7 Ellipsoids for unbalanced placement

STATION	FREQUENCY	SEMI1	SEMI2	SEMI3	TILT EU	TILT NU
WAMU-FM	88.50MHz	5030.659	33.470	18.816	-7.310	-1.808
WETA-FM	90.90MHz	1762.333	32.033	24.216	-8.483	1.614
WGTS-FM	91.90MHz	5497.694	62.985	47.416	-5.358	1.009
WKYS-FM	93.90MHz	11312.877	182.964	109.678	-23.711	-6.692
WIAD-FM	94.70MHz	5244.829	17.742	7.603	-6.054	-2.892
WASH-FM	97.10MHz	50089.594	519.381	272.448	-18.543	-6.519

Table 8 Ellipsoids for balanced placement

STATION	FREQUENCY	SEMI1	SEMI2	SEMI3	TILT EU	TILT NU
WAMU-FM	88.50MHz	22.860	4456.368	43.198	0.111	-0.793
WETA-FM	90.90MHz	29.311	2030.373	56.207	1.093	-0.598
WGTS-FM	91.90MHz	58.579	6500.863	112.746	0.678	-0.378
WKYS-FM	93.90MHz	106.447	197.124	7099.851	-0.031	-2.634
WIAD-FM	94.70MHz	11.193	3649.260	23.497	0.147	-0.730
WASH-FM	97.10MHz	300.733	32705.284	576.132	-0.077	-1.975

An example ellipsoid for 91.9MHz with an unbalanced collector placement is plotted in Figure 29 and shows a much larger error in the vertical direction.

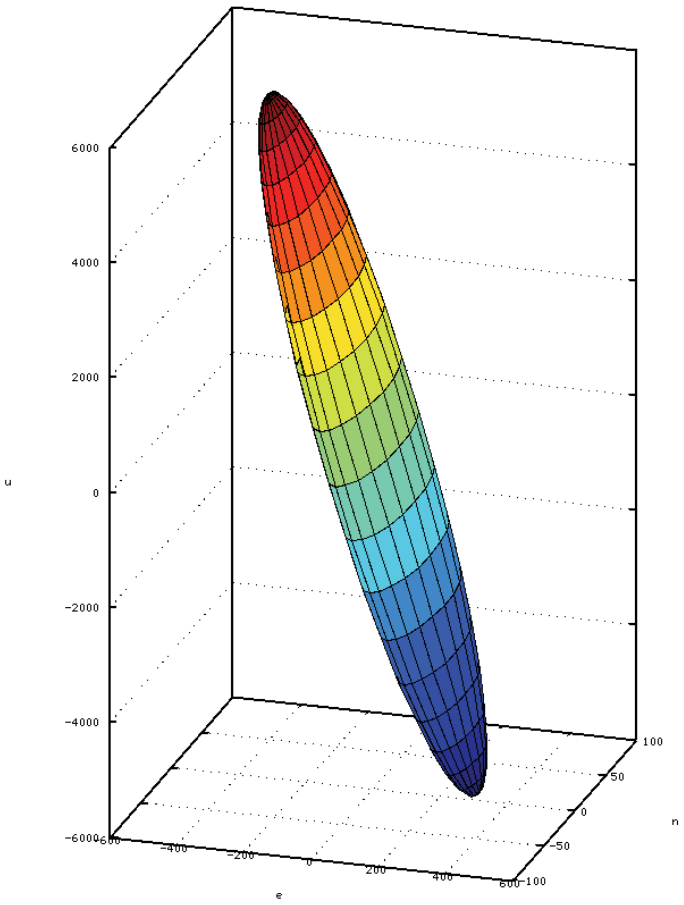

Figure 29 Error ellipsoid for 91.9MHz with unbalanced collector placement

5 Conclusions and Future Work

5.1 Conclusions

Software Defined Radio is changing the radio industry as we know it. Not only does it make experimenting with radio technology more accessible to students and hobbyists, but it also makes radio systems more flexible and able to adapt to the constantly changing field and even have the potential to drive the field due to the ease to which new designs can be prototyped. Frameworks like GNU Radio and OSSIE have allowed developers to easily utilize existing modular components to build complex radio systems that can take advantage of a distributed network of computing systems. This research aimed to take advantage of that distributed possibility and build a useful distributed application using the OSSIE SDR framework and to expand that framework to support this research. The resulting TDOA geolocation system was able to successfully locate emitters by first capturing signals using spatially distributed OSSIE nodes, correlating those signals to produce TDOA measurements, and then solving a system of equations to find the location of the emitter.

After building a spatially distributed OSSIE system, it is clear that this kind of system can be a great benefit to the SDR community. The ability to break up an application into components and spread the processing requirements out among a set of processors and machines is invaluable. The ability to deploy devices and components over a wide area network means that data sources can also be located in different locations. In addition to precision geolocation, there are many other uses for a system like this. One possible use would be a distributed beam-forming system. Another use could be a system to detect and analyze spectrum usages in an urban area.

A distributed OSSIE system requires careful thought because of added hurdles imposed in a wide area network. The limited connection speed and unpredictable latency mean that signals cannot always be transmitted at their full rate or in real time. Because of this, the signals must be processed as much as possible at the source such that the signal/data is reduced in size prior to transmission. For this research, this entailed decimating the

collected signal to a band-limited signal to reduce its data rate prior to transmission. Transmitting only required snapshots of the signal data further reduced the data rate. Several OSSIE software components were built to buffer and transmit snapshot data over the slow and unreliable link. Another issue was controlling a set of waveforms deployed to an OSSIE Domain. The TDOA waveform built for this research was built with the idea that if more radio nodes were available, multiple TDOA waveforms could be deployed to the same OSSIE domain. Direct coupling of the GUI to a waveform using the provided CORBA interfaces can be cumbersome, especially for a system with many waveforms. The CORBA Event Service makes it relatively simple to interact with a set of waveforms from a single, decoupled GUI.

The results showed that the USRP device could be used as a platform for TDOA signal collection. While not the ideal collectors for performing precision TDOA geolocation, the workarounds necessary to use the USRP boards without permanent hardware modifications proved to be a good learning experience. Even when the emitters were placed in an unbalanced arrangement to magnify the effects of the real TDOA measurement, the results were fairly accurate. The web application developed to visualize results proved to be invaluable in more ways than simply being a unique way to deliver results. The web application provided a streamlined way to load result sets from the collection system and analyze different simulated scenarios, all by simply changing an XML configuration file. The web application also provided the source for many of the tables and figures found in this research.

5.2 Future Work

The research presented in this paper concentrated on building an overall system. Time was not available to concentrate on any one aspect of the system. There are several areas of this research that could be expanded and improved upon in the future.

5.2.1 Implementation with a Full Set of Collectors

Because hardware was not available for this research, there were not enough collectors available to perform a TDOA geolocation without simulating several collectors. Future

research could add several collectors to the system in order to provide a complete TDOA geolocation system. These collectors could either be more USRP boards or more advanced collectors with built-in time references. Multiple TDOA waveforms could be deployed to the distributed network of collectors. All waveforms could then be triggered to make TDOA collections by the same decoupled, event-based user interface.

5.2.2 Analysis of Bias and Error in the System

From the results, it is clear that some amount of bias was present in the statistics, as the mean of the TDOA measurements did not always match exactly with the actual location of the emitter. The cable length delay was taken into account for the signal reception, but a more formal analysis could be performed to determine biases in the system. Specific sources of random error that contributed to the variance in TDOA measurements could also be found and analyzed.

5.2.3 Java OSSIE Development

Another area for improvement would be a formal integration of Java into the OSSIE framework. Some functionality was implemented for this research, but this could be extended to create a complete toolkit for Java users. This would open up OSSIE to a whole new set of developers with Java skills. An application for this would possibly be integration of OSSIE waveforms into interactive web applications, similar to the one built for displaying the results of this research. The web application could interact with, control, and display live results from installed waveforms.

5.2.4 Improvements in OSSIE to Support Distributed Applications

This research made improvements to OSSIE where necessary to support the distributed application built for this research. This work could be expanded to improve OSSIE to allow easier development of distributed applications. The concept of utilizing snapshot data could be integrated into the standard set of OSSIE *Ports* to add the signaling and buffering that is required to transmit data over a slower network. Work could also be completed to make it easier to deploy many *Device* nodes of the same type. For this research, two or more *USRP_TIME*/*GPP Device* nodes were required. Separate nodes had to be defined with different IDs for each *Device* so that the waveform components

would be deployed to the correct location. Nodes could be built that would allow deployment simultaneously across many nodes with minimal configuration changes. Automatic resource allocation could be implemented in OSSIE to allow automatic deployment of waveforms rather than relying on a predefined DAS file to specify where components are deployed.

References

[1] Jeffrey H. Reed, *Software Radio: A Modern Approach to Radio Engineering*. Upper Saddle River, NJ: Prentice Hall PTR, 2002.

[2] GNU Radio. (2012, April) GNU Radio - USRP. [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/USRP

[3] Wikipedia: The Free Encyclopedia. (2012, April) Software framework. [Online]. Available: http://en.wikipedia.org/wiki/Software_framework

[4] GNU Radio. (2012, April) GNU Radio Wiki. [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki

[5] Joint Program Executive Office. (2006, May) SOFTWARE COMMUNICATIONS ARCHITECTURE SPECIFICATION. [Online]. Available: http://jpeojtrs.mil/sca/Documents/SCAv2_2_2/SCA_version_2_2_2.pdf

[6] Wikipedia: The Free Encyclopedia. (2012, March) Joint Tactical Radio System. [Online]. Available: http://en.wikipedia.org/wiki/Joint_Tactical_Radio_System

[7] Seymour Stein, "Algorithms for Ambiguity Function Processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-29, pp. 588-599, June 1981.

[8] Carl B. Dietrich et al. (2010, March) Experiences From the OSSIE Open Source Software Defined Radio Project. [Online]. Available: http://www.osbr.ca/ojs/index.php/osbr/article/view/1054/1013

[9] WikiAudio.org. (2012, March) FM broadcasting. [Online]. Available: http://en.wikiaudio.org/FM_broadcasting

[10] Ettus Research. (2012, April) WBX 50-2200 MHz Rx/Tx. [Online]. Available: https://www.ettus.com/product/details/WBX

[11] Matt Ettus et al. (2010, Sept.) USRP inband FPGA code. [Online]. Available: https://github.com/etschneider/usrp-fpga-inband

[12] Ettus Research. (2010, December) USRP N210 and N200 schematic. [Online]. Available: http://code.ettus.com/redmine/ettus/attachments/87/n210.pdf

[13] Trimble Navigation Limited. (2000, September) Thunderbolt GPS Disciplined Clock Manual. [Online]. Available: http://www.novotech.com/productinfo/trimble/Timing/Thunderbolt%20Manual.pdf

[14] Object Management Group. (2004, October) Event Service Specification. [Online]. Available: http://www.omg.org/spec/EVNT/1.2/PDF/

[15] Matt Carrick et al. (2010, September) OSSIE 0.8.1 Installation and User Guide. [Online]. Available: http://ossie.wireless.vt.edu/download/user_guides/OSSIE_0.8.1_User_Guide.pdf

[16] Wikipedia: The Free Encyclopedia. (2012, April) C++. [Online]. Available: http://en.wikipedia.org/wiki/C++

[17] Wikipedia: The Free Encyclopedia. (2012, April) Python (programming language). [Online]. Available: http://en.wikipedia.org/wiki/Python_(programming_language)

[18] Wikipedia: The Free Encyclopedia. (2012, April) Java (programming language). [Online]. Available: http://en.wikipedia.org/wiki/Java_(programming_language)

[19] omniORB. (2012, Jan.) omniORB Frequently Asked Questions. [Online]. Available: http://www.omniorb-support.com/omniwiki/FrequentlyAskedQuestions

[20] Richard G. Lyons, *Understanding Digital Signal Processing*. Upper Saddle River, NJ: Prentice Hall, 2011.

[21] Fredric J. Harris, *Multirate Signal processing for Communications Systems*. Upper Saddle River, NJ: Prentice Hall PTR, 2008.

[22] Joseph D. Gaeddert. (2011) liquid: software defined radio signal processing library User's Manual. [Online]. Available: https://github.com/downloads/jgaeddert/liquid-dsp/liquid-dsp-1.0.0.pdf

[23] Huai-Jing Du and Jim P.Y. Lee, "Passive Geolocation Using TDOA Method from UAVs and Ship/Land-Based Platforms for Maritime and Littoral Area Surveillance," Defence R&D Canada - Ottawa, Ottawa, 2004. [Online]. http://pubs.drdc.gc.ca/PDFS/unc21/p521218.pdf

[24] Jonathan S. Abel and Julius O. Smith, "Closed-Form Least-Squares Source Location Estimation from Range-Difference Measurements," *IEEE Transactions on*

Acoustics, Speech, and Signal Processing, vol. ASSP-35, no. 12, p. 1661, December 1987.

[25] Wikipedia: The Free Encyclopedia. (2012, March) Geodetic system. [Online]. Available: http://en.wikipedia.org/wiki/Geodetic_system

[26] Maria Isabel Ribeiro, "Gaussian Probability Density Functions: Properties and Error Characterization," Institute for Systems and Robotics, Instituto Superior Tcnico, Lisboa, Portugal, February 2004.

[27] Geoffrey Blewitt, "Basics of the GPS Technique: Observation Equations," Department of Geomatics, University of Newcastle, Newcastle upon Tyne, 1997.

[28] Google. (2012, April) Google Web Toolkit. [Online]. Available: https://developers.google.com/web-toolkit/

[29] JAMA: A Java Matrix Package. [Online]. Available: http://math.nist.gov/javanumerics/jama/

[30] NeXtMidas. [Online]. Available: http://nextmidas.techma.com/

[31] Jason T. Isaacs, Daniel J. Klein, and Joao P. Hesphanha, "Optimal Sensor Placement For Time DIfference of Arrival Localization,".

APPENDIX A. TDOA Waveform Software Assembly Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE softwareassembly SYSTEM "../../xml/dtd/softwareassembly.dtd">
<!-- Created with OSSIE WaveDev-->
<!--Powered by Python-->
<softwareassembly id="DCE:899cbfec-1ea9-11e1-a6e5-000c297957ca"
name="OSSIE::ThesisWaveform">
  <componentfiles>
    <componentfile id="Throttler_5b1d57f4-187c-11e1-b36c-000c297957ca" type="SPD">
      <localfile name="/xml/Throttler/Throttler.spd.xml"/>
    </componentfile>
    <componentfile id="WFMDemod_9f5b8058-1ea8-11e1-b0b8-000c297957ca" type="SPD">
      <localfile name="/xml/WFMDemod/WFMDemod.spd.xml"/>
    </componentfile>
    <componentfile id="Resampler_fc93fc6a-187b-11e1-9d31-000c297957ca" type="SPD">
      <localfile name="/xml/Resampler/Resampler.spd.xml"/>
    </componentfile>
    <componentfile id="Correlator_fb88164e-187b-11e1-9ce8-000c297957ca" type="SPD">
      <localfile name="/xml/Correlator/Correlator.spd.xml"/>
    </componentfile>
    <componentfile id="USRP_Commander_0dccc114-189b-11e1-be79-000c297957ca" type="SPD">
      <localfile name="/xml/USRP_Commander/USRP_Commander.spd.xml"/>
    </componentfile>
  </componentfiles>
  <partitioning>
    <componentplacement>
      <componentfileref refid="Throttler_5b1d57f4-187c-11e1-b36c-000c297957ca"/>
      <componentinstantiation id="DCE:89a15aca-1ea9-11e1-96be-000c297957ca">
        <usagename>Throttler_1</usagename>
        <findcomponent>
          <namingservice name="Throttler_1"/>
        </findcomponent>
      </componentinstantiation>
    </componentplacement>
    <componentplacement>
      <componentfileref refid="WFMDemod_9f5b8058-1ea8-11e1-b0b8-000c297957ca"/>
      <componentinstantiation id="DCE:89ab125e-1ea9-11e1-9d01-000c297957ca">
        <usagename>WFMDemod_1</usagename>
        <findcomponent>
          <namingservice name="WFMDemod_1"/>
        </findcomponent>
      </componentinstantiation>
    </componentplacement>
    <componentplacement>
      <componentfileref refid="Throttler_5b1d57f4-187c-11e1-b36c-000c297957ca"/>
      <componentinstantiation id="DCE:89b1b848-1ea9-11e1-b3b4-000c297957ca">
        <usagename>Throttler_2</usagename>
        <findcomponent>
          <namingservice name="Throttler_2"/>
        </findcomponent>
      </componentinstantiation>
    </componentplacement>
    <componentplacement>
```

```
      <componentfileref refid="WFMDemod_9f5b8058-1ea8-11e1-b0b8-000c297957ca"/>
      <componentinstantiation id="DCE:89bb60aa-1ea9-11e1-8705-000c297957ca">
        <usagename>WFMDemod_2</usagename>
        <findcomponent>
          <namingservice name="WFMDemod_2"/>
        </findcomponent>
      </componentinstantiation>
    </componentplacement>
    <componentplacement>
      <componentfileref refid="Resampler_fc93fc6a-187b-11e1-9d31-000c297957ca"/>
      <componentinstantiation id="DCE:89c292ee-1ea9-11e1-b93d-000c297957ca">
        <usagename>Resampler_1</usagename>
        <findcomponent>
          <namingservice name="Resampler_1"/>
        </findcomponent>
      </componentinstantiation>
    </componentplacement>
    <componentplacement>
      <componentfileref refid="Resampler_fc93fc6a-187b-11e1-9d31-000c297957ca"/>
      <componentinstantiation id="DCE:89c95f70-1ea9-11e1-bd15-000c297957ca">
        <usagename>Resampler_2</usagename>
        <findcomponent>
          <namingservice name="Resampler_2"/>
        </findcomponent>
      </componentinstantiation>
    </componentplacement>
    <componentplacement>
      <componentfileref refid="Correlator_fb88164e-187b-11e1-9ce8-000c297957ca"/>
      <componentinstantiation id="DCE:89d03066-1ea9-11e1-ab55-000c297957ca">
        <usagename>Correlator_1</usagename>
        <findcomponent>
          <namingservice name="Correlator_1"/>
        </findcomponent>
      </componentinstantiation>
    </componentplacement>
    <componentplacement>
      <componentfileref refid="USRP_Commander_0dccc114-189b-11e1-be79-000c297957ca"/>
      <componentinstantiation id="DCE:89d8d446-1ea9-11e1-b9c3-000c297957ca">
        <componentproperties>
          <simpleref description="" name="rx_gain_max" refid="DCE:2d9c5ee4-a6f3-4ab9-834b-
2b5c95818e53" value="0"/>
          <simpleref description="" name="rx_gain" refid="DCE:99d586b6-7764-4dc7-83fa-
72270d0f1e1b" value="36"/>
          <simpleref description="" name="rx_freq" refid="DCE:3efc3930-2739-40b4-8c02-
ecfb1b0da9ee" value="92500000"/>
          <simpleref description="" name="rx_decim" refid="DCE:92ec2b80-8040-47c7-a1d8-
4c9caa4a4ed2" value="256"/>
        </componentproperties>
        <usagename>USRP_Commander_1</usagename>
        <findcomponent>
          <namingservice name="USRP_Commander_1"/>
        </findcomponent>
      </componentinstantiation>
    </componentplacement>
    <componentplacement>
```

```
        <componentfileref refid="USRP_Commander_0dccc114-189b-11e1-be79-000c297957ca"/>
        <componentinstantiation id="DCE:89df592e-1ea9-11e1-9b8c-000c297957ca">
          <componentproperties>
            <simpleref description="" name="rx_gain_max" refid="DCE:2d9c5ee4-a6f3-4ab9-834b-
2b5c95818e53" value="0"/>
            <simpleref description="" name="rx_gain" refid="DCE:99d586b6-7764-4dc7-83fa-
72270d0f1e1b" value="65"/>
            <simpleref description="" name="rx_freq" refid="DCE:3efc3930-2739-40b4-8c02-
ecfb1b0da9ee" value="92500000"/>
            <simpleref description="" name="rx_decim" refid="DCE:92ec2b80-8040-47c7-a1d8-
4c9caa4a4ed2" value="256"/>
          </componentproperties>
          <usagename>USRP_Commander_2</usagename>
          <findcomponent>
            <namingservice name="USRP_Commander_2"/>
          </findcomponent>
        </componentinstantiation>
      </componentplacement>
    </partitioning>
    <assemblycontroller>
      <componentinstantiationref refid="DCE:89a15aca-1ea9-11e1-96be-000c297957ca"/>
    </assemblycontroller>
    <connections>
      <connectinterface id="DCE:89a4b724-1ea9-11e1-9b04-000c297957ca">
        <providesport>
          <providesidentifier>IN</providesidentifier>
          <findby>
            <namingservice name="Throttler_1"/>
          </findby>
        </providesport>
        <usesport>
          <usesidentifier>RX_Data_1</usesidentifier>
          <findby>
            <namingservice name="USRP_TIME_1"/>
          </findby>
        </usesport>
      </connectinterface>
      <connectinterface id="DCE:89a7b578-1ea9-11e1-be74-000c297957ca">
        <providesport>
          <providesidentifier>dataIn</providesidentifier>
          <findby>
            <namingservice name="WFMDemod_1"/>
          </findby>
        </providesport>
        <usesport>
          <usesidentifier>OUT</usesidentifier>
          <findby>
            <namingservice name="Throttler_1"/>
          </findby>
        </usesport>
      </connectinterface>
      <connectinterface id="DCE:89ae9bae-1ea9-11e1-a71a-000c297957ca">
        <providesport>
          <providesidentifier>IN</providesidentifier>
          <findby>
```

```
        <namingservice name="Resampler_1"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>dataOut</usesidentifier>
      <findby>
        <namingservice name="WFMDemod_1"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:89b4b12e-1ea9-11e1-a2e5-000c297957ca">
    <providesport>
      <providesidentifier>dataIn</providesidentifier>
      <findby>
        <namingservice name="WFMDemod_2"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>OUT</usesidentifier>
      <findby>
        <namingservice name="Throttler_2"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:89b7ce22-1ea9-11e1-885d-000c297957ca">
    <providesport>
      <providesidentifier>IN</providesidentifier>
      <findby>
        <namingservice name="Throttler_2"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>RX_Data_1</usesidentifier>
      <findby>
        <namingservice name="USRP_TIME_2"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:89bf564c-1ea9-11e1-9b27-000c297957ca">
    <providesport>
      <providesidentifier>IN</providesidentifier>
      <findby>
        <namingservice name="Resampler_2"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>dataOut</usesidentifier>
      <findby>
        <namingservice name="WFMDemod_2"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:89c5dee0-1ea9-11e1-b7e2-000c297957ca">
    <providesport>
      <providesidentifier>IN1</providesidentifier>
```

```
      <findby>
        <namingservice name="Correlator_1"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>OUT</usesidentifier>
      <findby>
        <namingservice name="Resampler_1"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:89cd5daa-1ea9-11e1-9e0f-000c297957ca">
    <providesport>
      <providesidentifier>IN2</providesidentifier>
      <findby>
        <namingservice name="Correlator_1"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>OUT</usesidentifier>
      <findby>
        <namingservice name="Resampler_2"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:99d348a0-1ea9-11e1-97dc-000c297957ca">
    <providesport>
      <providesidentifier>TRIGGER_IN</providesidentifier>
      <findby>
        <namingservice name="Throttler_2"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>TRIGGER_OUT</usesidentifier>
      <findby>
        <namingservice name="Correlator_1"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:89d348a0-1ea9-11e1-97dc-000c297957ca">
    <providesport>
      <providesidentifier>TRIGGER_IN</providesidentifier>
      <findby>
        <namingservice name="Throttler_1"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>TRIGGER_OUT</usesidentifier>
      <findby>
        <namingservice name="Correlator_1"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:89dbcca0-1ea9-11e1-b0a0-000c297957ca">
    <providesport>
```

```
      <providesidentifier>RX_Control</providesidentifier>
      <findby>
        <namingservice name="USRP_TIME_1"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>RX_Control</usesidentifier>
      <findby>
        <namingservice name="USRP_Commander_1"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:89e279ce-1ea9-11e1-901d-000c297957ca">
    <providesport>
      <providesidentifier>RX_Control</providesidentifier>
      <findby>
        <namingservice name="USRP_TIME_2"/>
      </findby>
    </providesport>
    <usesport>
      <usesidentifier>RX_Control</usesidentifier>
      <findby>
        <namingservice name="USRP_Commander_2"/>
      </findby>
    </usesport>
  </connectinterface>
  <connectinterface id="DCE:89dbcca1-1ea9-11e1-b0a0-000c297957ca">
    <usesport>
      <usesidentifier>EVENTS_IN</usesidentifier>
      <findby>
        <namingservice name="USRP_Commander_1"/>
      </findby>
    </usesport>
    <findby>
      <domainfinder type="eventchannel" name="THESIS_EVENTS"/>
    </findby>
  </connectinterface>
  <connectinterface id="DCE:89dbcca2-1ea9-11e1-b0a0-000c297957ca">
    <usesport>
      <usesidentifier>EVENTS_IN</usesidentifier>
      <findby>
        <namingservice name="USRP_Commander_2"/>
      </findby>
    </usesport>
    <findby>
      <domainfinder type="eventchannel" name="THESIS_EVENTS"/>
    </findby>
  </connectinterface>
  <connectinterface id="DCE:89dbcca3-1ea9-11e1-b0a0-000c297957ca">
    <usesport>
      <usesidentifier>EVENTS_IN</usesidentifier>
      <findby>
        <namingservice name="Correlator_1"/>
      </findby>
    </usesport>
```

```
        <findby>
          <domainfinder type="eventchannel" name="THESIS_EVENTS"/>
        </findby>
      </connectinterface>
      <connectinterface id="DCE:89dbcca5-1ea9-11e1-b0a0-000c297957ca">
        <usesport>
          <usesidentifier>EVENTS_OUT</usesidentifier>
          <findby>
            <namingservice name="Correlator_1"/>
          </findby>
        </usesport>
        <findby>
          <domainfinder type="eventchannel" name="THESIS_EVENTS"/>
        </findby>
      </connectinterface>
    </connections>
</softwareassembly>
```

APPENDIX B. TDOA Geolocation Algorithm Implementation

```
package com.meuleners.thesis.webapp.server.util;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import nxm.sys.inc.Constants;
import nxm.sys.lib.Position;
import nxm.sys.lib.Transform;
import Jama.EigenvalueDecomposition;
import Jama.Matrix;

import com.meuleners.thesis.webapp.shared.beans.GeoLocation;

public class GeoUtil {
 public static final double SPEED_OF_LIGHT = 299792458;
 public static final double EARTH_FLATTENING_FACTOR = 1 / 298.257223563d;
 public static final double EARTH_EQUITORIAL_RADIUS = 6378137.00;
 public static final double A = EARTH_EQUITORIAL_RADIUS;
 public static final double C = A * (1 - EARTH_FLATTENING_FACTOR);

 /**
  * Computes the distance in meters from one position to another.
  *
  * @param p1
  *        Position 1
  * @param p2
  *        Position 2
  * @return distance in meters
  */
 private static double dist(Position p1, Position p2) {
  Transform.geo2car(p1);
  Transform.geo2car(p2);

  double dx = (p1.x - p2.x);
  double dy = (p1.y - p2.y);
  double dz = (p1.z - p2.z);

  double dist = Math.sqrt(dx * dx + dy * dy + dz * dz);

  return dist;
 }

 /**
  * Predict location of emitter using the Spherical Interpolation method.
  *
  * @param p      Positions of collectors
  * @param r      RDOAs (length = p.length - 1)
  * @return Position of emitter
  */
 public static Position predictSX(Position[] p, double[] r) {
  assert (p.length == r.length - 1);
```

```
 for (int i = 0; i < p.length; i++)
  Transform.geo2car(p[i]);

 double[][] s = new double[r.length][3];
 double[][] de = new double[r.length][1];
 double[][] delta = new double[r.length][1];
 for (int i = 0; i < r.length; i++) {
  s[i][0] = p[i + 1].x - p[0].x;
  s[i][1] = p[i + 1].y - p[0].y;
  s[i][2] = p[i + 1].z - p[0].z;
  de[i][0] = r[i];
  double R = dist(p[i + 1], p[0]);
  delta[i][0] = Math.pow(R, 2) - Math.pow(r[i], 2);
 }

 Matrix S = new Matrix(s);
 Matrix STSIS = S.transpose().times(S).inverse().times(S.transpose());
 Matrix De = new Matrix(de);
 Matrix Delta = new Matrix(delta);

 double a = 4 - De.times(4).transpose().times(STSIS.transpose()).times(STSIS)
    .times(De).get(0, 0);
 double b = De.times(4).transpose().times(STSIS.transpose()).times(STSIS)
    .times(Delta).get(0, 0);
 double c = -Delta.transpose().times(STSIS.transpose()).times(STSIS)
    .times(Delta).get(0, 0);
 double R = (-b + Math.sqrt(Math.pow(b, 2) - 4 * a * c)) / (2 * a);

 System.out.println("A: " + a + " B: " + b + " C: " + c + " R: " + R);

 Matrix X = STSIS.times(Delta.minus(De.times(2).times(R))).times(0.5);

 X.print(10, 5);r

 Position rVal = new Position();

 rVal.setCar(X.get(0, 0) + p[0].x, X.get(1, 0) + p[0].y, X.get(2, 0)
    + p[0].z);

 Transform.car2geo(rVal);

 return rVal;
}

public static Matrix computeJacobian(double xg, double yg, double zg,
   Position[] p1, Position[] p2, int len, boolean onEarth) {
 double[][] jacobian = new double[len + (onEarth ? 1 : 0)][3];
 for (int j = 0; j < len; j++) {
  jacobian[j][0] = (xg - p1[j].x)
     / Math.sqrt(Math.pow(zg - p1[j].z, 2) + Math.pow(yg - p1[j].y, 2)
        + Math.pow(xg - p1[j].x, 2))
     - (xg - p2[j].x)
     / Math.sqrt(Math.pow(zg - p2[j].z, 2) + Math.pow(yg - p2[j].y, 2)
        + Math.pow(xg - p2[j].x, 2));
```

```
    jacobian[j][1] = (yg - p1[j].y)
        / Math.sqrt(Math.pow(zg - p1[j].z, 2) + Math.pow(yg - p1[j].y, 2)
            + Math.pow(xg - p1[j].x, 2))
        - (yg - p2[j].y)
        / Math.sqrt(Math.pow(zg - p2[j].z, 2) + Math.pow(yg - p2[j].y, 2)
            + Math.pow(xg - p2[j].x, 2));
    jacobian[j][2] = (zg - p1[j].z)
        / Math.sqrt(Math.pow(zg - p1[j].z, 2) + Math.pow(yg - p1[j].y, 2)
            + Math.pow(xg - p1[j].x, 2))
        - (zg - p2[j].z)
        / Math.sqrt(Math.pow(zg - p2[j].z, 2) + Math.pow(yg - p2[j].y, 2)
            + Math.pow(xg - p2[j].x, 2));
  }
  if (onEarth) {
    // Initialize ellipsoid parameters
    jacobian[jacobian.length - 1][0] = (2 * xg) / Math.pow(A, 2);
    jacobian[jacobian.length - 1][1] = (2 * yg) / Math.pow(A, 2);
    jacobian[jacobian.length - 1][2] = (2 * zg) / Math.pow(C, 2);
  }
  return new Matrix(jacobian);
}

/**
 * Predict location of emitter based on TDOA measurements
 *
 * @param p1            Array of collector Positions for collector 1
 * @param p2            Array of collector Positions for collector 2
 * @param m             Array of TDOA measurements, in seconds
 * @param e             Array of TDOA error standard deviations, in seconds
 * @param start         Position to start iteration
 * @param maxError      Maximum error for iteration.
 * @param onEarth       On the surface of the earth?
 * @param weightWithInverse
 *        Should the inverse measurement covariance matrix be used for
 *        weighting?
 * @return GeoLocation product including position of emitter and error ellipse
 */
public static GeoLocation predictTDOA(Position[] p1, Position[] p2,
    double[] m, double[] e, Position start, double maxError, boolean onEarth,
    boolean weightWithInverse) {
  // Check to make sure all arrays are the same lenght
  assert (p1.length == p2.length && p1.length == m.length && p1.length == e.length);

  if (onEarth) {
    double[] et = new double[e.length + 1];
    System.arraycopy(e, 0, et, 0, e.length);
    e = et;
    e[e.length - 1] = 0.000000001;
  }

  // Convert TDOA to RDOA
  double[] r = new double[m.length + (onEarth ? 1 : 0)];
  for (int i = 0; i < m.length; i++)
    r[i] = m[i] * SPEED_OF_LIGHT;
```

```
// Transform positions to CARTESION (XYZ) Coordinates
for (int i = 0; i < p1.length; i++)
  Transform.geo2car(p1[i]);
for (int i = 0; i < p2.length; i++)
  Transform.geo2car(p2[i]);
Transform.geo2car(start);

// initialize initial guess
double xg = start.x;
double yg = start.y;
double zg = start.z;

GeoLocation geo = new GeoLocation();

// SET INITIAL POSITION FOR GEO
geo.setInitialPosition(start);

// KEEP TRACK OF INTERIM POSITIONS ON WAY TO FINAL SOLUTION
List<Position> interim = new ArrayList<Position>();

Position position = new Position(Position.GEODETIC);
position.setCar(xg, yg, zg);
position.referenceFrame = Position.GEODETIC;
position.car2geo();
interim.add(position);

// Build the measurement covariance matrixpre
double[][] covM = new double[e.length][e.length];
for (int i = 0; i < e.length; i++) {
  for (int j = 0; j < e.length; j++) {
    if (j == i)
      covM[i][j] = Math.pow(e[i] * SPEED_OF_LIGHT, 2);
    else
      covM[i][j] = 0;
  }
}
Matrix CovM = new Matrix(covM);

double len = 100;
for (int i = 0; i < 400; i++) {

  double[][] deltaM = new double[r.length][1];
  for (int j = 0; j < m.length; j++) {
    double wy = Math.sqrt(Math.pow((xg - p1[j].x), 2)
        + Math.pow((yg - p1[j].y), 2) + Math.pow((zg - p1[j].z), 2))
        - Math.sqrt(Math.pow((xg - p2[j].x), 2)
            + Math.pow((yg - p2[j].y), 2) + Math.pow((zg - p2[j].z), 2));
    deltaM[j][0] = r[j] - wy;
  }
  if (onEarth) {
    deltaM[r.length - 1][0] = 1 - (Math.pow(xg, 2) + Math.pow(yg, 2))
        / Math.pow(A, 2) + Math.pow(zg, 2) / Math.pow(C, 2);
  }
  Matrix DeltaM = new Matrix(deltaM);
```

```
  // Build the jacobian, approximating the set of TDOA equations with
  // the first order terms of the Taylor Series expansion
  Matrix J = computeJacobian(xg, yg, zg, p1, p2, m.length, onEarth);

  Matrix W = (weightWithInverse) ? CovM.inverse() : Matrix.identity(
     e.length, e.length);

  Matrix DeltaX = J.transpose().times(W).times(J).inverse()
     .times(J.transpose()).times(W).times(DeltaM);

  xg = xg + DeltaX.get(0, 0);
  yg = yg + DeltaX.get(1, 0);
  zg = zg + DeltaX.get(2, 0);

  // FIND THE MAGNATUDE OF DeltaX
  len = Math.sqrt(Math.pow(DeltaX.get(0, 0), 2)
     + Math.pow(DeltaX.get(1, 0), 2) + Math.pow(DeltaX.get(2, 0), 2));

  position = new Position();
  position.setCar(xg, yg, zg);
  position.referenceFrame = Position.GEODETIC;
  position.car2geo();
  interim.add(position);

  // IF DeltaX is sufficiently small, then quit
  if (len < maxError)
    break;
  if (Double.isNaN(len))
    break;

}

geo.setInterim(interim);

if (Double.isNaN(len) || len > 1)
  geo.setSolutionFound(false);
else
  geo.setSolutionFound(true);

Position rVal = new Position();
rVal.setX(xg);
rVal.setY(yg);
rVal.setZ(zg);
Transform.car2geo(rVal);
rVal.referenceFrame = Position.GEODETIC;

// Build the Jacobian once more for the final value of xg
Matrix J = computeJacobian(xg, yg, zg, p1, p2, m.length, onEarth);

// Compute covariance in XYZ coordinates
Matrix JTJI = J.transpose().times(J).inverse();

Matrix CovX = JTJI.times(J.transpose()).times(CovM).times(J).times(JTJI);

// Build Rotation matrix to rotate from earth centered to local ENU
```

```
  // (East-North-Up) Coordinates
  double sLat = Math.sin(Constants.DEG2RAD * rVal.lat);
  double cLat = Math.cos(Constants.DEG2RAD * rVal.lat);
  double sLon = Math.sin(Constants.DEG2RAD * rVal.lon);
  double cLon = Math.cos(Constants.DEG2RAD * rVal.lon);
  double[][] rot = { { -sLon, cLon, 0 },
     { -sLat * cLon, -sLat * sLon, cLat },
     { cLat * cLon, cLat * sLon, sLat } };
  Matrix R = new Matrix(rot);

  // Compute ENU covariance matrix
  Matrix CovENU = R.times(CovX).times(R.transpose());

  // Select the EN part of the ENU matrix
  Matrix covEN = CovENU.getMatrix(0, 1, 0, 1);

  // Compute Eigenvalues and Eigenvectors of covariance matrix
  EigenvalueDecomposition eig = covEN.eig();

  Matrix V = eig.getV();

  eig.getV().print(10, 5);
  // Compute tilt angle for ellipse
  double ang1 = Constants.RAD2DEG * Math.atan2(V.get(0, 0), V.get(1, 0));
  double ang2 = Constants.RAD2DEG * Math.atan2(V.get(0, 1), V.get(1, 1));

  double[] vals = eig.getRealEigenvalues();

  // Compute axis lengths of Error Ellipse
  double a1 = 2.447 * Math.sqrt(vals[0]);
  double a2 = 2.447 * Math.sqrt(vals[1]);

  if (a1 > a2) {
    geo.setPredTgt(rVal);
    geo.setSemiMajor(a1);
    geo.setSemiMinor(a2);
    geo.setTilt(90 - ang1);
  } else {
    geo.setPredTgt(rVal);
    geo.setSemiMajor(a2);
    geo.setSemiMinor(a1);
    geo.setTilt(90 - ang2);
    }
return geo;
  }
}
```

APPENDIX C. Polyphase Resampler Octave Code

Polyphaseresampler.m

```
clc;clear all;close all;

# Set up

#set ratio for polyphase filter
interp=256.06

# Number of Phases on polyphase Filter
M = 64;

# Set up Source Sampling rate, filter design sampling rate
# and final sampling rate

fs1=250;
fs2=M * fs1
fs3=interp * fs1

# Compute Nyquist Rates

fn1= fs1/2;
fn2= fs2/2;
fn3= fs3/2;


# Number of taps per filter phase
PS=5;

n=M*PS;

# build Kaiser windowed FIR filter
ftype='low'
fstop=75
Wn=fstop/fn3
sba = 100
# From http://www.mathworks.com/help/toolbox/signal/ref/kaiserord.html
if (sba > 50)
        beta = 0.1102*(sba-8.7);
elseif (sba > 21)
        beta = 0.5842*(sba-21)^0.4 + 0.07886*(sba-21);
else
        beta = 0;
end
```

```
filt = M*fir1(n,Wn,ftype,kaiser(n+1,beta),'noscale');

# Build filter partitions
partfilt = zeros(M,PS);
for i=1:M
        for j=1:PS
                partfilt(i,j) = filt(i+(j-1)*M);
        end

end


# Build initial signal
t=0:1/fs1:1;
x1 = sin(2*pi*72*t);
x2 = sin(2*pi*35*t);
x = x1+x2;

# Up sample signal and prepare for filtering with control case
xup = upsample(x,M);

# Filter signal for control case
y = filter(filt,1,xup);

# Filter and up sample signal with multirate method
y1=zeros(1,length(x)*interp);
yindex = 1;
xinterval=M/interp;
xindex=1;
xindexminor = 0;

for i=1:length(x)
        while (xindexminor < M)
                if (i>PS)
                        xf = floor(xindexminor+1);
                        for k=1:PS
                                y1(yindex) = y1(yindex)+x(xindex-k+1)*partfilt(xf,k);
                        end
                        yindex++;
                end
                xindexminor=xindexminor + xinterval;
        end
        xindexminor=xindexminor-M;
        xindex++;
end
```

```
figure(1)
clf
plot(x)
title('Original Signal');

figure(2)
clf
plot(y(1:75*M))
figure(3)
title('Control Filtered Signal');

clf
plot(y1(1:75*interp))
title('Multirate Filtered Signal');

figure(4)
clf
NFFT = length(y);
fs = fs1*M;
f=fs/2*linspace(0,1,NFFT/2);
f3 = 2*fft(y)(1:NFFT/2);
f3 = sqrt(real(f3).^2+imag(f3).^2);
plot(f,20*log10(f3/max(f3)))
xlabel('Frequency (Hz)')
ylabel('Power (dB)')
title('Control Spectrum')

figure(5)
clf
NFFT=length(y1);
fs = fs1*interp;
f=fs/2*linspace(0,1,NFFT/2);
f3 = 2*fft(y1)(1:NFFT/2);
f3 = sqrt(real(f3).^2+imag(f3).^2);
plot(f,20*log10(f3/max(f3)))
xlabel('Frequency (Hz)')
ylabel('Power (dB)')
title('Multirate Spectrum')
```

APPENDIX D. Fast Correlator Implementation

```
float Correlator_i::doCorrelation(CORBA::Float freq, int num) {

        char filename[100];

        short * I0 = cd1->getIData();
        short * Q0 = cd1->getQData();
        unsigned long length0 = cd1->getLength();
        short * I1 = cd2->getIData();
        short * Q1 = cd2->getQData();
        unsigned long length1 = cd2->getLength();

        float *maxes = new float[length0];
        float rval = 0;

        if (length0 == length1) {
                fftwf_complex *in1 = (fftwf_complex*)fftwf_malloc(sizeof(fftwf_complex)*length0);
                fftwf_complex *in2 = (fftwf_complex*)fftwf_malloc(sizeof(fftwf_complex)*length0);
                fftwf_complex *out1 =
(fftwf_complex*)fftwf_malloc(sizeof(fftwf_complex)*length0);
                fftwf_complex *out2 =
(fftwf_complex*)fftwf_malloc(sizeof(fftwf_complex)*length0);

                for (unsigned int i=0; i<length0; i++) {
                        in1[i][0] = I0[i];
                        in1[i][1] = 0;
                        in2[i][0] = I1[i];
                        in2[i][1] = 0;
                }

                fftwf_plan p1 =
fftwf_plan_dft_1d(length0,in1,out1,FFTW_FORWARD,FFTW_ESTIMATE);
                fftwf_plan p2 =
fftwf_plan_dft_1d(length0,in2,out2,FFTW_FORWARD,FFTW_ESTIMATE);
                fftwf_plan p3 =
fftwf_plan_dft_1d(length0,out1,out2,FFTW_BACKWARD,FFTW_ESTIMATE);

                fftwf_execute(p1);
                fftwf_execute(p2);

                for (unsigned int i=0; i<length0; i++) {
                        float real1 = out1[i][0];
                        float real2 = out2[i][0];
                        float imag1 = out1[i][1]; // CONJUGATE
                        float imag2 = -out2[i][1]; // CONJUGATE
                        out1[i][0] = real1 * real2 - imag1 * imag2;
                        out1[i][1] = (real1 + imag1) * (real2 + imag2) - real1 * real2 - imag1 *
imag2;
                }

                fftwf_execute(p3);
```

```
                float max = -1;
                float max2 = -1;

                unsigned int maxi = length0;
                unsigned int maxi2 = length0;

                for (unsigned int i=0; i<length0; i++) {
                        maxes[i] = sqrt(pow(out2[i][1]/length0,2) + pow(out2[i][0]/length0,2));
                        if (maxes[i] > max){
                                max = maxes[i];
                                maxi = i;
                        }
                }

                for (unsigned int i=0; i<length0; i++) {
                        if ((maxes[i] > max2) && (abs((int)i-(int)maxi) > 200000)) {
                                max2 = maxes[i];
                                maxi2 = i;
                        }
                }
                int out = ((int)maxi+(int)length0/2)%(int)length0-(int)length0/2;

                if (abs(out) < 20000)
                        tCorr++;

                rval = max-max2;

                correlation[maxi]++;

                correlation2->push_back(out);
                fftwf_destroy_plan(p1);
                fftwf_destroy_plan(p2);
                fftwf_destroy_plan(p3);
                fftwf_free(in1);
                fftwf_free(in2);
                fftwf_free(out1);
                fftwf_free(out2);

        }
        else {
                std::cout<<"BAD CORRELATION LENGTHS"<<std::endl;
        }


std::ofstream file;
        sprintf(filename,"/sdr/%d/",(int)freq,num);
mkdir(filename,S_IRWXU);

        sprintf(filename,"/sdr/%d/maxes.dat",(int)freq);
file.open(filename, std::ios::trunc | std::ios::binary);
std::cout<<"Writing "<<filename<<" of length "<<length0<<std::endl;
        file.write((char*)maxes,sizeof(float)*length0);
        file.flush();
        file.close();
        sprintf(filename,"/sdr/%d/file1I.dat",(int)freq);
```

```
    file.open(filename, std::ios::trunc | std::ios::binary);
    std::cout<<"Writing "<<filename<<" of length "<<length0<<std::endl;
            file.write((char*)I0,sizeof(short)*length0);
            file.flush();
            file.close();
            sprintf(filename,"/sdr/%d/file1Q.dat",(int)freq);
    file.open(filename, std::ios::trunc | std::ios::binary);
    std::cout<<"Writing "<<filename<<" of length "<<length0<<std::endl;
            file.write((char*)Q0,sizeof(short)*length0);
            file.flush();
            file.close();
            sprintf(filename,"/sdr/%d/file2I.dat",(int)freq);
    file.open(filename, std::ios::trunc | std::ios::binary);
    std::cout<<"Writing "<<filename<<" of length "<<length0<<std::endl;
            file.write((char*)I1,sizeof(short)*length1);
            file.flush();
            file.close();
            sprintf(filename,"/sdr/%d/file2Q.dat",(int)freq);
    file.open(filename, std::ios::trunc | std::ios::binary);
    std::cout<<"Writing "<<filename<<" of length "<<length0<<std::endl;
            file.write((char*)Q1,sizeof(short)*length1);
            file.flush();
            file.close();

            delete [] maxes;

            return rval;
}
```

APPENDIX E. USRP_TIME Process Implementation

```
void USRP_TIME_i::rx_data_process() {
        uint32_t *rx_buffer = NULL;

        unsigned int mux = usrp_rx->determine_rx_mux_value(0);
        if (number_of_rx_channels > 1)
                mux = usrp_rx->determine_rx_mux_value(0, 1);
        usrp_rx->set_mux(mux);

        unsigned int muxity = usrp_rx->mux();

        int nchan = usrp_rx->nchannels();
        std::cout << "nchan: " << nchan << "number of chan"
                        << number_of_rx_channels << std::endl;

        int decim = usrp_rx->decim_rate();
        std::cout << "decim: " << decim << std::endl;

        DEBUG(3, USRP_TIME, "rx_packet_size " << rx_packet_size)
        unsigned int buf_size = rx_packet_size * number_of_rx_channels
                        * (complex ? 2 : 1); // number of shorts to read

        DEBUG(3, USRP_TIME, "buf_size: " << buf_size) // 16384

        PortTypes::ShortSequence I0;
        PortTypes::ShortSequence Q0;

        usrp_rx->start();

        omni_thread *collection_thread = new omni_thread(Collect, (void *) this);

        collection_thread->start();

        unsigned long long time_per_1sec = 0;
        uint32_t lastPktTimestamp;

        while (rx_active && ((rx_packet_count == -1) || (rx_packet_count > 0))) {
                usrp_packet_t rx_data[10];

                unsigned int readLen = ReadData(rx_data, 10);
                I0.length(readLen * 126);
                Q0.length(readLen * 126);

                uint32_t ts;
                int ii = 0;

                for (int i = 0; i < readLen; i++) {
                        usrp_packet_t packet = rx_data[i];
                        if (i == 0) {
                                ts = packet.timestamp;
                        }
                        if (packet.timestamp < lastPktTimestamp) {
```

```
                                        time_per_1sec = lastPktTimestamp + (126 * decim)
                                                        - packet.timestamp;
                                        DEBUG(3,USRP_TIME, "One second has passed, clocks per sec: " <<
time_per_1sec<< " Buffer size: "<<bufLen)
                                }
                                lastPktTimestamp = packet.timestamp;
                                if (USRP1_HEADER_BURST_CHAN(packet.header) == 0) {
                                        for (int j = 0; j < 126; j++) {
                                                int j_1 = j * 2;
                                                int j_2 = j * 2 + 1;
                                                I0[ii] = (CORBA::Short) packet.data[j_1];
                                                Q0[ii] = (CORBA::Short) packet.data[j_2];
                                                ii++;
                                        }
                                }
                        }
                        PortTypes::MetaData md;
                        time_t tm = time(NULL);
                        md.timestamp = tm;
                        md.samples = ts;
                        md.sampling_frequency = time_per_1sec;
                        rx_data_1_port->pushPacket(I0, Q0, md);

                        if (rx_packet_count != -1)
                                --rx_packet_count;

                }

                DEBUG(3, USRP_TIME, "Exiting rx_data_process thread")

                usrp_rx->stop();
                rx_active = false;
                rx_thread->exit();
}
```