

MICHIGAN STATE
U N I V E R S I T Y

Geolocation of RF Emitters

Design Team 2

ECE 480

Sponsor: **DEPARTMENT OF ECE**
ELECTRICAL AND COMPUTER ENGINEERING

Facilitator: Dr. Jian Ren

George Godby
Justin Mascotto
Matthew Roach
Viktor Simovski
Kenneth Wilkins

Executive Summary

The ability to accurately detect the location of radio signals provides a wide variety of uses ranging from determining the location of a distress signal from a party or individual in need of help to locating an interfering signal that is jamming communications. This project focuses on software defined radio technology to receive and process an RF signal to ultimately calculate position and angle of arrival of emitting radio signal. Using this method to sample from multiple locations we will be able to paint a picture of the location of the radio signal. The main goal of the Geolocation of RF Emitters project was to create an RF sensor which could be remotely deployed and report acquired RF data back wirelessly to a central hub for further analysis to ultimately find position of the RF signal. Information passed back to the central hub includes: Radio Signal Strength (RSS), Angle of Arrival of signal (AOA), Geo-Location of sensor (via GPS), and Phase. The entire system is designed to be scalable such that by increasing the number sensors which are deployed, the more accurate the position location of the RF emitter will be. With a small, lightweight interface unit, known as a Universal Software Radio Peripheral (USRP), coupled with GPS data and a computer, our design has the ability to dynamically receive a wide range of radio signals and pinpoint their location.

Acknowledgments

David Hamill

Thanks to David Hamill, Sales Engineer at Ettus Research, for assisting in ordering the correct hardware and helping in replacing faulty hardware.

Dr. Jian Ren

A special thanks the Dr. Jian Ren, Design Team 2 Facilitator, for keeping us on track and providing insight into both the team's project as well as their careers.

Dr. Lalita Udpa

A special thanks to Dr. Lalita Udpa for being available to discuss design deliverables and answer any question the team might have.

Dr. Timothy Grotjohn

A special thanks to Dr. Timothy Grotjohn for sponsoring the project when the original sponsor fell through. Also an extra thanks for expanding our budget to allow the team to remain on the project given to us.

Electrical and Computer Engineering Department

A special thanks to the ECE department for sponsoring the project and providing adequate funds to accomplish the project.

Roxanne Peacock

A special thanks to Roxanne Peacock for assisting us in ordering all necessary hardware for the design. Roxanne was also instrumental in helping to sort out faulty hardware on two separate occasions.

Table of Contents

Executive Summary	1
Acknowledgments	2
1 Introduction and Background	5
2 Exploring the Solution Space and Selecting a Specific Approach	7
Critical Customer Requirements	8
Chosen Design Solution	8
Budget	10
GANTT Chart	12
3 Technical Description of Work Performed	13
USRP	13
GNU Radio	14
Phase – Motivation	15
Phase – Approach	16
Phase – Method	17
BeagleBone Black	27
Wireless Networking	28
GPS	29
User Interface	30
Design Issues	31
Project Lifecycle	31
Design	32
Testing	32
Standards	33
Reliability	34
Reparability	34
Retirement	34
4 Test Data with Proof of Functional Design	36
USRP	36
GNU Radio Companion Software	37
Python Programs	38
GPS	39
Project	41
5 Final Cost, Schedule, Summary and Conclusions	44
Successes	44
Failures	44
Suggestions for Future Work	44
Final Cost	45
Additional Projected Costs	45
Multiple Units Cost	46
Schedule	46
Conclusion	49

Appendix 1 Technical Roles, Responsibilities, and Work Accomplished.....	50
George Godby.....	50
Matthew Roach	51
Viktor Simovski.....	52
Justin Mascotto	53
Kenneth Wilkins.....	54
Appendix 2 Literature and Website References	55
Websites	55
Literature	56

1 | Introduction and Background

The main goal of the Geolocation of RF Emitters project was to create an RF sensor which could be remotely deployed and report acquired RF data back wirelessly to a central hub for further analysis. Information passed back to the central hub includes: Radio Signal Strength (RSS), Angle of Arrival of signal (AOA) and Geolocation of sensor (via GPS). The entire system is designed to be scalable such that by increasing the number sensors deployed, the more accurate the position location of the RF emitter will be. An additional key attribute of the system is that the sensor does not rely on any external cables to support it, such as, power cables or communications cabling. The system is fully capable of operating from batteries and communicates via Wi-Fi protocol to the host PC.

The purpose here was to develop a test-bed for acquiring above listed information. This RF 'front end' would allow researchers and software engineers to develop algorithms using the acquired raw data packages. The team developed a few data handler programs and a user interface to serve as an example for what could be further developed, but the underlying goal has always been to build an RF acquisition system capable of providing a raw data structure to the user for testing of RF algorithms. For the purposes of this document, raw data refers to the common 'binary formatted' data that is generally used in high end data processing software like MATLAB.

Motivation for this type RF capture system traces its roots back to academic research and military deployment. With RF data such as phase and signal strength coupled with GPS information, scientist, mathematicians and engineers are always creating and improving new algorithms to analyze raw RF data to make predictions about the location of the RF source. In addition, this raw data can be used to create RF maps, which help to paint an RF picture of the RF signals in a geographic area, such as the one seen in figure 1.

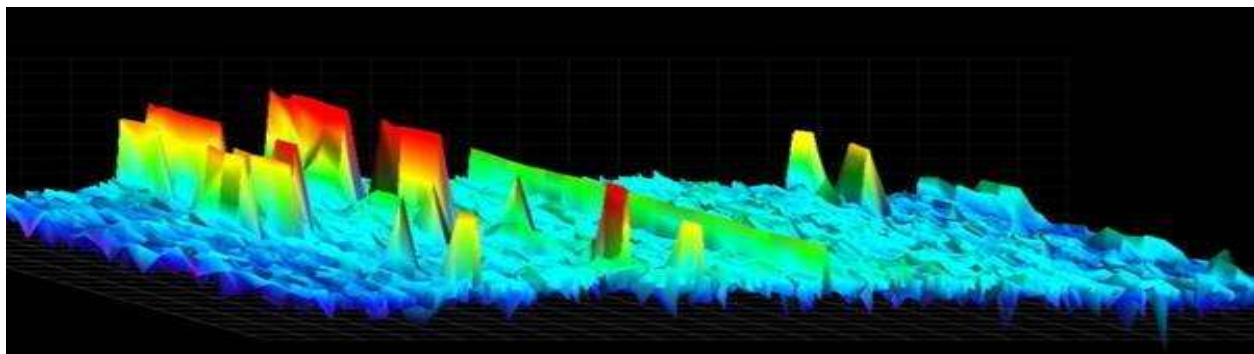


Figure 1 - Visual Graphic of Signal Properties

Other uses include development for cognitive radio applications. Cognitive radio research address the fact that the RF spectrum is very congested and having a 'smart transmitter' which would constantly sense its RF environment and dynamically change its transmitting frequency

and even direction and power based on real time information would be very useful in making the most efficient use out of an RF space at any given time. Military uses include mounting of these sensors on drones for locating enemy transmissions or locating lost equipment with an emitter beacon.

Although technically complicated, physically the components layout rather simply, as shown in figure 2.

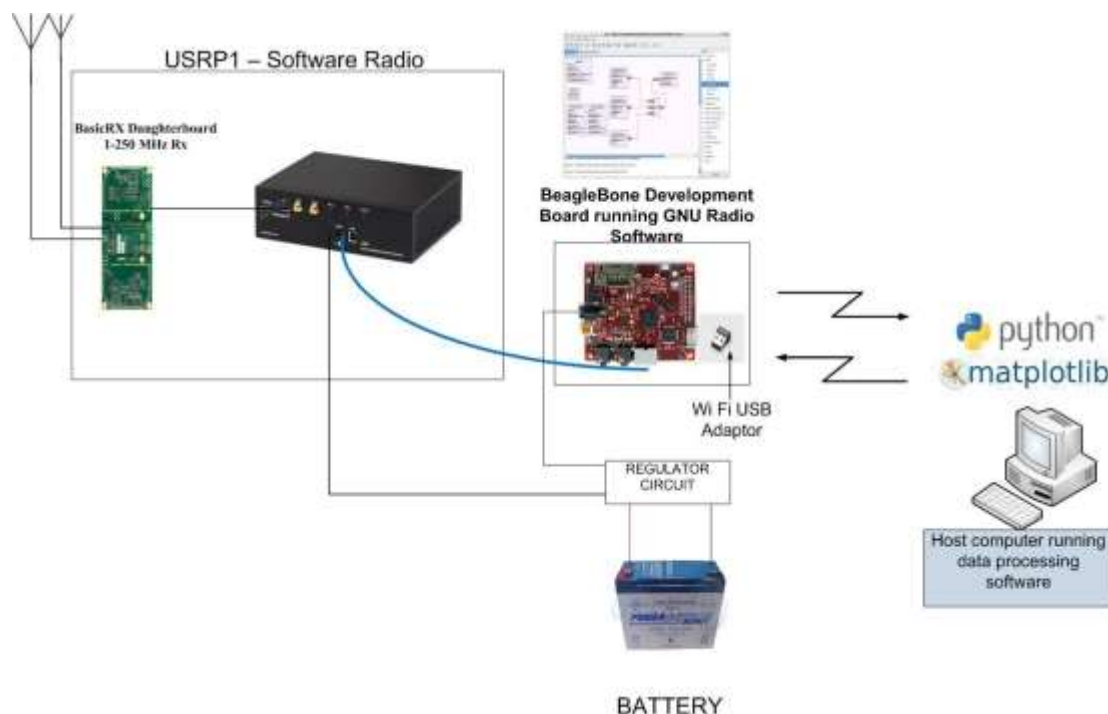


Figure 2 - Project component layout

Basically, the fundamental project breaks down into three parts, which are explained in far greater detail in the technical section of this report.

- Universal Software Radio Peripheral (USRP) – Interface of RF signal to signal processing program.
- Development Board – Microcontroller running Linux, which initially acts upon the USRP and handles wireless communications to the host PC.
- Host computer running raw data analyzer programs and User Interface (UI).

This simple construction allows for the USRP and development board to be packaged together and powered from a single battery in housing with a footprint no larger than 10x10x6. With the addition of wireless communication, the sensor unit is completely portable and capable of being placed in the field while data analysis is performed remotely.

2 | Exploring the Solution Space and Selecting a Specific Approach

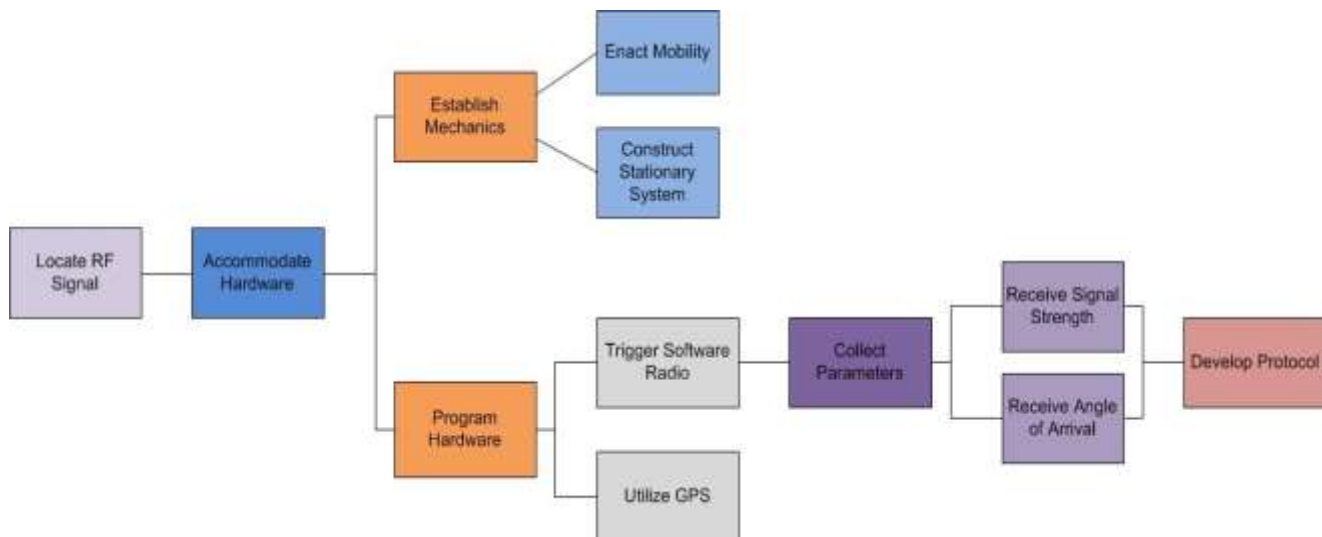


Figure 3 - FAST Diagram

As aforementioned, the main goal of the Geolocation of RF Emitters project is to create a remotely deployable RF sensor that can locate an RF signal. In order for this to be done, hardware must be acquired. The hardware must then be accommodated for in order for it to accomplish what is the project intends. The types of hardware needed were separated into two task groups, establishing mechanics and programming the hardware, which branch off into their own purpose to reach the final project goal.

To accommodate the hardware to serve the purpose intended by the project, establishing mechanical components is needed for the mobility expectations. The mobility requirement, however, leaves engineers open to designing the prototype to either enact mobility or to be constructed as a stationary system. Upon having the team's funds cut, it was soon inevitably so that the geolocator will be stationary due to that lack of funding. This is because implementing mobility to the project would have required more money the team simply did not have. However, more funding would have enabled the purchasing of remote controlled RC cars that would be strapped to the geolocator.

On the other end of the accommodating hardware task lies the issue of programming the hardware. One of the tools the team planned to use to geolocate an RF emitter is a GPS module. That GPS module will output data that will be used to locate where you, the geolocation, and the emitter are on a map. Therefore, a suitable microprocessor will be used to gather the data and implement it. That processor will also be used to trigger the software radio

to do a frequency sweep and collect certain wave characteristics. These said characteristics involve the reception of RSS of an RF signal as well as the angle of arrival. Finally this data must be in unison with the microprocessor to develop a protocol that will accurately help point out an RF emitter on a map.

Critical Customer Requirements

During the initial design phase, the team identified key characteristics that make up the core of the project that must be satisfied in order to ensure the team is making progress towards a successful design that will accomplish the sponsor's deliverables and goals:

- **Angle of Arrival of signal** – Staying with the theme of keeping the project as simple as possible, which would aid in its portability, we determined the best way to discover signal direction is by using two antennas and calculating the phase difference between the two antennas.
- **Signal Reception** - The best choice is to use a radio that can be controlled by software. A relatively new technology known as Software Defined Radios (SDR) now exists. This technology gives the user a wide range of options and tools for receiving radio signals.
- **Portability** – Since the sensors could not be directly connected to a host PC it was determined that a Software Defined Radio (SDR) outfitted in the sensor could be controlled by a development board running Linux. Additionally, the development board handles the wireless communications and GPS information back to the host PC.
- **Location of sensor** – In order to figure the location of a radio emitter, first you need to know the location of the receiver; the location information is provided through a Global Positioning System (GPS) unit.
- **Data handling** – Not be overlooked, was what to do with the raw data once it is acquired from the sensor system. The obvious choice here was to write data handler scripts in Python language. Python is a great fit here since it plays well with Linux operating and comes with very effective mathematical computational add-on libraries.

Chosen Design Solution

The team created a feasibility matrix to help identify the best antenna design for the geolocator. The feasibility matrix compares multiple designs to different design parameters such as functionality, cost, and time. Each design was ranked from 1-10 for each parameter then totaled for a complete score out of 30. See figure 4 below.

	<u>Stationary perpendicular dipole antennas</u>	<u>Single rotating dipole antenna</u>	<u>Two stationary monopole antennas</u>
<u>Functionality</u>	Least accurate in detecting AOA. Easiest to implement into design with respect to designing the antenna and the programming required to determine AOA. Feasibility (4/10)	Greatly increases the complexity of our design. Our antenna criteria for this design to work requires an antenna not on the market, therefore it would need to be fabricated. Should give a very accurate AOA. Feasibility (5/10)	Given USRP1 can handle 2 antennas simultaneously receiving data, this design should be relatively easy to implement in regards to antenna design and programming required to determine AOA. Feasibility (8/10)
<u>Cost</u>	Two dipole antennas \$20 Feasibility (10/10)	Single rotating dipole antenna \$50 Feasibility (8/10)	Two monopole antenna \$20 Feasibility (10/10)
<u>Time</u>	2 days Feasibility (10/10)	9 days+ Feasibility (5/10)	5 days Feasibility (9/10)
<u>Total Feasibility</u>	(8/10)	(6/10)	(9/10)

Figure 4 - Feasibility Matrix

As you could see from the feasibility matrix above, the two stationary monopole antenna designs outweighed the two other designs that were considered. Given that the USRP1 that the team plans to use can handle two antennas at once makes the functionality of that design highly desirable and basically made the final decision for the design to be used. To help in making the decision, the team also filled out a selection matrix.

Selection Matrix	Importance	Stationary perpendicular dipole antennas	Single rotating dipole antenna	Two stationary monopole antennas
Cost	4	9	3	9
AOA Accuracy	4	1	3	9
Mobility	5	9	3	9
Scalable	5	9	3	9
User Friendly	2	9	3	9
Durability	3	9	1	9
Total		130	63	162

Figure 5 - Selection Matrix

As you could see from figure 5 above, the two stationary monopole antenna design heavily outweighed the other two by at least 32 points for the selection matrix. Being among the cheapest designs as well as the easiest to accommodate, the choice was made and the team moved on.

Budget

Without digressing too much, a few notes on project progression warrant inclusion here. Initially the project was to be sponsored by the US Air Force and was to include an entire network of high quality, large bandwidth sensors and special funding, far exceeding the baseline class budget of \$600, was allocated. However, due to circumstances beyond the team's control, this fell through. Regrettably, this put the design almost three weeks behind schedule. Fortunately, Dr. Grotjohn and the MSU Department of Engineering agreed to take on the team and sponsor the geolocator project, however, the project cost, which initially totaled to nearly \$15,000, needed to be drastically reduced. The team rallied and researched less expensive equipment and came up with a scaled down, yet fully functional project for less than \$1300. By choosing an older model of software radio and lowering the received bandwidth to 1-250 MHz and reducing the sensors to a single sensor, the team was able to build the project that is demonstrated in this report.

PRODUCT	LINK	QUAN	PRICE	TOTAL
USRP N200 Kit (USRP N200)	https://www.ettus.com/product/details/UN200-KIT	2	\$1,500.00	\$3,000.00
WBX-40 USRP Daughterboard (50 MHz - 2.2 GHz, 40 MHz BW)	https://www.ettus.com/product/details/WBX120	2	\$475.00	\$950.00
MIMO Data and Sync Cable	https://www.ettus.com/product/details/MIMO-CBL	1	\$70.00	\$70.00
USRP GPS-Disciplined Oscillator Kit	https://www.ettus.com/product/details/GPSDO-MINI	1	\$750.00	\$750.00
VERT400 Antenna-(144 MHz, 400 MHz, 1200 MHz) Triband	https://www.ettus.com/product/details/VERT400	2	\$45.00	\$90.00
GPS ANTENNA	https://www.ettus.com/product/details/GPS-ANT-5V	1	\$51.00	\$51.00
BeagleBone Board	http://beagleboard.org/Products/BeagleBone	1	\$88.95	\$88.95
Wi-Fi Adaptor	http://www.adafruit.com/products/814	1	\$11.95	\$11.95
6V Battery 12 Ah. 4.3 lb	http://www.atbatt.com/	1	\$16.19	\$16.19
Misc.		1	\$100.00	\$100.00
				\$5,128.09

Figure 6 - Initial Air Force Proposal Price

After the initial budget of \$10,000, the new budget was cut to \$5,000. Shown above is the budget table the team constructed with that budget. As you can see, it was still over the available \$5K budget aforementioned. To accommodate, the team manager made calls to the OAI to seek a slightly high budget due to the short notice of budget cuts. That proposal was then resent and accepted. Unfortunately, this was when the Air Force dropped the team 2 project overall.

The team was now left looking for sponsors. Once the ECE department accepted the team's project, this was when the budget was cut to the final price of \$1,300. Shown below is the final proposal budget list made for the ECE department on the initial proposal. As you can see, the project scope was slightly reduced but the cost was cut by over 75%.

<u>Components:</u>	<u>Price:</u>
USRP1	\$700.00
BasicRX Daughterboard	\$75.00
BeagleBone Board	\$88.95
Wi-Fi Adaptor	\$11.95
6V Battery	\$16.19
Antenna(s)	\$170.00
GPS	\$100.00
TOTAL: \$1,192.09	

GANTT Chart

As seen below in Figure 7 is the GANTT chart for the design team. Overall the team was successful in staying on course, ignoring the unforeseen delays caused by the USRP and initial sponsor.

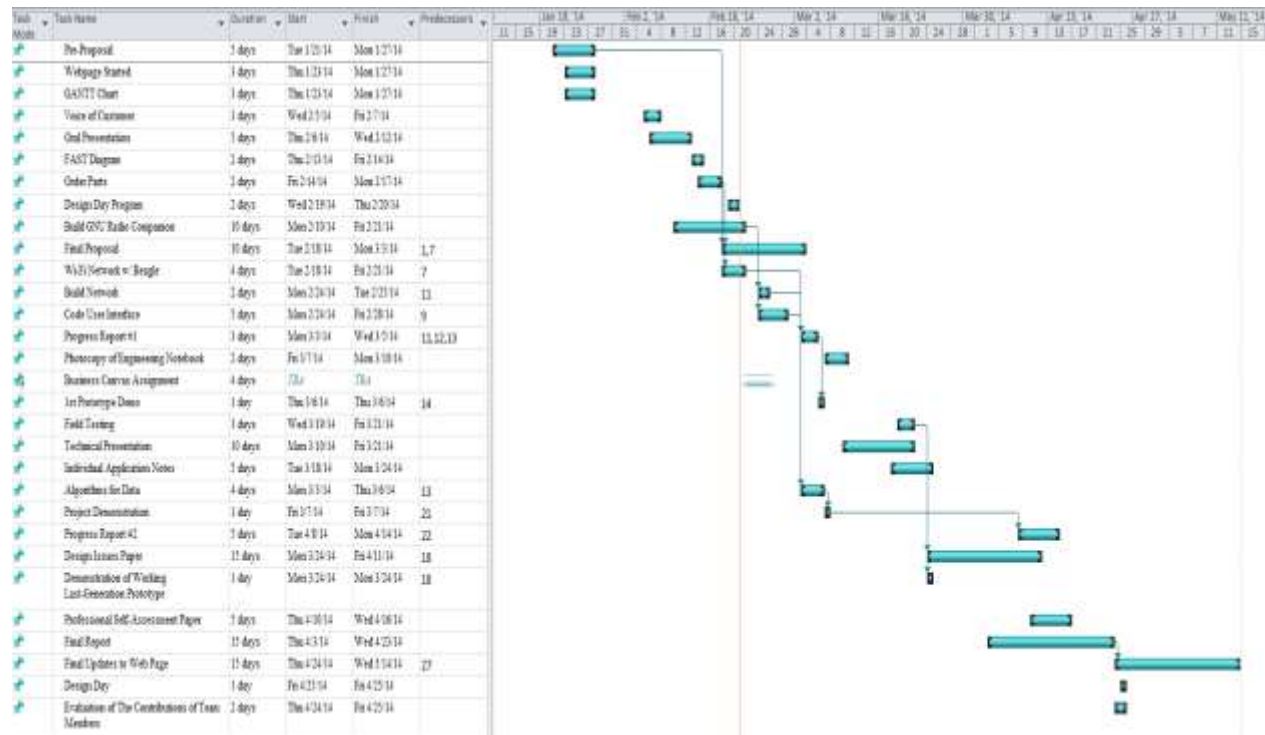


Figure 7 - GANTT Chart

3 | Technical Description of Work Performed

The Geolocator of RF Emitters project had many technical aspects, each deserving of its own explanation. The venture steered the team down a few roads that no one on the team had traveled before, so the project set a pretty steep learning curve for some of the components, such as, how to use GNU Radio companion software to control a Software Radio and how to install a Linux operating system on a development board. This chapter will explore each of the fundamental components of the project and explain in technical detail how they work.

USRP

The Universal Software Radio Peripheral (USRP), otherwise known as a Software-defined radio (SDR) and shown in figure 8, is a different approach to radio communications that implements components and functions through software rather than in hardware, as with most other radio communications devices.



Figure 8 - Universal Software Radio Peripheral (USRP)

The major advantage to using an SDR is the capability to transmit/receive a wide range of radio frequencies, which can be changed dynamically through software. The USRP unit is relatively simple, yet elegant in design. Figure 9 shows a block diagram of the components.

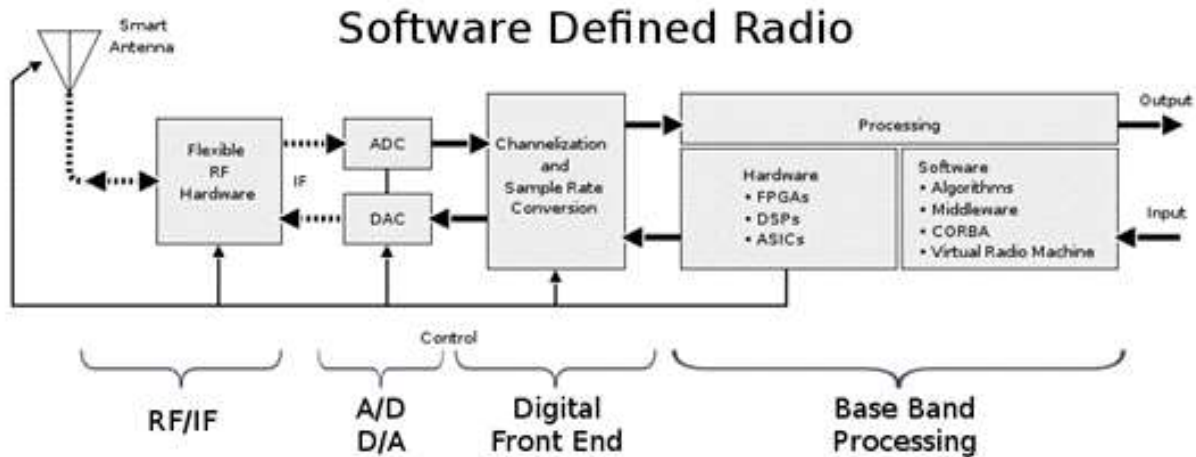


Figure 9 - Universal Software Radio Peripheral (USRP) block diagram

Walking through the diagram from left to side, the team first encountered the RF daughter board section. In this section the user installs a hardware RF board that will configure the system bandwidth to the specific needs and budget of the project at hand. The USRP is capable of both transmitting and receiving RF signals; these parameters are selectable through which daughter boards are installed.

Next, as shown in figure 10, the Analog to Digital Converter (ADC) converts the received signal to a digital signal and thus allowing it to be processed in the Field Programmable Gate Array (FPGA) section where it is converted to raw data and pushed out via the USB interface controller.

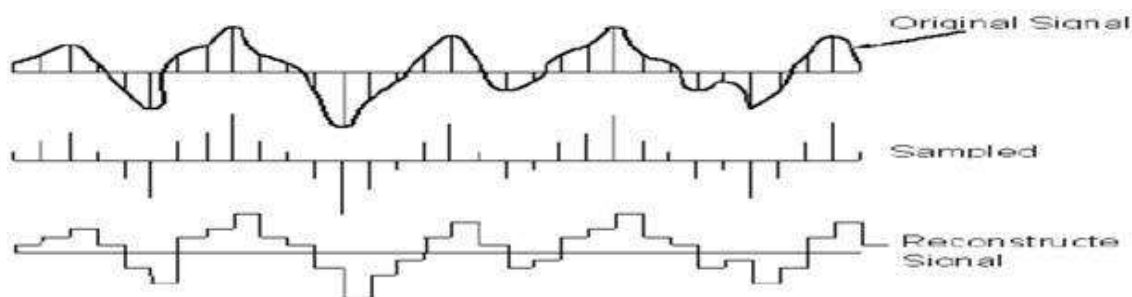


Figure 10 - Analog to digital conversion process

GNU Radio

Once the raw binary data has left the USRP it will need a capture program to handle this data and do something with it. Team 2 chose to use GNU Radio Companion for the device. GNU Radio is free and open sourced and offers an exhaustive list of operations to which one may perform on the data at hand. GNU Radio Companion has lots of pre-built open source code libraries written in 'C' language that are available to the programmer for use as they are written

or they may be modified. The user can display waveforms in both the time or frequency domains, apply many types of filters to the signal, perform a Fast Fourier Transform (FFT) in the signal, and split the signal into real and imaginary components. The software is also capable of handling multiple input signal and the signals may be added or subtracted. Figure 11 shows a very simple circuit in action, the top displays the flow graph circuit and the bottom shows the output from the circuit that was built.

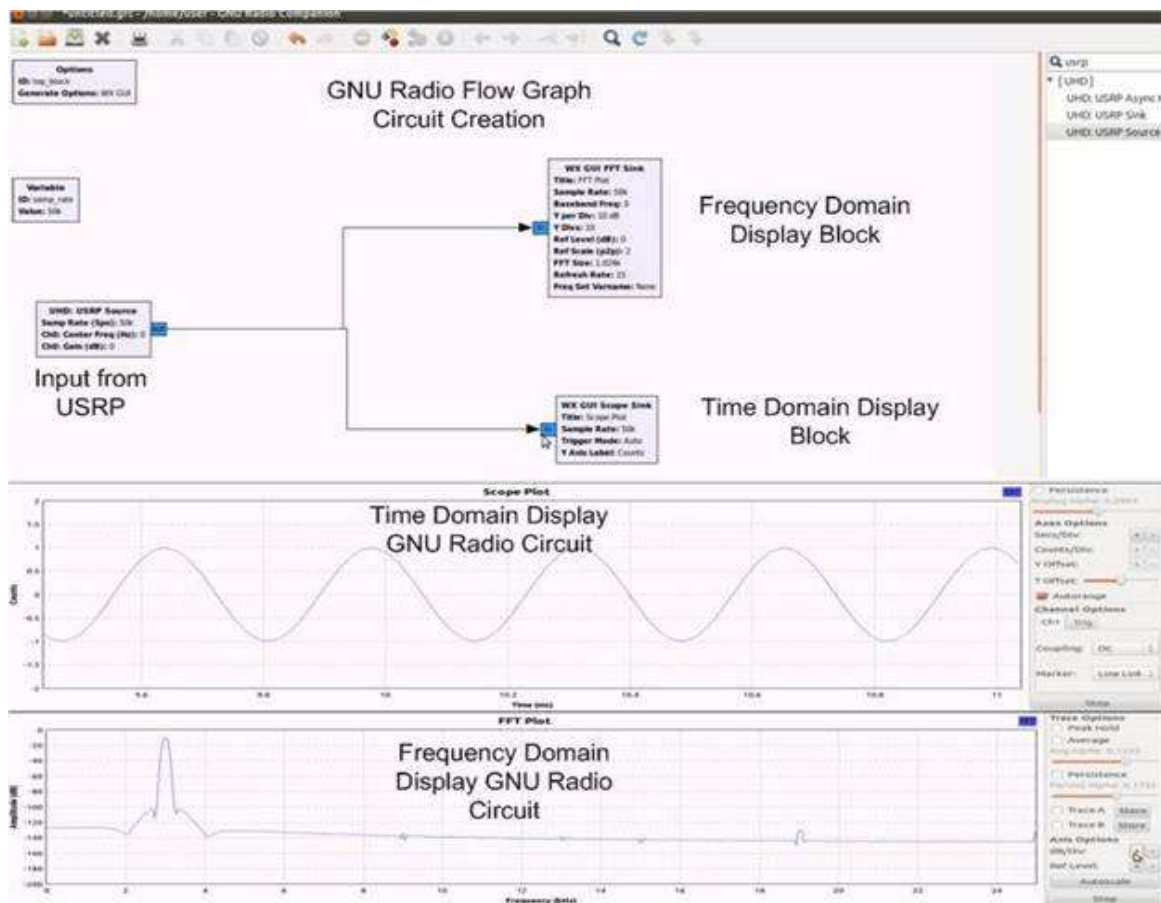


Figure 11 - GNU Radio Companion

A major accomplishment of this project was designing a circuit to handle signal phase. There are many aspects to how the phase works into this project and in order to best explain the procedure, it shall be broken into sub sections.

Phase - Motivation

As explained in the introduction, the geolocator chooses to capture phase in order to gather information on signal direction, this is referred to in this writing as Angle of Arrival (AOA).

Figure 12 demonstrates two antennas spaced sufficiently far apart for the wavelength being measured; here, each antenna will receive a different phase of the signal incident upon it. The team can exploit this fact by using a Software Defined Radio (SDR) that is capable of receiving two separate signals independently and by building a software circuit (flow graph), to extract this phase information from each antenna separately.

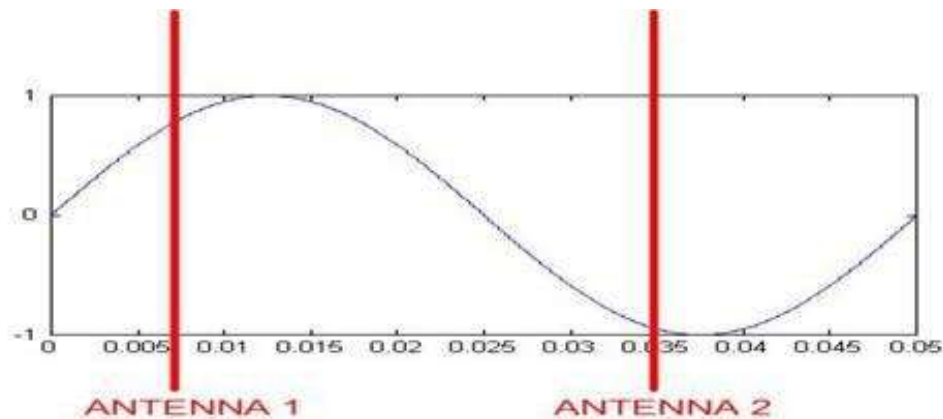


Figure 12 - Signal phase on two antennas

Phase - Approach

In the simplest of terms, the measurement of phase is trivial. Figure 13 shows two signals displayed in the time domain and in order to calculate phase, all one needs to do is calculate the time difference between the two.

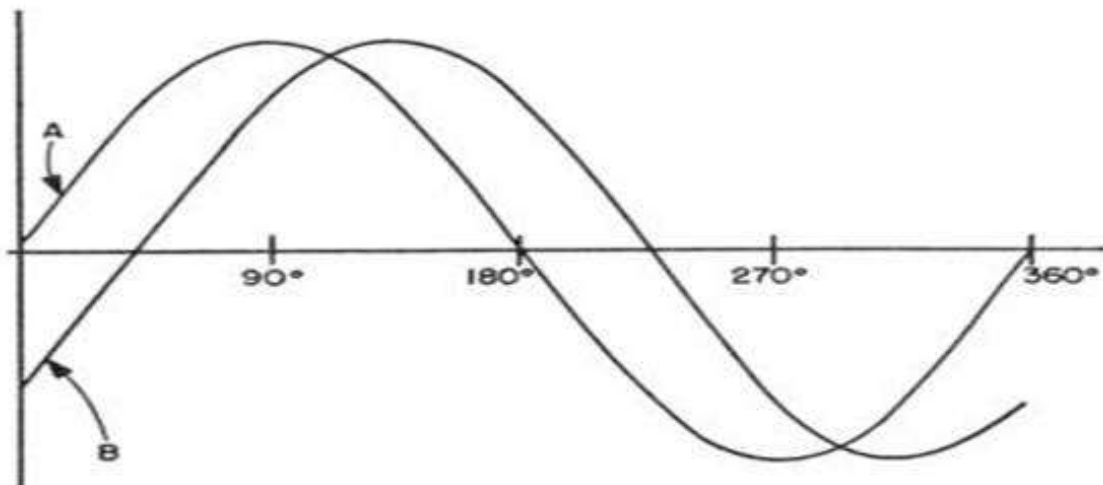


Figure 13 - Signal Phase

However, this method does not prove to work for the project at hand and there are two distinct reasons for this:

1. **Operating Frequency** – The frequency range in which is being dealt with in this project is in the 200 MHz range. In order to display a waveform in the time domain, the team must, at a minimum, sample the signal at a rate twice that of the original signal, this is known as Nyquist-Shannon sampling theorem. So if the signal is 200 MHz, the team would need to sample it at least 400×10^6 times a second, this far exceeds the maximum sample rate of 10×10^6 of the USRP. Therefore, the analysis is done in the frequency domain. This process will be elaborated on later in chapter 3.
2. **Acquisition of Data** – The other problem is transforming what you 'see' and understand as a phase offset as it is displayed on an oscilloscope screen in the time domain to something that can be passed along in a digital sense for further analysis. So even if the geolocator could display this phase offset between the two antennas at our given frequency of 200 MHz, there is still no good way to capture this information as it unfolds in real time such that the data could be later placed into an algorithm and this is one of our basic objectives.

Phase - Method

The goal here is, by using frequency domain evaluation techniques, to capture a specific frequency within a spectrum of many frequencies and perform an analysis on this one specific frequency of interest. Figure 14 shows a spectrum capture from the GNU Radio software in the frequency band range of the emitter signal. It can be observed that the device has a bandwidth of 5 MHz of frequencies to sort through to get the one specific target frequency of 222.5 MHz. As a side note, the given bandwidth of frequencies is set by one half of the sample rate of the system, which is 10 MHz in this case, so here you can see 5 MHz of bandwidth.

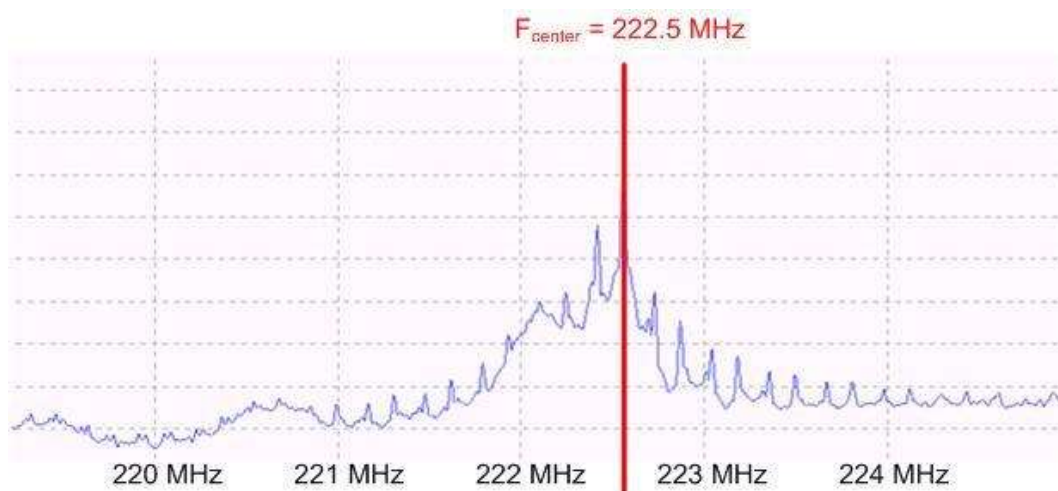


Figure 14 - Spectrum of interest

The method to extract one specific frequency from many, from a signal processing perspective, is shown in the block diagram, figure 15.

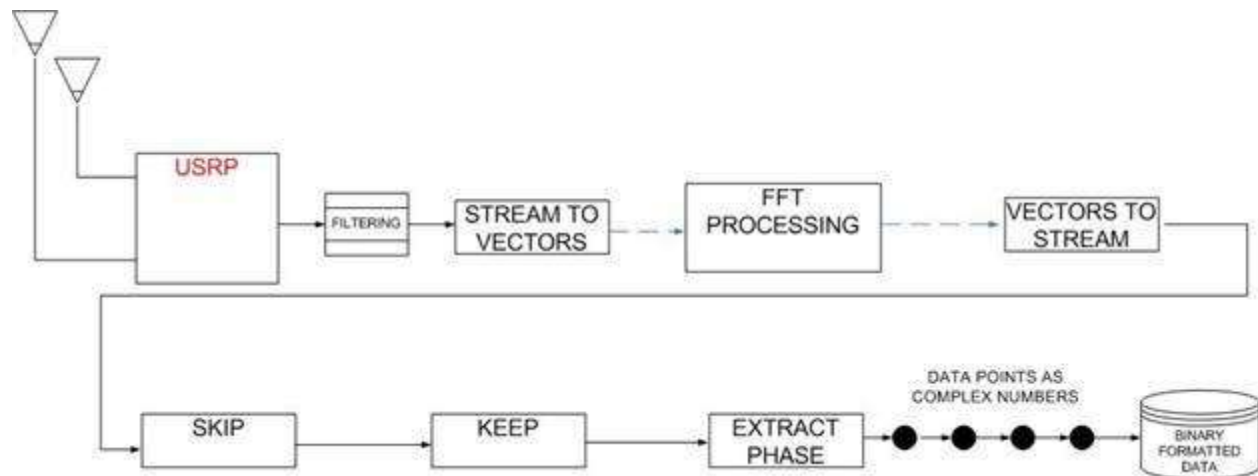


Figure 15 - FFT signal processing

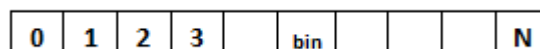
Let's take a closer look at the components individually.

Filter

This is pretty standard for any RF circuit, as it helps to keep noise from getting into the circuit

Stream to Vectors

In order to prepare for the conversion from Time Domain to Frequency Domain, the team needs to break the waveform down into an array of Time Domain samples like so:



Where N must be a power of 2, such as, 64, 128, 256.... The squares are referred to as 'bins'.

FFT

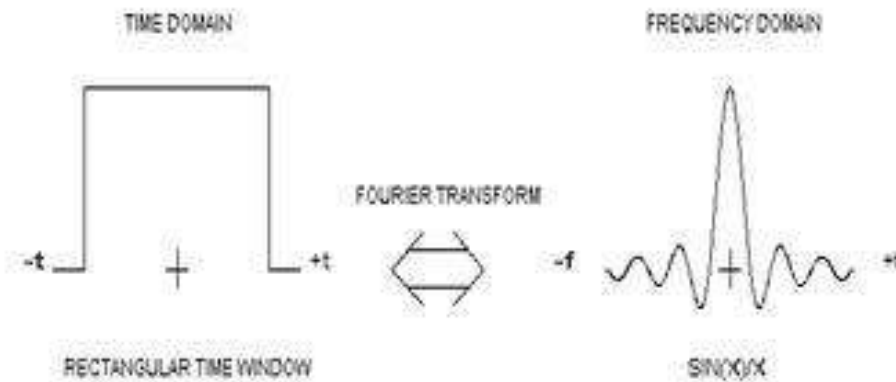


Figure 16 - Visual depiction of Fourier Transform

Once the signal is in vector format, it is run thru the Fast Fourier Transform (FFT). This is the signal processing block which converts the signal from the Time Domain to the Frequency Domain, by using algorithms beyond the scope of this writing. What is important here is that the FFT output will give an array of data which will later index in order to get only the specific frequency of interest and discard the others.

Vector to Streams

This block just strings everything back together again so the data is in one continuous flow.

Skip

Figure 17 depicts where in the signal process you stand. The squares represent the bins, keep in mind that everything here has been magnified for the purposes of illustration. Typically the numbers bins would be on the order of 1024, not 16.

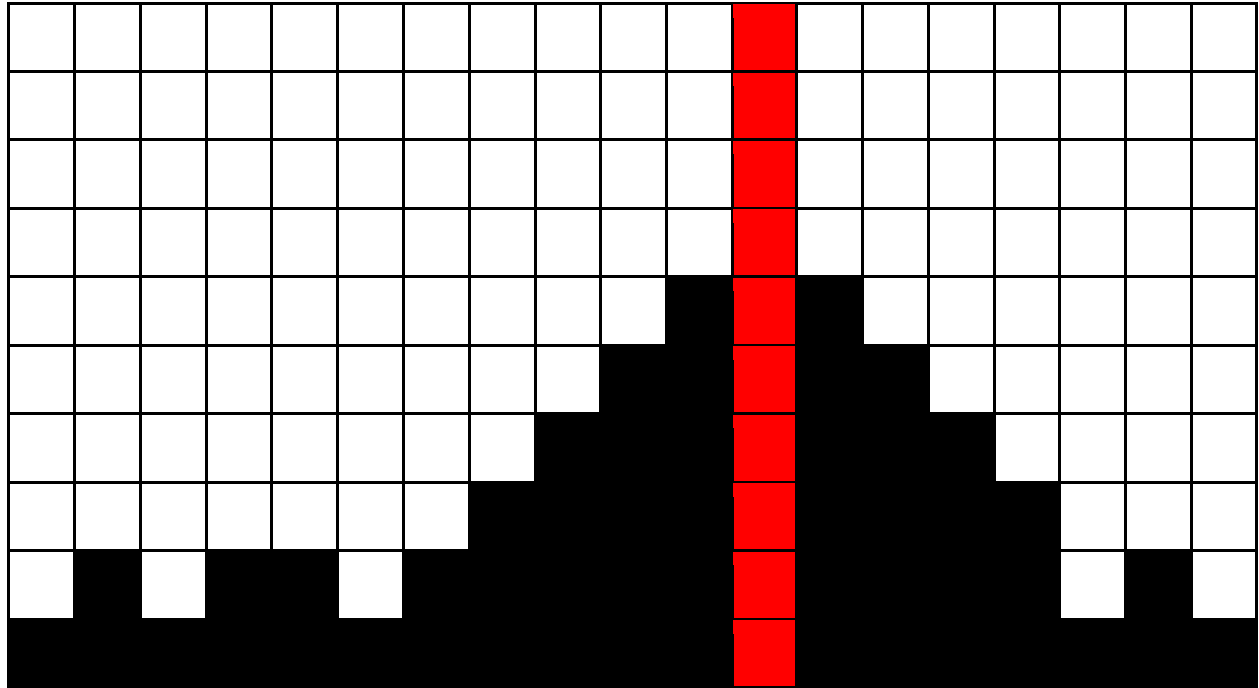


Figure 17 - FFT output showing bins

The goal here is to extract only the bin with the frequency desired or ‘skip’ to that bin. In order to calculate this, first let’s first consider the size of each bin.

$$\text{Bin size: } \frac{\text{Sample Rate}}{\text{FFT Size}}$$

With regards to FFT size, let’s give this some thought (remember it must be a power of 2); if FFT is small then the calculation will have big bins and if FFT is large then the calculation will have smaller bins, which means more resolution. On a practical note, theoretically, you can pick a very high number for the FFT size and have excellent signal resolution, however, if you are planning to actually implement this into receiving a real signal, keep in mind that real RF is somewhat of a moving target. If your bin sizes are too small you will never be able to hit the right bin, do to signal drift. So for practical purposes let’s make FFT size = 128.

$$\text{BIN SIZE: } \frac{5 \times 10^6}{128} = 39.062 \times 10^3$$

Now to find the bin of interest, simply divide the frequency that is desired by the bin size. Recall that even though the target frequency is 222.5 MHz, it is between a bandwidth of 200 MHz to 225 MHz. So really you are looking at 5 MHz in total and 222.5 MHz is exactly between 200 MHz and 225 MHz, thus you should use 2.5 MHz in the equation.

$$\frac{2.5 \text{ MHz}}{39.062 \text{ KHz}} = 64$$

Since you are using a zero based system here, you need to add 1. The actual bin of interest is bin 65. Figure 18 shows the 128 bins laid out over the spectrum

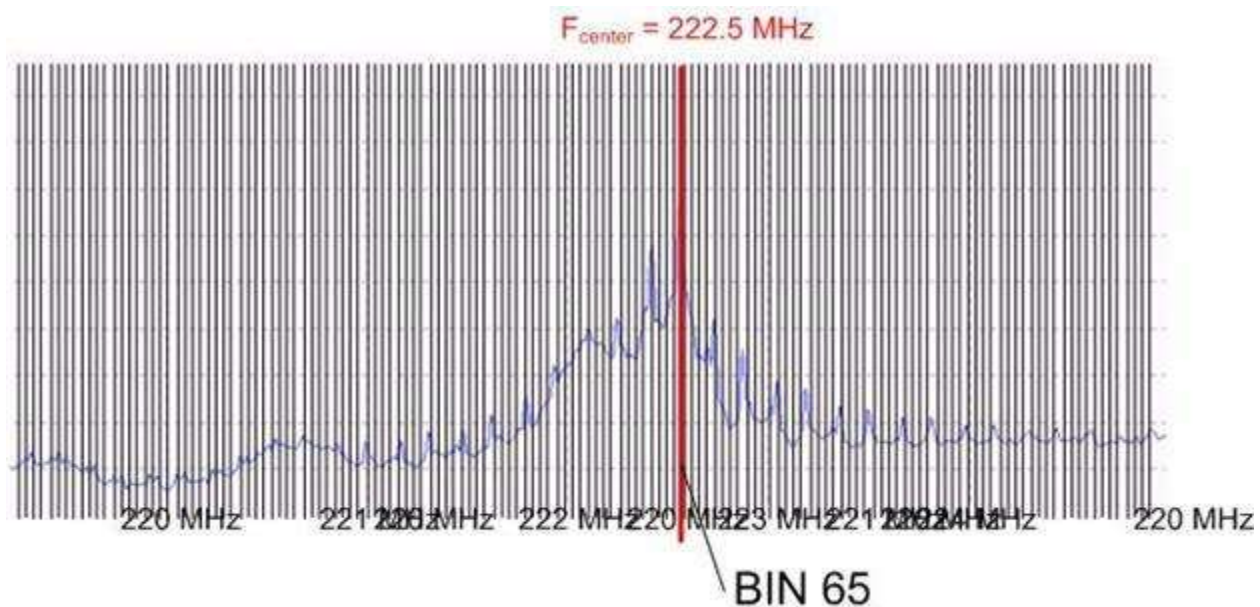


Figure 18 - 128 bins laid over FFT

Keep

This part of the software is called 'Keep 1 in N' and that is just what it does, where N is the bin of interest, bin 65. It passes only bin 65 along to the phase extraction circuit.

Phase

The phase block outputs the raw phase data points in binary format in the form of a complex number, which is good since this can be used in the post processing script to compute both phase and magnitude.

Implementation

Now that the team has demonstrated that it is theoretically possible to extract a specific frequency from a spectrum of many frequencies in the frequency domain, let's look at how to implement this in software. Here the true power of GNU Radio Companion software begins to shine. Figure 19 shows the basic idea of how the software circuit needs to lay out, the actual circuit, shown in figure 20a, is much more complicated.

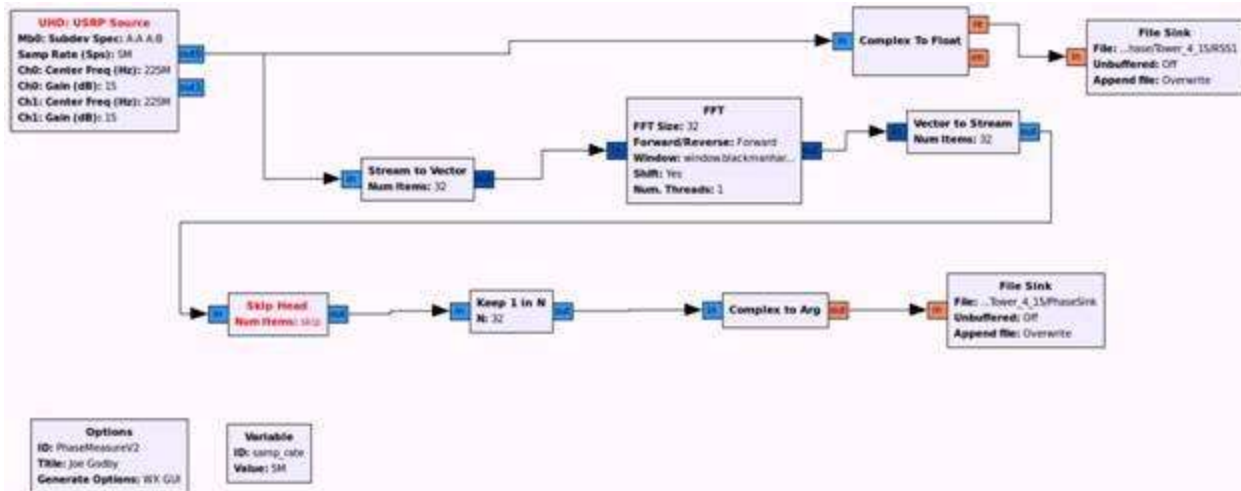


Figure 19 - Simple FFT Circuit.

It can be observed here that figure 19 is simply the software realization of the FFT diagram shown in figure 18. Taking a closer look, you can see the USRP in the upper left corner; it is configured to receive the target frequency. The *Subdev* spec property maps the two antennas to two separate and distinct inputs, thus allowing the use of one unit to measure two different signals. Sample rate is also shown as 5 MHz. There is a tap off from the USRP which goes directly to a File Sink. This is to achieve all of the data that is coming from the USRP for further analysis of the entire input spectrum. Most of the remaining portion of the flow chart is explained above in the method section. The Complex to Arg (arctan) is how the phase is recovered, by taking the arctangent of the complex signal components. Finally, another File Sink saves the raw binary data to a file for later analysis, discussed in the Python section.

Figure 20a shows the flowchart as it is actually implemented in the project. For the most part figure 20a simply takes the circuit in figure 19 and duplicates it for each antenna input and the

output of each section is run into a Subtracting block. The output data from the Subtraction block is saved in the File Sink block. Figure 20b shows the Phase Detection circuit in action.

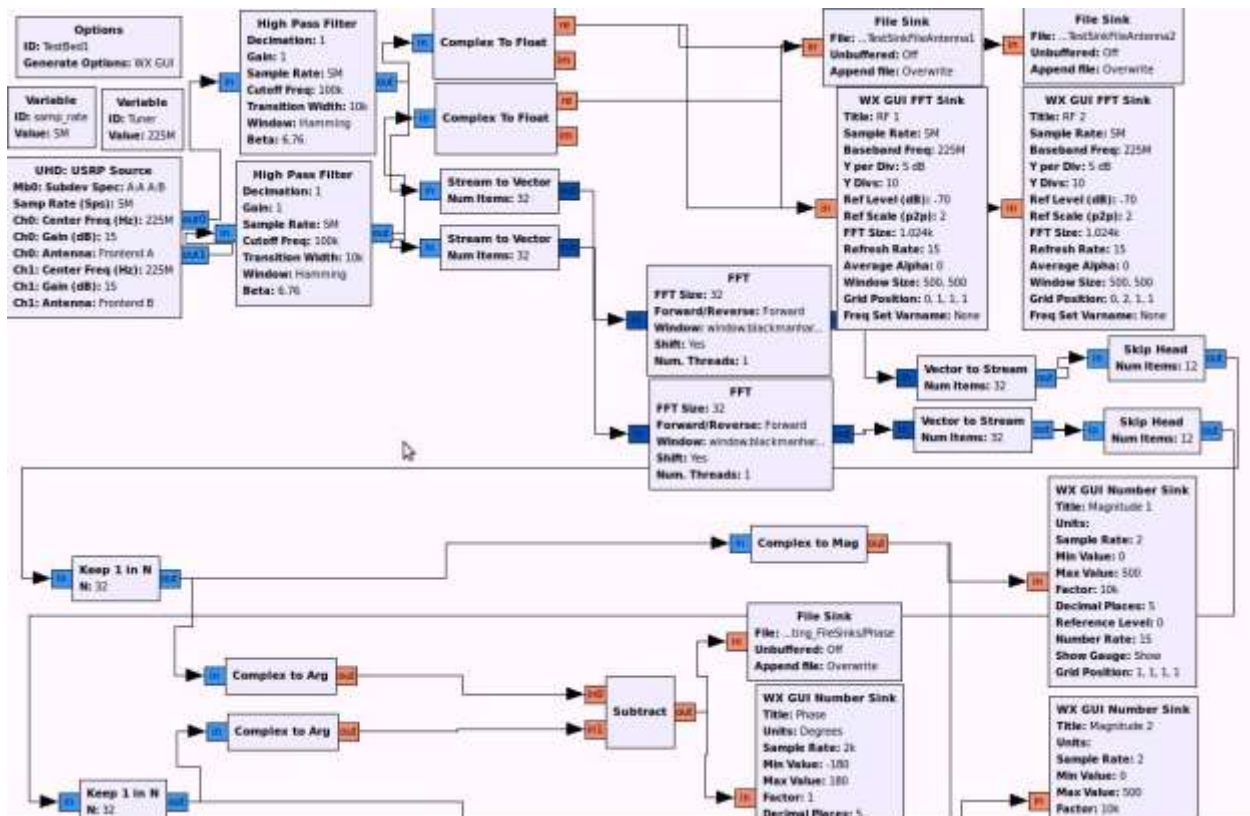


Figure 20a - Implementation of phase detection circuit in GNU Radio Companion software

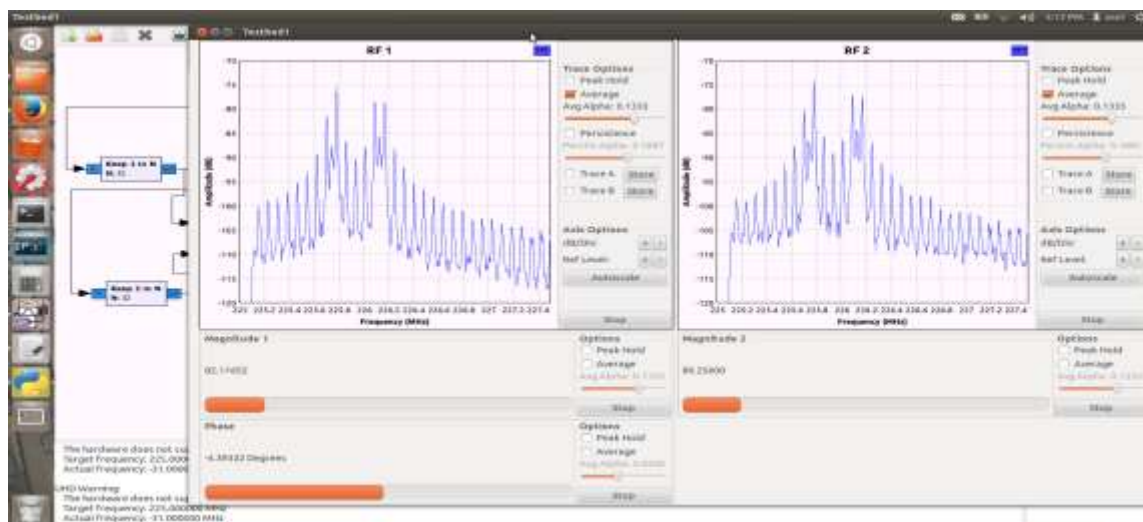


Figure 20b - Phase Detection circuit in operation

Python

Once the raw binary data has been captured into a data file, a handler program must exist to extract the data from the file and do something useful with it. These files quickly accumulate millions of data points and therefore they must be processed accordingly so the user is not overwhelmed with so much information that nothing makes sense. Although any programming language could be used to import the stored raw data files, Python is generally seen as best suited for the task. Python is a scripting language that runs smoothly on top of Linux and is capable of easily importing many free and powerful numerical analysis tools such as Numpy, Scipy and Matplotlib.

The following will provide a high level overview of the data handling scripts, such that it will demonstrate the writer's understanding of the coding process rather than explain the code line by line. As alluded to in the introduction, capturing of the raw data signal information from the USRP was one of the key goals for this project and how this raw data is used is up to the researchers using the test bed. However, in order to exhibit that the design works as described, it was necessary to create demonstration scripts to show proof of progress. Not to be overlooked is the complexity of these scripts. With that said, it is important to consider that the script creates millions of data points that are in a binary format, convert binary format to ascii format, access the file size, loop through the data in small blocks, build arrays to hold the data, index the arrays, perform mathematical computations on the indexed information, save new data to ascii files for use by user interface section and display plots.

Basically, three programs to be developed:

- Max RSS – Finds maximum RSS and list it's frequency within a spectrum of frequencies.
- Phase Handler – Takes in data from phase circuit and plots phase as a function of time, also saves plot information to an ASCII file to be read by the user interface.
- Real Time Display – Reads from the raw data files in real time and plots information giving the user feedback that the USRP is collecting data.

Although each of these programs has their own distinct differences, they also share some common methods. First, let's explore these common attributes before moving on to their unique aspects.

File Handling Method

The NumPy library provides us with a file handler method that when called takes in the file name and options. An options parser routine handles all of the file options that can be passed when the program is called; however, the only options relevant here are the Block and Sample Rate option. The Block option allows you to choose how many data point to bring in at one time and defaults to 1000. This will be useful in determining the array size in the Get Data method. Sample Rate is used for scaling the frequency so it is correct when saved to final ascii

files or plotted. When the GNU Radio software is logging the signal information, a new data point is set every 10.8 microseconds and each data point is eight bytes in length. A data point is in complex form (i.e. $23 + j12$). This all is passed in the raw data, there is no metadata of any kind. Frequency can be calculated using the sample rate passed in at the option block. Obviously, eight bytes every 10.8 microseconds adds up very quickly, to put this into perspective, that's 700000 data points per minute. With these sizes of datasets one needs to use some powerful data analysis tools or in this case, import math libraries. This is why the team has chosen the Block size option that can be set when the file is called. This way the software doesn't 'choke' on the data, such that for example, if you set Block = 10000 only 10000 data points are imported at a time. This shall be discussed in the Get Data section next.

Get Data Method

The get data method converts the binary data to ASCII format so that it can be processed. The amount of data processed at one time is determined by the block option. For instance, if Block = 10000, then 10000 data points are loaded into an array. Even though 10000 seems large it is less than 500 mS worth of data, so the best thing to do is take the average of all 10000 point and loop to the next block of 10000 points. The data iteration is done with a for loop: for count in range (self.files/self.block_length); where the *range* is determined by (file size/block size). Count is initialized at zero and then incremented by one on each loop. Using a loop will allow you to accumulate enough average data points to begin drawing some conclusions about what you are measuring. A note about measuring phase is that the data points are prone to random spikes for short periods of time, so that data is best view averaged out over time.

Now let's examine some of the important distinct aspects of each of the three handler programs that differentiate them from one another.

Max RSS

The Maxx RSS program was the first Python program written for this project and it was created out of a need to understand how the binary data looked after it emerged from the GNU Radio software. Its purpose is to scan a spectrum of frequencies, usually 5 MHz wide, and find which frequency has the greatest signal strength. The information is updated once every 0.10 seconds and continually saved to a file. An initial problem that the team faced was that nobody really understood what the binary data was. Figure 21 displays how the binary data looked coming from the USRP.



Figure 21 - Binary data from USRP

Once the binary data is imported through the file handling method described above it may be mathematically manipulated as seen fit. The Max RSS needs to know two things for a given block of data, the magnitude of each data point and its frequency. Let's take a block of 10000 points for example, then find magnitude by simply taking the absolute value of the complex number and if you take $20 \cdot \log$ of this, you get the value in decibels. You will do this 10000 times, storing each value into an array named (Mag). Next, find the maximum value out of those 10000 numbers with: `MaxVal.max()`. Then you must index this maximum value in the array with `maximum = np.argmax(MaxVal)`. Frequency is found with the following routine:

```
def calc_freq(self, time, sample_rate):  
  
    N = len(time)  
  
    Fs = 1.0 / (time.max() - time.min())  
  
    Fn = 0.5 * sample_rate  
  
    freq = scipy.array([-Fn + i*Fs for i in xrange(N)])
```

So with this you have two arrays, [freq] & [Mag], each containing 10000 numbers and since you know the index of the maximum magnitude, one can simply call this value in the freq array and have both the maximum magnitude and at which frequency that magnitude is found for a given block. The program iterates through all the blocks and saves on line a data for each loop to a text file.

Phase Handler

The purpose of the phase handler program is to take the raw data collected by the phase program in the GNU Radio Companion software flow graph discussed earlier in this document.

Recall that phase information is collected from each antenna and that output is run into a common subtraction block, whose output is the difference between the two phase signals. This data is what is saved to a file and pulled into the phase handler program. A new data point is save every 37 microseconds, so it doesn't take long for them to add up. The first thing that needs to be done is that the complex number needs to be converted to a phase angle, like this: `ang = np.angle(scidata2, deg=True)`. The phase angles are next loaded into an array of 10000 elements and then an average per 10000 points is calculated as such: `averageANG = np.mean(ang)`. It was observed in preliminary research that the phase angle would flip 180 degrees constantly, which made to data difficult to comprehend, so a few lines of code were added to keep all of the phase data positive. After the program completely iterates through a file it will display a plot of both the RSS and Phase data over time. Figure 22 shows the output of the phase handler program. Data from the program, including RSS from both antennas,

phase and time, is saved to an ASCII text file for use in the user interface. The data is also plotted and display on the terminal screen

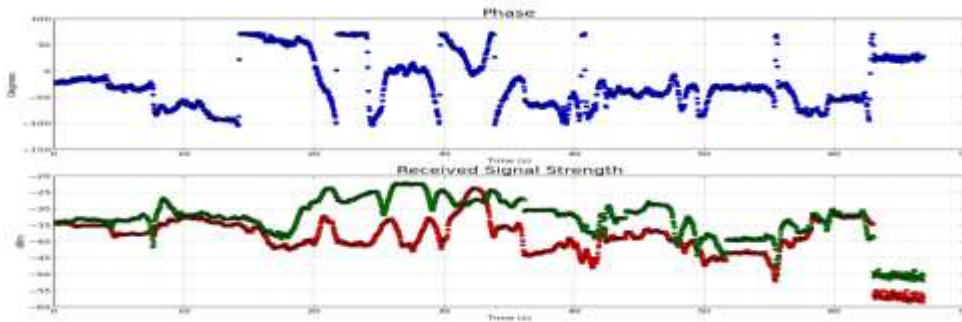


Figure 22 - Phase Handler program output

Real Time Display

The purpose of the real time display program was to give the user some sense of how the phase and RSS data looks as it is being save to a text file. The body is the same as the phase handler program; however it is designed to be ran at the same time that the GNU Radio companion phase program that is pulling data from the USRP. The Real Time program continuously pulls the last eight bytes data from the file that the URSP is updating as it is receiving the spectrum frequency information. This is done like this: `f.seek(filesize-8)`, where `filesize` is: `filesize = os.path.getsize(path)`. This information is then plotted in real-time in a scatter plot.

BeagleBone Black

Release only a year ago, the BeagleBone Black (BBB), seen below in figure 23, is a single-board computer produced by Texas Instruments and improves upon the BeagleBone. The device is powered by TI's AM3358/9 chip and provides USB and HDMI connectivity, Ethernet networking, low-level peripherals via onboard pins, and is capable of running numerous full operating systems including Android, Ubuntu, and Windows Embedded.

The BBB came install with the Linux distribution Angstrom. This was replaced by another distro called Arch on Arm; the port of Arch Linux for ARM based processors. Although not official supported by the makers of BBB, Angstrom's lacking package repository and complicated build mechanisms force a need for change. Furthermore, the Arch community is well known for their package maintenance, active community and provides more stable support for features. Documentation provided by the Arch on Arm community detailed the installation steps which included partitioning a micro SD card, formatting each partition, downloading and extracting the boot loader and Arch image, and lastly booting the BBB from the SD.



Figure 23 – BeagleBone Black

Wireless Networking

Sensor mobility and connectivity are vital features of the RF test bed as they allow the testing of RF geo-location algorithms that rely on an arbitrary number of sensors. In order to achieve these design goals, wireless communication and a portable power supply are needed. The portable power supply was not implemented though as two USRP devices experienced fatal power issues under official conditions and the development of a stable and tested mobile power source overreached the time constraints of the project.

Wireless communication, though not enough by itself to provide mobility, has been implemented by taking advantage of the fact the BBB runs a complete OS with WiFi interfacing. WiFi is a well-known standard for wireless communication and is easily implemented by a USB WiFi dongle connected to the BBB. Another advantage of using WiFi is the ease in which an ad hoc network can be implemented. An ad hoc network is a decentralized type of wireless network that allows any number of sensors to communicate directly with each other. In the future, this type of network can be extended into more complicated and stable forms.

GPS

The module used in the global positioning system utilized in the geolocator is called the Adafruit Ultimate GPS Breakout v3. This module is important because without it, the geolocator would not be able to position where on a map the desired emitter could be. Reasons for choosing this module include low price (\$40), a high (-165 dBm) sensitivity, it is RTC battery-compatible, and has only a 20 mA current draw.

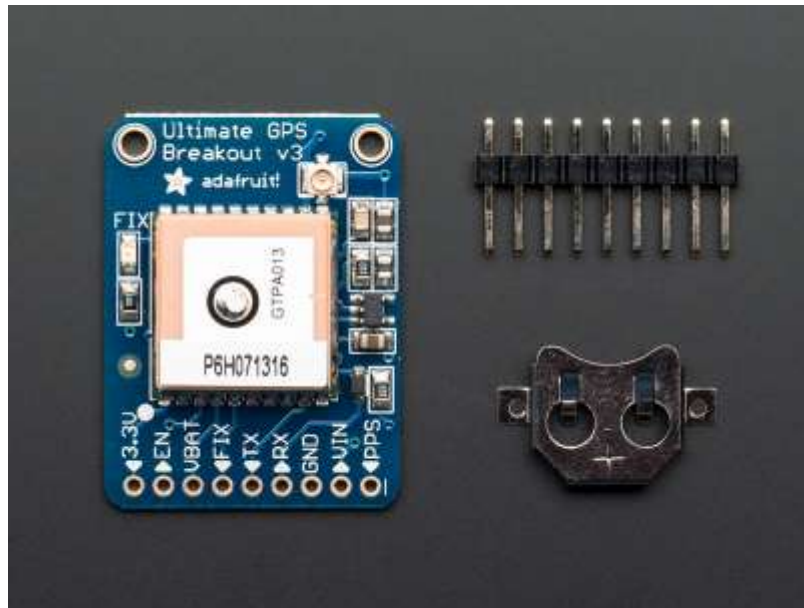


Figure 24 - Ultimate GPS Breakout v3

The GPS module was shipped attached to its cape, as shown in figure 24 above, so the only soldering that was necessary was that to the I/O pins. Once the I/O pins were soldered on, the GPS could easily be plugged into any breadboard. Once the GPS module is powered on, it takes a moment for it to get a fix ("FIX" LED blinks when GPS is finding a fix). In order for a GPS to work, it must lock on to the nearest 4 satellites to determine its location. Without this fix, no data is output from the GPS. Once the fix is found, however, the GPS module outputs a PPS signal used for keeping time accurately. The functionality of the GPS was then implemented in unison with the BeagleBone Black as follows:

- GPS TX output → BBB P9.11 UART4_TX (mode 6)
- GPS RX input → BBB P9.13 UART4_RX (mode6)

With these pins in place, the BBB can use a device tree overlay file (*.dts) that will be compiled into a binary Device Tree Overlay file (*.dtbo) for our BBB to be able to read it. What a device tree does is simple; it is an input type file that asserts values (binary, in our case) at certain nodes in a kernel, making kernel files easier to control without having to download several different versions. This device tree file contains named nodes and properties, in which the

nodes contain properties and child nodes.

In order for the BBB to gather data, a global positioning system daemon (GPSD) must be installed on the BBB. It is important to note that a daemon is simply a computer program that runs as a background process rather than being in direct control of the interactive user. So what a GPSD does is it receives data from the GPS module and relays that data back to an array of applications. Without this daemon, there would be nothing telling the BBB to grab any sort of data from the GPS module. In the most basic of terms, the GPSD tells the BBB to grab and input data in to the Tx and Rx pins, respectively.

Next came installing a daemon for the network time protocol (NTP). A network time protocol daemon is an operating system daemon program that maintains system time in synchronization with servers using the NTP. This daemon tells the BBB to periodically check with servers that the time it has registered is correct. The NTPD then stores time data and then uses the PPS (Pulse per Second) output to accurately measure time even when not connected to the servers.

User Interface

The graphical user interface (GUI) was built in java from the ground up. Netbeans was used as the integrated development environment (IDE) to construct the application. The application graphically displays the location of a source generating a radio frequency. The flow of the application is connecting to the beagle bone. Then it retrieving a data file from a designated directory on the beagle bone. The UI then parses out the data file to calculate usable data in estimate a geo-location for the emitter. Then uses the geo-location to construct a map and displays that map to the user. Google provides an application programmable interface (API) for retrieving maps via a HTTP request. Unlike C++, java provides a library, `java.net.URL`, that makes connecting, retrieving and sending data via HTTP much easier. Based on the parsed data, a formatted string is constructed and sent to Google's API. The API then returns an image file. The API accepts parameters that make it possible to change the format of the image file, size, map type, pin placement and much more. There are four buttons on the application which makes it extremely easy for anyone to use. These buttons components come prebuilt in Java Development Kit 7. Each button is assigned a mouse click action listener that is specific to each button. These listeners will notify the application if a button is clicked and execute specific code for each. All buttons are disabled until the "Sweep" button is clicked. This button starts the functionality of the program. After the "Sweep" button is clicked, the list on the application is populated with all java objects containing information on frequencies found and enables all other buttons. The two buttons next to the list are loaded with icons indicating up and down movement. These allow the user to move up and down the list, displaying different frequencies on the Google map. The fourth button is a toggle button meaning when it is clicked its property `isSelected()` is set to true until it is clicked again. This button is called "Satellite View" and allows the user to toggle between street view and satellite view of the Google maps as seen in the figure 25 below.

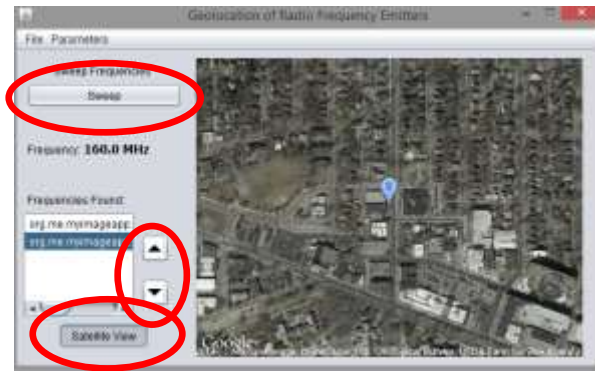


Figure 25 – UI Satellite View

Design Issues

When developing and creating a test bed environment such as the one at hand, many issues with design efforts can be experienced. Due to the broad generality of the team's design, this project is specifically more susceptible to issues due to the need for the project to adapt to the end user's needs, whatever aspect they may be. With that said, the development has required very intricate work to ensure that no uses or flexibility are being cut off for the end user. While tedious, this work on the design has also allowed us to experience minimal issues with the development. Like any development though, many issues were encountered.

Project Lifecycle

With a project that can be as technically loaded as this one, project lifecycle management becomes an even larger part of the project. If certain steps in the project lifecycle cause technicalities, there needs to be time calibration for these errors to be corrected. In the case of team 2, the project suffered a major setback issue with the team's design efforts when communicating with the United State Air Force and Ohio Aviation Institute (OAI) to sort through how funding for the project was to be handled.

At first, when the team was sponsored through the US Air Force, the project requirements stipulated much more detail and accuracy. When trying to understand certain specifications about the details of the project during mid-January, the sponsors put off communications until February 3. Meanwhile, after constructing a lifecycle plan for the project as well as the proposal, the team set even further back when the OAI informed the team that the \$10,000 proposed budget was being lowered to \$5,000 on January 31. With the project requirements that were currently in place, developing the project with less than a calculated minimum of \$6,077 was going to be impossible. The team then took the responsibility of speaking to the OAI (who was handling the proposal letters and project funding) and was able to get them to raise the budget to our bare minimum of \$6,077. When going back to the initial sponsors at the US Air Force with the newly defined budget, the team was informed that, due to time constraints, the Air

Force could no longer sponsor the project. The team's funding were then completely gone and left without a project or funding. To solve the issue, team 2 went to Dr. Grotjohn and expressed the issues at hand and he proposed that the ECE department fund the project. With a more general set of guidelines and requirements, the team was able to make the project work with the funding made available through the ECE department.

Design

The design scope of the geolocator is to provide a test bed for research and development of geolocating RF emitters. Originally, as per the project description, the intention was to build a full-scale network of up to six RF sensor units that would report back received RF data to a central node. This raw data reported from each sensor would contain information about the received signal at each specific sensor location. This information includes: Received Signal Strength (RSS), Angle of signal Arrival (AOA), GPS stamped information about sensors location, as well as a time and date stamp. Additional sponsor specifications called for the sensor units to be capable of receiving an RF signal in the 50 MHz to 2.2 GHz range. Unfortunately due to issues beyond the team's control, the team was unable to use their original project sponsor, so Michigan State University, College of Engineering agreed to sponsor the project; However, they needed reduce to scope of our project, as the original cost of each sensor unit was over \$5000 and the project description called for us to use four to six sensors. The solution the team arrived at was to create a single sensor unit and a system that is scalable, so that multiple sensors units could be added to improve accuracy later. Additionally, to further reduce cost, the model of Universal Software Radio Peripheral (USRP) was changed to a less expensive model; also the team had to reduce to frequency specifications to 1 to 250 MHz, which allowed the use of a much less expensive RF board for the USRP. To further cut cost, the number of USRP units per sensor was decreased from two to one. The original plan was to use two USRPs for calculating phase to meet the angle of arrival specification, but it was discovered that one USRP could be configured to receive two separate signals independently of each other. Towards the end of the semester, the team had problems with the USRP not powering up and had to work with the manufacturer, Ettus Research, to get another one shipped out. As a result, the team was without the USRP for about ten days. Fortunately this was during a phase of the project that involved setting up the development board for the wireless dongle and writing the interface code; the USRP was not absolutely necessary during this phase.

Testing

The testing for the geolocator included staging up outdoors the sensor unit on a tri-pod with two antennas and setting up a transmitter in various locations and at different angles, relative to the receive antennas, 50 feet away from the receive rig. The team

mapped out each location that the transmitter was located at, the time and the angle. Test sessions would last about two minutes and to be viewed later, if needed. Once back in the lab, the data can be analyzed, particularly how the phase changes with respect to the angle of the transmitter changing (AOA), as well as, verifying the Received Signal Strength (RSS) is changing as the geolocator moves closer and further to the transmitter. The RSS and AOA are software programs written by the team, so it was important to verify that phase is being calculated and working correctly. One issue that was noted when analyzing the data was that the signal phase kept jumping around a lot and flipping 180 degrees. It is believed that this is attributed to signal multipath and fading, this is a phenomena that is attributed to the fact that as a signal propagates through its environment, it will tend to bounce off objects which results in changes to the signal characterizes and thus giving erroneous phase data. This problem was solved in two ways: It was found in our research that with more data points collected, the better the signal phase averaged out over time and since the design records millions of data points, the team considered the data points as datasets of 10,000 points each and then only the average of each dataset was saved to an array, thus when only the averaged data points are plotted over time it is much easier to see the true phase change. The second software solution was for handling the phase flipping 180 degrees and going negative. An extra line of code was added to check if the phase was negative and if so, 180 was added to the number, which flipped it back into the positive region. This simplified things such that the geolocator only needed to consider a positive set of numbers.

Standards

Standards are an important part in any development. While standards may be counter intuitive to the test bed design they are a necessity when dealing with aspects of a project such as GPS and Wi-Fi. Due to these standards, the team has experienced minimal issues with respect to the GPS and Wi-Fi. Wi-Fi by definition is any wireless local area network (WLAN) using the Institute of Electrical and Electronics Engineers' (IEEE) 802.11 standards. The Wi-Fi Alliance is a trade association that certifies Wi-Fi products if they conform to IEEE's standards. As of 2012 the Wi-Fi Alliance had over 550 member companies, each having a voice in the Wi-Fi standard. A potential problem with this is it is limited range, limiting the project's scope to a football field sized area. While this will not pose a problem to the results it will limit the test bed to the same size. This issue doesn't directly apply to team 2, but it could be an issue for the end customer who may need a larger project scope.

Global Positioning System (GPS) is a positioning system that provides location and time information anywhere on earth. GPS is maintained by the United States government and is free to anyone with a GPS receiver. There are, however, restrictions on civilian use of GPS receivers. Any receiver capable of operating above 18km altitude and faster than 515m/s is considered "munitions" which require a State Department export

license. A potential problem that we may experience in the future of our development is reading the data from the GPS. The GPS only outputs data in a binary format. Decoding this to a usable format on the Beagle bone may prove to be a challenge and ultimately add to the delay in which the requested information is delivered to the end user.

Reliability

As previously mentioned, a major component of the geolocator is angle of arrival. The current design attempts to use phase to detect an angle of arrival (AOA) within 30°. Because detecting phase accurately enough is proving to be a difficult process, this inaccuracy is directly affecting the reliability of the AOA. Due to these issues, the team has concluded that AOA might be hard to achieve with phase alone. To fix this issue, a design is being implemented that uses both phase and received signal strength in calculating AOA for a more reliable reading that will accomplish the desired reliability and accuracy.

Reparability

The ability to repair a device or product can have great value to consumers knowing that their money is buying something that isn't just thrown in the trash after it shows signs of failure. Team 2's project focuses on two main pieces for the design, the GNU Radio Software, and the USRP 1. The GNU Radio application is completely software based so there isn't a need for reparability in this aspect of our project. Reparability would only extend to re-download the software and installing as necessary. Only the owners and developers of this software have to worry about such encounters. When it comes to the USRP 1, the main hardware component, the manufacturer of the hardware handles reparability. As discussed above in the design section of this paper during the development, problems were experienced such as the USRP 1 not turning on. Due to the complexity of the hardware of the USRP 1, reparability isn't something that can be done easily, unless you have the expertise. To fix this issue the team had to ship the USRP 1 back the Ettus Research, the original manufacturer, where they tested the hardware. After determining the problem was not due to the team's errors, they were sent a second brand new USRP 1. Overall in terms of reparability, the design is unfortunately non-user friendly. Because the hardware comes as a "kit", it is very difficult to do any repairs yourself or from someone other than Ettus.

Retirement

When dealing with the retirement of a product such as the one at hand, there are three main concepts to focus on. These include the useful life of the product, the upgradability of the product, and how the product will be disposed of when it no longer usable. The life of a product is also an import aspect to the customer. The GNU Radio

Software and the USRP 1 can both be considered when discussing the life of the product. The software aspect, unfortunately, life of the product is solely dependent upon the community. Because GNU Radio Software is an open source program, the community and how long they wish to support and continue developing it determine the life of the software. GNU Radio Software is relatively new so currently there is no end to support as of now. With regards to the USRP 1, it is not design to handle intense environment conditions. If used in correct environment and correct power is used there should be no issues regarding life of the hardware. But, hardware does have problems and no product will last forever. Another aspect of life of product that is important is how relative is it. The USRP 1 is relatively new so the issue of being old technology is not important in this aspect. The upgradeability of a product can be crucial in some instances and a valid portion of life of the product. As stated previously the GNU Radio Software is solely dependent upon the community. Any type of upgrades or patches will only be produced so long as there is community support of the software. While open source software may have a lack of consistent upgrades, open source allows for possible customizations and patches that normal licensed software wouldn't release. The USRP 1 on the other hand provides limited, but useful, upgradability. When purchasing a USRP 1 is it necessary to also purchase a Daughterboard associated with it. This daughterboard allows for connection of antennas, both single and dual options. Ettus research provides a wide variety of antennas that you can purchase and upgrade to depending upon your specific bandwidth of bandwidths that the USRP 1 is using. This aspect of the USRP 1 provides upgradability and also versatility in design. While costly, there are options to upgrade the USRP 1. Purchasing an upgraded product is the only method to upgrading the USRP 1, excluding the daughterboard. The upgradability of a product plays an important role in marketing to the consumer and disposability can also be a factor in marketing your product. The software obviously doesn't play a role in disposability, but the USRP 1 does. Your choice of power source for the USRP 1 may require special disposal, for example batteries. The USRP 1 itself only consists of a motherboard, daughterboard, and plastic enclosure, all of which are easily disposed of having a relatively minuscule environmental effect.

4 | Test Data with Proof of Functional Design

USRP

Testing of the USRP is relatively simple, but important since we wanted to insure that we could connect to the device and our software designed programs. Figure 26 shows a trial flow graph that our team designed to test the initial operation of the USRP after we first received it.

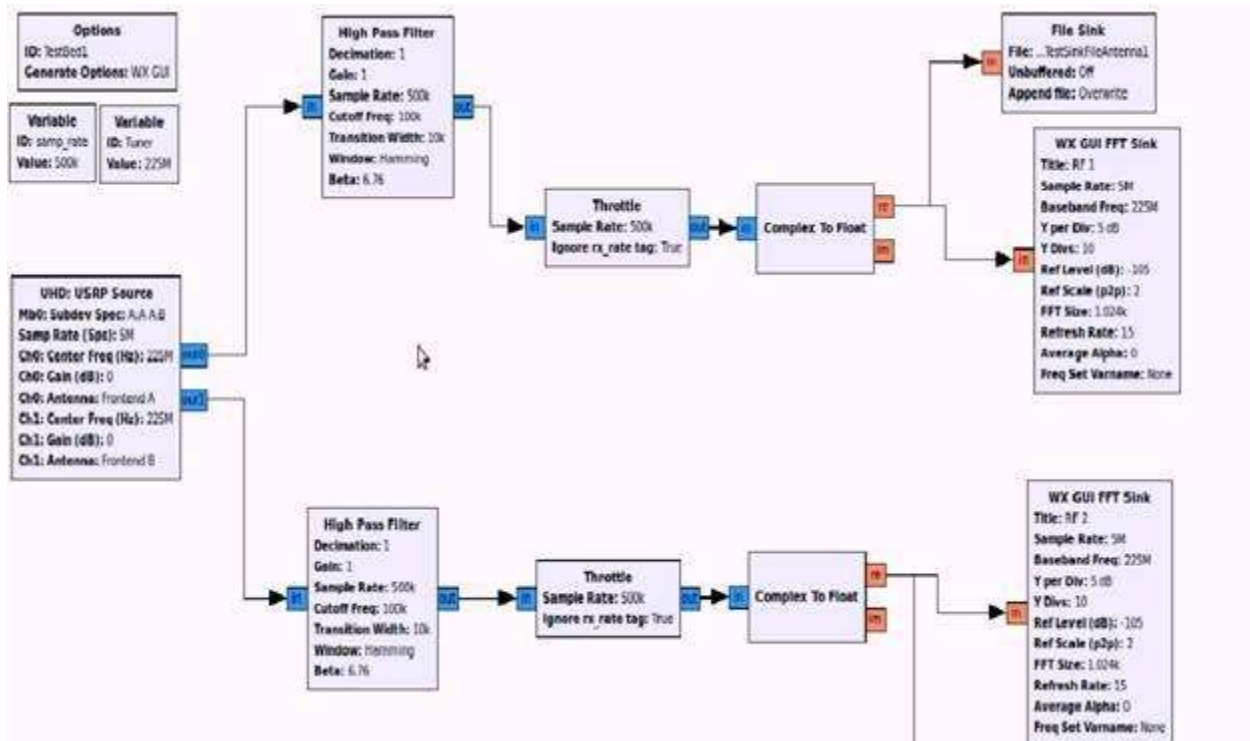


Figure 26 - Initial testing flow graph

Key things that we needed to test were:

- USB interface – Nothing will work if the USB drivers that connect the USRP to the computer are not functional, so this is the first important check to make. We had no problems here.
- Antenna ports – It is possible to configure the USRP so that each of its two antenna ports can handle two signals independently. Note how the USRP source shows two output connections. The documentation on how to map the antenna ports was a little vague and we ended up contacting the manufacture and resolved the issue.
- Ability for USRP to run a simple software program – After hooking up to a computer, we found that we could establish communication with the USRP. We had no problems here.

GNU Radio Companion Software

Once confirmed that the hardware end of things was working correctly (USRP Testing) the team moved on to testing increasingly more complex flow graphs. The user has two forms of feedback available as to the operation of the GNU Radio Companion Software once it is running. They can either display on the screen the received signal in an oscilloscope format (Time Domain) or display the signal in a Fast Fourier Transform (FFT) spectrum analyzer format (Frequency Domain). Figure 27 shows how this would look to the user.

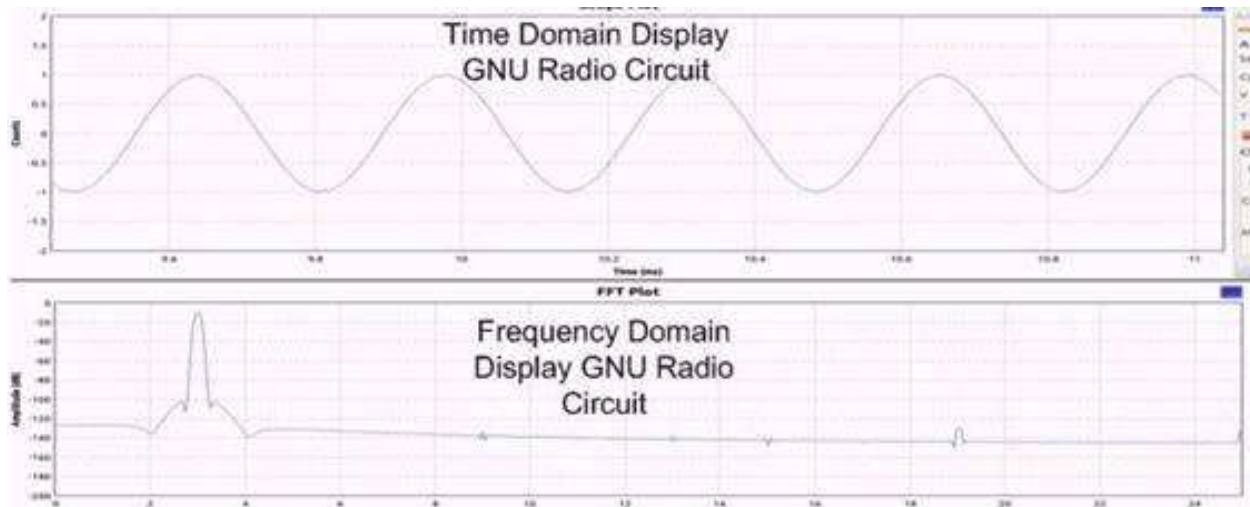


Figure 27 - Display options for GNU Radio Companion

The other form of feedback is what is termed 'File Sink'; this is a method for saving the data to a file.

Testing was mostly trial and error, building a flow graph and seeing how it worked, modifying a parameter so it worked better. The phase flow graph, figure 27, was built in sections since it is so large.

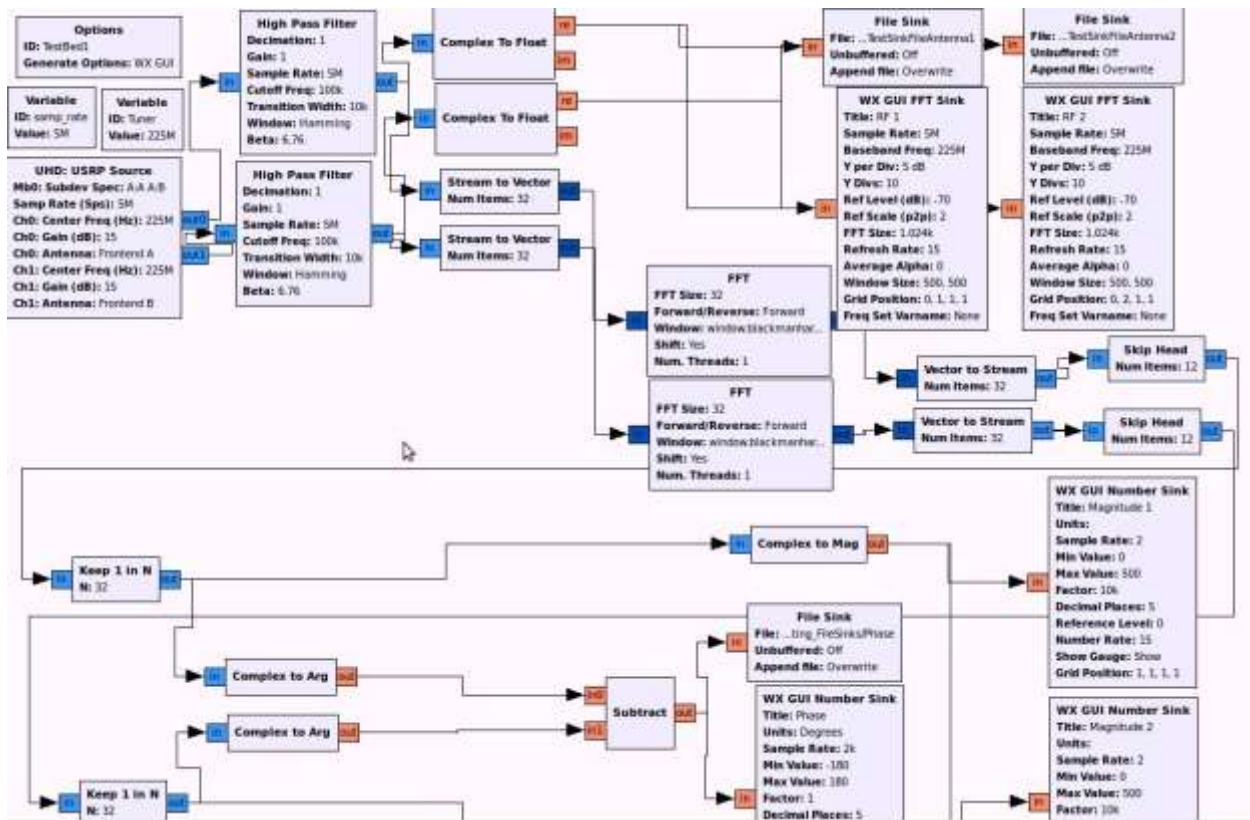


Figure 28 - Phase flow graph

As discussed in the technical section – Phase Method, the flow graph is basically duplicated itself, once for each antenna input, so initially the team built one graph and got it working and then combined it with another graph via a subtraction block. Naturally, it didn't work correctly right away and had to trouble-shoot issues. One drawback is that since everything is software based, you can't just stick a voltmeter or oscilloscope probe in somewhere and test things. You need to get creative and place in scope or FFT GUI sinks into the test circuit, run the program and see what values are at different parts of the circuit. Additionally, there is a 'numbers sink' option that will display magnitude, so this can be used as sort of a 'virtual voltmeter'.

Python Programs

Program writing and testing in Python can prove to be a little tricky, since it doesn't really have a good Integrated Development Environment (IDE) to code from, all of the coding needs to take place in a simple text editor. For those used to coding from Visual Studio, this can prove to be a big change. Two useful aspects of using a IDE is that it checks your code in real time as it is entered and also it allows you to set breakpoints. Python, as we used it, has none of this, which can prove challenging for those of us who got a little lazy using Visual Studio and became accustomed to quickly pushing code into the editor and not worry about mistakes, as they would immediately be point pointed and suggestion give on how to fix them; also, an IDE has an auto complete feature, so usually only a few characters are needed to be entered before a

suggestion is made. Needless to say, it became a rather drawn out trial and error session, but once the writer became more accustomed to the language, mistakes were far less prevalent. One thing about the lack of IDE that proved frustrating through the whole process was not having the ability to set breakpoints, which allow the coder to halt the program at a given location in the code and step through it line by line. One reason this is useful is to check variable values at a certain point, the work around for this is to seed the code with print statements and then when it is ran, variable values can be printed to the screen. Figure 29 shows this operation in action. Note how in the top of the terminal screen we see: *files, block, loops (files/blocks)* after this we see the actual text showing signal properties that is useful to the end user. These short print statements give the coder a glimpse of what is going on in the code and will be removed for the final product.

```

user@ubuntu: ~
files: 2381283
block: 100000
Loops (files/block): 23
Phase: ; 4.37102375 ; Magnitude: ; 3.5357453125 ; Time: ; 0.434
Phase: ; 3.442755625 ; Magnitude: ; 3.5685475 ; Time: ; 0.868
Phase: ; 3.520225 ; Magnitude: ; 3.5831475 ; Time: ; 1.302
Phase: ; 3.6186621875 ; Magnitude: ; 3.57107375 ; Time: ; 1.736
Phase: ; 12.4647775 ; Magnitude: ; 2.9795040625 ; Time: ; 2.17
Phase: ; 27.14563 ; Magnitude: ; 0.90172671875 ; Time: ; 2.604
Phase: ; 32.860855 ; Magnitude: ; 0.875538515625 ; Time: ; 3.038
Phase: ; 33.373835 ; Magnitude: ; 0.843179296875 ; Time: ; 3.472

```

Figure 29 - Python code after execution

GPS

Testing the GPS is actually a quite simple process. The Ultimate GPS Breakout v3 is set up so that once the module gets a fix on four GPS satellites, it outputs a 1PPS signal from the PPS pin on the module itself. Once this was observed with a simple set of commands shown in the figure below, you must check that the GPSD is working as well as the NTP.

```

ubuntu@dragon-BBB-bone28:~$ sudo su
root@dragon-BBB-bone28:/home/ubuntu# ppstest /dev/pps0
trying PPS source "/dev/pps0"
found PPS source "/dev/pps0"
ok, found 1 source(s), now start fetching data...
source 0 - assert 1386573099.016106876, sequence: 31 - clear 0.000000000, sequence: 0
source 0 - assert 1386573100.016138373, sequence: 32 - clear 0.000000000, sequence: 0
source 0 - assert 1386573101.016232249, sequence: 33 - clear 0.000000000, sequence: 0
source 0 - assert 1386573102.016286932, sequence: 34 - clear 0.000000000, sequence: 0
source 0 - assert 1386573103.016353207, sequence: 35 - clear 0.000000000, sequence: 0
^C

```

Figure 30 - Terminal window commands to test PPS signal

The process in testing that the GPSD was working goes as follows:

```

ubuntu@dragon-BBB-bone28:~$ cgps

```

And you should see an output of something like:

```

ubuntu@dragon-BBB-bone28: ~
Time: 2013-12-09T09:45:27.000Z
Latitude: 4 N
Longitude: 1 W
Altitude: 360.6 ft
Speed: 0.3 mph
Heading: 227.4 deg (true)
Climb: 0.0 ft/min
Status: 3D FIX (3 secs)
Longitude Err: +/- 8 ft
Latitude Err: +/- 9 ft
Altitude Err: +/- 15 ft
Course Err: n/a
Speed Err: +/- 13 mph
Time offset: 0.717
Grid Square: CN87vu

PRN: Elev: Azim: SNR: Used:
28 80 020 34 Y
9 60 121 21 Y
8 59 109 20 Y
26 54 255 24 Y
17 39 184 24 Y
135 34 194 28 Y
15 30 305 23 Y
11 25 070 27 Y
7 25 121 22 Y
1 19 093 00 N
19 11 039 13 N
18 01 335 00 N

{"class":"DEVICES","devices":[{"class":"DEVICE","path":"/dev/ttyO4","activated":"2013-12-09T09:45:23.715Z","flags":1,"driver":"Gener

{"class":"TPV","tag":"RMC","device":"/dev/ttyO4","mode":3,"time":"2013-12-09T09:45:24.000Z","ept":0.005,"lat":4.0,"lon":-12
{"class":"SKY","tag":"GSV","device":"/dev/ttyO4","xdop":0.69,"ydop":0.80,"vdop":1.58,"tdop":1.08,"hdop":1.06,"gdop":2.12,"pdop":1.90
,"satellites":[{"PRN":128,"el":80,"az":20,"ss":34,"used":true},{PRN":9,"el":60,"az":121,"ss":21,"used":true},{PRN":8,"el":59,"az":1
09,"ss":20,"used":true},{PRN":26,"el":54,"az":255,"ss":24,"used":true},{PRN":17,"el":39,"az":184,"ss":24,"used":true},{PRN":135,"
el":34,"az":194,"ss":28,"used":false},{PRN":15,"el":30,"az":305,"ss":23,"used":true},{PRN":11,"el":25,"az":70,"ss":27,"used":true}
,{PRN":7,"el":25,"az":121,"ss":22,"used":true},{PRN":1,"el":19,"az":93,"ss":0,"used":false},{PRN":19,"el":11,"az":39,"ss":13,"use
{"class":"TPV","tag":"RMC","device":"/dev/ttyO4","mode":3,"time":"2013-12-09T09:45:25.000Z","ept":0.005,"lat":4.0,"lon":-12
{"class":"TPV","tag":"RMC","device":"/dev/ttyO4","mode":3,"time":"2013-12-09T09:45:26.000Z","ept":0.005,"lat":4.0,"lon":-12
{"class":"TPV","tag":"RMC","device":"/dev/ttyO4","mode":3,"time":"2013-12-09T09:45:27.000Z","ept":0.005,"lat":4.0,"lon":-12
2.177798333,"alt":109.900,"epx":2.589,"epy":2.987,"epv":4.830,"track":227.3900,"speed":0.139,"climb":0.000,"eps":5.97}

```

Figure 31 - GPSD test command output

Now to check NTP:

```

ubuntu@dragon-BBB-bone28:~$ ntpq -p
      remote          refid          st t when poll reach    delay    offset    jitter
=====
*SHM(0)             .GPS.             0 l   6   16   377     0.000    -4.117    27.602
+240.140.8.72.in    64.147.116.229    2 u   3   32   377    45.490   -31.781     7.176
-yogi.raggedstaf    128.10.19.24      2 u  10   32   377    60.539   -43.184     6.124
-dns2-ha.uk.syra    94.125.129.7      3 u  11   32   377   155.399  -55.203    11.849
+ntp3.Housing.Be    128.32.206.54     2 u   9   32   377    32.431   -33.210    12.803
ubuntu@dragon-BBB-bone28:~$
ubuntu@dragon-BBB-bone28:~$
ubuntu@dragon-BBB-bone28:~$
ubuntu@dragon-BBB-bone28:~$
ubuntu@dragon-BBB-bone28:~$ ntpq -c rv
associd=0 status=0415 leap_none, sync_uhf_radio, 1 event, clock_sync,
version="ntpd 4.2.6p5@1.2349-o Wed Oct 9 18:59:40 UTC 2013 (1)",
processor="armv7l", system="Linux/3.8.13-bone28", leap=00, stratum=1,
precision=-20, rootdelay=0.000, rootdisp=57.518, refid=GPS,
reftime=d65011a3.c8b889cc Mon, Dec 9 2013 9:53:39.784,
clock=d65011a5.03750ae1 Mon, Dec 9 2013 9:53:41.013, peer=62883, tc=4,
mintc=3, offset=27.279, frequency=22.274, sys_jitter=22.168,
clk_jitter=29.412, clk_wander=8.130
ubuntu@dragon-BBB-bone28:~$

```

Figure 32 - How to check for NTP

Project

The project is broken into stages, each stage may be thought of as something that may be independently tested and is not contingent upon another stage to for testing. This way each individual can work on a separate aspect of the project, which helps in overall efficiency. The staging breaks down like this:

- Stage 1 – USRP/GNU Radio Programs/Python Scripts
- Stage 2 – User Interface

Stage 1 – USRP/GNU Radio Programs/Python Scripts Testing

Figure 33 shows all of the needed equipment for testing stage 1.

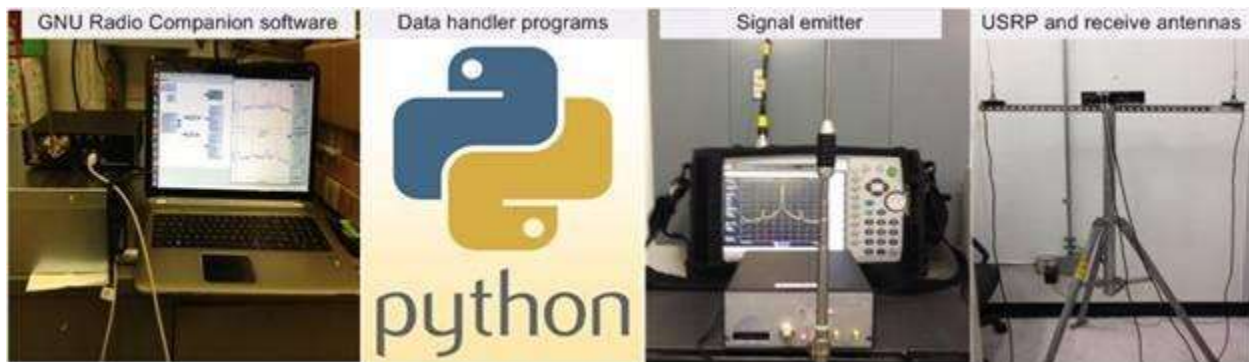


Figure 33 - Stage 1 testing

Once each of the individual elements of stage 1 proved to work, they could all be hooked together to form a usable system. The method for testing stage one was to set up the USRP rig and the node computer running the GNU Companion software in a central stationary location. Then, move the signal emitter around, all the while acquiring the data produced by USRP. Once a few minutes of RF data signal data were recorded the test was stopped and the acquired data was run through the phase handler program and test results can be seen in figure 34a and figure 34b.

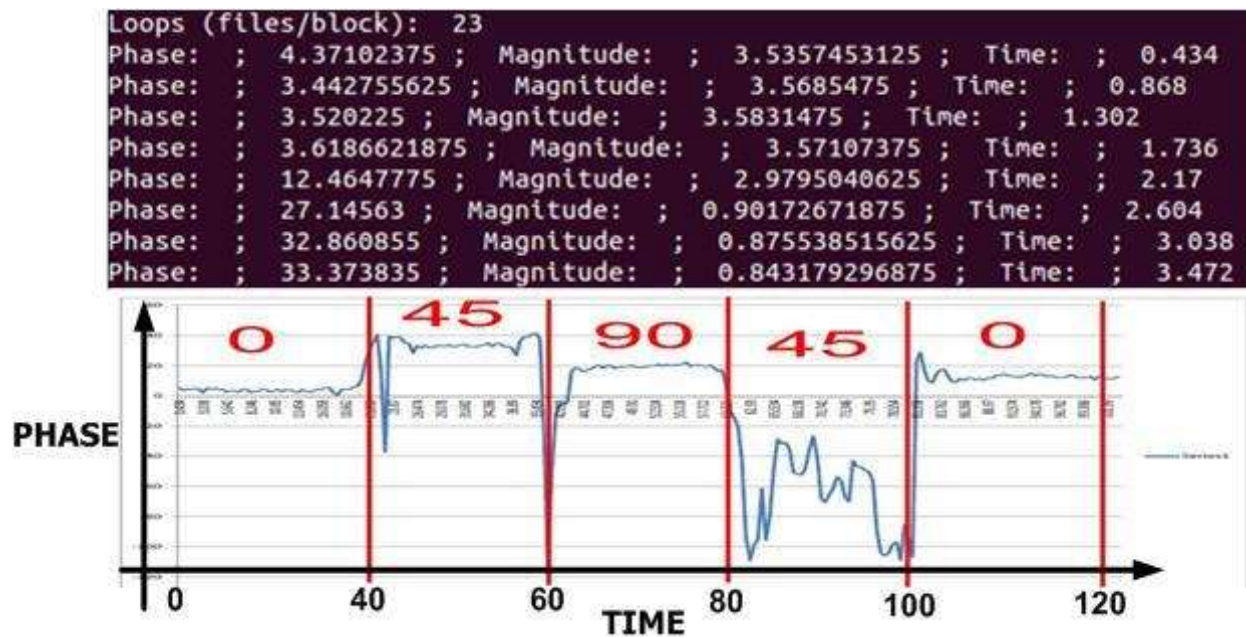


Figure 34a - Beta testing results

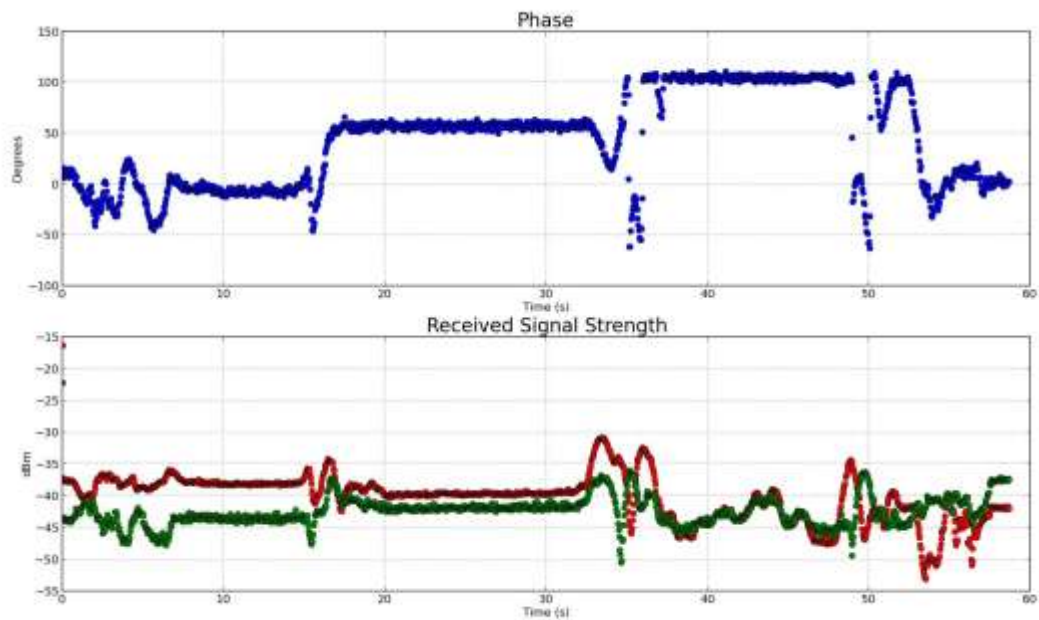


Figure 34b - Finished product testing results

Taking a closer look at displayed results in figure 34a, you can see proof of concept for phase detection. By plotting phase a function of time, one can see that as the angle of the emitter antenna is changed with respect to the USRP antenna and there is an obvious change in phase. Figure 34b shows how the quality of the project output evolved and improved towards the end. Here you see a similar test, but this time a separate plot is added for Radio Signal Strength (RSS) and the RSS from both antennas is plotted (green is antenna 1 and green is antenna 2) All data collection is done by GNU radio scripts executed on the BeagleBone Black via the USRP hardware.

Stage 2 – User Interface

In testing the user interface (UI), there were many aspects that required attention. The first phase of testing the UI involved setting the display to arrange information in a way that was presentable and easily understood. After all the ground work was laid for the UI, the team then implemented the function of mapping. As seen below in Figure 35, are the satellite image and street view image of the UI.



Figure 35 – Street View Image

The mapping portion of the UI proved to be the most challenging and difficult portion of the UI. The Java code takes in latitude and longitude coordinates from the output of the GNU software and produces pinpoints that can be displayed on the map for the current location of the USRP.

Due to the easy usability of our development environment (NetBeans) we were able to reduce the testing required for the UI. NetBeans allows for simple drag and drop of buttons, images, entry fields, etc. and implements the coding for you. Because of this, all that was left for the team was the connection of components and reading the actual data file. This help return significantly less errors when developing the UI.

5 | Final Cost, Schedule, Summary and Conclusions

Successes

Throughout the course of development and design the team succeeded in many aspects. The major aspect of success was the USRP. The USRP was definitely the most difficult portion of the design in a technical aspect. Achieving the functional operation of the USRP that the design team did was a project of its own. Another major success that the design team achieved was the graphical user interface. The user interface is a polished, professional, interface that would be usable in any professional setting. It is efficiently coded to provide a slim overhead.

Failures

With any design project it is expected that one will face failures. There was no short in failures with the design of this project. One major area of failure initially was the GPS and BeagleBone. While helpful to our budget, the low cost of the GPS came at a cost. The GPS can take from 30 seconds to 45 minutes to establish a fix. Due to this the data stream coming in, the user interface can lag behind and not provide accurate assessment on an instant as we had hoped it would.

Another area in which the design team experienced failure was the BeagleBone aspect. While in the end not a complete failure, this was definitely the most difficult portion and the required the most debugging. Do to unsupported software and lack of hardware of the BeagleBone the computation speed and memory size used did not allow the storing of the unanticipated large data file of the GNU Software. Because of this our design does not store old data; rather it deletes old data to provide room for new information.

Suggestions for Future Work

As a test bed development project the team's design has many aspects that could be expanded upon for future work. One aspect of future work, as seen in the Multiple Units Cost table, is expanding the project to a four USRP test bed. Accomplishing this expansion would provide better accuracy and more reliable data to compute. Another aspect of the design that could be expanded is the data provided by the USRP. While this design mainly uses phase and received signal strength the USRP is a powerful piece of hardware that could be used in a different functionality with regards to radio frequency. Expanding upon these two mentioned aspects would provide for a vast amount of work that could be used for a future project.

Final Cost

<u>Components:</u>	<u>Price:</u>
USRP 1	\$ 700.00
BasicRX Daughterboard	\$ 75.00
BeagleBone Black	\$ 64.99
Adafruit Ultimate GPS	\$ 39.95
GPS Antenna	\$ 12.95
SMA Adapter Cable (for GPS)	\$ 3.95
8GB Memory Card (x2)	\$ 10.86
Total Shipping	\$ 46.08
TOTAL:	\$953.78

Additional Projected Costs

The team's final cost came in below the budget specified at \$1,200. Due to the high cost associated with the initial USRP hardware the team used every measure to reduce budget to a manageable cost. One way in which this was done was through the use of already accessible hardware. The components below were instrumental in the project's development and testing. In normal circumstance these costs would be mandatory in the final cost structure.

<u>Components:</u>	<u>Price:</u>
Programmable RF Emitter	\$ 700.00
USRP Antenna (x2)	\$ 160.00
Wi-Fi Dongle	\$ 10.00
Total Shipping	N/A
TOTAL:	\$870.00

Multiple Units Cost

As specified throughout the report this project is an open test bed for future development and implementation. With that said, to effectively implement this design in the field the use of four USRP units would be necessary. This design of 4 USRPs was the original designed proposed to the AFRL Air Force Research Laboratory, the original sponsor of the project. Below is the budget if multiple units were made.

<u>Components:</u>	<u>Price:</u>
USRP 1 (x4)	\$ 2800.00
BasicRX Daughterboard (x4)	\$ 300.00
BeagleBone Black (x4)	\$ 249.96
Adafruit Ultimate GPS (x4)	\$ 159.80
GPS Antenna (x4)	\$ 51.80
SMA Adapter Cable (x4)	\$ 15.80
8GB Memory Card (x8)	\$ 43.44
USRP Antenna (x8)	\$ 640.00
RF Emitter	\$ 700.00
Programmable Power Converter	\$ 100.00
Total Shipping (x4)	\$ 184.32
TOTAL: \$5245.12	

Schedule

<u>Task:</u>	<u>Duration</u>	<u>Start</u>	<u>Finish</u>	<u>Project Work (PW) / Class Deliverable (CD)</u>	<u>Resource(s) / Person(s)</u>	<u>Deadline</u>
Pre-Proposal	5 Days	1/21/14	1/27/14	CD	All	1/27/14
Webpage Started	3 Days	1/23/14	1/27/14	CD	Kenneth	1/27/14
GANTT Chart	3 Days	1/23/14	1/27/14	CD	All	1/27/14
Voice of Customer	3 Days	2/5/14	2/7/14	CD	Joe	2/7/14

<u>Task:</u>	<u>Duration</u>	<u>Start</u>	<u>Finish</u>	<u>Project Work (PW) / Class Deliverable (CD)</u>	<u>Resource(s) / Person(s)</u>	<u>Deadline</u>
Oral Presentation	5 Days	2/6/14	2/12/14	CD	All	2/12/14
FAST Diagram	2 Days	2/13/14	2/14/14	CD	All	2/14/14
Order Parts	2 Days	2/14/14	2/17/14	PW	All	2/17/14
Design Day Program	2 Days	2/19/14	2/20/14	CD	Joe, Viktor	2/20/14
Build GNU Radio Companion	10 Days	2/10/14	2/21/14	PW	All	2/21/14
Final Proposal	10 Days	2/10/14	2/21/14	CD	All	2/21/14
Wi-Fi Network w/ Beagle	4 Days	2/25/14	2/28/14	PW	All	2/28/14
Build Network	2 Days	2/27/14	2/28/14	PW	All	2/28/14
Code User Interface	5 Days	2/27/14	3/5/14	PW	Ken, Matt, Justin	3/5/14
Progress Report #1	3 Days	3/6/14	3/10/14	CD	All	3/10/14
Photocopy of Engineering Notebook	2 Days	3/7/14	3/10/14	CD	All – Individual	3/10/14
Business Canvas Assignment	4 Days	3/17/14	3/20/14	CD	All	3/21/14
Prototype Demo	1 Day	3/17/14	3/17/14	CD	All	3/17/14
Field Testing	3 Days	3/19/14	3/21/14	PW	All	3/21/14
Technical Presentation	10 Days	3/10/14	3/21/14	CD	All	3/21/14
Individual Application Notes	5 Days	3/18/14	3/24/14	CD	All - Individual	3/24/14
Algorithms for Data	4 Days	3/24/14	3/27/14	PW	Joe, Ken	3/27/14

<u>Task:</u>	<u>Duration</u>	<u>Start</u>	<u>Finish</u>	<u>Project Work (PW) / Class Deliverable (CD)</u>	<u>Resource(s) / Person(s)</u>	<u>Deadline</u>
Project Demonstration	1 Days	4/7/14	4/7/14	CD	All	4/7/14
Progress Report #2	5 Days	4/1/14	4/7/14	CD	All	4/7/14
Design Issues Paper	15 Days	3/24/14	4/11/14	CD	All	4/11/14
Demonstration of Working Last-Generation Prototype	1 Day	4/16/14	4/16/14	CD	All	4/16/14
Professional Self-Assessment Paper	5 Days	4/10/14	4/16/14	CD	All - Individual	4/16/14
Final Report	15 Days	4/3/14	4/23/14	CD	All	4/23/14
Final Updates to Web Page	15 Days	4/3/14	4/23/14	CD	Ken	4/23/14
Design Day	1 Day	4/25/14	4/25/14	CD	All	4/25/14
Evaluation of The Contributions of Team Members	2 Days	2/24/14	4/25/14	CD	All - Individual	4/25/14

Conclusion

During the design of this project the team has experienced both technical issues and success. Even with setbacks such as loss of sponsorship and hardware failures the team was able to prevail and find a way to accomplish the project. The main aspect that we set out to accomplish, a functional tested, was accomplished and will provide a great platform for future user to develop and use.

The design of the geolocator of RF emitters was not only challenging but fit directly into the team's strength and interests. This was the main reasoning behind sticking with the initial project as opposed to changing it. The team also fit well in terms of personalities. There was never any major disagreement and each member was open to new ideas when developing the design and implementation of the project.

Throughout the semester the team has had to utilize many skills to accomplish this project. Skills used include programming (multiple languages), physical hardware construction, teamwork, presentation technique, etc. Some of these skills have had to be learned on the spot, while some the team had already acquired. Using these skills and expanding upon them has helped the design team to learn a tremendous amount about RF technology.

Overall the entire project was a success and each team member will walk away with a very valuable learning experience. Experiences from the project were not the only learning aspect, but also from the business and writing aspect of designing a project as stressed in the classroom. These are fundamental critical skills that can be applied to any discipline and will no doubt play an important part in the profession careers of each team member.

Appendix 1 | Technical Roles, Responsibilities, and Work Accomplished

George Godby



George's main technical role was handling the RF aspects, GNU Radio Companion software and Python coding aspect of the project. As an RF specialist for Michigan State University and coding hobbyist George was a natural fit for his role. His credits included reaching out to the USRP manufacturer, Ettus Research, and opening up a professional relationship with a sales engineer which proved to be helpful throughout the entire project. He worked with our team at the beginning to ensure we received the right equipment for the RF acquisition stage of our project and acquired technical assistance from Ettus Engineering when we had questions. George created the software flow graphs in the GNU Radio Companion software which were used to interface with the USRP and collect RF data. Much time was spent early to mid-semester searching websites, reading online academic papers and even watching videos on YouTube for doing research for how the GNU Radio software works and how to build to necessary flow graphs. It was quickly realized that we were entering some fairly undocumented territory and most of the flow graph sections would need to be built from scratch, as opposed to doing a 'cut and paste' job from other work. From mid to end of semester George put together the data handling scripts in Python that handled the information from the USRP and performed numerical analysis, created plots and dumped raw data in ASCII format to files for use by the user interface.

Matthew Roach



Matthew's main technical role was working on the design and assisting in the development of the Graphical User Interface (GUI) to handle the necessary computations and display of final data that would be seen by the user. As a Computer Engineer, this role perfectly suited Matthew and fit the group objective well. With this role, in combination with Justin Mascotto, Matthew had to design a GUI that would be not only easy to use, but also provide the information need and flexibility that the design team needed to maintain throughout their development. The GUI was originally developed with the Python programming language for easy integration with our Linux operating system, as seen with

Matthew's Application Note on "Creating a Python Graphical User Interface". Justin and Matthew, responsible for the GUI, later redeveloped the user interface in Java for better design as well as functionality as the design team was better suited to effectively implement the design through Java. Other technical roles that Matthew participated in included the testing of the GPS hardware. The GPS unit was initially difficult to test do to the inability to get a fix (GPS signal). Due to the Initial lack of mobility with the project, the team had to come up with some creative ways to wire the GPS outside to get a fix. In testing the GPS it was also a necessity to learn how the GPS worked to understand how outputs and different operating modes worked on the GPS hardware. Matthew also played an important part in all documentation and helped to provide the professional quality deliverables that have been demonstrated throughout the semester.

Viktor Simovski



In ECE 480, my team (team 2) is responsible for building a test bed with a geolocator for RF emitters. This project is heavy in the antenna aspect of electromagnetics engineering. Some of the ideas I was working with involved the electromagnetics side of this project and the Ultimate GPS Breakout v3. In this application note, I'll go through the success the class has had with me as an individual while going through some previously discussed goals in the class.

When the project brainstorming was well on its way, we had to begin thinking about some of the constrictions and parameters the project is presenting/will present. First of those things were the most important: Angle of Arrival. With angle of arrival, I immediately knew that multiple antennas were to come into play. Typically, angle of arrival estimation is performed using an antenna array, in which the phase difference between the received signals at each array element is mapped to the incident direction of the signal. So essentially, I had to determine what kind of antennas would best fit our needs to best complete the project requirements as effectively as possible. Also, as always, money has to be kept in mind because the successfulness of the project was also scaled on how much money has to be spent on each part. To do this, I had to research certain types of antennas and determine which would fit our needs and whether there is room to cut cost on the parts being considered. The antenna I decided would do all of this was the standard monopole antenna commonly used RF applications in the real world.

Using knowledge that I gained in ECE 405, I knew to observe the radiation charts of specific antennas to decide on which to use. When doing so, it was pretty elementary thinking to decide on the monopole antenna after also considering a dipole antenna and the patch antenna. For convenience, I cut out the dipole antenna because when comparing between the monopole and dipole, the radiation charts are nearly the same. With that said, a monopole antenna can also be as effective and present a +3dB gain when comparing to the dipole. Before explaining our chosen antenna, it's important to remember that because of angle of arrival, we had to cut down on our frequency band because the wavelength of signals below 60 MHz is far too long for us. This is because in order to calculate AOA, you must have two antennas a certain distance apart and having those 300m apart is not very convenient or realistic for 1 MHz waves since $\lambda = \frac{c}{f}$. With that said, I proposed we chose a monopole antenna that that could be readily available somewhere in the middle of our frequency band. We ended up choosing an 800 MHz monopole antenna that Joe made available to us through the RF department at the physics plant at MSU. With the help of Justin, we went to the EM lab and tested the antennas S_{11} parameters and found the antenna has a -14 dB reflection at about 210 MHz (our ideal frequency).

Justin Mascotto



Justin's main technical role was in the development of the graphical user interface. Having experience from an internship at Intrepid Control Systems in application development and inter language communication in C++ and Java, he was best suited for this task. His role became more prominent midway to the end of the project when the team started integrating all the components and usable data was generated. Although it was never implemented, he also came up with a schema to locate a radio frequency emitter using only RSS as long as the sensor was mobile. The idea behind this schema was if the sensor moved in a circular motion and took the difference in RSS meter to meter, the emitter would lie on the vector with the greatest difference in RSS. This idea was a last resort if phase detection by USRP1 was not accurate enough to implement in our final design. Justin, also having knowledge of antenna's and their behavior, was a great help in thinking of potential antenna design and angle of arrival schema. This knowledge helped with being able to comprehend technical papers and reports researched by the team to develop a feasible design. He also came up with the idea of using a programmable buck power converter to supply the sensor with power from the motor cycle battery. In all, he proved to be a great asset to the design process and UI.

Kenneth Wilkins



Kenneth had technical roles in the high-level design of the RF test-bed, implementing the development board, and wireless networking. Past hobbyist experience with Linux-based operating systems gave him a level of familiarity with the BeagleBone Black setup. Ken was instrumental in the design of the overall system, playing a large role in both selecting components and describing the way in which they would interact. He worked early on with George and the GNU Radio Suite and later moved to implementing the software on the BeagleBone Black.

Much time was allocated in learning the Arch framework and building packages developed for i836 architectures on the arm7vh platform. The development of the BeagleBone Black done by Ken also included the pin setup required to create communication between the GPS module and the board. This hardware level communication was done by the relatively new Device Tree Overlay mechanism found in recent Linux kernels which created some difficulty in finding good documentation. Likewise, the relative 'newness' of the BeagleBone Black, and large popularity of other single-hardware computers forced Ken to adapt documentation from other platforms to accomplish project goals. Lastly Ken setup a simple ad hoc wireless network config that allows the BeagleBone black to communicate with the host PC via the WiFi USB Dongle as proof of concept of a larger decentralized wireless network.

Appendix 2 | Literature and Website References

Websites

Main source of information for Python:

<https://www.python.org/>

Main source of information for GNU Radio software:

<http://gnuradio.org/redmine/projects/gnuradio/wiki>

Main source of information for plotting software library:

<http://matplotlib.org/>

Discussion forum used for questions regarding Python and numerical analysis:

<http://stackoverflow.com/>

Manufacturer of USRP:

<http://home.ettus.com/>

BeagleBone Black Hardware:

<http://beagleboard.org/Products/BeagleBone+Black>

GUI Development Software:

<https://netbeans.org/>

Arch Linux ARM:

<https://archlinuxarm.org/>

Adafruit:

<https://www.adafruit.com>

Literature

Source of reference for data analysis techniques

Python for Data Analysis - Wes McKinney

Copyright © 2013 Wes McKinney.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.