THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF ELECTRICAL ENGINEERING

SOFTWARE-DEFINED RADIO TECHNIQUES FOR GEOLOCATION OF FIRST-RESPONDER TRANSCEIVERS

CHRISTOPHER B. GARDNER
Spring 2010

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Electrical Engineering
with honors in Electrical Engineering

Reviewed and approved* by the following:

Dr. Sven G. Bilén
Associate Professor of Engineering Design,
Electrical Engineering, and Aerospace Engineering
Thesis Supervisor

Dr. John D. Mitchell
Professor of Electrical Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

Abstract

This work explores the implementation of a software-defined solution to radio geolocation. In the event of a natural disaster or other catastrophic event where existing communications infrastructures were destroyed, multiple relief organizations would arrive on-scene and find available spectrum space limited. To assist in dynamically allocating spectrum space, it is valuable to know where each emitter is positioned in the field. A software-defined radio presents an advantage in this type of environment because its waveform parameters, such as operating frequency and modulation type, can be changed in software without any hardware modifications. This allows for a greater degree of interoperability with existing hardware radios that the relief organizations already own.

Classical methods of geolocation rely on signal subspace separation techniques that use computationally complex calculations, such as matrix covariance estimation and eigen-decomposition. These methods are not suitable for mobile applications where size, power, and speed must be optimized. By taking advantage of the multi-stage Wiener filter, which separates the signal space more efficiently, an angle of arrival geolocation algorithm is implemented that is more suitable for the harsh environment of a disaster relief effort.

The investigation of software-defined radio technology is an important topic of research within The Pennsylvania State University's Systems Design Lab. With research interests in wireless sensor networks and satellite communications, the study of software-defined radio techniques is relevant to reducing development times and increasing flexibility in these areas.

Table of Contents

List of Figures

Acknowledgments

This thesis was made possible by the great support from those around me. I would like to especially thank my thesis advisor, Dr. Sven Bilén, for the inspiration to take on this project and for the continued support to see it through. Dr. John Mitchell, my honors advisor, has been instrumental in my success at The Pennsylvania State University—offering guidance at every step of my four years as an undergraduate.

I owe many thanks to Pradyumna Desale, whose work before me on the software models of geolocation techniques provided the foundation for this project. My fellow teammates in the Smart Radio Challenge, Okhtay Azarmanesh, Arnab Banik, and Tejas Nagarmat provided valuable collaboration. I could not have accomplished this without your support.

Most importantly, I would like to thank my parents, Paul Gardner and Vicki Gardner. My education would not have been possible without your encouragement and support throughout the years.

Chapter 1

Introduction

1.1 Need For Geolocation

Many engineering applications require geolocation information. These include navigation, wireless communication, tracking and surveillance, search and rescue, and other emergency response applications. In each of these examples a wireless signal exists, either in the form of constant communication or as a beacon, which can be analyzed to obtain information about the emitter's or receiver's location. The most commonly known use of geolocation today is the Global Positioning System constellation of satellites, upon which much of our aeronautical, maritime, and terrestrial navigation relies. Knowing location has very practical implications. A smart antenna for a wireless telecommunications carrier uses this information to electronically steer the beam of its tower towards a mobile user for improved reception. Airports all around the world use the estimated location of aircraft radios to coordinate flight paths, takeoffs, and landings to avoid collisions. The improvements that geolocation provides to convenience, efficiency, and safety are not limited to these few examples.

The emergency scenario, in which geolocation provides added efficiency and safety, is an area of particular interest. The Federal Communication Commission, for example, has enacted into law the Enhanced 911 requirements in which a wireless carrier must use geolocation information to guide emergency response personnel towards a mobile user in the event of an emergency. In the wireless telephone environment, where the only discriminating characteristic about each

mobile user is which tower the user is connected to, emergency responders could be left with a search area of tens of square kilometers. An exhaustive search would take far too long to ensure a timely response. By using various geolocation and direction finding methods, this search area can be reduced to well under one square kilometer, greatly reducing response time.

1.2 Existing Solutions

Several different methods exist for the geolocation of a radio emitter. Each method presents its own set of advantages and disadvantages, and no single method works perfectly in all environments. Consequently, each method needs to be examined for its suitability in a certain environment before deployment.

1.2.1 Time (Difference) of Arrival

Time of Arrival (ToA) methods use the travel time of a radio signal between its emitter and multiple receiver nodes to estimate the emitter's location. Each time measurement correlates to a distance from one specific receiver using the relation between the speed of light and the signal's carrier frequency. As shown in Figure

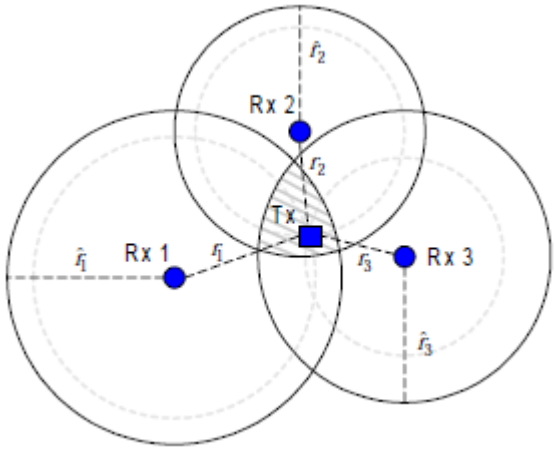

Figure 1.1. Time of Arrival Geolocation Method [2]

1.1, in the 2D case, this distance measurement traces a circle around the receiver for all possible locations of the emitter. By introducing a second ToA measurement,

the new circle will intersect the first locus at either one or two points in space. To eliminate this ambiguity, a third ToA measurement must be used, resulting in a total system using three receivers to locate one emitter. The three circles will converge on a single point in space, and this result represents the determined location of the emitter. Due to the necessity of absolute time measurements, ToA requires precise clock synchronization between the emitter and all receiver nodes, and for this reason is difficult to implement accurately in real-time systems.

To overcome the difficulty of clock synchronization between emitter and receiver, methods of geolocation can be implemented that use the Time Difference of Arrival (TDoA) to determine an emitter's location. When a signal is transmitted, it will reach spatially separated receiver nodes at slightly different times. The TDoA is a result of the different distances from each receiver to the emitter. In

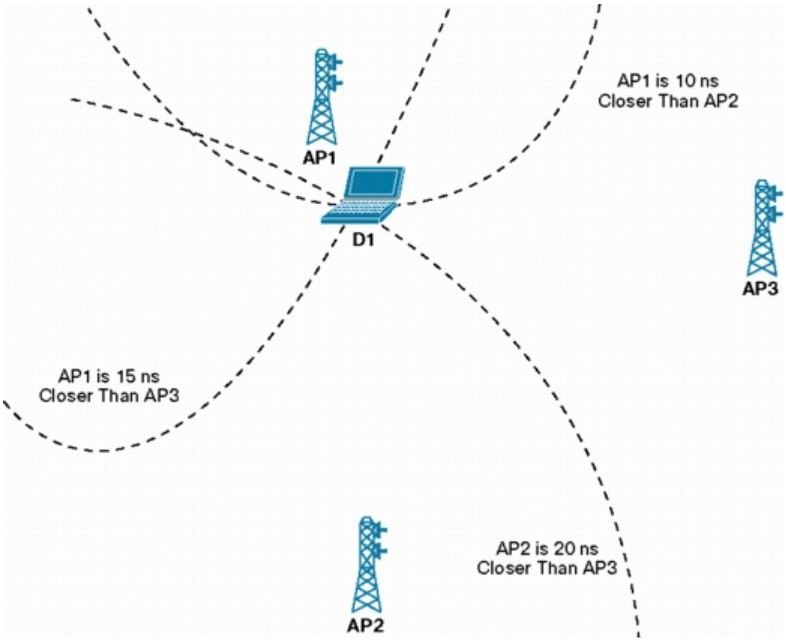

Figure 1.2. Time Difference of Arrival [3]

other words, a single TDoA measurement defines a locus of all points in space from which a transmitted signal would reach the two defined receiver locations with the same time difference. Referring to Figure 1.2, in the 2D case, a single TDoA locus traces a hyperbola of possible emitter locations. Just like the ToA method, a third receiver is required to accurately resolve the emitter's location. The third receiver provides two more TDoA measurements and two more hyperbolic loci that will

converge on a single point in space with the first hyperbola. The point of convergence is the determined location of the emitter. The TDoA measurement only requires the precise clock synchronization of the receiver nodes and therefore is easier to implement accurately in a real-time system. However, the ToA and TDoA geolocation methods rely on both a stationary emitter and stationary receivers and are sensitive to the Doppler shifts that result due to motion.

1.2.2 Earth Absolute Coordinates (EAC)

Perhaps the most common form of geolocation today is the modern Global Positioning System (GPS). GPS uses a reciprocal case of ToA/TDoA with sophisticated error correction calculations for atmospheric and relativistic effects on the time measurements. The system consists of a constellation of 31 actively broadcasting satellites positioned with at least four in each of six orbital planes. From any position on earth, at least six satellites are within line of sight providing a continuous transmission of the time and ephemeris data. This enables a geolocation technique using multiple emitters with known positions to locate one receiver. For example, multilateration can be used to realize a TDoA solution, while a multidimensional Newton–Raphson algorithm can be used in a ToA approach. Solving either of these methods in polar form with respect to Earth absolute coordinates yields the latitude, longitude, and altitude of the receiver.

The main advantage of an EAC-based approach to geolocation is the increased accuracy that satellite navigation provides. A commercially available receiver can locate to within 15 meters using the GPS signals alone and to within 3 meters using Wide Area Augmentation System (WAAS) corrections. WAAS is a system of ground-based reference stations that measure the inaccuracies of the GPS signals and transmit correction signals to geostationary satellites. These satellites broadcast the corrections alongside the GPS signals to improve the accuracy of calculations at the receiver. There is a crucial downside to EAC approaches: they rely on the line of sight reception of multiple satellite signals. Indoor operation is essentially impossible because of the low signal strength, and accuracy diminishes in harsh urban environments due to multipath effects.

1.2.3 Differential Doppler

Analogous to TDoA methods, differential doppler techniques use the Frequency Difference of Arrival (FDoA) of a received signal at spatially separated moving receivers to locate the emitter. With a moving receiver, the frequency observed is different from the frequency transmitted due to the Doppler effect. Because the transmit frequency is unknown, the moving receiver can not use TDoA to accurately resolve the emitter's location. Consider a signal of the form $A\cos(2\pi f_c t)$, where f_c is the unknown transmit frequency. By measuring this signal from a moving receiver for an interval T, the average frequency observed with Doppler is given by

$$f_{av_i} = f_c - (r_{i2} - r_{i1})/\lambda T, \tag{1.1}$$

where r_{ij} is the range from receiver i to the emitter at time $t = t_j$, $T = t_2 - t_1$ is the measurement interval, and λ is the signal wavelength. This f_{av_i} measurement for a single receiver provides a measurement of the change in distance to the emitter over the interval T. However, since f_c is unknown, the use of a second moving receiver to obtain another f_{av_i} allows the FDoA to be calculated for the receiver pair so that

$$f_d = f_{av_1} - f_{av_2} \tag{1.2}$$

$$= \frac{r_{22} - r_{21} - r_{12} + r_{11}}{\lambda T}. \tag{1.3}$$

In this form, the reliance on a known constant value of f_c is no longer necessary. In the 2D case, a single FDoA measurement defines a curve in space where the emitter must lie, which is similar but distinct from the hyperbola defined in TDoA. By adding a third receiver, two more FDoA measurements become available, and the resulting loci will converge on a single point in space identifying the emitter's location.

FDoA implementations have similar challenges as TDoA in that all receiver nodes must be precisely synchronized to correlate measurements. Since the FDoA measurements are dependent on receiver motion, the values for r_{ij} must be accurately measured during the interval T. The advantage of FDoA, however, is the

independence from prior knowledge of the carrier frequency and the ability to use mobile receivers such as those found on surveillance airplanes.

1.2.4 Angle of Arrival

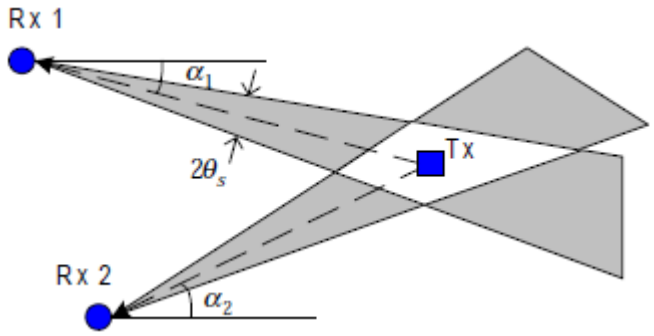

Figure 1.3. Angle of Arrival Geolocation Method [2]

Angle of arrival (AoA) methods can be implemented in two ways. The first requires a topology similar to all of the previous methods in that spatially separated receiver nodes with known positions are required. With each receiver calculating an AoA for the emitter, the intersection of two or more of these rays defines a point in space relative to the receivers where the emitter lies. Noisy signals introduce errors in the AoA method that degrade the estimate to a region in space as shown in Figure 1.3. AoA geolocation in this sense is also known as triangulation, since the mechanism for positioning the emitter is analogous to the angle–side–angle geometric definition for a triangle. By knowing the two angles at each receiver and the distance between them, the triangle containing the emitter is fully defined.

The second way of implementing AoA geolocation is to use a linear array of antennas at one receiver location. Due to spatial separation of about $\lambda/2$, each antenna in the array receives the emitter's signal slightly differently, and knowing the array's parameters, these differences can be used to extract the AoA of the emitter. The advantage of this method is that spatially separated receiver nodes aren't necessary; however, the drawback is that the emitter can only be located on a line, not at a point in space.

1.2.5 Solving Overdetermined Systems

Each of the methods of geolocation theoretically converge to a single point in space where the emitter or receiver must lie. However, the presence of noise in received signals adds error to the range and angle measurements, which prevents obtaining this single point of convergence. Consequently, the solution yields a region of convergence in space that can be further approximated by a least squares method. The single point convergence, such as three perfectly intersecting circles, hyperbolas, or lines, can be used as an adjustable model of the form $f(x, \beta)$ to fit to the measured data and approximate the emitter's location. In this model, β contains all of the adjustable parameters. Given n geolocation measurements, (x_i, y_i), $i = 1, 2 \ldots n$ represent the different curves from a single measurement. The least squares method finds an optimum approximate by minimizing the residual error between the model and the measured data. This sum

$$S = \sum_{i=1}^{n} {r_i}^2, \tag{1.4}$$

where

$$r_i = y_i - f(x_i, \beta) \tag{1.5}$$

produces the optimal approximation of (x, y) when S is minimized. To increase the accuracy of the approximation, more receiver nodes can be added to the system to further overdetermine the system of equations. The least squares method takes advantage of this redundant data to further reduce the residuals.

1.3 Contributions of This Work

Classical geolocation techniques are all fairly well documented and published. They have been used in a wide variety of applications such as radio astronomy, GPS location, aircraft navigation, and sonar ranging in submarines. However, very little has been written in the literature on geolocation using an SDR platform. SDR is a new, cutting edge, rapidly growing field, and the coupling of the Universal Software Radio Peripheral (USRP) boards with GNU Radio to develop SDR applications

is popular among open source developers. Analysis of a geolocation technique for SDR would be of value to future developers in the community. This is the focus of this thesis, in which a low complexity angle of arrival geolocation method is proposed for use on the USRP with GNU Radio.

1.4 Overview of Thesis

With recent events such as Hurricane Katrina and the 2010 earthquakes in Haiti and Chile, one particular area of interest is the ability to track all of an organization's assets in the field using their radios. Ubiquitous positioning of first responders is a very relevant problem in the wireless industry that can be advanced by analyzing low computational complexity geolocation methods, which would be best suited for this scenario and environment. This thesis investigates an efficient implementation of angle of arrival geolocation on a USRP board with GNU Radio that addresses the needs of geolocation for a disaster environment

Chapter 2

Geolocation System

2.1 Overview

The geolocation system described herein is a response to the Wireless Innovation Forum's 2009 Smart Radio Challenge. Designed to show the usefulness of SDR for public safety applications, the challenge is as follows:

> An earthquake has occurred centered in a major metropolitan area measuring 10.0 on the Richter scale. Existing communications infrastructure is out, and as emergency medical services, police, fire, state and federal emergency management personnel arrive on the scene from all over the world, they all begin setting up their own communications systems to aid in rescue efforts. As more and more personnel arrive, finding available spectrum becomes a challenge resulting in unintentional interference between communications of various services. Develop a cooperative sensing system that will create and maintain a database of public safety emitters on the scene, including emitter location, physical layer parameters such as modulation type and transmit frequency, and an association to which emergency team is using this frequency and waveform.[6]

The overall architecture for the cooperative sensing system involves three major functional components. There is the cognitive relay network, which provides a reliable way to receive data from all emitters in the presence of a harsh RF en-

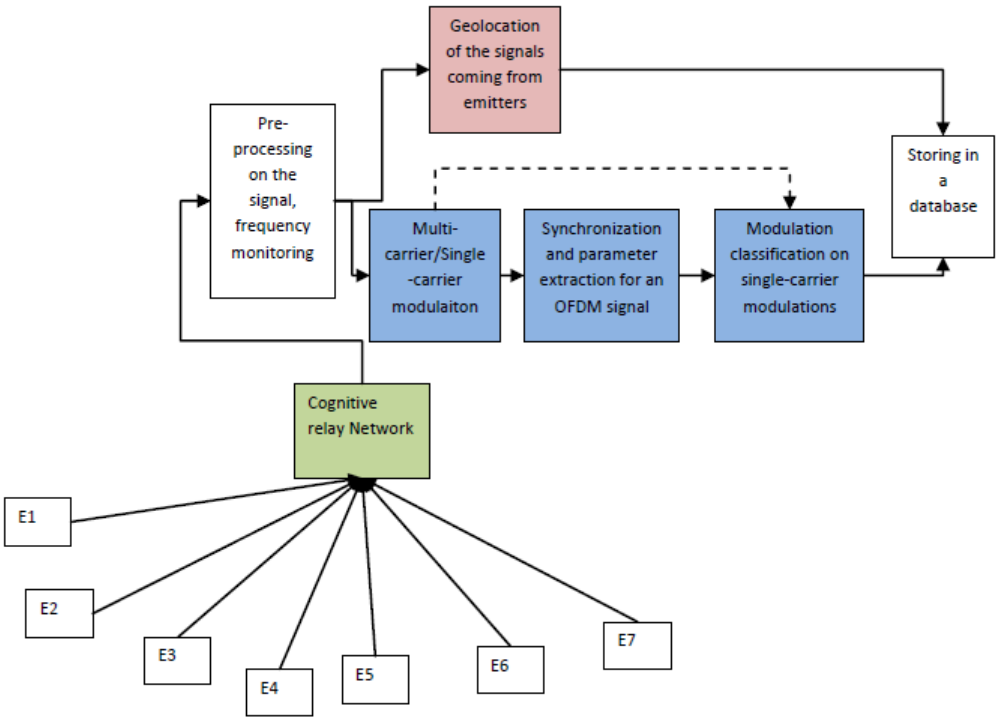

Figure 2.1. Block Diagram of Complete Model [4]

vironment, the modulation classification component, which extracts all necessary physical layer parameters from the received signal, and the geolocation component, which identifies where each received signal is originating. Together, all three components provide the foundation for a spectrum-usage database.

2.1.1 Past Work and Contributions

This software-defined radio technique for geolocation builds on past work completed at The Pennsylvania State University Systems Design Lab (SDL). The hardware implementation is a subsystem of a larger development effort by *Azarmanesh et al.* [4] for the 2009 Smart Radio Challenge. Specifically, the work done by Desale on MATLAB geolocation simulations was a building block for the software-defined radio implementation.

Previous work by *Pauraj et al.* [7] on Estimation of Signal Parameters via Rotational Invariance Techniques (ESPRIT) and *Schmidt* [8] on MUltiple SIgnal Classification (MUSIC) provide the theoretical background for AoA determination given the more recently improved subspace separation method by *Ricks and*

Goldstein [9, 5] using multi-stage Wiener filters.

2.1.2 Theory of Operation

Of the various geolocation methods described in Chapter 1, AoA is the best match for this public safety application. A disaster relief effort presents several conceivable constraints on a hardware implementation. The hardware must be as small, power-efficient, and easy to deploy as possible.

ToA's requirement to have precise clock synchronization between all emitters and receivers makes it a poor match for this system. By the virtue of SDR, each of the public safety organizations on-scene may be using a different hardware configuration in their radios, which would make this clock synchronization difficult, if not impossible, to achieve. A true SDR implementation should not rely on hardware homogeneity as the primary means for radio interoperability. The same argument holds against TDoA implementations as well. Even though emitter–receiver synchronization is not necessary, the need for multiple spatially separated receiver nodes all to be clock synchronized is just as prohibitive. An EAC-based approach using GPS has the potential to be the most accurate implementation. If every node were position-aware, then the transmitting and databasing of this information would become trivial. However, since the RF environment is unknown—ranging anywhere from a rolling rural setting to an underground urban subway system—line of sight access to the GPS constellation cannot be assumed. FDoA presents a new set of challenges in its implementation. The cornerstone assumption is that the receiver antennas are in motion relative to each other [10]. This would require an antenna configuration that is not as small, power-efficient, or easy to deploy as a fixed antenna array.

AoA becomes the most attractive geolocation method for the following reasons: spatially separated receiver locations are not required, which eliminates the need for difficult clock synchronization, and antenna configurations can be as simple as a $\lambda/2$-spaced linear array. The main drawback of AoA is that geolocation is limited to a 1-dimensional line, where the other methods can locate to a point in space.

2.1.2.1 Subspace-Based Methods [1]

Several well-known AoA geolocation methods take advantage of the orthogonality of the noise and signal subspaces within the antenna array's observation data. This fact allows for the separation of the signal subspace from the noise subspace. Then by extracting parameters from the signal subspace using various algorithms, the signal AoAs can be be found. Classical methods for extracting the signal subspace require the estimation of the covariance matrix and its eigen-decomposition. Considering a uniform linear array of M antennas that receive P signals from unique angles $\theta_i = \theta_1, \theta_2, \cdots, \theta_P$, the complex signal output of the M antennas can be written as

$$x(t) = A(\theta)s(t) + n(t), \tag{2.1}$$

where $A(\theta) = [a(\theta_1), \cdots, a(\theta_P)]$ and the individual steering vectors for each direction of arrival are

$$a(\theta_i) = [a_1(\theta_i)e^{j\frac{2\pi d}{\lambda}\sin\theta_i}, \cdots, a_M(\theta_i)e^{j\frac{2\pi(M-1)d}{\lambda}\sin\theta_i}]^T, \tag{2.2}$$

with $a_k(\theta_i)$ being the complex gain and phase response of each antenna in the array, d the array element spacing, and the superscript $(\bullet)^T$ the transpose operator. The vector $s(t)$ is composed of the P complex transmitted signals, and is written as

$$s(t) = [s_1(t), \cdots, s_P(t)]^T, \tag{2.3}$$

while $n(t)$ is considered to be white Gaussian noise. From here, classical methods estimate the covariance matrix of x, the compiled vectors in (2.1), by

$$R_{XX} = E[xx^H] = AR_{SS}A^H + \sigma_n^2\Sigma_n, \tag{2.4}$$

where the superscript $(\bullet)^H$ denotes the Hermitian operator, or the complex conjugate transpose, R_{SS} is the signal covariance matrix $E[s(t)s^H(t)]$, and σ_n^2 is the noise variance. The eigen-decomposition of R_{XX} is as follows:

$$R_{XX}\bar{E} = \Sigma_n\bar{E}\Lambda \tag{2.5}$$

$$AR_{SS}A^H\bar{E} + \sigma^2\Sigma_n\bar{E} = \Sigma_n\bar{E}\Lambda \tag{2.6}$$

$$\bar{E}^H AR_{SS}A^H\bar{E} = \bar{E}^H\Sigma_n\bar{E}\Lambda - \sigma^2\bar{E}^H\Sigma_n\bar{E} \tag{2.7}$$

$$\bar{E}^H AR_{SS}A^H\bar{E} = \Lambda - \sigma^2 I \tag{2.8}$$

$$AR_{SS}A^H = \bar{E}^{-H}[\Lambda - \sigma^2 I]\bar{E}^{-1} \tag{2.9}$$

$$AR_{SS}A^H = \Sigma_n\bar{E}[\Lambda - \sigma^2 I]\bar{E}^H\Sigma_n. \tag{2.10}$$

Since $AR_{SS}A^H$ is of rank P, the P largest general eigenvectors of R_{XX} are the nonzero general eigenvectors of $AR_{SS}A^H$. Finally, the signal subspace E_S can be represented by these P eigenvectors, written as $E_S = \Sigma_n[e_1|\cdots|e_P]$ and the remaining eigenvectors represent the noise subspace, $E_N = \Sigma_n[e_{P+1}|\cdots|e_M]$.

2.1.2.2 Wiener Filters

The classical subspace separation method, although fairly straightforward in theory to extract the signal subspace, is computationally complex requiring $O(m^2N)$ computations for the covariance matrix estimation and $O(m^3)$ computations for the eigen-decomposition. This results in a total computational effort of $O(m^2N + m^3)$ [11]. It can be shown that, by using Wiener Filters, the signal subspace separation can be achieved with a computational effort of only $O(PMN)$.

The Wiener filter (WF) provides a way to estimate a desired signal $d(t)$ from the observation data $x(t)$ that contains noise by using minimum mean-square error (MMSE) performance criteria,

$$w_{w_f} = \arg\min E\{[d(t) - w^H x(t)]^2\}, \tag{2.11}$$

where $w^H x(t)$ is the estimate of $d(k)$, and the WF w_{w_f} is a $M \times 1$ complex linear filter. The classical WF has the form $w_{w_f} = R_X^{-1} r_{Xd}$, which, since it includes the inverse of the covariance matrix, is just as computationally complex as the earlier subspace separation methods. It has been derived in [5] that a multi-stage Wiener filter (MSWF) based on the data-level lattice structure is defined by:

Initialization

$d_0(t) = s_1(t)$ and $x_0(t) = x(t)$ as defined in (2.3) and (2.1), respectively

Forward Recursion

For $i = 1, 2, \ldots, D$ where $(D \leq P)$

$$h_i = \frac{E[x_{i-1}(t)d(t)^*_{i-1}]}{||E[x_{i-1}(t)d(t)^*_{i-1}]||} \tag{2.12}$$

$$d_i = h_i^H x_{i-1}(t) \tag{2.13}$$

$$x_i(t) = x_{i-1}(t) - h_i d_i(t) \tag{2.14}$$

Backward Recursion

For $i = D, D-1, \ldots, 1$

$$w_i = \frac{E[d_{i-1}(t)]}{||E[x_{i-1}(t)d(t)^*_{i-1}]||} \tag{2.15}$$

$$d_i = h_i^H x_{i-1}(t) \tag{2.16}$$

Huang et al. [11, 12] show that all the filters $h_i, h_2, \ldots, h_P$ span the signal subspace of x, while the last $M - P$ filters, $h_{P+1}, h_{P+2}, \ldots, h_M$ span the noise subspace. Thus, the signal subspace has been separated from the noise subspace in a very computationally efficient fashion. The output of the MSWF will be a subspace of orthogonal WFs representing the signal and noise subspaces

$$T = [h_1, h_2, \cdots, h_P, h_{P+1}, \cdots, h_M]. \tag{2.17}$$

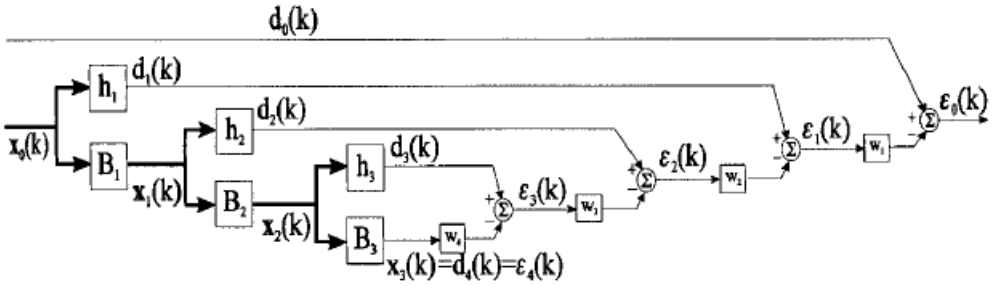

Figure 2.2. A Lattice-Structured Multi-Stage Wiener Filter [5]

2.1.2.3 ESPRIT

The first classical AoA algorithm, which relies on the separated noise and signal subspaces, is called the Estimation of Signal Parameters via Rotational Invariance Techniques (ESPRIT). The method is as follows:

1. Define the signal subspace matrix $E_S = T[:, 1:P] = [h_1, h_2, \cdots, h_P]$
2. Define $E_{S1} = E_S[1:M-1, :]$ and $E_{S2} = E_S[2:M, :]$
3. Estimate $\bar{\Psi} = E_{S1}/E_{S2}$ using either least-squares or total-least-squares
4. Compute eigenvalues $\bar{\mu_1}, \bar{\mu_2}, \cdots, \bar{\mu_P}$ of $\bar{\Psi}$
5. Calculate AoAs using $\theta_i = \arcsin \frac{\lambda \arg(\bar{\mu_i})}{2\pi d}$ for $i = 1, 2, \ldots, P$

2.1.2.4 MUSIC

The second classical AoA algorithm is called MUltiple SIgnal Classification. The method is as follows:

1. Define the noise subspace matrix $E_N = T[:, P+1:M] = [h_{P+1}, \cdots, h_M]$
2. Define $a(\theta_i)$ the same as in (2.2)
3. Define $P_{\text{MUSIC}}(\theta) = \frac{1}{a(\theta)^H E_N E_N^H a(\theta)}$
4. Evaluate $P_{\text{MUSIC}}(\theta)$ for $\{\theta | 0 \leq \theta \leq \pi\}$
5. Identify AoAs as the P largest peaks in P_{MUSIC}

2.2 Simulation Model

The work of Desale [4] provides a foundational software model in MATLAB that simulates the effectiveness of low-complexity ESPRIT and MUSIC AoA methods under various conditions. The main parameter that was tested in software, but unable to be replicated in hardware, was varying levels of SNR. The MATLAB model in Appendix A.1 simulated three point-source signals being received on a twelve-element, $\lambda/2$-spaced linear array of antennas. The simulated signals used

random bits of data modulated using 4-QAM over a white Gaussian channel. SNR levels were modeled at 5 dB, 15 dB, 25 dB, and 35 dB. The AoA estimation results for both ESPRIT and MUSIC under varying SNR conditions are shown in Figures 2.3 and 2.4 respectively.

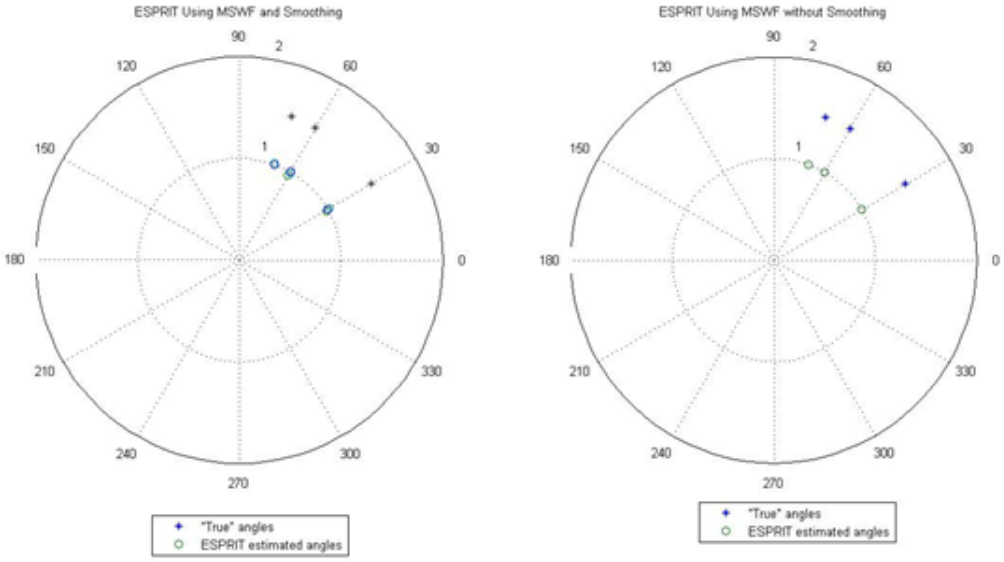

Figure 2.3. ESPRIT Simulation using MATLAB

Since ESPRIT computes a direct estimate of signal AoAs, as opposed to MUSIC which uses an estimator function over all possible angles, the effects of varying SNR are less apparent with ESPRIT. Neither method failed to converge with SNR values ranging from 5–35 dB; however, Figure 2.4 shows that for decreasing SNR, the convergence of MUSIC slows down.

2.3 Real-Time Implementation

Building upon the theory and MATLAB simulations of various AoA geolocation techniques, a real- or near real-time software-defined radio platform serves as another testbed for comparing these algorithms. The runtime environment between a pure software simulation in MATLAB and a true hardware implementation varies greatly. MATLAB has many advanced functions and signal processing blocks

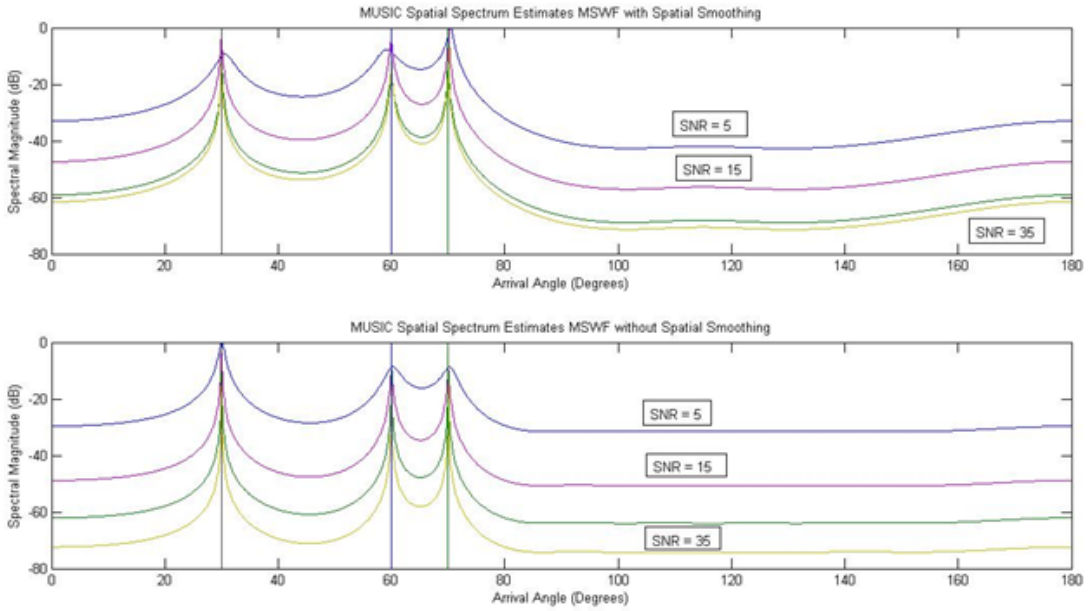

Figure 2.4. 1-D MUSIC Simulation using MATLAB

that can make complex operations easy to perform on large sets of data. However, on a resource-constrained hardware implementation, like the scenario for first-responders in a disaster region suggests, these complex operations translate directly into the speed, size, and power consumption of the mobile sensor. Algorithms chosen must not rely on such computationally complex operations. Using a hardware testbed gives the advantage of evaluating the AoA algorithms in an environment similar to what would be required in the field.

2.4 Hardware Setup

The testbed used to evaluate the software-defined approach to geolocation relies on two key components: the USRP system depicted in Figure 2.5 provides the RF front end and basic signal conditioning blocks, while the GNU Radio software package provides all of the signal processing capabilities needed to create a radio platform. This integrated system is capable of transmitting or receiving any variety of waveforms by interchanging daughter boards with different RF front ends, and it is capable of performing any signal processing that would be required. GNU

Radio's structure allows for a combination of simplicity, flexibility, and power. The heart of the platform is a set of predefined C++ signal processing blocks glued together with the Python scripting language. Rather than having to custom define basic operations such as filters and mixers every time they are required, the included C++ blocks can be called within a Python script, passed the necessary arguments to customize their operation, and run in real-time in conjunction with the USRP. If a certain signal processing block is required that is not native to GNU Radio, a C++ block can be custom written to accomplish any task, and then called just like any other block using Python.

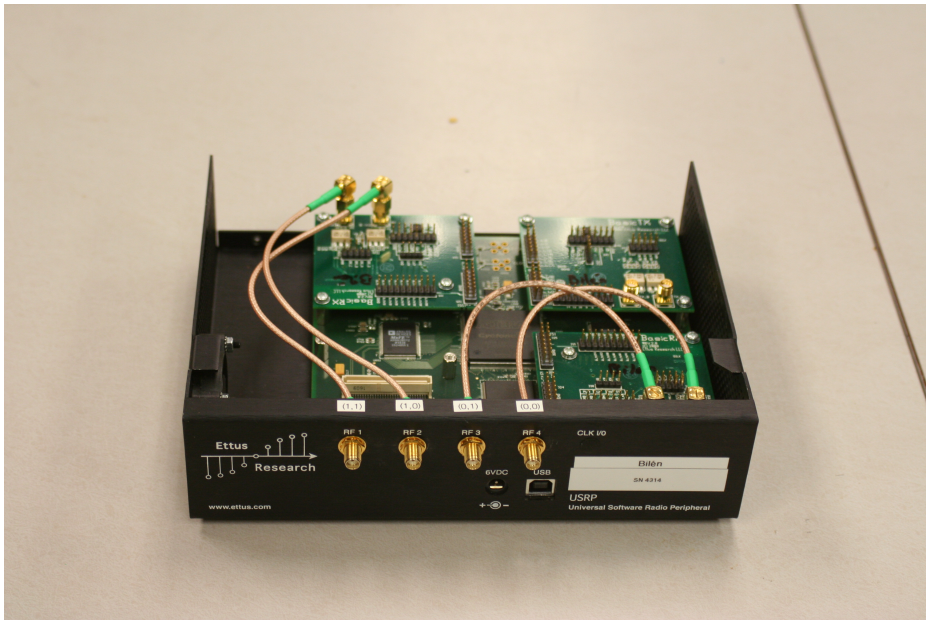

Figure 2.5. The USRP 1.0 by Ettus Research LLC

Developed by Ettus Research LLC, the USRP is intended to be an inexpensive hardware platform used for the development of software radios. The USRP 1.0 features are provided below:

Four 64-MS/s 12-bit ADCs
Four 128-MS/s 14-bit DACs
Four digital downconverters with programmable decimation rates
Two digital upconverters with programmable interpolation rates
High-speed USB 2.0 interface (480 Mb/s)
Capable of processing signals up to 16 MHz wide
Fully coherent multi-channel systems (MIMO capable)

Chapter 3

Results and Analysis

The software-defined radio implementation of low-complexity ESPRIT and MUSIC algorithms was created using the Python code in Appendix B. Several files from the GNU Radio 3.2.2 release needed to be modified in order to perform the AoA calculations on data from the USRP system.

3.1 Tests

The tests for each algorithm were performed according to the data flow shown in Figure 3.1. The USRP was used as a signal sink to record three unique streams of data. These data were then input to GNU Radio and passed through the DQPSK modulator flowgraph in Appendix B.11, which performed the required symbol mapping, modulation, and upconversion to RF where white Gaussian noise was added. The three RF signals were then processed to emulate the effects of being received on an M-element antenna array using the code in Appendix B.4. In the 1D cases, the array was a linearly separated $\lambda/2$ configuration, while in the 2D cases a circular array was used. The M incident signals were then passed through GNU Radio processing blocks to handle the DQPSK signal reception, downconversion, demodulation, and symbol extraction in Appendix B.10. Before the data is extracted in this receiver chain, the signal with carrier information is stripped off and sent to the various AoA algorithms. A fully custom Python code can be found in Appendix B.7 which performs the AoA calculations on received data from the USRP. The results of each algorithm are then plotted using Python's

Matplotlib library. The algorithms of interest for this hardware test were the 1D ESPRIT and 1D and 2D MUSIC methods. All methods utilized multi-stage Wiener filters (MSWF) and were tested both with and without spatial smoothing.

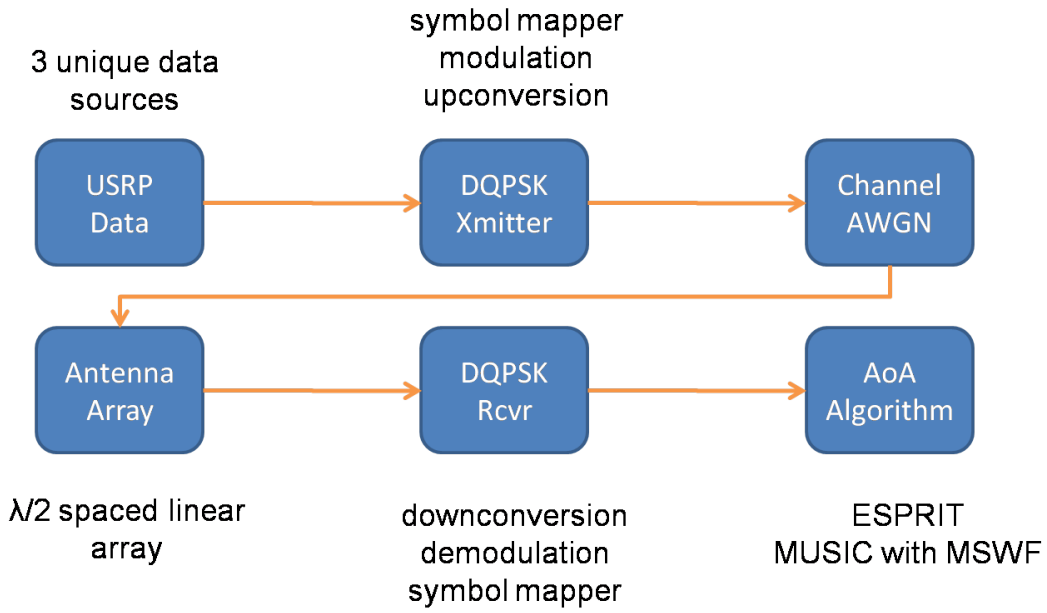

Figure 3.1. USRP and GNU Radio Testbed Configuration

3.2 Analysis

For both the 1D ESPRIT and MUSIC algorithms, test parameters of a $M = 12$ linearly spaced antenna array with $P = 3$ incident signals were used. Both of these methods rely on the separation of the signal subspace from the noise subspace in the observed data, so MSWFs were used to reduce the computational complexity for a mobile, resource constrained platform.

Figures 3.2 and 3.3 show the 1D ESPRIT and MUSIC results. Both algorithms converged on the three predefined angles of arrival: 37°, 109° and 144°. It appears that, similar to the MATLAB simulation, the smoothing method used did not improve accuracy, and actually slowed the convergence of MUSIC.

In Figures 3.4 and 3.5, the 2D MUSIC test used an $M = 8$ antenna circular array with $P = 2$ incident signals at azimuth and elevation angles of (25°, −30°) and (−68°, 50°) respectively. The MSWF method converged on the appropriate

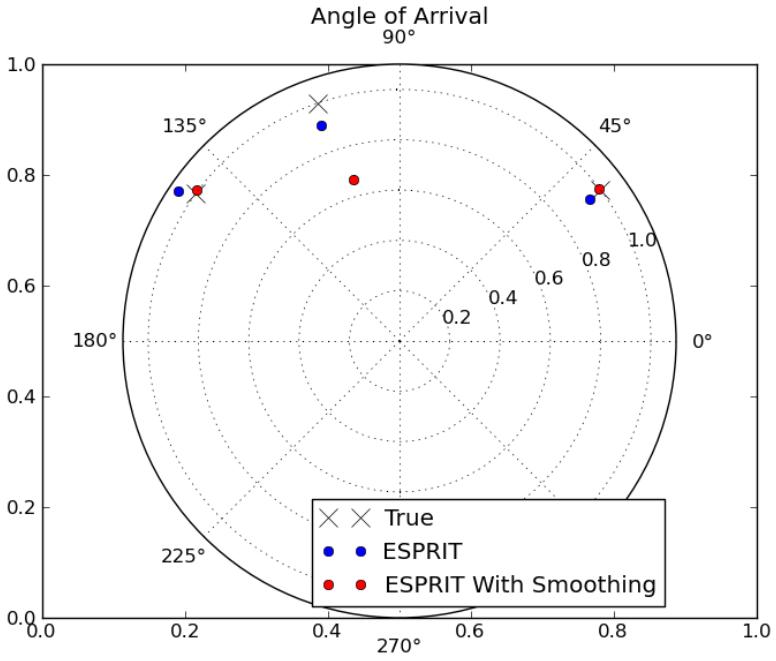

Figure 3.2. ESPRIT Using MSWF with and without Spatial Smoothing

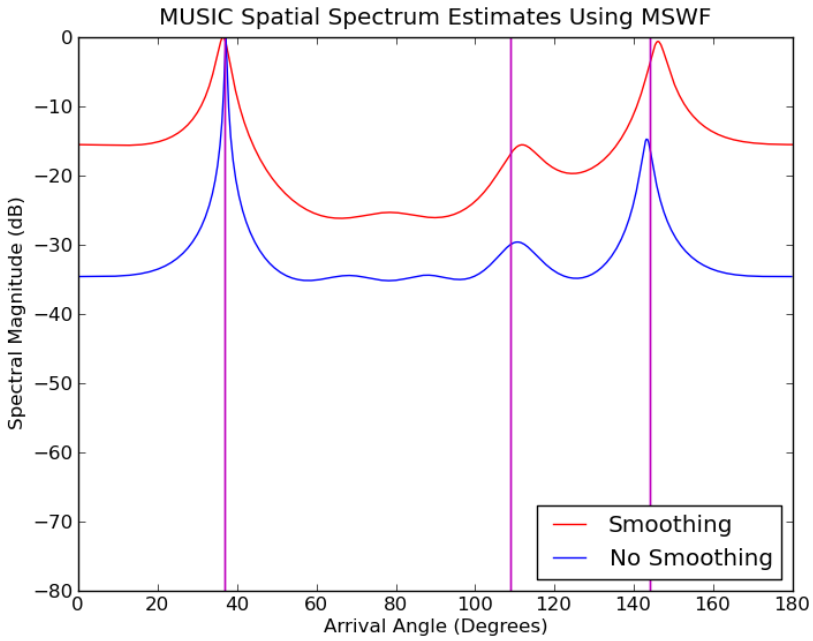

Figure 3.3. MUSIC Using MSWF with and without Spatial Smoothing

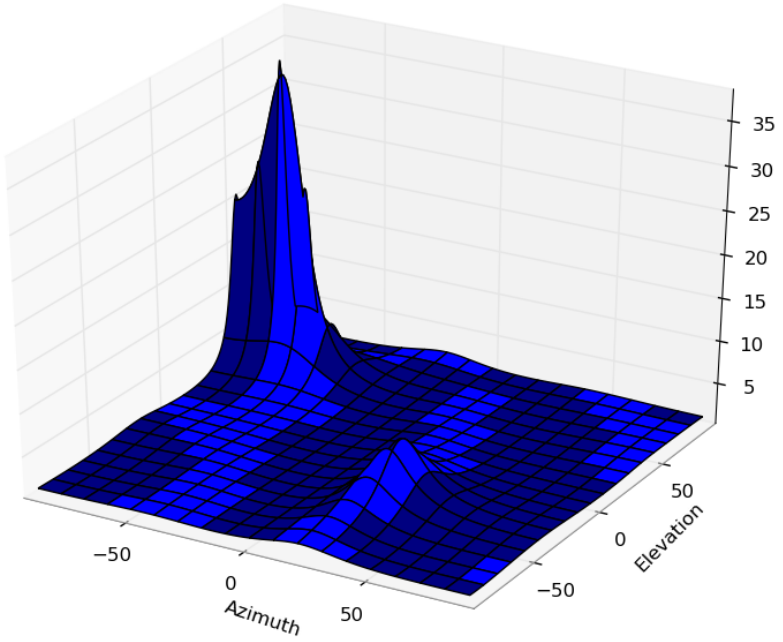

Figure 3.4. 2D MUSIC Using MSWF without Spatial Smoothing

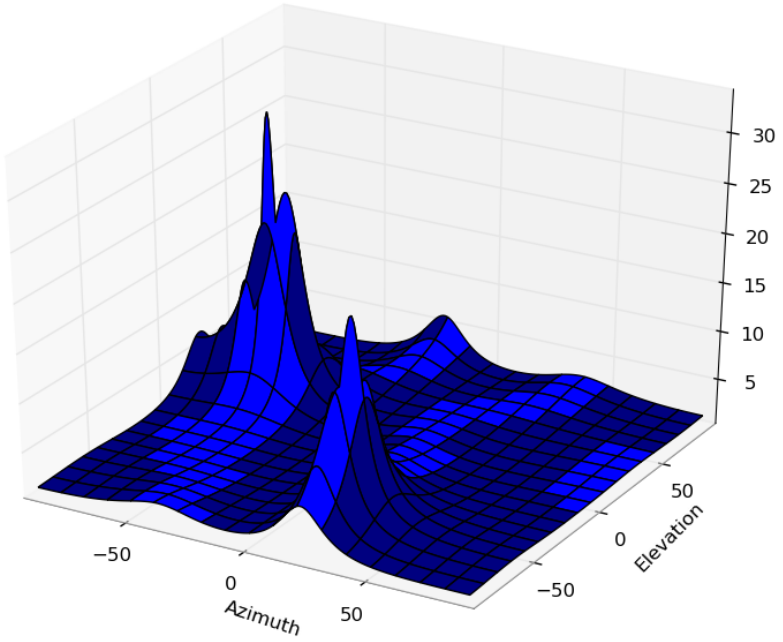

Figure 3.5. 2D MUSIC Using MSWF with Spatial Smoothing

angles as noted by the two peaks in the surface plot. In the 2D case, the effectiveness of the smoothing method is questionable. It sped the convergence of the (25°, −30°) peak; however, it introduced an ambiguous local-maximum near the second peak.

Chapter 4

Summary and Future Work

4.1 Project Summary

These hardware tests show that the USRP with GNU Radio is perfectly capable of performing both 1D and 2D AoA calculations on digital waveforms such as DQPSK. The great benefit of this software-defined radio approach is that all of the waveform parameters such as modulation type, packet structure, and coding scheme are set in software and can be quickly modified. These tests could just have easily been performed using DBPSK or GMSK, and the extension to OFDM is also possible in GNU Radio. Use of MSWF greatly reduced the computational burden on the software-defined radio, while converging on the appropriate angles. Performance on the hardware platform was not as good as in MATLAB simulations, especially in the 2D MUSIC case, which is likely attributable to the usage of real data and noise levels recorded from the USRP.

Overall, a very strong proof of concept for software-defined radio geolocation has been presented. The extension to running live geolocation tests rather than the current off-line processing of AoAs should be straightforward.

4.2 Improvements for Future Design

To further show the applicability of software-defined radio geolocation methods, a new testbed can be created using multiple time-synchronized USRP boards each

with two receive antennas. This will create an actual $\lambda/2$ linearly spaced array that can be used to perform live geolocation tests. In addition, geolocation approaches that combine AoA with other ToA or TDoA measurements, such as Hybrid AoA/TDoA, may be used to resolve the emitter's location to a fixed point in either 2D or 3D space rather than along a 1-dimensional line. Applications of the 2D AoA methods can serve as spatial channel estimation, or as this software-defined radio platform matures, can apply to MIMO communication. Finally, expanding the compatible modulation types of this geolocation system to include OFDM will open the door for today's modern waveforms.

Appendix A

MATLAB Code

A.1 1D ESPRIT and MUSIC

Contents

- Setup
- Multi Antenna Reception
- Channel Noise
- Smoothing Matrix
- Multi Stage Wiener Filter Implementation
- ESPRIT Algorithm
- MUSIC Algorithm

```
close all; clear all; clc;
```

Setup

Define parameters.

```
L = 4;                      % Size of signal constellation
bitspersymbol = log2(L);    % Number of bits per symbol
Nb = 10e3;                  % Number of bits to process
n = Nb;
nsamp = 1;                  % Oversampling rate
fc = 2400*10^6;             % Carrier Frequency = 2400 MHz
```

```
c = 3*10^8;                 % Velocity of Light in m/s
Fs = 1/fc;                  % Sampling Frequency
ts = 4*fc;
```

Multi Antenna Reception

```
angdeg = [30 60 90];        % Angles in degree
angrad = angdeg*pi/180;     % Angles in radians
angles=angdeg*(pi/180);
powers=[10 5 0.5];          % Powers of received signals
P=length(angdeg); % Aassume that the number of signals is available
dfactor=0.5;      % d/lambda = spacing between the antennae
M=8;                        % No of sensors

Rideal = zeros(M,M);        % Predeclare for speed
for i=1:P,
a=exp(1i*2*pi*dfactor*cos(angles(1,i))*(0:M-1)).'; %a(theta)
Rideal=Rideal+a*a';%
bits = randint(Nb,1);     % Random binary data stream signal Source
symbols = bi2de(reshape(bits,bitspersymbol,...
    length(bits)/bitspersymbol).','left-msb'); % Bits to symbols
mod = modem.qammod(16);         % Create modulator object
s(i,:) = modulate(mod,symbols); % Modulate the symbols
if i == 1
    tx = zeros(M,length(s(i,:)));
end
tx=tx+a*s(i,:);
end
```

Channel Noise

```
rx=awgn(tx,15,'measured');  % Add noise due to measurements
```

Smoothing Matrix

```
Ml = 5;                         % Subarray size
L = M-Ml+1;
l = 1;
J1 = [zeros(Ml,l-1), eye(Ml, Ml),zeros(Ml,M-l-Ml+1)];
l = 2;
```

```
J2 = [zeros(Ml,l-1), eye(Ml, Ml),zeros(Ml,M-l-Ml+1)];
l = 3;
J3 = [zeros(Ml,l-1), eye(Ml, Ml),zeros(Ml,M-l-Ml+1)];
l = 4;
J4 = [zeros(Ml,l-1), eye(Ml, Ml),zeros(Ml,M-l-Ml+1)];

X = [J1*rx J2*rx J3*rx J4*rx];  % Smoothed data matrix
```

Multi Stage Wiener Filter Implementation

Assume that the receiver has complete knowledge of transmitted data

```
d0 = [s(1,:) s(1,:) s(1,:) s(1,:)];
d0 = d0.';
x0 = X;

h1 = x0*d0/norm(x0*d0);
d1 = h1'*x0;
x1 = x0 - h1*d1;

h2 = x1*d1.'/norm(x1*d1.');
dot(h1,h2)
d2 = h2'*x1;
x2 = x1 - h2*d2;

h3 = x2*d2.'/norm(x2*d2.');
dot(h1,h3)
dot(h2,h3)
d3 = h3'*x2;
x3 = x2 - h3*d3;

h4 = x3*d3.'/norm(x3*d3.');
dot(h2,h4)
d4 = h4'*x3;
x4 = x3 - h4*d4;

h5 = x4*d4.'/norm(x4*d4.');
d5 = h5'*x4;
x5 = x4 - h5*d5;
```

```
T = [h1 h2 h3 h4 h5];
```

ESPRIT Algorithm

```
Es=T(:,1:P);
Es1=Es(1:Ml-1,:);
Es2=Es(2:Ml,:);
Psi=Es1\Es2;
[T,Phi]=eig(Psi);
Phivec=diag(Phi);
%plot eigenvalues from ESPRIT and compare with true frequencies
polar(0,1,'*')
hold on
plot(real(Phivec),acos(angle(Phivec)/pi)*(180/pi),...
    'ro','MarkerSize',12,'Linewidth',2);
hold off
legend('True "poles"','ESPRIT estimated angles')
angests=sort(acos(angle(Phivec)/pi)*(180/pi))
angdeg;
hold off

T = [h1 h2 h3 h4 h5];
Pn=T(:,P+1:Ml)*T(:,P+1:Ml)';
```

MUSIC Algorithm

```
%compute MUSIC spatial spectrum using all noise eigenvectors
%plot and compare with Minimum Variance spatial spectrum

figure
ell=0;
for theta=0:.1:180,
ell=ell+1;
atheta=exp(1i*pi*cos(theta*(pi/180))*(0:Ml-1));
SMUSIC(ell)=-10*log10(real(conj(atheta)*Pn*atheta.'));
end
theta=0:.1:180;
plot(theta,(SMUSIC-max(SMUSIC)),'r','Linewidth',2);
axis([0 180 -80 0]);
hold on
```

```
for k=1:P,
plot([angdeg(1,k) angdeg(1,k)], [-80 0], 'b','Linewidth',2);
end
legend('MUSIC');
title('Spatial Spectrum Estimates');
xlabel('Arrival Angle (Degrees)');
ylabel('Spectral Magnitude (dB)');
hold off
```

A.2 2D ESPRIT with and without Spatial Smoothing

Contents

- Setup
- Multi Antenna Reception
- Channel Noise
- Smoothing Matrix
- SVD MUSIC Implementation
- Multi Stage Wiener Filter Implementation
- MUSIC without Spatial Smoothing
- MUSIC with Spatial Smoothing

```
close all; clear all;
```

Setup

Define parameters.

```
close all; clear all; clc;
L = 4;                          % Size of signal constellation
bitspersymbol = log2(L);        % Number of bits per symbol
Nb = 6e4;                       % Number of bits to process
n = Nb;
nsamp = 1;                      % Oversampling rate
fc = 2400*10^6;                 % Carrier Frequency = 2400 MHz
c = 3*10^8;                     % Velocity of Light in m/s
Fs = 1/fc;                      % Sampling Frequency
ts = 4*fc;
```

Multi Antenna Reception

```
theta = [-30 50];               % Elevation angles in degree
thetarad = theta*pi/180;        % Elevation angles in radians
phi = [25 -68];                 % Azimuth angles in degrees
phirad = phi*pi/180;            % Azimuth angles in radians

powers=[10 5 0.5];  % Powers of received signals
P=length(theta);  % Assume that the number of signals is available
dfactor=0.5;        % d/lambda = spacing between the antennae
M=8;                % No of sensors

Rideal = zeros(M,M);% Predeclare for speed
k = 0:M-1;
for i=1:P,
a=exp(-1i*pi*cos(2*pi*k/M-thetarad(i))*sin(phirad(i))).';
bits = randint(Nb,1);   % Random binary data stream signal Source
symbols = bi2de(reshape(bits,bitspersymbol,...
    length(bits)/bitspersymbol).','left-msb'); % Bits to symbols
mod = modem.qammod(16);         %Create modulator object
s(i,:) = modulate(mod,symbols); %Modulate the symbols
if i == 1
    tx = zeros(M,length(s(i,:)));
end
tx=tx+a*s(i,:);
end
```

Channel Noise

```
rx=awgn(tx,5,'measured');  % Add noise due to measurements
```

Smoothing Matrix

```
Ml = 7;         % Subarray size = 7
L = M-Ml+1;
l = 1;
J1 = [zeros(Ml,l-1), eye(Ml, Ml),zeros(Ml,M-l-Ml+1)];
l = 2;
J2 = [zeros(Ml,l-1), eye(Ml, Ml),zeros(Ml,M-l-Ml+1)];

X = [J1*rx J2*rx];
```

SVD MUSIC Implementation

```
Rxx = X*X'/Nb;
[U,S,V]=svd(Rxx);
Vs=U(:,1:P);
Vn=U(:,P+1:Ml);

for theta=-90:90
    for fei=-90:90
        k=[0:Ml-1]';
        AA=exp(-1i*pi*cos(theta*pi/180-2*pi*k/8)*sin(fei*pi/180));
        WW=AA'*Vn*Vn'*AA;
        Smusic(theta+91 ,fei+91)=(abs((AA'*AA)./WW));
    end
end
theta=-90:90;
phi=-90:90;
mesh(theta,phi,Smusic);
title('MUSIC with Singular Value Decomposition');
xlabel('Azimuth');
ylabel('Elevation');
zlabel('Estimation');
grid on;
```

Multi Stage Wiener Filter Implementation

Assume receiver has complete knowledge of transmitted data

```
d0 = s(1,:);
x0 = rx;

h1 = x0*d0.'/norm(x0*d0.');
d1 = h1'*x0;
x1 = x0 - h1*d1;

h2 = x1*d1.'/norm(x1*d1.');
dot(h1,h2)
d2 = h2'*x1;
x2 = x1 - h2*d2;
```

```
h3 = x2*d2.'/norm(x2*d2.');
dot(h1,h3)
dot(h2,h3)
d3 = h3'*x2;
x3 = x2 - h3*d3;

h4 = x3*d3.'/norm(x3*d3.');
dot(h2,h4)
d4 = h4'*x3;
x4 = x3 - h4*d4;

h5 = x4*d4.'/norm(x4*d4.');
d5 = h5'*x4;
x5 = x4 - h5*d5;

h6 = x5*d5.'/norm(x5*d5.');
d6 = h6'*x5;
x6 = x5 - h6*d6;

h7 = x6*d6.'/norm(x6*d6.');
d7 = h7'*x6;
x7 = x6 - h7*d7;

h8 = x7*d7.'/norm(x7*d7.');
d8 = h8'*x7;
x8 = x7 - h8*d8;
```

MUSIC without Spatial Smoothing

```
T = [h1 h2 h3 h4 h5 h6 h7 h8];
Vn = T(:,P+1:M);

for theta=-90:90
    for phi=-90:90
        k=[0:M-1]';
        AA=exp(-1i*pi*cos(theta*pi/180-2*pi*k/8)*sin(phi*pi/180));
        WW=AA'*Vn*Vn'*AA;
        Kmusic(theta+91, phi+91)=(abs((AA'*AA)./WW));
    end
end
```

```
figure
theta=-90:90;
phi=-90:90;
mesh(theta,phi,Kmusic);
title('MUSIC with Multi Stage Wiener Filtering');
xlabel('Azimuth');
ylabel('Elevation');
zlabel('Music');
```

MUSIC with Spatial Smoothing

```
d0 = [s(1,:) s(1,:)];
x0 = X;

h1 = x0*d0.'/norm(x0*d0.');
d1 = h1'*x0;
x1 = x0 - h1*d1;

h2 = x1*d1.'/norm(x1*d1.');
dot(h1,h2)
d2 = h2'*x1;
x2 = x1 - h2*d2;

h3 = x2*d2.'/norm(x2*d2.');
dot(h1,h3)
dot(h2,h3)
d3 = h3'*x2;
x3 = x2 - h3*d3;

h4 = x3*d3.'/norm(x3*d3.');
dot(h2,h4)
d4 = h4'*x3;
x4 = x3 - h4*d4;

h5 = x4*d4.'/norm(x4*d4.');
d5 = h5'*x4;
x5 = x4 - h5*d5;

h6 = x5*d5.'/norm(x5*d5.');
d6 = h6'*x5;
```

```
x6 = x5 - h6*d6;

h7 = x6*d6.'/norm(x6*d6.');
d7 = h7'*x6;
x7 = x6 - h7*d7;

h8 = x7*d7.'/norm(x7*d7.');
d8 = h8'*x7;
x8 = x7 - h8*d8;

T = [h1 h2 h3 h4 h5 h6 h7];
Vn = T(:,P+1:Ml);

for theta=-90:90
    for phi=-90:90
        k=[0:Ml-1]';
        AA=exp(-1i*pi*cos(theta*pi/180-2*pi*k/8)*sin(phi*pi/180));
        WW=AA'*Vn*Vn'*AA;
        Pmusic(theta+91, phi+91)=(abs((AA'*AA)./WW));
    end
end
figure
theta=-90:90;
phi=-90:90;
mesh(theta,phi,Pmusic);
title('MUSIC with Multi Stage Wiener Filtering and spatial smoothing');
xlabel('Azimuth');
ylabel('Elevation');
zlabel('Music');
```

Appendix B

Python Code

If a file includes "This file is a part of GNU Radio," the following GNU General Public License applies:

Copyright Free Software Foundation, Inc.

GNU Radio is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3, or (at your option) any later version.

GNU Radio is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with GNU Radio; see the file COPYING. If not, write to the Free Software Foundation, Inc., 51 Franklin Street, Boston, MA 02110-1301, USA.

B.1 benchmark_rx.py

```
#!/usr/bin/env python
#
# This file is part of GNU Radio
#

from gnuradio import gr, gru, modulation_utils
from gnuradio import usrp
```

```
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

import random
import struct
import sys

# from current dir
import usrp_receive_path

#import os
#print os.getpid()
#raw_input('Attach and press enter: ')

class my_top_block(gr.top_block):
    def __init__(self, demodulator, rx_callback, options):
        gr.top_block.__init__(self)

        # Set up receive path
        self.rxpath = usrp_receive_path.usrp_receive_path(
demodulator, rx_callback, options)

        self.connect(self.rxpath)

# ////////////////////////////////////////////////////////////////////////
#                                     main
# ////////////////////////////////////////////////////////////////////////

global n_rcvd, n_right

def main():
    global n_rcvd, n_right

    n_rcvd = 0
    n_right = 0

    def rx_callback(ok, payload):
        global n_rcvd, n_right
        (pktno,) = struct.unpack('!H', payload[0:2])
```

```
        n_rcvd += 1
        if ok:
            n_right += 1

        print "ok = %5s  pktno = %4d  n_rcvd = %4d  n_right = %4d" % (
            ok, pktno, n_rcvd, n_right)


    demods = modulation_utils.type_1_demods()

    # Create Options Parser:
    parser = OptionParser (option_class=eng_option, conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")

    parser.add_option("-m", "--modulation", type="choice",
choices=demods.keys(), default='dqpsk',
  help="Select modulation from: %s [default=%%default]"
                           % (', '.join(demods.keys()),))

    usrp_receive_path.add_options(parser, expert_grp)

    for mod in demods.values():
        mod.add_options(expert_grp)

    (options, args) = parser.parse_args ()

    if len(args) != 0:
        parser.print_help(sys.stderr)
        sys.exit(1)

    if options.rx_freq is None:
        sys.stderr.write("You must specify -f FREQ or --freq FREQ\n")
        parser.print_help(sys.stderr)
        sys.exit(1)


    # build the graph
    tb = my_top_block(demods[options.modulation], rx_callback, options)

    r = gr.enable_realtime_scheduling()
```

```
    if r != gr.RT_OK:
        print "Warning: Failed to enable realtime scheduling."

    tb.start()        # start flow graph
    tb.wait()         # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

B.2 benchmark_tx.py

```
#!/usr/bin/env python
#
# This file is part of GNU Radio
#

from gnuradio import gr, gru, modulation_utils
from gnuradio import usrp
from gnuradio import eng_notation
from gnuradio.eng_option import eng_option
from optparse import OptionParser

import random, time, struct, sys

# from current dir
import usrp_transmit_path

#import os
#print os.getpid()
#raw_input('Attach and press enter')

class my_top_block(gr.top_block):
    def __init__(self, modulator, options):
        gr.top_block.__init__(self)
```

```
        self.txpath = usrp_transmit_path.usrp_transmit_path(modulator, options)

        self.connect(self.txpath)

# /////////////////////////////////////////////////////////////////////////////
#                                   main
# /////////////////////////////////////////////////////////////////////////////

def main():

    def send_pkt(payload='', eof=False):
        return tb.txpath.send_pkt(payload, eof)

    def rx_callback(ok, payload):
        print "ok = %r, payload = '%s'" % (ok, payload)

    mods = modulation_utils.type_1_mods()

    parser = OptionParser(option_class=eng_option, conflict_handler="resolve")
    expert_grp = parser.add_option_group("Expert")

    parser.add_option("-m", "--modulation", type="choice", choices=mods.keys(),
                      default='dqpsk',
                      help="Select modulation from: %s [default=%%default]"
                            % (', '.join(mods.keys()),))

    parser.add_option("-s", "--size", type="eng_float", default=1500,
                      help="set packet size [default=%default]")
    parser.add_option("-M", "--megabytes", type="eng_float", default=1.0,
                      help="set megabytes to transmit [default=%default]")
    parser.add_option("","--discontinuous", action="store_true", default=False,
                      help="enable discontinous transmission (bursts of 5 pkts)")
    parser.add_option("","--from-file", default=None,
                      help="use file for packet contents")

    usrp_transmit_path.add_options(parser, expert_grp)

    for mod in mods.values():
        mod.add_options(expert_grp)
```

```
(options, args) = parser.parse_args ()

if len(args) != 0:
    parser.print_help()
    sys.exit(1)

if options.tx_freq is None:
    sys.stderr.write("You must specify -f FREQ or --freq FREQ\n")
    parser.print_help(sys.stderr)
    sys.exit(1)

if options.from_file is not None:
    source_file = open(options.from_file, 'r')

# build the graph
tb = my_top_block(mods[options.modulation], options)

r = gr.enable_realtime_scheduling()
if r != gr.RT_OK:
    print "Warning: failed to enable realtime scheduling"

tb.start()                      # start flow graph

# generate and send packets
nbytes = int(1e6 * options.megabytes)
n = 0
pktno = 0
pkt_size = int(options.size)

while n < nbytes:
    if options.from_file is None:
        data = (pkt_size - 2) * chr(pktno & 0xff)
    else:
        data = source_file.read(pkt_size - 2)
        if data == '':
            break;

    payload = struct.pack('!H', pktno & 0xffff) + data
    send_pkt(payload)
    n += len(payload)
```

```
        sys.stderr.write('.')
        if options.discontinuous and pktno % 5 == 4:
            time.sleep(1)
        pktno += 1

    send_pkt(eof=True)

    tb.wait()                       # wait for it to finish

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
```

B.3 generic_usrp.py

Unmodified from GNU Radio 3.2.2 Release

B.4 modify_dat.py

```
#!/usr/bin/env python

import scipy as sc
from scipy.io.numpyio import fread, fwrite
from scipy import array

from gnuradio import gr, gru, eng_notation, optfir
from gnuradio import audio
from gnuradio import usrp
from gnuradio import blks2
from gnuradio.eng_option import eng_option
from gnuradio.wxgui import slider, powermate
from gnuradio.wxgui import stdgui2, fftsink2, form
from optparse import OptionParser
from usrpm import usrp_dbid
import sys
```

```
import math
import wx

#use GNU Radio to filter and demod
class my_graph(gr.top_block):

def __init__(self):
gr.top_block.__init__(self)

#deifne paramerters
M, Nb, P = 12, 12160, 3
angdeg = array([37, 144, 109])
angles = angdeg*(sc.pi/180)
powers = array([10, 5, 0.5])
dlambda = 0.5
X = sc.zeros((M,Nb))

#open and read all .dat files for P emitters
f = open('dqpsk0.dat', 'rb')
if (f < 0):
v = 0
else:
t = fread(f, 2*Nb,'f')
t = sc.reshape(t,(Nb,2))
f0 = t[:,0] + t[:,1]*1j
f = open('dqpsk1.dat', 'rb')
if (f < 0):
v = 0
else:
t = fread(f, 2*Nb,'f')
t = sc.reshape(t,(Nb,2))
f1 = t[:,0] + t[:,1]*1j
f = open('dqpsk2.dat', 'rb')
if (f < 0):
v = 0
else:
t = fread(f, 2*Nb,'f')
t = sc.reshape(t,(Nb,2))
f2 = t[:,0] + t[:,1]*1j
```

```
b = [f0,f1,f2]

#modify data for linear array of M antennae
for k in range(P): #0,1,2
mu = sc.pi*sc.cos(angles[k])
a = sc.exp(1j*mu*sc.arange(M))[:, sc.newaxis]
X = X+powers[k]*a*b[k]

#add some noise
X = X+0.1*(sc.randn(M,Nb)+1j*sc.randn(M,Nb))

#write .dat files for each of M antenna observed data
for i in range(M):
f = open('mod%d.dat'%(i), 'wb')
fwrite(f, X[i,:].size, X[i,:])
f.close

exit()

def main ():
    my_graph().run()
if __name__ == '__main__':
main ()
```

B.5 pick_bitrate.py

Unmodified from GNU Radio 3.2.2 Release

B.6 receive_path.py

Unmodified from GNU Radio 3.2.2 Release

B.7 test_esprit_mswf.py

```
#!/usr/bin/env python
```

```
from scipy.io.numpyio import fread
from scipy import linalg
from scipy import *
import matplotlib.pyplot as plt

#define parameters
M, Nb, P = 8, 12160, 3
angdeg = array([37, 144, 109])
poles = zeros((1,P))
angles = angdeg*(pi/180)
powers = array([10, 5, 0.5])
dlambda = 0.5

#import M-antenna data streams
X = zeros((M,Nb)) + zeros((M,Nb))*1j
for k in range(M):
f = open('demod%d.dat' % (k), 'rb')
t = fread(f, Nb*2,'f')
t = reshape(t,(Nb,2))
f2 = t[:,0] + t[:,1]*1j
X[k] = f2

# Smoothing Matrix
Ml = 5  #sub-array size
L = M-Ml+1
l=1
J1 = hstack([zeros((Ml,l-1)), eye(Ml,Ml), zeros((Ml,M-l-Ml+1))])
l=2
J2 = hstack([zeros((Ml,l-1)), eye(Ml,Ml), zeros((Ml,M-l-Ml+1))])
l=3
J3 = hstack([zeros((Ml,l-1)), eye(Ml,Ml), zeros((Ml,M-l-Ml+1))])
l=4
J4 = hstack([zeros((Ml,l-1)), eye(Ml,Ml), zeros((Ml,M-l-Ml+1))])
Xs = hstack([dot(J1,X), dot(J2,X), dot(J3,X), dot(J4,X)])

#new method, Multi-Stage Weiner Filters
d0 = hstack([ X[0,:], X[0,:], X[0,:], X[0,:] ])
d0 = d0[:,newaxis]
x0 = Xs
```

```
h1 = dot(x0,d0)/linalg.norm(dot(x0,d0))
d1 = dot(h1.conj().T, x0)
x1 = x0 - dot(h1, d1)

h2 = dot(x1, d1.T)/linalg.norm(dot(x1, d1.T))
d2 = dot(h2.conj().T, x1)
x2 = x1 - dot(h2, d2)

h3 = dot(x2, d2.T)/linalg.norm(dot(x2, d2.T))
d3 = dot(h3.conj().T, x2)
x3 = x2 - dot(h3, d3)

h4 = dot(x3, d3.T)/linalg.norm(dot(x3, d3.T))
d4 = dot(h4.conj().T, x3)
x4 = x3 - dot(h4, d4)

h5 = dot(x4, d4.T)/linalg.norm(dot(x4, d4.T))
d5 = dot(h5.conj().T, x4)
x5 = x4 - dot(h5, d5)

T = hstack([h1, h2, h3, h4, h5])

#ESPRIT algorithm:
Es=T[:,0:P] #all, 0 to P-1
Es1=Es[0:Ml-1,:] #0 to M-2, all
Es2=Es[1:Ml,:] #1 to M-1, all
Psi = linalg.lstsq(Es1,Es2) #Psi = Es1\Es2
Phivec, T = linalg.eig(Psi[0])
angests = arccos(angle(Phivec)/pi)*(180/pi)

#plot angles from ESPRIT and compare with true angles
plt.figure(1)
plt.hold(True)
plt.polar(angles,angles/angles,'kx',ms=12)#,'markersize',12,'linewidth',1)
plt.polar(angests/(180/pi),abs(Phivec),'bo',ms=6)#,
'markersize',12,'linewidth',1)

print "angests=\n",angests
print "angdeg=\n", angdeg
```

```
plt.figure(2)
T = hstack([h1, h2, h3, h4, h5])
Pn = dot(T[:,P:Ml],T[:,P:Ml].conj().T)

#compute MUSIC spatial spectrum using all noise eigenvectors
ell=0
SMinVar = zeros(len(arange(0,180,.1)))
SMUSIC = zeros(len(arange(0,180,.1)))
for theta in arange(0,180,.1):
atheta = exp(1j*pi*cos(theta*(pi/180))*r_[:Ml])
SMUSIC[ell] = -10*log10(real(dot(dot(conj(atheta),Pn),atheta.T)))
ell = ell+1

theta = arange(0,180,.1)
plt.plot(theta,(SMUSIC-amax(SMUSIC)),'r')
plt.axis([0, 180, -80, 0])
plt.hold(True)
plt.plot(theta,(SMinVar-amax(SMinVar)),'b')
for k in range(P):
plt.plot((angdeg[k], angdeg[k]), (-80, 0), 'm')

plt.legend(['MUSIC'])
plt.title('Spatial Spectrum Estimates')
plt.xlabel('Arrival Angle (Degrees)')
plt.ylabel('Spectral Magnitude (dB)')


# without Smoothing
d0 = hstack([ X[0,:]])
d0 = d0[:,newaxis]
x0 = X

h1 = dot(x0,d0)/linalg.norm(dot(x0,d0))
d1 = dot(h1.conj().T, x0)
x1 = x0 - dot(h1, d1)

h2 = dot(x1, d1.T)/linalg.norm(dot(x1, d1.T))
d2 = dot(h2.conj().T, x1)
x2 = x1 - dot(h2, d2)
```

```
h3 = dot(x2, d2.T)/linalg.norm(dot(x2, d2.T))
d3 = dot(h3.conj().T, x2)
x3 = x2 - dot(h3, d3)

h4 = dot(x3, d3.T)/linalg.norm(dot(x3, d3.T))
d4 = dot(h4.conj().T, x3)
x4 = x3 - dot(h4, d4)

h5 = dot(x4, d4.T)/linalg.norm(dot(x4, d4.T))
d5 = dot(h5.conj().T, x4)
x5 = x4 - dot(h5, d5)

h6 = dot(x5, d5.T)/linalg.norm(dot(x5, d5.T))
d6 = dot(h6.conj().T, x5)
x6 = x5 - dot(h6, d6)

h7 = dot(x6, d6.T)/linalg.norm(dot(x6, d6.T))
d7 = dot(h7.conj().T, x6)
x7 = x6 - dot(h7, d7)

h8 = dot(x7, d7.T)/linalg.norm(dot(x7, d7.T))
d8 = dot(h8.conj().T, x7)
x8 = x7 - dot(h8, d8)

T = hstack([h1, h2, h3, h4, h5, h6, h7, h8])

#ESPRIT algorithm:
Es=T[:,0:P] #all, 0 to P-1
Es1=Es[0:M-1,:] #0 to M-2, all
Es2=Es[1:M,:] #1 to M-1, all
Psi = linalg.lstsq(Es1,Es2) #Psi = Es1\Es2
Phivec, T = linalg.eig(Psi[0])
angests = arccos(angle(Phivec)/pi)*(180/pi)

#plot angles from ESPRIT and compare with true angles
plt.figure(1)
plt.polar(angests/(180/pi),abs(Phivec),'ro',ms=6)#,
'markersize',12,'linewidth',1)
plt.hold(False)
plt.title('Angle of Arrival')
```

```
plt.legend(['True', 'ESPRIT','ESPRIT With Smoothing'],4)

print "angests=\n",angests
print "angdeg=\n", angdeg

plt.figure(2)
T = hstack([h1, h2, h3, h4, h5, h6, h7, h8])
Pn = dot(T[:,P:M],T[:,P:M].conj().T)

#compute MUSIC spatial spectrum using all noise eigenvectors
ell=0
SMUSIC = zeros(len(arange(0,180,.1)))
for theta in arange(0,180,.1):
atheta = exp(1j*pi*cos(theta*(pi/180))*r_[:M])
SMUSIC[ell] = -10*log10(real(dot(dot(conj(atheta),Pn),atheta.T)))
ell = ell+1

#plot 1D MUSIC with and without smoothing
theta = arange(0,180,.1)
plt.plot(theta,(SMUSIC-amax(SMUSIC)),'b')
plt.axis([0, 180, -80, 0])
for k in range(P):
plt.plot((angdeg[k], angdeg[k]), (-80, 0), 'm')

plt.legend(['Smoothing','No Smoothing'],4)
plt.title('MUSIC Spatial Spectrum Estimates Using MSWF')
plt.xlabel('Arrival Angle (Degrees)')
plt.ylabel('Spectral Magnitude (dB)')
plt.hold(False)
plt.show()
```

B.8 transmit_path.py

Unmodified from GNU Radio 3.2.2 Release

B.9 usrp_options.py

Unmodified from GNU Radio 3.2.2 Release

B.10 usrp_receive_path.py

```
#!/usr/bin/env python
#
# This file is part of GNU Radio
#

from gnuradio import gr
import usrp_options
import receive_path
from pick_bitrate import pick_rx_bitrate
from gnuradio import eng_notation

import scipy as sc
from scipy.io.numpyio import fread
from scipy import array

def add_freq_option(parser):
    """
    Hackery that has the -f / --freq option set both tx_freq and rx_freq
    """
    def freq_callback(option, opt_str, value, parser):
        parser.values.rx_freq = value
        parser.values.tx_freq = value

    if not parser.has_option('--freq'):
        parser.add_option('-f', '--freq', type="eng_float",
                          action="callback", callback=freq_callback,
                          help="set Tx and/or Rx frequency to FREQ",
                          metavar="FREQ")

def add_options(parser, expert):
    add_freq_option(parser)
    usrp_options.add_rx_options(parser)
    receive_path.receive_path.add_options(parser, expert)
```

```
    expert.add_option("", "--rx-freq", type="eng_float", default=None,
                          help="set Rx frequency to FREQ, metavar="FREQ")
    parser.add_option("-v", "--verbose", action="store_true", default=False)

class usrp_receive_path(gr.hier_block2):

    def __init__(self, demod_class, rx_callback, options):
        '''
        See below for what options should hold
        '''
        gr.hier_block2.__init__(self, "usrp_receive_path",
                gr.io_signature(0, 0, 0), # Input signature
                gr.io_signature(0, 0, 0)) # Output signature
        if options.rx_freq is None:
            sys.stderr.write("-f FREQ or --freq FREQ or
--rx-freq FREQ must be specified\n")
            raise SystemExit
        rx_path = receive_path.receive_path(demod_class, rx_callback, options)

        for attr in dir(rx_path): #forward the methods
            if not attr.startswith('_') and not hasattr(self, attr):
                setattr(self, attr, getattr(rx_path, attr))

#define parameters
        M, Nb, P = 12, 12160, 3
        angdeg = array([37, 144, 109])
        angles = angdeg*(sc.pi/180)
        powers = array([10, 5, 0.5])
        dlambda = 0.5
        X = sc.zeros((M,Nb))

        #open and read all .dat files
        f = open('dqpsk0.dat', 'rb')
        if (f < 0):
        v = 0
        else:
        t = fread(f, 2*Nb,'f')
        t = sc.reshape(t,(Nb,2))
        f0 = t[:,0] + t[:,1]*1j
        f = open('dqpsk1.dat', 'rb')
```

```
    if (f < 0):
    v = 0
    else:
    t = fread(f, 2*Nb,'f')
    t = sc.reshape(t,(Nb,2))
    f1 = t[:,0] + t[:,1]*1j
    f = open('dqpsk2.dat', 'rb')
    if (f < 0):
    v = 0
    else:
    t = fread(f, 2*Nb,'f')
    t = sc.reshape(t,(Nb,2))
    f2 = t[:,0] + t[:,1]*1j

    b = [f0,f1,f2]

    #modify data for linear array
    for k in range(P): #0,1,2
    mu = sc.pi*sc.cos(angles[k])
    a = sc.exp(1j*mu*sc.arange(M))[:, sc.newaxis] #.' = .T, ' = .H
    X = X+powers[k]*a*b[k]

    #add some noise
    X = X+0.1*(sc.randn(M,Nb)+1j*sc.randn(M,Nb))

    src = gr.vector_source_c(X[11])

    #setup usrp
    self._demod_class = demod_class

    #connect
    self.connect(src, rx_path)

def _setup_usrp_source(self, options):
    if options.verbose:
        print 'USRP Source:', self.u
    (self._bitrate, self._samples_per_symbol, self._decim) = \
    pick_rx_bitrate(options.bitrate, self._demod_class.bits_per_symbol(), \
    options.samples_per_symbol, options.decim, adc_rate,  \
    self.u.get_decim_rates())
```

```
        self.u.set_decim(self._decim)

        if not self.u.set_center_freq(options.rx_freq):
            print "Failed to set Rx frequency to %s" % (
eng_notation.num_to_str(options.rx_freq))
            raise ValueError, eng_notation.num_to_str(options.rx_freq)
```

B.11 usrp_transmit_path.py

```
#!/usr/bin/env python
#
# This file is part of GNU Radio
#

from gnuradio import gr
import usrp_options
import transmit_path
from pick_bitrate import pick_tx_bitrate
from gnuradio import eng_notation

def add_freq_option(parser):
    """
    Hackery that has the -f / --freq option set both tx_freq and rx_freq
    """
    def freq_callback(option, opt_str, value, parser):
        parser.values.rx_freq = value
        parser.values.tx_freq = value

    if not parser.has_option('--freq'):
        parser.add_option('-f', '--freq', type="eng_float",
                action="callback", callback=freq_callback,
                help="set Tx and/or Rx frequency to FREQ [default=%default]",
                metavar="FREQ")

def add_options(parser, expert):
    add_freq_option(parser)
    usrp_options.add_tx_options(parser)
```

```
    transmit_path.transmit_path.add_options(parser, expert)
    expert.add_option("", "--tx-freq", type="eng_float", default=None,
                help="set transmit frequency to FREQ, metavar="FREQ")
    parser.add_option("-v", "--verbose", action="store_true", default=False)

class usrp_transmit_path(gr.hier_block2):
    def __init__(self, modulator_class, options):
        '''
        See below for what options should hold
        '''
        gr.hier_block2.__init__(self, "usrp_transmit_path",
                gr.io_signature(0, 0, 0), # Input signature
                gr.io_signature(0, 0, 0)) # Output signature
        if options.tx_freq is None:
            sys.stderr.write("-f FREQ or --freq FREQ or
--tx-freq FREQ must be specified\n")
            raise SystemExit
        tx_path = transmit_path.transmit_path(modulator_class, options)
        for attr in dir(tx_path): #forward the methods
            if not attr.startswith('_') and not hasattr(self, attr):
                setattr(self, attr, getattr(tx_path, attr))
        #setup file sink
        self._modulator_class = modulator_class
        self.file_sink = gr.file_sink(gr.sizeof_gr_complex, "dqpsk2.dat")
        #connect
        self.connect(tx_path, self.file_sink)

    def _setup_usrp_sink(self, options):
        """
        Creates a USRP sink, determines the settings for best bitrate,
        and attaches to the transmitter's subdevice.
        """
        #self.u = usrp_options.create_usrp_sink(options)
        #dac_rate = self.u.dac_rate()
        if options.verbose:
            print 'USRP Sink:', self.u
        (self._bitrate, self._samples_per_symbol, self._interp) =
        pick_tx_bitrate(options.bitrate, self._modulator_class.bits_per_symbol(),
        options.samples_per_symbol, options.interp, dac_rate,
        self.u.get_interp_rates())
```

```
        self.u.set_interp(self._interp)
        self.u.set_auto_tr(True)

        if not self.u.set_center_freq(options.tx_freq):
            print "Failed to set Rx frequency to %s" % (
eng_notation.num_to_str(options.tx_freq))
            raise ValueError, eng_notation.num_to_str(options.tx_freq)
```

Bibliography

[1] Roy, R. and T. Kailath (1989) "ESPRIT-Estimation of Signal Parameters via Rotational Invariance Techniques," *IEEE Transactions on Acoustics Speech Signal Processing*, **37**(7), pp. 984–995.

[2] Pahlavan, K., X. Li, M. Yilanttila, R. Chana, and M. Latva-aho (2000) *An Overview of Wireless Indoor Geolocation Techniques and Systems*, *Tech. rep.*, Worcester Polytechnic Institute.

[3] (2008) *The Technologies behind a Context-Aware Mobility Solution*, *Tech. rep.*, Cisco Systems, Inc.

[4] Azarmanesh, O., P. Desale, C. Gardner, A. Banik, and T. Nagarmat (2010) *Report for Smart Radio Challenge 2009*, *Tech. rep.*, The Pennsylvania State University.

[5] Ricks, D. and J. Goldstein (2000) "Efficient Implementation of Multi-Stage Adaptive Wiener Filters," *Antenna Applications Symposium*.

[6] Wireless Innovation Forum (2009) *Smart Radio Challenge Problem*.

[7] Pauraj, A., R. Roy, and T. Kailath (1986) "A Subspace Rotation Approach to Signal Parameter Estimation," *IEEE*, **74**, pp. 1044–1046.

[8] Schmidt, R. (1986) "Multiple Emitter Location and Signal Parameter Estimation," *IEEE Transactions on Antennas and Propagation*, **AP-34**, pp. 276–280.

[9] Goldstein, J., I. Reed, and L. Scharf (1998) "A Multistage Representation of the Wiener Filter Based on Orthogonal Projections," *IEEE Transactions on Information Theory*, **44**(7), pp. 2943–2959.

[10] Chestnut, P. C. (1982) "Emitter Location Accuracy Using TDOA and Differential Doppler," *IEEE Transactions on Aerospace and Electronic Systems*, **AES-18**(2).

[11] Huang, L., S. Wu, and L. Zhang (2005) "Low Complexity ESPRIT Method for Direction Finding," *IEEE Transactions on Acoustics Speech Signal Processing*, **4**, pp. 929–932.

[12] ——— (2005) "A Novel MUSIC Algorithm for Direction-of-Arrival Estimation Without the Estimate Covariance Matrix and its Eigendecomposition," *IEEE Vehicular Technology Conference*.

Vita

CHRISTOPHER B. GARDNER

EDUCATION: Bachelor of Science in Electrical Engineering — Expected: 5/2010
The Pennsylvania State University, University Park, PA
Schreyer Honors College
GPA: 3.80/4.00
Dean's List (7/7 semesters)
Relevant Courses:

FPGA Design	Software-Defined Radios
Digital Design, Microcontrollers	Systems Engineering Seminar
Probability and Stochastic Processes	Effective Speech

THESIS: **Software-Defined Radio Techniques for Geolocation of First-Responder Transceivers**
Thesis Supervisor: Dr. Sven G. Bilén
Honors Adviser: Dr. John D. Mitchell

RELATED EXPERIENCE:

DRS Signal Solutions, Inc., Gaithersburg, MD — 5/2009 – 8/2009
Intern—Tactical Applications R&D
- Designed high-speed digital Test Fixture for wideband (20 MHz – 6 GHz) upconverter.
- Developed Xilinx CoolRunner2 CPLD core architecture in VHDL for Test Fixture.
- Specified design routing requirements for PCB Group to place and route board.

Bose Corporation, Framingham, MA — 5/2008 – 8/2008
Intern—Home Entertainment, Advanced Development
- Prototyped and debugged an "Impedance-Controlled Impedance" for amplifier safety testing.
- "ICI" models a transducer's complex, instantaneous impedance while reducing SPL by 40dB.
- "ICI" will increase productivity by eliminating the need to use sound-damping rooms for testing.

Adaptive Methods, Inc., Rockville, MD — 5/2007 – 8/2007
Intern—Sensors Integration Group
- Built a sea-worthy RS-232 to RS-422 signal converter for a SURTASS stabilizer.
- Researched for and helped create two SBIR proposals—one of which was awarded.
- Improved a sensor data gathering program by adding remote networking support.

National Security Agency — Summer 2005, 2006
Intern—Microelectronics Reverse Engineering
- Reverse-engineered obsolete ICs to incorporate their functions in newer technology.
- Hold a **Top Secret/SCI clearance** since June 2005, re-cleared in 2008.
- Had an Agency Special Background Investigation and polygraph.

Student Space Programs Lab, University Park, PA
PCB Designer—Hybrid Plasma Probe Team (OSIRIS CubeSat) — 8/2007 – 4/2008
- Designed and tested a direct digital synthesizer for a swept bias Langmuir Probe experiment.

PCB Designer—CanSAT Team — 11/2006 – 5/2007
- Designed a solenoid controller in Orcad Capture and populated components by hand.

COMPUTER SKILLS:
- Application Programs (MATLAB, MultiSim, Mentor Graphics DxDesigner, Cadence Orcad, FEKO)
- Languages (VHDL, Embedded C/Basic, C, C++)
- Operating Systems (Windows, UNIX/Linux)

HONORS:
- James M. Barnak Outstanding Junior Award – Eta Kappa Nu Association
- Matthew P. Blickley Memorial Scholarship for International Studies
- Lockheed Martin Scholarship Award Winner, 2008/2009
- College of Engineering Scholarship
- Schreyer Honors College Scholarship
- Washington, D.C. Alumni Association Scholarship

ACTIVITIES:
- Teaching Intern for EE 310 – Electronic Circuit Design
- Member of Electrical Engineering Student Advisory Committee
- Schreyer Honors College Student Council