A Channel Model and Geolocation Simulation System for Cooperative Spectrum Sensing Networks

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements
for the degree of Bachelor of Science

Devin Kelly and Ishrak Khair

March 16, 2010

Prof. Alexander M. Wyglinski
Advisor

This report represents work of WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review. For more information about the projects program at WPI, see http://www.wpi.edu/Academics/Projects.
Abstract

This report documents the design of a cooperative spectrum sensing network for emergency response applications. The network discerns radio modulation, center frequency, and the geographic coordinates of each emergency responder. The simulation is designed to emulate the wireless propagation characteristics of a real disaster environment by creating channel models. The project culminates with a graphical user interface that interactively displays system capabilities.
Executive Summary

This report describes the work done in this MQP to develop a software define radio simulation system in MATLAB using cooperative sensor networks that senses and manages the communication spectrum, performing wireless tracking of transmitters in conjunction. The system is developed in the context of emergency response scenarios. In the event that multiple emergency response teams using different communication protocols work together, interoperability of radios becomes an issue. The medium of radio transmission can become quite congested as emergency personnel attempt to make radio transmissions at the same time due to a lack of coordination of these transmissions. In addition, it is not always possible to know where each emergency responder is geographically during emergency response missions. Teams may become scattered and members may get lost or even injured without the rest of the team being aware of this. This project attempts to address these issues by developing a system that manages the radio frequency spectrum and performs wireless location tracking so that responder’s locations are known and the radio frequency traffic is managed efficiently.

The system being developed is based on software-defined-radio technology. Software-defined-radio, or SDR in short, is essentially a radio with most of it’s processing capabilities centered in its software core that is running on a dedicated processor in the radio. This is in contrast with today’s conventional radios that are primarily comprised of specialized hardware that perform the necessary processing tasks. Any change in the foundation of the system or protocols, would render conventional radios ineffective due to their static nature. As a result, different teams with conventional radios have difficulty communicating with each other wirelessly. Software-defined-radio systems overcome this challenge as they can be reconfigured easily to change the processing protocols as they are implemented in software. This MQP simulates a software-defined-radio based communication network that overcomes the difficulties of interoperability and wireless tracking.

Events such as the terrorist attacks that occurred on September 11 2001 and Hurricane Katrina on August 23 2005 have caused countless lives of innocent civilians and emergency responders to be lost. One of the main impediments to efficient rescue operations during
these events was the difficulty of radio communications. There is very little coordination between teams in such situations due to lack of interoperability. Cooperation between teams of responders is done effectively only when proper communication is established. Many lives could have been saved during tragedies if communications were managed effectively, which would have led to proper coordination of a large scale emergency response. Software-defined-radio technology shows a lot of promise on this front, as it can facilitate interoperability. The Software-Defined-Radio Forum holds annual competitions for student teams from Universities all across the globe to develop software-defined-radio systems that can address challenges of interoperability, radio channel management and wireless location tracking. This MQP is a competitor in the SDR Forum’s Smart Radio Challenge 2009-2010 to build and demonstrate such a system. The radio challenge is sponsored by the MathWorks Inc. They have generously provided us with MATLAB and Simulink softwares and accompanying toolboxes necessary as support for the project.

The report begins by describing the motivation for this project and problem statement. A technical foundation is then laid for the reader so that concepts and terms used in this report can be understood. The basic foundation presented consists of channel modelling theory, with explanations of various channel models and how they are used to characterize a signals passage through the respective environments. The three main channel models explained are rural environment, suburban environment and urban environment. Concepts of noise, Doppler shift and multipath fading are also introduced with respect to the aforementioned channels. Theoretical background necessary for understanding the main techniques of wireless tracking using RF signals is then provided. This background specifically explains signal-flight-time or time-of-arrival based localization techniques and received signal strength based techniques, which are based on the channel models that have been described. The VITA 49 Radio Transport Standard[1] is a data encapsulation standard that is developed specifically for software-defined-radio systems. This standard specifies an effective data format protocol that enables software-radio specific and user specific data to be fused with voice or text data so that communications can be managed more effectively. The report describes this protocol in detail and how it is leveraged in the system developed.

The design of the system is presented in terms of the different aspects of the design, namely the network design aspect, geolocation system aspect and user interface design aspect. The network design of the system comprises of signal detection and classification, channel modelling and VRT packet generation. The geolocation system design aspect includes algorithms that use TOA based location sensing, RSS based location sensing and hybrid location sensing techniques that use both TOA and RSS. All models are then evaluated individually and experimental data used for validation is provided. System simulation
design includes integration of individual system components and visualization by a graphical user interface. The graphical-user-interface is designed in MATLAB using the built-in GUI Development Environment, or GUIDE. The user-interface design requirements are first identified and then the graphical design of each component of the interface is described in detail. Integration of the system into the graphical-user-interface is described, including how the original MATLAB code needed to be modified specifically to work in an event-driven manner as required by the MATLAB graphical-user-interface environment.

The speed of our simulation system was improved by using the Embedded MATLAB toolbox that enabled us to pre-compile the complex signal detection algorithm. The report describes use of the Embedded MATLAB toolbox for this purpose in detail, including code modifications required for Embedded MATLAB compliance and subsequent C code generation that is compiled for speed. Challenges encountered in this process are described in detail, such as datatype conflicts between the interpreted MATLAB environment and the Embedded MATLAB compiled environment. These challenges were overcome by modifying data structures employed initially to Embedded MATLAB compliant data structures.

Hardware implementation of our communication system on the USRP2 platform is analysed. Challenges beyond the scope and time period of the MQP are presented in detail, and necessary steps that need to be taken in order to implement the system in hardware are presented. Challenges included the inability of the USRP2 platform to provide signal logistics such as time of arrival at the host system’s buffer not being available, and this information is needed to employ accurately the time-of-arrival based geolocation aspect of the system. In the case that the signal arrival time is facilitated by future updates of the hardware platform, a method to extract the time of arrival of the signal at the USRP2 is briefly discussed.

The report discusses the engineering design process in detail, with the provision of a Gantt chart indicating the timeline of the project and major milestones identified. Areas of research needed for developing a novel and effective cooperative wireless sensor network are discussed with respect to wireless location tracking, signal detection and classification and channel modelling techniques. Division of labor within the project members is discussed and a team strategy is presented for approaching the MQP. The team strategy included meetings for brainstorming, peer reviews of system level concepts and monthly code reviews.

The final sections of the report discuss in detail the results obtained from the simulations. Discussions and analysis of the results are presented, and the simulations are evaluated. The trend seen in the simulation results is that an increase in the number of sensors increases geolocation accuracy, while signal detection and classification performance remains stable. The simulation scenarios assume a 2000m by 2000m field of operation, where sensors and
and obstructions to line-of-sight are placed randomly. For a rural channel employing only five sensors yields an average geolocation error of about 37.22m with 64 out of 68 signals detected and classified correctly. Employing more sensors in the same rural channel provides improvements in the geolocation accuracy, with an average of 9.82m distance error with 30 sensors deployed. In a harsher environment of a suburban channel, deploying five sensors yields an average geolocation error of 50.45m and 34 out of 40 signals being detected and classified correctly. This is a large increase than the results seen for a rural channel. The geolocation algorithm accuracy improves with more sensors. With 30 sensors deployed in the suburban channel, we obtain an average geolocation error of 20.61m and 111 out of 120 signals being detected and classified correctly. In the harshest environment, the urban channel, we first simulate with five sensors and obtain an average geolocation error of 90.03m and 77 out of 84 signals being detected and classified correctly. As the number of sensors is increased in the urban environment to 30 sensors, the average geolocation error drops down to 34.89m with 91 out of 94 signals being detected and classified correctly. This geolocation accuracy and signal detection and classification rate shows that the algorithms are performing well with moderate complexity. Increasing the resolution of the algorithm parameters will yield higher accuracy. A detailed discussion of these results are provided in the report.

The accompanying MATLAB codes are provided in the Appendices at the end of the report.
Acknowledgements

We would like to thank the MathWorks Inc. for their generous support in providing us with the resources we need from USRPs, MATLAB and SIMULINK software to sharing their valuable time with us to provide feedback on how to use their tools more effectively.

Our project would not have been possible if it weren’t for Dr. Alexander M. Wyglinski’s continuous support throughout the entire MQP. Thank you for educating, inspiring and motivating us. We had started under your guidance, and now we look forward to exciting new paths as a result of your generous support.

We would like to thank Michael Calabro for all his support during and after the completion of his MQP. Our weekends were interesting indeed.

Finally, we would like to thank all the professors and staff members in the ECE Department at Worcester Polytechnic Institute for all their hard work that make everyone’s lives better everyday.
# Contents

1 Introduction ................................................................. 1
   1.1 Project Motivation .................................................... 1
   1.2 Problem Statement .................................................... 3
   1.3 History of the software-defined radio ............................ 4
   1.4 Project Objectives ................................................... 5
   1.5 Report Organization .................................................. 6

2 Background ................................................................. 8
   2.1 Fundamentals of Channel Modeling ................................. 8
      2.1.1 Noise ............................................................ 9
      2.1.2 Doppler Effects ............................................... 9
      2.1.3 Fading .......................................................... 10
      2.1.4 Characteristics of Environments ............................. 15
   2.2 Fundamentals of Geolocation ....................................... 17
      2.2.1 Received Signal Strength ................................. 17
      2.2.2 Time of Arrival ............................................. 18
   2.3 VITA 49 Radio Transport Standard ............................... 19
   2.4 Literature Review .................................................... 23
      2.4.1 Channel Models ............................................... 23
      2.4.2 Geolocation .................................................. 23
      2.4.3 VRT .......................................................... 24
   2.5 Chapter Summary .................................................... 24

3 Engineering Design Process ............................................ 25
   3.1 Research Areas ...................................................... 25
   3.2 Project Organization ................................................ 25
   3.3 Division of Labor .................................................... 26
   3.4 Meetings ............................................................. 27
List of Figures

1.1 A comparison of radio implementation in hardware versus software . . . . . . 6
2.1 A Channel Environment with Rician Fading ......................................... 14
2.2 A Channel Environment with Rayleigh Fading ...................................... 14
2.3 Simple Triangulation ........................................................................... 18
2.4 A VRT Information Stream .................................................................... 20
2.5 A Typical VRT Network Stack ............................................................... 20
2.6 Another Typical VRT Network Stack [1] ................................................ 21
2.7 VRT Data Packet Structure ................................................................... 21
2.8 VRT Meta-Data Packet Structure .......................................................... 22
4.1 Network Architecture ............................................................................ 30
4.2 Matched filter response with different values for $\beta$ ............................. 32
4.3 Signal With No Noise ........................................................................... 34
4.4 Signal With Noise Added ....................................................................... 35
4.5 Overall sensor activity ......................................................................... 39
4.6 Code Blocks for Simulation .................................................................... 42
4.7 Circles intersect at radical center .......................................................... 45
4.8 Location estimation using a large grid step size ..................................... 48
4.9 Location estimation using a smaller grid step size .................................. 49
4.10 Location estimation using a small number of sensors ........................... 50
4.11 Location estimation using large number of sensors (7 sensors) .............. 51
4.12 Error of estimates with only non line of sight impairments .................. 52
4.13 Error of estimates with non line of sight and multipath impairments ....... 53
4.14 Path loss exponents estimated correctly .............................................. 54
4.15 Overall Layout of GUI ......................................................................... 58
4.16 Final working GUI ............................................................................... 60
4.17 Visual Project Goals ............................................................................ 68
4.18 Overall Gantt Chart ............................................................................ 69
5.1 Simulation capture at time instant 1 for the rural channel ........................................ 75
5.2 Simulation capture at time instant 2 for the rural channel ........................................ 76
5.3 Simulation capture at time instant 3 for the rural channel ........................................ 77
5.4 Simulation capture at time instant 4 for the rural channel ........................................ 78
5.5 Simulation capture at time instant 5 for the rural channel ........................................ 79
5.6 Simulation capture at time instant 1 for the suburban channel ................................. 81
5.7 Simulation capture at time instant 4 for the suburban channel ................................. 82
5.8 Simulation capture at time instant 2 for the urban channel ....................................... 84
5.9 Simulation capture at time instant 4 for the urban channel ....................................... 85
5.10 Simulation capture at time instant 4 for the urban channel ...................................... 86
5.11 Simulation capture at time instant 5 for the urban channel ...................................... 87
Chapter 1

Introduction

1.1 Project Motivation

Software radio technology is very attractive when considering the benefits and ease of radio design and deployment. All digital signal processing can be done in software and all that is needed to be done in hardware are signal reception, filtering, amplification, downsampling, and digitizing the signal. This gives the user control to the communication system designers. In addition, any changes to communication protocol or signal processing can be implemented simply by reprogramming the FPGA or microcontroller core of the software radio.

Software radios are very flexible since most of the radio’s functional blocks are implemented in software. For example, modulation and demodulation algorithms can be updated, changed or replaced by newer algorithms without having to change any hardware. If symbol period of a wireless system needs to be changed, on a software radio this would equate to a simple code change or firmware update. Therefore, software radios are better than conventional radios in that they can be used in different configurations, where as conventional radios cannot without a change in hardware.

There are many challenges that first responders face when called into action in emergency situations. One of the largest challenges came during September 11th, 2001. According to the 9/11 commission report \[2\], there were many problems with the radio communications that first responders were using to coordinate the rescue efforts. These problems included:

- **Range of transmission** - The range of the radios did not cover first responder communications over large distances. A repeater system could have been used, but they were installed in the South Tower, which collapsed and resulted in impaired communications.

- **Channel Saturation** - Radio channels were being saturated as too many responders
would attempt to make radio transmissions at the same time. This means that if another responder were to attempt to transmit they would find the channel ”busy”.

- **Lack of radios** - Some responders did not have radios.

- **Lack of responder location tracking** - There was no effective way to know where individual responders were located. As a consequence, many responders were lost from their units.

A lack of radio interoperability has been problematic during other rescue efforts as well. During the Columbine High School massacre of April 20, 1999 radio interoperability was a problem for emergency responders. The emergency personnel response was massive: there were the local fire, police, and medical responders, but there was also 6 area sheriff departments, 20 other police departments, 46 ambulances, and 2 helicopters [3]. However, none of these departments could communicate with one another.

These problems are not limited to just a few select events. In fact, emergency responders in every disaster in recent history have had these problems. Emergency personnel who responded to Hurricane Katrina in New Orleans had similar experiences. However, the problem of radio interoperability became much larger during Hurricane Katrina, this is for several reasons. First, the disaster was more prolonged. Second, there were also many more parties involved. There were local responders, but also personnel from the military and the Federal Emergency Management Agency (FEMA). With all these groups, wireless inter-agency communications was not possible. These problems are only those seen directly by emergency responders. The Katrina Hurricane also had other problems unseen in other disasters.

As one may expect the problem during Katrina was much worse than in Columbine. A large reason for this was that many cell towers were blown down, or had their power knocked out. To make the situation worse, some landlines were destroyed [4]. These systems being down prevented not only emergency personnel from communicating with one another, but also stranded victims of the Hurricane. Radio interoperability during the Katrina disaster was a complete failure. What makes this failure so startling was that in 2003 - two years before Hurricane Katrina - the City of New Orleans received a $5.5 million grant specifically to address radio interoperability. Before the Katrina disaster only 5% of that money had been spent, [4] showing a lack of priority that made the disaster much worse two years later.

During these large disaster events, a chain of command must be established. This chain
of command integrates the rescue efforts of all the emergency responders. Without the chain of command emergency responders can be dispatched slowly or not at all. The key to establishing an effective chain of command is an equally effective communications system.

Software radios and cognitive radio networks could have improved communications and could have provided location information inside the building. A communication system designed for interoperable radios and cooperative use of the radio channel in a controlled manner would have been invaluable to ensuring the safety of people who ensure our own safety, the firefighters, police officers, paramedics, and other emergency responders. Our project is a step in this direction.

This project is the result of a collaborative effort between the MathWorks Inc. and WPI to develop and simulate software radio communication systems. Our MQP is also a competitor in this year’s SDR forum Radio Challenge of developing a MATLAB based cooperative communication system that provides geolocation information in a large scale disaster scenario [5].

1.2 Problem Statement

Disasters in the first decade of the twenty-first century exposed flaws with modern radio. This flaw is the lack of interoperability. There are numerous points in a radio design that break radio interoperability, these break points exist on both the hardware and the software side. As an increasing number of radios are implemented in software, the opportunity to make radios interoperable grows. This is because a software radio is a reconfigurable radio. When a radio can be reconfigured with either a software update or reconfigurable software, a disjoint group of radio users can potentially communicate with on another.

On September 11th, the lack of radio interoperability became apparent to all the emergency responders at the scene. Since the magnitude of this disaster was significant, emergency responders came to New York City from different boroughs, counties, and even states. Since there were so many different first responders from different places, it quickly became obvious that directly communicating between departments was impossible.

The problem of interoperability arose again in New Orleans during the disaster of Hurricane Katrina. This disaster highlights the need for interoperable radios even more. This is because the presence of emergency responders from Hurricane Katrina lasted even longer than September 11th. In New Orleans there was also a wider variety of radios after the
military arrived at the city. Hurricane Katrina showed that it is not nearly enough for emergency responders to be able to communicate with one another, but they must also be able to communicate with military personnel when the situation calls for it.

The challenge of radio interoperability becomes very critical in a large scale emergency response effort. Coordination of actions between different teams of responders and different responders is key to making an effective collaborative rescue effort. The first step towards achieving this is employing an interoperable wireless communication system that is used by all rescue teams. This is where our MQP comes into play. The challenge of this project is to develop a distributed cooperative spectrum sensing network. Specifically, a distributed set of sensors will work together to monitor wireless communications in a disaster environment. These sensors will listen across all the relevant parts of the electro-magnetic spectrum then record and interpret what they find. This means gathering the appropriate data:

- **Modulation Schemes** - How the data is encoded into waveforms.
- **Geographic Position** - The coordinates of the emergency responder.
- **Center Frequencies** - The carrier frequency the emergency responder’s radio.

With these three items, a central coordinator can listen to the data emergency responders transmit and keep track of where the emergency responders are located.

1.3 **History of the software-defined radio**

Software-defined radio has its roots in hardware radios. Hardware radios can be divided into three categories:

1. The analog radio
2. The digital radio
3. The software-defined radio

The first radios were all analog. The information they transmitted and received was all analog as well, meaning that these radios only operated on analog signals. Analog radios are implemented nearly entirely in hardware, that is, there is a minimal implementation in software. Digital radios encode information into streams of bits. The main advantages to the digital radio over the analog radio is that digital signals are more resilient to noise. The first generation of digital radios were also built all in hardware. The analog radio and the
early digital radio will be referred to as the traditional radio. All traditional radios have
on common characteristic: they are designed once, implemented, and cannot be changed
afterwards.

The traditional radio is built entirely in hardware, where once the radio is constructed, the
radio’s functionality cannot be changed. With a traditional radio, all the functionality must
be determined at design time, ranging from the desired frequency bands to the modulation
schemes. Since all these characteristics are permanent, the traditional radio cannot adapt to
new technologies, design errors cannot be changed, and interoperability becomes increasing
difficult. Recently, the digital radio has started to evolve to the software-defined radio.

As technology has evolved, a larger and larger portion of the digital radio has been
implemented in software. As one can see in Figure 1.1, now only the RF portions of software-
defined radios are implemented in hardware. Implementing a radio in software yields two
critical advantages:

1. The radios are more reconfigurable.

2. The radios easier to develop.

This makes the software-defined radio a very attractive platform. Using a software-defined
radio, one may update the radio software to get a device with completely different function-
ality without changing any hardware. Updating radio software has many other advantages
over hardware radios as well. For example, in a hardware radio, if an error is made the
hardware must be physically changed, which can be expensive and cumbersome. However,
with a software radio, all that is needed to fix a software error is an update.

Since the beginning of radio, these devices have proliferated. However, the devices have
always been specialized. These radios were built for a small set of functions and could not be
changed. With the advent of the software radio, the devices have become general purpose.

1.4 Project Objectives

This project has three main objectives:

1. *Creating accurate channel models* - The purpose of the channel models is to re-create
the impairments that a real wireless signal would be exposed to in software. The
challenge lies in matching the impairments created in simulation to those in the real
world. The channel models must take a given environment and produce the appropriate
 corresponding channel.
2. Creating a functional geolocation system - The aim of geolocation in this project is to establish and employ a methodology that locates personnel in a variety of channels, including harsh indoor environments. Once the location of the personnel are established, this information will be stored in a master database under control by the supernode or Master Control Station. This information can then be used to coordinate operations or convey location information to other teams.

3. Integrating all of these systems together - The preceding two systems must be integrated together, along with the cyclostationary detector, to form one functional simulation system. The integrated system will produce the data, add in impairments, apply the geolocation algorithms, pass the signals through the cyclostationary detector, and keep track of the system performance.

The first two systems were implemented independently, then integrated along the cyclostationary detector in the system integration stage.

1.5 Report Organization

This report proceeds with Chapter 2, where the background of technologies and technical concepts needed to understand the rest of the report. Next, is Chapter 3, this chapter discusses the specific aspects of the design chosen and explains what decisions were made and why. Then there is Chapter 4, this is the chapter that discusses how the design worked in simulation. Next, is Chapter 5, the concluding chapter of this report. This chapter discusses
how successful the project was, and the need and nature of any future work. Finally, there are the appendices where this project’s source code lies.
Chapter 2

Background

This chapter will go over the technical backgrounds of all the technical concepts necessary to understand the project design.

2.1 Fundamentals of Channel Modeling

The purpose of the channel model is to emulate the effects of a channel experienced by a real-life wireless signal. There are many effects, also called impairments, to consider. Not only must the presence of each impairment be considered, but also the magnitude of the impairment with respect to a given environment.

Impairments done by the channel to the transmitted signal are composed of:

1. Fading
2. Additive Noise
3. Doppler Effects

However, there are also several difference types of fading:

1. Very Slow Fading, usually a consequence of path loss
2. Slow Fading, usually due to shadowing
3. Fast Fading, a result of multipath

In this portion of the report, the techniques and mathematics that will be used for creating the channel models will be discussed. The most basic description of a channel is
a description of the fading characteristics. Although, there are more characteristics listed above, these three characteristics are the starting point to create any channel model. Each of these three fading characteristics varies from one environment to another, thus each must be accounted for in every environment.

2.1.1 Noise

All real signals are subjected to noise, such as thermal noise, shot noise, or black-body noise. All noise is a random additive phenomenon. That is, when one adds noise to a signal they are adding a random quantity of energy to the signal. Since noise is random, it can be described by its statistics.

For wireless applications, the type of noise that is most significant is thermal noise. Thermal noise is modeled by a white Gaussian random process. This type of noise is referred to as white because the noise power is constant for all frequencies and Gaussian becomes the random process follows a Gaussian distribution, which is listed in Equation (2.1). In this Equation $(f_x(x))$ represents the probability density function, $\sigma_x^2$ the variance of $x$, $\mu_x$ the mean of $x$.

$$f_x(x) = \frac{1}{\sqrt{2 \cdot \pi \sigma_x^2}} \cdot e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}} \quad (2.1)$$

For wireless applications, the mean is $\mu_x$, and the variance, $\sigma^2$ determines the noise power, which is denoted as $N_0$ and defined in Equation (2.2). For the rest of this paper, this is how noise power will be referred to.

$$N_0 = \frac{\sigma^2}{2} \quad (2.2)$$

2.1.2 Doppler Effects

A disaster environment with the proposed spectrum sensing network will have both stationary and mobile radios. That is, our sensing radios will be stationary, but the radios being sensed are mobile. Since these mobile radios are moving, their radiation will be subject to Doppler shifts.

The Doppler shift takes into affect when either a transmitting radio or receiving radio is moving with respect to other radio. For example, if the transmitting radio is in motion and the receiving radio is stationary, then the Doppler Shift will affect the received signal.
What the Doppler effect does is shift the frequency that the receiver gets. Thus, when two radios are moving away from each other, the perceived frequency by the receiver will be lower, depending on the magnitude of the velocity difference.

For wireless communications, a popular technique to model the Doppler effect is the Jakes model \cite{7}, as is described in Clark Model \cite{9} for fading channels. Since the Doppler effect does not impact the transmitting radio, the Jakes model only considers the shift perceived by the receiving radio. The Jakes makes the following assumptions:

- Radiation propagates horizontally.
- The probability density function of the phase for the received signal is
  \[ f_x(x) = \frac{1}{2\pi}, \quad x \in [-\pi, \pi], \quad (2.3) \]
- The receiver antenna radiation pattern is circular symmetric, that is an omnidirectional antenna is used.

From these assumptions it follows that the probability density function (PDF), $p$, of the Doppler shift is:

\[
p_f(f) = \begin{cases} 
  \frac{1}{\pi f_{max} \sqrt{1-(f/f_{max})^2}}, & |f| < f_{max} \\
  0, & |f| > f_{max} 
\end{cases} \quad (2.4)
\]

With this PDF we need only choose $f_{max}$ for the simulation. This way the Doppler shift is kept random, yet also realistic with respect to the chosen environment.

2.1.3 Fading

Fading is the most interesting and diverse of all channel impairments. There are different types of fading, that apply to different environments. The types of fading are as diverse as the names that apply to each. In this paper there are three types of fading but several names apply to each \cite{10}:

- Very Slow Fading - Path Loss.
- Slow Fading - Shadowing.
- Fast Fading - Multipath Fading, Rayleigh Fading, Rician Fading.
Sometimes different names have different implications. For example, Rayleigh fading and Rician Fading are similar, but not the same. With so many different terms that describe fading, it can difficult to precisely define the word. However, for this paper, fading will be defined as a change to the signal attenuation, that may or may not vary with time.

As one may expect, fading has multiple sources though all are due to the environment in which the wireless signal propagates. Common causes of fading are:

- Reflections off of objects such as buildings or automobiles.
- Shadowing, when large objects that lie in the path of the transmitter and receiver that the signal must diffract around.
- Fading from path loss when the distance between the transmitter and receiver.

**Flat Fading and Frequency Selective Fading**

Fading can affect different parts of the spectrum equally or unequally. When considering flat fading and frequency selective fading, one must first consider the receiver spectrum of interest. A receiver is only interested in listening to a specific band of frequencies that it expects a signal to arrive on. Thus, one must consider coherence bandwidth. If a receiver is interested in a specific band of frequencies, $B$, then coherence bandwidth may or may not be larger than $B$. The coherence bandwidth is the bandwidth where similar fading occurs. That is, the magnitude of the fading in the coherent bandwidth is roughly constant when there is flat fading. When the fading is frequency selective, the magnitude of the fading is non-uniform over the coherence bandwidth.

**Very Slow Fading**

Very slow fading is due to path loss. This is when the transmitted signal is attenuated as a function of the distance from the transmitter. The function that determines the attenuation depends on many factors. Generally, the loss is a inverse square function of the distance from base station and the mobile radio. Our proposed network infrastructure dictates that the base station antennas are going to be below rooftop level. Considering this, an appropriate model for suburban and urban environments is the following \[ L = -10 \log \left( \frac{\lambda}{2\sqrt{2\pi d}} \right)^2, \] (2.5)
Where the loss is measured in decibels (dB), \( \lambda \) is the wavelength, and \( d \) is the distance between the two radios in kilometers (km). The path loss of a rural channel can be modeled in a more simple fashion such as Equation (2.6).

\[
L = \frac{1}{d^4}
\]  
(2.6)

**Slow Fading**

When considering slow fading, primarily caused by shadowing, rural areas can be divided into two categories: rural flat and rural hilly. When considering rural flat channel, such as the U.S. Midwest where tornadoes occur, the effects of shadowing are small. However, when considering rural hilly environments the shadowing effects are no longer negligible. The shadowing effects that occur in a rural hilly environment can be modeled with knife edge diffraction techniques [10]. Buildings in urban and suburban environments can be modeled in a similar fashion. However, in these environments the geometry must be changed. The coordinates chosen to model a hill will result in an inverted triangle shape, but those of a building will be a square.

A hill shall be modeled by an obstruction parameter, \( h \), defined as [10]

\[
h = y_a - \left( \frac{y_r - y_t}{x_r - x_t} (x_a - x_t + y_t) \right),
\]  
(2.7)

Where \( x \) and \( y \) are the x and y coordinates of the transmitter and receiver. The obstruction parameter is then used to create the upper limit for the Fresnel integrals used to create the channel model.

\[
u = \frac{\sqrt{2h}}{R_1},
\]  
(2.8)

Where \( R_1 \) is the radius of the Fresnel zone. The upper limit is then used here:

\[
E = (1 - j)(j/2) \left( 1/2 - \int_0^\nu \cos \left( \frac{\pi x^2}{2} \right) dx. - j \left( 1/2 - \int_0^\nu \sin \left( \frac{\pi x^2}{2} \right) \right) \right),
\]  
(2.9)

The process will repeated for each hill, or similar obstruction, in the proposed environment.
For fading in other environments, the Hata’s model is an improvement on the Okumura model, these models are listed below. Here are the set of equations that will be used to model the path loss for each environment pertinent to this project. For urban environments, the function $a(h_m)$ provides a correction for the size of urban environment. The model for the loss in each environment changes depending on the size of the city.

1. Urban

$$L = [69.55 + 26.16 \log(f) - 13.82 \log(h_t) - a(h_m)] + [44.9 - 6.55 \log(h_t)] \log(d), \quad (2.10)$$

(a) Large Urban - Correction

$$a(h_m) = 3.2[\log(11.75h_m)]^2 - 1.1, \quad (2.11)$$

(b) Medium Urban - Correction

$$a(h_m) = [1.1 \log(f) - 0.7]h_m - [1.56 \log(f) - 0.8], \quad (2.12)$$

2. Suburban

$$L_{suburban} = L - 2[\log(f/28)]^2 - 5.4, \quad (2.13)$$

3. Rural

$$L_{rural} = L - 4.78[\log(f)]^2 + 18.33 \log(f) - 40.94, \quad (2.14)$$

Where $f$ is the frequency in Megahertz (MHz), $d$ is the distance between receiver and transmitter (in kilometers), $h_t$ is height of the transmitter antenna (in meters), and $h_m$ is the height of the personnel radio antenna (in meters).

Fast Fading

Fast fading occurs due to Rayleigh and Rician fading, that is any kind of multipath. The difference between the two types fading is the line of sight between transmitter and receiver. When Raleigh fading occurs, there is no line of sight on the other hand, when Rician fading occurs there is line of sight component. Since Rayleigh fading has a more significant impact on the channel, only Rayleigh fading will be addressed in this project.

Channels with Rayleigh and Rician multipath fading are illustrated in Figure 2.1 and
In these figures, "TX" denotes a transmitter, "RX" denotes a receiver. The blocks are reflectors, except for the middle block in the Rayleigh model, which is an obstruction.

Figure 2.1: A Channel Environment with Rician Fading

Figure 2.2: A Channel Environment with Rayleigh Fading

There are two approaches to simulating a Rayleigh fading channel for this project.
• Channel Simulation Features of the MATLAB Communications Toolbox

• Design the channel from scratch.

To simplify the process, the MATLAB Communications Toolbox will be used, unless some type of customization is needed. Although the Communications Toolbox is going to be used, the underlying mathematics are the same.

The Communications Toolbox models a Rayleigh fading channel as a Discrete Finite Impulse Response Filter (FIR Filter). The filter that the Communications Toolbox uses is as follows:

\[ y_i = \sum_{n=N_1}^{N_2} s_{i-n}g_n, \]  \hspace{1cm} (2.15)

Where \( s_i \) is the input vector for the filter, \( y_i \) is the output of the filter, and \( g_n \) is the weights of the filter’s taps:

\[ g_n = \sum_{k=1}^{K} a_k sinc \left( \frac{\tau_k}{T_s} - n \right), \]  \hspace{1cm} (2.16)

Where \(-N_1 < n < N_2\), \( T_s \) is the sample period, \( \tau_k \) is the set of path delays, \( a_k \) is the set of complex path gains, \( 1 < k < K \) and \( K \) is number of paths [13].

There are certain gains and losses in specifying the channel when using the MATLAB Communications Toolbox. The underlying mathematics of describing the channel need not be dealt with. What is important to describe is the parameters passed to the Rayleigh Channel Objects. The most relevant of these parameters are the sample period \( T_s \), number of propagation paths \( \tau_k \), the propagation delays, \( a_k \), and the average path gains, \( k \).

2.1.4 Characteristics of Environments

As one can see from the prior section, the impairments wireless propagation experiences depends heavily on the environment. As one may imagine, nearly every feature that can describe an environment influences the channel. Take a building in an urban environment for example. The materials the buildings are constructed with will influence signal reflections and scattering, the height of the building will influence how the signal get attenuated, etc.

For this project three environments were chosen, along with the sub-environments, rural, rural hilly, rural barren, urban, urban suburban, medium urban, large urban, and under-
Table 2.1: Channel Profiles

<table>
<thead>
<tr>
<th>Environment</th>
<th>Multipath</th>
<th>Scattering</th>
<th>Diffraction</th>
<th>Doppler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rural</td>
<td>Small</td>
<td>Small</td>
<td>Small</td>
<td>Large</td>
</tr>
<tr>
<td>Urban</td>
<td>Large</td>
<td>Large</td>
<td>Large</td>
<td>Small</td>
</tr>
<tr>
<td>Subway</td>
<td>Very Large</td>
<td>Small</td>
<td>Medium</td>
<td>Small</td>
</tr>
</tbody>
</table>

Before describing each environment in depth, it may help to go over the general characteristics of each environment, as seen in Table 2.1.

**Rural**

The rural channel is by far the simplest to understand and implement. In a rural setting, there are few obstructions for transmitted signals. With few obstructions, wireless signals have fewer objects to reflect off of leading to much less multipath interference. An environment with fewer objects to reflect signals also has many other beneficial properties. These include less diffraction and signal scattering. There are also typically fewer people in rural settings. Fewer people leads to fewer spectrum users and a more accessible spectrum for emergency service providers.

In rural environments there are few objects for wireless signals to reflect off. What this means is that the magnitude of the multipath fading is less than that of all the urban environments. Since multipath is usually the most destructive impairment this leads to rural environments being better environments for wireless communications.

The single largest obstacle in outdoor environments is the Doppler effect. For example, a disaster that may occur in an rural environment is a tornado. Tornadoes create large disaster areas that must traversed by emergency vehicles. Emergency vehicles generally travel to a disaster area as quickly as possible, creating a large Doppler shift for wireless communications systems.

**Urban**

In an urban environment there are many more challenges to developing and transmitting in the channel model. In urban environments there are many large buildings, as well as other structures (vehicles, street signs, billboards, etc.). This leads to worse conditions for all variables in the channel. These extra objects in the environment also increase the amount of
reflections, leading to more multipath interference, more signal scattering, and more signal diffraction. Objects in the environment are likely to be above the our system.

However, there is one benefit wireless communications systems benefit from in urban environments. This lone benefit is a reduced Doppler effect. Cities are more compressed than rural environments, the need to transportation is less. Also, transportation in cities, even for emergency vehicles, is slower than transportation in rural environments.

2.2 Fundamentals of Geolocation

Geolocation is a methodology of locating a target wirelessly. This can be done using RF signals or other methods such as inertial navigation. In our project, we use RF based geolocation. The two most commonly used geolocation techniques are:

- **Received signal strength based geolocation** - RSS localization uses a signal pathloss model to calculate the signal attenuation from transmitter to receiver. This pathloss metric can be used to calculate the distance the signal has travelled, and this is the distance between transmitter and receiver for a line-of-sight signal path.

- **Signal flight time based geolocation** - An RF signal will travel from the transmitter’s antenna to the receiver’s antenna at the speed of light. If the time of flight of a signal is known, the distance the signal has travelled can be calculated. For a line-of-sight signal path, this is the distance between the transmitter and the receiver.

2.2.1 Received Signal Strength

Received signal strength methods of geolocation are based on the fact that pathloss can be related to distance travelled by an RF signal with [?]:

\[ RSS_d = 10 \log_{10}(P_t) - 10 \alpha \log_{10}(d) + X, \]  

where \( \alpha \) is the attenuation factor or distance power gradient and \( d \) is the distance from receiver to transmitter. \( X \) is the lognormal random variable representing shadow fading. Shadow fading is attenuation in signal power that occurs when a mobile terminal is shadowed by an obstruction or scatterer. In an ideal outdoor environment, there would be no shadow fading.

A minimum of 3 transceivers performing this range measurement are needed to perform trilateration to converge the loci of the possible location to a point, in this case the fourth
transceiver. The algorithm simply consists of calculating the distances and where the locus circles intersect is the estimated position as shown in Figure 2.3 [14].

\[ d = c_{\text{light}} \times t_{\text{flight}} \]  

(2.18)

where \( c_{\text{light}} \) is the speed of light in air and \( t_{\text{flight}} \) if the time it takes the signal to travel from the transmitter to the receiver. This calculation is done by three or more sensors, and the intersection of all loci circles is the transmitter's location relative to the sensors.

Difficulties arise in geolocation when the measured values of the aforementioned metrics are inaccurate due to noise or inaccurate estimation of pathloss gradients. For example, in TOA based ranging, if the signal does not have a direct line of sight from transmitter to receiver it will most likely bounce off other obstacles and eventually arrive at the receiver.
This means that the signal’s flight time will be longer, and hence the range estimate will be inaccurate. The error can be very large for a few tens of nanoseconds. In RSS based methods, if the signal passes through an object, some of it will be absorbed and the remaining will be transmitted. The $\alpha$ used in the RSS equation will no longer predict the signal strength accurately. Generally, the gradient is not known or needs to be calculated for a particular environment. It can also change over time as the environment changes.

### 2.3 VITA 49 Radio Transport Standard

As part of this project, a firm understanding the VRT Digital IF Standard is required. Understanding this standard has been gained through reading the VITA 49.0 Base Standard Document. This document explains the overall concepts of the VRT Standard, such as the different types of streams in the VRT standard (Information Streams and Context Streams). The document continues by going into further detail, describing, for example, the header field information.

The VRT 49 standard was designed for software-defined radios. With software-defined radios, the datatypes needed are meta-data and data. In VRT literature, meta-data is also called context data. Let us consider a first responder sending a message to his teammates. The data would be the voice or text data and the meta-data would be the GPS coordinates of the transmitter. The separation between data and meta-data is key for SDRs, since it provides extra data needed without loading the user down with extra responsibilities.

The VRT standard works by transferring information from transmitter to receiver as streams. In VRT, a stream can transfer data, or meta-data. A stream from to one radio to another cannot be the same stream as from the first radio to a third radio. All of these streams can be superimposed on one another to form one information stream. This concept is best illustrated in Figure 2.4 [1]. In this figure, the square represent one user, the diamonds another user. The 'D’s represent data and ’M’s represent meta-data. The tube represents the channel that VRT transmissions propagate through.

The VRT standard is also highly versatile. VRT is a network protocol, just like TCP, UDP, IP, etc. Since VRT is a network protocol it can be used to make a network stack, where VRT can sit on other protocols, like UDP. In Figure 2.5 and 2.6 are two VRT network stacks. In both stacks the application layer is the payload of the VRT layer, typically this is voice data, but could be anything. VRT is technically a transport layer protocol, also like UDP and TCP. However it can be used over UDP to facilitate communications between a
host PC and a software-defined Radio over an Ethernet Link, Figure 2.5 shows this network stack. VRT can also be used for radio platforms where the physical layer is wireless, as shown in Figure 2.6.

The versatility of the VRT protocol does not end with implementing network stacks. VRT streams and the packets that make up those streams are also highly configurable. For the meta-data packets, there are twenty-four optional fields. The user or system designer can use any combination of these fields to make up the meta-data packets, the fields are listed on Table 2.3. In this table, column two 'Words' shows the number of 32-bit words each field will take in the context packets.

There is indeed more configurability with the VRT protocol, as one can see in the structures of the data packets and meta-data packets. These structures as shown in Figures 2.7 and 2.8 respectively. In both of these Figures the 1’s and 0’s are bit that are hard coded, they may not change as by the VRT Standard.

The VRT standard provides even more flexibility, with the format of the packet headers. All fields except the main header and payload are optional. Their presence is denoted by the bits that are set or not set in the main header. Furthermore there are several ways to
Figure 2.6: Another Typical VRT Network Stack

Figure 2.7: VRT Data Packet Structure
<table>
<thead>
<tr>
<th>Field</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context Field Change Indicator</td>
<td>0</td>
</tr>
<tr>
<td>Reference Point ID</td>
<td>1</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>2</td>
</tr>
<tr>
<td>IF Reference Freq</td>
<td>2</td>
</tr>
<tr>
<td>RF Reference Freq</td>
<td>2</td>
</tr>
<tr>
<td>RF Reference Offset</td>
<td>2</td>
</tr>
<tr>
<td>IF Band Offset</td>
<td>2</td>
</tr>
<tr>
<td>Reference Level</td>
<td>1</td>
</tr>
<tr>
<td>Gain</td>
<td>1</td>
</tr>
<tr>
<td>Over range Count</td>
<td>1</td>
</tr>
<tr>
<td>Sample Rate</td>
<td>2</td>
</tr>
<tr>
<td>Timestamp Adjustment</td>
<td>2</td>
</tr>
<tr>
<td>Timestamp Calibration Time</td>
<td>1</td>
</tr>
<tr>
<td>Temperature</td>
<td>1</td>
</tr>
<tr>
<td>Device ID</td>
<td>2</td>
</tr>
<tr>
<td>State and Event Indicators</td>
<td>1</td>
</tr>
<tr>
<td>IF Data Packet Payload Format</td>
<td>2</td>
</tr>
<tr>
<td>Formatted GPS Geolocation</td>
<td>11</td>
</tr>
<tr>
<td>Formatted INS</td>
<td>11</td>
</tr>
<tr>
<td>Earth Centered Earth Fixed, Ephemeris</td>
<td>13</td>
</tr>
<tr>
<td>Relative Ephemeris</td>
<td>13</td>
</tr>
<tr>
<td>Ephemeris Reference ID</td>
<td>1</td>
</tr>
<tr>
<td>GPS ASCII</td>
<td>Variable</td>
</tr>
<tr>
<td>Context Association List</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Table 2.2: Inclusion of VRT Context Fields

Figure 2.8: VRT Meta-Data Packet Structure
set the format of the timestamps depending on whether the integer Seconds or fractional seconds timestamp are being set. For the integer seconds timestamp, there are three options: coordinated universal time (UTC), GPS time, or a custom time format. For the fractional seconds timestamp the timestamps can be set by the sample count, real-time (resolution to one picosecond), or the free running timestamp.

2.4 Literature Review

2.4.1 Channel Models

Three references were instrumental to completing the channel models used in this project, namely [7], [10], and [?]. These three texts formed the basis for all subsequent research. With these books, the most popular channel modeling techniques were researched by finding the original papers. The two most popular models are the Hata and Okumura techniques which were researched further in references [6] and [12]. These two models are used to find the fading characteristics of different environments. These two papers provided a basis for creating path loss models, which were further explored in reference [11].

For the research pertaining to the Doppler effects, two sources were used reference [9] and [9]. Doppler effects are the shift in frequency when there is movement between the transmitter and receiver.

Other sources were also examined. One such example is reference [15] which provided an overview of many different channel models, providing guidance for matching environmental situations with specific models.

2.4.2 Geolocation

There are a number of sources that have shaped the final design of our system. Some of these sources provided motivation for a certain methodology, other simply consolidated our background in necessary concepts for geolocation. [14] gave insight on selective decision making on TOA metrics in cooperative networks. [16] is a one of the first texts solely written for ground based non-GPS localization techniques and provides good background material. [17] provides insight on pathloss exponent estimation in sensors networks. [18] uses TOA, AOA and RSS techniques for self localization of sensors in cognitive radio networks. [8] is a detailed treatment of the fundamentals of channel models and error sources for geolocation. [19] introduces the grid based TOA localization method that our system employs. [20] talks
about RSS weightor believable factor calculation and pathloss gradient estimation for non line of sight conditions. [21] introduces effective cooperative 802.11 localization system for WLANs in indoor locations. All of these sources were studied during design of the system in order to determine critical aspects that needed to be developed.

2.4.3 VRT

Only one source was used while researching the VRT, reference [1]. This source is essentially the only source one the VRT, as it is the definition of the standard. The VRT standard is written and distributed by the VITA standards organization. The VRT standard has become the only standard specifically created for SDRs.

2.5 Chapter Summary

This chapter provides the necessary background to the reader to understand the following chapters. In this section mathematics needed to understand channel modeling and geolocation are reviewed. These mathematical properties are essential to the following chapter of this report. Then the VRT Standard is discussed as a standard on which the communications simulation system is implemented. Finally, there is a literature review, which discusses the sources used in this paper.
Chapter 3

Engineering Design Process

This chapter details the engineering design process followed during the course of this project. The research areas, project organization, division or labor, meeting schedule, significant problems, and deliverables will all be discussed.

3.1 Research Areas

The first step to completing any project is to identify the areas of need with respect to research. Three basic areas of research were found for this project:

1. Channel Models
2. Geolocation
3. Spectrum Sensing

From these basic research areas, more specific areas were found and researched further. These more specific research areas are specific channel models, the Hata, Okumura, and Jakes models. Specific research areas for geolocation were RSSI and TOA algorithms and hybrid algorithms that employed both TOA and RSS based localization [17]. The specific research areas for the spectrum sensing are the cyclostationary detector and data fusion techniques.

3.2 Project Organization

A project of this size needs to be organized centrally. All team members need to have access to the other members work as well as have a place to back-up their own work.
The solution to this problem is to use a version control system. The version control system chosen was subversion. Subversion was chosen because it was already available on the department servers and it provides the functionality needed.

The needed functionality was to share work and backup work. The subversion server was set up such that each member had their own branch, along with a main branch called 'trunk’. When an individual team members source code was finished, it was move to the trunk directory. There was also another folder for the project reports. The project reports were written in latex.

3.3 Division of Labor

This is a team project, made up of three people. Each team member has different responsibilities. The responsibilities were divided in such a way that each portion of the project could be completed individually and then easily combined with the work of the other team members.

The responsibilities were:

- Michael Calabro
  - Cyclic detector – The cyclic detector requires knowledge of statistics and random processes with respect to digital communications systems.
  - Data fusion – The data fusion algorithm works closely with the cyclic detector.

- Devin Kelly
  - VRT packet generator – The VRT packet generator requires understanding of network protocols and theory.
  - Channel and environment design – Requires knowledge of communications systems.
  - Simulation design – Must be closely integrated with channel and environment models. In addition object oriented design knowledge needed.

- Ishrak Khair
  - Geolocation algorithm design – Requires knowledge of localization methods and RF propagation.
  - GUI design – The GUI design requires knowledge of the MATLAB GUI design environment.
The division of labor was made to match the background of each team member. Devin Kelly and Micheal Calabro both had full wireless communications backgrounds, so they were assigned to the channel models and cyclic detector. Ishrak Khair, however, had more experience with RF and MATLAB GUI design, so those task were assigned to him.

The interface from one module to another were well defined, taking a “black box approach.” This approach simplifies combining each module.

### 3.4 Meetings

Several weekly meetings were scheduled. These meetings were scheduled between the advisor and team members as well as meetings between the three team members.

The first set of meetings were held between the three team members and the project advisor. These meetings were held to review the overall system design and review progress.

The second set of meetings are for the team members to discuss specific problems and review one another’s code.

### 3.5 Problems Encountered

There were two significant problems encountered as during this project. The first was with the implementing the project in hardware, the second was the simulation performance.

Implementing the project in hardware was a problem that was not overcome. The lack of a hardware solution comes from the need of the geolocation system to use TOA and/or RSSI. These techniques require either precise timing or access to the USRPs automatic gain control. Accessing either of these would require work that is out of the scope of this project as documented in Chapter 4.9.

Increasing the performance was successful. The performance bottleneck is the cyclic detector. After compiling the cyclic detector, performance increased, however the bottleneck remains at the cyclic detector. The process of compiling the cyclic detector and the analysis of the performance gains are documented in 4.6.
Chapter 4

Proposed Design

The design proposed to solve this problem focuses on combining modular components. This design choice fits this project well as it encourages data to flow in a clearly defined way. This data flow is triggered by first responders transmitting their data of their own.

Whenever there is a transmission by a first responder, the first of several blocks are called, the VRT packet generation block. From here, the channel models are applied to the signal. Namely, each sensor receives the signal. From this point the cyclostationary and geolocation algorithms are applied. Then all this data is deposited in a database. Finally, our simulation compares the statistics our simulation finds with the real data and prints efficiency our the system as a whole. This entire is data flow is represented in figure 4.1 Network Design

For the proposed network design a top-down approach was taken. Part of the design requirements was to create and maintain a database of the emergency responders radios. This implies that the database must be centralized, or in one place. However, the data must be gathered in a distributed manner.

The structure of the network must have both distributed elements and centralized elements. This leads to an architecture where spectrum is sensed locally and a database is kept in a centralized location.

With these requirements in mind, a list of network trade-offs was developed:

- Power Consumption: Should the sensors process the data or transmit it wirelessly to the centralized location
• Sensors: Should the sensors be disposable battery powered devices, or embedded into emergency responder vehicles

• Bandwidth: How much bandwidth can be used to transmit raw data back to centralized location without interfering with the emergency responders.

The decisions made with regard to the trade-offs in Listing 4.1 dictated how the network architecture was designed. First, in a disaster environment emergency vehicles may not be able to get to all the places where the sensors need to be. For example, in no emergency vehicles could climb the twin towers. So the design decision was to used battery powered disposable sensors.

Second, the trade-off between data processing locally (on the sensor) compared to on the centralized location leans towards processing the data locally. This is for two reasons, the first being that wireless transmission of processed data has many problems. For example bandwidth is already scarce, so adding many long (in time) and wide (in frequency) transmissions would only make bandwidth more scarce. Having these extra transmissions also increases the probability of false-positive detection. That is, if one sensor were to transmit to the centralized location another sensor may detect that transmission and falsely treat it as an emergency responders signal. When processing the data on the sensor there is also the concern of power consumption. However, since the only two options are to process or transmit, both of which consume large amounts of power, the trade-off in this respect is minimal. In order to preserve bandwidth it was chosen to do all the processing on the sensors.

The network architecture chosen was a two tier architecture. At the top, there is the centralized database. At the bottom are all the sensors. The duty of the sensor is sense spectrum, then report the emergency responders ID, the estimated modulation scheme, the estimate confidence, and the estimated coordinates. Then it is up to centralized database to fuse all the data, into one database. For a network architecture diagram, see Figure 4.1.

4.2 VRT Packet Generator

In order to test the algorithms that are being developed, there must be data to manipulate. To make the simulations are realistic as possible, that data should take the form of VRT packets. In order to generate realistic VRT packets, several factors must be considered, these are largely filling in the VRT headers correctly. These factors are the packet count, packet size, and timestamps.
The other header and trailer fields should remain constant. The fields in these packets were filled out in as exemplified by GNU Radio. These three fields are unique in that they change for each packet sent or received. An object oriented approach was chosen, a VRT packet object was created giving the number of samples per packet and the number of packets to be created. From there, a matrix is created with dimensions of the number of samples per packet by the number of packets. The strategy to have accurate packet count number was to start at one and increment from there. The way in which the code was structured made this easy to do. The packet size is the number of samples plus the number of headers, for GNU radio this number is 6. Finally the timestamps were found using the MATLAB function `now`.

The headers were formatted according to the specification of GNU Radio. That is, GNU Radio is switching its host PC to USRP2 communications to use the VRT standard. The headers for this VRT Generator was generated according to the GNU Specification. This specification is as shown in Table 4.1.

From this point the VRT Packet Generator took on a variety of other roles. One of the most important is modulation of the generated signal. The modulation schemes that compatible are listed in the listing below.

The VRT generator has more functionality. The generator also performs matched filtering. There are two main varieties, the square root raised cosine filter and the raised cosine filter. Both are FIR filters with a user controllable roll-off factor, $\beta$. Figure 4.2 depicts how
Table 4.1: Header and Trailer Description

<table>
<thead>
<tr>
<th>5491006a</th>
<th>Header and Packet Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaaaaaa</td>
<td>Stream ID</td>
</tr>
<tr>
<td>94cedde8</td>
<td>Integer Time Stamp</td>
</tr>
<tr>
<td>00000000</td>
<td>Fractional Time Stamp</td>
</tr>
<tr>
<td>00000064</td>
<td>Fractional Time Stamp</td>
</tr>
<tr>
<td>0dcc6eff</td>
<td>Sample</td>
</tr>
<tr>
<td>16b6c51a</td>
<td>Sample</td>
</tr>
<tr>
<td>1b034ee8</td>
<td>Sample</td>
</tr>
<tr>
<td>1dd5ee6</td>
<td>Sample</td>
</tr>
<tr>
<td>2038799</td>
<td>Sample</td>
</tr>
<tr>
<td>...</td>
<td>Continued</td>
</tr>
<tr>
<td>55555555</td>
<td>Trailer</td>
</tr>
</tbody>
</table>

Table 4.2: Modulation Schemes Implemented

<table>
<thead>
<tr>
<th>BPSK</th>
</tr>
</thead>
<tbody>
<tr>
<td>QPSK</td>
</tr>
<tr>
<td>16 QAM</td>
</tr>
<tr>
<td>64 QAM</td>
</tr>
<tr>
<td>4 PAM</td>
</tr>
<tr>
<td>8 PAM</td>
</tr>
<tr>
<td>8 PSK</td>
</tr>
<tr>
<td>16 PSK</td>
</tr>
<tr>
<td>2 FSK</td>
</tr>
<tr>
<td>4 FSK</td>
</tr>
<tr>
<td>8 FSK</td>
</tr>
</tbody>
</table>

manipulating $\beta$ can change filter response. In this figure, $H(f)$ is the filter response, $f$ is the frequency, in Hertz, and $T$ is the symbol period, in seconds.

The research performed by our group indicates that the type of data to that will be sent by radio users is voice data. The samples that are placed in the VRT packets are generated with the MATLAB function `randn`. To better simulate voice data, the samples were passed through a simple low pass filter. This VRT generator stimulated the development of both the channel models and the cyclostationary signal detection.

### 4.3 Channel Model Design

All of the functionality that the channel models provide has been consolidated into one MATLAB object. This design was chosen to give the user most flexibility possible. This flexibility
Figure 4.2: Matched filter response with different values for $\beta$ 

is provided by creating default values for the user who does not want to specify everything about their channel. However, optional arguments are provided to the user allowing them to have control if needed.

As stated prior, many of the parameters of the channels are tunable, as per design. The constructor allows for a few basic parameters to be set, setting other parameters to their defaults. Another advantage of pursuing this channel design, is the step-by-step approach to designing the channel. That is, if a user wants only a test signal with multipath impairments, they call only the one corresponding method. This allows for any combination of impairments the user desires.

Developing realistic channels involves tuning many parameters, these include:

- Transmitting Antenna Height -
  The length of the transmitting antenna.

- Receiving Antenna Height -
  The length of the receiving antenna.

- Transmitting Antenna Gain -
  The gain of the transmitting antenna.

- Receiving Antenna Gain - The gain of the receiving antenna.

- Number of Propagation Paths -
  The total number of reflections the receiver receives.
The Delay Associated with Each Path -
The amount of time delay for each path.

The Maximum Doppler Shift for Each Channel -
The upper limit of the Doppler shift for each channel

Some of these parameters are easier to estimate than others. A simple parameter to estimate is the antenna heights. For the purposes related to this project there are essentially three types of antennas: vehicle mounted (base stations), sensor antennas, and handheld antennas (mobile stations).

The design of channel models again focused on giving the user the most choice in the model. The user can set nearly any parameter of the channel they wish. These parameters include are the noise power, with the users units of choice - dB, dBm, or watts. The fading options are Rayleigh or Rician fading with the number of paths, the multipath delay spread, average path gain. If the fading is Rician, the k factor may also be set by the user Doppler Spread.

The channel models were designed to be used any user, even users who are unfamiliar with channels. To accommodate, the user may set no parameters of the channel, or at least set the environment type. From there the channel object can set its own default configuration.

4.3.1 Noise Impairments

Additive noise is the basis for all channel models, as it is present in all communications media. Additive noise plays an especially large role for wireless communications, as wireless signal strength dissipates faster than in wired signals. Since noise plays such a large role, it is important to simulate additive noise properly.

Additive noise is added to the transmitted data using the MATLAB function `awgn`. The `awgn`, or additive white Gaussian noise, function adds noise of a given power to the signal. As outlined in Chapter 2, noise is modeled as a additive, white, zero mean, Gaussian random process. The MATLAB function `awgn` matches these needs perfectly. This function will add noise, with constant average power across all frequencies, with zero mean. Most importantly, `awgn` allows the user to set the noise variance, or the noise power.

Figures 4.3 and 4.4 show how the function `awgn` can be used. Figure 4.3 represents a signal generated with no noise that has just been passed through the matched filter. In Figure 4.4 there is the same signal, but with noise added.
Using the MATLAB function `awgn` is simple, one need only pass the original signal, the desired signal-to-noise ratio, and the unit of the noise power. However, the process is further simplified by allowing the user need only pass the desired signal-to-noise ratio to the channel object and the rest is handled automatically.

### 4.3.2 Fading Impairments

The fading effects are currently handled with MATLAB objects. The fading property, most importantly the fast fading property, of the channel is calculated using the `Rayleighchan` or `Ricianchan` objects. These objects allow the user to specify the number of paths and the delays of each path. This will allow testing of many channels, with different multipath components, quickly and easily. For the purposes of this project, the Rayleigh channel object will be used primarily, but if needed the `Ricianchan` object is also available to use and is easy to swap with the `Rayleighchan` object.

One of the design goals is to simplify the user interface while providing unique channels, details about the fading characteristics of the channels. Knowing the environments types
(rural, suburban, urban) ahead of time made this simplification easier. Data about each channel was gathered, specifically on the delay spread and the maximum delay. The delay spread, or the maximum delay, and the maximum Doppler shift is as defined in Table 4.3:

Table 4.3: Delays Spreads and Doppler Shifts for Various Environments

<table>
<thead>
<tr>
<th>Environment</th>
<th>Delay Spread (µS)</th>
<th>Doppler Shift (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat Rural</td>
<td>0.5</td>
<td>190</td>
</tr>
<tr>
<td>Hilly</td>
<td>20</td>
<td>190</td>
</tr>
<tr>
<td>Urban</td>
<td>5</td>
<td>120</td>
</tr>
<tr>
<td>Indoors</td>
<td>0.1</td>
<td>5</td>
</tr>
</tbody>
</table>

Additionally, the number of paths wireless signals propagate in is dependent on the specific environment. However, some generalizations can be made. For example, urban environments typically have more paths than any other, save underground environments. For urban environments, there are typically 6 to 12 paths depending on the specific environment. From this Figure of 6 to 12 paths, the number of paths decreases for the other environments.
The technique chosen to generate an accurate channel at this time is to first generate the number of paths. In an urban environment, this will be a random number between 6 and 12. Then, for each path a delay will be generated randomly, using the appropriate delay spread as the maximum delay. In more mature models, this technique will change slightly. Instead of randomly generating the number of paths, an environment will be given, along with the number of paths for a transmission. That is, the number of paths will depend on the obstacles in the environment.

4.3.3 Doppler Impairments

Doppler fading is the result of difference in velocity between the receiver and transmitter. When there is a difference, the perceived frequency by the receiver shifts as a function of the velocity difference. Simulating the Doppler effect was left to using the tools provided by the Communications Toolbox. The tool used was the Jakes Doppler object.

The Doppler effect and scattering effects are calculated with the MATLAB object Doppler.Jakes. This object constructs a Jakes Doppler spectrum which is used in the Clark model. This MATLAB object is dynamic as it allows the user to specify the Doppler shift. The Jakes Doppler object also adds scattering. The specified frequency is the maximum Doppler frequency. That is it is the maximum difference in frequencies between the transmitted frequency and the received frequency.

The Doppler.Jakes object was used in the channel object developed as part of this project. It was designed similarly to the other modules of the channel object. That is the typical values for each environment were found, as in the Chapter 2 and then used in the channel object. For example, when the channel object is initialized, the environment type is passed to the channel constructor. With this information, the channel object choses the best Doppler object.

4.4 Geolocation System Design

A desirable geolocation method for our system needs to take advantage of our network topology. Some important criteria are listed below.

- *Leverage distributed nature of network* - The geolocation system needs to take advantage of the wireless sensors nodes that will be deployed.
• **Leverage VITA 49 Standard** - The geolocation system should take advantage of the time stamps embedded in the context fields of a VRT data packet.

• **Ease of deployment** - The geolocation system should be deployable in a wide variety of disaster scenarios. This means no prior knowledge of pathloss exponents or other channel characteristics is assumed to be known. We expect heavy multipath conditions.

• **Mitigate NLOS errors** - NLOS (non line of sight) conditions can be substantial, especially in indoor and urban environments. In addition, these NLOS conditions can be time varying as the obstacles blocking the direct path may be in motion. The geolocation system needs to be able to cope with this challenge.

• **No dependence on ultra-wideband techniques** - Our sensor network will be deployed with only one assumption made: the radios used by responders will employ the VITA 49 standard. Our system shall not assume that the radios have any kind of UWB capability. We only assume that our spectrum sensing capabilities will allow us to demodulate the sensed signal and extract information in the contextual packet. This will provide us with TOA metrics for geolocation.

Single metric algorithms, such as RSSI-based methods or TOA methods suffer with multipath fading or temporal pathloss conditions. This means severe degradation of accuracy. A combination of metrics, RSSI and TOA metrics can be used to improve the accuracy and robustness of geolocation systems. This algorithm is proposed in [20]. The paper suggests an algorithm that involves sensors with known locations together with TOA and RSSI measurements that are used to calculate a confidence factor to select the best location estimate. Although RSSI is used, the algorithm also estimates pathloss exponents. This allows for a better performance than single metric systems. The algorithm assumes no prior knowledge of pathloss exponents. The algorithm also expects NLOS conditions.

Our sensor network will employ this algorithm. The USRP2s are equipped with analog RSSI indicators and this algorithm will take advantage of this to perform ranging. The VITA 49 standard will allow us to compute flight time of the signal to perform ranging. We shall use this algorithm as a starting point for our geolocation system.

### 4.4.1 Algorithm Details

Sensors are placed at the emergency location by the responders. The coordinates of these sensors are known to the base station. The base station then generates a grid of locations points and fixes the position of the sensors with their known coordinates.
The sensors are continuously performing cyclostationary analysis on received signals and when the signal is detected (corresponding to modulation detection and hence signal identification) a time stamp of arrival will be recorded. Latency offsets due to propagation delays from the antenna to the digital domain must be considered. This will be found through the USRP2 datasheet and RF Front end components.

Once the modulation is detected, the signal will be demodulated and decoded to reveal context information embedded in the VITA 49 based packet. The time of signal germination, reference point of measurement \cite{24} and ID information will be extracted by the sensors. The sensors will also measure RSS values using pathloss exponents assigned to them from the base station. The base station will select optimum pathloss exponents and assigned them to the correct sensors. The RSS and TOA metrics are then transmitted by each sensor to the base station. At the base station, location estimation is done using the TOA of the received signal and RSS estimates.

### 4.4.2 TOA Ranging

The time of arrival of the signal at the sensor can be calculated as:

\[
t_i = t_{\text{detect}} + t_{\text{latency}}
\]  

where \( i \) is the sensor number and \( t_i \) is the time at which the receiver has received and decoded context information from the VRT packet. \( t_{\text{detect}} \) is the time when the signal arrives at the antenna of the radio. \( t_{\text{latency}} \) is the time it takes the signal to propagate from the front end until it is actually demodulated and identified by our signal detector. The latency offset can be found by using the reference point and time delays for the RF front end being used. We must also remember to account for the latency of the transmitter’s RF front end. As mentioned above, this information will be furnished to the base station as context information. The distance of the responder from sensor \( i \) is then:

\[
d_i = t_i \times c
\]  

where \( c \) is the speed of light. Distances from the candidate grid points to the sensor \( i \) have already been calculated by the base station using:

\[
\delta_{g,i} = \sqrt{(x_i - x_{\text{grid}})^2 + (y_i - y_{\text{grid}})^2}
\]  

where \( x_{\text{grid}} \) and \( y_{\text{grid}} \) are the coordinates of the candidate grid point. \( x_i \) and \( y_i \) are the coordinates of the sensor \( i \). \( \delta_{g,i} \) is the distance between the candidate grid point \( g \) and the
and the sensor $i$. The RMS error of the calculation is found using the difference between the candidate grid point’s distance from the sensor node and the estimated distance using TOA from:

$$e_t = \sqrt{\frac{\sum_{i=1}^{n} (\delta_{g,i} - d_i)^2}{n}}$$ \hfill (4.4)

where $e_t$ is the RMS error of the TOA range measurement. As can be seen, this needs to be performed for all grid points, and all sensor nodes. Once this is calculated, the grid point that minimizes the root mean squared error is the estimated position vector given by:

$$\bar{X}_t = \arg \min_{X_t} e_t$$ \hfill (4.5)
where $X_t$ is the resolved location of the responder.

**RSS Ranging**

RSS ranging is based on the fact that the wave power decreases with the square of the distance travelled by the wave. The estimated distance between the sensing node $i$ and the responder, $r_i$, is given by:

$$r_i = 10^{h(k_i)}$$  \hspace{1cm} (4.6)

where $k_i$ is the pathloss exponent that is set to the sensor $i$. $h(k_i)$ is defined as:

$$h(k_i) = \left( \frac{P_0 - P_i}{10 k_i} + \log_{10} r_0 \right)$$  \hspace{1cm} (4.7)

$k$ can generally range from 2 to 4 [18]. $P_0$ is the transmitted power and can be obtained from the context packet of the transmission that was detected. $P_i$ is the received signal power at the sensor $i$ and can be computed from the signal itself as the RMS power. All power measurements are in units of dBm. One dBm is found by normalizing the actual power (in watts) to 1 milliwatt of power, and taking the logarithm multiplied by 10. The RMS error, $e_{rss}$, is computed as:

$$e_{rss} = \sqrt{\frac{\sum_{i=1}^{n} (\delta_{g,i} - r_i)^2}{n}}$$  \hspace{1cm} (4.8)

where $\delta_{g,i}$ is the distance from the candidate grid point $g$ to the sensor $i$. $r_i$ is the distance from the sensor $i$ to the transmitter, calculated using Equation (4.6). The computation is done for all grid points, and all pathloss exponents $k$. It is important to note that a proper stepsize for $k$ must be chosen, since the smaller stepsize can increase accuracy, but at the expense of processing time. The pathloss exponent, $\bar{k}_i$, and corresponding gridpoint that yields the lowest error using RSS, $\bar{X}_t$, are given by:

$$< \bar{X}_t, \bar{k}_i > = \arg \min_{X_t, k_i} e_{rss}$$  \hspace{1cm} (4.9)

where $\bar{X}_t$ is the chosen location of the responder and $\bar{k}_i$ is the chosen optimum pathloss exponent for the sensor $i$.

### 4.4.3 Believable Factor

We define the believe factor as a dimensionless weighting that is applied to the TOA estimated range to reduce errors from NLOS or shadow fading. Some sensors will inherently
have better ‘view’ to some areas. This can be exploited using the believe factor weighting, by allowing a degree of selectivity when evaluating a series of range measurements from various sensors to increase accuracy. The believe factor is computed using the following equations:

\[
\alpha_i = \begin{cases} 
1 - \frac{|r_i - d_i|}{d_i}, & d_i \geq r_i \\
1 - \frac{|r_i - d_i|}{r_i}, & d_i < r_i 
\end{cases}
\]  

(4.10)

where \(\alpha_i\) is the believe factor for the sensor \(i\). \(r_i\) is the distance of the responder to the sensor \(i\) calculated from the RSS measurement, and \(d_i\) is the distance of the transmitter to the sensor \(i\) calculated from the TOA measurement. Now that the weight has been calculated using TOA and RSS range estimates, the final weighted position estimate is obtained by minimizing the following error function:

\[
e_{wtoa} = \sqrt{\frac{\sum_{i=1}^{n} \left( \delta_{g,i} - \alpha_i \times d_i \right)^2}{n}}
\]  

(4.11)

where \(e_{wtoa}\) is the error for the weighted TOA distance measurement for each grid point over all sensors. \(\delta_{g,i}\) is the distance between the candidate grid point \(g\) and the sensor \(i\). \(\alpha_i\) is the believe factor for sensor \(i\) and \(d_i\) is the range measurement calculated from the TOA measurement. \(n\) is the total number of sensors in the operational field. The final position estimate \(X_{\text{final}}^{-}\) is given by

\[
X_{\text{final}}^- = \arg \min_X e_{wtoa}
\]  

(4.12)

where \(X_{\text{final}}^-\) is the final position estimate of the responder.

### 4.4.4 Geolocation Simulation Model

The model will have an environment block, that will be responsible for generating received signal characteristics for each sensor. This includes generating signal impairments to allow for investigating at what point the algorithm’s performance is no longer satisfactory. We also plan to have a sensor block that extracts context packet metrics and prepares for transmission to the base station. The final base station block will perform the grid setup and implement the above mentioned geolocation algorithm.

In our final simulation we will have the target responder transmit a modulated signal. The environment will degrade this signal before it reaches the sensor. The sensor will then perform cyclostationary analysis and demodulate the received signal, extract context information and transmit to the receiver. The receiver will then use this information to perform geolocation.

In this simulation design, we make an assumption for purposes of validating solely the
geolocation algorithm. When the target node transmits, our sensors will attempt to capture this and perform signal recognition (cyclostationary) analysis and demodulate the signal. Thus the sensor block output is going to be in terms of demodulated context information, and contamination aspects will be added directly. In short, the sensors output to the base station will be contaminated by NLOS offsets, multipath offsets, and pathloss changes. Since our simulation is being done in MATLAB, we perform all processing in baseband. This allows us to run the simulation efficiently and quickly. Some of the key parameters that will enable us to simulate impairments to our geolocation algorithm are:

- **NLOS Offset** - NLOS conditions will mean that the strongest and earliest signal may be received after the signal has bounced off a scatterer. This will mean that the actual time of arrival will be offset by a delay factor.

- **True target position vector** - This will be needed to calculate the actual time of flight of the signal before adding the NLOS offset to it.

- **True sensor positions** - This is needed to calculate the TOA and RSS after pathloss exponents have been set for each sensor.

- **Multipath offset** - due to multipath, the signal that is received at the sensor may be corrupted by intersymbol interference. This directly equates to not recognizing the
first path of arrival of the signal. Therefore, we will model this as a delay offset at the reciever.

- **Pathloss range** - the range of values pathloss exponents can take for each sensor. We must choose the range of pathloss values carefully as there is a tradeoff between computational accuracy and speed of the algorithm.

### 4.4.5 Distance-Power Gradient Estimation

Our geolocation system employs a hybrid TOA-RSS method. The TOA metrics are validated by the likeness of the RSS counterpart. Our design will work well when the number of sensors is three or greater and a good path loss model is used for RSS ranging. Thus, we encounter the difficulty of obtaining an accurate path loss model for the environment the system is being employed in, without prior knowledge of this environment. A calibration phase where the path loss exponents are calculated beforehand can be helpful, but the environment can vary over time, and this will render previous values obsolete.

We have investigated a method of determining path loss exponents. In this method, the path loss exponents are determined based on a geometric compatibility of estimates. We will now describe this methodology.

If we have three sensors, and the path loss model is accurate for all the sensors, the range measurements from RSS will yield intersection between three circles at one unique point. This point is also the radical center of all three circles that result from the individual range measurements. The geometric power of this point with respect to each circle is zero if this point lies on the circumference of each circle. Thus, the algorithm will choose path loss exponents such that the geometric power tends to zero, with respect to each circle.

The geometric power of a point, \( h \), \((x_T, y_T)\) with respect to a circle with center \((x_s, y_s)\) and radius \(r\) is given by [25] as

\[
h = (x_T - x_s)^2 + (y_T - y_s)^2 - r_s^2
\]

where \(r_s\) is the radius (the range measurement) from the sensor. This range measurement is given by [17]:

\[
r_s = 10^{h(p_s)}
\]

where \(r_s\) is the range measurement and \(p_s\) is the path loss exponent that is set to the sensor
s. \( h(p_s) \) is defined as:

\[
h(p_s) = \left( \frac{P_0 - P_s}{10 p_s} + 10 \log_{10} r_0 \right)
\] (4.15)

where \( h(p_s) \) is the logarithm of the distance and \( P_0 \) is obtained from the context packet of the transmission that was detected. \( P_s \) is the received signal strength and is computed from the signal itself as the RMS power in dBm. \( r_0 \) is the attenuation of the signal for unit distance. This is known prior to deployment. Therefore, if we can find the correct value of \( p_s \), we can accurately estimate \( r_s \). If we now have three circles, A, B, C, corresponding to the three sensors’ range measurements, they will converge at a single point of intersection where the target is, if the path loss exponents are correct.

We will now derive the objective function for obtaining the optimum pathloss exponents. The equations of the three circles, on the same plane are:

\[
(x_T - x_A)^2 + (y_T - y_A)^2 - r_A^2 = 0 \quad \text{(4.16)}
\]
\[
(x_T - x_B)^2 + (y_T - y_B)^2 - r_B^2 = 0 \quad \text{(4.17)}
\]
\[
(x_T - x_C)^2 + (y_T - y_C)^2 - r_C^2 = 0 \quad \text{(4.18)}
\]

where \( x_A \) is the x coordinate of sensor A, \( x_B \) is the x coordinate of sensor B and \( x_C \) is the x coordinate of sensor C.

Multiplying out the brackets, subtracting equations, we can take the result and solve for \( x \) and \( y \) using Cramer’s law:

\[
x_T = \frac{\begin{vmatrix} K_{A,B} & 2(y_A - y_B) \\ K_{A,C} & 2(y_A - y_C) \end{vmatrix}}{\begin{vmatrix} 2(x_A - x_B) & 2(y_A - y_B) \\ 2(x_A - x_C) & 2(y_A - y_C) \end{vmatrix}}
\] (4.19)

and similarly:

\[
y_T = \frac{\begin{vmatrix} 2(x_A - x_B) & K_{A,B} \\ 2(x_A - x_C) & K_{A,C} \end{vmatrix}}{\begin{vmatrix} 2(x_A - x_B) & 2(y_A - y_B) \\ 2(x_A - x_C) & 2(y_A - y_C) \end{vmatrix}}
\] (4.20)
Figure 4.7: Circles intersect at radical center
where we define $K_{A,B}$ as:

$$K_{A,B} = r_A^2 - r_B^2 - x_A^2 - y_A^2 + x_B^2 + y_B^2$$ \hspace{1cm} (4.21)

and $K_{A,C}$ as:

$$K_{A,C} = r_A^2 - r_C^2 - x_A^2 - y_A^2 + x_C^2 + y_C^2$$ \hspace{1cm} (4.22)

$K$ is the group of constants in the related equations. Let us call $N_x$ as the numerator for the $x$ solution, and $D$ as the denominator for the $x$ solution. Similarly, $N_y$ is the numerator for the $y$ point solution, and $D$ is the denominator for the $y$ solution.

The power of the point $x_T$, $y_T$ with respect to sensor $A$ is then given by:

$$h_A = (x_T - x_A)^2 + (y_T - y_A)^2 - r_A^2$$ \hspace{1cm} (4.23)

where $h_A$ is the power of the point $A$.

Replacing $x_T$ and $y_T$ with their respective solutions, and multiplying the expression by the denominator $D^2$, we get:

$$h_A D^2 = (N_x - x_A D)^2 + (N_y - y_A D)^2 - r_A^2 D^2$$ \hspace{1cm} (4.24)

We now have an expression for the power of the point $x_T$ without having to solve for $x_T$. The $N$’s and the $D$ encode the positions of the sensors, and the range measurements performed with the previous path loss exponent (or initialization value). This can be done for the other sensors as well, although this is not required. Equation (4.24) is essentially of the same form as that described in [17].

The square of the LHS of this expression can be minimized by choosing path loss exponents that yield correct range estimates for each of the sensors, encoded in the $K$'s. One method suggested by [17] is the Levenberg-Marquardt minimization.

### 4.4.6 Hybrid RSS-TOA Algorithm Evaluation

The proposed hybrid RSS-TOA based geolocation algorithm is evaluated using a MATLAB simulation. The algorithm creates a grid of coordinates, and the distances from each point to the sensor positions are calculated. TOA metrics from a target were used to calculate the best point on the grid that most closely agrees with the range calculated using TOA. However, if the LOS to a sensor is obstructed, this can cause ranging errors. The signal that has bounced, reflected and has been attenuated will have a different RSS-Range than its TOA counterpart. Using the believability factor computation, these signal paths may
be ignored and priority assigned to sensors that have better, more reliable readings. It can be inferred that as the number of sensors increase, accuracy is improved. The simulations conducted confirm this hypothesis. The simulation parameters that the user is asked to enter are:

- **Number of Sensors** - This is the total number of sensors that will be used to perform geolocation in the operational field. The position of the sensors in the field is assigned randomly in MATLAB. The more sensors, the better the algorithm’s performance.

- **Grid Step Size** - The grid step size will determine the spacing between two adjacent grid points. The smaller the step size, the higher the location resolution. However, a smaller step size will increase memory use and processing time.

- **Max X Value, Max Y Value** - These maximum values determine the field area of coverage. This is in meters. The Grids Range from 0 to these values on the x and y axis respectively.

- **PLE (Path Loss Exponent)** - This is the dimensionless number to be used in signal attenuation characteristics. Important assumptions for this parameter are explained below.

Based on these parameters, different simulation scenarios were set up. In each scenario, the TOA metrics are corrupted by NLOS errors.

### Geolocation Simulation Results and Analysis

1. **Scenario A: Performance using a large grid step size**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Sensors</td>
<td>4</td>
</tr>
<tr>
<td>Grid Step Size</td>
<td>2m</td>
</tr>
<tr>
<td>Grid Size</td>
<td>20m by 20m</td>
</tr>
<tr>
<td>PLE</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Figure 4.8 shows that the raw TOA algorithm estimate is far off from the actual true position of the target. The RSS and proposed algorithm position estimates are very similar and are off by approximately 1m. The grid step size of 2m has reduced the resolution of geolocation. Regardless, the ToA NLOS errors are still mitigated.
2. **Scenario B: Performance using a small grid step size**

Table 4.5: Scenario B Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Sensors</td>
<td>4</td>
</tr>
<tr>
<td>Grid Step Size</td>
<td>0.5m</td>
</tr>
<tr>
<td>Grid Size</td>
<td>20m by 20m</td>
</tr>
<tr>
<td>PLE</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Figure 4.9 shows that the effect of reducing the stepsize is an increase in location estimation resolution. Notice that the proposed algorithm and RSS estimates accurately estimate the true position. Also notice that the raw TOA estimate is corrupted.
3. *Scenario C: Performance using a small number of sensors*

Only three sensors were used in the simulation scenario, and the grid step size was a moderate 1m. Figure 4.10 shows that the large NLOS offset to the TOA range measurements throw the ToA position estimate off. The proposed algorithm is able to adapt using RSS weighting. Although a small number of sensors may allow faster tracking, the trade off is error that appears in Figure 4.10.
4. Scenario D: Performance using a large number of sensors

In contrast with Scenario C, the number of sensors in this scenario has increased to 7. The raw TOA estimate is corrupted by NLOS errors. However, the large number of sensors improve the performance of the algorithm, and the location estimate is quite accurate. Figure 4.11 shows how increasing the number of sensors with the given parameters changes the distance error between the estimated position of our proposed algorithm and the true location.

The error estimates quickly go down to approximately 0.5m as more than one sensor is
Figure 4.11: Location estimation using large number of sensors (7 sensors)

used in the estimation. The error value converges so quickly because every sensor for this simulation has perfect RSS measurements. If more complex impairments such as multipath or shadow fading are introduced, Figure 4.12 of estimated distance against the number of sensors used in position would resemble Figure 4.13.

Figure 4.13 shows that multipath or shadow fading errors reduce the algorithm’s performance from the error estimates in Figure 4.12. Note that all sensors were assigned a fading induced measurement error in Figure 4.13.

To summarize, the results of the above simulations confirm our predictions of the algorithm’s performance. The algorithm works well when certain conditions are met:

- *The number of sensors is large* - The larger the number of sensors, the more likely that more sensors will have a line of sight signal by employing spatial diversity of our sensor network. This would mean that the algorithm will weight out incorrect location estimates strongly and bring the location estimate closer to the true location of the responder.
Figure 4.12: Error of estimates with only non line of sight impairments

- **The grid resolution is high** - If the grid resolution is high, smaller distances can be resolved by the algorithm and more accurate grid points will be chosen as location estimates for the responder. This will, however, increase computation complexity and simulation time.

- **At least one sensor has line of sight** - LOS range measurements greatly increase the accuracy of our algorithm, as errors arising due to reflections are absent.

- **Pathloss exponents are known or can be estimated correctly** - The key to our geolocation algorithm working accurately in NLOS environments is estimating the pathloss exponents correctly. If they are estimated correctly, the NLOS errors in the TOA range measurements can be weighted out accurately and our location estimates will not deviate from the true location by very much.
4.4.7 Distance-Power Gradient Estimation Algorithm Evaluation

A MATLAB function has been written that evaluates the objective function described as per the algorithm. As expected the function tends to zero when the correct path loss exponent is estimated. The MATLAB optimization toolbox was used and the function `fminsearch` was employed. The Figure 4.14 below is generated by the optimization toolbox. The plot converges to zero when the PLE values are near the actual values.

The actual path loss exponents used in this simulation were 3, 2 and 2 respectively for sensors A, B and C respectively. The seed value was [1,1,1]. As can be seen from Figure 4.14, the minimization converges correctly. However, the minimization does not always converge correctly, as is the case for different seed values. This is because the objective function is non smooth and has many local minima, although the global minimum is what we desire.
4.5 User Interface Design

In order to test our overall system, we have designed and developed a graphical user interface that visually shows our system in action. It is designed so as to reflect the envisioned control system at the base station of our cooperative network.

4.5.1 User Requirements

When designing any GUI, it is important to know how the GUI should be designed, based on the user’s requirements. The GUI should be simple, but informative at the same time. It should be easy to use, and deliver information quickly and effectively to the user.

We have identified the following requirements in our GUI design:

- **Simplicity** - The GUI should be simple and visually clear.

- **Location visualization** - Responder’s locations should be shown in the field of operation.

- **Responder attribute visualization** - Transmissions that are received by the sensors should be stored in a database. This database should be accessible to the user.

- **Visual data filtering** - The user should be able to choose which team’s information is to be displayed. Specific responder’s information should be available as well.
• **Visualization of simulation time** - Simulation time should be displayed as the simulation is progressing.

• **Control of simulation parameters** - Simulation parameters such as the number of sensors employed, simulation length, environment type and the max number of teams cooperating should be user-specified

• **Visualization of algorithm performance** - Algorithm Statistics should also be displayed showing how well the algorithms performed in the selected environment
4.5.2 Element Layout

Figure 4.15 shows the layout of elements in the MATLAB GUI. The screenshot is taken from GUIDE (Graphical User Interface Development Environment) in MATLAB. The GUI is designed to have three axes. The first axes plots the WPI logo. The second axes is designed to plot the responders location. The third axes is designed to plot the spectral coherence function of the signal that is received at the receiver most recently. The UITable element displays data that is stored in the base station database. This data is accessible to the user. The table contains columns for the following information:

- **Time instant of signal reception** - This is the time instant in the simulation when a signal was received and decoded by our sensor network.

- **Team ID number** - The team identification number is embedded in the context data fields of the VRT packet.

- **Responder ID number** - The context fields of the received and decoded signal will also contain the identification number of the responder registered with the radio that has made the transmission.

- **X and Y Locations resolved from the signal** - The $x$ and $y$ coordinates describe the location of the responder with respect to the origin of our sensor networks coordinate grid.

- **Modulation Scheme detected** - This is simply the name of the modulation scheme that was found using the cyclostationary signal detector and data fusion blocks of our system.

There are three panels that group simulation controls. The first panel allows the user to select which team is to be displayed, and which responder in the selected team is to be displayed. The second panel shows the simulation statistics when the simulation is complete. The statistics shown are the average geolocation error in the simulation, and the percentage of modulation schemes that the signal detection and classification algorithm resolved correctly. The third panel is the main simulation control. The user enters numerical values into the edit boxes labelled, number of sensors, simulation time and number of teams. The user also chooses an environment type from rural, suburban or urban environments.
4.5.3 General Program Flow

The user begins using the GUI by calling:

sdrgui

in the MATLAB command window. The GUI is then created and displayed. The user is required to enter simulation parameters and hit run. If invalid parameters are entered, then the user is asked to enter non-negative simulation parameters. If correct parameters are entered, the GUI then disables all edit boxes and the drop down menu for channel type in the simulation controls panel and begins the simulation. The Simulation results panel will show 'Running...' indicating that results are still being computed. The simulation time will be shown above the 'Display' panel and will update with each time instant. If a transmission is received by the base station from the sensors, the database will show some data. By default, the database does not filter any data. If the user selects a team to be displayed and a responder within a team, that responder’s information including location estimate and modulation type will be shown in the graph and table respectively. The bottom right axes will show the spectral coherence features of the signal most recently received by the sensors. Therefore, the user can switch between teams and responders on the fly, while the simulation is running. This is what we envision the coordinating officer’s view will be like in a real system.

MATLAB’s GUIs are programmed with a very basic concept of event-driven programming called callbacks. Callbacks are functions that get fired when certain events occur. The listener ‘listens’ for these event cues, such as mouse clicks, keypresses and cursor movements and fires appropriate callbacks for these events. In our case., when the user hits the 'Run' button, the callback to be executed is the simulation script. In addition, our simulation script was modified for incorporation into the GUI. The main modification was to include a flag that is set when the GUI calls the script from the 'Run' button's callback. The script sees this flag and updates the tables and plots based on the user’s input into the GUI, in addition to simulating the environment and the algorithms. The final design is shown in Figure 4.16.
Figure 4.15: Overall Layout of GUI
4.6 Cyclic Detector

A large part of this project was the completion of the cyclic detector. The role of the cyclic detector, is to first find if there is a signal present, and if so, identify the signal. Due to the nature of our network, all the sensors are performing the same operations on the same data, so the this data must also be fused to make a singular decision. This duty is also the responsibility of the cyclic detector.

Of all the modules in this project, the cyclic detector is the most computationally intensive. On top of this, the detector is executed by each sensor, so the computations are executed many times for each transmission. Due to the high computational complexity of cyclic detector and the frequency of use, its performance must be maximized. This presented two options:

1. Use the MATLAB profiler to optimize the code
2. Use the Embedded MATLAB toolbox to compile the code

Both of these options provide techniques to improve the cyclic detector’s performance, but the magnitude of the performance gains and the complexity involved in implementing each solution differs.

Using the profiler to improve performance would be complex to implement and the magnitude of the performance gains would likely be small. The cyclic detector source code is delicate and was written by neither of the authors. Many of the values used in the code are hard coded, meaning that changing a particular hard coded value in one place could break the code in other places. Also, the re-implementing the cyclic detector would be very complicated as, there are multiple nested for loops with equally complex indices. Finally, the cyclic detector is computationally intensive by nature. Meaning that after re-implementing the code the performance gains would likely be small.

The second option, compiling the cyclic detector presents the possibility of much larger performance gains, but also presents implementation challenges of its own. However the potential performance gains are so large compared to re-implementing the cyclic detector, this was the approach taken.

4.6.1 Compilation

Initial tests show that compiling M Scripts can improve performance by a factor of 10. Compiling an M script with the Embedded MATLAB compiler isn’t necessarily straightforward
Figure 4.16: Final working GUI
however. Since embedded MATLAB is intended for embedded platforms, memory cannot be allocated from the heap, that is memory cannot be allocated dynamically. What this means is that the amount of memory that must be used must be known at compile time. For example, if one wishes to pass a vector to the cyclic detector, the size and type of that vector must be known when the script is being compiled. This is not the case with regular MATLAB at all, where one may whatever they wish to a script. With regular interpreted MATLAB all the memory allocation and data typing is hidden from the user.

Compiling with Embedded MATLAB is where these two paradigms collide and create problems for the user. Since the MATLAB type system is hidden from the user, the user can have difficulty manipulating these data types. MATLAB has a complex type structure. There are two major types, MATLAB types and native types. Native types are the underlying types of MATLAB. They are the types recognized by C/C++ compilers. Examples of these types are int, float, and double. There are also the MATLAB types. These are classes created for and used by MATLAB. When using interpreted MATLAB, these are types that MATLAB uses. MATLAB types are not recognized by C/C++ compilers.

A complicating matter is the use of the complex data types. For communicates the use of complex data, that is data with imaginary components, is essential. The cyclic detector relies upon complex data types. Complex data is stored in memory slightly different than data that belongs only in the real domain. This is because complex data has two components, a real component and an imaginary component. To store complex data, one needs to include both components leading to needing twice the memory space. In MATLAB this is hidden from the user, when the using declares, for example, a complex double, the fact that the two data types are in any way different is hidden.
This complicated matters greatly when compiling the cyclic detector. Since an complex signal is passed to the detector, the compiler must be told this explicitly, along with the size of the signal. Typically, the way to do this, is by adding a set of example inputs to the function in the build script. This is done like so -eg \{zeros(1, 100, 'double')\}. This code tells the Embedded MATLAB compiler, the size of the vector along with the data type. However the signal that is being passed is complex and the Embedded MATLAB compiler must be told this explicitly. There is also no complex data type. The same problem occurs while initializing data from within the script. Since all memory allocations must be known at compile time, the size of vector along with the data must be declared explicitly. Solving this problem was not straightforward, the solution was found by trial and error. The solution was simple however, data can be declared as complex, the example input to the cyclic detector were changed to -eg \{complex(zeros(1,1000, 'double'))\}

There were also other problems while compiling the cyclic detector. The system that MATLAB uses to hide data typing has a flaw when it comes to embedded MATLAB. Embedded MATLAB works by taking an M script, generating C code that performs the same function, and compilers that code using a C compiler on already on the machine. In this process there must be type translation. When the Embedded MATLAB binary is called in the MATLAB environment, a MATLAB type must be passed in the function call. Then the MATLAB type is converted to a native type and passed to compiled C code. Then the binary is executed and MATLAB collects the output. The output data types are native types and not converted to MATLAB types by MATLAB. So if one wishes to pass the outputs of the function back into the function iteratively, an error is given. The first solutions attempted were to attempt to type cast the data to a MATLAB type, as in Listing 4.2, these attempts

Listing 4.1: MATLAB Types

<table>
<thead>
<tr>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&gt;&gt; class(3+4j)</td>
</tr>
<tr>
<td>2</td>
<td>ans =</td>
</tr>
<tr>
<td>3</td>
<td>double</td>
</tr>
<tr>
<td>4</td>
<td>&gt;&gt; class(3)</td>
</tr>
<tr>
<td>5</td>
<td>ans =</td>
</tr>
<tr>
<td>6</td>
<td>double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>-eg {zeros(1, 100, 'double')}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>complex(zeros(1,1000, 'double'))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>-eg {complex(zeros(1,1000, 'double'))}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-eg {zeros(1,1000, 'double')}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>complex(zeros(1,1000, 'double'))</td>
</tr>
</tbody>
</table>
failed.

Listing 4.2: MATLAB Type Casting Attempts

```matlab
output = compiledFunction(input);
compiledFunction(output);    % fails

casted = double(output);
compiledFunction(casted);    % fails

casted = complex(output);
compiledFunction(casted);    % fails

casted = typecast(output, 'double');
compiledFunction(casted);    % fails

casted = typecast(output, 'complex');
compiledFunction(casted);    % fails

compiledFunction(output * 1); % succeeds
```

As one can see from Listing 4.2 explicitly casting in MATLAB always failed, but multiplying by one succeeded in converting `output` to a MATLAB type. The authors do not know why this technique works and the others fail. MATLAB is closed source, so the matter cannot be further investigated.

We suspect that, when multiplying by a cast performed implicitly, the cast converts the returned value, a native type, into a MATLAB type. The embedded MATLAB compiled functions can only make

### 4.6.2 Performance: Interpreted versus Compiled

Compiling the Cyclic detector, definitely improves performance. The magnitude of the performance gains however, must still be investigated. The magnitude of the performance gains on different architectures must also be measured, since different architectures are available for simulation.

In Appendix I there is the script used to compare the compiled and interpreted Cyclic Detector. The script works in the following way:
1. There are 200 trials.

2. The interpreted and compiled versions are run alternately.

3. The total time of each trial is measured using the MATLAB profiler.

4. Once all the trials are complete, the mean simulation time for the compiled and interpreted versions are found.

5. The result is printed to the user.

After the 200 trials on each architecture, the performance gains are as listed in Table 4.8

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Performance Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>i686</td>
<td>5.500180</td>
</tr>
<tr>
<td>x86_64</td>
<td>2.898874</td>
</tr>
</tbody>
</table>

There are certainly gains in performance. However, the magnitude of the gains are disappointing, especially for the 64 bit architecture. Although, the performance gains are not as high as expected, these tests show that using the compiled code will speed a simulation significantly.

4.7 System Simulation Design

This section of the project is where the preceding sections are all combined to make a functioning simulation. The integrated system can be thought of a flow with several different parts. There are several different challenges that lie in integrating part of the system flow. These challenges are generating the data, keeping track of the data, confirm that passing the data to and from the other systems properly, and determining the statistics of the system as a whole.

The approach taken in the systems integration stage is to precompute and then store as much data as possible, as opposed to generating data in mid-simulation. For example, the first responders who transmit at a given time are computed during the system initialization. Essentially, the system knows everything that will happen before the simulation starts. The generated data is all kept in an array of MATLAB data structures. This approach simplifies the process of each subsystem accessing the data, manipulating it, and then return its own results.
There are two categories of data that this simulation will use. First is the data generated at system initialization, which is the data described in the preceding paragraph. The other category is the data that the system will ascertain. This second category is essentially the system’s estimation of the first category. The estimate data is isolated from the first set of data, that is, it is kept in a different data structure entirely. As the simulation progresses the pertinent data is saved to this second data structure.

### 4.7.1 Simulation Data Generation

In this section the techniques used to generate the simulation data will be discussed. Whenever a simulation is made, there is much data to generate. All the data that describes the simulation must be generated. That is, data describing the environment must be generated, like the coordinates of the disaster zone. Data for the emergency responders must also be generated, like the number of emergency responder teams and emergency responders per team. These are just examples.

Luckily, a large portion of this data can be generated during the system initialization phase. Data is generated using three techniques. The first technique is just hard coding the data. For example, if the user decides that a rural environment will be simulated, a constant in the MATLAB script must be changed. Another technique is to generate data given some statistics about the data. For example, the number of emergency responders per team is uniformly generated given a minimum number of emergency responders per team and a maximum number of emergency responders per team. Finally, some data is generated from picking items from a list. An example of this picking the type of matched filter, there raised cosine filters and square root raised cosine filters, the filters used for each team are chosen from this list randomly. The techniques used to generate all the data is described in Listing 4.7.1

- Data that is hard coded:
  - Number of first response teams
  - Length of time to simulate
  - Disaster zone dimensions
  - Probability that a given emergency responder will transmit
  - Environment type
  - Noise power
- Doppler Offset
- Number of sensors

- Data that is generated randomly, given information about statistics
  - Number of people per team
  - Sets of center frequencies for each teams
  - Locations of each person in the simulation

- Data that is randomly chosen from a list
  - Modulation schemes for each team
  - Pulse shape
  - Roll off factor

When the simulation data is generated this way, each simulation is unique. However, the key variables can remain constant from simulation to simulation. For example, the number of people on each first response team will vary from simulation to simulation, but this will not make the geolocation system more or less accurate. However, manipulating the noise power will make the geolocation system more or less accurate.

### 4.7.2 System Evaluation

Once the simulation is complete, the system must be evaluated. With so much data generated and simulated, this could be a complicated process, however the approach chosen attempted to simplify the system evaluation as much as possible.

At the end of the simulation a script is called that takes both sets of data as an input and describes them. This script evaluates the system, producing the statistics necessary to see how well the system is working. The script produces two statistics that describe the effectiveness of the system:

- The mean error of the geolocation system
- The percentage of modulation schemes that the system finds correctly

The mean error calculated with Equation (4.25). In this equation, the actual x-y coordinates of each first responder are denoted as $x_{i,\text{real}}$ and $y_{i,\text{real}}$ where $i$ is the summation index.
The x-y coordinates that the system finds are denoted as $x_{i,\text{real}}$ and $y_{i,\text{real}}$, again $i$ represents the index:

$$
\mu_{\text{error}} = \frac{1}{n} \cdot \sum_{i=1}^{n} \sqrt{(x_{i,\text{real}}^2 - x_{i,\text{est}}^2) + (y_{i,\text{real}}^2 + y_{i,\text{est}}^2)},
$$

(4.25)

The percentage of correct modulation schemes calculation is more simple. For each emergency responder, there is a data kept about what modulation scheme they are using. The record exists as an estimate and the actual scheme used. Simply, the number of correct modulation scheme estimates is divided by the total number of emergency responders, as in Equation (4.26)

$$
M_{\text{correct}} = \frac{c_{\text{correct estimates}}}{n_{\text{responders}}},
$$

(4.26)

In Equation (4.26) $M$ is the percentage of correct modulation scheme estimates, $c$ is the number of correct estimates, and $n$ is total number of first responders.

These two statistics are the basis for evaluating the system as a whole. Although more statistics can be found about the system’s performance these two provide a quick and meaningful way to evaluate the system.

### 4.8 Timeline

In this project, goals were kept track of visually, with the aid of Figure 4.17. Each month, a new figure was generated highlighting goals that were reached that month and the noting the goals that had yet to be achieved.
Figure 4.18 shows an overall Gantt Chart of the MQP’s progress. The three milestones identified were:

- **Research various algorithms for each challenge** - A key milestone in this project was to gather enough resources and information to get the team started on the right track in terms of system design. Once proper research was completed, design goals were correctly identified.

- **Develop blocks that addresses the problems** - With our design goals in mind, complete working blocks that each address an individual challenge were developed. This is an important milestone because integration cannot take place if the individual blocks are not fully functional.

- **Integration of all the blocks to build a coherent system** - Once the individual blocks were working, our final milestone was reached when all blocks were integrated into graphical-user-interface that depicted our individual blocks working together in a system.

We have not encountered any serious delays in the progression of our MQP. The only change we had to make to our approach was to exclude the hardware set up for testing.
![Overall Gantt Chart](image)

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Start</th>
<th>Finish</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Design Channel, Detection System and Signal Detection and Classification System</td>
<td>9/13/2009</td>
<td>12/18/2009</td>
<td>3.4w</td>
</tr>
<tr>
<td>3</td>
<td>Evaluation and Design Modifications</td>
<td>10/3/2009</td>
<td>1/30/2009</td>
<td>5w</td>
</tr>
</tbody>
</table>
geolocation algorithm. The meta-data or context information of a packet sent from the USRP2 to the host controller could not be accessed due to the FPGA firmware not being up to date by the vendor at the time. This was needed to calculate the true TOA signature of the signal for the geolocation algorithm. Thus, we continued development of our system in the MATLAB environment, but at the same time keeping in mind that the code should be written such that if the firmware update is made available, our code should be easily ported to C/C++ for use with the USRP2.

Tasks were assigned based on technical background of each of the team members. Devin Kelly has extensive background in digital communication systems. Ishrak Khair is proficient in RF and Microwave propagation. As such, Devin Kelly was assigned the task of developing channel models for our simulations and algorithms. Ishrak Khair was assigned development and implementation of the geolocation system. Micheal Calabro has contributed significantly to the project through development of the cyclostationary signal classifier and data fusion algorithm.

4.9 Hardware Implementation

After preliminary research into implementing our system in hardware was done. In order for a successful implementation in hardware to occur several needs would have to be met by our platform, the USRP2. The needs are:

- Signal Strength
- Time of Signal Arrival
- Hardware stability needed for cyclostationary analysis.

4.9.1 Signal Strength Measurements

Upon investigation, our platform, the USRP2, can not currently meet these of these needs. The needs requires access to different parts of the USRP2 system. In order to capture the signal strength, access to the automatic gain controller is needed. This is because, when a signal is received by the USRP2, an unknown (to the user) gain is applied to the signal. The gain is applied as part of the automatic gain control (AGC). The purpose of AGC is to keep the signal in the dynamic range of the sampler. For example, if there were no gain added, then all the samples would quantized to the same value - losing information. Essentially, the gain is applied before sampling so the full dynamic range of the ADC can be used.
There are hooks into the some of the USRP2’s daughter cards to that allow for the AGC’s gain to be recorded. However, the software, that is GNU Radio, to access the hooks in the daughtercards is not mature. That is, the software provides only basic functionality with USRP2. In the future, GNU Radio will be mature enough to support received signal strength measurements, but for the duration of this project, this will not be an option.

### 4.9.2 Time of Arrival Measurements

Measuring the time of arrival provides some similar and some new challenges. It turns out that the GNU Radio can indirectly allow for the measurement of time of arrival. There are two problems with the provided approach though. The first is that it is not a direct method, meaning that USRP2 will not explicitly give the time of arrival. The time of arrival can be estimated based on timestamps the USRP2 writes onto packets that travel from the USRP2 to the host PC. Time of arrival can then be estimated if:

1. The processing delay in the USRP2 is constant.
2. The processing delay in the USRP2 is known.

The other challenge is that for the host PC to interpret these timestamps an interpretation of a signal handler is needed. That is, GNU Radio does not provide an implementation of the signal handler needed to read these time stamps. We feel that implementing the needed software lies outside the scope of this project.

### Hardware Stability

The cyclostationary analysis developed is integral to our spectrum sensing network, for the network to function properly, the cyclostationary analysis must function properly. Unfortunately, the higher order statistics needed for the cyclostationary detector to succeed are sensitive to jitter in the receiver clock. In this case the clock in question is implemented in the FPGA of the USRP2. As a result of this, there is large performance gap for cyclostationary detector in simulation compared to hardware. In the simulation, the cyclostationary performs as expected, even when additive white Gaussian noise (AWGN) is added along with multipath fading. However, when the detector is applied to data captured using the USRP2 the clock jitter is large enough to disrupt the cyclostationary detector.

### 4.9.3 Overall Problems

All the problems with proceeding with a hardware implementation all relate to firmware for the USRP2. At this time, the USRP2 is still an immature platform, the firmware is missing
many key features and is a work in progress. As one can see, with more developed firmware all the problems stated above would be able to be implemented. However at the same time developing this firmware lies far outside the goals of this project.

4.10 Chapter Summary

The design approach implemented was chosen with the user in mind. All the simulation objects and tools made it easy for the user to simulate with default information. Simultaneously, the functions are provided which allow the user to customize the simulation as much as they wish. The user was kept in mind again with the design of the GUI. The GUI allows the user to keep track of all the statistics accumulated by the system.
Chapter 5

Experimental Results

5.1 Simulation

In this section, we present results that are generated from the simulation script and analyses of the results.

Our simulation can run with or without a GUI. As mentioned earlier, calling the simulation script with the GUI flag set to 0 runs the simulation without the GUI. We ran most of our simulations without the GUI for speed. Our algorithms are predicted to work well when the radio channel does not have too much multipath, or NLOS (non line of sight) conditions for the signal. Interference can be an issue if interfering transmissions are too strong and weaken the desired signal. Nonetheless, the signal detection algorithm can be quite robust to this impairment as it does not rely on any thresholding, but simply cyclic patterns. The geolocation results however, may be affected a lot more.

We present simulation runs for with several combinations of parameters. These conditions and results are listed in Table 5.1. The disaster zone is 2km by 2km. The sensors are placed with uniformly distributed random $x$ and $y$ locations bounded within the 2km by 2km zone.
### Table 5.1: Simulation Results

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Duration</th>
<th>(N_{\text{Sensors}})</th>
<th>(N_{\text{Teams}})</th>
<th>Channel</th>
<th>(\overline{\text{Err}}_{\text{location}}) (m)</th>
<th>Schemes Correct/Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>Rural</td>
<td>37.22</td>
<td>64/68</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>Rural</td>
<td>26.65</td>
<td>42/53</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>Rural</td>
<td>12.71</td>
<td>67/69</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>Rural</td>
<td>9.82</td>
<td>56/57</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>Suburban</td>
<td>50.45</td>
<td>34/40</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>Suburban</td>
<td>42.24</td>
<td>41/58</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>Suburban</td>
<td>33.32</td>
<td>40/56</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>Suburban</td>
<td>20.61</td>
<td>111/120</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>5</td>
<td>10</td>
<td>Urban</td>
<td>90.03</td>
<td>77/84</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>Urban</td>
<td>77.67</td>
<td>45/48</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>Urban</td>
<td>54.92</td>
<td>108/110</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>Urban</td>
<td>34.89</td>
<td>91/94</td>
</tr>
</tbody>
</table>

#### 5.2 Discussion

In this section, we discuss the results presented in Table 5.1. We also present some of the corresponding screen captures taken during each simulation run. The screen captures show how motion updates are shown on the graph for responder location. The base station database table is also updated in real-time.
Figure 5.1: Simulation capture at time instant 1 for the rural channel.
Figure 5.2: Simulation capture at time instant 2 for the rural channel.
Figure 5.3: Simulation capture at time instant 3 for the rural channel.
Figure 5.4: Simulation capture at time instant 4 for the rural channel.
Figure 5.5: Simulation capture at time instant 5 for the rural channel.
5.2.1 Rural Channel

For the rural channel, we see that as the number of sensors increases the average location error for all locations decreases. This is expected as more sensors are feeding back information to the base station to make a better informed decision on location. It must be noted that all sensors in the simulation have a non-line of sight condition, and this is why we see a 9m error even for 30 sensors in the rural channel/environment. The number of transmission modulations schemes identified correctly also get better as the number of sensors increase, although the increase is not very dramatic. This is because the cyclostationary detector is not affected by NLOS conditions or multipath. We note that 98% of modulation schemes are decoded correctly in the 4th scenario.

With five sensors in the rural area of operation, our average geolocation error is about 37.5m and the system identified 64 out of 68 signals transmitted correctly. The cyclostationary signal detector is working well in the rural area as we expect it. The geolocation algorithm however is not performing very well. By increasing the number of sensors to 10 sensors the geolocation error falls to 26.65m. This is because there are now more sensors to make a geolocation estimate and this the estimated location is closer to the true value. Increasing the number of sensors further, we reduce our geolocation error to 12.75m. Again, the larger number of sensors moves the location solution closer to the true solution. One may ask why the solution is weak even in a relatively good channel. The reason for this is that we have used a large step size for our geolocation grid in order to gain simulation speed, since our operational zone is 2000m by 2000m. We see that if under the given circumstances, we increase the number of sensors to 30, we obtain an average geolocation error of about 9m.

5.2.2 Suburban Channel

The suburban channel is more harsh and scatterers are distributed more frequently in the simulation. As such, we see that with 10 sensors the system generates an average location error of 50.45m. This is a relatively larger error than the one seen for similar conditions in the rural channel. This large error arises due to the first arriving path of the signal detected by the sensors is bouncing off many scatters before arriving. The increase in delay equates to a larger ranging error. However, as more sensors are deployed we observe our average location error reducing to 20.61m. As explained above, the signal detection and classification system is quite robust and operates well in the suburban environment. With 5 sensors the system decodes 85% of the schemes and with 30 sensors the system decodes 92.5% of the schemes correctly.
Figure 5.6: Simulation capture at time instant 1 for the suburban channel.
Figure 5.7: Simulation capture at time instant 4 for the suburban channel.
Deploying five sensors on to the field we obtain an initial geolocation error of 50.45m. This is a larger error for the same number of sensors as for the rural area. The reason for this is that this channel has more line-of-sight obstructions than the rural channel. Although these obstructions are physically static, as responders move to these areas their transmissions bounce of these objects before arriving at the sensors. This results in a larger flight time than that compared to line-of-sight flight path. Therefore, there is a disagreement between the actual range and estimated range measured using TOA by the sensor network. Nevertheless, increasing the number of sensors can mitigate this due to the believe factor. The believe factor will weight down these errors by weighting sensors that have NLOS obstructions appropriately. Using 10 sensors in this suburban environment now reduced the average geolocation error to 42.24m as expected. A further increase to 20 sensors reduces our average geolocation error to 33.32m. The geolocation error decreases indeed with an increase in the number of sensors. Finally with 30 sensors, we have an average geolocation error of 20.61m.

5.2.3 Urban Channel

The urban channel is the most harsh environment where Doppler shifts are very frequent, the channel response is rich in multipath and the density of scatterers/obstructions is very large. Therefore, we can see that we have a large error of 90.03m with only 5 sensors used. As we increase the number of sensors to 30, the location error is reduced significantly to 34.89m. This error may seem large even with 30 sensors. However, the reader is reminded that all sensors have one or more NLOS conditions and the error is quite small in relation to the 2km by 2km operational field. The modulation scheme detection system continues to perform relatively well, with an accuracy of 96.8% even in the harshest of environments.

With five sensors, the urban channel has an average geolocation error of 90.3m. This is a much larger number when compared to the errors of the previous scenarios. We obtain a larger error in this case due to the fact that we have a very multipath rich environment. In addition, this environment also has a high density of line-of-sight obstructions. Therefore, it is very likely that none of the sensors have a line-of-sight signal path from the transmitter. If we increase the number of sensors from five to 10, we observe an average geolocation error of 77.67m. The increase in number of sensors has reduced the average location error, but we are still not performing well. In emergency situations, an error of 77.67m would mean the difference between being inside a burning room and being outside of it. Further increasing the number of sensors to 20, our geolocation error drops to 54.92m. Again, this is a significant improvement in performance but we would like to have more accuracy in emergency situations. A final increase to 30 sensors brings the average geolocation error down
Figure 5.8: Simulation capture at time instant 2 for the urban channel.
Figure 5.9: Simulation capture at time instant 4 for the urban channel.
Figure 5.10: Simulation capture at time instant 4 for the urban channel.
Figure 5.11: Simulation capture at time instant 5 for the urban channel.
to 34.89m. For an urban environment this is a significant result in our perspective, given the area of coverage assumed by our sensor network. The following figures were generated as screen captures when the simulations were running.
5.3 Chapter Summary

In this chapter, we have presented the results of our simulations without the GUI, and some screenshots of our GUI running. We have discussed the causes of the errors seen in the simulations and evaluated the performance based on these errors. We observe that the geolocation accuracy increases with the number of sensors. The signal detection and classification system proved to be robust in multipath rich environments. We have shown the real-time nature of the graphical-user-interface by showing motion and automatic updates of tabular information as new information is available to the base station.
Chapter 6

Conclusion

Technologies developed as part of this project are essential to any communications system. So, many of the systems developed have been integral components to communications systems historically, such as the channel models or geolocation systems. However, some of the other systems developed are just beginning to grow, the VRT standard has only ratified only in the past year and shows tremendous promise as a transport layer protocol for software-defined radio.

Applications of these technologies are far ranging. Starting with public safety, technologies developed for this project have applications in distributed communications networks, spectrum sensing, spectrum usage evaluation, and cognitive radio.

The following goals were met in the project have been accomplished:

1. **Channel Models** – Channels models developed can accurately model channels of several environment types and are parameterizable.

2. **Geolocation** – The geolocation subsystem accurately locate emergency responders to 20 to 100 meters, depending on the channel.

3. **Modulation Identification** – The modulation subsystem can identify 75% to 90% of the modulation schemes, depending on the environment.

4. **VRT** – The VRT standard has been successfully implemented in the form of VRT Packet generator.

Each of these technologies has demonstrated the potential of future communications systems, with an emphasis on communication systems for emergency responders. Although, the primary goal of this project was to develop a distributed spectrum sensing network for
emergency responders, the long term goal to develop a distributed emergency responder integration network.

The developed spectrum sensing network only listens to emergency responders, providing data to centralized, relatively remote station. In future applications sensors will facilitate the communications between different groups of emergency responders using different radios. This project represents the first steps to accomplishing this long term goal.

The underlying challenge that this project and any future projects will address is that of interoperability. The transition from hardware radios to software radios presents more flexibility to the radio designer. In terms of interoperability, this is both positive and negative. Positive, because radios are more flexible to interoperate with one another, negative because there are so many new possibilities for radio.

6.1 Future Work

In this section, open problems and future work are discussed.

- **Data Sharing** – Data must be shared in a distributed network, but how to share that data and with what members of the network remains unanswered. If a user wishes to communicate with a member outside of their group, the physical layer parameters must be shared with that user in advance. Future work be allow data to be shared freely with those intended, yet kept from parties that are restricted.

- **System Integration** – Ideally, in a disaster environment, any emergency responder must be able to communicate with any another. But how this will be done is unclear. Sensors may be able to facilitate communications, but only if data can be shared properly. In the future SDRs may be able to communicate directly without prior knowledge of one another.

- **Identification of Different M-ary Modulation Schemes** – The current solution can only differentiate between frequency shift keying modulation and quadrature amplitude modulation. At this point, identifying BPSK over QPSK is not possible.

- **Hardware Implementation** – The design system performs as expected in simulation, but a real system must perform in real environments with real hardware. Realizing this hardware implementation is an important step in deploying a system simulation.
• **Alternative Network Architectures** – Research may need to be done in order to explore alternative network architectures. The architecture implemented is primitive unidirectional system, where sensors only communicate with the singular base station. A multi-tier approach may work better, or a system where sensors relay information to one another.

• **Lingual Translations** – During a disaster emergency responders can come from all over the planet, so not all emergency responders may speak the same language. An autonomous system to translate language would solve this.

• **Cyclic Detector Optimization** – The cyclic detector is overwhelmingly the most complex operation the developed system performs. On sensors that must operate in real time and be battery powered optimization is essential. The cyclic detector algorithm may need to be optimized or implemented in an application specific integrated circuit.

• **Integration Between GPS and Geolocation Triangulation** – The SDR of the future will have a GPS radio and the VRT standard accommodates GPS data. An integration between the GPS radios in the SDRs and the geolocation system will produce more accurate results.

• **Dynamic Channel Estimation Integration with the Geolocation System and Cyclic Detector** – The cyclic detector’s accuracy is heavily dependent on the characteristics of the channel. A slightly noisier channel than estimated may cause erroneous results. A sophisticated channel estimator would improve the cyclic detector’s accuracy.

All of these technologies will play imperative roles across the wireless industry, whether for communications system designers or communications systems users. Whether these users be emergency responders, industrial user, military personnel, or consumers.
Bibliography


Appendix A

VRT Generator Object
Contents

- VRT Packet Generator
- Private Variables
- Public Variables
- Methods
- Constructor for the vrt_gen object
- Sets the modulation scheme
- Adds random samples to the data payload
- Makes a new packet, with random data
- Make Packets Method, by calling makePkt
- Modulation Method
- Sets up for matched filter
- Method to Filter and Sample the signal
- Plots TD & FD info for mixed signal
- Demodulates Signal
- Sets fs, fc

VRT Packet Generator

WPI Smart Radio Challenge Filename vrt_gen.m Authors: Devin Kelly

Description:
This is the class definition for the vrt_gen.m
This has been done to the GNU Radio Spec.

classdef vrt_gen < handle

Private Variables

properties (SetAccess = private)
    % hdr/trailer
    hdr;
    trailer;

    % items in the header
    word0;
psize;
stream = uint16(0);
int_ts;
frac_ts_0;
frac_ts_1;
count;

% object characteristics
nSamples;
nPkts;
pkt;
pkts;

% variables used to make the payload
payload;
lpf;

% modulation items
symbols;
dsymbols;
M;
N;
fs;
ts;
fc;
time;
upsample;
mixed;

% matched filter items
filt_y;
filt_t;
filt_type;
filt_beta;
filt_Fd;
filt_Fs;
Public Variables

properties (SetAccess = public)
    scheme;
    h; %modulator object
    g; %demodulator object
    filtered_sig;

end

Methods

methods

Constructor for the vrt_gen object

function start = vrt_gen(word0, nSamples, nPkts, scheme)
    start.word0 = uint32(word0);
    start.nSamples = nSamples;
    start.nPkts = nPkts;
    start.psize = uint32(nSamples + 6);
    start.count = uint32(1);
    start.hdr = uint32(zeros(5,1));
    start.trailer = uint32(hex2dec('55555555'));
    start.hdr(2) = hex2dec('aaaaaaaa');
    start.pkts = uint32(zeros(start.nSamples+6,start.nPkts));
    start.payload = uint32(zeros(start.nSamples,1));
    start.pkt = uint32(zeros(start.psize, 1));
    start.pkts(:,1) = start.pkt;
    start.lpf = 1;
    start.symbols = 0;
    start.dsymbols = 0;

start.filt_type = ''; 
start.filt_beta = 0; 
start.filt_Fd = 0; 
start.filt_Fs=0;

% this switch statement sets up all the variables for
% modulation and demodulation, depending on what
% the users chooses.
switch scheme
  case 'bpsk'
    start.scheme='bpsk';
    start.M = 2;
    start.N = 1;
    start.h = modem.qammod(start.M);
    start.g = modem.qamdemod(start.M);
  case 'qpsk'
    start.scheme = 'qpsk';
    start.M = 4;
    start.N = 2;
    start.h = modem.qammod(start.M);
    start.g = modem.qamdemod(start.M);
  case 'qam16'
    start.scheme = 'qam16';
    start.M = 16;
    start.N = 4;
    start.h = modem.qammod(start.M);
    start.g = modem.qamdemod(start.M);
  case 'qam64'
    start.scheme = 'qam64';
    start.M = 64;
    start.N = 6;
    start.h = modem.qammod(start.M);
    start.g = modem.qamdemod(start.M);
  case 'psk8'
    start.scheme = 'psk8';
    start.M = 8;
start.N = 3;
start.h = modem.pskmod(start.M);
start.g = modem.pskdemod(start.M);

case 'psk16'
    start.scheme = 'psk16';
    start.M = 16;
    start.N = 4;
    start.h = modem.pskmod(start.M);
    start.g = modem.pskdemod(start.M);

case 'psk32'
    start.scheme = 'psk32';
    start.M = 32;
    start.N = 5;
    start.h = modem.pskmod(start.M);
    start.g = modem.pskdemod(start.M);

case 'pam4'
    start.scheme = 'pam4';
    start.M = 4;
    start.N = 2;
    start.h = modem.pammod(start.M);
    start.g = modem.pamdemod(start.M);

case 'pam8'
    start.scheme = 'pam8';
    start.M = 8;
    start.N = 3;
    start.h = modem.pammod(start.M);
    start.g = modem.pamdemod(start.M);

case 'pam16'
    start.scheme = 'pam16';
    start.M = 16;
    start.N = 4;
    start.h = modem.pammod(start.M);
    start.g = modem.pamdemod(start.M);

case 'pam32'
    start.scheme = 'pam32';
    start.M = 32;
\[ \text{start.N} = 5; \]
\[ \text{start.h} = \text{modem.pammod(start.M)}; \]
\[ \text{start.g} = \text{modem.pamdemod(start.M)}; \]
\[ \text{case} \{ \text{’fsk’, ’fsk4’} \} \]
\[ \quad \text{start.scheme} = \text{’fsk4’}; \]
\[ \quad \text{start.M} = 4; \]
\[ \quad \text{start.N} = 2; \]
\[ \text{otherwise} \]
\[ \quad \text{error(’bad scheme’)} \]
\[ \end \]
\end

Sets the modulation scheme

function \( \text{setScheme}(\text{obj}, s) \)
\[ \quad \text{obj.scheme} = s; \]
\end

adds random samples to the data payload

function \( \text{add_random_samples}(\text{obj}) \)
\[ \quad \text{S} = 10e8*\text{rand}(1, 2*\text{obj.nSamples}); \]
\[ \quad \% \text{32 bit samples -- an unsigned 32 bit is max 4e9} \]
\[ \quad \text{F} = \text{filter}(\text{obj.lpf}, 1, \text{S}); \]
\[ \quad \text{F} = \text{F}((\text{obj.nSamples}/2)+1:(3*\text{obj.nSamples})/2); \]
\[ \quad \text{obj.payload} = \text{uint32(F)}'; \]
\end

Makes a new packet, with random data

function \( \text{new_packet}(\text{obj}) \)
\[ \quad \text{obj.hdr}(1) = \text{uint32(bitor(bitshift...} \]
\[ \quad \text{(obj.word0, 16),obj.psize));} \]
\[ \text{for x = 1:4} \]
\[ \quad \text{if bitget(obj.count, x) } \sim= \text{0} \]
\[ \quad \quad \text{obj.hdr(1) = bitset(obj.hdr(1), 16+x, 1);} \]
end
end
tmp = typecast(now,'uint32');
obj.hdr(3) = tmp(1); % TSI word

% assumes TSF = Sample Count Timestamp
% dealing with TSF words, rollover

tmp=hex2dec('FFFFFFFF')-obj.hdr(5);
if tmp < obj.nSamples
    obj.hdr(4) = obj.hdr(4) + 1; % TSF HI word
    obj.hdr(5) = tmp; % TSF LO word
else
    obj.hdr(5) = obj.hdr(5) + obj.nSamples;
end

add_random_samples(obj);
obj.pkt = [obj.hdr; obj.payload; obj.trailer];
obj.pkts(:,obj.count) = obj.pkt';
obj.count = obj.count + 1;
end

Make Packets Method, by calling makePkt

function makePkts(obj)
    for x=1:obj.nPkts
        new_packet(obj);
    end
end

Modulation Method

function mod(obj)
    line = reshape(obj.pkts, 1, (obj.psize) .* obj.nPkts); % ready the bits
    tmp = reshape(dec2bin(line, 32), 1, ... 32*size(obj.pkts,1)*size(obj.pkts,2));
switch obj.scheme
    case {'qpsk', 'qam16', 'qam64'}
        obj.h = modem.qammod(obj.M);
        obj.g = modem.qamdemod(obj.M);
    case {'bpsk', 'psk8', 'psk16', 'psk32'}
        obj.h = modem.pskmod(obj.M);
        obj.g = modem.pskdemod(obj.M);
    case {'pam4', 'pam8', 'pam16', 'pam32'}
        obj.h = modem.pammod('M', obj.M);
        obj.g = modem.pamdemod('M', obj.M);
    case {'fsk', 'fsk4'}
        obj.h = -1;
        obj.g = -1;
    otherwise
        error('bad scheme')
end

if strcmp(obj.scheme,'fsk') || strcmp(obj.scheme, 'fsk4')

    % modulo tmp to see if the vector is the right size
    a = mod(length(tmp), obj.M-1);
    % if the length will cause the modulate
    % function to error, so pad with zeros
    % (if necessary)
    if a ~= 0
        tmp = [tmp 48*ones(1, (obj.M-1)-a)];
    end

    tmp2 = reshape((double(tmp)-48)', obj.M-1, ...
        length(tmp)/(obj.M-1));
    tmp2 = sum(tmp2,1);
    obj.symbols = fskmod(tmp2', obj.M, 2, 5, 10);
    % arg3: 2Hz Freq separation
    % arg4: samples per symbol
    % arg5: Fs
    % since these are hard coded,
% if you change something make
% sure you make the corresponding
% changes to fksdemod below

else

% modulo tmp to see if the vector is the right size
a = mod(length(tmp), log(obj.M)/log(2));
% if the length will cause the modulate
% function to error, so pad with zeros
% (if necessary)
if a ~= 0
    tmp = [tmp 48*ones(1, obj.N-a)];
end

obj.h.inputType = 'bit';
obj.symbols = modulate(obj.h, (double(tmp)-48));
end
end

Sets up for matched filter

% only call this once
function makePMF(obj, s, varargin)
    % for s:
    %
    % with the fdesign tool you make you pmf one of three ways:
    % you can specify the filter 4 ways:
    % 0) use the defaults
    % 1) specify stop band attenuation (Ast), Beta
    % 2) Filter order in symbols (Nsym), Beta
    % 3) filter order (N), Beta
    %
    % If you choose 1,2,3 you must specify the fields above
    %
    % There are a few options here:
    % 1) SRRC filter

104
% 2) RC filter
% I suggest using option number and using the defaults
% Ast (30dB for SRRC, 60dB for RC)
numvarargs = length(varargin);
% optional inputs are filter type,
% beta, Fd, Fs, in that order

maxArgs = 5;
if numvarargs > maxArgs;
    error(['Too Many Inputs for PMF requires no'... 
       ' more than ' maxArgs ' optional inputs']);
end

%magnitudeUnits = 'dB';
% you can specify the mag units, they can be:
% linear
% dB
% squared

% defaults:
optargs = {'Square Root Raised Cosine'
           % pulse shape
           2  % samples per symbol
           .5 % description is read only
           30 % stopband attenuation
           % OR filter order
           % OR symbol order
           true %NormalizedFrequency
        };

optargs(1:numvarargs) = varargin;
shape = optargs{1};
sps = optargs{2};
atenOrOrder = optargs{4};
beta = optargs{3};
% f.PulseShape = shape;
% Options for the shape
%'Raised Cosine'
%'Square Root Raised Cosine'
%'Gaussian' <-I don’t think we need this though

if s == 0
    f = fdesign.pulseshaping(sps, ...
        'Square Root Raised Cosine', ...
        'Ast,Beta', ...
        atenOrOrder, ...
        beta);
elseif s == 1
    f = fdesign.pulseshaping(sps, ...
        shape, ...
        'Ast,Beta', ...
        atenOrOrder, ...
        beta);
elseif s == 2
    f = fdesign.pulseshaping(sps, ...
        shape, ...
        'Nsym,Beta', ...
        atenOrOrder, ...
        beta);
elseif s == 3
    N = optargs{4}; %#ok<PROP>
    f = fdesign.pulseshaping(sps, ...
        shape, ...
        'N,Beta', ...
        N, ...
        beta); %#ok<PROP>
else

error('Arg 2 of makePMF invalid, must be <= 3');
end

obj.filt_y = design(f);
end

Method to Filter and Sample the signal

function vrtFilter(obj)
    if ~ishandle(obj.filt_y)
        error('You must first create the filter using makePMF');
    end

    obj.filtered_sig = filter(obj.filt_y, obj.symbols);
end

function viewFilt(obj)
    fvtool(obj.filt_y);
end

Plots TD & FD info for mixed signal

function plotMixed(obj)
    plotspec(obj.mixed, obj.ts);
end

Demodulates Signal

function demod(obj)

    if strcmp(obj.scheme,'fsk') || strcmp(obj.scheme, 'fsk4')
        y = fskdemod(obj.symbols, obj.M, 2, 5, 10);
        obj.symbols = y;
    else
        y = demodulate(obj.g, obj.symbols);
        magic = length(y)/(obj.psize * obj.nPkts);
        nWord = (size(y,1))/magic;
    end
y = sum(reshape(y', [magic nWord]) .* ...
    repmat(2.^(30:(-2):0)',[1 nWord]));
obj.dsymbols = reshape(y, obj.psize,length(y)...
    /obj.psize);
end
end

Sets fs, fc

function setFreq(obj, fs, fc, upsample)
    obj.fs = fs;
    obj.ts = 1/fs;
    obj.fc = fc;
    if(fs < 2*fc)
        error('fs must be at least 2*fc');
    end
    obj.upsample = upsample;
end
end %ends methods
end
Appendix B

Channel Object
Contents

- WPI Smart Radio Challenge
- Variables – Properties
- Methods
- Constructor for channel object
- Sets the channel environment type
- Calculates distance between RX and TX
- Find the power of a signal
- Set the signal manually
- Adds noise to the signal
- Get the output signal
- Calculates Pathloss (Very Slow Fading)
- Plot the Signal after Very Slow Fading
- Add slow fading
- Signal Plotter
- Set the Fast Fading Channel Parameters
- Calculate the fast losses (multipath fading)
- Signal Plotter for after Fast Fading

WPI Smart Radio Challenge

Filename:
    chan.m
Author:
    Devin Kelly
Description:
    This M file contains the code for channel model object

classdef chan < handle

Variables – Properties

properties (SetAccess = private)
    % Channel Properties
    initSignal;
    cVacuum;
index;
c;
wavelength;

% transmitter / receiver properties
dist_m;
dist_km;
% tx and rx are both structs with:
% coordinates, x
% coordinates, y
% rx power (mW)
% gain
% signal
tx;
rx;

% center frequency, sampling frequency, etc
fc;
fs;
fd;
ts;

% SNR
signal;
power;
powerdb;
SNR;

% Faded Signals
vslowL;
vslowLdb;
vslowSig;
slowL;
slowLdb;
slowSig;
fastL;
fastLdb;
fastSig;

% fast fading channel variables
ffchan;
ffchantype;

% Variables with respect to fading
fadedSignal;
outputSignal;

end % ends properties

properties (SetAccess = public)
    fdoppler;
    type;
    path_delays;
    path_gains_avg;
    rician_k_factor;
    rician_DPD; % direct path doppler
    rician_DPP; % direct path phase
end

Methods

methods

Constructor for channel object

function start = chan(tx, rx, fdoppler, type)
    % the two radios communicating
    start.rx = rx;
    start.tx = tx;
    start.dist_m;
    start.dist_km;

    % channel properties
start.type = type;
start.fdoppler = fdoppler;
start.cVacuum = 299792458; % speed of light in m/s
start.index = 1.003; % index of refraction for air
start.c = (start.cVacuum)/(start.index);

start.wavelength = start.c ./ start.tx.fc;
% c = wavelength * freq

% signal and signal properties
start.initSignal = tx.signal;
start.power;
end

Sets the channel environment type

function setType(obj, env_type)
    if (strcmp(env_type, 'rural') || ...
        strcmp(env_type, 'suburban') || ...
        strcmp(env_type, 'urban') || ...
        strcmp(env_type, 'underground'))
        obj.type = env_type;
        disp(['WARNING: you have changed the ’ ... 
            ’channel type, you must now ’ ... 
            ’re-compute every variable that ’ ... 
            ’depends on the channel type’]);
    else
        error([env_type ... 
            ’ is not a valid environment type’])
    end
end

calculates distance between RX and TX

function findDist(obj)
    % kilometers
    obj.dist_m = sqrt((obj.rx.x - obj.tx.x).^2 + ...
(obj.rx.y - obj.tx.y).^2);

% meters
obj.dist_km = obj.dist_m / 10e3;
end

Find the power of a signal

function findPower(obj)
    % Signals and Systems, Oppenheim and Willsky, Equation 1.3
    obj.power = 1/length(obj.signal) * sum(obj.signal.^2);
end

Set the signal manually

function setSig(obj, initialSignal)
    disp(['WARNING, you are changing the signal '...
        'to be passed through the channel ' ...
        'away from what was transmitted']);
    obj.initSignal = initialSignal;
end

Adds noise to the signal

function obj = addAWGN(obj, SNR)
    % SNR is the snr per sample, in dB.
    % assumes signal power is 0 dBW
    if isempty(obj.outputSignal)
        obj.outputSignal = awgn(obj.initSignal, ... SNR, 'measured', 'db');
    else
        obj.outputSignal = awgn(obj.outputSignal, SNR, ... 'measured', 'db');
    end
end

Get the output signal

function s = getOutputSignal(obj)
Calculates Pathloss (Very Slow Fading)

```matlab
function verySlow(obj)

% in dB or 1/d^n
switch obj.type
    case 'rural'
        obj.vslowL = 1/(obj.dist_m.^4);
        obj.vslowLdb = pow2db(obj.vslowL);
    otherwise
        obj.vslowLdb = -10 * log(obj.wavelength/ ... 
                        (2*pi*sqrt(2)*obj.dist_m))^2;
        obj.vslowL = db2pow(obj.vslowLdb);
end
end
```

Plot the Signal after Very Slow Fading

```matlab
function plotVS(obj)

if isempty(obj.vslowSig)
    error('Cannot plot a null signal')
end

disp(
    ['Plotting very slowly faded ' ... 
     'signal using Ts at receiver']);
plotspec(obj.vslowSig, 1/obj.rx.fs)
end
```

Add slow fading

```matlab
function slow(obj)

% Estimates for antenna heights
hm = 1.5;
ht = 4;
```
\% using the Hata Model...
\[ a = (1.1 \times \log(\text{obj.rx.fc}) - 0.7) \times \]
\[ \text{hm} - (1.56 \times \log(\text{obj.rx.fc}) - 0.8); \]

\texttt{if strcmp(obj.type, 'urban')}
\[ \text{obj.slowLdb} = (69.55 + \]
\[ 26.16 \times \log(\text{obj.rx.fc})/\log(10) - \]
\[ 13.82 \times \log(\text{ht})/\log(10) - a) + \]
\[ (44.9 - 6.55 \times \log(\text{ht})/\log(10)) \times \]
\[ \log(\text{obj.dist_km})/\log(10); \]
\texttt{obj.slowL = db2pow(obj.slowLdb);} 
\texttt{elseif strcmp(obj.type, 'suburban')}
\[ \text{obj.slowLdb} = (69.55 + \]
\[ 26.16 \times \log(\text{obj.fc})/\log(10) - \]
\[ 13.82 \times \log(\text{ht})/\log(10) - a) + \]
\[ (44.9 - 6.55 \times \log(\text{ht})/\log(10)) \times \]
\[ \log(\text{obj.dist_km})/\log(10) \]
\[ - 2 \times (\log(f/28)/\log(10))^2 - 5.4; \]
\texttt{obj.slowL = db2pow(obj.slowLdb);} 
\texttt{elseif strcmp(obj.type, 'rural')}
\[ \text{obj.slowLdb} = (69.55 + \]
\[ 26.16 \times \log(\text{obj.rx.fc})/\log(10) - \]
\[ 13.82 \times \log(\text{ht})/\log(10) - a) + \]
\[ (44.9 - 6.55 \times \log(\text{ht})/\log(10)) \times \]
\[ \log(\text{obj.dist_km})/\log(10) \]
\[ - 4.78 \times (\log(\text{obj.rx.fc})/\log(10))^2 + \]
\[ 18.33 \times \log(\text{obj.rx.fc})/\log(10) - 40.94; \]
\texttt{obj.slowL = db2pow(obj.slowLdb);} 
\texttt{end}

\texttt{L = obj.slowL;}

\% Rappaport, "Wireless Communications", equation 3.1
\% calculate the received power
\texttt{P_r = ((obj.tx.pwr)/100 * (obj rx gain) ...}
\texttt{* (obj.tx.gain) * obj.wavelength.^2) ...}
/ ((4*pi)^2 * obj.dist_km * L);
obj.rx.signal = obj.tx.signal * P_r;
end

Signal Plotter

function plotS(obj)
    if isempty(obj.slowSig)
        error('Cannot plot a null signal')
    end
    disp('Plotting slowly faded signal using Ts at receiver');
    plotspec(obj.slowSig, 1/obj.rx.fs)
end

Set the Fast Fading Channel Parameters

function setFFChan(obj, chantype, varargin)
    if (strcmp(chantype, 'rayleigh') || strcmp(chantype, 'rician'))
        obj.ffchantype = chantype;
    else
        error([chantype ' is not a valid channel type, try rayleigh or rician'])
    end

    nvargs = length(varargin);
    optargs{1} = [8e-6 2e-6 .5e-6]; % defaults
    optargs{2} = [.3 .18 .077]; % defaults
    optargs(1:nvargs) = varargin;

    obj.path_delays = optargs{1};
    obj.path_gains_avg = optargs{2};
end

Calculate the fast losses (multipath fading)

function fast(obj)
    if(strcmp(obj.ffchantype, 'rayleigh'))
        obj.ffchan = rayleighchan(obj.rx.ts, obj.fdoppler);
end
% rayleighchan properties
obj.ffchan.DopplerSpectrum = doppler.jakes;
obj.ffchan.PathDelays = obj.path_delays;
obj.ffchan.AvgPathGaindB = obj.path_gains_avg;
obj.ffchan.NormalizePathGains = 1;
obj.ffchan.StoreHistory = 0;
obj.ffchan.StorePathGains = 0;
obj.ffchan.ResetBeforeFiltering = 1;

else  \%\% ffchantype == rician
obj.ffchan = ricianchan(obj.rx.ts, ...
    obj.fdoppler, obj.rician_k_factor);

% rayleighchan properties
obj.ffchan.DopplerSpectrum = doppler.jakes;
obj.ffchan.PathDelays = [0 obj.path_delays];
obj.ffchan.AvgPathGaindB = [1 obj.path_gains_avg];
obj.ffchan.DirectPathDopplerShift = obj.rician_DPD;
obj.ffchan.DirectPathInitPhase = obj.rician_DPP;
obj.ffchan.NormalizePathGains = 1;
obj.ffchan.StoreHistory = 0;
obj.ffchan.StorePathGains = 0;
obj.ffchan.ResetBeforeFiltering = 1;
end

obj.ffchan.StoreHistory = 1;
obj.fastL = filter(obj.ffchan, obj.tx.signal);
obj.rx.signal = filter(obj.ffchan, obj.tx.signal);
end

Signal Plotter for after Fast Fading

function plotFS(obj)
    if isempty(obj.fastSig)
        error('Cannot plot a null signal')
    end

    disp('Plotting fast faded signal using Ts at receiver');
plotspec(obj.fastSig, 1/obj.rx.fs)
end

% ends methods
end

% ends classdef
end
Appendix C

Environment Object
WPI Smart Radio Challenge

Filename: env.m

Authors: Devin Kelly, Ishrak Khair

Description: This is the class definition for the environment object.

Contents

- Environment Class Definition
- Variables – Properties
- Methods
- Method to add base stations
- Method to add a Mobile Sensor manually
- Method to generate sensors automatically
- Method to define the disaster zone
- Method to add roads to environment
- Method used to plot the environment
- Method for transmitting
- Method for rural channels
- Method for suburban channels
- Method for urban channel

Environment Class Definition

classdef env < handle

Variables – Properties

    properties (SetAccess = private)
        x = 0;
        y = 0;
        type = ’’;
        max_bs = 0;
        max_ms = 0;
max_sensors = 0;

% these will be arrays of BS, Radio, Sensor objects
bs_arr;
ms_arr;
sensors;
road_arr;
nroads;
zone;

% Cyclostationary analysis
qpsk_precompute;
fsk_precompute;
qupsk_thress;
fsk_thress;
qupsk_map;
fsk_map;
end

properties(SetAccess = public)
    SCF;
    scfon;
end

Methods

methods

    function start = env(x,y,type, max_bs, max_ms, max_sensors)
        if (x > 0 && y > 0 && max bs > 0 ... 
            && max ms > 0 && max sensors > 0)

            % cyclostationary parameters

            % pre-compute all the signals for all the SNRs
            % genLib();
% Generate signals for each simulations
% or just load the library already computed
load library.mat

% setup a map so we can
% associate SNRs with the right variables
start.qpsk_map = containers.Map(...
    {10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20},...
    {qpsk_profile10, qpsk_profile11, ...}
    {qpsk_profile12, qpsk_profile13, ...}
    {qpsk_profile14, qpsk_profile15, ...}
    {qpsk_profile16, qpsk_profile17, ...}
    {qpsk_profile18, qpsk_profile19, ...}
    {qpsk_profile20});

start.fsk_map = containers.Map(...
    {10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20},...
    {fsk_profile10, fsk_profile11, ...}
    {fsk_profile12, fsk_profile13, ...}
    {fsk_profile14, fsk_profile15, ...}
    {fsk_profile16, fsk_profile17, ...}
    {fsk_profile18, fsk_profile19, ...}
    {fsk_profile20});

start.fsk_thresh = .5;
start.qpsk_thresh = .5;
load 'fski.mat';
load 'qpski.mat';

start.x = x;
start.y = y;
start.max_bs = max_bs;
start.max_ms = max_ms;
start.max_sensors = max_sensors;

start.road_arr = cell(2,30);
start.nroads = 0;

% Setup Sensors
% distribute them uniformly
tmp1=x/sqrt(max_sensors);
tmp2=y/sqrt(max_sensors);
start.sensors=zeros(2,max_sensors);

counter=1;
for i=tmp1:tmp1:tmp1*floor(x/tmp1)
    for j=tmp2:tmp2:tmp2*floor(y/tmp2)
        start.sensors(:,counter)=[i j];
        counter=counter+1;
    end
end
else
    %throw some type of error
    error('a parameter that must be greater than 0 is incorrect')
end

if (strcmp(type,'rural') == 1 || ...
    strcmp(type, 'urban') == 1 || ...
    strcmp(type, 'suburban'))
    start.type = type;
else
    error('bad type')
end

Method to add base stations

function addBaseStation(obj, bsList)
    if isempty(bsList)
        error('Entered invalid sensor list');
    else
        n_stations = length(bsList);
for i=1:n_stations
    if bsList(i).x > obj.x || bsList(i).y > obj.y
        error(['base station placed '...
            'outside environment']);
    end
    if n_stations+length(obj.bs_arr) > obj.max_bs
        error('Too many stations');
    end
end
obj.bs_arr = [obj.bs_arr bsList];
end

Method to add a Mobile Sensor manually

function addMobiles(obj, sensorList)
    if isempty(sensorList)
        error('Entered invalid sensor list');
    else
        obj.sensor_arr = sensorList;
    end
end

Method to generate sensors automatically

function genSensors(obj, n, sensorType, layout, varargin)
    numvarargs = length(varargin);
    if strcmp(sensorType, 'sensor') || ...
        strcmp(sensorType, 's')
        str = 's';
    elseif strcmp(sensorType, 'base station') || ...
        strcmp(sensorType, 'base_station') || ...
        strcmp(sensorType, 'bs')
        str = 'bs';
    elseif strcmp(sensorType, 'mobile station') || ...
        strcmp(sensorType, 'mobile_station') || ...
        strcmp(sensorType, 'ms')
        str = 'ms';
    else
        error('Invalid sensor type');
    end
end
str = 'ms';
end

switch layout
  case 'circle'
    if numvarargs > 2 || numvarargs < 0
      error(['TooManyInputs requires '...
            'no more than 4 optional inputs']);
    end
    %defaults: radius=100, center = [200 200]
    optargs = {100, [200 200]};
    optargs(1:numvarargs) = varargin;
    radius = optargs{1};
    center = optargs{2};
    a_x = center(1)+zeros(1,n);
    a_y = center(2)+zeros(1,n);
    b_x = radius*sin(linspace(0,2*pi, n));
    b_y = radius*cos(linspace(0,2*pi, n));
    nv = 0.975:0.0001:1.025;
    nv1 = length(nv);
    stream0 = RandStream('mt19937ar','Seed',0);
    RandStream.setDefaultStream(stream0);
    noise = randsrc(2,n,...
                     [nv; 1/nv1*ones(1,length(nv))]);
    c_x = a_x + b_x;
    c_y = a_y + b_y;
    c_x = noise(1,:).*c_x;
    c_y = noise(2,:).*c_y;
    arr = struct('x', -1, 'y', -1);
for i = 1:n
    struct_element = [str num2str(i)];
    struct_element = ...
    struct('x', c_x(i), 'y', c_y(i));
    arr(i) = struct_element;
end

case 'square'
    optargs = {100, [200 200]};
    optargs(1:numvarargs) = varargin;
    width = optargs{1};
    center = optargs{2};

    if n ~= 4 && n ~= 8
        error(['Bad number of points ' ...
               'for a square, only 4 and ' ...
               '8 are supported']);
    end
end

a_x = center(1)+zeros(1,n);
a_y = center(2)+zeros(1,n);

b_x(1) = width*2/sqrt(2);
b_y(1) = width*2/sqrt(2);
b_x(2) = width*-2/sqrt(2);
b_y(2) = width*2/sqrt(2);
b_x(3) = width*-2/sqrt(2);
b_y(3) = width*-2/sqrt(2);
b_x(4) = width*2/sqrt(2);
b_y(4) = width*-2/sqrt(2);

if n == 8
    b_x(5) = width;
b_y(5) = width;
b_x(6) = width;
Method to define the disaster zone

function defineZone(obj, zoneType, varargin)
    numvarargs = length(varargin);
    switch zoneType
        case 'circle'
            if numvarargs > 2 || numvarargs < 0

b_y(6) = -width;
b_x(7) = -width;
b_y(7) = -width;
b_x(8) = -width;
b_y(8) = width;
end

c_x = a_x + b_x;
c_y = a_y + b_y;

arr = struct('x', -1, 'y', -1);

for i = 1:n
    struct_element = [str num2str(i)]; %#ok<*NASGU>
    struct_element = ...
        struct('x', c_x(i), 'y', c_y(i));
    arr(i) = struct_element;
end

if strcmp(str, 's')
    obj.sensors = arr;
elseif strcmp(str,'bs')
    obj.bs_arr = arr;
elseif strcmp(str, 'ms')
    obj.ms_arr = arr;
end
end
Error(['TooManyInputs requires no ', ... 
'more than 4 optional inputs']);
end

%%defaults: radius=100, center = [200 200]
optargs = {100, [200 200]};
optargs(1:numvarargs) = varargin;
radius = optargs{1};
center = optargs{2};

obj.zone = zeros(360,2);
for i = 1:360
    obj.zone(i,1) = center(1) + ... 
        radius*cos((i*2*pi)/360);
    obj.zone(i,2) = center(2) + ... 
        radius*sin((i*2*pi)/360);
end
end

Method to add roads to environment

function addRoad(obj, startPoint, endPoint)
    obj.nroads=obj.nroads+1;
    N=obj.nroads;
    % just to keep the code below easier to work with

    d_x = abs(startPoint.x-endPoint.x);
    d_y = abs(startPoint.y-endPoint.y);
    l = max([d_x d_y]);

    obj.road_arr{1,N} = ... 
        linspace(startPoint.x,endPoint.x, l);
    obj.road_arr{2,N} = ... 
        linspace(startPoint.y,endPoint.y, l);
end
Method used to plot the environment

```matlab
function plotEnv(obj)

% plot sensors
scatter([obj.sensors(:,).x], [obj.sensors(:,).y],...
    'green', '+')
axis equal

% plot base stations
hold on
scatter( [obj.bs_arr(:,).x], [obj.bs_arr(:,).y], ...
    'blue', 's');
hold off

% KEY for plots
% o - base station
% + - sensors

% draw disaster zone
hold on
plot(obj.zone(:,1), obj.zone(:,2), 'red');
hold off

hold on
for i = 1:obj.nroads;
    plot(obj.road_arr{1,i}, obj.road_arr{2,i}, 'black')
end
hold off

% draw the entire plot
xlim([0 obj.x]);
ylim([0 obj.y]);

% label it
xlabel('Coordinates for X direction');
ylabel('Coordinates for Y direction');
switch obj.type
    case 'rural'
        title('Rural Environment Plot')
end
```

130
case 'suburban'
title('Suburban Environment Plot')
case 'urban'
title('Urban Environment Plot')
case 'underground'
title('Underground Environment Plot')
otherwise
error('Environment Type set incorrectly')
end

end

Method for transmitting

function [out s] = transmission(obj, tx_info, chan)
% Description:
% This is a function implementation for
% getting the symbols for a first
% responder. This function takes a
% struct with the following fields:
%   info.scheme
%   info.shape
%   info.beta
%   info.pkts
%   info.samples
%   info.beta
%   info.sps   % samples per symbol
%   c.envType;  % rural, urban, etc
%   c.SNR
%   c.dop
%   tx # transmitter coordinates
%   rx # receiver (sensor) coordinates
%   This function returns a vector
with the desired symbols at baseband
and the center frequency

NOTE: Tx and Tx must have:

  TX.x
  TX.y
  TX.gain
  TX.sfc
  TX.fs
  TX.pwr

v=vrt_gen(hex2dec('5490'), tx_info.samples, tx_info.pkts, ...
     tx_info.scheme);
makePktso(v);
mod(v);
makePMF(v, 3, 'Square Root Raised Cosine', 8, .5, 40);
vrtFilter(v);
ix_info.signal = v.filtered_sig;

decisions = zeros(1,length(obj.sensors));
con = zeros(1,length(obj.sensors));

handle = str2func(chan.envType);
for i=1:length(obj.sensors)
    rx_coords.x = obj.sensors(i).x;
    rx_coords.y = obj.sensors(i).y;
    % it might be much of a pain to
    % figure out how much space
    % to allocate before hand
    RX = handle(obj, tx_info, ...
        rx_coords, chan.dop);

    if obj.scf
        % cyclostationary analysis on received signal
        [S Cx f_profile d c] = feval(obj.SCf, ...
RX.signal(1 : 1000)*1, ...
obj.fsk_map(chan.SNR)*1, ...
obj.qpsk_map(chan.SNR)*1, ...
1,1); %#ok<ASGLU>

decisions(i) = d;
con(i) = c;
if(max(con)==c)
    CxSave = Cx;
end
end
end
end

[dfinal cfinal] = fusion(decisions, con);
s=dfinal;

% geolocation block
grid.XMax = tx_info.center(1)+tx_info.pv;
grid.YMax = tx_info.center(2)+tx_info.pv;
grid.XMin = tx_info.center(1)-tx_info.pv;
grid.YMin = tx_info.center(2)-tx_info.pv;
grid.step = 0.5;
responderlocation = Alg_block(RX, grid);

% return a structure, w
% For now, add geolocation
% results to out struct. Add fc and
% scheme once Signal Classification
% is integrated. These will
% be used for plotting on the GUI.
out.location.x = responderlocation.TOA_POS.x;
out.location.y = responderlocation.TOA_POS.y;
if obj.scfon
    out.Cx = CxSave;
end
end
Method for rural channels

function ret = rural(obj, tx, rx, varargin) %#ok<MANU>
% default optional arguments
optargs = {10, 5, -1};
% f_doppler = 10 Hz, default
% SNR = 5, default

nargs = length(varargin);
optargs(1:nargs) = varargin;

[f_doppler, SNR, other_opt] = optargs{:};

c = chan(tx, rx, f_doppler, 'rural');
findDist(c);
findPower(c);
addAWGN(c, SNR);

RX = struct;
RX.signal = getOutputSignal(c);
% set condition if user chooses LOS
% impairments, and add LOS measurement
% noise for toa
RX.toa = tx.tod + sqrt((rx.x-tx.x)^2 + ...
    (rx.y-tx.y)^2) / 2.981e8 + 1e-13*randn();
RX.tod = tx.tod;
RX.x = rx.x;
RX.y = rx.y;
ret = RX;
end

Method for suburban channels

function ret = suburban(obj, tx, rx, varargin) %#ok<MANU>
% default optional arguments
optargs = {80, 4, 5e-8, 5};
%% f_doppler = 10 Hz, default
%% SNR = 5, default

nvargs = length(varargin);
optargs(1:nvargs) = varargin;

[f_doppler, SNR, other_opt] = optargs{:};

c = chan(tx, rx, f_doppler, 'suburban');
findDist(c);
findPower(c);
addAWGN(c, SNR);

RX = struct;
RX.signal = getOutputSignal(c);
%% set condition if user chooses NLOS
%% impairments, and add NLOSE measurement
%% noise for toa
RX.toa = tx.tod + sqrt((rx.x-tx.x)^2 + ...
  (rx.y-tx.y)^2) / 2.981e8 + 1e-9*randn();
RX.tod = tx.tod;
RX.x = rx.x;
RX.y = rx.y;
ret = RX;
end

Method for urban channel

function ret = urban(obj, tx, rx, varargin) %#ok<MANU>
  % default optional arguments
  optargs = {120, 2, 5e-6, 9};
  % f_doppler = 120 Hz, default
  % SNR = .5

  nvargs = length(varargin);
optargs(1:nvargs) = varargin;
[f_doppler, SNR, delay_spread, npaths] = optargs{:};

c = chan(tx, rx, f_doppler, 'urban');
findDist(c);
findPower(c);
addAWGN(c, SNR);

RX = struct;
RX.signal = getOutputSignal(c);
% set condition if user chooses NLOS
% impairments, and add NLOSE measurement
% noise for toa
RX.toa = tx.tod + sqrt((rx.x-tx.x)^2 + ...
    (rx.y-tx.y)^2) / 2.981e8 + 3.3e-8*randn();
RX.tod = tx.tod;
RX.x = rx.x;
RX.y = rx.y;
ret = RX;
end

end %ends methods

end %ends classdef
Appendix D

Geolocation Script
function Target = Alg_block(SensFeedback, AlgParams)

%RSS-BF WEIGHTED TOA localization

XMax = AlgParams.XMax;
YMax = AlgParams.YMax;
XPoints = AlgParams.XMin:AlgParams.step:XMax;
YPoints = AlgParams.YMin:AlgParams.step:YMax;

[XGrid YGrid] = meshgrid(XPoints, YPoints);

% PLE = AlgParams.PLE;

for ntarget = 1:SensFeedback.NTargets
    SumSquaredError_TOA = 0;
    SumSquaredError_RSS = 0;
    SumSquaredError_H = 0;
    PLE_RangeMatrix = [];
    for nsensor = 1:length(SensFeedback)
        TOF = SensFeedback(nsensor).toa-SensFeedback(nsensor).tod;
        EstDistTOA = TOF*2.981e8;
        sensx = SensFeedback(nsensor).x;
        sensy = SensFeedback(nsensor).y;
        CandidateGridDistances = sqrt((XGrid-sensx).^2 + (YGrid-sensy).^2);
        % TOA
    end
end
SumSquaredError_TOA = SumSquaredError_TOA + (CandidateGridDistances-EstDistTOA).^2; % % RSS % %
for PLE_idx = 1:length(PLE_Array)
    Pr = SensFeedback.Sensors(nsensor).Target(ntarget).RSS;
    Pt = 10*log10(SensFeedback.Sensors(nsensor).Target(ntarget).TransmitPower);
    PLE_Range = 10^((Pt-Pr)/(10*PLE));
    SumSquaredError_RSS = SumSquaredError_RSS+(CandidateGridDistances-PLE_Range).^2;
end
EstDistRSS = PLE_Range;
SumSquaredError_RSS = SumSquaredError_RSS+(CandidateGridDistances-EstDistRSS).^2;

%BF
if EstDistTOA>EstDistRSS
    BF = 1 - (abs(EstDistTOA-EstDistRSS)/EstDistTOA);
else
    BF = 1 - (abs(EstDistRSS-EstDistTOA)/EstDistRSS);
end
SumSquaredError_H = SumSquaredError_H + (CandidateGridDistances-EstDistTOA).^2;

%TOA
RMSE_TOA = sqrt(SumSquaredError_TOA/length(SensFeedback));
[RowIdx ColIdx] = minel(RMSE_TOA);
Target.TOA_POS.x = XGrid(RowIdx, ColIdx);
Target.TOA_POS.y = YGrid(RowIdx, ColIdx);

%RSS

% RMSE_RSS = sqrt(SumSquaredError_RSS/SensFeedback.NSensors);
% [r c] = minel(RMSE_RSS);
% Target(ntarget).RSS_POS.x = XGrid(r, c);
% Target(ntarget).RSS_POS.y = YGrid(r, c);
%
% %BF-Weighted TOA
%
% RMSE_H = sqrt(SumSquaredError_H/SensFeedback.NSensors);
% [rh ch] = minel(RMSE_H);
% Target(ntarget).BFTOA_POS.x = XGrid(rh, ch);
% Target(ntarget).BFTOA_POS.y = YGrid(rh, ch);

% Target(ntarget).FinalAvg.x = Target(ntarget).TOA_POS.x;
% Target(ntarget).FinalAvg.y = Target(ntarget).TOA_POS.y;
Appendix E

Geolocation Helper
WPI Smart Radio Challenge

Filename:
    minel.m
Author:
    Ishrak Khair
Description:
    This M file contains the code used as a helper for the Alg_block script

function [row, col]=minel(ErrorMatrix)
    % return the row and col indices of the minimum error that correspond to the
    % indices of the XGrid element and YGrid element

    RowMins = min(ErrorMatrix);
    [Min ColIdx] = min(RowMins);
    [Min RowIdx] = min(ErrorMatrix(:, ColIdx));

    row = RowIdx;
    col = ColIdx;
Appendix F

Spectral Correlation Function
WPI Smart Radio Challenge

Filename:
   get_SCF.m

Author:
   Micheal Calabro, Devin Kelly, Ishrak Khair

Description:
   This M file contains the code
   implements the Spectral Correlation Function

function [S Cx f_profile signal con] = ...
   get_SCF(tmp,fski,qpski,Cthfsk,Cthqpsk)
   %#eml

   % get_SCF
   % Input parameters:
   %    x    signal
   %    fski ideal vector for FSKi
   %    initialize as something random -
   % and then pre-compute
   %    qpski ideal vector for QPSK profile -
   %    initialize as something random,
   % and then pre-compute
   %    Cthfsk - Threshold for FSK detection
   %    Cthqpsk - Threshold for QPSK detection

   %External EML function
   eml.extrinsic('corrcoef', 'sum', 'load');%, 'hamming');

   x = tmp;
   y = x;
   %N must be even and divisible by 4 and < lx
   N = 512;
   lx=1000;
   signal=-1;
   con=0;
   %If no Cth specified, default to these
These should be tweaked during network start up
for best performance. The worse the channel,
the lower they should be. In an ideal channel,
they should be closer to 0.8

if(nargin==3)
    Cthfsk = 0.5;
    Cthqpsk = 0.5;
end

n=0:lx-N;
ln=lx-N+1;

Compute windowing functions for later.
The hamming function isn’t available
for eml, so we have to copy of the vectors
a=double(feval('hamming',N));
g=double(feval('hamming',ln));
g=g/sum(g);
a=a/sum(a);

g=[0.000303490136571
   0.000303634779272
   0.000304068683397
   0.000304791777017
   0.000305803940262
   0.000307105005342
   0.000308694756577
   0.000310572930428
   0.000312739215544
   0.000315193252813
   0.000317934635423
   0.000320962908925
   0.000324277571313
   0.000327878073106
   0.000331763817436
   0.000335934160153}
0.000340388409925
0.000345125828357
0.000350145630113
0.000355446983045
0.000361029008331
0.000366890780621
0.000373031328191
0.000379449633103
0.000386144631375
0.000393115213156
0.000400360222912
0.000407878459613
0.000415668676940
0.000423729583483
0.000432059842960
0.000440658074439
0.000449522852564
0.000458652707794
0.000468046126643
0.000477701551935
0.000487617383061
0.000497791976241
0.000508223644804
0.000518910659457
0.000529851248582
0.000541043598522
0.000552485853888
0.000564176117861
0.000576112452509
0.000588292879109
0.00060071538474
0.000613377891288
0.000626278318446
0.000639414521404
0.000652784322533
0.000666385505480

146
<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000680215815534</td>
</tr>
<tr>
<td>0.000694272960001</td>
</tr>
<tr>
<td>0.000708554608586</td>
</tr>
<tr>
<td>0.000723058393774</td>
</tr>
<tr>
<td>0.000737781911227</td>
</tr>
<tr>
<td>0.000752722720184</td>
</tr>
<tr>
<td>0.000767878343859</td>
</tr>
<tr>
<td>0.000783246269857</td>
</tr>
<tr>
<td>0.000798823950589</td>
</tr>
<tr>
<td>0.000814608803696</td>
</tr>
<tr>
<td>0.000830598212471</td>
</tr>
<tr>
<td>0.000846789526303</td>
</tr>
<tr>
<td>0.000863180061105</td>
</tr>
<tr>
<td>0.000879767099768</td>
</tr>
<tr>
<td>0.000896547892607</td>
</tr>
<tr>
<td>0.000913519657818</td>
</tr>
<tr>
<td>0.000930679581938</td>
</tr>
<tr>
<td>0.000948024820313</td>
</tr>
<tr>
<td>0.000965552497570</td>
</tr>
<tr>
<td>0.000983259708090</td>
</tr>
<tr>
<td>0.001001143516494</td>
</tr>
<tr>
<td>0.001019200958127</td>
</tr>
<tr>
<td>0.001037429039552</td>
</tr>
<tr>
<td>0.001055824739042</td>
</tr>
<tr>
<td>0.001074385007085</td>
</tr>
<tr>
<td>0.001093106766888</td>
</tr>
<tr>
<td>0.001111986914886</td>
</tr>
<tr>
<td>0.00113102321258</td>
</tr>
<tr>
<td>0.001150209830446</td>
</tr>
<tr>
<td>0.001169546261676</td>
</tr>
<tr>
<td>0.001189028409489</td>
</tr>
<tr>
<td>0.001208653044267</td>
</tr>
<tr>
<td>0.001228416912775</td>
</tr>
<tr>
<td>0.001248316738693</td>
</tr>
<tr>
<td>0.001268349223167</td>
</tr>
<tr>
<td>0.001288511045348</td>
</tr>
</tbody>
</table>
0.002093490289383
0.002115946912375
0.002138392364167
0.002160822923907
0.002183234873210
0.002205624496780
0.002227988083017
0.002250321924640
0.002272622319300
0.002294885570190
0.002317107986661
0.002339285884835
0.002361415588213
0.002383493428284
0.002405515745137
0.00242747888063
0.002449379216163
0.002471213098953
0.002492976916961
0.002514667062332
0.002536279939422
0.002557811965399
0.002579259570830
0.002600619200279
0.002621887312894
0.002643060382995
0.002664134900655
0.002685107372285
0.002705974321215
0.002726732288264
0.002747377832320
0.002767907530907
0.002788317980753
0.002808605798354
0.002828767620535
0.002848800105009
0.002868699930928
0.002888463799435
0.00290808434214
0.002927570582026
0.002946907013257
0.002966094522445
0.002985129928817
0.003004010076815
0.003022731836617
0.003041292104661
0.003059687804151
0.003077915885575
0.003095973327209
0.003113857135613
0.003131564346133
0.003149092023389
0.003166437261765
0.003183597185885
0.003200568951095
0.003217349743935
0.003233936782598
0.003250327317400
0.003266518631231
0.003282508040007
0.003298292893113
0.003313870573846
0.003329238499844
0.003344394123519
0.003359334932475
0.003374058449929
0.003388562235117
0.003402843883701
0.003416901028169
0.003430731338223
0.003444332521169
0.003457702322299

150
0.003470838525257
0.003483738952415
0.003496401465228
0.003508823964593
0.003521004391194
0.003532940725842
0.003544630989815
0.003556073245180
0.003567265595121
0.003578206184246
0.003588893198899
0.003599324867461
0.003609499460642
0.003619415291768
0.003629070717060
0.003638464135909
0.003647593991138
0.003656458769263
0.003665057000742
0.003673387260220
0.003681448166763
0.003689238384089
0.003696756620791
0.003704001630546
0.003710972212328
0.003717667210600
0.003724085515512
0.003730226063082
0.003736087835372
0.003741669860658
0.003746971213589
0.003751991015346
0.003756728433778
0.00376182683550
0.003765353026266
0.003769238770597

151
<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001074385007085</td>
</tr>
<tr>
<td>0.001055824739042</td>
</tr>
<tr>
<td>0.001037429039552</td>
</tr>
<tr>
<td>0.001019200958127</td>
</tr>
<tr>
<td>0.001001143516494</td>
</tr>
<tr>
<td>0.000983259708090</td>
</tr>
<tr>
<td>0.000965552497570</td>
</tr>
<tr>
<td>0.000948024820313</td>
</tr>
<tr>
<td>0.000930679581938</td>
</tr>
<tr>
<td>0.000913519657818</td>
</tr>
<tr>
<td>0.000896547892607</td>
</tr>
<tr>
<td>0.000879767099768</td>
</tr>
<tr>
<td>0.000863180061105</td>
</tr>
<tr>
<td>0.000846789526303</td>
</tr>
<tr>
<td>0.000830598212471</td>
</tr>
<tr>
<td>0.000814608803696</td>
</tr>
<tr>
<td>0.000798823950589</td>
</tr>
<tr>
<td>0.000783246269857</td>
</tr>
<tr>
<td>0.000767878343859</td>
</tr>
<tr>
<td>0.000752722720184</td>
</tr>
<tr>
<td>0.000737781911227</td>
</tr>
<tr>
<td>0.000723058393774</td>
</tr>
<tr>
<td>0.000708554608586</td>
</tr>
<tr>
<td>0.000694272960001</td>
</tr>
<tr>
<td>0.000680215815534</td>
</tr>
<tr>
<td>0.000666385505480</td>
</tr>
<tr>
<td>0.000652784322533</td>
</tr>
<tr>
<td>0.000639414521404</td>
</tr>
<tr>
<td>0.000626278318446</td>
</tr>
<tr>
<td>0.000613377891288</td>
</tr>
<tr>
<td>0.000600715378474</td>
</tr>
<tr>
<td>0.000588292879109</td>
</tr>
<tr>
<td>0.000576112452509</td>
</tr>
<tr>
<td>0.000564176117861</td>
</tr>
<tr>
<td>0.000552485853888</td>
</tr>
<tr>
<td>0.000541043598522</td>
</tr>
</tbody>
</table>
0.000305803940262
0.000304791777017
0.000304068683397
0.000303634779272
0.000303490136571]

a=[0.000289834069995
0.000289960049457
0.000290337968796
0.000290967770877
0.000291849360482
0.000292982604326
0.000294367331079
0.000296003331389
0.000297890357915
0.000300028125365
0.000302416310537
0.000305054552370
0.000307942451999
0.000311079572812
0.000314465440520
0.000318099543225
0.000321981331503
0.000326110218478
0.000330485579920
0.000335106754335
0.000339973043063
0.000345083710389
0.000350437983650
0.000356035053352
0.000361874073297
0.000367954160703
0.000374274396344
0.000380833824689
0.000387631454040
0.000394666256690
<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000401937169072</td>
</tr>
<tr>
<td>0.000409443091924</td>
</tr>
<tr>
<td>0.000417182890454</td>
</tr>
<tr>
<td>0.000425155394509</td>
</tr>
<tr>
<td>0.000433359398757</td>
</tr>
<tr>
<td>0.000441793662864</td>
</tr>
<tr>
<td>0.000450456911685</td>
</tr>
<tr>
<td>0.000459347835456</td>
</tr>
<tr>
<td>0.000468465089990</td>
</tr>
<tr>
<td>0.000477807296883</td>
</tr>
<tr>
<td>0.000487373043721</td>
</tr>
<tr>
<td>0.000497160884296</td>
</tr>
<tr>
<td>0.000507169338818</td>
</tr>
<tr>
<td>0.000517396894147</td>
</tr>
<tr>
<td>0.000527842004017</td>
</tr>
<tr>
<td>0.000538503089269</td>
</tr>
<tr>
<td>0.000549378538095</td>
</tr>
<tr>
<td>0.000560466706274</td>
</tr>
<tr>
<td>0.000571765917427</td>
</tr>
<tr>
<td>0.000583274463269</td>
</tr>
<tr>
<td>0.000594990603865</td>
</tr>
<tr>
<td>0.000606912567895</td>
</tr>
<tr>
<td>0.000619038552921</td>
</tr>
<tr>
<td>0.000631366725660</td>
</tr>
<tr>
<td>0.000643895222261</td>
</tr>
<tr>
<td>0.000656622148588</td>
</tr>
<tr>
<td>0.000669545580503</td>
</tr>
<tr>
<td>0.000682663564160</td>
</tr>
<tr>
<td>0.000695974116299</td>
</tr>
<tr>
<td>0.000709475224548</td>
</tr>
<tr>
<td>0.000723164847723</td>
</tr>
<tr>
<td>0.000737040916142</td>
</tr>
<tr>
<td>0.000751101331932</td>
</tr>
<tr>
<td>0.000765343969351</td>
</tr>
<tr>
<td>0.000779766675107</td>
</tr>
<tr>
<td>0.000794367268685</td>
</tr>
</tbody>
</table>
0.002853853778894
0.002871051955695
0.002888111846615
0.002905030872432
0.002921806475221
0.002938436118741
0.002954917288816
0.002971247493720
0.002987424264549
0.003003445155597
0.003019307744724
0.003035009633725
0.003050548448689
0.003065921840359
0.003081127484489
0.003096163082194
0.003111026360298
0.003125715071676
0.003140226995597
0.003154559938055
0.003168711732107
0.003182680238194
0.003196463344471
0.003210058967119
0.003223465050668
0.003236679568300
0.003249700522162
0.003262525943664
0.003275153893777
0.003287582463328
0.003299809773288
0.003311833975055
0.003323653250734
0.003335265813412
0.003346669907430
0.003357863808644
0.003368845824689
0.003379614295236
0.003390167592238
0.003400504120182
0.003410622316326
0.003420520650938
0.003430197627524
0.003439651783059
0.003448881688204
0.003457885947522
0.003466663199695
0.003475212117720
0.003483531409120
0.003491619816130
0.003499476115894
0.003507099120649
0.003514487677898
0.003521640670595
0.003528557017304
0.003535235672368
0.003541675626066
0.003547875904766
0.003553835571069
0.003559553723957
0.003565029498922
0.003570262068103
0.003575250640407
0.003579994461629
0.003584492814570
0.003588745019139
0.003592750432461
0.003596508448973
0.003600018500515
0.003603280056414
0.003606292623567
0.003609055746517
0.003611569007517
0.003613832026595
0.003615844461614
0.003617606008323
0.003619116400398
0.003620375409491
0.003621382845254
0.003622138555379
0.003622642425611
0.003622894379774
0.003622894379774
0.003622642425611
0.003622138555379
0.003621382845254
0.003620375409491
0.003619116400398
0.003617606008323
0.003615844461614
0.003613832026595
0.003611569007517
0.003609055746517
0.003606292623567
0.003603280056414
0.003600018500515
0.003596508448973
0.003592750432461
0.003588745019139
0.003584492814570
0.003579994461629
0.003575250640407
0.003570262068103
0.00356502948922
0.003559553723957
0.003553835571069
0.003547875904766
0.003541675626066

166
0.003535235672368
0.003528557017304
0.003521640670595
0.003514487677898
0.003507099120649
0.003499476115894
0.003491619816130
0.003483531409120
0.003475212117720
0.003466663199695
0.003457885947522
0.003448881688204
0.003439651783059
0.003430197627524
0.003420520650938
0.003410622316326
0.003400504120182
0.003390167592238
0.003379614295236
0.003368845824689
0.003357863808644
0.003346669907430
0.003335265813412
0.003323653250734
0.003311833975055
0.003299809773288
0.003287582463328
0.003275153893777
0.003262525943664
0.003249700522162
0.003236679568300
0.003223465050668
0.003210058967119
0.003196463344471
0.003182680238194
0.003168711732107
<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.002542921383403</td>
</tr>
<tr>
<td>0.002523696991279</td>
</tr>
<tr>
<td>0.002504386828588</td>
</tr>
<tr>
<td>0.002484993814763</td>
</tr>
<tr>
<td>0.002465520881763</td>
</tr>
<tr>
<td>0.002445970973628</td>
</tr>
<tr>
<td>0.002426347046036</td>
</tr>
<tr>
<td>0.002406652065857</td>
</tr>
<tr>
<td>0.002386889010703</td>
</tr>
<tr>
<td>0.002367060868475</td>
</tr>
<tr>
<td>0.002347170636919</td>
</tr>
<tr>
<td>0.002327221323165</td>
</tr>
<tr>
<td>0.002307215943275</td>
</tr>
<tr>
<td>0.002287157521790</td>
</tr>
<tr>
<td>0.002267049091268</td>
</tr>
<tr>
<td>0.002246893691829</td>
</tr>
<tr>
<td>0.002226694370693</td>
</tr>
<tr>
<td>0.002206454181721</td>
</tr>
<tr>
<td>0.002186176184952</td>
</tr>
<tr>
<td>0.002165863446141</td>
</tr>
<tr>
<td>0.002145519036298</td>
</tr>
<tr>
<td>0.002125146031217</td>
</tr>
<tr>
<td>0.002104747511020</td>
</tr>
<tr>
<td>0.002084326559681</td>
</tr>
<tr>
<td>0.002063886264570</td>
</tr>
<tr>
<td>0.002043429715980</td>
</tr>
<tr>
<td>0.002022960006660</td>
</tr>
<tr>
<td>0.002002480231351</td>
</tr>
<tr>
<td>0.001981993486314</td>
</tr>
<tr>
<td>0.001961502868864</td>
</tr>
<tr>
<td>0.001941011476902</td>
</tr>
<tr>
<td>0.001920522408446</td>
</tr>
<tr>
<td>0.001900038761163</td>
</tr>
<tr>
<td>0.001879563631900</td>
</tr>
<tr>
<td>0.001859100116216</td>
</tr>
<tr>
<td>0.001838651307915</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>0.001818220298575</td>
</tr>
<tr>
<td>0.001797810177087</td>
</tr>
<tr>
<td>0.001777424029181</td>
</tr>
<tr>
<td>0.001757064936963</td>
</tr>
<tr>
<td>0.001736735978450</td>
</tr>
<tr>
<td>0.001716440227102</td>
</tr>
<tr>
<td>0.001696180751358</td>
</tr>
<tr>
<td>0.001675960614174</td>
</tr>
<tr>
<td>0.001655782872558</td>
</tr>
<tr>
<td>0.001635650577108</td>
</tr>
<tr>
<td>0.001615566771552</td>
</tr>
<tr>
<td>0.001595534492285</td>
</tr>
<tr>
<td>0.001575556767915</td>
</tr>
<tr>
<td>0.001555636618799</td>
</tr>
<tr>
<td>0.001535777056592</td>
</tr>
<tr>
<td>0.001515981083788</td>
</tr>
<tr>
<td>0.001496251693266</td>
</tr>
<tr>
<td>0.001476591867840</td>
</tr>
<tr>
<td>0.001457004579807</td>
</tr>
<tr>
<td>0.001437492790496</td>
</tr>
<tr>
<td>0.001418059449823</td>
</tr>
<tr>
<td>0.001398707495843</td>
</tr>
<tr>
<td>0.001379439854307</td>
</tr>
<tr>
<td>0.001360259438217</td>
</tr>
<tr>
<td>0.001341169147391</td>
</tr>
<tr>
<td>0.001322171868019</td>
</tr>
<tr>
<td>0.001303270472230</td>
</tr>
<tr>
<td>0.001284467817655</td>
</tr>
<tr>
<td>0.001265766746999</td>
</tr>
<tr>
<td>0.001247170087607</td>
</tr>
<tr>
<td>0.001228680651041</td>
</tr>
<tr>
<td>0.001210301232648</td>
</tr>
<tr>
<td>0.001192034611147</td>
</tr>
<tr>
<td>0.001173883548200</td>
</tr>
<tr>
<td>0.001155850787999</td>
</tr>
<tr>
<td>0.001137939056851</td>
</tr>
</tbody>
</table>
Ts = 1/N;

% Pre-allocate for speed
S = complex(zeros(N+1,N/2+1));
X = complex(zeros(2*N+1,ln));
Y = complex(zeros(2*N+1,ln));

%Freq. Smoothed Cyclic Periodogram
for f = -N:N
    % N point FFTs of the signal are computed
    xf = x.*exp(-1i*2*pi*f*(0:lx-1)*Ts);
    yf = y.*exp(-1i*2*pi*f*(0:lx-1)*Ts);
    for i = 1:ln
        % Multiply the FFT of X with the conj of Y and vice versa
        n_r = n(i)+(1:N);
        X(f+N+1,i) = a'*xf(n_r)';
        Y(f+N+1,i) = conj(a'*yf(n_r)');
    end
end

for alpha = -N/4:N/4
for f=-N/2:N/2
    f1=f+alpha;
f2=f-alpha;
    if ((abs(f1)<N/2)&&(abs(f2)<N/2))
        \%g acts to smooth X*Y out, this is more obvious if you plot g 
        \%s is the cross correlation of X’s and Y’s frequency components
        \%seperated by f +/- alpha
        S(f+N/2+1,N/4+alpha+1)=g’*(X(f1+N+1,:).*Y(f2+N+1,:))’;
    end
end
end

\%Compute correlation coefficients

Cx = complex(zeros(513,513));
Cx = fftshift(corrcoef(S’).^2);
\%Extract feature vector region of interest
features = abs(Cx(1:200,240:280));
f_profile = complex(zeros(1,200));

a1=zeros(2,2);
b=zeros(2,2);
\%Take a snapshot of the most outstanding features in the region.
for f = 1:length(features(:,1))-1
    f_profile(f)=max(features(f,:));
end
\%Start Classification
\%Determine which profile most closely matches the signal.
a1 = abs(corrcoef(f_profile,fski));
b = abs(corrcoef(f_profile,qpsi));
\%Now determine if that match is sufficient to deem the signal present.
if a1(1,2) >= b(1,2)
    st = 1;
else
    st = 2;
end

174
end

switch st
    case 1
        if a1(1,2) >= Cthfsk
            signal = 1;
            con = a1(1,2);
        else
            signal = 0;
            con = a1(1,2);
        end
    case 2
        if b(1,2) >= Cthqpsk
            signal = 2;
            con = b(1,2);
        else
            signal = 0;
            con = b(1,2);
        end
end
Appendix G

Spectral Correlation Function Build Script
WPI Smart Radio Challenge

Filename:
   get_SCF_build.m
Author:
   Devin Kelly
Description:
   This M file contains the code used to build the get_SCF function
   
   close all;
   clear all;
   clc;

   [tmp, arch_type] = system('uname -m'); %#ok<ASGLU>
   clear tmp;

   disp('Starting Build Script, arch is')
   disp(arch_type)

   if strcmp(arch_type(1:4), 'i686') || strcmp(arch_type, 'i386')
     % generate the MEX function for 32 bit architectures
     emlc   get_SCF ...
     -report ...
     -T mex ...
     -eg {complex(zeros(1,1000, 'double')), zeros(1,200, 'double'), zeros(1,200,'double'), 1,1}
     -o ./get_SCF_mex.mexglx
   end

   if strcmp(arch_type(1:6), 'x86_64')
     disp('building')
     % generate the MEX function for 64 bit architectures
     emlc   get_SCF ...
     -report ...
     -T mex ...

177
-eg {complex(zeros(1,1000, 'double')), zeros(1,200, 'double'), zeros(1,200,'double'), 1, 1,
-o ./get_SCF_mex.mexa64
end

% generate the static library
%emlc euler01 ...
%   -report ...
%   -T rtw:lib ...
%   -o ./emcprj/rtwlib/euler01/euler01_mex.a
Appendix H

Channel Profile Generator
WPI Smart Radio Challenge

Filename:  
    genLib.m  
Author:  
    Devin Kelly  
Description:  
    This M file contains the code  
    generates the channel profile  
    library for get_SCF

function genLib()

    load fski.mat;
    load qpski.mat;

    open_system('datagen.mdl');  
    sim('datagen.mdl');

    f=str2func('get_SCF_mex');

    % get rid of MLINT warnings
    %#ok<*NASGU>

    [S Cx qpsk_profile10 signal con1] = ...
        feval(f, qpsk10', fski, qpski, .5, .5);

    [S Cx qpsk_profile11 signal con1] = ...
        feval(f, qpsk11', fski, qpski, .5, .5);

    [S Cx qpsk_profile12 signal con1] = ...
        feval(f, qpsk12', fski, qpski, .5, .5);

    [S Cx qpsk_profile13 signal con1] = ...
        feval(f, qpsk13', fski, qpski, .5, .5);
[S Cx qpsk_profile14 signal con1] = ...
feval(f, qpsk14', fski, qpski, .5, .5);

[S Cx qpsk_profile15 signal con1] = ...
feval(f, qpsk15', fski, qpski, .5, .5);

[S Cx qpsk_profile16 signal con1] = ...
feval(f, qpsk16', fski, qpski, .5, .5);

[S Cx qpsk_profile17 signal con1] = ...
feval(f, qpsk17', fski, qpski, .5, .5);

[S Cx qpsk_profile18 signal con1] = ...
feval(f, qpsk18', fski, qpski, .5, .5);

[S Cx qpsk_profile19 signal con1] = ...
feval(f, qpsk19', fski, qpski, .5, .5);

[S Cx qpsk_profile20 signal con1] = ...
feval(f, qpsk20', fski, qpski, .5, .5);

[S Cx fsk_profile10 signal con1] = ...
feval(f, fsk10', fski, qpski, .5, .5);

[S Cx fsk_profile11 signal con1] = ...
feval(f, fsk11', fski, qpski, .5, .5);

[S Cx fsk_profile12 signal con1] = ...
feval(f, fsk12', fski, qpski, .5, .5);

[S Cx fsk_profile13 signal con1] = ...
feval(f, fsk13', fski, qpski, .5, .5);

[S Cx fsk_profile14 signal con1] = ...
feval(f, fsk14', fski, qpski, .5, .5);
[S Cx fsk_profile15 signal con1] = ...  
feval(f, fsk15', fski, qpski, .5, .5);

[S Cx fsk_profile16 signal con1] = ...  
feval(f, fsk16', fski, qpski, .5, .5);

[S Cx fsk_profile17 signal con1] = ...  
feval(f, fsk17', fski, qpski, .5, .5);

[S Cx fsk_profile18 signal con1] = ...  
feval(f, fsk18', fski, qpski, .5, .5);

[S Cx fsk_profile19 signal con1] = ...  
feval(f, fsk19', fski, qpski, .5, .5);

[S Cx fsk_profile20 signal con1] = ...  
feval(f, fsk20', fski, qpski, .5, .5);

clear S Cx con1 f fsk fsk10 fsk11 fsk12 fsk13 fsk14 fsk15 fsk16 ...  
fsk17 fsk18 fsk19 fsk20 qpsk10 qpsk11 qpsk12 qpsk13 ...  
qpsk14 qpsk15 qpsk16 qpsk17 qpsk18 qpsk19 qpsk20 tout

save 'library.mat' *

der
Appendix I

Interpreted and Compiled Compare Script
WPI Smart Radio Challenge

Filename compareCompiled.m
Authors: Devin Kelly
Date: 2/26/2010

Description:
This is a script that compares the performance
of the compiled version and the interpreted version
the get_SCF function

The output of this script is the ratio of the
average run time for the interpreted script
and the average run time for the compiled script
for N trials.

clc;

N = 200; %number of trials
regular_times = zeros(1,N);
mex_times = zeros(1, N);

load fski.mat;
load qpski.mat;
load library.mat

% set parameters for test signal
fshape = 'Square Root Raised Cosine';

for i=1:N

    % generate different data each time
    clear vrt_qpsk;
    vrt_qpsk = vrt_gen(hex2dec('5490'), 10, 10, 'qpsk');
    makePkts(vrt_qpsk);
    mod(vrt_qpsk);
    makePMF(vrt_qpsk, 3, fshape, 4, .3, 40);
vrtFilter(vrt_qpsk);
qpsk_iq = vrt_qpsk.symbols;

profile on;
f=str2func('get_SCF');
profile('clear');
% M-script
[S Cx interp_out signal con1] = ... 
    feval(f, qpsk_iq(1:1000)', ...
    fski, qpski, .5, .5);
stats = profile('info');
regular_times(i) = ...
    sum([stats.FunctionTable(1).TotalTime]);

f=str2func('get_SCF_mex');
profile('clear');
% Mex Function
[S Cx mex_out signal con1] = ... 
    feval(f, qpsk_iq(1:1000)', ...
    fski, qpski, .5, .5);
stats = profile('info');
mex_times(i) = ...
    sum([stats.FunctionTable.TotalTime]);

profile('clear');
profile off
end
profile off;

m_reg = mean(regular_times);
m_mex = mean(mex_times);
ratio = m_reg/m_mex;

fprintf(['The ratio of regular M script to MEX'...'
' functions is %f
There were %d trials
'], ratio,N);
Appendix J

Testing Script
Contents

- WPI Smart Radio Challenge
- What to Test
- Test Modulating all the different schemes
- Test all the methods in vrt_gen
- Plot Constellations
- chan class functionality
- Testing the root-raised-filter
- Testing Very Slow Fading
- Testing Slow Fading
- Testing Fast Fading
- Testing Fs
- Creating multiple transmissions
- Testing add sensors to the environment
- A Test case for plotting an environment
- A test case for generating an urban environment with roads
- A Test case for generating a list of sensor structure
- A Test case for the getSys function call

WPI Smart Radio Challenge

Filename test.m
Authors: Devin Kelly

Description:
This is a script to show the various classes

clc
clear all;
clear;
close all;

What to Test

what are you testing?
VRT
1) modulating all the different schemes
2) all the vrt methods
3) plot constellations
 CHANNELS
4) Channel Obj
5) Adding Noise
6) Very Slow Fading
7) Slow Fading
8) Fast Fading (multipath)
9) Setting the sampling frequency
10) Multi-Transmission Simulation
11) rural test
12) urban test
13) Environment Testing
14) test placing radios
15) test the sensor generator
16) test adding roads
17) test plotting

testing = 10;

switch testing

case 1

Test Modulating all the different schemes

   vrt = vrt_gen(hex2dec('5490'), 100, 100, 'qpsk');
   makePkts(vrt);
   mod(vrt);

   vrt.scheme = 'qam16';
   mod(vrt);
   demod(vrt);
   setScheme(vrt, 'qam64');
mod(vrt);
demod(vrt);
setScheme(vrt, 'psk8');
mod(vrt);
demod(vrt);
setScheme(vrt, 'psk16');
mod(vrt);
demod(vrt);
setScheme(vrt, 'psk32');
mod(vrt);
demod(vrt);
setScheme(vrt, 'pam4');
mod(vrt);
demod(vrt);
setScheme(vrt, 'pam8');
mod(vrt);
demod(vrt);
setScheme(vrt, 'pam16');
mod(vrt);
demod(vrt);
setScheme(vrt, 'pam32');
mod(vrt);
demod(vrt);
setScheme(vrt, 'fsk');
mod(vrt);
demod(vrt);
disp('Success');

case 2

Test all the methods in vrt_gen

fshape = 'Square Root Raised Cosine';
vrts_qpsk = vrt_gen(hex2dec('5490'), 100, 100, 'qpsk');
makePktS(vrts_qpsk);
mod(vrts_qpsk);
makePMF(vrt_qpsk, 3, fshape, 4, .3, 40);
vrtFilter(vrt_qpsk);
demod(vrt_qpsk);
qpsk_iq = vrt_qpsk.symbols; %for the Is and Qs...
clear vrt_qpsk;
disp('Success');

case 3

Plot Constellations

vrt_qpsk = vrt_gen(hex2dec('5490'), 100, 100, 'qpsk');
makePkts(vrt_qpsk);
mod(vrt_qpsk);
qpsk_iq = vrt_qpsk.symbols; %variable names can’t begin with a number
clear vrt_qpsk;

vrt_16qam = vrt_gen(hex2dec('5490'), 100, 100, 'qam16');
makePkts(vrt_16qam);
mod(vrt_16qam);
qam16_iq = vrt_16qam.symbols; %variable names can’t begin with a number
clear vrt_16qam;

vrt_64qam = vrt_gen(hex2dec('5490'), 100, 100, 'qam64');
makePkts(vrt_64qam);
mod(vrt_64qam);
qam64_iq = vrt_64qam.symbols;
clear vrt_64qam;

figure(1);
scatter(real(qpsk_iq),imag(qpsk_iq))
title('Constellation for QPSK');
figure(2);
scatter(real(qam16_iq),imag(qam16_iq))
title('Constellation for 16-QAM');
figure(3);
scatter(real(qam64_iq),imag(qam64_iq))
title('Constellation for 64-QAM');

disp('Success');

case 4

chan class functionality

fs = 2700e6;
fc = 900e6;
beta = .4;

vrtp = vrt_gen(hex2dec('5490'), 100, 2, 'qam16');
makePkt(vrt);
mod(vrt);
setup(vrt, fc, fs, 5, 'sqrt', beta);
vrtrFilter(vrt);

RX.x = 120; % in meters
RX.y = 80; % in meters
RX.gain = .9;
RX.fc = fc;
RX.fs = fs;
RX.pwr = 100; % mW
RX.signal = -1;

TX.x = 460; % in meters
TX.y = 580; % in meters
TX.gain = .7;
TX.fc = fc;
TX.pwr = 1; % mW
TX.signal = vrt.filt_y;

fdopp = 80;
type = 'rural';
c = chan(TX, RX, fdopp, type);
findDist(c);
slow(c);

case 5

Testing the root-raised-filter

upsamp = 16;
fc = 900e6;
fs = upsamp * fc;
beta = .4;

vrt = vrt_gen(hex2dec('5490'), 100, 2, 'qam16');
makePkts(vrt);
mod(vrt);
setup(vrt, fc, fs, 5, 'sqrt', beta);
vrtFilter(vrt);

sig = vrt.filt_y;
sig_n = awgn(sig, .25, 'measured');
noise = sig_n - sig;

%find a slice of sig, about halfway through
N=round(.45*length(sig));
sig_slice = sig(N:N+50);
sig_n_slice = sig_n(N:N+50);
o noise_slice = sig_n_slice - sig_slice;

figure(1)
plot(sqrt(real(sig_slice).^2+imag(sig_slice).^2))
title('Original Signal');
figure(2)
plot(sqrt(real(noise_slice).^2+imag(noise_slice).^2))
title('Noise');
figure(3)
plot(sqrt(real(sig_n_slice).^2+imag(sig_n_slice).^2))
Testing Very Slow Fading

upsamp = 16;
fcc = 900e6;
fs = upsamp * fc;
beta = .4;

vrts = vrt_gen(hex2dec(‘5490’), 100, 2, ‘qam16’);
makePkts(vrts);
mod(vrts);
setup(vrts, fc, fs, 5, ‘sqrt’, beta);
vrtFilter(vrts);

RX.x = 120; % in meters
RX.y = 80; % in meters
RX.gain = .9;
RX.fc = fc;
RX.fs = fs;
RX.pwr = 100; % mW
RX.signal = -1;

TX.x = 460; % in meters
TX.y = 580; % in meters
TX.gain = .7;
TX.fc = fc;
TX.pwr = 1; % mW
TX.signal = vrt.mixed;

fdopp = 10;
type = ’rural’;

c = chan(TX, RX, fdopp, type);
findDist(c);
verySlow(c);

plotVS(c)

case 7

Testing Slow Fading

upsamp = 16;
fc = 900e6;
fs = upsamp * fc;
beta = .4;

vrt = vrt_gen(hex2dec('5490'), 100, 2, 'qam16');
makePkts(vrt);
mod(vrt);
setup(vrt, fc, fs, 5, 'sqrt', beta);
vrtFilter(vrt);

RX.x = 3500; % in meters
RX.y = 8000; % in meters
RX.gain = .9;
RX.fc = fc;
RX.fs = fs;
RX.pwr = 100; % mW
RX.signal = -1;

TX.x = 460; % in meters
TX.y = 580; % in meters
TX.gain = .7;
TX.fc = fc;
TX.pwr = 1; % mW
TX.signal = vrt.mixed;

fdopp = 10;
type = 'rural';
c = chan(TX, RX, fdopp, type);
findDist(c);
slow(c);

plotS(c)

case 8

Testing Fast Fading

t.x = 1000;
t.y = 700;
t.fc = 875e6;

r.x = 1100;
r.y = 900;
r.ts = 1800e-9;

upsamp = 16;
f.c = 900e6;
fs = upsamp * fc;
beta = .4;

vrt = vrt_gen(hex2dec('5490'), 100, 2, 'qam16');
makePkts(vrt);
mod(vrt);
setup(vrt, fc, fs, 8, 'sqrt', beta);
vrtFilter(vrt);
t.signal = vrt.filt_y;

c = chan(t, r, 15, 'rural');
setFFChan(c,'rayleigh', [0 8e-6 10e-6], [.3 .18 .077]);
fast(c);

case 9
Testing Fs

\[
\begin{align*}
fs &= 3600e6; \\
fc &= 915e6; \\
\beta &= .4; \\
vrt &= \text{vrt}\_\text{gen}(\text{hex2dec}('5490'), 100, 2, 'qam16'); \\
\text{makePkts}(vrt); \\
\text{mod}(vrt); \\
\text{setup}(vrt, fc, fs, 16, 'sqrt', \beta); \\
\text{vrtFilter}(vrt); \\
\text{plotMixed}(vrt);
\end{align*}
\]

\text{case 10}

Creating multiple transmissions

\[
\begin{align*}
fs &= 3600e6; \\
u &= 32; \\
\text{fshape} &= '\text{Square Root Raised Cosine}'; \\
\text{p1.header} &= \text{hex2dec}('5490'); \\
\text{p1.pktsize} &= 100; \\
\text{p1.npkkt} &= 2; \\
\text{p1.mod} &= 'qam16'; \\
\text{p1.fc} &= 915e6; \\
\text{p1.fs} &= fs; \\
\text{p1.upsample} &= u; \\
\text{p1.mftype} &= 'sqrt'; \\
\text{p1.beta} &= .6; \\
\text{p2.header} &= \text{hex2dec}('5490'); \\
\text{p2.pktsize} &= 100; \\
\text{p2.npkkt} &= 2; \\
\text{p2.mod} &= 'qpsk'; \\
\text{p2.fc} &= 870e6; \\
\text{p2.fs} &= fs; \\
\text{p2.upsample} &= u; \\
\text{p2.mftype} &= 'sqrt';
\end{align*}
\]
p2.beta = .4;

p3.header = hex2dec('5490');

p3.pktsize = 100;

p3.npkt = 2;

p3.mftype = 'sqrt';

vrt_obj = vrt_gen(p1.header, ...

   p1.pktsize, p1.npkt, p1.mod);
makePkts(vrt_obj);
mod(vrt_obj);
makePMF(vrt_obj, 3, fshape, 4, .3, 40);

vrt_obj = vrt_gen(p2.header, ...
   p2.pktsize, p2.npkt, p2.mod);
makePkts(vrt_obj);
mod(vrt_obj);
makePMF(vrt_obj, 3, fshape, 4, .3, 40);

vrt_obj = vrt_gen(p3.header, ...
   p3.pktsize, p3.npkt, p3.mod);
makePkts(vrt_obj);
mod(vrt_obj);
makePMF(vrt_obj, 3, fshape, 4, .3, 40);
% zero pad, make the matrices all the same size
tmp=zeros(length(pb_sig2)-length(pb_sig1),1);
a=[tmp; pb_sig1] pb_sig2 [tmp; pb_sig3]];

pb_sig = sum(a,2);
case 11

fs = 2e9;
u = 8;

p.header = hex2dec('5490');
p.pktsize = 100;
p.npkt = 2;
p.mod='qam16';
p.fc = 915e6;
p.fs = fs;
p.upsample=u;
p.mftype = 'sqrt';
p.beta=.6;

[pb_sig, time] = doEverything(p);

RX.x = 120; % in meters
RX.y = 80; % in meters
RX.gain = .9;
RX.fc = p.fc;
RX.fs = p.fs;
RX.pwr = 100; % mW
RX.signal = -1;

TX.x = 460; % in meters
TX.y = 580; % in meters
TX.gain = .7;
TX.fc = p.fc;
TX.pwr = 1; % mW
TX.signal = pb_sig;

fdopp = 5;

RX = rural(TX, RX, fdopp);

case 12
case 13
    disp('The null case, for now')
  case 14

Testing add sensors to the environment

myEnv = env(2000,1500,'rural', 4, 100, 50);

bs1.x = 200;
s1.x = 250;
bs1.y = 200;
s1.y = 750;
bs2.x = 1300;
s2.x = 500;
bs2.y = 200;
s2.y = 812.5;
bs3.x = 200;
s3.x = 750;
bs3.y = 1300;
s3.y = 1125;
bs4.x = 1300;
s4.x = 969;
bs4.y = 1300;
s4.y = 812.5;
bs_list = [bs1 bs2 bs3 bs4];
addBaseStation(myEnv, bs_list);

 bs1.x = 200;
s5.x = 1125;
bs2.x = 1300;
s5.y = 750;
bs3.y = 1300;
s6.x = 969;
bs4.y = 1300;
s6.x = 969;
s6.y = 468;
s7.x = 500;
s7.y = 375;
s8.x = 500;
s8.y = 468;
sList = [s1 s2 s3 s4 s5 s6 s7 s8];

addSensors(myEnv, sList);

case 15

A Test case for plotting an environment

    myEnv = env(2000,1500,'rural', 4, 100, 50);

    bs1.x = 200;
    bs1.y = 200;
    bs2.x = 1300;
    bs2.y = 200;
    bs3.x = 200;
    bs3.y = 1300;
    bs4.x = 1300;
    bs4.y = 1300;
    bs_list = [bs1 bs2 bs3 bs4];
    addBaseStation(myEnv, bs_list);

    s1.x = 500;
    s1.y = 950;
    s2.x = 725;
    s2.y = 1175;
    s3.x = 950;
    s3.y = 1400;
    s4.x = 1175;
    s4.y = 1175;
    s5.x = 1400;
    s5.y = 950;
    s6.x = 725;
A test case for generating an urban environment with roads

```matlab
r1s.x = 350;
r1s.y = 1500;
r1e.x = 350;
r1e.y = 500;
r2s.x = 400;
r2s.y = 1500;
r2e.x = 400;
r2e.y = 500;
r3s.x = 400;
r3s.y = 500;
r3e.x = 1200;
r3e.y = 500;
r4s.x = 400;
r4s.y = 450;
r4e.x = 1200;
r4e.y = 450;
r5s.x = 1200;
r5s.y = 500;
r5e.x = 1200;
r5e.y = 1500;
```
r14s.x = 0;
r14s.y = 450;
r14e.x = 350;
r14e.y = 500;

myEnv = env(2000,1500,'urban', 4, 100, 50);
genSensors(myEnv, 10, 's', 'circle', 300, [800 900]);
genSensors(myEnv, 4, 'bs', 'square', 300, [800 900]);
defineZone(myEnv, 'circle', 380, [800 900]);
addRoad(myEnv,r1s,r1e);
addRoad(myEnv,r2s,r2e)
addRoad(myEnv,r3s,r3e)
addRoad(myEnv,r4s,r4e)
addRoad(myEnv,r5s,r5e)
addRoad(myEnv,r6s,r6e)
addRoad(myEnv,r7s,r7e)
addRoad(myEnv,r8s,r8e)
addRoad(myEnv,r9s,r9e)
addRoad(myEnv,r10s,r10e)
addRoad(myEnv,r11s,r11e)
addRoad(myEnv,r12s,r12e)
addRoad(myEnv,r13s,r13e)
addRoad(myEnv,r14s,r14e)
plotEnv(myEnv)

case 17

A Test case for generating a list of sensor structure

myEnv = env(2000,1500,'rural', 4, 100, 50);
genSensors(myEnv, 10, 's', 'circle', 300, [800 1000]);
genSensors(myEnv, 4, 'bs', 'square', 300, [800 1000]);
defineZone(myEnv, 'circle', 400, [800 1000]);
plotEnv(myEnv)

case 18
A Test case for the getSys function call

```
in.scheme

in.shape = 'Square Root Raised Cosine';
in.scheme = 'qpsk';
in.beta = .4;
in.pkts = 10;
in.samples = 70; % samples per pkt
in.sps = 4;

s = getSyms(in);

end

format long e;
```
Appendix K

SDR GUI
WPI Smart Radio Challenge

Filename:
    sdrgui.m
Author:
    Ishrak Khair
Description:
    This M file contains the code
    for the SDR simulation GUI

%#ok<*NASGU>
%#ok<*INUSL>
%#ok<*INUSD>
% get rid of MLINT Warnings

function varargout = sdrgui(varargin)
% SDRGUI M-file for sdrgui.fig
% Authors: Ishrak Khair, Devin Kelly, Micheal Calabro and Alexander Wyglinksi
%
% sdrgui calls the gui for the SDR simulation. Enter the simulation parameters
% and run the simulation.

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;

if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before sdrgui is made visible.
function sdrgui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to sdrgui (see VARARGIN)

% Choose default command line output for sdrgui
handles.output = hObject;

% Update handles structure
setappdata(0,'frows',-1); %display everything
guidata(hObject, handles);
set(handles.axes1, 'Xlim', [0 2000], 'YLim', [0 2000], 'Alimmode', 'manual', 'nextplot',
xlabel(handles.axes1, 'X coordinate (m)');
ylabel(handles.axes1, 'Y coordinate (m)');

colordata = imread('logo.tiff');
image(colordata, 'Parent', handles.logo);
set(handles.logo, 'visible', 'off');
columns = {'Time','Team ID', 'Responder ID', 'X', 'Y', 'Modulation Scheme'};
cformats= {'numeric', 'numeric','numeric','numeric','char'};
set(handles.uitable1, 'ColumnName', columns, 'Columnwidth', 'auto','Columnformat', cformats);
channelchoices = {'rural','urban','suburban'};
set(handles.p_chantype, 'String',channelchoices);

% UIWAIT makes sdrgui wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = sdrgui_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
%

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on selection change in d_teamdisplay.
function d_teamdisplay_Callback(hObject, eventdata, handles)

% hObject handle to d_teamdisplay (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns d_teamdisplay contents as cell array
% contents{get(hObject,'Value')} returns selected item from d_teamdisplay

selected = get(hObject,'Value');
BSDatatmp = getappdata(0, 'BSData');
display_rows = find([BSDatatmp(:, 2)]==selected);
if isempty(display_rows)
    % handles.frows = display_rows;
    % guidata(hObject, handles);
    setappdata(0,'frows',display_rows);
    %set(handles.axes1, 'nextplot', 'replacechildren');

    %fbsdata = BSDatatmp(display_rows, :);
    %setappdata(0,'fbsdata',fbsdata);
    %guidata(hObject, handles);

208
```matlab
scatter(handles.axes1, [fbsdata{:,4}], [fbsdata{:,5}], 'marker', 'x');
set(handlesuitable1,'Data', BSDatamp(display_rows, :));

else
setappdata(0,'frows', -1);
end

% --- Executes during object creation, after setting all properties.
function d_teamdisplay_CreateFcn(hObject, eventdata, handles)
% hObject handle to d_teamdisplay (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

%handles.numsensors = inputdlg('Please enter the number of sensors in the simulation');

% Retrieve user specified sim params
nt = str2double(get(handles.e_nteams, 'string'));
sim_t = str2double(get(handles.e_simtime, 'string'));
nsens = str2double(get(handles.e_nsensors, 'string'));

if ( isnan(nt) || isnan(sim_t) || isnan(nsens))
    errordlg('Please enter valid numerical non-negative simulation parameters', 'WPI SDR
else
teamselects = num2str((1:nt)');
set(handles.d_teamdisplay, 'String', teamselects);
```
% Set the sdr script to run in gui mode
guiflag = 1;
set(handles.e_simtime, 'enable', 'off');
set(handles.p_chantype, 'enable', 'off');
set(handles.e_nsensors, 'enable', 'off');
set(handles.e_nteams, 'enable', 'off');
set(handles.errscheme, 'String', 'Running...');
set(handles.errlocs, 'String', 'Running...');
sdr;
errlocs = estlocs - actlocs;
errlocs = sqrt(errlocs.^2)/length(errlocs(:,1));

set(handles.e_simtime, 'enable', 'on');
set(handles.p_chantype, 'enable', 'on');
set(handles.e_nsensors, 'enable', 'on');
set(handles.e_nteams, 'enable', 'on');
if scfon
    set(handles.errscheme, 'String', ...
    [’% Schemes Correct: ’ num2str(100*schemesRight(1)/schemesRight(2))]);
end
set(handles.errlocs, 'String', [’ Average Location Error(m): ’ num2str(avgderr) ]);
end

function e_simtime_Callback(hObject, eventdata, handles)
% hObject       handle to e_simtime (see GCBO)
% eventdata     reserved - to be defined in a future version of MATLAB
% handles       structure with handles and user data (see GUIDATA)

% Hints: get(hObject,’String’) returns contents of e_simtime as text
% str2double(get(hObject,’String’)) returns contents of e_simtime as a double

210
function p_chantype_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to p_chantype (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: popupmenu controls usually have a white background on Windows.
    %   See ISPC and COMPUTER.

    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

% --- Executes during object creation, after setting all properties.
function uitable1_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to uitable1 (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

% --- Executes during object creation, after setting all properties.
function e_simtime_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to e_simtime (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: edit controls usually have a white background on Windows.
    %   See ISPC and COMPUTER.

    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

function e_nsensors_Callback(hObject, eventdata, handles)
    % hObject    handle to e_nsensors (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

211
function e_nteams_Callback(hObject, eventdata, handles)
    hObject    handle to e_nteams (see GCBO)
    eventdata reserved - to be defined in a future version of MATLAB
    handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of e_nteams as text
    %        str2double(get(hObject,'String')) returns contents of e_nteams as a double

% --- Executes on selection change in p_chantype.
function p_chantype_Callback(hObject, eventdata, handles)
    hObject    handle to p_chantype (see GCBO)
    eventdata reserved - to be defined in a future version of MATLAB
    handles    structure with handles and user data (see GUIDATA)

    % Hints: contents = get(hObject,'String') returns p_chantype contents as cell array
    %        contents{get(hObject,'Value')} returns selected item from p_chantype

% --- Executes on selection change in p_memberdisplay.
function p_memberdisplay_Callback(hObject, eventdata, handles)
    hObject    handle to p_memberdisplay (see GCBO)
    eventdata reserved - to be defined in a future version of MATLAB
    handles    structure with handles and user data (see GUIDATA)

    % Hints: contents = cellstr(get(hObject,'String')) returns p_memberdisplay contents as cell array
    %        contents{get(hObject,'Value')} returns selected item from p_memberdisplay
% --- Executes during object creation, after setting all properties.
function p_memberdisplay_CreateFcn(hObject, eventdata, handles)
% hObject    handle to p_memberdisplay (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function team_menu_CreateFcn(hObject, eventdata, handles)
% Blah

function edit1_Callback(hObject, eventdata, handles)

function edit4_Callback(hObject, eventdata, handles)
function nsensors_Callback(hObject, eventdata, handles)
Appendix L

SDR Simulation
WPI Smart Radio Challenge

Filename

sdr.m

Authors:

Devin Kelly, Ishrak Khair

Description:

This M script integrates all the blocks that have been written as part of this project. This script also controls the entire simulation and records all the data the simulation generates.

Contents

- Initialization
- Environment Creation
- Build the environment
- Build the teams
- Setup users array of structures
- Run the Simulation through each time instant

Initialization

scfon = 1;

Environment Creation

%define the disaster zone
dZoneCenter = [500, 500]; % just an arbitrary number for now
dZoneRadis = 200;
zone = [2000 2000]; % the overall area is zone

% Timing
% For now, we will lay this out as a series of Time-instants
% with an arbitrary length of time inbetween each instant
if (exist('guiflag', 'var'))
    if (guiflag ==1)
Ninstants = sim_t;
Nteams = nt;
n_sensors = nsens;
end
else
disp('No GUI')
Ninstants = 20;
Nteams = 10;
n_sensors = 30;
end
t = 1:Ninstants;

% Team Structure
% The team struct has these fields:
% nPeople
% fc []
% mod scheme
% array of coordinates of people on the team

% Initialize Variables

% setup spectrum sensing variables

% function handle for the get_SCF function
% can be either get_SCF (interpreted) or get_SCF_mex (compiled)
SCF = str2func('get_SCF_mex');

% setup team variables
maxPerTeam = 5;
minPerTeam = 1;

membersdisplay = num2str((1:maxPerTeam)');
if(exist('guiflag', 'var'))
set(handles.p_memberdisplay, 'String', membersdisplay);
end
ismLow = 902e6; % all in Hz
ismHigh = 928e6;
ismBand = 12.5e3;
nBands = floor(ismHigh-ismLow)/ismBand;
centerFrequencies = ...
    ismLow+ismBand/2:ismBand:ismHigh+ismBand/2;

txProb = 0.1;

peoplePerTeam = randi([minPerTeam maxPerTeam],1,Nteams);
people_cells = cell([Nteams, 1]);

% modelling the positions of people
% as clusters of people around random
% parts of the disaster zone, so
% clusters (Nteamsx2) is the location [x y]
% that each team will tend to clustered around
spread = 400;
centers = randn(2,Nteams);
clusters = [ zone(1)/2+randi([0 spread],1,Nteams); ...
            zone(2)/2+randi([0 spread],1,Nteams)]';

% so the same frequency isn’t picked twice
teamBands = centerFrequencies(randi([1 nBands],1,Nteams));
teams(1:Nteams) = struct;
% Nteams with a random number of people per team

% other variables
%schemes={'bpsk' 'qpsk' 'qam16' 'fsk'};
%scheme_probs = [.3 .3 .1 .3];
% schemeMap = containers.Map(schemes, {1, 2, 3, 4});
schemes={'qpsk' 'fsk'}; % we can only detect these two
scheme_probs = [.5 .5];
schemesRight = [0 0];
schemeMap = containers.Map({0, 1, 2},...
{'Undetected' schemes{:}};
pulse_shapes={'Square Root Raised Cosine' 'Raised Cosine'};
ps_probs=[.6 .4];% .1];
betas=[.3 .5 .7];
beta_probs=[.33 .34 .33];
pv = 150; % 'people variance'
%this is the spread of people locs around
%their clusters and it’s not
%really a variance

Build the environment

if(exist('guiflag', 'var'))
    env_type = get(handles.p_chantype, 'String');
    env_type = env_type{get(handles.p_chantype, 'Value')};
else
    env_type = 'rural';
end
f_handle = str2func(env_type);
% function handle for envionment
n_bases = 4; n_mobiles = 5;
x = 2000; y = 1500;
sim_env = env(x, y, env_type, n_bases, n_mobiles, n_sensors);
sim_env.SCF = SCF;
sim_env.scfon=scfon;
doppler_offset = 5; %hz
defineZone(sim_env,'circle', 300, [800 800]);
genSensors(sim_env, n_sensors, 'sensor', 'circle', 350, [800 800]);
genSensors(sim_env, n_bases, 'base_station', 'square', 280, [800 800]);
clear x y n_bases n_mobiles;
total_people = 0;

Build the teams

using a for loop seems to be the only way to do this

for i = 1:Nteams
teams(i).nPeople = peoplePerTeam(i);
teams(i).fc = teamBands(i);
teams(i).center_locs = clusters(i,:);

tmp1=min(sim_env.x, teams(i).center_locs(1));
% upperbound: individual xpos
tmp2=min(sim_env.y, teams(i).center_locs(2));
% upperbound: individual ypos
teams(i).individual_locs = ... 
    randi([tmp1-pv tmp1],1,teams(i).nPeople);...
    randi([tmp2-pv tmp2],1,teams(i).nPeople]);

% prob that a given person on a given
% team transmits at time Ninstant
teams(i).transmits = randsrc(teams(i).nPeople, Ninstants,...
    [0 1; 1-txProb txProb]);

teams(i).fc = randsrc(teams(i).nPeople, 1, ... 
    [centerFrequencies; ... 
    1/length(centerFrequencies)*...
    ones(1,length(centerFrequencies))]);

% pick a scheme, pulse shape, beta for each team
teams(i).scheme = schemes(randsrc(1,1,... 
    [1:length(schemes);scheme_probs]));
teams(i).pulse_shape = pulse_shapes(randsrc(1,1,... 
    [1:length(pulse_shapes);ps_probs]));
teams(i).beta = betas(randsrc(1,1, [1:length(betas);beta_probs]));
teams(i).samples = randsrc(1,1,[20:2:58;1/20*ones(1,20)]);
teams(i).sps = randsrc(1,1,[2 4 5 6; ones(1,4)*1/4]);
end
clear schemes scheme_probs pulse_shapes ps_probs;
clear peoplePerTeam teamBands clusters;
Setup users array of structures

users is an array of structures that this simulation will develop. This is the struct where the estimates of modulation scheme, location, etc are kept. The struct has the following fields:
fc modulation scheme x y

We use the index numbers to do the record keeping. That is, since we will need to compare this array of structs to the teams array of structures we will compare index to index to see how accurate our system is

Run the Simulation through each time instant

set the channel, assume that envType and max doppler offset don’t vary with time

c.envType=env_type;  % rural, urban, etc
c.SNR=15;            % 10 >= SNR >= 20
c.dop=doppler_offset;
entrycount = 1;
filtered_flag =0;
estlocs = [];
actlocs = [];

for i = 1:Ninstants
    for j = 1:Nteams
        teams(j).individual_locs = [...
            [randi([teams(j).center_locs(1)-pv ...
            teams(j).center_locs(1)+pv], 1, teams(j).nPeople); ...
            randi([teams(j).center_locs(2)-pv ...
            teams(j).center_locs(2)+pv], 1, teams(j).nPeople]);

for k=1:teams(j).nPeople
    tx_info.scheme = char(teams(j).scheme);
    tx_info.shape = char(teams(j).pulse_shape);
    tx_info.center = teams(j).center_locs;
% if the person wants to transmit at this time...
if teams(j).transmits(k,i)==1
% generate the VRT packet, pass it through the channel
Tx_info.fc = teams(j).fc(k);
Tx_info.beta = teams(j).beta;
Tx_info.pkts = randsrc(1,1,[10:2:58;1/25*ones(1,25)]);
Tx_info.samples = teams(j).samples;
Tx_info.sps = teams(j).samples;% samples per symbol
Tx_info.x = teams(j).individual_locs(1,k);
Tx_info.y = teams(j).individual_locs(2,k);
Tx_info.pv = pv;
Tx_info.tod = 1;

[BTData scheme] = transmission(sim_env, Tx_info, c);

scheme_detected = schemeMap(scheme);
BSData(entrycount,:) = {i, j, k, ...
    BTData.location.x, BTData.location.y,...
    scheme_detected};
actlocs= [actlocs ;[Tx_info.x Tx_info.y]];
entrycount = entrycount+1;

if(strcmp(Tx_info.scheme, scheme_detected))
    schemesRight(1)=schemesRight(1)+1;
    schemesRight(2)=schemesRight(2)+1;
else
    schemesRight(2)=schemesRight(2)+1;
end

fbsdata = BSData;
setappdata(0,'BSData', BSData);
if (exist('guiflag', 'var'))
    if guiflag
        % CHECK DROP DOWN MENU FOR SELECTED TEAM TO DISPLAY
        display_rows = find([BSData{:, 2}]==...
            get(handles.d_teamdisplay, 'Value'));
        if isempty(display_rows)
display_rows = -1;
end
setappdata(0,'frows',display_rows);
if (max(getappdata(0,'frows')) == -1 || ...
    max(getappdata(0, 'frows')) ...
    > entrycount-1)
    fbsdata = BSData;
    filtered_flag =0;
else
    fbsdata = BSData(display_rows, :);
    maxid = max(unique([fbsdata{:,3}]'));

    filtered_flag = 1;
end

if filtered_flag && (maxid >= ...
    get(handles.p_memberdisplay, 'Value'))
    person_locs = find([fbsdata{:, 3}]==...
        get(handles.p_memberdisplay, 'Value'));
    set(handles.axes1, ...
        'nextplot', 'replacechildren');
    if isempty(person_locs)
        xpoint = 0;
        ypoint = 0;
    else
        xpoint = [fbsdata{person_locs,4}];
        xpoint = xpoint(1,end);
        ypoint = [fbsdata{person_locs,5}];
        ypoint = ypoint(1,end);
    end
    s = scatter(handles.axes1, ...
        xpoint, ypoint, 'marker', 'x');
    set(s, 'markeredgcolor', 'yellow');
set(handles.uitable1,'Data', ...
    fbsdata(person_locs, :));
set(gcf, 'Currentaxes', handles.axes1);
text(xpoint+100, ypoint+100, ...
    {'Responder ' num2str(get(...
    handles.p_memberdisplay, 'Value'));
    ['X: ' num2str(xpoint)]; ...
    ['Y: ' num2str(ypoint)]}, ...
    'Color', 'red', ...
    'EdgeColor', 'red');
drawnow;
else

set(handles.axes1, 'nextplot', ...
    'replacechildren');
scatter(handles.axes1, ...
    [fbsdata(:,4)], [fbsdata(:,5)],...
    'marker', 'x');
set(handles.uitable1,'Data', fbsdata);
drawnow;
end

if scfon
    h1 = surf(handles.axes2, abs(BTData.Cx));
    set(h1, 'EdgeColor', 'interp');
    if getappdata(0,'frows') ~= -1
        set(handles.uitable1,'Data',...
            BSData(getappdata(0,...
                'frows'), :));
    else
        set(handles.uitable1,...
            'Data', BSData);
    end
    drawnow;
end
end
223
if(exist('guiflag','var'))
    set(handles.t_simtime, 'string', ['Simulation Time: ' num2str(i)]);
drawnow;
end

if exist('BSData', 'var')
    disp('entering average error calc')
estlocs = [[BSData(:,4)]' [BSData(:,5)]'];
errocs = estlocs - actlocs;
errocs = errocs.^2;
errocs = sqrt(errlocs(:,1) + errlocs(:,2));
avgderr = sum(errdist)/length(errdist);
end

if(~exist('guiflag', 'var'))
disp(['For environment ', env_type])
disp(['There are ', num2str(Ninstants), ' Time instants, ', num2str(Nteams), ' number of'])
disp(['The average error is for the geolocation block is ', num2str(avgderr)])
disp(['Of ', num2str(schemesRight(2)), ', ', num2str(schemesRight(1)), ' were found correctly'])
end