LOW PROBABILITY OF DETECTION COMMUNICATION USING INVERSE BEAMFORMING IN GNU RADIO AND CODE DIVISION MULTIPLE ACCESS

THESIS

Travis B. Rennich, 2 Lt, USAF

AFIT-ENG-MS-17-M-064

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-17-M-064

LOW PROBABILITY OF DETECTION COMMUNICATION USING INVERSE BEAMFORMING IN GNU RADIO USING CODE DIVISION MULTIPLE ACCESS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Travis B. Rennich, B.S.E.E.

2 Lt, USAF

March 2017

AFIT-ENG-MS-17-M-064

LOW PROBABILITY OF DETECTION COMMUNICATION USING INVERSE BEAMFORMING IN GNU RADIO USING CODE DIVISION MULTIPLE ACCESS

THESIS

Travis B. Rennich, B.S.E.E.
2 Lt, USAF

Committee Membership:

Dr. R. Martin
Chair

Dr. A. Temple
Member

Dr. J. Pennington
Member

Abstract

The primary goal of this thesis was to design a communications system that is more covert than existing systems while maintaining a described Bit Error Rate (BER). Starting with a simple Binary Phase Shift Keyed (BPSK) system, continuing to a Direct Sequence Spread Spectrum (DSSS) system, and finishing with an Inverse Beamforming system, each segment was validated and tested for accuracy. Due to longer simulation times for longer spreading codes, short spreading codes were used to develop and test these systems. Then Signal to Noise Ratios (SNR) of interest were selected and simulations conducted for the longer codes.

Inverse Beamforming is the use of multiple spreading codes by a transmitter, each with a fraction of the power that would have been used by a single spreading code, to transmit the same message over multiple antennas. At the intended receiver, there is a single antenna that splits the signal into multiple channels. Each channel locks to a different spreading code, and is used to reconstruct the original message. In Chip-Offset Inverse Beamforming, successive users have an increasing number of chip offsets added in between each replication of the spreading code for each bit of the data message. This is done to decrease autocorrelation spikes to lower detection performance.

Using Inverse Beamforming alone did not maintain the described BER while degrading intercept receiver detection performance. However, with the addition of chip offsets, the BER was maintained while the intercept receiver detection performance decreased in some situations.

Acknowledgements

I would like to first thank my advisor, Dr. Richard Martin, for the guidance and support over the last year and a half. I would also like to thank my committee members, Dr. Michael Temple and Dr. Jason Pennington for their support in completing this project. I would like to thank my family for providing me an outlet of stress every now and again. My friends also deserve some thanks; without the countless hours of racquetball, I am not sure I would have ever completed my work here.

Travis B. Rennich

Table of Contents

Page

List of Figures

Figure Page

List of Tables

List of Acronyms

AUC	Area Under the Curve
AWGN	Additive White Gaussian Noise
BPSK	Binary Phase Shift Keying
BER	Bit Error Rate
CDMA	Code Division Multiple Access
DoD	Department of Defense
DS	Direct Sequence
DSSS	Direct Sequence Spread Spectrum
EER	Equal Error Rate
FDMA	Frequency Division Multiple Access
GPS	Global Positioning System
GRC	GNU Radio Companion
LPD	Low Probability of Detection
LPE	Low Probability of Exploitation
LPI	Low Probability of Intercept
OOTM	Out Of Tree Module
RAM	Random Access Memory
RF	Radio Frequency
ROC	Receiver Operating Characteristic
SNR	Signal to Noise Ratio
SDR	Software Defined Radio
SS	Spread spectrum
TDMA	Time Division Multiple Access
UHD	Universal Software Radio Peripheral (USRP) Hardware Driver
USRP	Universal Software Radio Peripheral

LOW PROBABILITY OF DETECTION COMMUNICATION USING
INVERSE BEAMFORMING IN GNU RADIO
USING CODE DIVISION MULTIPLE ACCESS

I. Introduction

Beamforming has been used in many systems as a way to concentrate more of the transmitted signal power of a broadcast station onto friendly receivers. This concentration of power accomplishes two things, 1) the Signal to Noise Ratio (SNR) at the receiver can be increased with no increase in transmitting power because less of the energy is radiated into unneeded directions, and 2) the system becomes inherently more covert due to the fact that there is not as much energy radiating into unintended directions, where enemy eavesdroppers could be located.

Beamforming with many different receiving stations has been a problem for a couple of reasons. If these receiver stations are angularly spread with respect to each other (i.e. they are not all on the same direction vector from the transmitting station), simultaneous beamforming gets more difficult and less efficient, and starts to give an omni-directional pattern to the radiated energy. One way to combat this effect is to divide transmitting time between each receiver and to individually form a beam to each receiver one at a time. This technique has the advantage that it is much more covert than either forming beams to all receivers simultaneously or not using beamforming at all, but decreases the throughput of the system because the same message must be transmitted once for each receiver.

Another method of wireless communication that has inherent covertness is the DSSS system. These systems use much more bandwidth to convey their information

than is necessary. This allows these systems to hide in the background noise. This means that traditional energy detection schemes that use the average noise level present in the spectrum to estimate if there is a real signal present can be fooled. Another convenient side effect of this type of system is that multiple users can use the same spectrum simultaneously by using different spreading codes, the cause of the increase in bandwidth used. DSSS systems will be covered in much more detail in the next chapter.

DSSS techniques can be used to transmit the same message at lower power levels over multiple channels. The usage of multiple DSSS channels allows for the recovery of a larger percentage of the message, while the spreading of the packet energy over a larger time period and DSSS techniques allow for a decreased detection rate. DSSS and beamforming are the two main concepts of Inverse Beamforming.

1.1 Brief Background

Beamforming is a type of spatial filtering that is used to direct electromagnetic power from a transmitting array of antennas into one or more specific direction(s) [2]. Beamforming is used to limit the power of transmission in undesirable directions as well as to focus power into specific directions to increase the SNR at the receiver or to require less power from the DSSS transmitter.

DSSS communications systems employ spreading codes oscillating at much faster rates than the underlying data stream to spread the bandwidth of the signal. At the receiver, the same operation is completed again, but this time it de-spreads the signal back to its original form. From here, basic demodulation techniques are used to estimate the bits from the waveform.

Both of these technologies can be used to provide levels of covertness in their own manner. Beamforming provides spatial covertness; as long as none of the transmit

beams point in the direction of an enemy receiver, the probability of detection is lower than the case where beamforming is not performed. DSSS systems are more covert because they allow the users of the system to drop the transmitted power levels below the noise floor, which evades simple energy detection schemes. Note however, that if the enemy receiver has access to the spreading codes used, or is in the direct path of a formed beam, then both of these techniques fail.

1.2 Motivation

Inverse Beamforming could provide a more stealthy communications system in two ways. First, the receiver can use knowledge of the specific spreading codes used to extract phase information for each transmitted signal to better differentiate and eventually demodulate the signals in concert with each other. Since the individual signals can mostly be separated at the receiver, the hope is that the presence of multiple redundant paths for each bit can help to ensure that bits get demodulated at a lower BER than using only one unique spreading code (traditional DSSS system).

Another possible mechanism for improvement on existing systems is that the interaction of the spreading codes at an eavesdropping receiver produces a more random looking waveform than using a single spreading code would. This effect may be especially prevalent if the signals from each transmitting antenna are not perfectly phase aligned. However, even if not, the eavesdropping receiver will likely be at a non-zero angle relative to bore sight of the transmit array, so will experience phase shifts between the different spread signals anyway. Because of these effects, the eavesdropper may have a tougher time estimating when there are packets being transmitted. However, the intended receiver will be able to use its knowledge of the spreading codes to track each of the signals, even when a phase difference exists between each of them.

Both of these effects may give Inverse Beamforming a way to reduce the probability

of detection by intercept receivers. The usefulness of such a system is great; for instance, flying missions behind enemy lines would be safer because the members of the team are able to communicate with each other, alerting others of possible dangers, and the enemy would have a decreased chance of detecting that anyone was there.

1.3 Goals

The overall goal of this project is to implement a communications system with a Low Probability of Detection (LPD) that also decreases or keeps constant BERs for friendly receivers. To measure the system success, two different measures will be recorded. The BERs of the friendly receivers will be the most important. The second is the detection rate of the packets by the intercept receiver.

The overall goal of this project is a system that either decreases the BER with no change in the probability of detection, or no change in the BER with a decreased probability of detection.

1.4 Scope

The scope of the experiments will be limited to only simulations. These simulations will vary the number of users and the power of the Additive White Gaussian Noise (AWGN) added, and will compare the BER of the friendly receiver with the packet detection rates for many different detection thresholds of the intercept receiver.

This project will not deal with hardware validation of the results, simulated frequency or timing offsets, or multi-path effects. Also, the receiver will have knowledge of the locations of the packets and will not need to perform any sort of timing recovery to demodulate the signal.

1.5 Applicability to DoD

The Department of Defense (DoD) has many uses for a communications system that is difficult to detect when operating. Such a system could allow for missions behind enemy lines to be much safer, as the participants could communicate with less worry of being detected. If kept on a Software Defined Radio (SDR) platform, a system with Inverse Beamforming capabilities could dynamically change its operating characteristics to maintain low detectability in response to evolving threats.

LPD communications systems are of great interest to the DoD and would allow war-fighters to move and communicate with one another while remaining undetected. [3, 4].

1.6 Overview of Thesis

Chapter II will provide the necessary information to understand the rest of the document, Chapter III will discuss the scope and methods used to conduct the simulations, Chapter IV will provide the results and an analysis of the results of the simulations, and finally, Chapter V will draw conclusions based on those results.

II. Background

This chapter will provide an introduction and background material on all topics necessary to understand the rest of this document. These topics include digital modulation, beamforming, DSSS/CDMA, LPD systems, Detection of DSSS signals, Inverse Beamforming, and GNU Radio.

2.1 Modulation

Modulation involves mapping a waveform to another waveform that allows for efficient transmission across a channel. There are two types of modulations, analog and digital. Digital modulations are a modulation from a discrete set of symbols (i.e. a bit stream) to a waveform that can be transmitted across a channel [5, 6].

Modulation is used to transmit information from one location to another. To transmit the information over a channel, a carrier wave is used [5]. The amplitude, phase, and frequency of the wave are the three possible parameters to change about the wave [5]. The base modulation type used in this thesis will be BPSK, which varies the phase of the carrier wave to transmit the information.

Figure 1 shows the idea behind BPSK modulation. The top plots show the two possible communications symbols, a sample data stream is shown in the middle plot, and the bottom shows the resultant waveform ready for transmission over the air. Note that the communications symbols are phase shifted version of each other. This is the origin of the name of the modulation (BPSK).

The BER of a BPSK system can be predicted using the following,

$$P_B = Q\left\{\sqrt{\frac{2E_B}{N_0}}\right\}, \tag{1}$$

where P_B is the probability of bit error, E_B is the average received energy per bit,

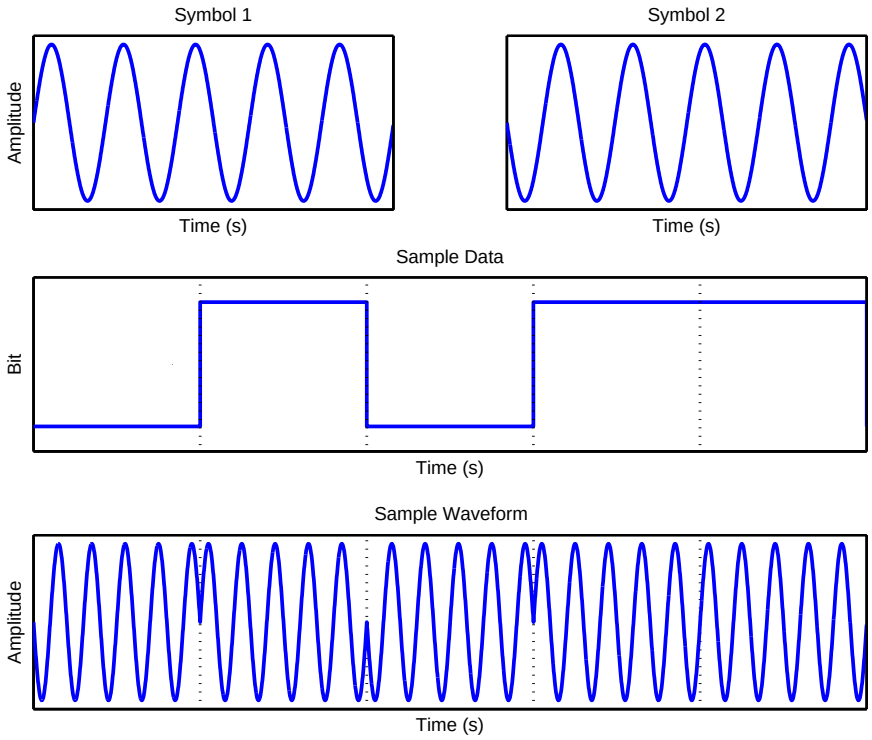

Figure 1. BPSK symbols (top), Data stream (middle), and modulated data stream (bottom)

and N_0 is the average two-sided power spectral density of the noise [7].

Note that technically modulation refers to the mixing of a signal with a carrier frequency. Within the GNU Radio framework however, modulation is taken to mean the conversion of a bit stream, usually in the form of bytes, either unpacked or packed, to complex samples. This is roughly equivalent to the technically correct definition of modulation because the up-conversion of the complex samples to a carrier frequency is done implicitly in the GNU Radio hardware transmitting blocks.

2.2 Beamforming

Beamforming is a process by which multiple transmit antennas can "steer" the direction of the beam, or the main lobe of energy, into a specific direction. This allows the transmitter to concentrate the output signal power along desirable vectors,

as well as to limit that power in other directions. This process is achieved by very finely controlling the phase delays between each of the transmit antennas in such a way as to ensure that the energy from each antenna constructively adds in the desired direction, and destructively adds in undesired directions [8]. Note that this can be done at a receiver as well; due to the reciprocity theorem, receiving antennas and transmitting antennas have identical properties, provided that there are no active components in the antenna [9].

2.3 Code Division Multiple Access (CDMA)

2.3.1 Introduction to DSSS Systems.

Spread spectrum (SS) communications refers to a system that uses a much wider bandwidth than is required for the given data-rate of the system [7]. Direct Sequence (DS) refers to using a pseudo-random bi-polar spreading signal that varies at a faster rate than the data to be transmitted [10]. These DS signals are referred to as spreading codes due to the fact that they spread the spectrum of the waveform during mixing.

There are multiple benefits to using SS systems. This design can be used 1) for conditions where multiple users must access and share communication resources, 2) for interference rejection, and 3) for covert applications where a low probability of detection is desired [10]. These types of systems also present anti-jam capabilities [7, 11]. DSSS signals will be a main focus of this thesis.

The structure of a DSSS signal is as follows. Assume that a message waveform $m(t)$ is to be modulated. Also assume that $m(t)$ is a digital waveform that takes on the values ± 1. The modulated signal is then $w(t) = Re\{m(t)\, e^{j\omega_c t}\}$ where ω_c is the carrier frequency of the system and $Re\{\}$ is the function that returns the real parts of its input. Now assume that a spreading code $c(t)$ is applied. This leaves $s(t) = Re\{c(t)\, m(t)\, e^{j\omega_c t}\}$ as the form of the modulated and spread signal. At this

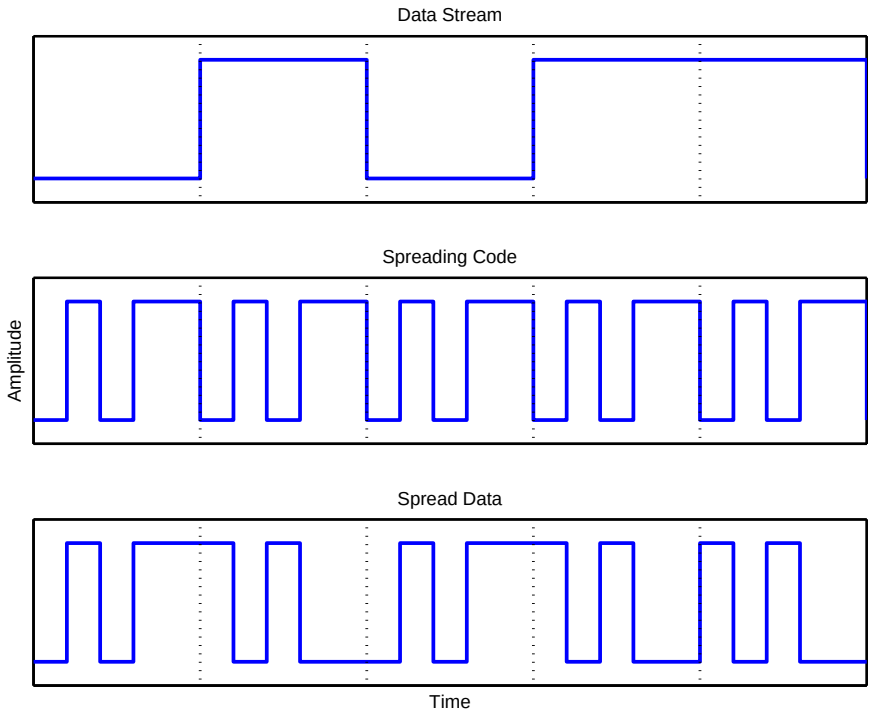

Figure 2. Data stream (top) spread with a repeating code (middle) to produce the final waveform (bottom)

point, the bandwidth of $s(t)$ is R_C, where R_C is the chip rate, or the rate at which the spreading code changes [10]. In typical systems, this R_C is the length of the spreading code times faster than R_D [7]. The fraction $\frac{R_C}{R_D}$ gives the processing gain of the system, which gives the difference in performance of the system when the SS techniques are used versus not used when all other factors are kept equal [12].

In Figure 2, the top plot shows a data stream before the spreading operation. There are 5 total symbol durations in this plot. The middle plot shows the spreading code that will be used to spread the data stream from the top plot. Notice that it changes at a much faster rate than the data stream (in this case, 5 times as fast). Also note that the spreading code has been repeated, once for each symbol duration in the original data stream. To obtain the bottom spread data stream plot, the repeated spreading code and data stream are simply modulo-2 summed.

2.3.2 Introduction to CDMA.

CDMA is an extension on DSSS systems that allows multiple users to access the same frequency spectrum simultaneously [10]. In these systems, each separate user is assigned a different DS code, usually called a spreading code. These codes are (or are approximately) orthogonal, which allows receivers to extract a particular user's signal given that the code used to generate that signal is available, even when multiple users are using the system [10]. The receiver must have a method to synchronize itself to the symbol transitions of the transmitted signal; without it, the receiver incorrectly applies the spreading code, and the resultant waveform will not be equivalent to the original signal before spreading at the transmitter [10].

CDMA provides a number of advantages over similar technologies (Frequency Division Multiple Access (FDMA) and Time Division Multiple Access (TDMA) methods) that provide multiple access capabilities. FDMA systems divide the available frequency spectrum between the users, and each user is free to use their slice of the spectrum at all times. Time division divides the time that signals are allowed to arrive at a receiver between all users, and the users are free to use their time period to transmit information. Advantages of CDMA over these two methods include privacy, fading channel sharing, jamming resistance, and flexibility [7].

CDMA provides more privacy by the fact that a receiver must know the exact spreading code used to generate the signal in order to obtain the original signal [7]. With FDMA signals, if the eavesdropper knows the frequencies used by any users of the system, it can detect when the user is transmitting with a simple energy detector because the signal must still be above the noise floor for demodulation. The same idea applies to TDMA, but now the eavesdropper must know what time slice corresponds to the user of interest.

In FDMA, in a frequency selective channel, users can be affected differently by

channel conditions. One user could be assigned a frequency range within a band of high attenuation by the environment and not be reassigned for some time. This will cause this user's communication link to suffer, but no others as long as the band of high attenuation does not extend into their assigned frequency bands. TDMA can have similar issues if there is a bursty interference source, but the issue is not as large as in FDMA. Because in CDMA all users share the same frequency band and can transmit at any time, all users also share any of the channel fading characteristics or outside interference sources [7].

CDMA provides resistance to jamming because it motivates the jammer to spread the jamming energy over the same spectrum as the spread signal, but at the same time de-spreads the original signal. In this way, only a fraction of the jamming energy is able to affect the demodulation process [7]. This fraction is roughly equivalent to the processing gain of the system, or the ratio between the spreading code rate and the data rate. In TDMA and FDMA, the jamming energy is either at the correct time (for TDMA) or at the correct frequency (for FDMA) or it is not. Given that the accuracy and precision of the jammer can be very high, jamming can affect different users differently, while in the CDMA case, all users would be affected roughly equivalently.

CDMA is much more flexible than TDMA because highly precise transmit timing is not required[1]. TDMA requires that user transmissions arrive at the receiver at precisely defined times, while CDMA can be implemented asynchronously [7]. Especially in mobile networks, this is a great advantage as the relative movement between transmitter and receiver does not need to be known or accounted for.

[1]This does not mean that timing **recovery** at the receiver is not needed, just that the transmitter does not need to account for path length to the receiver when transmitting as in a TDMA system.

2.3.3 Spreading Codes.

There are many different types of spreading codes, and each has varying statistical properties that make them more suitable than others in a given situation. Two common types include Hadamard and Gold codes. Hadamard codes are all mutually orthogonal with each other at a specified code phase, and can be useful in synchronous CDMA systems [13]. Gold codes families are generated from well chosen maximal length sequences such that they have very low autocorrelations, even with a code phase offset [12]. These properties are helpful in asynchronous CDMA systems where the symbols are not guaranteed to be synchronized at the receiver. Gold codes are the type of spreading codes used during this thesis.

2.3.4 Synchronous/Asynchronous CDMA.

There are two kinds of CDMA systems, synchronous and asynchronous systems. In a synchronous system, the transmitters time their signal transmissions such that they arrive at the receiver with a specified code phase. This means that the receiver only needs to deal with finely synchronizing with the data symbols, but defeats one of the main advantages to moving to a CDMA system from a TDMA system. In an asynchronous system however, the different users are not required to time their transmissions at all. This gives a lot more freedom to the transmitters, but the receivers are now required to be able to determine when each separate user is transmitting and synchronize to the entire message [7].

2.3.4.1 Gold Codes.

Gold codes are generated from preferred pairs of maximal length sequences. They have very well defined autocorrelation properties such that the possible autocorrelation values at every code phase are given by Equation (2). Gold codes are all of

length $N = 2^n - 1$ and there are $N + 1$ total codes in each family, where n is the number of taps in the linear feedback shift register used to generate the maximal length sequences used in the code, and N is the length of the spreading code [12].

$$\begin{matrix} -\frac{1}{N}(n) \\ -\frac{1}{N} \\ \frac{1}{N}[t(n) - 2] \end{matrix} \tag{2}$$

where

$$t\,(n) = \begin{cases} 1 + 2^{0.5(n+1)}, & \text{for } n \text{ odd} \\ 1 + 2^{0.5(n-2)}, & \text{for } n \text{ even} \end{cases} \tag{3}$$

Gold codes are used in the Global Positioning System (GPS) [14, 15] and NASA's Tracking and Data Relay Satellite System [12].

2.3.5 Interaction of Spreading Codes.

The sum of the signal contributions from each user and noise is the input to the CDMA receiver. To have good performance, the contributions from all users except for the user of interest should be canceled by mixing with the user of interest's spreading code. If not, the receiver has a much higher probability of mis-estimating the symbol, which leads to an increased BER. This points to the use of orthogonal spreading codes. In the context of a synchronous system, this idea will work, but often orthogonal codes have good cross-correlation properties only if they are chip-synchronous, i.e. there is no code-phase offset between different user codes at the receiver. This is only possible in a synchronous system, and therefore not as useful in the systems considered here [12].

2.3.6 CDMA System Design.

The design of the transmitter is very simple. Aside from the spreading operation, the functionality is exactly the same as a BPSK transmitter. Figure 3 shows the addition of the spreading mixer. Assuming that the input data stream is in the form of unpacked bi-polar bits, this is the only addition over the BPSK modulator/transmitter.

The general design of a CDMA receiver is shown in Figure 4. The first step is to receive the signal and convert to baseband. This is shown on the left, outside the dashed line box. Inside the box is the actual receiver, starting with the despreading operation. This operation involves mixing the received signal with a replica of the spreading code used to spread the signal at the transmitter. The next step is to down-sample the stream so that there is only 1 sample per symbol, then finally to estimate the bits using a threshold operator.

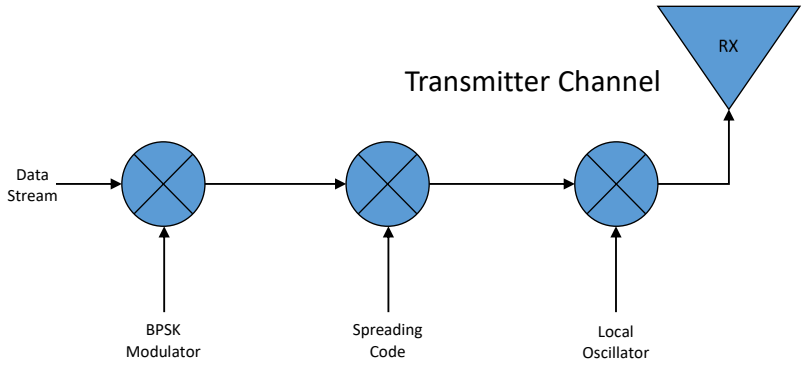

Figure 3. Single DSSS/CDMA transmitter chain.

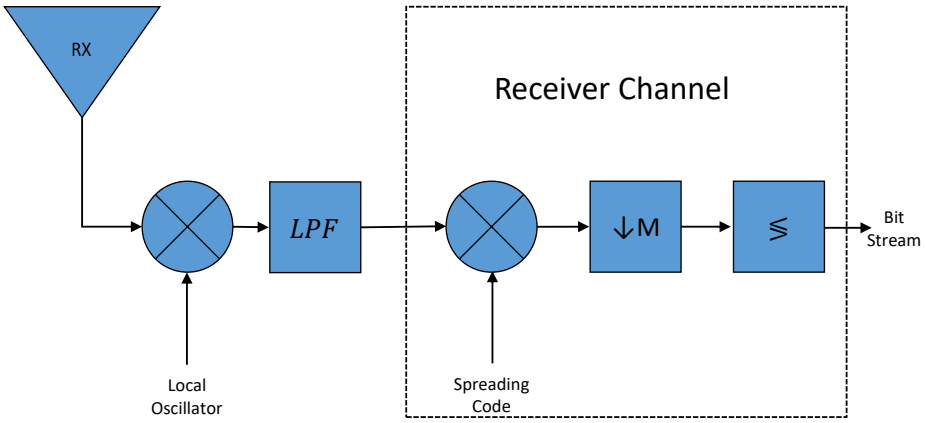

Figure 4. Single DSSS/CDMA receiver chain.

2.4 Low Probability of Detection (LPD)

LPD and Low Probability of Intercept (LPI) systems have gotten a lot of attention in the past years, especially within the military sector. LPD systems are designed to be difficult to detect when operating, while LPI systems are designed to be difficult to intercept during operation. Being able to communicate covertly is obviously a great tactical advantage, and as such, systems that can do this effectively are highly sought after.

LPD operation implies that the probability for an unfriendly receiver to detect signal presence is low. The unfriendly receiver is generally called an intercept receiver in the literature [16].

SS systems inherently provide some level of covertness due to the fact that the energy in these systems is spread over a much larger bandwidth than is needed to relay the necessary information across the channel [12].

Another phrase, Low Probability of Exploitation (LPE) refers to the difficulty of the intercept receiver to exploit any information about the communications waveform to estimate information. This information could be the position of the transmitter, as well as the actual information being transmitted [16]. To exploit the system however, one must first be able to detect that it is operating. Prescott describes four stages to signal exploitation [16]:

1. Tune intercept receiver to at least some of the operating frequencies of the target transmitting system.

2. Detect when the signal is present.

3. Intercept the signal, extract features of the waveform to determine if it is a signal of interest.

4. Given that the signal is of interest, exploit all possible information.

There are many different parameters to consider when designing a LPD/LPI communications system. Obviously, the transmitting power plays a large part in the ability for an intercept receiver to detect and exploit the signal. Some techniques to mitigate transmitting power entering an intercept receiver include beamforming, null-steering, using low-sidelobe antennas and frequency control. Beamforming and null-steering are closely related ideas that attempt to radiate power only in wanted directions and not in unwanted directions. Low-sidelobe antennas refer to antennas that transmit low amounts of power in directions that are not aligned with the main beam of the antenna. Frequency control refers to the use of frequencies that are atmospherically attenuated more strongly when the receiver is a short range from the

transmitter and only use frequencies that are not as sharply attenuated when the friendly receivers are further away [16].

Other techniques to consider are to use SS ideas, error correcting codes, adaptive interference suppression, and signal masking. These techniques rely more on characteristics and methods of modulating the data stream than antenna and channel characteristics. Error correcting codes are used to correct and detect bit errors in the demodulated data stream. Adaptive interference suppression is used to notch out the frequencies used by jamming signals to require less power use by the transmitter. Finally, signal masking can be used by selecting a modulation scheme with some processing gain and transmitting on a frequency used by another entity, but with less power. The processing gain allows the receiver to demodulate the signal, but the higher power entity tends to mask the signal from the intercept receiver [16].

SS methods show LPD characteristics due to the large processing gain deriving from the use of the spreading codes. Since the system can operate at signal levels below the ambient noise floor, simple energy detectors cannot accurately detect when the signal is present [7].

However, the intercept receiver also has many techniques available for use. Beamforming, null-steering, and low-sidelobe antennas can also be used by this receiver in an attempt to raise the SNR. Given knowledge of the location of the transmitter, these techniques can provide a great increase in the SNR at the intercept receiver.

The other LPD/LPI techniques are more difficult to exploit without inside knowledge of the system. For example, if the intercept receiver has knowledge of the specific spreading code used, it can more easily demodulate the signal for exploitation, but the spreading codes are purposely only shared with the intended receivers.

2.5 Non-Cooperative Detection of DSSS Signals

Detection of signals has traditionally been done by comparing the amount of power in a specific bandwidth with the normal amount (background noise). If this difference is statistically significant, then the detector estimates that a signal is present. With DSSS signals however, due to the use of the spreading code, it is possible that the amount of power at the center frequency of the system is less than the average ambient noise power (negative SNR condition). In this case, the simple energy detector will fail to accurately predict when the system is transmitting and when it is not. Another method of detection will need to be devised [17].

Note that cooperative DSSS receivers work by correlating the received signal with a local replica of the spreading code. Detectors do not have access to this spreading code, but can use correlation to help detect DSSS signals. Since the spreading code is repeated for each bit in the data sequence, there are peaks in the autocorrelation of the signal at the period of the spreading code. The detector can use these peaks to help determine when a signal is present and when there is not one present [17]. Autocorrelation can be used in the presence of AWGN because ideal AWGN has an autocorrelation approaching 0 for all time delays except at a lag of 0 [18].

The main goal of Inverse Beamforming is to reduce the autocorrelation peak magnitudes so that the detector has a tougher time determining which peaks are due to noise alone and which are due to signal and noise.

2.6 Inverse Beamforming

Inverse Beamforming takes some advantageous properties from both regular beamforming and DSSS systems. The transmitting station uses multiple chains that each spread the data stream with a unique spreading code and transmit on separate antennas. At the receiver, a single antenna is used to receive the signal, then separate

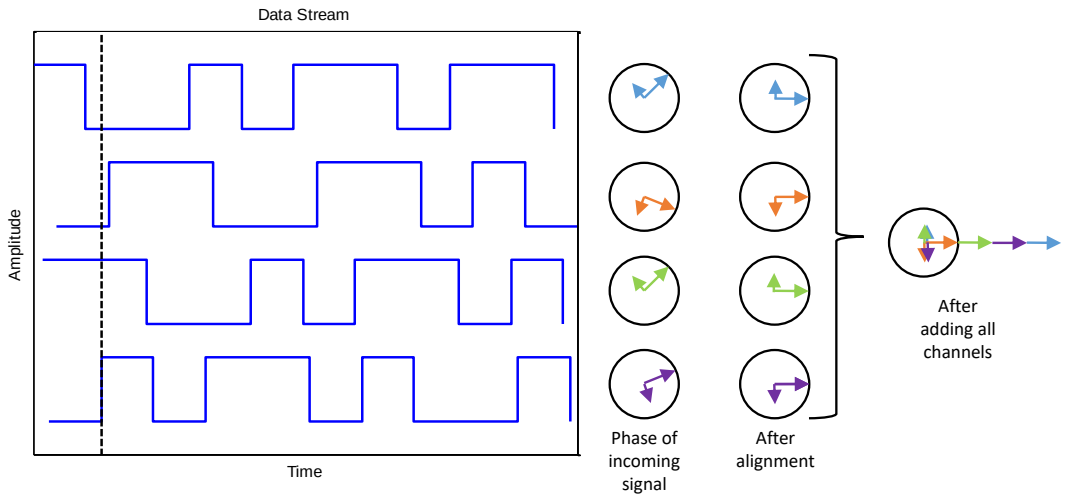

Figure 5. Inverse Beamforming phaser diagrams showing the process to retrieve the original signal. (Adapted from [1])

channels are used to find the phase differences and align the signals to add and demodulate. This process is shown in Figure 5. The plots on the left are data bits as they arrive at the receiver. The design allows for phase offsets between each channel. After cancellation of this phase offset for all channels, the contributions are combined and used to estimate the data bits.

Dragonov et al. [1] used Inverse Beamforming as a way to navigate in high multipath environments where access to GPS signals could be diminished. Inverse beamforming works well in this case for a few reasons: 1) the transmitter station can serve multiple users simultaneously, 2) the station does not require knowledge of the users' locations, and 3) each user can carry only a single antenna [1]. Note that the use of the term "Inverse Beamforming" in this thesis is different from the likes of [19].

The Inverse Beamforming transmitter looks like several separate DSSS/CDMA transmitters all fed by the same data stream. The data stream is fed to separate mixers that each spread the signal with a unique code, then modulate with the same local oscillator, shared by all channels. From there, each signal is transmitted separately via its own antenna [1]. Figure 6 shows the transmitter configuration in an

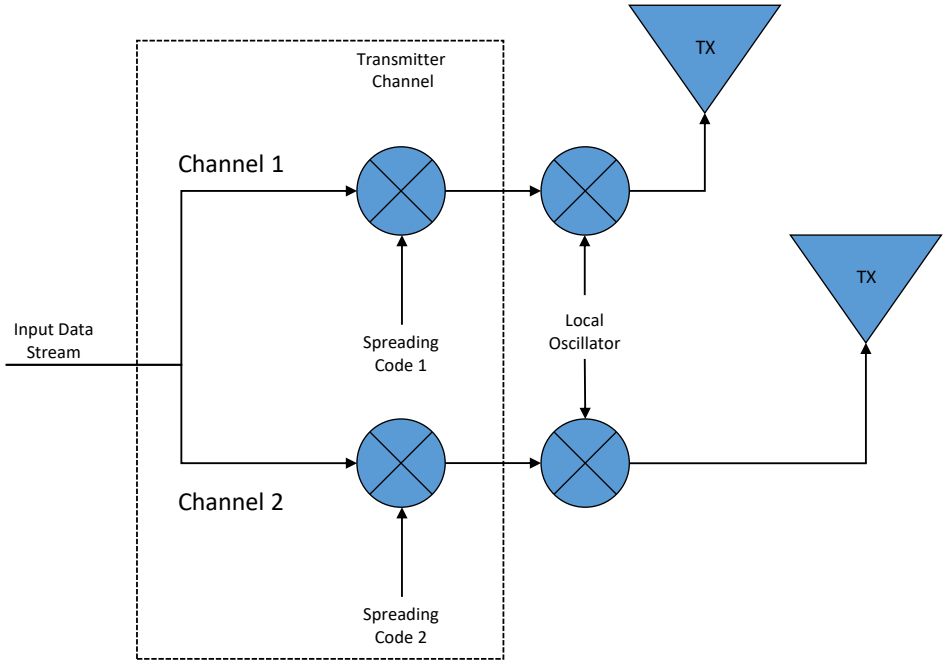

Figure 6. Two channel DSSS/CDMA transmitter chains being fed with the same input data stream.

Inverse Beamforming system.

Fortunately, DSSS methods allow the receiver to separate the contributions of each signal at the receiver, even when only a single antenna is used. To accomplish this, the signal from the antenna is fed into multiple separate channels. These channels each contain the components required to demodulate a single spreading code. After each spreading code is demodulated, the bit streams are recombined and used to estimate the original bit stream. Compiling the results from each separate channel is meant to correct errors that are expressed in less than half of the channels for that symbol. Figure 7 shows how the multiple receiver chains are combined to produce a single output.

More receiver channels could easily be added to introduce more unique spreading codes as well. To do this, the new channels would only require that they get a copy of the received samples, and that the recombination would also add the contributions

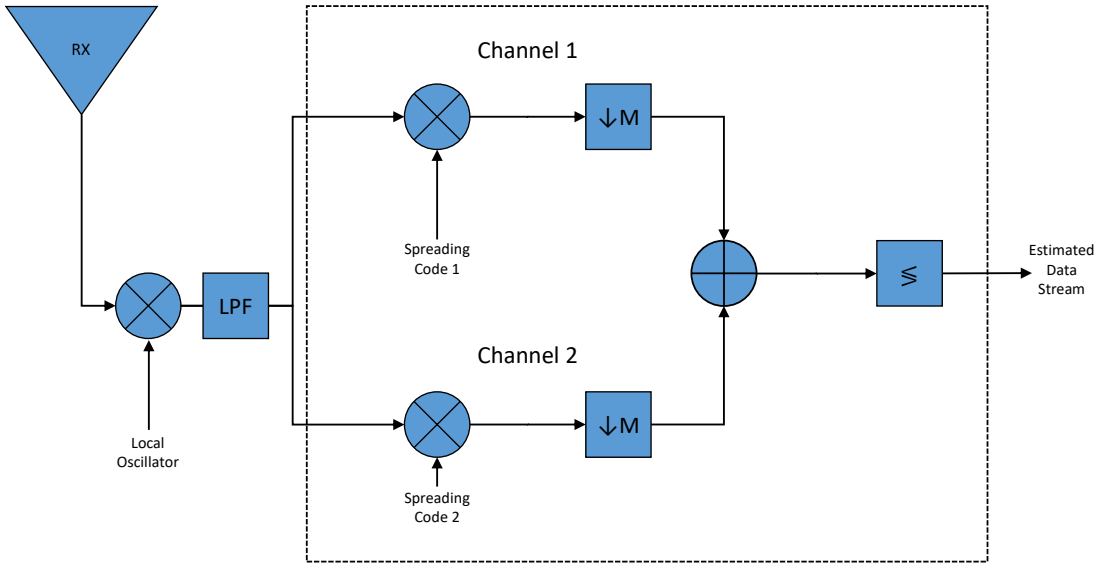

Figure 7. Multiple DSSS/CDMA receiver chains combined with a summing operation.

from the new chains. In this way, scaling is linear in the number of unique spreading codes used.

Note that much of the computational power needed to perform Inverse Beamforming has been transferred from the transmitter to the receiver. In Inverse Beamforming, the transmitter must spread each of the N_C signals separately, but this process does not require much computational power to complete. At the receiver however, there are now N_C separate DSSS channels (until recombination after the symbol correlator). Each channel requires 1 correlator to despread the signal, so with the addition of each channel, more computation power is required by the receiver. This increase in computational power is linear in the total number of spreading code channels.

2.7 GNU Radio

GNU Radio is an open-source development platform that provides many built-in tools for signal processing to implement radio designs. GNU radio can be used with many external Radio Frequency (RF) platforms to implement SDRs. [20]

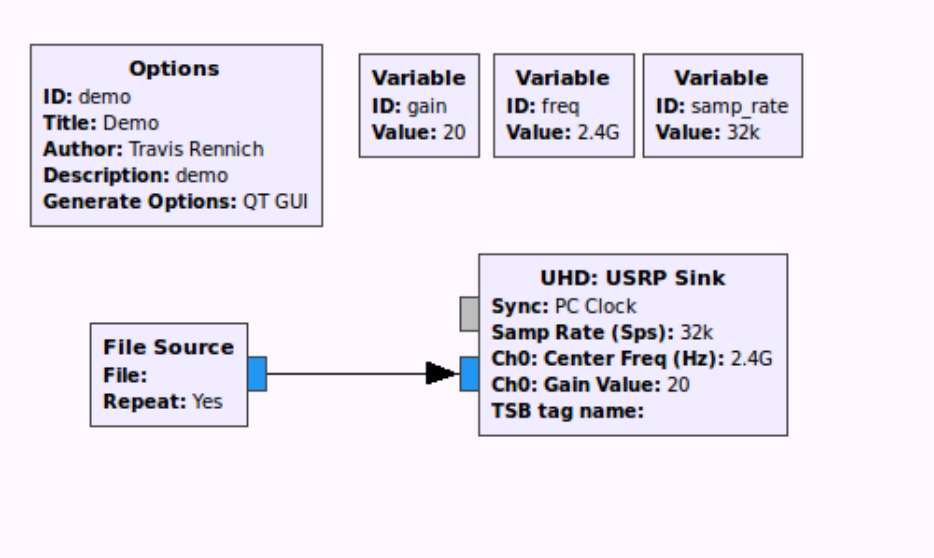

Figure 8. Basic flow graph showing file source connected to a UHD sink.

2.7.1 Blocks and Flow Graphs.

GNU Radio functionality is based on flow graphs that can perform widely varying tasks, from Fast Fourier Transforms to wireless channel models to modulating/demodulating data streams. These flow graphs are comprised of blocks that each perform as specific a function as possible to promote re-usability. Blocks in a flow graph are connected to produce more complex signal processing functionality. Each block has a predefined input and output signature. This signature determines the type and number of items that can be input into and output from the block [21].

Figure 8 shows a basic flow graph with only two functional blocks. These blocks are a file source and a USRP Hardware Driver (UHD) sink. The UHD is a library that allows users to interact and control with SDRs [22]. The file source reads data from a file and sends it downstream to the UHD sink block, which modulates to the carrier frequency and transmits it. The Options block controls metadata information about the flow graph such as the name of the graph, the author, and the type of graph to generate (type of graphical user interface to use, if any). The variable blocks allow

the creator of the graph to use variables to specify values, and those variables can be used in algebraic statements to allow for more flexibility within the flow graph. Variables of many data types are possible.

2.7.1.1 Types of GNU Radio Blocks.

There are many different kinds of blocks. These include source blocks that do not have any data inputs, but produce data; sink blocks, which have data inputs but no outputs; and blocks that have both data inputs and outputs. Within this third category, there are multiple types of blocks, defined based on the relative rates of data flowing into the block as coming out of the block. These types are decimation blocks that have a rate of $N : 1$, where for every N inputs there is exactly 1 output, interpolation blocks, whose rate is $1 : M$, and sync blocks, whose rate is $1 : 1$. The more general type of block that the previous three derive from is the general block, which can have any fixed rate, or even a rate that changes between calls to the `general_work()` function, discussed later [23]. The last major type of block is the hierarchical block, which is composed of blocks itself. This type of block can also have any input to output rate [23, 24].

2.7.1.2 Components of GNU Radio blocks.

GNU Radio Blocks can be written in either Python or C^{++} [21], although well written C^{++} blocks are likely to run faster than their Python counterparts. Each block inherits from a general block that contains many methods related to block operation. The two most important methods are the constructor and `general_work()` functions. The constructor is responsible for initializing everything that the block could potentially use during its lifetime, and the `general_work()` function is responsible for the actual calculations and signal processing during operation. This method is

called when there are enough input samples to process, with a request to produce a certain number of output samples, the number of input samples available, and the input and output vector locations. However, the `general_work()` function does not need to produce the full amount of requested output samples, nor does it need to consume all input samples. Upon completion, it informs the scheduler through the use of the `consume()` function how many input samples it actually used, and through the use of the return value the number of output items actually produced [25].

Other important methods include one that sets the number of samples passed into the block to always be at least a certain value (`set_history()`) [25–27], another that calculates the number of input samples needed to produce a certain number of output samples (`forecast()`) [25, 27], and the ability to set the multiple of output items requested on each call to the work function (`set_output_multiple()`) [25, 27]. Many more such functions exists, but the previously mentioned were the most heavily used in this project.

2.7.1.3 Flow graph operation.

Each time a flow graph runs, each block is constructed. This involves calling its instance constructor, which initializes all components necessary for operation. When all of the blocks have been constructed, the first block's `general_work()` function is called, which, when complete, informs the scheduler how many items it produced. From here, the scheduler determines how many items to allow the next connected block to process, and calls its `general_work()` function [25]. This process continues indefinitely until either there are no more samples to process or the flow graph is closed by the user.

2.7.2 Items.

Blocks operate on items. Each item is one piece of data in the stream and can have one of a number of different types (byte, short, int, float, complex, etc). These items are passed through connections between each block. The items are fed into the blocks via streams. Streams are the flow of items from one block to another. The block takes in the items via its input stream(s), performs its functionality, then outputs the transformed data items through its output stream(s), likely the input to another block [21].

2.7.3 Stream Tags.

Stream tags allow for the tagging of specific items in the output buffer to pass information between blocks in a synchronous manner. Once created, the tag is fixed to a specific item [21]. Stream tags can be read by downstream blocks, and allow for changes in functionality based on the contents of the tag. The contents can be any data type, even a list or dictionary of multiple different types. This allows for great flexibility with extra information that can be passed along with the actual data samples [28].

2.7.4 Built In Functionality.

GNU Radio contains many built in modules that have blocks that perform many common signal processing tasks. A general module that contains blocks that perform many tasks that are not necessarily signal processing oriented is provided as well. This module contains blocks that write data streams to files, perform basic arithmetic functions on data streams, convert the types of data streams and more. Modules for computing the fast Fourier Transform, filtering (including both infinite and finite impulse response filters as well as fast Fourier Transform filters), noise sources, digital

modulations, and more are also provided [29].

2.7.5 GNU Radio Companion.

GNU Radio Companion (GRC) is a graphical user interface tool that allows the user to drag and drop blocks into a flow graph. It also allows the user to connect these blocks, and, when finished designing the flow graph, to compile and run that graph. The tool gives easy access to any blocks that are available, and allows for the searching of these blocks to find a specific one [30]. Note however, that flow graphs do not need to be built in GRC, and, in fact, more complicated graphs that activate different block paths in the graph depending on variable conditions are not possible to construct within GRC, but are possible when directly editing a Python script. Before running a flow graph, GRC must first compile the graph into a python script. An easy way to create a more advanced python script is to first create the basic flow graph structure in GRC, then to compile the graph into a script. Once this has been done, the user can copy and modify the script to add more advanced functionality.

2.7.6 Out Of Tree Modules (OOTM).

A major feature of GNU Radio is the ability to create user defined blocks. While many possible signal processing functions are provided out of the box, many projects also need the ability to add custom functionality. This ability is provided with the creation of an Out Of Tree Module (OOTM) [27].

This allows the user to create modules that can be installed over top of an existing GNU Radio installation to augment functionality. A list of some of the open sourced OOTMs developed is available at [31]. Any of the modules listed there can be downloaded and installed assuming that the required core components of GNU Radio have already been installed on the system.

2.7.7 Debugging.

Debugging a GNU Radio application is generally done by completing unit tests for each block. When these tests pass, the block is likely to be finished. However, there is no debugger to view intermediate states of internal variables of specific blocks during execution, so one of the only options is to use print statements to dump data to the screen or temporary files to show the contents of variables. Note that it is also possible to attach a debugger, such as gdb [32], to the process.

III. Methodology

The primary goal of this thesis is to design a communications system that is more covert than existing systems while maintaining a constant BER. As discussed previously, an Inverse Beamforming system with chip offsets was designed to accomplish this task. Simulations were first conducted to ensure that the system operates as theoretically predicted. Next, more simulations were performed to ensure that the system does in fact decrease the probability of detection by enemy members, as well as to ensure that BERs do not increase. The remaining parts of this section describe the procedure for these simulations.

3.1 Assumptions

Some assumptions are needed to limit the scope of this project. These assumptions are listed in the following sections.

3.1.1 Channel.

For this project, only channels with AWGN were studied. While not an accurate assumption under all cases, AWGN is often used as a basic noise model for most channels. Additive noise means that the noise is independent of the signal, and white Gaussian noise assumes that the power spectral density of the noise is a constant equal to $\frac{N_0}{2}$ [33].

In addition, only channels with constant, non-varying, frequency responses will be studied. This means that the channel will not be frequency selective, i.e. that it will have a constant attenuation for all frequencies involved. No multi-path components will be included, nor will jamming signals. The only signals present in all simulations will be the contributions from each spreading code and AWGN.

3.1.2 Intercept Receiver.

To make fair comparisons between the detection performance of the intercept receiver and friendly receiver, the same received signal will be used to complete both functions. This means that both receivers will have access to the same stream of baseband samples from which to make their estimates. In a real-world situation, this is obviously impossible; the intercept receiver will never be using the same antenna as a friendly receiver of the communications signal.

3.1.3 Relative Angle to Receiver.

The simulations will be conducted to assume that the receiver is placed perpendicularly to the transmit array. This also means that the relative angle to the receiver from the transmit array is 90°. This means that there will be no phase differences between any of the separate signals at the receiver.

3.2 Component Descriptions

The overall goal of this thesis is a more covert method of communication, so one of the measures of success will be the signal detection probability. This will be measured using a detector block that receives only information from the channel, no other blocks in the flow graph. This will ensure that no cross-contamination from friendly receiver functionality to intercept receiver functionality will occur. The non-cooperative detection receiver results are compiled into ROC curves for analysis. To ensure that the system does not have a drop in performance from a friendly user perspective, the BER of the friendly receiver will also be recorded. These measures will then be compared to theoretical values.

3.2.1 Simulation Design.

Simulations will be the primary results of this thesis. The simulations will all take place by running a single, highly configurable flow graph. This flow graph will alter the configuration of the connected blocks based on the inputs and parameters passed to it. In this way, all the runs can be completed using a single flow graph with varied input parameters.

3.2.1.1 Packet Design.

Because the locations of the start and end of the packets are well defined and known at the receiver, the packets in this communications system will consist purely of a data portion. In a real world system, there would likely need to be a preamble or header sequence for the receiver to lock on to before demodulating the data message. The packet will consist of two bytes of randomly generated data.

3.2.1.2 Chip Offset Inverse Beamforming.

The way that Inverse Beamforming thwarts detectors is by spreading the energy present in each bit over time as well as in frequency. This allows the power to vary in time more slowly, as well as to be much lower overall. The time spreading is done by increasing the number of empty chips between each bit according to the spreading code index used for that channel. Figure 9 shows this technique more clearly.

In Figure 9, the index of the spreading code used for the modulation of a row is shown on the left. The entire row represents the chip positions for a single packet. The numbers $1-4$ represent the first through fourth chips of the spreading code. For the first bit, all users have aligned chips. In this context, bit is taken to mean the entire spreading sequence. For the second bit however, the chip offsets start to affect the placement of the first chip of each bit. The zeroth user continues placing chips

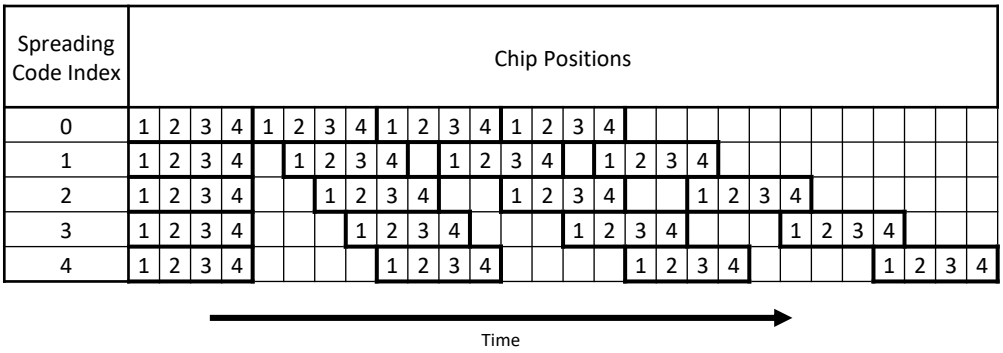

Figure 9. Chip Offset Technique to lower autocorrelation magnitudes (code length 4)

as in a normal DSSS system, but after finishing the bits in the current packet will wait until all other spreading code channels have also finished. The first channel will place exactly one chip in between each bit, the second channel will place 2 chips, and so forth. Note that this means that the length of the packet is expanded by:

$$PL = ((CL + NU - 1) * (PB - 1) + CL) * SPS \tag{4}$$

where PL is the length of the packet, in samples, CL is the length of the spreading code used, NU is the number of unique spreading codes used, PB is the total number of bits in the packet, and SPS is the number of samples per symbol.

Adding chip offsets for each spreading code channel so that the contributions from each channel do not constructively add in the autocorrelation of the signal is done to reduce the probability of detection. By introducing an increasing offset, the autocorrelations of each new code channel have peaks in lags increasing by one. Therefore, with small packet sizes, the autocorrelation peaks will be lower and harder to detect. Another way to look at this effect is that the packet energy is spread in both time and frequency content, so the total noise energy the detector must ignore increases.

3.2.1.3 Transmitting Function.

The transmitter block will function much like that shown in Figure 6. The overall block will accept packed data[1]. After unpacking, the block will modulate the data and spread the signal with the specified spreading codes. Each of the spread signals will be synchronous at the chip level at this point with the exception of the purposefully added chip offsets.

3.2.1.4 Channel.

In a hardware experiment, the separately spread signals would be implicitly added after transmission in the wireless channel. However, in a simulation, this has to be done manually. Luckily, GNU Radio provides a block to add a variable number of signals, [34]. Each spread signal, as well as the generated AWGN are added by this block, and the output is a good approximation to the signal that would be received by an antenna in an environment with the same noise power.

3.2.1.5 Friendly Receiver Function.

After passing through the channel, the samples arrive at the demodulator. Given that there are N unique spreading codes used, the received signal will get copied N times and each channel will get its own copy. Each channel will undo packet spacing effects for that channel to process the correct samples. For each bit in the packet, each channel will produce a floating point value that describes how confident that channel is that it knows the true value of that bit via its magnitude. All of these floating point values are summed across all channels for each bit, then fed into a thresholding device. Positive sums equate to a '1' bit, while negative sums translate

[1]Packed bytes are those in which the meaningful grouping of bits is smaller than 8 bits; i.e. there are multiple symbols per byte. In unpacked bytes, there is only one symbol per byte, no matter how many bits/symbol are required.

to a '0' bit.

3.2.1.6 Adversarial Detection Function.

The adversarial detection block chain has a much tougher job than either of the other two functions (transmitter and friendly receiver). This chain must try to detect when the transmitter is transmitting and when it is not. To do this, it will use an autocorrelation method to try to detect the packets.

The idea behind the metric is described in [17]. In short, the detector will find the autocorrelation of the input stream over a given window:

$$\hat{R}_{yy}(\tau) = \frac{1}{T}\int_0^T y(t)y^*(t-\tau)dt \tag{5}$$

where $y(t)$ is the received signal, T is the window size over which the autocorrelation was taken, and τ is the time lag associated with a specific autocorrelation sample. When the number of standard deviations above the mean for any time lag in the window (but the zeroth) is above a certain threshold, a packet is detected in that window. The reason the zeroth time lag is not used is because at this lag, the output from the autocorrelation is simply the power in the received waveform. This method works best when using a large window size that encompasses many symbol boundaries. If Figure 10 were the real packet structure used for this project, the window size would be 37 samples. The first window would start at sample #1 and end at sample #37, while the second window would start at sample #38 and continue to sample #74.

To simplify the detection module, the window size used will be exactly the same size as one packet, as well as the space between each packet. Note that this may unfairly help the detection module. In a real-world situation, the intercept would likely not know the length of the packet exactly. However, to reduce the difficulty of analysis of data, the windows will either contain a packet or it will not. The easiest

Spreading Code Index	Chip Positions																																					
	Packet 1																																						
0	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B	8A	8B																						
1	1A	1B		2A	2B		3A	3B		4A	4B		5A	5B		6A	6B		7A	7B		8A	8B															
2	1A	1B			2A	2B			3A	3B			4A	4B			5A	5B			6A	6B			7A	7B			8A	8B							
3	1A	1B				2A	2B				3A	3B				4A	4B				5A	5B				6A	6B			7A	7B				8A	8B	
Sample Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	

	Inter-Packet Space																																					
0																																						
1																																						
2																																						
3																																						
Sample Number	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	

Figure 10. Packet and inter-packet lengths with 4 unique spreading codes and a code length of 2

way to accomplish this is to force the window size to be the same length as the packet length, which is the same length as the space between packets. The length of the packets is given in Equation (4). Figure 10 shows this more clearly. The numbers in the far left column of Figure 10 are the index of the spreading code for a given channel. In the remaining columns, the output from each channel is given. For the cells with a number and letter in them, the number represents the bit and the letter represents the chip of the spreading code currently being transmitted by that channel,. The last row in each section gives the sample number of that column. So, during sample 11, channel 0 is transmitting the first chip of bit 6, channel 1 is transmitting the second chip of bit 4, channel 2 is not transmitting, and channel 3 is transmitting the first chip of bit 3. During the packet space period, shown in the bottom section of Figure 10, no channels are transmitting anything.

Because the communication system developed is packet based, the measure of detection used for the system will be the percentage of packets that were detected as being present compared to the total number of packets transmitted. The BERs at the receiver will also be measured to prove that the new communications scheme does not reduce overall performance of the system. In the analysis stage, ROC curves will be generated for each run.

3.3 Software Description

The software components used for this project are discussed. There are multiple types of tools developed to aid in the completion of this project. These include GNU Radio blocks, MATLAB® scripts, Python Scripts, and Bash scripts.

3.3.1 GNU Radio Blocks.

This section will describe the blocks that were developed or used to provide new functionality in the GNU Radio environment. Blocks with citations at the end are provided in the standard GNU Radio modules, while those that those that are specified "Custom" were designed specifically for this project. Most of the functionality was divided into C++ classes to provide modularity and portability, as well as to reduce the duplication of code.

3.3.1.1 Modulator.

The following blocks and C++ classes were used to easily modulate signals with a high degree of customizability.

Unpack K Bits - The purpose of this class is to unpack bits in preparation for processing. Transporting packed bytes is much easier due to the compressed size, but processing unpacked bytes is far easier due to the fact that there is only one item per byte to deal with [35].

Spreading Code Class - Custom This class provides methods to read spreading code files, store spreading codes, and to use the spreading codes to spread and despread signals. It contains special despreading methods designed to combine the results from multiple channels to estimate bits in unison.

Map Bits Class - Custom - This class provides methods to map bits to symbols and vice versa.

Modulator Class - Custom - This class provides methods to modulate packed bytes to symbols. It uses the three classes described above to accomplish this. First, it unpacks the bytes, then spreads the data stream and maps it to symbols for each unique spreading code.

Packet Spacer Class - Custom - This class provides methods to add space between packets, to re-sample at an integer ratio, and to invert these actions.

Transmitter - Custom - This block instantiates the above described components and uses them to perform the transformation of a byte stream into a packet-modulated stream of samples. This block can perform BPSK, DSSS and Inverse Beamforming, all depending on the inputs. The block has one input, the data stream to be transmitted, and N outputs, the number of spreading codes to be used in the transmission. Note that BPSK modulation can be accomplished by setting $N = 1$ and setting the spreading code used to $\{1\}$.

3.3.1.2 Channel.

The channel blocks are used to simulate real world effects that could be present in a channel.

Noise Source - This block provides AWGN with a given standard deviation [36].

Adder - This block sums the inputs from each of the inputs streams (in a synchronous fashion) and outputs the result. This is needed for two reasons. First, the output from the transmitter is such that there are N different sample streams (one for each unique spreading code), and second, the AWGN also needs to be added to obtain the final signal [34].

3.3.1.3 Demodulator.

The following blocks were created to fill gaps in the existing GNU Radio blocks for demodulation of DSSS/CDMA signals.

Pack K Bits - The purpose of this class is to pack bytes. Since the input to the modulator is packed bytes, the output from the demodulator should be packed bytes as well for consistency [37].

Receiver - Custom - This block first removes the packet spacing and resampling done in the transmitter, then despreads each signal and uses them to estimate the original bits. Note that this block also uses the spreading code, map bits, and packet spacer classes described in the modulator section to perform these actions.

3.3.1.4 Autocorrelation Detector - Custom.

The autocorrelation detector first autocorrelates the received signal with itself. It then finds the mean value for the window, then the number of standard deviations each point lies from the mean. If any of the points within the window are above a certain threshold number of standard deviations above the mean, the block outputs that it detected a packet in that window.

3.3.1.5 GNU Radio Flow Graph.

Figure 11 shows the flow graph that was used to construct the more specialized version for all simulations. The reason this exact flow graph was not used is that there are some capabilities that the GRC cannot provide, most notably switches between use of blocks based on variable values. However, GRC was used to generate a basic python script, then that script was modified to include the switches.

3.3.1.6 Python Utilities.

Python utilities were developed to make constructing instances of various blocks easier. These utilities include functions to read headers and data from files, read spreading codes from files, and spread data.

3.3.1.7 Matlab® Utilities.

MATLAB® functions were used in two different stages. First, in debugging it was useful to generate scripts to plot correlations of demodulated bits with the bits that were originally transmitted. Later on, it was used to generate plots for this thesis.

3.3.1.8 Bash Utilities.

Multiple Ettus SDR devices are used to complete simulations, so Bash shell scripts were created to automate a few different tasks. First, since the GNU Radio modules were developed on a laptop separate from the SDRs, the modules needed to be copied to each SDR, then compiled and installed. Another task was to copy simulation data from the SDRs back to the laptop. Bash shell scripts were made to accomplish both of these tasks easily.

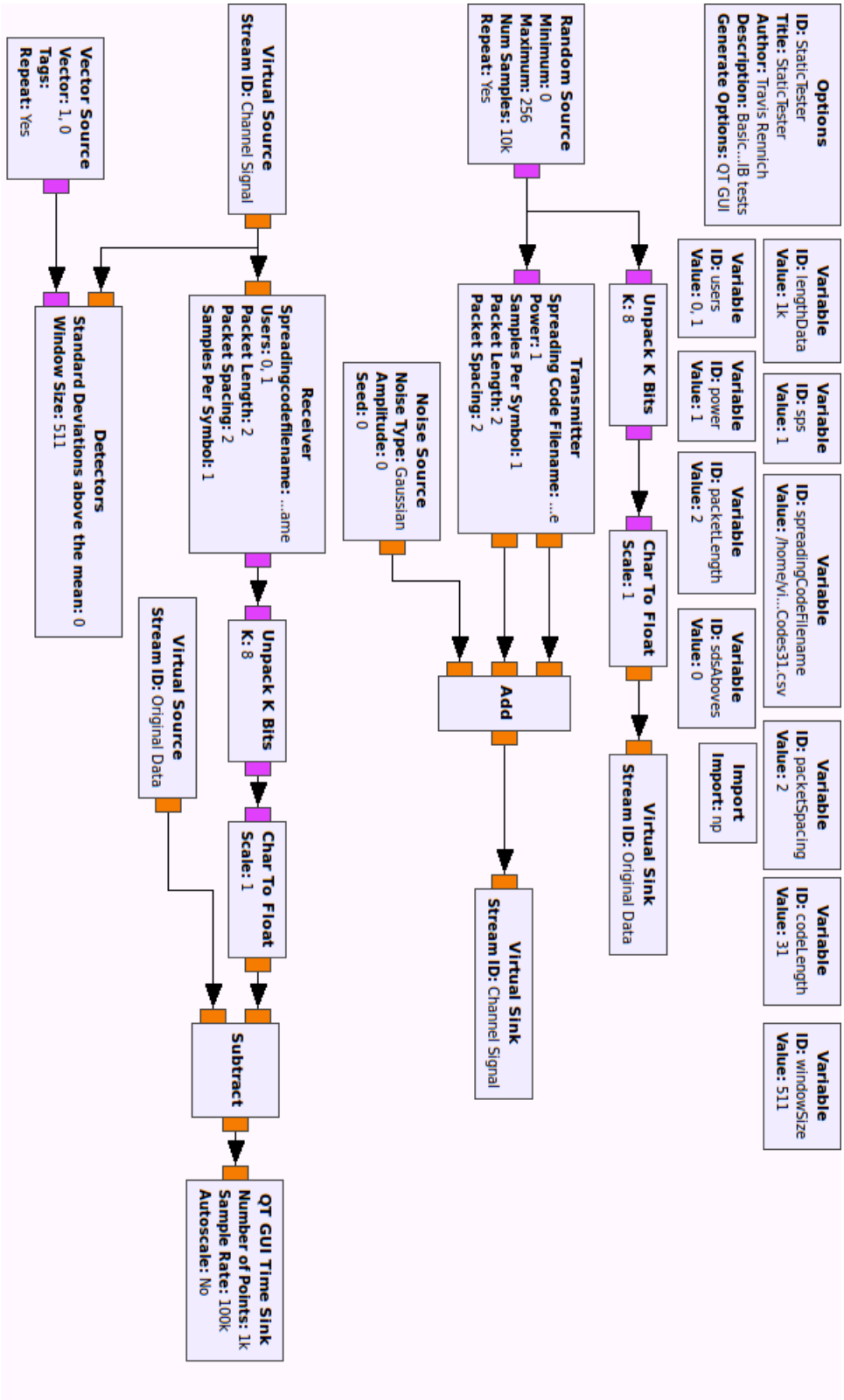

Figure 11. Base GNU Radio flow graph used for all simulations

3.4 Simulation Validation

This section describes the validation simulations that will take place. These simulations will be performed to guarantee that each piece of the project works as described.

3.4.1 Validation of Modulations.

Initial validation of modulations will take place to ensure that all future data is valid. Results from this step are compared to theoretical results to ensure that all systems are operating as they theoretically should. An overview of these simulations is given in the next sub-sections.

3.4.1.1 Validation of BPSK Modulation.

To validate that the general BPSK modulator used for this project works correctly, a BER curve over varying SNRs will be generated. This will consist of a GNU Radio simulation in which noise is artificially added to the received signal. Table 1 shows the noise amplitudes used for these simulations. The results from this stage will be compared to theoretical predictions as made by Equation (6) [7].

$$P_B = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{6}$$

where P_B is the probability of bit error, E_B is the average received energy per bit, and N_0 is the average received noise power (assuming a two-sided frequency spectrum of $\frac{N_0}{2}$). When the simulations match the theoretic predictions to the 95% confidence level, the second stage of validation will start.

Table 1. Noise standard deviation used in the validation of BPKS modulation and demodulation. Signal Amplitude $= 1$

Start of Range	Increment	End of Range	Total Number of Runs in Range
0	0.1	2	21
2.5	0.5	20	36

Table 2. Noise standard deviation used in the validation of the CDMA system. Signal amplitude $= 1$

Start of Range	Increment	End of Range	Total Number of Runs in Range
0	0.1	2	21
2.5	0.5	20	36

3.4.1.2 Validation of BPSK and CDMA Modulation.

After the performance of simple BPSK modulation has been validated, the spreading code will be changed to Gold codes of length $N_L = 31$. We must also ensure that the CDMA system can function in the presence of other users, not just noise. The noise amplitudes found in Table 2 will be used to validate the performance of the CDMA system.

First, basic functionality will be tested. These tests will be constructed very similarly to the BPSK validation tests. The test will be a simulation with widely varying noise amplitudes to obtain a BER curve. These results will be plotted and compared to theoretic results given by Equation (6).

In the second stage, multi-user functionality will be tested. These tests will vary both the number of other users of the same frequency band, as well as the SNR. For decreased complexity, it will be assumed that the signal powers of all users are at the same level at the receiver. For the purposes of this thesis, this is a valid assumption because the signals from each modulation chain will be transmitted at the same power and there will not be a large difference in attenuation between any of the transmit antenna - receive antenna pairs.

3.5 Simulation

After the validation of each part of the experiment has taken place, real simulation data can be taken. This section describes the procedures that will be used to collect and record the data from each simulation, as well as the simulation hierarchy.

3.5.1 Simulation setup.

As discussed previously, all runs will be conducted from the same simulation script that modifies the flow graph used before starting each simulation. The simulations will be performed on 6 different platforms. The first is HP Envy dv6 laptop with 8 GB of RAM and a Intel Core i7-3630QM 2.4 GHz $\times 8$ processor. The laptop runs Ubuntu 16.04 LTS, GNU Radio version 3.7.7. The second type of machine is an Ettus Research USRP E310. These devices have an ARM Cortex A9 667 MHZ dual-core processor [38]. The last type of machine used for simulations is the Ettus Research USRP E312 with an ARM Cortex A9 866 MHZ dual-core processor [39]. The Ettus USRP E310 SDRs run GNU Radio version 3.7.7, while the Ettus USRP E312 devices run version 3.7.9.

3.5.2 Simulation Procedure.

The simulation script has the capability to run many separate simulations in series so that the experimenter can start the series and check back when all are done. This functionality will be used to vary the noise standard deviation for each basic setup. The first type will be a BPSK system. The purpose of this test is to provide a baseline measure for detection and BER performance. The second type will be a single spreading code CDMA system. The purpose of this is again to provide a control against which to compare both the BPSK system and later multiple CDMA system performances. The final type will be the multiple spreading code CDMA system,

in which the Inverse Beamforming idea is used to provide a more covert system. The results from later tests will be compared with the results from earlier tests in order to validate conclusions. Each test will be divided into multiple runs that vary transmission and reception parameters.

3.5.2.1 Definition of a Run.

For the purposes of this experiment, a run will consist of transmitting a number of packets from the transmitting block with fixed delays between packets. The friendly receiving block will attempt to demodulate the signal, and the BER for the packets will be recorded. The non-cooperative detector block will attempt to detect when packets were transmitted. A ROC curve is generated with the results of the detector for relevant thresholds. The following parameters define a run:

- Spreading code(s)
- Noise power/standard deviation
- Number of packets

3.5.2.2 Parameter Settings.

Before the start of each run, all parameter settings will be recorded to ensure that all results get analyzed correctly. The settings are automatically recorded by the simulation script and are stored to a formatted file before the simulation even starts. When there are multiple simulations run sequentially, the parameter settings are re-stored for each individual run, ensuring to keep track of any changes to the parameters since the last run started.

3.5.2.3 Measured Quantities.

The data that will be immediately recorded during each run are:

- Cooperative demodulation receiver
  - Total number of bits simulated in the run.
  - Total number of incorrectly estimated bits.
- Non-cooperative detection receiver
  - Number of true positive detections (1 value for each threshold).
  - Number of false positive detections (1 value for each threshold).
  - Number of true negative detections (1 value for each threshold).
  - Number of false negative detections (1 value for each threshold).
  - Number of predicted detections (1 value for each threshold).
  - Number of predicted non-detections (1 value for each threshold).
  - Number of detection windows that actually held packets.
  - Number of detection windows that did not actually hold packets.
  - Total number of packets

Each ROC curve point is processed during the simulation to efficiently use storage space. The simulation only requires enough storage to hold six arrays the length of the threshold array, plus three scalars to make analysis of the data more efficient. The BER data can be calculated from only two values, the total number of bits and the total number of incorrectly estimated bits. The total memory requirement for each run is then:

$$\text{Memory Required} = (5 + 6 \times \text{Number of Thresholds}) \times \text{ sizeof(unsigned long)} \quad (7)$$

3.5.3 Conduct Runs.

Runs will be conducted for the settings shown in Tables 4-6 in Appendix A. Note that the right columns of Tables 4-6 show the number of **packets**, not bits. To get the number of bits simulated, simply use Equation (8).

$$\text{Total Bits} = \text{Number of Packets} * \text{Bytes Per Packet} * 8 \tag{8}$$

For this project, there are 2 bytes per packet, and therefore 16 bits per packet. Also note that noise amplitudes are used because the GNU Radio block used to generate the noise accepts an amplitude, not power. As noted previously, the runs will be conducted in three separate stages: the first using only BPSK modulation, the second using DSSS modulation with a single spreading code, and the third using CDMA modulation with multiple spreading codes.

Shortly put, noise amplitudes for each stage of validation will start at 0, then increase by 0.1 until 2. At 2, the increment switches to 0.5 and continues to 20. The number of packets simulated will change depending on the BER for that specific setting, i.e. a setting with a very low BER will likely get many more packets simulated at that setting than when the BER approaches $0.5 * 10^{-1}$. Also, the indices of spreading codes used when multiple codes are used start at 2 due to the fact that the cross correlation properties of Gold codes are not as good when the two maximal length sequences that were used to generate the family are used in the simulation. However, for single spreading code cases, these properties do not matter.

3.5.3.1 BPSK.

The reason for including this stage of runs in the experiment is to ensure that there is a baseline performance metric for the other two stages. Because the next two stages both build on simple BPSK modulation, this is a good control group.

3.5.3.2 Single DSSS.

This set of runs is meant to be a more advanced control group for the Inverse Beamforming set of runs. There exist many deployed systems that use single DSSS as the basis for their communications scheme, and that is what we want to compare Inverse Beamforming to.

3.5.3.3 Multiple DSSS/Inverse Beamforming.

This set of runs is the novel portion of the experiment. The results gathered in this stage will be compared to the previous two stages to gage success of the project. Due to the length of time required to run simulations for larger code lengths, initial simulations will be run with smaller codes (Gold codes in the length $N_L = 31$ family). In this stage, the number of spreading code channels in the transmitter and receiver (N_C) is varied. After noting general trends, larger codes will be simulated at a lower resolution of noise amplitudes.

3.5.4 Analysis of Data.

Analysis of the data gathered will include multiple steps. The first will be to compare the BERs of the single DSSS stage with the BPSK stage. The next step will be to ensure that the DSSS system functions effectively. Next, the validation of the CDMA functionality will take place, followed by the analysis of Inverse Beamforming simulations.

IV. Results

This chapter will present and discuss the results from each of the simulations. First, in Section 4.1, the basic properties of the simulation tools will be validated, then in Section 4.2 details of Inverse Beamforming simulations will be presented. Finally, comparisons between the Inverse Beamforming simulations with and without an added chip offset will be provided in Section 4.3.

4.1 Validation Simulations

4.1.1 BPSK Modulation Validation.

Figure 12 shows that the BER of the BPSK runs agree with the theoretical predictions according to Equation (1) to within 95% confidence intervals. Note that the confidence intervals are shown in Figure 12, but are very small and are hard to see. The inset shows the range between E_B/N_0= 4.9-5.

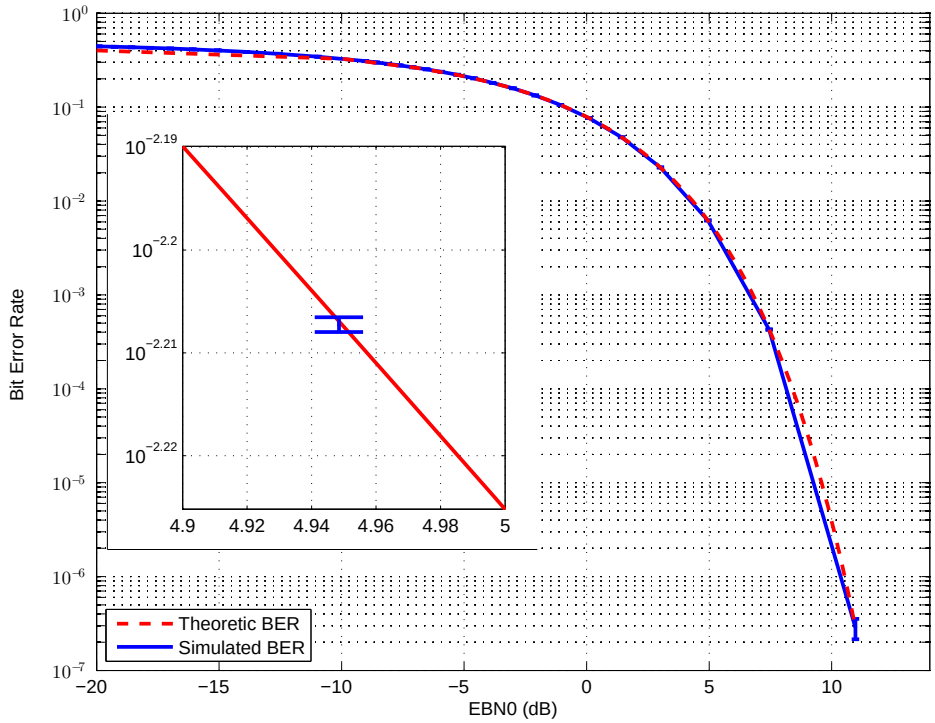

Figure 12. Results from BPSK validation run

4.1.2 Single DSSS Modulation Validation.

Figure 13 shows that the BER of the single DSSS validation runs do not perfectly agree with theoretical predictions according to Equation (1) to within 95% confidence intervals. This is likely due to the noise bandwidth for the theoretical prediction not being equivalent to the noise bandwidth for the single DSSS validation runs.

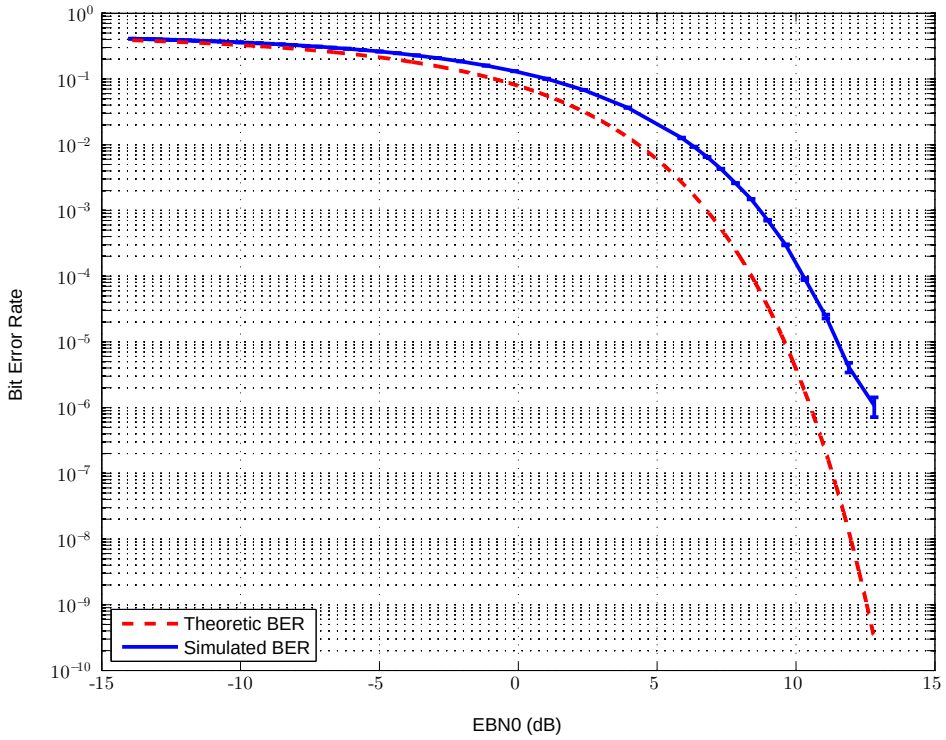

Figure 13. Results from single DSSS validation run

4.1.3 Multiple CDMA Modulation Validation.

Figure 14 shows that the BER of the multiple CDMA validation runs also do not agree perfectly with theoretical predictions according to Equation (1). However, in this case there are multiple users of the system so there is an expectation the BERs will drop.

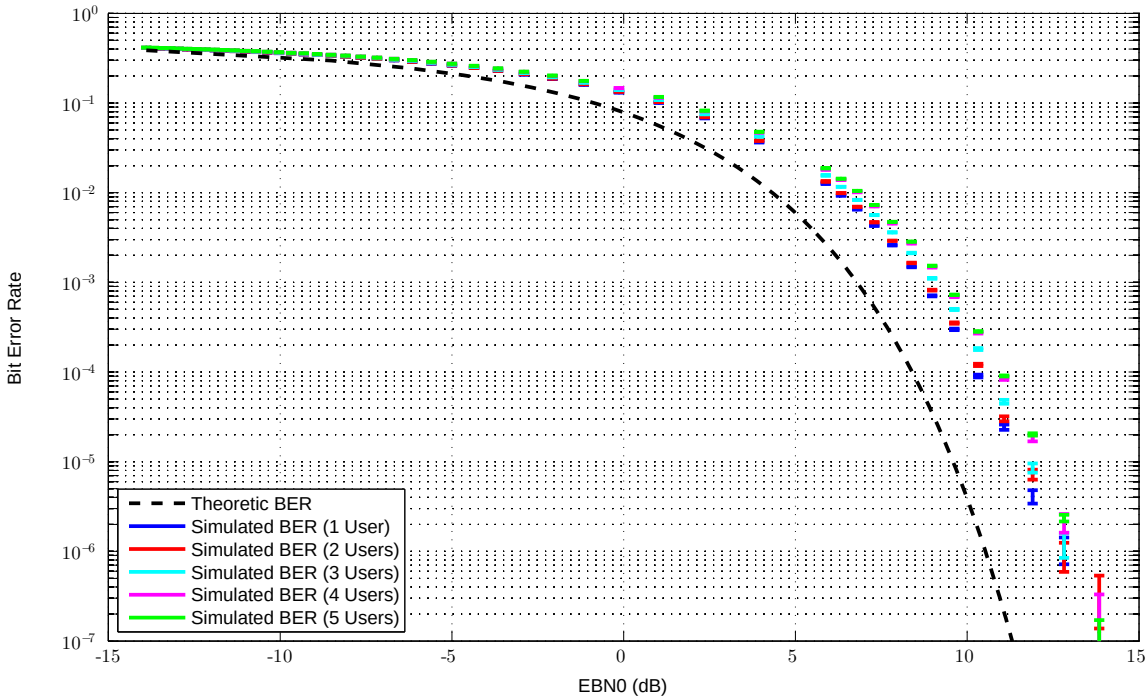

Figure 14. Results from Multiple CDMA validation run

One of the original ideas of Inverse Beamforming was that simply transmitting the same data message using multiple different spreading codes would be enough to lower detection performance. This may be true in the real world where there would likely be some timing and frequency offsets between each of the user channels, even if they are all transmitted from the same hardware device. However, in these simulations, there was no such timing or frequency offsets introduced, so the effective transmitted signal was basically equivalent to using one spreading code that was the summation of the actual spreading codes used.

There are two reasons that could explain the higher BERs associated with the multiple CDMA simulations. Firstly, the effective summed spreading code is multi-valued in its amplitude, which means that it could be more sensitive to noise. Secondly, the receiver is still trying to de-spread and demodulate via the original spreading codes. Gold codes are known for their good autocorrelation properties but are not perfectly

orthogonal. For these reasons, the chip-offset Inverse Beamforming technique was developed. The results from these simulations are shown in the next section.

4.2 Inverse Beamforming Simulations

This section will show and explain the results of the main simulations of this project. Table 3 gives a layout of all of the figures in this section and their purpose. All simulations past this point make use of the chip offset to try to lower detection performance.

4.2.1 Inverse Beamforming with Chip Offsets, Code Length $N_L = 31$.

Due to the increasing amounts of time required to simulate large code lengths, a code length of $N_L = 31$ was tested to provide a proof of concept as well as areas of interest before moving to larger codes with limited data points.

Figure 15 shows the BERs of the Inverse Beamforming simulations with $N_C = 1$ to -$N_C = 4$ users and a code length of $N_L = 31$. The legend shows the BER of the friendly receiver under the same noise conditions, as well as the Area Under the Curve (AUC) metric. This metric calculates the area under the detection curve. The bounds on the metric are $0.5 < AUC < 1$. The closer to 1 the AUC is, the more perfectly the detector performed. The detection results from more selected noise amplitudes are shown in Figures 21-26 in Appendix B. The BERs of the simulations with different numbers of spreading codes show an increasing difference with an increase in the E_B/N_0. This is likely due to the fact that the inter-spreading code interference is a larger relative noise source when the Gaussian noise is lower powered. Therefore, it is unsurprising that the single spreading code case maintains the lowest BER in the higher E_B/N_0's.

Figure 16 shows a selected ROC curve at an E_B/N_0= 5.88 dB. At this E_B/N_0,

Table 3. Explanation of the order of figures presented in this section

Figure Number	Caption
15	BER of Inverse Beamforming runs with $N_C = 1$ to $N_C = 4$ spreading code channels and a code length of $N_L = 31$
16	Selected ROC curve of Inverse Beamforming runs with $N_C = 1$ to $N_C = 4$ spreading code channels, code length of $N_L = 31$ and E_B/N_0= 5.88 dB
17	BER of cooperative receiver (top) and EER of intercept receiver (bottom) of Inverse Beamforming runs with $N_C = 1$ to $N_C = 4$ spreading code channels and a code length of $N_L = 31$
18	BER of Inverse Beamforming runs with $N_C = 1$ to $N_C = 6$ spreading code channels and a code length of $N_L = 255$
19	Selected ROC curve of Inverse Beamforming runs with $N_C = 1$ to $N_C = 6$ spreading code channels, code length of $N_L = 255$ and E_B/N_0= 15.03 dB
20	BER of cooperative receiver (top) and EER of intercept receiver (bottom) of Inverse Beamforming runs with $N_C = 1$ to $N_C = 6$ spreading code channels and a code length of $N_L = 255$

the $N_C = 2$ case is detected better than the $N_C = 1$ case, and has a higher BER. However, the $N_C = 3$ and $N_C = 4$ cases are detected more poorly than either the $N_C = 1$ and $N_C = 2$ cases, although the BER is still higher in both cases.

Figure 17 shows both the BER, top, and EER, bottom for each number of spreading codes. Note that the higher the EER, the worse the detector performance, which is a good thing in the context of this thesis. When looking at EER plots and ROC curves simultaneously, note that a lower EER tends to mean coincide with a ROC curve that tends towards the top left point (0,1), or the perfect classification point. In the context of this thesis, it is better for the the EER to be high, and the ROC curve to approach the diagonal line from (0,0) to (1,1). As evident in Figure 17, the detector performance is best for the $N_C = 2$ case, and the poorest for the $N_C = 4$ case. The $N_C = 4$ case proves the concept that Inverse Beamforming with chip offsets

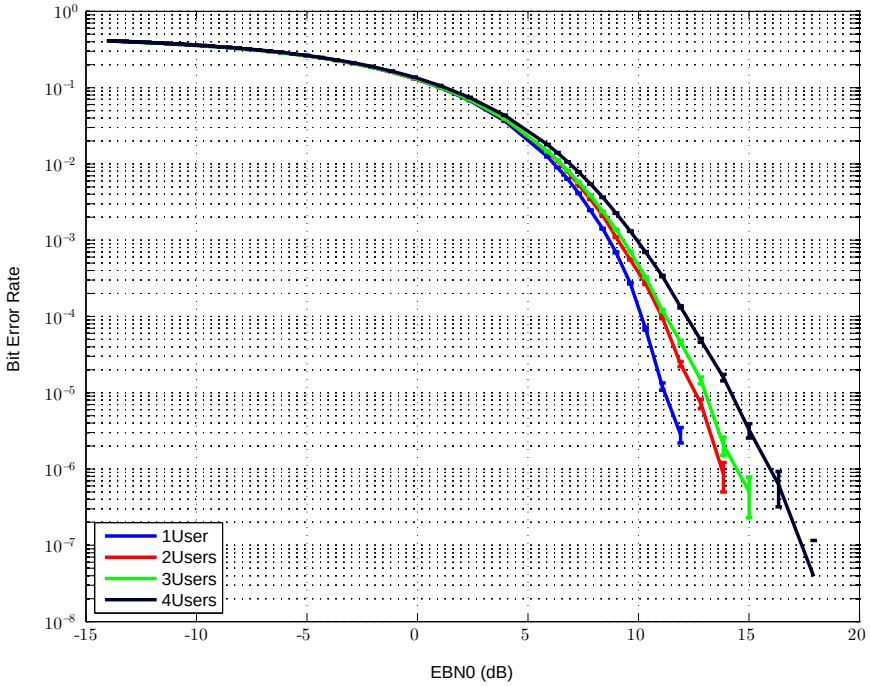

Figure 15. BER of Inverse Beamforming runs with $N_C = 1$ to $N_C = 4$ spreading code channels and a code length of $N_L = 31$

can work, but with a code length of 31, the BER increases too much for the decreased detection performance to overcome. Longer length codes are needed to further prove the concept.

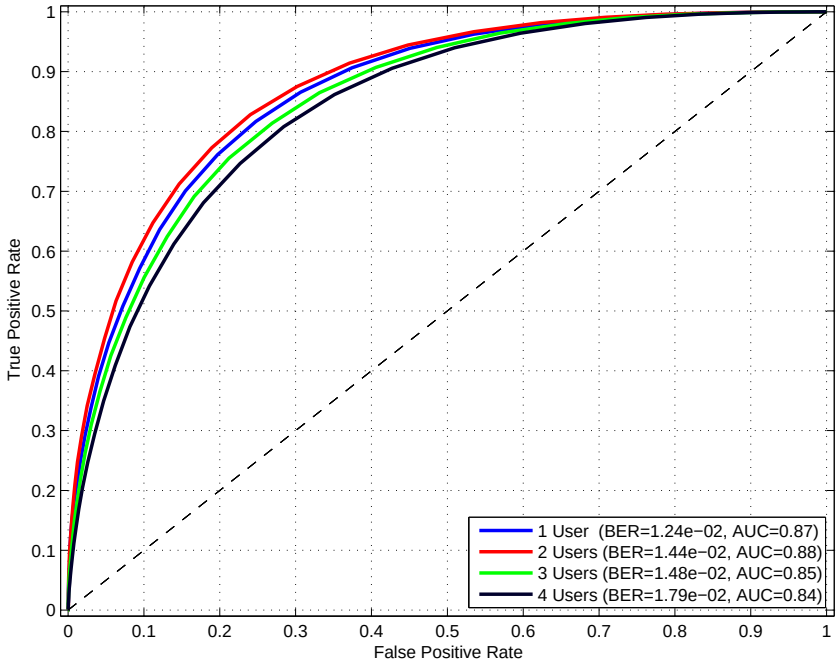

Figure 16. Selected ROC curve of Inverse Beamforming runs with $N_C = 1$ to $N_C = 4$ spreading code channels, code length of $N_L = 31$ and $E_B/N_0 = 5.88\ dB$

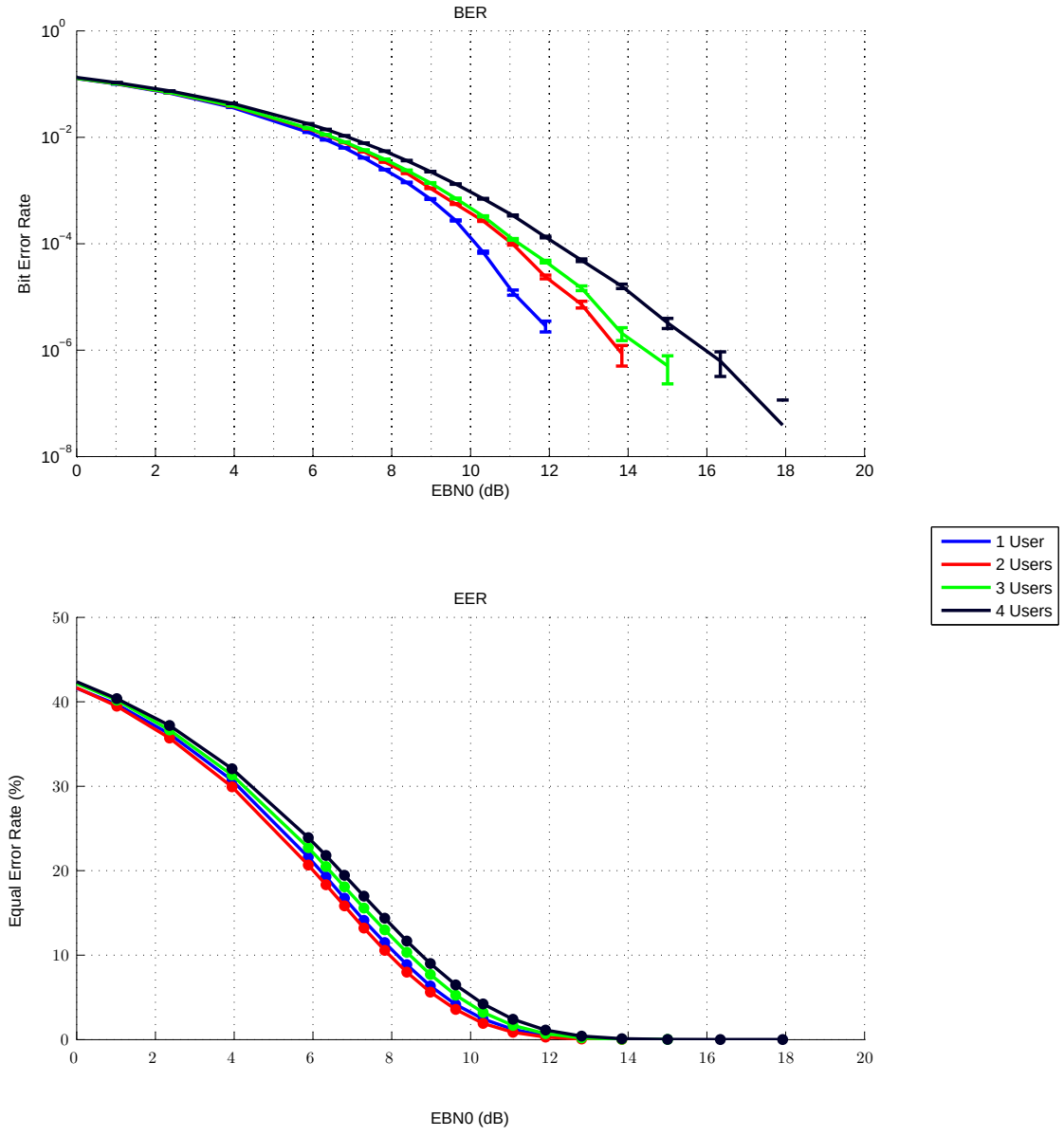

Figure 17. BER of cooperative receiver (top) and EER of intercept receiver (bottom) of Inverse Beamforming runs with $N_C = 1$ to $N_C = 4$ spreading code channels and a code length of $N_L = 31$

4.2.2 Inverse Beamforming, Code Length $N_L = 255$.

Figure 18 shows the BERs of the Inverse Beamforming simulations with $N_C = 1$ to $N_C = 6$. The detection results from all simulated noise powers are shown in Figures 27-36 in Appendix C. With the longer spreading codes, the BER is much more constant with the number of spreading code channels (N_C) than with the $N_L = 31$ codes, especially in the higher E_B/N_0's. This is expected because the code length to number of users ratio is much larger. This allows each code to depend much less on the integrity of each chip, and decreases the magnitude of the inter-code noise.

Figure 19 shows a ROC curve at a single noise power. From this figure, it is clear that adding spreading code channels with increasing chip offsets lowers the detection performance with small changes in the BER. In the case of $N_C = 2$, the BER actually decreases with a decrease in detection performance. However, it appears that there may be a limit to the number of spreading code channels (N_C) that can be used before the detection performance starts to increase again. According to Figure 19, the case of $N_C = 3$ seems to provide the worst detection performance with only a slight increase (˜15%) in the BER.

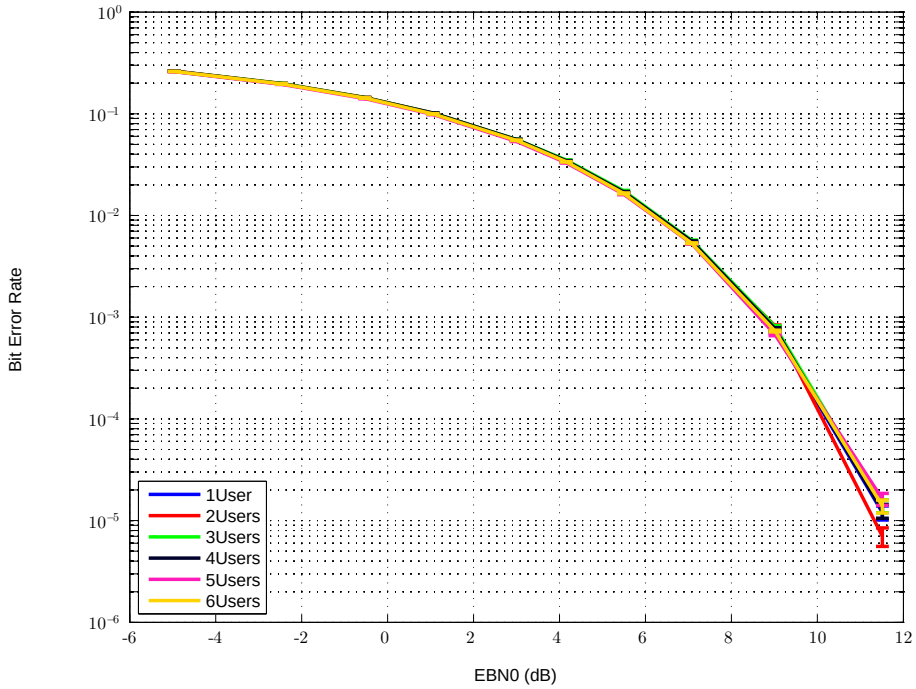

Figure 18. BER of Inverse Beamforming runs with $N_C = 1$ to $N_C = 6$ spreading code channels and a code length of $N_L = 255$

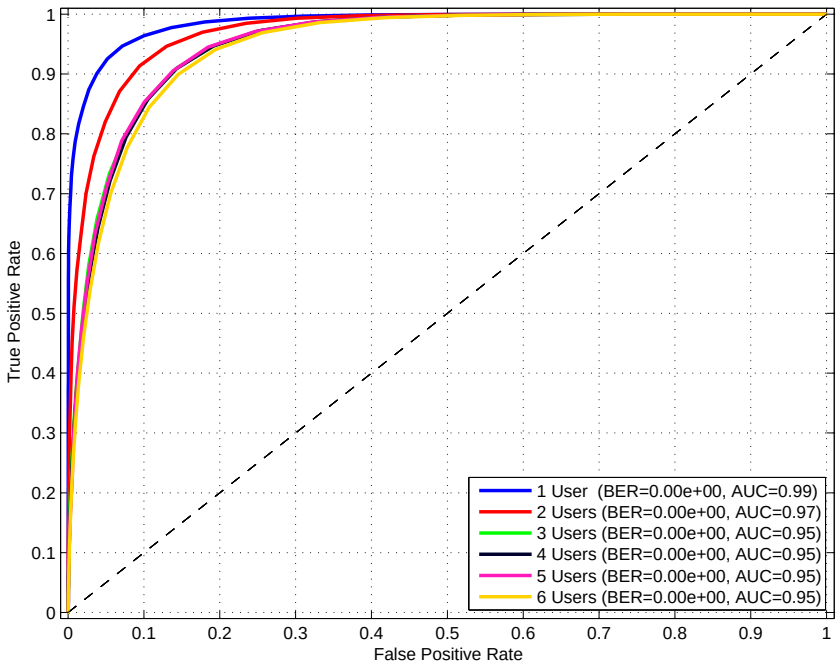

Figure 19. Selected ROC curve of Inverse Beamforming runs with $N_C = 1$ to $N_C = 6$ spreading code channels, code length of $N_L = 255$ and E_B/N_0= 15.03 dB

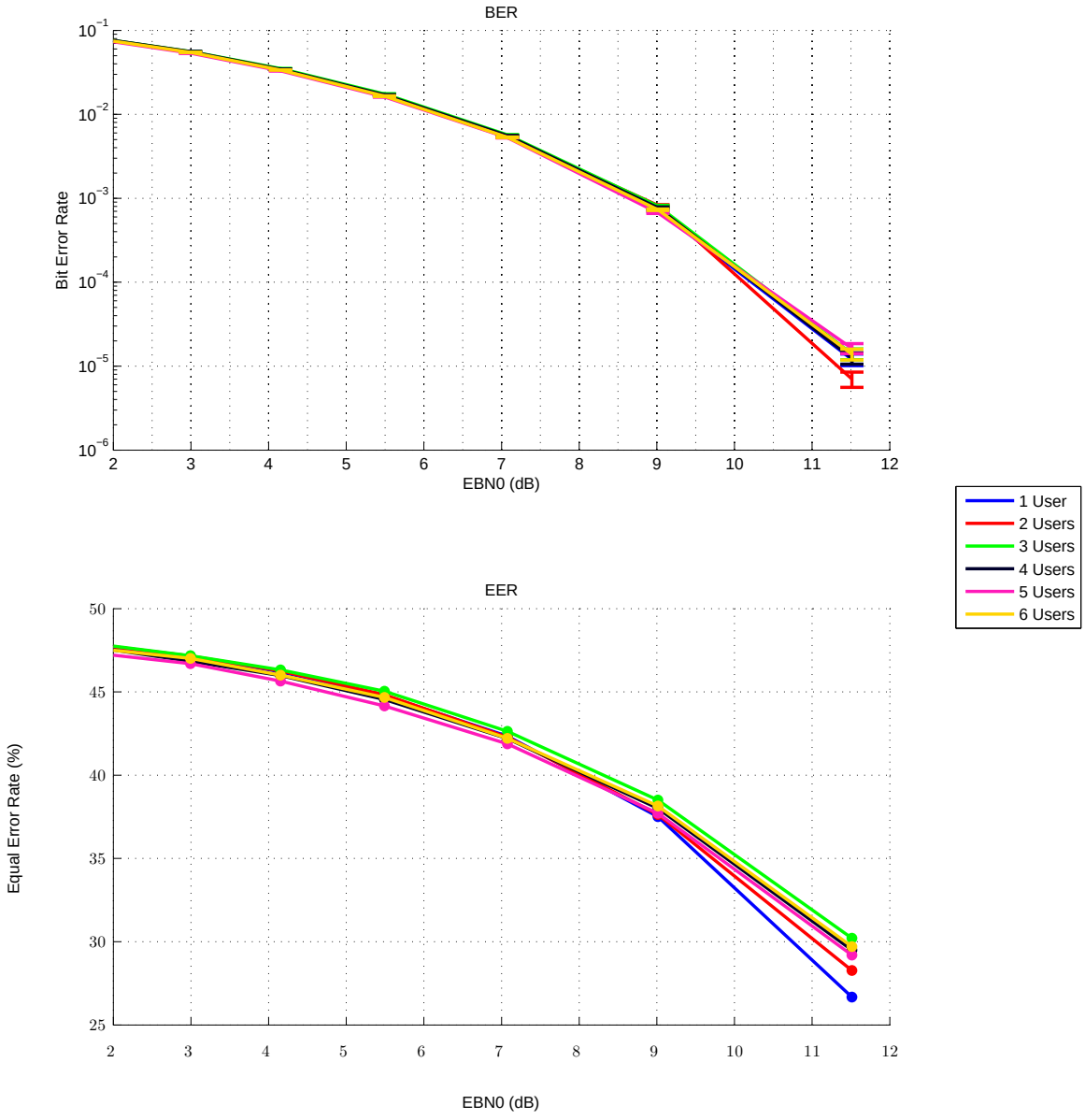

Figure 20. BER of cooperative receiver (top) and EER of intercept receiver (bottom) of Inverse Beamforming runs with $N_C = 1$ to $N_C = 6$ spreading code channels and a code length of $N_L = 255$

Figure 20 shows both the BER, top, and EER, bottom for each number of spreading codes (N_C). For lower E_B/N_0values, the $N_C = 2$ case provides the best detection performance, while the $N_C = 3$ case provides the worst detection performance for all E_B/N_0. Since the BER of the $N_C = 3$ case is essentially equivalent to the single spreading code case, Inverse Beamforming with a chip offset shows potential as a more covert way to communicate than existing DSSS techniques.

4.3 Comparisons

The shorter code length in the first set of simulations allowed a much smaller resolution of noise powers. However, the $N_L = 255$ simulations are more realistic due to their length being closer to real-world systems [15].

The addition of the incrementing chip offset between codes of separate channels clearly decreases the detection performance of an intercept receiver. One possible avenue for the decrease in performance is the spreading of the energy over time. The intercept receiver now must make the detection windows larger to account for the fact that the signal energy is more spread out in time. However, this also captures more noise, which the receiver will have to ignore.

V. Conclusion

5.1 Summary

Inverse Beamforming is a concept that allows for the transmission of a data message with a decreased probability of detection and a constant level of friendly performance (BER) as compared to simple DSSS systems. The technique shows promise in increased threat areas such as the modern battlefield. Soldiers could use Inverse Beamforming with chip offsets to communicate with one another with a decreased level of fear of discovery. The basic concept behind Inverse Beamforming was explained in Chapter II.

The goal of this research was to develop a method to communicate between an array of transmitting antennas and a single receive antenna that maintains a constant BER, while the intercept receiver performance decreases. Chapter III explains the steps used to verify that this has been accomplished.

Initial simulation results show that Inverse Beamforming can work; the BER remains constant with an increase in number of spreading codes used and show a decrease in the intercept receiver detection performance. This was later confirmed by the simulation of longer $N_L = 255$ spreading code where the BER remained constant while increasing the number of user channels and detection performance dropped.

5.2 Future Work

There are many areas in which to continue the research presented here. First of all, the existing GNU Radio blocks require that the entire packet is processed at once. This works well for smaller spreading code lengths and low numbers of samples per symbol, but with larger values for these parameters, the packet length quickly exceeds the maximum buffer size available to GNU Radio blocks. Currently the longest code

length available for simulation is $N_L = 255$. After implementing this change however, the only limitation to code length would be the amount of on-board Random Access Memory (RAM).

Another area of continuing research would be to develop a method to synchronize to the individual spreading codes. Because they are at a much lower power than normal DSSS signals, this might prove to be an extremely challenging task.

The final area of future work is to add functionality to locate and demodulate the Inverse Beamforming signals in hardware. This process will combine the efforts presented in this paper as well as the two areas presented above.

Appendices

LOW PROBABILITY OF DETECTION COMMUNICATION USING
INVERSE BEAMFORMING IN GNU RADIO
USING CODE DIVISION MULTIPLE ACCESS

A. Validation Run Details

Table 4. Settings used in the validation of BPSK modulation.

Run Number	Noise Amplitude	E_B/N_0	Number of Packets
1	0.00	Inf	14040000
2	0.10	16.99	14040000
3	0.20	10.97	14040000
4	0.30	7.45	14040000
5	0.40	4.95	14040000
6	0.50	3.01	14040000
7	0.60	1.43	14040000
8	0.70	0.09	14040000
9	0.80	-1.07	14040000
10	0.90	-2.10	14040000
11	1.00	-3.01	14040000
12	1.10	-3.84	14040000
13	1.20	-4.59	14040000
14	1.30	-5.29	14040000
15	1.40	-5.93	14040000
16	1.50	-6.53	12076073
17	1.60	-7.09	4040000
18	1.70	-7.62	4040000
19	1.80	-8.12	4040000
20	1.90	-8.59	4040000
21	2.00	-9.03	202000
22	2.50	-10.97	202000
23	3.00	-12.55	202000
24	3.50	-13.89	202000
25	4.00	-15.05	202000
26	4.50	-16.07	202000
27	5.00	-16.99	202000
28	5.50	-17.82	202000

29	6.00	-18.57	202000
30	6.50	-19.27	202000
31	7.00	-19.91	202000
32	7.50	-20.51	202000
33	8.00	-21.07	202000
34	8.50	-21.60	202000
35	9.00	-22.10	202000
36	9.50	-22.56	202000
37	10.00	-23.01	202000
38	10.50	-23.43	202000
39	11.00	-23.84	202000
40	11.50	-24.22	202000
41	12.00	-24.59	202000
42	12.50	-24.95	202000
43	13.00	-25.29	202000
44	13.50	-25.62	202000
45	14.00	-25.93	202000
46	14.50	-26.24	202000
47	15.00	-26.53	202000
48	15.50	-26.82	202000
49	16.00	-27.09	202000
50	16.50	-27.36	202000
51	17.00	-27.62	202000
52	17.50	-27.87	202000
53	18.00	-28.12	202000
54	18.50	-28.35	202000
55	19.00	-28.59	202000
56	19.50	-28.81	202000

Table 5. Parameter settings for validation of Simple DSSS modulation/demodulation.

Run Number	Spreading Code Index	Noise Amplitude	E_B/N_0	Number of Packets
1	1	0.00	Inf	12040000
2	1	0.10	31.90	12040000
3	1	0.20	25.88	11280616
4	1	0.30	22.36	2040000
5	1	0.40	19.86	2040000
6	1	0.50	17.92	2040000
7	1	0.60	16.34	2040000
8	1	0.70	15.00	2040000
9	1	0.80	13.84	2040000

10	1	0.90	12.82	2040000
11	1	1.00	11.90	2040000
12	1	1.10	11.08	2040000
13	1	1.20	10.32	2040000
14	1	1.30	9.62	2040000
15	1	1.40	8.98	2040000
16	1	1.50	8.38	2040000
17	1	1.60	7.82	2040000
18	1	1.70	7.29	2040000
19	1	1.80	6.80	2040000
20	1	1.90	6.33	2040000
21	1	2.00	5.88	102000
22	1	2.50	3.94	102000
23	1	3.00	2.36	102000
24	1	3.50	1.02	102000
25	1	4.00	-0.14	102000
26	1	4.50	-1.16	102000
27	1	5.00	-2.08	102000
28	1	5.50	-2.90	102000
29	1	6.00	-3.66	102000
30	1	6.50	-4.35	102000
31	1	7.00	-5.00	102000
32	1	7.50	-5.60	102000
33	1	8.00	-6.16	102000
34	1	8.50	-6.69	102000
35	1	9.00	-7.18	102000
36	1	9.50	-7.65	102000
37	1	10.00	-8.10	102000
38	1	10.50	-8.52	102000
39	1	11.00	-8.92	102000
40	1	11.50	-9.31	102000
41	1	12.00	-9.68	102000
42	1	12.50	-10.03	102000
43	1	13.00	-10.38	102000
44	1	13.50	-10.70	102000
45	1	14.00	-11.02	102000
46	1	14.50	-11.32	102000
47	1	15.00	-11.62	102000
48	1	15.50	-11.90	102000
49	1	16.00	-12.18	102000
50	1	16.50	-12.45	102000
51	1	17.00	-12.71	102000
52	1	17.50	-12.96	102000

53	1	18.00	-13.20	102000
54	1	18.50	-13.44	102000
55	1	19.00	-13.67	102000
56	1	19.50	-13.90	102000

Table 6. Parameters settings for multiple DSSS simulations.

Run Number	Spreading Code(s)	Noise Amplitude	E_B/N_0	Number of Packets
1	0	0.00	Inf	12040000
2	0	0.10	31.90	12040000
3	0	0.20	25.88	11280616
4	0	0.30	22.36	2040000
5	0	0.40	19.86	2040000
6	0	0.50	17.92	2040000
7	0	0.60	16.34	2040000
8	0	0.70	15.00	2040000
9	0	0.80	13.84	2040000
10	0	0.90	12.82	2040000
11	0	1.00	11.90	2040000
12	0	1.10	11.08	2040000
13	0	1.20	10.32	2040000
14	0	1.30	9.62	2040000
15	0	1.40	8.98	2040000
16	0	1.50	8.38	2040000
17	0	1.60	7.82	2040000
18	0	1.70	7.29	2040000
19	0	1.80	6.80	2040000
20	0	1.90	6.33	2040000
21	0	2.00	5.88	102000
22	0	2.50	3.94	102000
23	0	3.00	2.36	102000
24	0	3.50	1.02	102000
25	0	4.00	-0.14	102000
26	0	4.50	-1.16	102000
27	0	5.00	-2.08	102000
28	0	5.50	-2.90	102000
29	0	6.00	-3.66	102000
30	0	6.50	-4.35	102000
31	0	7.00	-5.00	102000
32	0	7.50	-5.60	102000
33	0	8.00	-6.16	102000

34	0	8.50	-6.69	102000
35	0	9.00	-7.18	102000
36	0	9.50	-7.65	102000
37	0	10.00	-8.10	102000
38	0	10.50	-8.52	102000
39	0	11.00	-8.92	102000
40	0	11.50	-9.31	102000
41	0	12.00	-9.68	102000
42	0	12.50	-10.03	102000
43	0	13.00	-10.38	102000
44	0	13.50	-10.70	102000
45	0	14.00	-11.02	102000
46	0	14.50	-11.32	102000
47	0	15.00	-11.62	102000
48	0	15.50	-11.90	102000
49	0	16.00	-12.18	102000
50	0	16.50	-12.45	102000
51	0	17.00	-12.71	102000
52	0	17.50	-12.96	102000
53	0	18.00	-13.20	102000
54	0	18.50	-13.44	102000
55	0	19.00	-13.67	102000
56	0	19.50	-13.90	102000
57	2, 3	0.00	Inf	12040000
58	2, 3	0.10	34.91	12040000
59	2, 3	0.20	28.89	9881040
60	2, 3	0.30	25.37	2040000
61	2, 3	0.40	22.87	2040000
62	2, 3	0.50	20.93	2040000
63	2, 3	0.60	19.35	2040000
64	2, 3	0.70	18.01	2040000
65	2, 3	0.80	16.85	2040000
66	2, 3	0.90	15.83	2040000
67	2, 3	1.00	14.91	2040000
68	2, 3	1.10	14.09	2040000
69	2, 3	1.20	13.33	2040000
70	2, 3	1.30	12.63	2040000
71	2, 3	1.40	11.99	2040000
72	2, 3	1.50	11.39	2040000
73	2, 3	1.60	10.83	2040000
74	2, 3	1.70	10.30	2040000
75	2, 3	1.80	9.81	2040000
76	2, 3	1.90	9.34	2040000

77	2, 3	2.00	8.89	102000
78	2, 3	2.50	6.95	102000
79	2, 3	3.00	5.37	102000
80	2, 3	3.50	4.03	102000
81	2, 3	4.00	2.87	102000
82	2, 3	4.50	1.85	102000
83	2, 3	5.00	0.93	102000
84	2, 3	5.50	0.11	102000
85	2, 3	6.00	-0.65	102000
86	2, 3	6.50	-1.34	102000
87	2, 3	7.00	-1.99	102000
88	2, 3	7.50	-2.59	102000
89	2, 3	8.00	-3.15	102000
90	2, 3	8.50	-3.67	102000
91	2, 3	9.00	-4.17	102000
92	2, 3	9.50	-4.64	102000
93	2, 3	10.00	-5.09	102000
94	2, 3	10.50	-5.51	102000
95	2, 3	11.00	-5.91	102000
96	2, 3	11.50	-6.30	102000
97	2, 3	12.00	-6.67	102000
98	2, 3	12.50	-7.02	102000
99	2, 3	13.00	-7.37	102000
100	2, 3	13.50	-7.69	102000
101	2, 3	14.00	-8.01	102000
102	2, 3	14.50	-8.31	102000
103	2, 3	15.00	-8.61	102000
104	2, 3	15.50	-8.89	102000
105	2, 3	16.00	-9.17	102000
106	2, 3	16.50	-9.44	102000
107	2, 3	17.00	-9.70	102000
108	2, 3	17.50	-9.95	102000
109	2, 3	18.00	-10.19	102000
110	2, 3	18.50	-10.43	102000
111	2, 3	19.00	-10.66	102000
112	2, 3	19.50	-10.89	102000
113	2, 3, 4	0.00	Inf	12040000
114	2, 3, 4	0.10	36.67	12040000
115	2, 3, 4	0.20	30.65	9540234
116	2, 3, 4	0.30	27.13	2040000
117	2, 3, 4	0.40	24.63	2040000
118	2, 3, 4	0.50	22.70	2040000
119	2, 3, 4	0.60	21.11	2040000

120	2, 3, 4	0.70	19.77	2040000
121	2, 3, 4	0.80	18.61	2040000
122	2, 3, 4	0.90	17.59	2040000
123	2, 3, 4	1.00	16.67	2040000
124	2, 3, 4	1.10	15.85	2040000
125	2, 3, 4	1.20	15.09	2040000
126	2, 3, 4	1.30	14.40	2040000
127	2, 3, 4	1.40	13.75	2040000
128	2, 3, 4	1.50	13.15	2040000
129	2, 3, 4	1.60	12.59	2040000
130	2, 3, 4	1.70	12.07	2040000
131	2, 3, 4	1.80	11.57	2040000
132	2, 3, 4	1.90	11.10	2040000
133	2, 3, 4	2.00	10.65	102000
134	2, 3, 4	2.50	8.72	102000
135	2, 3, 4	3.00	7.13	102000
136	2, 3, 4	3.50	5.79	102000
137	2, 3, 4	4.00	4.63	102000
138	2, 3, 4	4.50	3.61	102000
139	2, 3, 4	5.00	2.70	102000
140	2, 3, 4	5.50	1.87	102000
141	2, 3, 4	6.00	1.11	102000
142	2, 3, 4	6.50	0.42	102000
143	2, 3, 4	7.00	-0.23	102000
144	2, 3, 4	7.50	-0.83	102000
145	2, 3, 4	8.00	-1.39	102000
146	2, 3, 4	8.50	-1.91	102000
147	2, 3, 4	9.00	-2.41	102000
148	2, 3, 4	9.50	-2.88	102000
149	2, 3, 4	10.00	-3.33	102000
150	2, 3, 4	10.50	-3.75	102000
151	2, 3, 4	11.00	-4.15	102000
152	2, 3, 4	11.50	-4.54	102000
153	2, 3, 4	12.00	-4.91	102000
154	2, 3, 4	12.50	-5.26	102000
155	2, 3, 4	13.00	-5.60	102000
156	2, 3, 4	13.50	-5.93	102000
157	2, 3, 4	14.00	-6.25	102000
158	2, 3, 4	14.50	-6.55	102000
159	2, 3, 4	15.00	-6.85	102000
160	2, 3, 4	15.50	-7.13	102000
161	2, 3, 4	16.00	-7.41	102000
162	2, 3, 4	16.50	-7.68	102000

163	2, 3, 4	17.00	-7.93	102000
164	2, 3, 4	17.50	-8.19	102000
165	2, 3, 4	18.00	-8.43	102000
166	2, 3, 4	18.50	-8.67	102000
167	2, 3, 4	19.00	-8.90	102000
168	2, 3, 4	19.50	-9.13	102000
169	2, 3, 4, 5	0.00	Inf	12040000
170	2, 3, 4, 5	0.10	37.92	12040000
171	2, 3, 4, 5	0.20	31.90	12040000
172	2, 3, 4, 5	0.30	28.38	9547177
173	2, 3, 4, 5	0.40	25.88	2040000
174	2, 3, 4, 5	0.50	23.94	2040000
175	2, 3, 4, 5	0.60	22.36	2040000
176	2, 3, 4, 5	0.70	21.02	2040000
177	2, 3, 4, 5	0.80	19.86	2040000
178	2, 3, 4, 5	0.90	18.84	2040000
179	2, 3, 4, 5	1.00	17.92	2040000
180	2, 3, 4, 5	1.10	17.10	2040000
181	2, 3, 4, 5	1.20	16.34	2040000
182	2, 3, 4, 5	1.30	15.65	2040000
183	2, 3, 4, 5	1.40	15.00	2040000
184	2, 3, 4, 5	1.50	14.40	2040000
185	2, 3, 4, 5	1.60	13.84	2040000
186	2, 3, 4, 5	1.70	13.31	2040000
187	2, 3, 4, 5	1.80	12.82	2040000
188	2, 3, 4, 5	1.90	12.35	2040000
189	2, 3, 4, 5	2.00	11.90	102000
190	2, 3, 4, 5	2.50	9.97	102000
191	2, 3, 4, 5	3.00	8.38	102000
192	2, 3, 4, 5	3.50	7.04	102000
193	2, 3, 4, 5	4.00	5.88	102000
194	2, 3, 4, 5	4.50	4.86	102000
195	2, 3, 4, 5	5.00	3.94	102000
196	2, 3, 4, 5	5.50	3.12	102000
197	2, 3, 4, 5	6.00	2.36	102000
198	2, 3, 4, 5	6.50	1.67	102000
199	2, 3, 4, 5	7.00	1.02	102000
200	2, 3, 4, 5	7.50	0.42	102000
201	2, 3, 4, 5	8.00	-0.14	102000
202	2, 3, 4, 5	8.50	-0.66	102000
203	2, 3, 4, 5	9.00	-1.16	102000
204	2, 3, 4, 5	9.50	-1.63	102000
205	2, 3, 4, 5	10.00	-2.08	102000

206	2, 3, 4, 5	10.50	-2.50	102000
207	2, 3, 4, 5	11.00	-2.90	102000
208	2, 3, 4, 5	11.50	-3.29	102000
209	2, 3, 4, 5	12.00	-3.66	102000
210	2, 3, 4, 5	12.50	-4.01	102000
211	2, 3, 4, 5	13.00	-4.35	102000
212	2, 3, 4, 5	13.50	-4.68	102000
213	2, 3, 4, 5	14.00	-5.00	102000
214	2, 3, 4, 5	14.50	-5.30	102000
215	2, 3, 4, 5	15.00	-5.60	102000
216	2, 3, 4, 5	15.50	-5.88	102000
217	2, 3, 4, 5	16.00	-6.16	102000
218	2, 3, 4, 5	16.50	-6.43	102000
219	2, 3, 4, 5	17.00	-6.69	102000
220	2, 3, 4, 5	17.50	-6.94	102000
221	2, 3, 4, 5	18.00	-7.18	102000
222	2, 3, 4, 5	18.50	-7.42	102000
223	2, 3, 4, 5	19.00	-7.65	102000
224	2, 3, 4, 5	19.50	-7.88	102000
225	2, 3, 4, 5, 6	0.00	Inf	14000000
226	2, 3, 4, 5, 6	0.10	38.89	14000000
227	2, 3, 4, 5, 6	0.20	32.87	14000000
228	2, 3, 4, 5, 6	0.30	29.35	14000000
229	2, 3, 4, 5, 6	0.40	26.85	14000000
230	2, 3, 4, 5, 6	0.50	24.91	14000000
231	2, 3, 4, 5, 6	0.60	23.33	14000000
232	2, 3, 4, 5, 6	0.70	21.99	14000000
233	2, 3, 4, 5, 6	0.80	20.83	14000000
234	2, 3, 4, 5, 6	0.90	19.81	14000000
235	2, 3, 4, 5, 6	1.00	18.89	14000000
236	2, 3, 4, 5, 6	1.10	18.07	14000000
237	2, 3, 4, 5, 6	1.20	17.31	7856071
238	2, 3, 4, 5, 6	1.30	16.61	4000000
239	2, 3, 4, 5, 6	1.40	15.97	4000000
240	2, 3, 4, 5, 6	1.50	15.37	4000000
241	2, 3, 4, 5, 6	1.60	14.81	4000000
242	2, 3, 4, 5, 6	1.70	14.28	4000000
243	2, 3, 4, 5, 6	1.80	13.79	4000000
244	2, 3, 4, 5, 6	1.90	13.32	4000000
245	2, 3, 4, 5, 6	2.00	12.87	200000
246	2, 3, 4, 5, 6	2.50	10.93	200000
247	2, 3, 4, 5, 6	3.00	9.35	200000
248	2, 3, 4, 5, 6	3.50	8.01	200000

249	2, 3, 4, 5, 6	4.50	5.83	200000
250	2, 3, 4, 5, 6	5.00	4.91	200000
251	2, 3, 4, 5, 6	5.50	4.09	200000
252	2, 3, 4, 5, 6	6.00	3.33	200000
253	2, 3, 4, 5, 6	6.50	2.63	200000
254	2, 3, 4, 5, 6	7.00	1.99	200000
255	2, 3, 4, 5, 6	7.50	1.39	200000
256	2, 3, 4, 5, 6	8.00	0.83	200000
257	2, 3, 4, 5, 6	8.50	0.30	200000
258	2, 3, 4, 5, 6	9.00	-0.19	200000
259	2, 3, 4, 5, 6	9.50	-0.66	200000
260	2, 3, 4, 5, 6	10.00	-1.11	200000
261	2, 3, 4, 5, 6	10.50	-1.53	200000
262	2, 3, 4, 5, 6	11.00	-1.93	200000
263	2, 3, 4, 5, 6	11.50	-2.32	200000
264	2, 3, 4, 5, 6	12.00	-2.69	200000
265	2, 3, 4, 5, 6	12.50	-3.05	200000
266	2, 3, 4, 5, 6	13.00	-3.39	200000
267	2, 3, 4, 5, 6	13.50	-3.71	200000
268	2, 3, 4, 5, 6	14.00	-4.03	200000
269	2, 3, 4, 5, 6	14.50	-4.33	200000
270	2, 3, 4, 5, 6	15.00	-4.63	200000
271	2, 3, 4, 5, 6	15.50	-4.91	200000
272	2, 3, 4, 5, 6	16.00	-5.19	200000
273	2, 3, 4, 5, 6	16.50	-5.46	200000
274	2, 3, 4, 5, 6	17.00	-5.72	200000
275	2, 3, 4, 5, 6	17.50	-5.97	200000
276	2, 3, 4, 5, 6	18.00	-6.21	200000
277	2, 3, 4, 5, 6	18.50	-6.45	200000
278	2, 3, 4, 5, 6	19.00	-6.68	200000
279	2, 3, 4, 5, 6	19.50	-6.91	200000

B. Inverse Beamforming $N_L = 31$ Figures

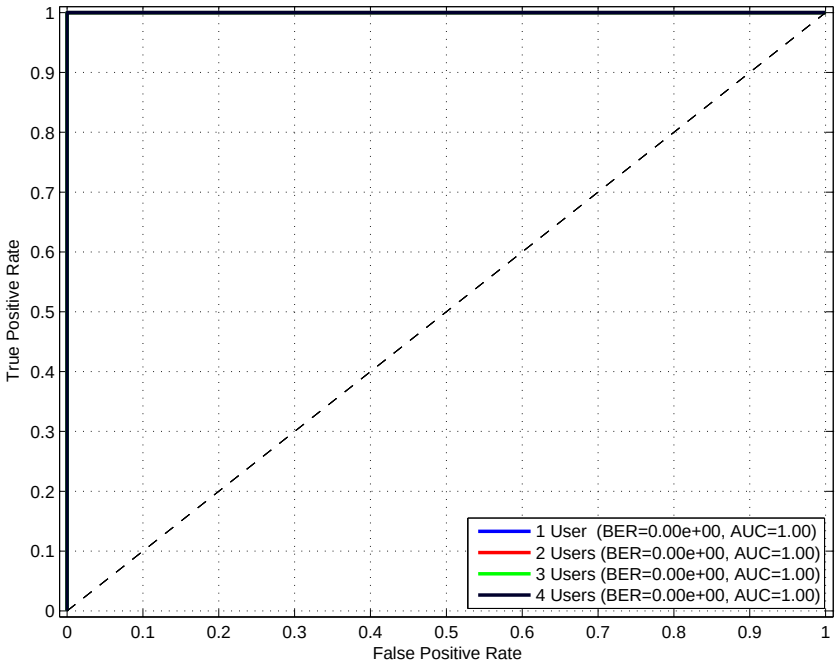

Figure 21. Detection ROC for $E_B/N_0 = Inf$

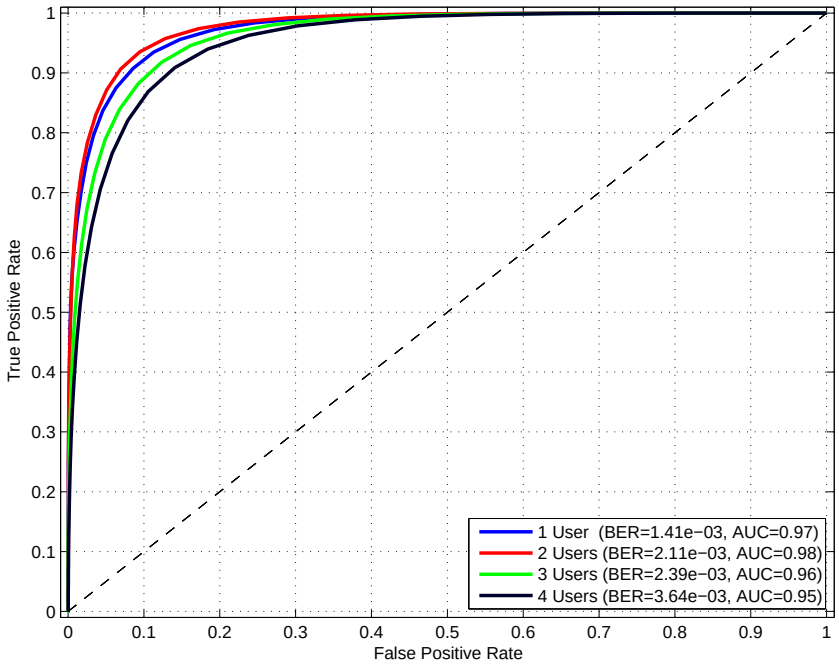

Figure 22. Detection ROC for E_B/N_0= 8.38

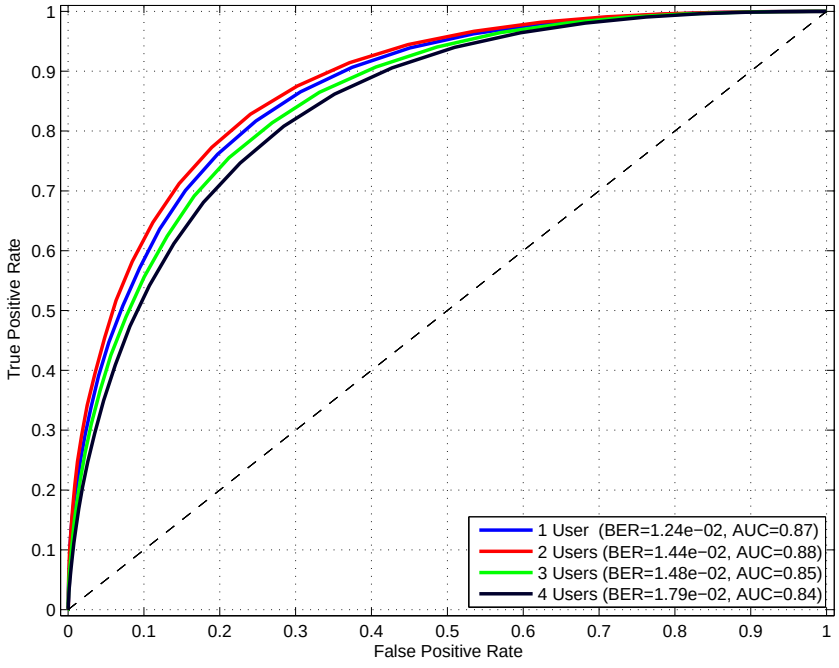

Figure 23. Detection ROC for E_B/N_0= 5.88

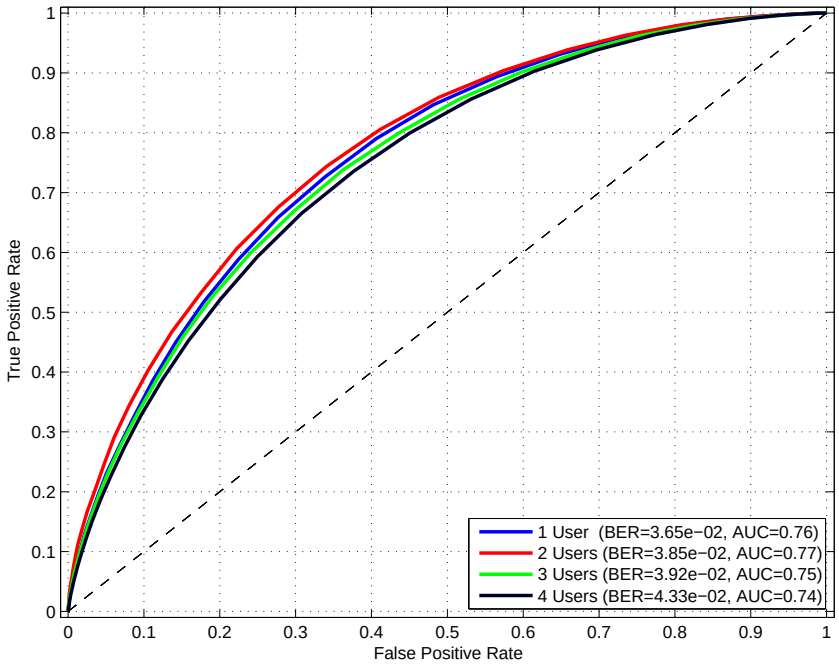

Figure 24. Detection ROC for $E_B/N_0 = 3.94$

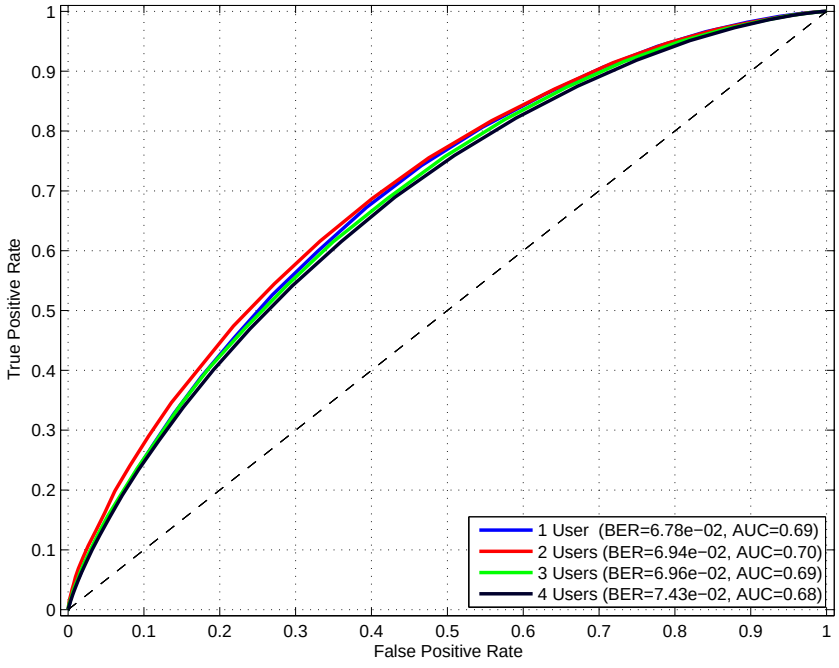

Figure 25. Detection ROC for $E_B/N_0 = 2.36$

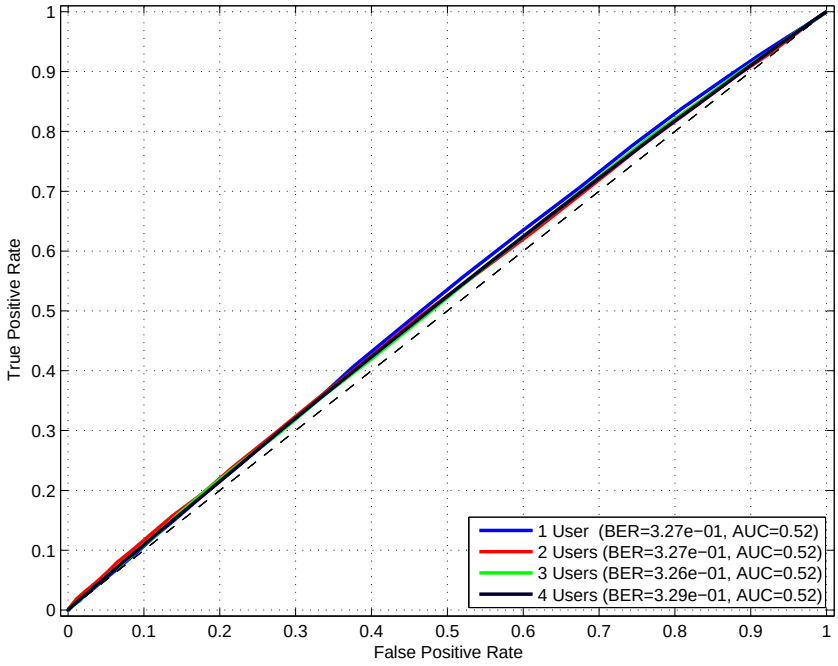

Figure 26. Detection ROC for $E_B/N_0 = -8.10$

C. Inverse Beamforming $N_L = 255$ Figures

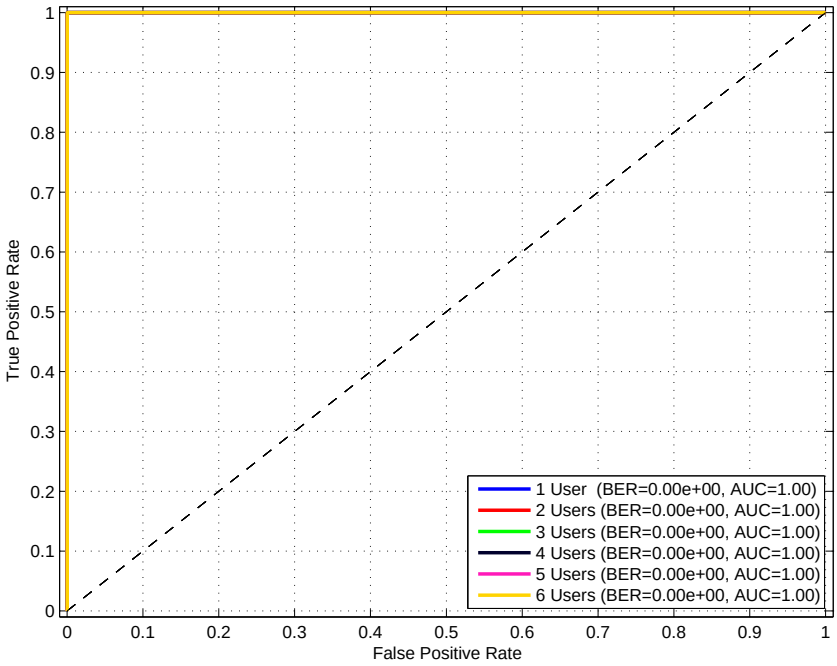

Figure 27. Detection ROC for E_B/N_0= 21.06

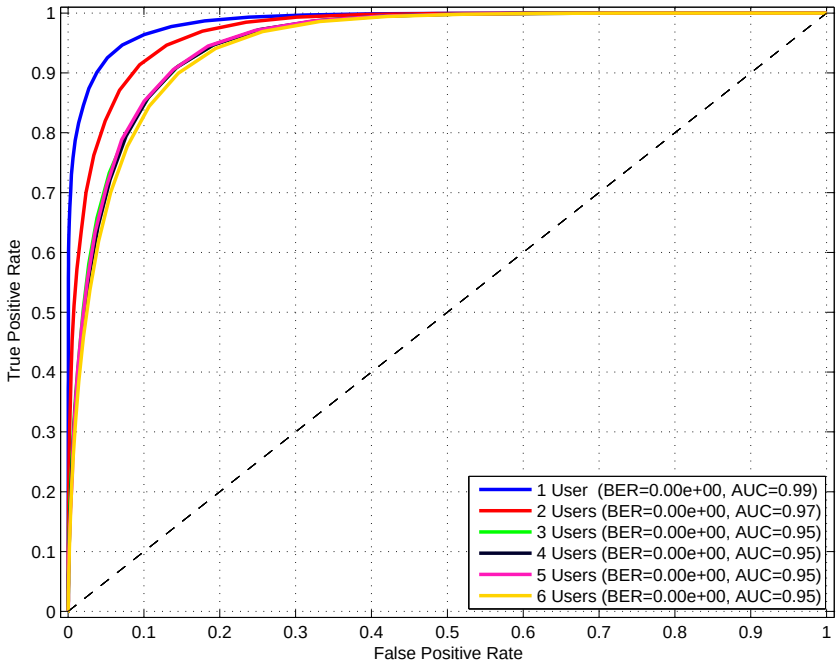

Figure 28. Detection ROC for $E_B/N_0 = 15.03$

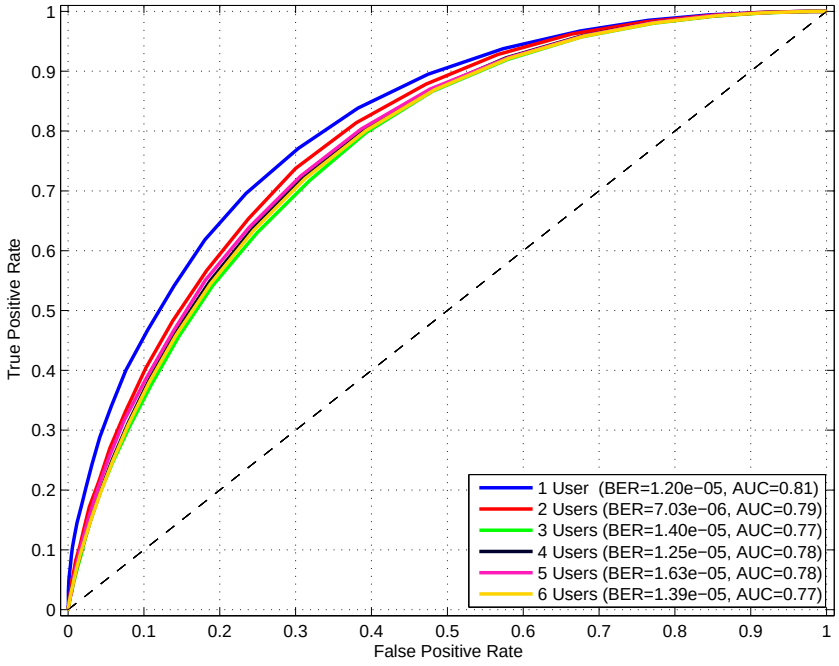

Figure 29. Detection ROC for $E_B/N_0 = 11.51$

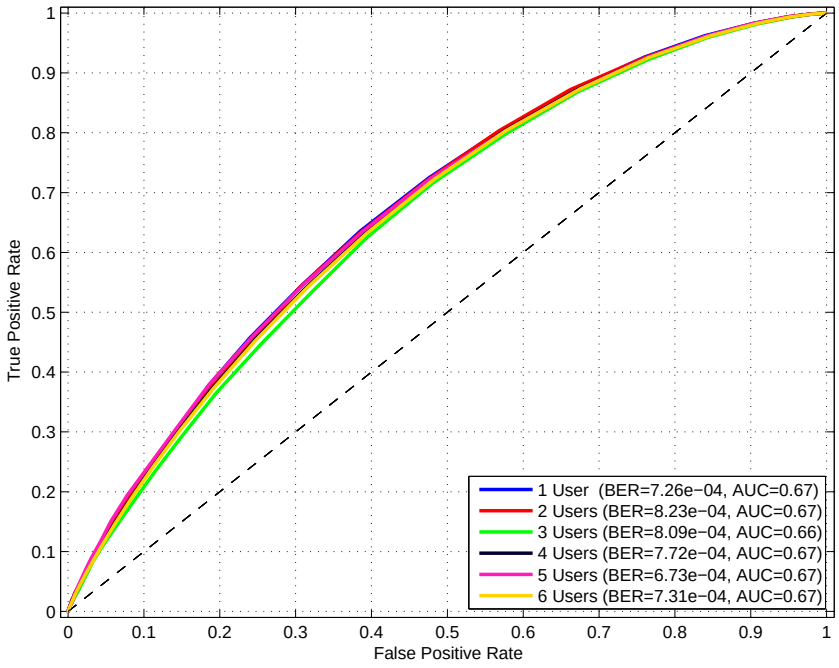

Figure 30. Detection ROC for E_B/N_0= 9.01

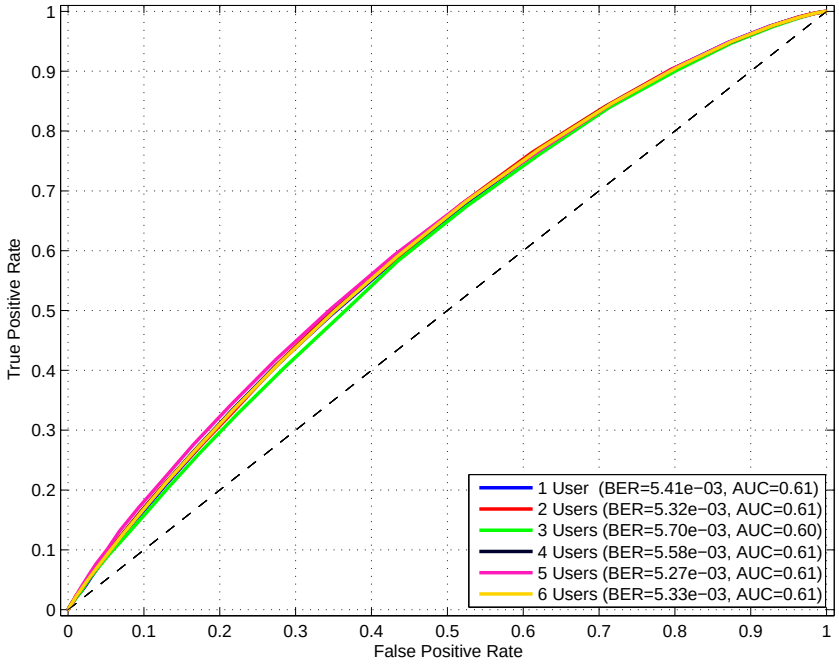

Figure 31. Detection ROC for E_B/N_0= 7.08

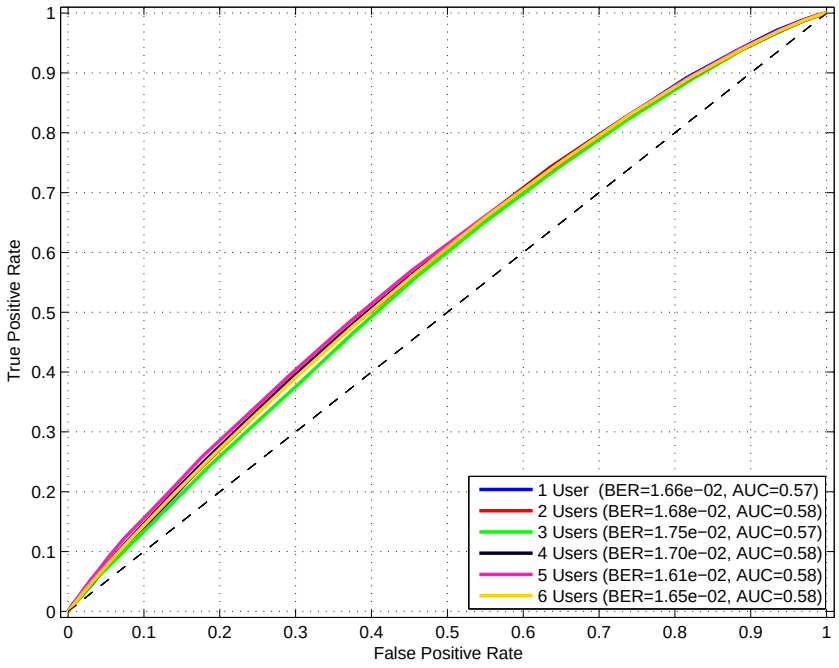

Figure 32. Detection ROC for E_B/N_0= 5.49

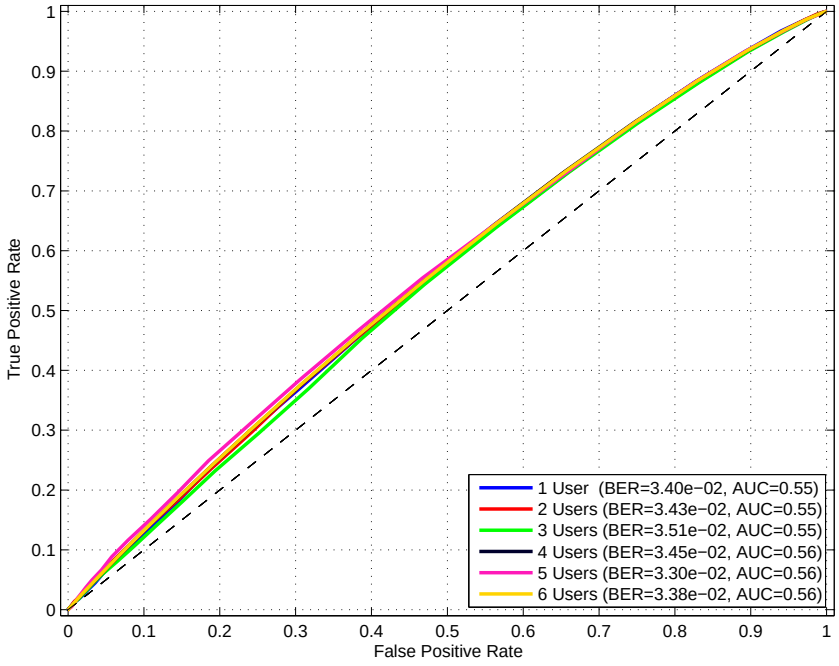

Figure 33. Detection ROC for E_B/N_0= 4.15

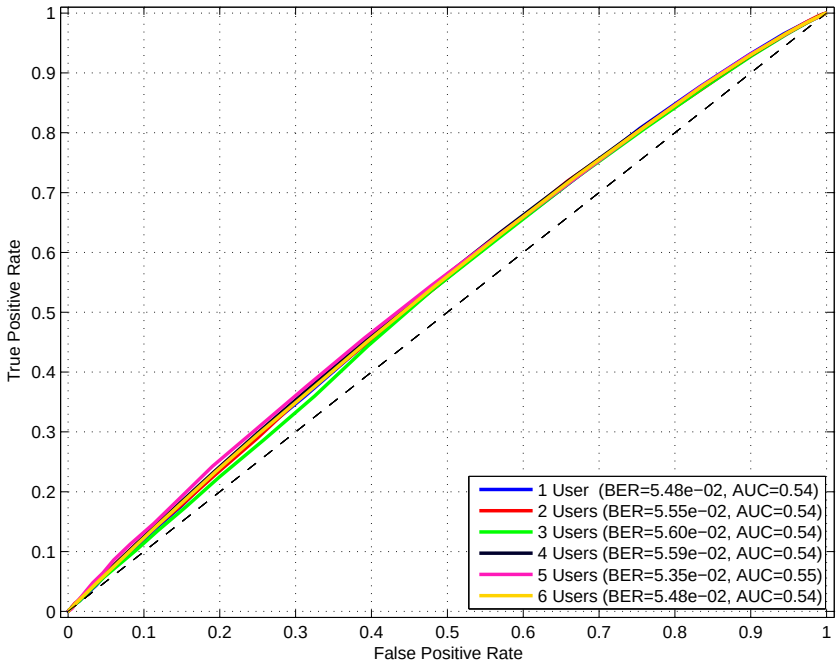

Figure 34. Detection ROC for $E_B/N_0 = 2.99$

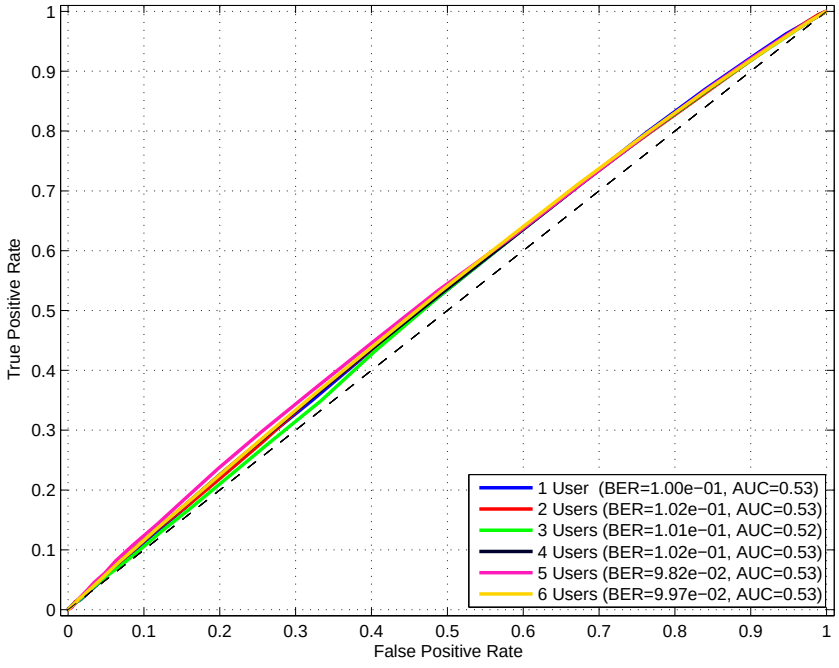

Figure 35. Detection ROC for $E_B/N_0 = 1.06$

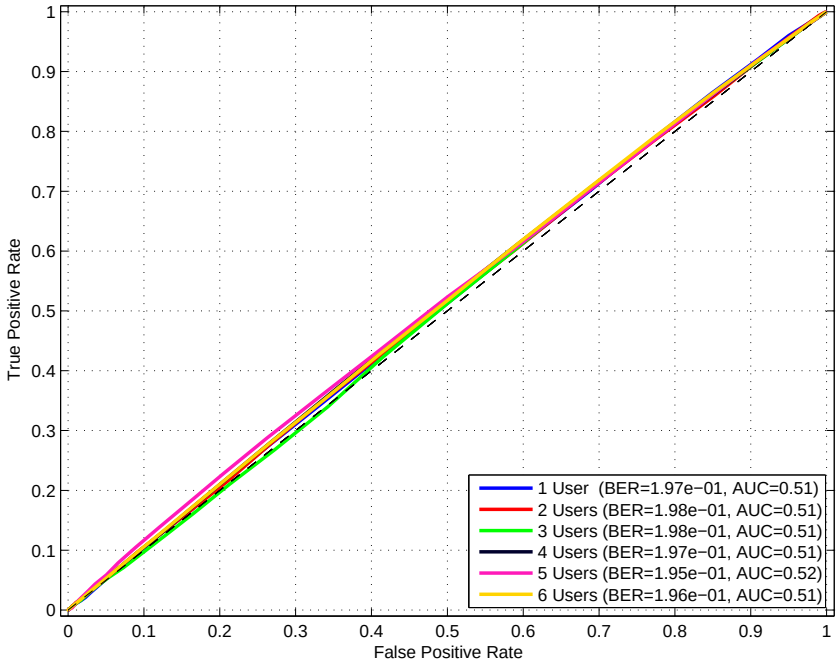

Figure 36. Detection ROC for $E_B/N_0 = -2.47$

D. Lessons Learned

This section describes the issues that were encountered during this thesis.

4.0.0.0.1 Module Not Found. If you are using GNU Radio components such as the FFT or filters, you must go into the CMakeLists.txt file in the top module directory and add the component used in all caps ie (BLOCKS or FFT) to ensure that the Python-C^{++} interface works. Find the lines that look like below, and ensure to add the components as in bold:

```
########################################
# Find gnuradio build dependencies
########################################
find_package(CppUnit)
find_package(Doxygen)

# Search for GNU Radio and its components and versions. Add any
# components required to the list of GR_REQUIRED_COMPONENTS (in all
# caps such as FILTER or FFT) and change the version to the minimum
# API compatible version required.
set(GR_REQUIRED_COMPONENTS RUNTIME BLOCKS FFT)
find_package(Gnuradio "3.7.2" REQUIRED)

if(NOT CPPUNIT_FOUND)
message(FATAL_ERROR "CppUnit required to compile staticIB")
endif()
```

[40] has more information on this issue.

4.0.0.0.2 Module Error on SDR. In my experience when this error message occurs in the GNU Radio Companion console window when you try to run a flow graph, but the module has been installed, it is an issue with the `PYTHONPATH`. To fix, just make sure to add `/usr/local/lib/python/site-packages/` to the `PYTHONPATH` environment variable.

4.0.0.0.3 Attribute Error on SDR.

If instead of the module error in the previous situation, you get an attribute error that specifically says `AttributeError: 'module' has no object '<block you are trying to use from the module>'`, then it is likely an issue with the `LD_LIBRARY_PATH`. To fix, just add `/usr/local/lib` to the path.

4.0.0.0.4 .bashrc/.profile.

On Ubuntu systems, the `.bashrc` is the bash script that is called on creation of a new shell instance. The Ettus SDRs run a version of OpenEmbedded Linux, this script is the `.profile` script.

4.0.0.0.5 Network Mode on Ettus E310.

There are a few things that you need to be sure of before trying this. First and most importantly, the UHD versions on the host computer and the SDR must match exactly. Next, the UHD build on the host computer must have been made with the `DENABLE_E300=ON` flag when running `cmake`.

If both of those conditions have been met, and you are able to ping the device, `ssh` into the SDR and execute `usrp_e3x0_network_mode`. The radio will initialize itself, and look like it hangs after loading FPGA images. This is normal. Once it stops, it is ready to be used in network mode.

Note: this mode is being deprecated in future releases of the FPGA firmware.

4.0.0.0.6 Sampling Rates.

You must ensure that the sampling rates are equivalent at both the transmitter and receiver. Sometimes when choosing arbitrary sampling rates, the physical hardware does not support the option picked. In this case, you must modify the actual rate until both radios are able to support the values picked. In the case of an unsupported sample rate, the radio will pick the closest

sample rate that is supported and display a message saying it did so in the GNU Radio Companion console window.

4.0.0.0.7 Passing Strings to C++ Blocks. When trying to pass a string to a C^{++} block, the module will compile without errors, but upon trying to use the block in a flow graph, it stops with this error: `error: TypeError: Arugment 2 of type std::string &`. To fix, add the line: `%include ''std_string.i''` to the .i file in the swig directory. While not tested, I am assuming that adding `%include ''std_vector.i''` will take care of issues with vectors.

Bibliography

1. S. Draganov, J. Weinfield, and L. Haas, "Inverse Beamforming for navigating in multipath environments," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 5316–5319, 2008.

2. B. V. Veen and Kevin M. Buckley, "Beamforming - A Versatile Approach to Spatial Filtering," *IEE*, apr 1988.

3. "BROAD AGENCY ANNOUNCEMENT (BAA) FY 17 Communications and Networking Discovery and Invention," Office of Naval Research, Arlington, Tech. Rep., 2016.

4. T. A. Halverson, "DoD Instruction 4630.09, July 15, 2015," 2015.

5. K. Q. Zhang, *Wireless Communications: Principles, Theory and Methodology*, 1st ed. John Wiley & Sons, Inc., 2015.

6. D. Sharma, A. Mishra, and R. Saxena, "Analog & Digital Modulation Techniques: An Overview," *International Journal of Computing Science and Communication Technologies*, vol. 3, no. 1, 2010. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:ANALOG+&+DIGITAL+MODULATION+TECHNIQUES+:+AN+OVERVIEW#1

7. B. Sklar, *Digital Communications: Fundamentals and Applications*, 2nd ed. Prentice Hall, 2004.

8. R. C. Hansen, *Phased Array Antennas*. John Wiley & Sons, Inc., 1998.

9. S. E. Schwarz, *Electromagnetics for Engineers*. Saunders College Publishing, 1990.

10. L. W. Couch, *Digital and Analog Communications Systems*, 8th ed. Pearson, 2001.

11. C. Qiao, C. Zhou, W. Dai, and Y. Wang, "Improved anti-jamming scheme for direct-sequence spread-spectrum receivers," *Electronics Letters*, vol. 52, no. 2, pp. 161–163, 2016. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/el.2015.3304

12. R. L. Peterson, R. E. Ziemer, and D. E. Borth, *Introduction to Spread Spectrum Communications*. Prentice Hall, 1995.

13. M. Malek, "Coding Theory Hadamard Codes," 2008. [Online]. Available: http://www.mcs.csueastbay.edu/{~}malek/TeX/Hadamard.pdf

14. "Navstar GPS space Segment/Navigation User Interface," *Navstar GPS space Segment*, p. 213, 2013. [Online]. Available: http://www.gps.gov/technical/icwg/IS-GPS-200H.pdf

15. "Interface Specification IS-GPS-800," p. 127, 2013. [Online]. Available: http://www.gps.gov/technical/icwg/IS-GPS-800D.pdf

16. G. E. Prescott, "Performance Metrics for Low Probability of Intercept Communication Systems," Air Force Office of Scientific Research, Washington D.C., Tech. Rep., 1993.

17. J. D. Vlok, "Detection of Direct Sequence Spread Spectrum Signals," *M.Eng (Electronic)*, 2003.

18. J. S. Lee and L. E. Miller, *CDMA Systems Engineering Handbook*, 1998.

19. P. Zavala, W. De Roeck, K. Janssens, J. Arruda, P. Sas, and W. Desmet, "Generalized inverse beamforming with optimized regularization strategy," *Mechanical Systems and Signal Processing*, vol. 25, no. 3, pp. 928–939, 2011.

20. "What is GNU Radio and why do I want it?" [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/whatisGR

21. Martin Braun, "Core Concepts of GNU Radio," 2012. [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/TutorialsCoreConcepts

22. "UHD," 2016. [Online]. Available: https://kb.ettus.com/UHD

23. "BlocksCodingGuide - GNU Radio - gnuradio.org." [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/BlocksCodingGuide

24. "Tutorials Write Python Applications." [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/TutorialsWritePythonApplications

25. T. Rondeau, "Scheduler Details," Tech. Rep., 2013. [Online]. Available: https://static.squarespace.com/static/543ae9afe4b0c3b808d72acd/543aee1fe4b09162d0863397/543aee20e4b09162d0863578/1380223973117/gr_scheduler_overview.pdf

26. "FAQ - GNU Radio." [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/FAQ

27. Martin Braun, "Out-of-tree modules." [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules

28. "GNU Radio Manual and C++ API Reference: Stream Tags," 2016. [Online]. Available: http://gnuradio.org/doc/doxygen/page_stream_tags.html

29. "GNU Radio Manual and C++ API Reference: Components," 2016. [Online]. Available: http://gnuradio.org/doc/doxygen/page_components.html

30. "GNU Radio Companion." [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion

31. Ravi Sharan and Nathan West, "The Comprehensive GNU Radio Archive Network," 2015. [Online]. Available: http://cgran.org/

32. "GDB: The GNU Project Debugger," 2016. [Online]. Available: http://www.gnu.org/software/gdb/

33. J. C. Bic, D. Duponteil, and J. Imbeaux, *Elements of Digital Communication*. John Wiley & Sons, Inc., 1991.

34. "GNU Radio Manual and C++ API Reference: gr::blocks::add_cc Class Reference," 2016. [Online]. Available: http://gnuradio.org/doc/doxygen/classgr_1_1blocks_1_1add__cc.html

35. "GNU Radio Manual and C++ API Reference: gr::blocks::unpack_k_bits_bb Class Reference," 2016. [Online]. Available: http://gnuradio.org/doc/doxygen/classgr_1_1blocks_1_1unpack__k__bits__bb.html

36. "GNU Radio Manual and C++ API Reference: gr::analog::noise_source_f Class Reference," 2016. [Online]. Available: http://gnuradio.org/doc/doxygen/classgr_1_1analog_1_1noise__source__f.html

37. "GNU Radio Manual and C++ API Reference: gr::blocks::pack_k_bits_bb Class Reference," 2016. [Online]. Available: http://gnuradio.org/doc/doxygen/classgr_1_1blocks_1_1pack__k__bits__bb.html

38. "USRP E310 Embedded Software Defined Radio (SDR) - Ettus Research," 2017. [Online]. Available: https://www.ettus.com/product/details/E310-KIT

39. "USRP E312 Battery Operated Embedded Software Defined Radio (SDR) - Ettus Research," 2017. [Online]. Available: https://www.ettus.com/product/details/USRP-E312

40. Manos, "python - gnuradio 'ImportError undefined symbol' - Stack Overflow," 2015. [Online]. Available: http://stackoverflow.com/questions/28859517/gnuradio-importerror-undefined-symbol

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23–3–2016	Master's Thesis	Sept 2015 — Mar 2017

4. TITLE AND SUBTITLE

Low Probability of Detection Communication using Inverse Beamforming in GNU Radio using Code Division Multiple Access

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Rennich, Travis, B, 2 LT, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-17-M-064

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL (Attn: Dr. Vasu Chakravarthy, AFRL/RYWE)
2241 Avionics Circle
WPAFB OH 45433
COMM 937-785-5579 ext. 4245
Email: vasu.chakravarthy@wp.afb.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

Air Force Research Laboraroty (AFRL/RYWE)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

The primary goal of this thesis is to design a communications system that is more covert than existing systems while maintaining a constant Bit Error Rate (BER). Inverse Beamforming is the use of multiple spreading codes by a transmitter, each with a fraction of the power that would have been used by a single spreading code, to transmit the same message over multiple different antennas. At the receiver, there is a single antenna that splits the signal multiple ways. Each channel locks to a different spreading code, and is used to reconstruct the original message. In Chip-Offset Inverse Beamforming, successive users have an increasing number of chip offsets added in between each replication of the spreading code for each bit of the data message. This is done to decrease autocorrelation spikes to lower detection performance. With the inclusion of chip offsets, both the BER and the intercept receiver detection performance decreased in some situations.

15. SUBJECT TERMS

Inverse Beamforming,GNU Radio

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. R. Martin, AFIT/ENG
U	U	U	U	102	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, x4625; richard.martin@afit.edu