Implementation of 5G Technologies with Heterogeneous Cooperative Spectrum Sensing and LTE-R

by

Kuldeep S. Gill

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Electrical and Computer Engineering
by

December 2017

APPROVED:

Professor Alexander Wyglinski, Major Advisor

Professor Kaveh Pahlavan

Dr. Travis Collins

Abstract

There have been an unprecedented increase in demand for high data rates and mobility required by new wireless applications which has led to intensive research on fifth generation (5G) wireless communication system. By 2020, 5G is projected to be employed worldwide supporting the connectivity of up to 20 billion devices and will be crucial in the success of vehicular networking and internet of things (IoT). It is also believed that 5G systems would be capable of providing significant improvements in cell capacity and will support high data rates up to 5 and 50 Gb/s for high-mobility and pedestrian users, respectively. However, to achieve such data rates for high-mobility scenarios there are still many challenges for wireless system engineers.

In this thesis, we propose two test-beds namely Heterogeneous Cooperative Spectrum Sensing (CSS) and Long Term Evolution for Railways (LTE-R) performance analysis test-bed to further advance the 5G system development. Heterogeneous CSS test-bed is implemented using Software-Defined Radios (SDRs) with different radio characteristics. We used both soft and hard data fusion schemes to compare the signal source detection performance in real-time fading scenario. For 5G technologies, the most effective solution is to use the underutilized spectrum as a secondary user via dynamic spectrum access (DSA). It is very challenging to get an accurate estimate of incumbent users with a single-sensor system under a practical fading environment. Various non-idealities such as shadowing, multipath and fluctuating noise variance can make it difficult to detect the primary user. Cooperative spectrum sensing can mitigate the effects of multipath and shadowing by utilizing the spatial and temporal diversity of a multiple radio network. LTE-R test-bed analyze the performance of high speed trains (HSTs) in a tunnel environment and can be used to test 5G systems for high mobility scenario in drastically impaired channels.

Acknowledgements

I would like to express my deepest gratitude to my advisor Professor Alexander Wyglinski for his continuous guidance and support towards my course of degree. I am very thankful for the opportunity to work with him in Wireless Innovation Laboratory at Worcester Polytechnic Institute.

I want to thank Professor Kaveh Pahlavan and Dr. Travis Collins for serving on my committee and providing valuable suggestions and comments with regards to my thesis.

I would also like to thank my WILab team members Dr. Srikanth Pagadarai and Dr. Paulo Ferreira for their immense support during my graduate studies. Finally, I'm also thankful to my parents, without their constant support I wouldn't be here.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

There has been an exponential growth in the Internet landscape. The devices like smartphones, computers, laptops, tablets will continue to increase along with multitude of other devices that are connected to Internet. This has facilitated the research in dynamic spectrum access (DSA) [1, 2] for efficient utilization of spectrum resources to sustain billions of Internet of Things (IoT) devices. There has been a significant increase in the study of cognitive radios for efficiently utilizing the electromagnetic spectrum. It has been observed that the spectrum occupancy is not uniform across all frequency bands, resulting in numerous spectral white spaces [3]. To opportunistically access the idle channel, spectrum sensing is considered to be a significant technology enabling DSA. Although several spectrum sensing techniques have been proposed in the open literature, energy detection is widely used due to its low implementation complexity [4]. We discuss some of the spectrum sensing techniques along with energy detection below.

- In energy detection (ED) scheme the energy of the signal is detected in the frequency location and based on the threshold value we decide whether the signal is present or absent.

- Cyclostationary Feature Detection is a complex scheme to implement compared to ED and it is mostly used when we need to also classify the signal present based on

their modulation scheme.

- When secondary user has a priori knowledge of primary user signal, matched filter (MF) detection is applied. Detection by using matched filter needs less detection time compared to ED but primary user information is required.

These spectrum sensing techniques can be used in a non-cooperative manner but it is very challenging to get an accurate estimate using a single-sensor system under a practical fading environment. Various non-idealities such as shadowing, multipath and fluctuating noise variance can make it difficult to detect the primary user [5, 6]. Cooperative spectrum sensing can mitigate the effects of multipath and shadowing by utilizing the spatial and temporal diversity of a multiple radio network [7, 8]. In cooperative spectrum sensing, each sensor node collects the spectral data and transmits it to a fusion center (FC) for decision making. Figure 1.1 shows how a heterogeneous sensor network exploits the spatial diversity.

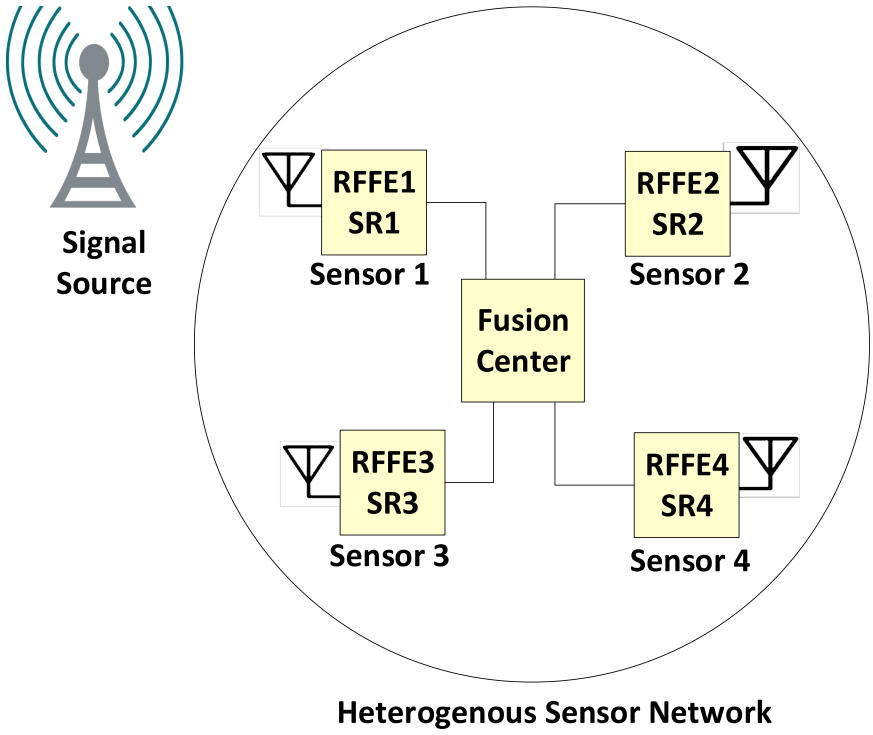

Figure 1.1: Heterogeneous sensor network employing cooperative spectrum sensing. $RFFE_i$ and SR_i represents different front end and sampling rates for the SDR units.

We have also analyze the performance of Long Term Evolution for Railways (LTE-R) in a

tunnel environment. In recent years, the use of trains have witnessed tremendous growth due to their increasing speeds, which has led to the demand for reliable wireless communication systems with these transportation systems. The development of a reliable wireless network for high speed trains is not a simple task and it is still an emerging technology. Global System for Mobile Communication (GSM-R) [9], was a wireless communications standard designed for high speed trains, but it turned out not to be reliable enough and possess several limitations. Subsequently, LTE [10] proposed a promising solution for achieving broadband data rates in high speed trains that can overcome various GSM-R limitations [11, 12].

LTE-R is a high speed communication standard based on the existing LTE system architecture [12]. There has been several studies regarding the assessment of LTE-R as a viable choice for next generation high speed communications for railway applications [13,14]. Most LTE systems operate at 1.8 GHz – 2.6 GHz bands, which possesses a high propagation loss and severe fading effects. Highly mobile trains inside tunnel environment makes the design of reliable communication links very challenging. To achieve reliable radio coverage inside tunnels, leaky feeder cables have been proposed [15]. With LCX, more uniform coverage can be achieved and installation is also comparatively simple. Each slot in the cable is equivalent to an antenna, which can transmit and receive signals. Figure 1.1 shows the LOS propagation environment inside a tunnel for a high speed train with velocity v.

1.2 State of the Art

The cooperative spectrum sensing testbed using normalized energy detection has been implemented and and has been compared for both soft and hard data fusion scheme. Both soft data fusion and hard data fusion have been extensively studied [16–18], with several algorithms being implemented for each scheme. In a hard decision approach, each local decision statistic from sensor node is transmitted to an FC via overhead channels. The FC merges the sensing data and makes a global decision based on various algorithms such as majority rule, OR rule and AND rule [19]. For a soft decision scheme, each SU sends its local sensing data to the FC, which makes decision based on a global test statistic G. Soft decision combining improves the cooperative gain but it also possesses several limitations.

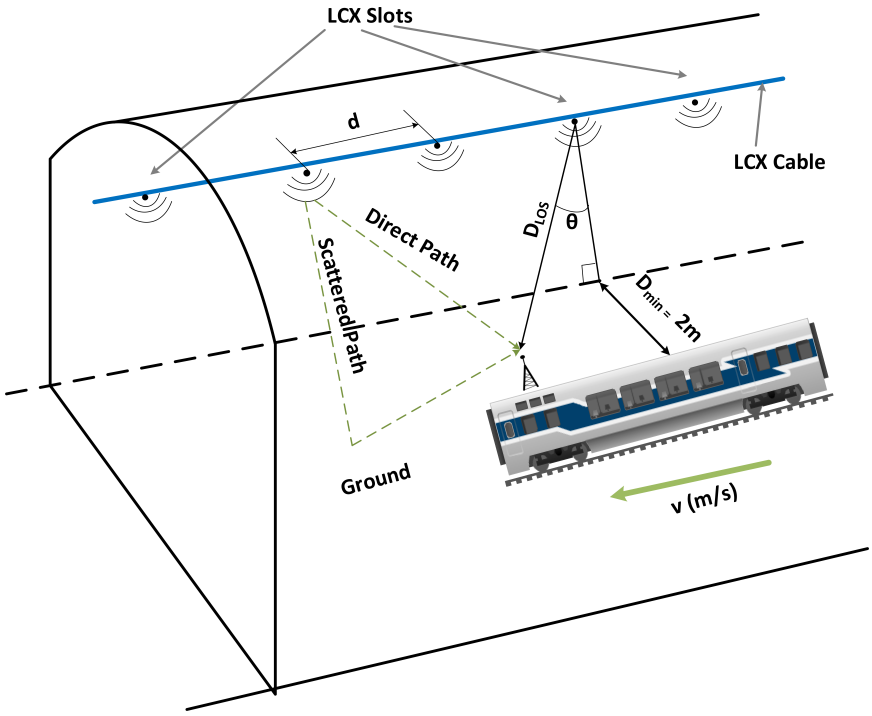

Figure 1.2: High speed train inside a tunnel for LTE-R. D_{LOS} is the distance between transmitter and receiver, d is the distance between the LCX cable transmission slots.

With an infinite bandwidth, the real floating values can be transmitted to the FC, which can lead to a reliable decision mechanism. However, due to bandwidth constraint we have to quantize the data and this leads to error in the energy values. In hard decision combining, we can just transmit the decisions of the sensor nodes to the FC which can be binary values with 1 indicating that signal source is present and 0 indicating that a signal source is absent.

In open literature, most channel modeling have considered open areas with high speed trains [13, 14], while relatively little research has been conducted in tunnel environments. Due to various challenges presented by tunnel environments, it is important to derive a channel model for LTE-R involving high speed trains. In this paper, we analyze the effects of high Doppler shift and multipath due to tunnel environments. Experimental studies conducted inside tunnel environments have shown that the field amplitude distribution fits smoothly over a Rician distribution [20]. Several research efforts have been conducted on large-scale and small-scale fading of wideband communication systems inside tunnel environments. To the best of the authors knowledge, none of these studies have been

conducted for LTE-R, which employs Orthogonal Frequency Division Multiplexing (OFDM) signals for data transmission inside tunnels. The large Doppler shifts caused by high speed trains will potentially lead to ambiguity in extracting the carrier frequency, which can drastically increase the BER [21]. Therefore, it is important to study the effects of high Doppler shift and multipath fading for LTE-R communications in tunnel environments such that equalizers can be design efficiently.

1.3 Thesis Contributions

This thesis will contribute the following to the cognitive radio communications and cellular wireless field:

- A cooperative spectrum sensing test-bed with normalized energy detection using both soft data fusion and hard data fusion implemented on available software defined radios.
- For soft data fusion, Maximum Normalized Energy (MNE) and Equal Gain Combination (EGC) algorithms are used. Hard data fusion is also implemented using majority rule, AND, and OR approaches. Both USRP N210s [22] and RTL-SDRs [23] are employed for the implementation of the heterogeneous sensor network.
- A performance assessment and simulation of LTE-R communications in a tunnel environment experiencing severe fading.
- Dynamic K-factor for a tunnel environment is derived using the classical two-ray propagation model [24] and is used to build Rician fading model for the tunnel.

1.4 Thesis Organization

This thesis will be organized into the following chapters: Chapter 2 provides background knowledge on heterogeneous cooperative spectrum sensing and focuses on heterogeneous networks, cooperative spectrum sensing and software-defined radios. Chapter 3 discusses LTE-R in detail and provides the necessary understanding of the LTE-R implementation

in a tunnel environment. Leaky Coaxial Cable (LCX) , channel impairments and two-ray propagation model is also discussed in detail. In chapter 4 and 5 implementation of heterogeneous cooperative spectrum sensing (CSS) test-bed and LTE-R performance analysis in a tunnel is discussed. Chapter 6 presents the results of the implementation of heterogeneous CSS test-bed and LTE-R in a tunnel. Chapter 7 concludes this thesis, summarizing the accomplishments and outlines possible future work.

Chapter 2

Tutorial on 5G Communication System

Material Related to 5G

2.1 Summary

This chapter outlined and examined the topics of jamming and anti-jamming techniques, and provided a foundation in communication system theory and advanced equalizer design. Secondly it setup an understanding of Software-Defined Radio, the power of such an architecture, and examples of implementations and existing software for future designs. Next, this thesis will consider a new anti-jamming technique and design an implementation of such a system. After the implementation is investigated, the result of specific experiments on such an implementation will be analyzed.

Chapter 3

Heterogeneous Cooperative Spectrum Sensing (CSS)

This chapter provides the background information needed to understand the chapters that follows. It examines the basic outlines of a heterogeneous networks and how cooperative spectrum sensing (CSS) can help in enhancing the accuracy of signal source estimation. The fusion center (FC) collects the data from the sensor node network and process it to make the reliable decision. Secondly, this chapter investigates various algorithms which can be used in heterogeneous network to estimate signal source. Finally, it also outlines the necessary hardware and software tools used in the implementation of heterogeneous CSS chapter.

3.1 Cognitive Radios

Cognitive radio (CR) [25] is a communication systems paradigm that focuses on employing highly agile, environmentally aware, intelligent wireless platforms in order to autonomously select and configure device operating parameters based on the prevailing radio and network environmental conditions [3]. In general the cognitive radio may be expected to look at parameters such as channel occupancy rate, available channels, bandwidth required for data transmission and the modulation types that may be used. It must also look at the regulatory requirements set by the Federal Communications Commission. In

some instances a knowledge of geography and this may alter what it may be allowed to do. Software-defined radios (SDR) are mainly responsible for making cognitive radios used in wireless communications system a reality. Software radios provide a vast untapped potential to personalize services, and they make the process of modifying the radio characteristics extremely simple.

The work is underway to determine the best possible methods of developing the cognitive radio communications system that can fulfill all its requirements. To facilitate the intelligent decision making capabilities in these cognitive radio systems, machine learning algorithms have been proposed in the literature [26–29] to automate the reconfiguration process. The Figure 3.1 describes the various building blocks of a cognitive radio system. The spectrum sensing is performed to estimate the spectrum holes in the band and after the analysis the decision strategy is prepared. The radio is configured with the new parameters based on the radio environment and the spectrum decision made.

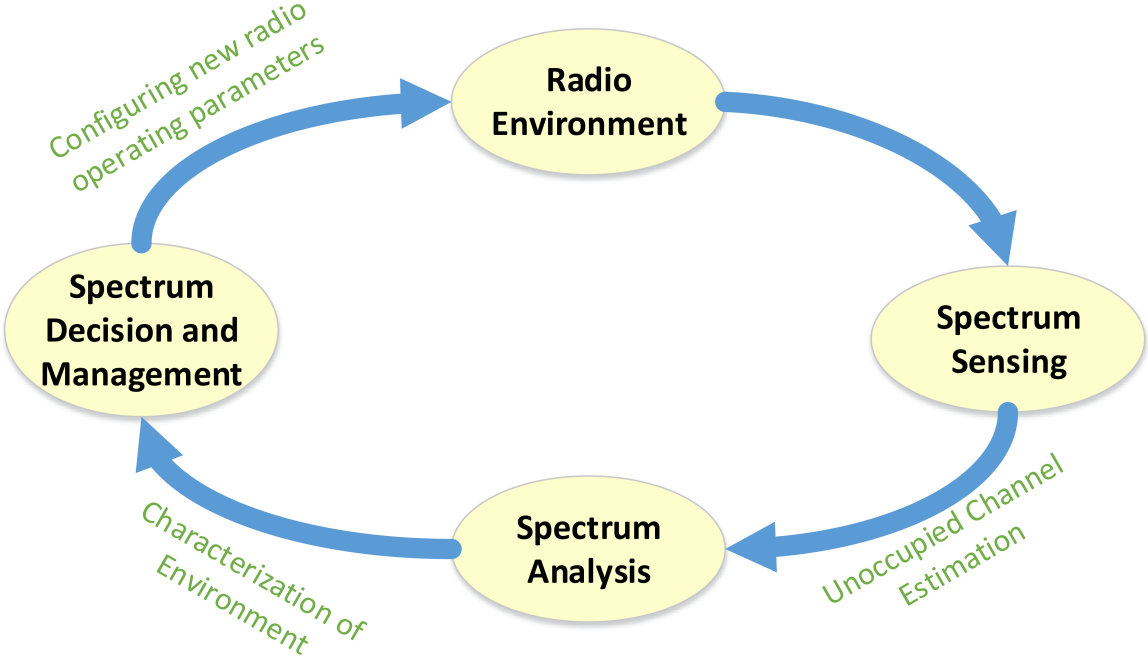

Figure 3.1: The block diagram explaining the basic parts of Cognitive Radio system. The operating parameters are configured based on the characterization of the wireless environment.

3.2 Cooperative Spectrum Sensing in Heterogeneous Networks

In Heterogeneous CR network, each radio is equipped with different numbers of antennas, sampling rates and RF characteristics. In addition, each sensor node may experience distinct channel fading and suffer from different noise levels due to their respective locations and device performances, such as amplifier and ADC. As a result, each node may have different sensing capabilities and reliability values. This is a universal and fundamental characteristic of a heterogeneous CR network which requires robust algorithms to achieve high accuracy in signal source detection for estimating the presence of a primary user [30]. In this thesis, we investigate the cooperative spectrum sensing in heterogeneous networks with a centralized FC, a transmitter acting as a signal source and four sensor nodes. As explained earlier we are using energy detection as one of the spectrum sensing technique since it possesses a very low implementation complexity [4]. The energy detection scheme detects the presence or absence of a signal source based on its intercepted energy signature. If the energy of the signal is higher than a certain threshold, this indicates that the channel is occupied. The ED can be modeled by the equation:

$$y(n) = \begin{cases} w(n), & \mathbf{H}_0 \\ s(n) + w(n), & \mathbf{H}_1 \end{cases} \tag{3.1}$$

where $y(n)$ represents the received signal, $s(n)$ represents the signal source PU, and $w(n)$ is the white Gaussian noise $w(n) \sim N(0, \sigma_n^2)$. $\mathbf{H}_0$ describes the hypothesis when there is no signal present, while the hypothesis $\mathbf{H}_1$ is the presence of signal.

The decision whether the signal is present or absent is decided by evaluating a local test statistic L to see whether it is above or below certain fixed threshold τ. The local test statistic L, which is the complex-magnitude squared of the FFT samples, is compared with τ using equation:

$$L = \sum_{n=1}^{M} |y(n)|^2 = \begin{cases} < \tau, & \mathbf{H}_0 \\ > \tau, & \mathbf{H}_1 \end{cases} \tag{3.2}$$

where $|y(n)|^2$ is the energy of a specific FFT bin and n=1,2,3...M are the number of samples received.

The probability of false alarm P_{fa} and probability of detection P_d are given by:

$$P_f = Q\left(\frac{\tau - M(2\sigma_n^2)}{\sqrt{M}(2\sigma_n^2)}\right), \tag{3.3}$$

$$P_d = Q\left(\frac{\tau - M(2\sigma_n^2)(1+\gamma)}{\sqrt{M(1+2\gamma)}(2\sigma_n^2)}\right). \tag{3.4}$$

In cooperative spectrum sensing, each sensor node transmits the local sensing data to the fusion center for signal source detection. The local sensing data has to be quantized, thus yielding quantization errors. To minimize the quantization error in local test statistic L and to reduce the effect of noise variance, the energy of the received signal $y(n)$ is normalized [30]. The local test statistic L for the r^{th} sensor node is given as:

$$L_r = \frac{1}{M_r \sigma_{n,r}^2} \sum_{r=1}^{M_r} |y(n)|^2 \tag{3.5}$$

where M_r is the number of samples used to estimate the power of the signal source in the node, $\sigma_{n,r}$ is the noise power variance.

In Eq (3.1) $s(n)$ is considered as a deterministic signal and $w(n)$ is a Gaussian random variable with a variance of σ_n^2. Based on CLT, L_r will have a following distribution [7]:

$$L_r = \begin{cases} N(1, \frac{1}{M_r}), & \mathbf{H}_0 \\ N(\gamma_r + 1, \frac{1+2\gamma_r}{M_r}), & \mathbf{H}_1 \end{cases} \tag{3.6}$$

where γ_r is the received SNR of the r^{th} SU. The local decision statistic L_r is quantized before transmission due to the bandwidth constraint, and this can lead to quantization errors. The values of L_r received by FC can be modeled as:

$$\beta_r = L_r + w_{q,r}, \tag{3.7}$$

where β_r is the decision statistic received by the FC and w_q is the noise added to the signal due to fading and quantization error. In [31], the w_q is modeled as a Gaussian noise with zero mean and σ_q^2 variance.

3.3 Software Defined Radios

We have already explained about the cooperative spectrum sensing in heterogeneous networks, now in this section we will look at the platform for testing the CSS algorithms. The platform which we use is a new hardware frontier called Software-Defined Radio (SDR) and is discussed in details.

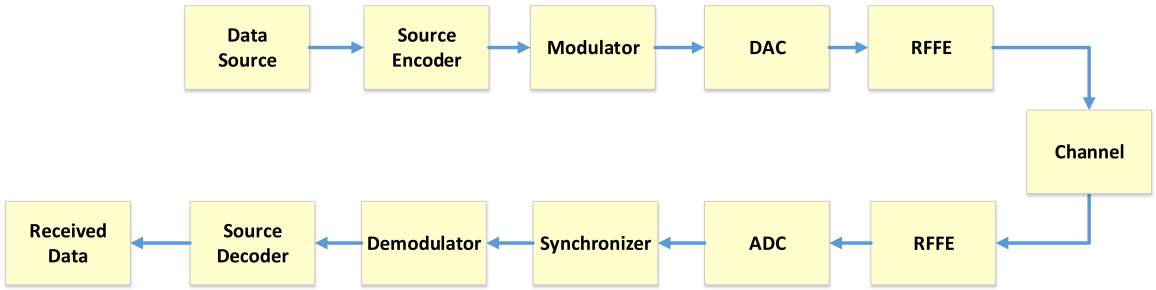

Figure 3.2: Software defined radio pushes all the adaptive elements and data manipulation operation into software. The goal of SDR is to provide or define all of the radio operation in software.

There has been a huge shift in the definition of a Software-Defined Radio and it has a lot to do with the question of where the hardware ends and software begins. Dr. J. Mitola III coined the Software Defined Radio which he described as a of digital signal processing (DSP) primitives, a meta-level system for combining the primitives into communication system (Tx, channel model, Rx, etc.), functions and a set of target processors on which the software radio is hosted for real-time communications [32]. Dr. J. Mitola explains in the paper how software provides the flexibility which using hardware alone can never be achieved. And as time progresses SDR would become more dominant and close to ideal Cognitive Radio system.

The SDR technology existed since the 1970s but the key milestone in the advancement of SDR technology took place in early 1990s with the U.S. military initiative called SpeakEasy I/II. The SpeakEasy project was implemented to use programmable processing to emulate more than ten existing military radios, operating in frequency bands between 2 MHz and 2 GHz [33]. With SpeakEasy the operator could talk to ten radios operating under different standards with any hardware modifications. With all these features, unfortunately there

were some shortcomings which left much to be desired. The device was large enough to fit on the back of a pickup truck [33] which is good for ground station but not if the mobility is an important factor. And in 1992 the field programmable gate arrays (FPGA) were not computationally efficient, hence required large time to change their operating characteristics. The two software-defined radios we used in the thesis are Universal Software Radio Peripheral (USRP N210) and RTL-SDR R2832U. In the subsequent subsections we discuss the two SDRs in detail.

3.3.1 Universal Software Radio Peripheral (USRP) N21 0

The USRP N210 has a very different RF characteristics compared to RTL-SDR hence modeling an ideal heterogeneous network. The USRP N210 provides a very high bandwidth, dynamic range processing capability. The product architecture includes a Xilinx Spartan 3A-DSP 3400 FPGA [34], 100 MS/s dual ADC, 400 MS/s dual DAC and gigabit ethernet connectivity to stream data to and from host processors. A modular design allows the USRP N210 to operate from DC to 6 GHz, while an expansion port allows multiple USRP N210 series device

Figure 3.3: USRP N210.

An optional GPSDO module can also be used to discipline the USRP N210 reference clock to within 0.01 ppm of the worldwide GPS standard. The USRP N210 can stream up to 50 MS/s to and from host applications. Users can implement custom functions in

the FPGA fabric, or in the on-board 32-bit RISC softcore. The USRP N210 provides a larger FPGA than the USRP N200 for applications demanding additional logic, memory and DSP resources. The FPGA also offers the potential to process up to 100 MS/s in both the transmit and receive directions. The FPGA firmware can be reloaded through the Gigabit Ethernet interface [22].

3.3.2 RTL-SDR Software-Defined Radio

RTL-SDR is a very cheap software defined radio that uses a DVB-T TV tuner dongle based on the RTL2832U chipset. With the combined efforts of Antti Palosaari, Eric Fry and Osmocom it was found that the signal I/Q data could be accessed directly, which allowed the DVB-T TV tuner to be converted into a wideband software defined radio via a new software driver.

Figure 3.4: RTL-SDR R2832u

Essentially, this means that a cheap $20 TV tuner USB dongle with the RTL2832U chip can be used as a computer based radio scanner. This sort of scanner capability would have cost hundreds or even thousands of dollars just a few years ago. The RTL-SDR is also often

referred to as RTL2832U, DVB-T SDR, RTL dongle or the $20 Software Defined Radio.

3.3.3 GNURadio and Software-Defined Radio

The first software package to be discussed by this thesis is GNU Radio. GNU Radio provides the reconfigurable signal processing blocks that are necessary for software defined radios. GNU Radio is an open source project allowing for SDR developers to develop unique signal processing blocks and SDR systems. GNU Radio was started in 2001, originally forked from the SpectrumWare project developed at the Massachusetts Institute of Technology [46]. Since 2001, the code base has undergone massive changes, containing almost no code from the original SpectrumWare project. Physically the code consist of three languages Python, C++, and SWIG. Python provides the overarching control of the system or program, while C++ provides the actual signal processing blocks and mathematics. SWIG is a wrapper for C++ which allows Python to dynamically wrap around C++ and control or compile with it. A diagram below better illustrates this architecture. It is also important to mention that there as significant paradigm shifts in the community, pushing more and more code to Python rather than C++, due to its easier programming syntax and structure.

GNU Radio provides a very structured framework of flow design. Data processing segments are extremely self contained to minimize error propagation during system debugging. Since the software is open-source full access to all code is provide, giving low-level access to all operation within GNU Radio. Much of the actions have been abstracted to limited the knowledge of the lower layers, but if specific actions are required for an application. Then serious depth or knowledge is needed about the overall projects structure, which is quite overloading.[Travisthesis]

3.3.4 MATLAB

MATLAB is an extremely well known engineering, mathematical, biological, and financial software suite. MATLAB provide massive data leverage and advanced communication system models and algorithm for significant data processing. Since 2007, they have also provided hardware compliance with specific SDR platforms through their Simulink platform, and more recently within MATLAB itself [47]. This thesis primarily utilizes the sig-

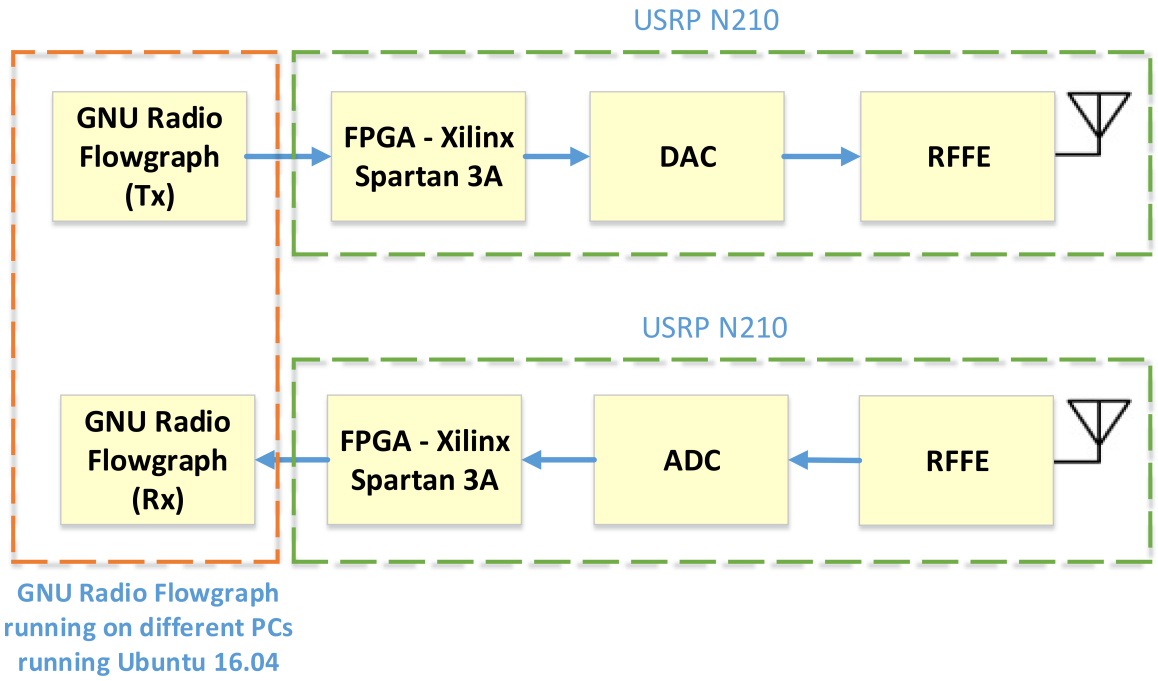

Figure 3.5: GNURadio and Software-Defined Radio

nal processing and communication system aspects of MATLAB, since MATLAB cannot fully utilize all aspects of the chosen hardware. It is important to note under alternate constraints, MATLAB can provide adequate performance directly interfacing with hardware, especially when accessing its targeting features seen here [48]. Figure 2.10 shows an example of a common MATLAB SDR model through Simulink.[Travis]

3.4 Summary

This chapter outlined and examined the topics of jamming and anti-jamming techniques, and provided a foundation in communication system theory and advanced equalizer design. Secondly it setup an understanding of Software-Defined Radio, the power of such an architecture, and examples of implementations and existing software for future designs. Next, this thesis will consider a new anti-jamming technique and design an implementation of such a system. After the implementation is investigated, the result of specific experiments on such an implementation will be analyzed.

Chapter 4

Long Term Evolution for Railways (LTE-R)

4.1 LTE-R Communication System

LTE-R System Description To provide improved and more efficient transmission for HSR communications, it is vital to consider frequency and spectrum usage for LTE-R. HSRs are important stra- tegic infrastructure, and, in some countries, this argument is being leveraged to convince governments that large spectrum chunks need to be allocated specifically for it. Some industry bodies, including the European Railway Agency (ERA), China Railway, and UIC, are working to secure spectrum allocation for HSR use. Currently, most LTE systems work at the bands above 1 GHz, such as 1.8, 2.1, 2.3, and 2.6 GHz, although 700900-MHz bands are also used in some countries. Large bandwidth is available in the upper bands, giving a higher data rate, whereas lower frequency bands offer longer distance coverage. Figure 1(b) summarizes the possible frequen- cy bands for LTE-R in China, Europe, and Korea. As a high-frequency band has larger propagation loss and more severe fading, the radius of an LTE-R cell would be ¡2 km [due to the strict requirement of signal-to-noise ratio (SNR) and BER in HSR], leading to frequent handovers and a requirement of substantial investment for higher BS density. Therefore, the low-frequency bands, such as 450470 MHz, 800 MHz, and 1.4 GHz, have been widely considered. The 450470-MHz band is already

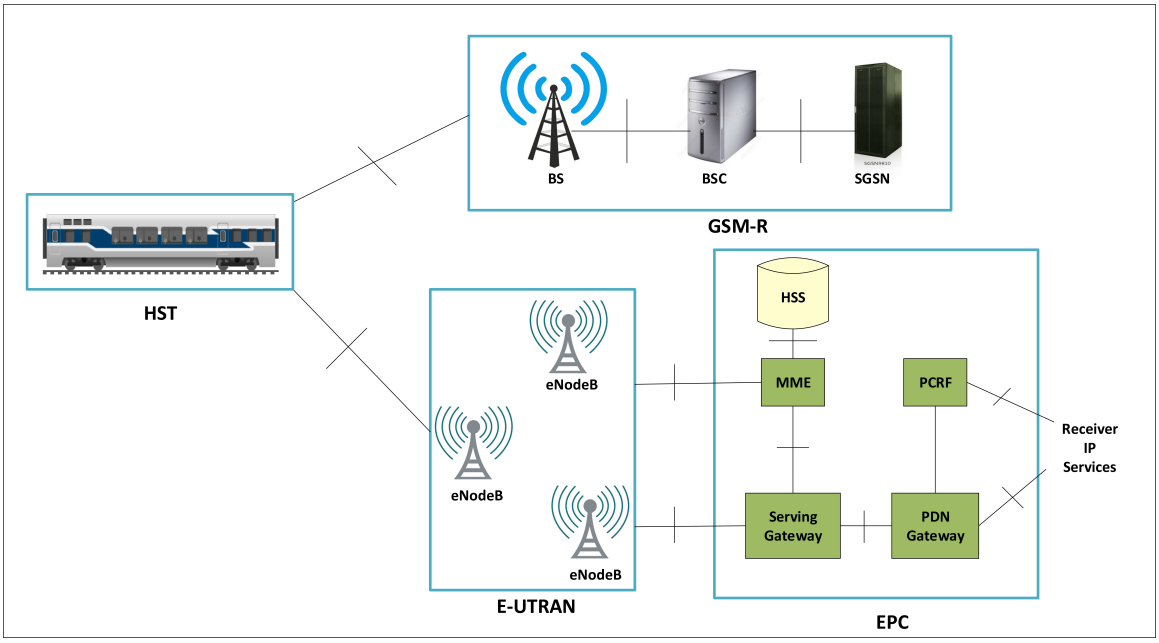

Figure 4.1: High speed train inside a tunnel for LTE-R. D_{LOS} is the distance between transmitter and receiver, d is the distance between the LCX cable transmission slots.

well adopted by the railway industry; therefore, dedicated bandwidth for professional use can still be allocated from local regulators. Furthermore, the carrier aggregation capability of LTE will permit the use of different bands to overcome problems of capacity. Figure 1(b) presents the detailed frequency allocation of 450 470 MHz in China [12], and it is feasible to allocate enough bandwidth for LTE-R within this band. In Europe, the FRMCS of UIC would like to build on the current GSM-R investment by reusing the existing mast sites, which could save as much as 8090% of the cost of a network. Railways are also concerned about continu- ing to make use of their GSM-R masts, and, therefore, a spectrum allocation under 1 GHz is more cost effective in Europe. However, the selection of frequency band depends on government policy and differs by country.

Standard LTE includes a core network of evolved packet core (EPC) and a radio access network of Evolved Universal Terrestrial Radio Access Network (E-UTRAN). The Internet protocol (IP)-based EPC supports seamless handovers for both voice and data to cell towers, and each E-UTRAN cell will support high data and voice capacity by high-speed packet access (HSPA). As a candidate for the next-generation communication system of HSR, LTE-

R in- herits all the important features of LTE and provides an extra radio access system to exchange wireless signals with onboard units (OBUs) and to match HSR-specific needs. The future architecture of LTE-R according to [4] is pre- sented in Figure 2, and it shows that the core network of LTE-R is backward compatible with GSM-R.

Compared with the public LTE networks, LTE-R has many differences, such as architecture, system parameters, network layout, services, and QoS. The preferred parameters of LTE-R are summarized in Table 1, based on the future QoS requirements of HSR communications. Note that LTE-R will be configured for reliability more than capacity. The network must be able to operate at 500 km/h in complex railway environments. Therefore, quadrature phase-shift keying (QPSK) modulation is preferred, and the packet number of retransmission must be reduced as much as possible.[RuisiHeBoAiGongpuWang/HSRCommunications]

4.2 Leaky Coaxial Cable

Substantial work has been done for the radio communication in tunnel environments.

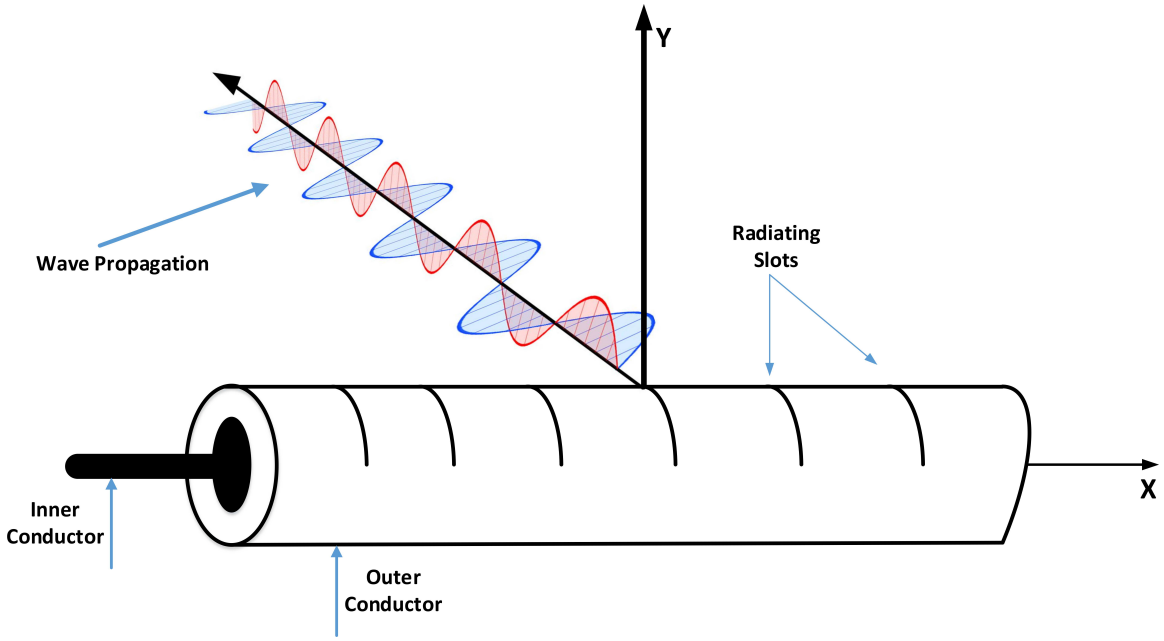

Figure 4.2: Leaky Coaxial Cable

Authors of [2]measure the radio channel frequency response; the large scale and small

scale fading in tunnel are investigated through field experiments. Statistical characteristics of the propagation channels are suggested in [3]. Both [2] and [3] indicate that WLAN cannot achieve good performance when used in tunnel environments. As a result, leaky waveguide and leaky coaxial cable are used in wireless communication due to their stability and robustness from the interference. The availability of train ground communication in subway systems by waveguide is analyzed in [4] and [5]. Meanwhile, leaky coaxial cable is lso studied. H. Cao [7] introduces the way to calculate frequency range of LCX and shows experimental results about LCX performance at 1.8GHz. Authors of [11] introduce the propagation at 2GHz in an enclosed area using a leaky coaxial cable. Authors of [12] study the theory of leaky coaxial cable with periodic slots, and frequency band of the studied LCXs is from 450MHz to 900MHz. However, few people specialize in the performance of leaky coaxial cable at 2.4GHz band, especially the application in tunnels.[HongweiWangBinNingandHailinJiang]

4.3 Channel Impairements Inside a Tunnel

The tunnel environment is affected by multipath and diffraction effects due to multiple reflections from the tunnel walls, which leads to a substantial fading environment. By deploying LCX cables, we can eliminate the large penetration loss due to tunnel walls. However, small-scale fading can still cause a large amount of errors and decrease the QoS for the communication link.

High velocity trains experience very high Doppler shifts and a fast fading channel. These problems can lead to significant BER degradation of the LTE system. The frequency shifts caused by the Doppler phenomenon can lead to shifts in the sub-carrier frequencies for OFDM, which leads to synchronization errors. The maximum Doppler shifts for a train traveling at 500 km/h is 2.314 kHz for a 5 GHz carrier frequency. This large Doppler shift can also lead to significant drops in the quality of wireless signals and increase the bit error rate. Thus, to develop an efficient and reliable communication link inside tunnels, we need to properly model this channel impairment and build our proposed channel model by taking into account these tunnel phenomenons. These impairments are described in detail in the

following subsections.[OwnPaper]

4.3.1 Multipath Fading

The following time-varying multipath channel impulse response considers the effects of Doppler shift and scattering [11]: where is the path delay, t is time in seconds, [k (t)] is the impulse response, f c is the carrier frequency, h k (t) is the envelope of the time-varying channel and consists of both large and small-scale fading components. Since the structure of LCX is almost the same as a leaky waveguide, the large scale fading of channel can be modeled linearly [8]. There is also no signal shadowing and the line-of-sight (LOS) signal component is always present along the tunnel. This type of channel fading can be best described by a Ricean fading model. The probability density function p() of a Rician fading model is given by [10]: where K is the Rician factor and is the complex amplitude of the channel response function that has a unity second moment.

4.3.2 Doppler Shift

The 3GPP channel model [14] is used for its Doppler shift profile in high speed railway environment. The measurements obtained for the Doppler frequency shift are implemented for two scenarios. The first scenario is for an open space while the second scenario is for high speed trains. Doppler shift is not taken into consideration. There exists a third scenario for tunnels using multiple antennas. Since the slots of LCX can be modeled as multiple antenna system, we use this Doppler shift profile for our proposed channel. The Doppler shift variation f s (t) is described by

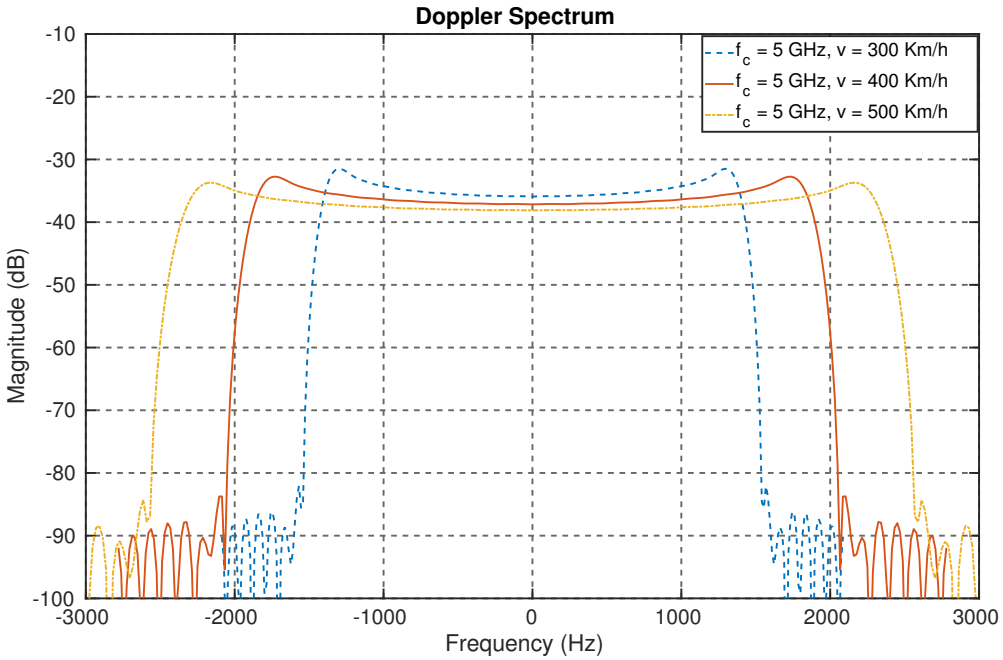

Figure 4.3: Doppler spectrum for LTE-R at different train velocities v (km/h) = 300, 400 and 500 and $f_c = 5$ GHz.

4.4 Two-Ray Propagation Model

4.4.1 Classical Two-Ray Propagation Model

4.4.2 Dynamic K-factor

4.5 Summary

This chapter outlined and examined the topics of jamming and anti-jamming techniques, and provided a foundation in communication system theory and advanced equalizer design. Secondly it setup an understanding of Software-Defined Radio, the power of such an architecture, and examples of implementations and existing software for future designs. Next, this thesis will consider a new anti-jamming technique and design an implementation of such a system. After the implementation is investigated, the result of specific experiments on such an implementation will be analyzed.

Chapter 5

Implementation

5.1 Overview

5.2 Experimental Setup for Heterogeneous CSS

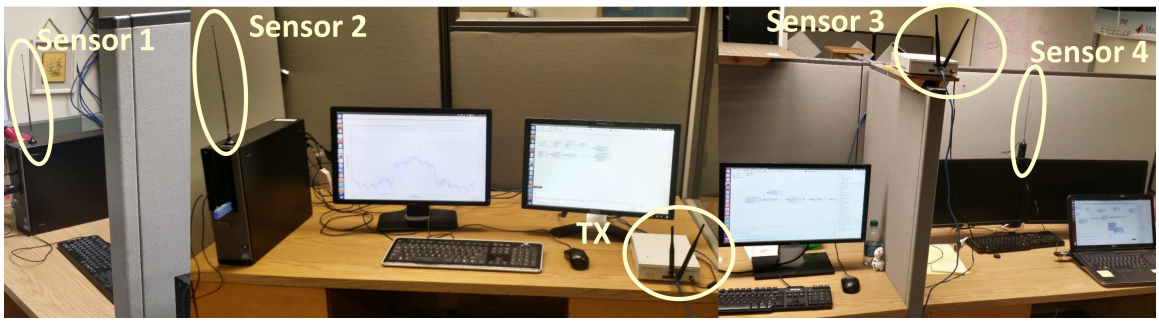

Figure 5.1: Experimental Test-Bed For Cooperative Sensing in Heterogeneous Network. Sensors 1, 2 and 4 are RTL-SDR units, sensor 3 is USRP N210 and TX is another USRP N210 unit which is used as a signal source for this work.

5.2.1 Transmitter Setup for CSS Measurements

5.2.2 Sensor Nodes Setup

The central frequency is kept at 450 MHz and the operating parameters of each sensor node is provided in the table.

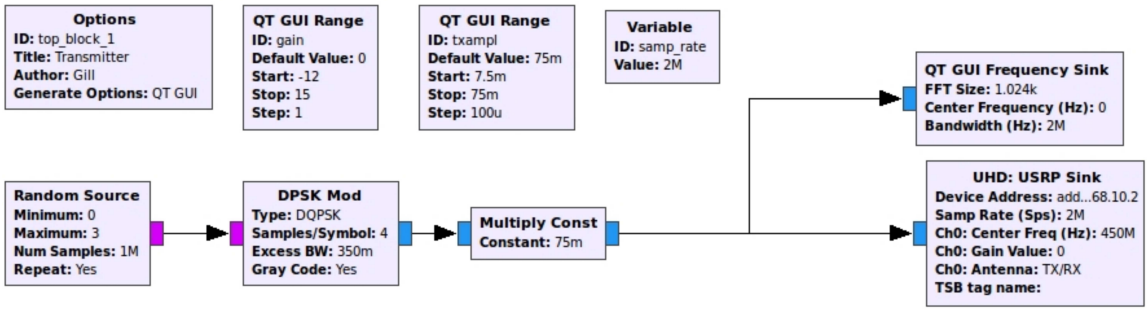

Figure 5.2: GNU Radio Flowgraph For Transmitter Running on USRP N210.

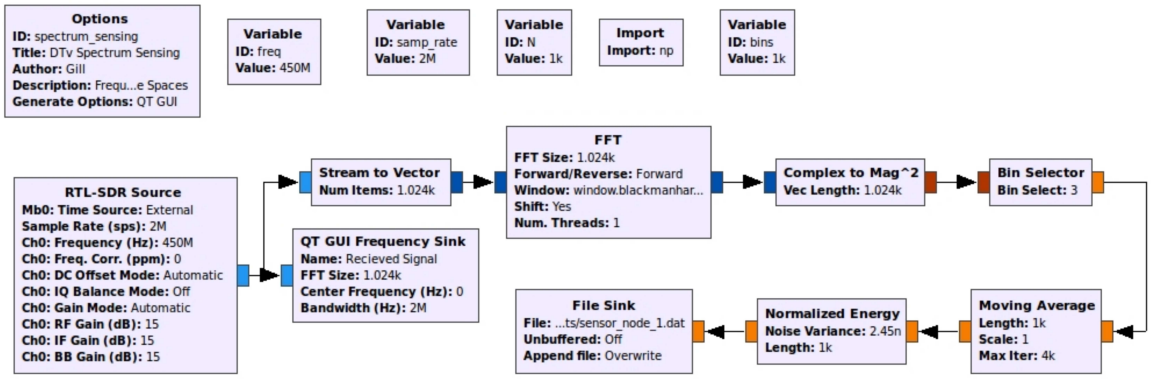

Figure 5.3: GNURadio Flowgraph For USRP and RTL-SDR sensor nodes.

Table 5.1: Operating Characteristics of Sensor Nodes

Nodes	F_s	Gain	FFT Size	Bin Size
RTL-SDR-1	1.1 Msps	10 dB	512	2.148 KHz
RTL-SDR-2	1.8 Msps	10 dB	512	3.515 KHz
RTL-SDR-3	2.4 Msps	10 dB	512	4.687 KHz
USRP N210	7.2 Msps	10 dB	512	14.062 KHz

5.3 Hard Fusion

The flowchart describes the three hard data fusion schemes.

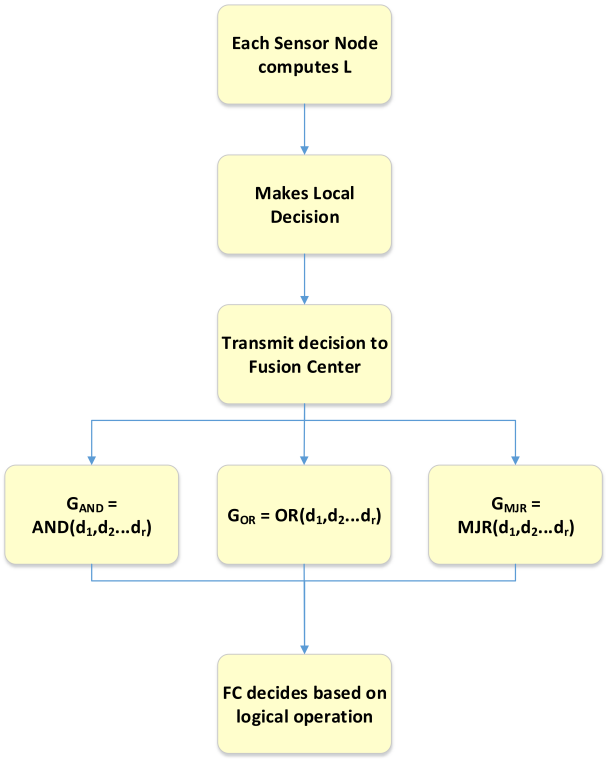

Figure 5.4: Flowchart showing AND, OR and Majority Rule Fusion schemes..

Table 5.2: Tunnel and Tx/Rx Characteristics

	Dimensions	Simulation Parameters
Tunnel	Width = 8.6 m, Height = 7.3 m	$\varepsilon_r = 5$, $\sigma = 0.1$ Sm^{-1}
Leaky Feeder Cable (Tx)	Height = 6.1 m	f_c (GHz) = 2, 3, 5
Train (Rx)	Height = 4.2 m	v (km/h) = 300, 400, 500

5.4 Soft Fusion

5.4.1 Maximum Normalized Energy Scheme

5.4.2 Equal Gain Combining Scheme

5.5 LTE-R Testbed in Matlab

The below table gives the tunnel parameters which we used in the thesis for MATLAB simulaton.

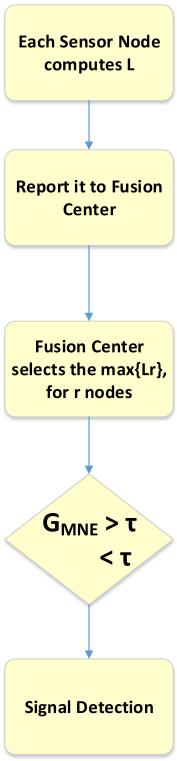

Figure 5.5: Flowchart showing Equal Gain Combining Scheme.

5.5.1 HST Channel Model

5.5.2 LTE-R OFDMA

5.6 Summary

This chapter outlined and examined the topics of jamming and anti-jamming techniques, and provided a foundation in communication system theory and advanced equalizer design. Secondly it setup an understanding of Software-Defined Radio, the power of such an architecture, and examples of implementations and existing software for future designs. Next, this thesis will consider a new anti-jamming technique and design an implementation of such a system. After the implementation is investigated, the result of specific experiments on such an implementation will be analyzed.

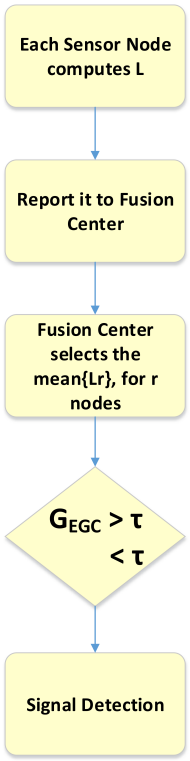

Figure 5.6: Flowchart showing Maximum Normalized Energy Scheme.

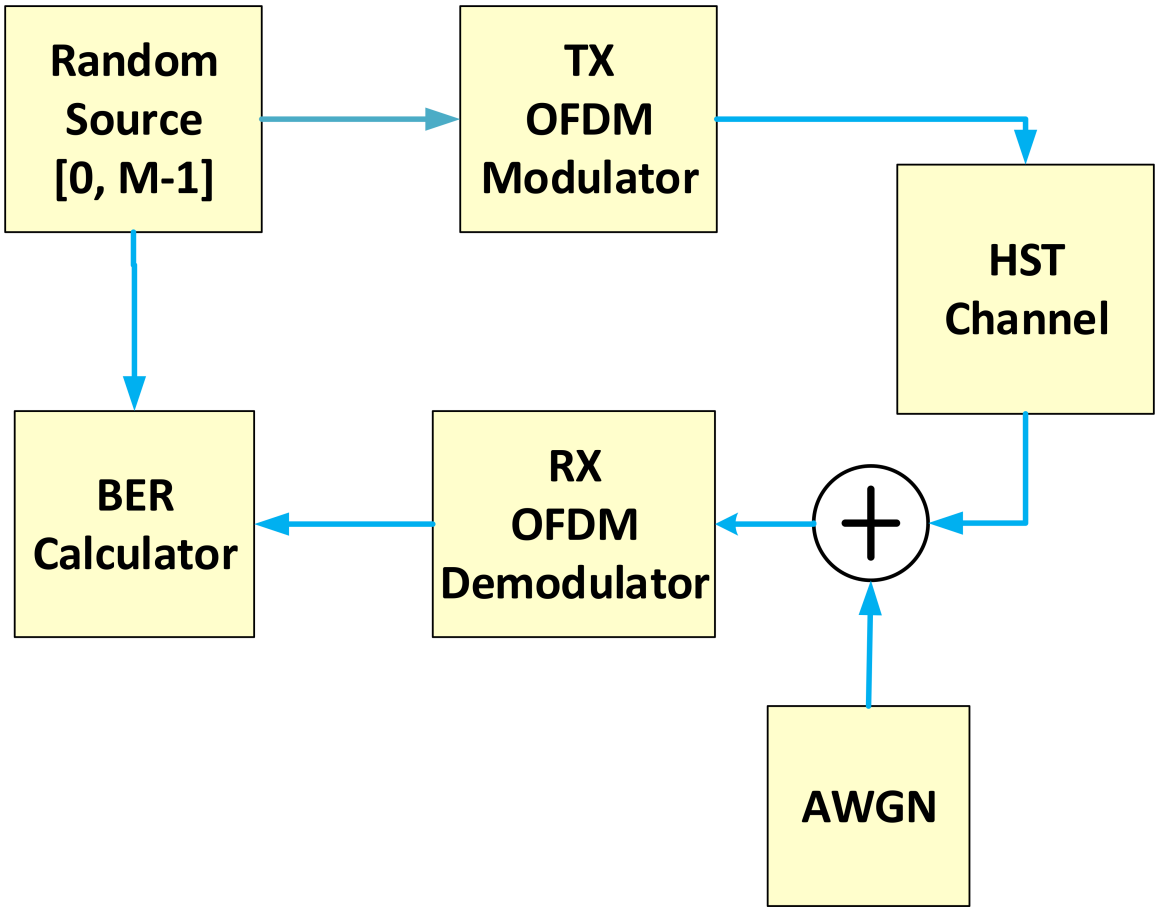

Figure 5.7: Block diagram of a communication system through a HST channel using QPSK, 16-QAM and 64-QAM.

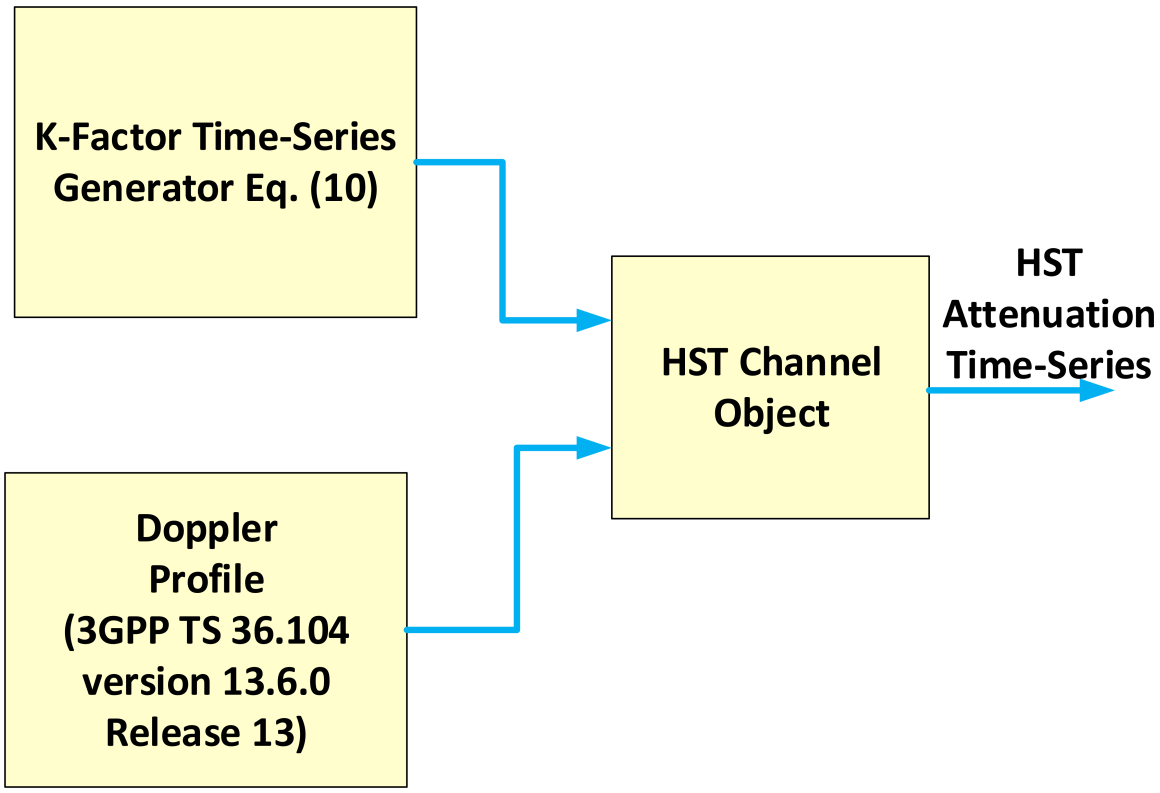

Figure 5.8: HST channel model consisting of time-series K-factor and Doppler shift caused due to velocity of the train.

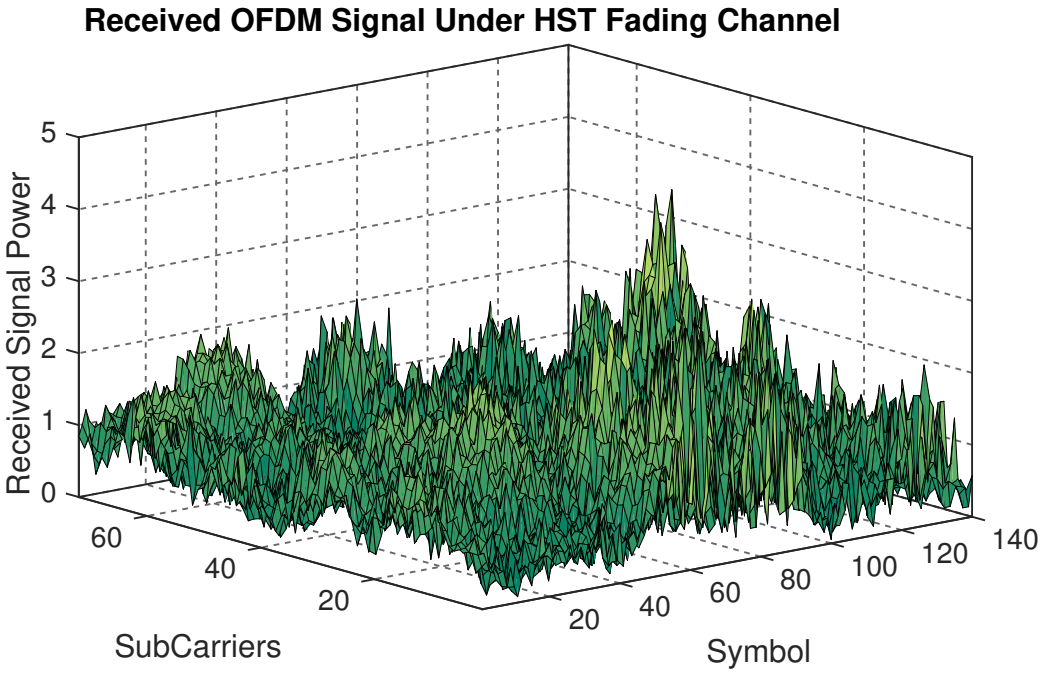

Figure 5.9: Received LTE-R OFDM signal under HST Ricean Fading Environment.

Chapter 6

Experimental Results

6.1 Overview

6.2 Heterogeneous CSS Results

6.2.1 Hard Decision Combining

6.2.2 Soft Decision Combining

6.3 HST LTE-R in a Tunnel

6.3.1 K-factor in a Tunnel

6.3.2 BER Performance

6.3.3 Real-time BER in a Tunnel

6.4 Summary

Spectral Subtraction in a extremely well studied area in signal processing, but no existing literature exists for its application in a digital communication system. It has been shown here that it can be quite difficult for it to be applied, even under strict constraints. Under the conditions of this thesis, the assumptions are quite reasonable, but due to the large amount of error in the results, more may need to be considered. These may include

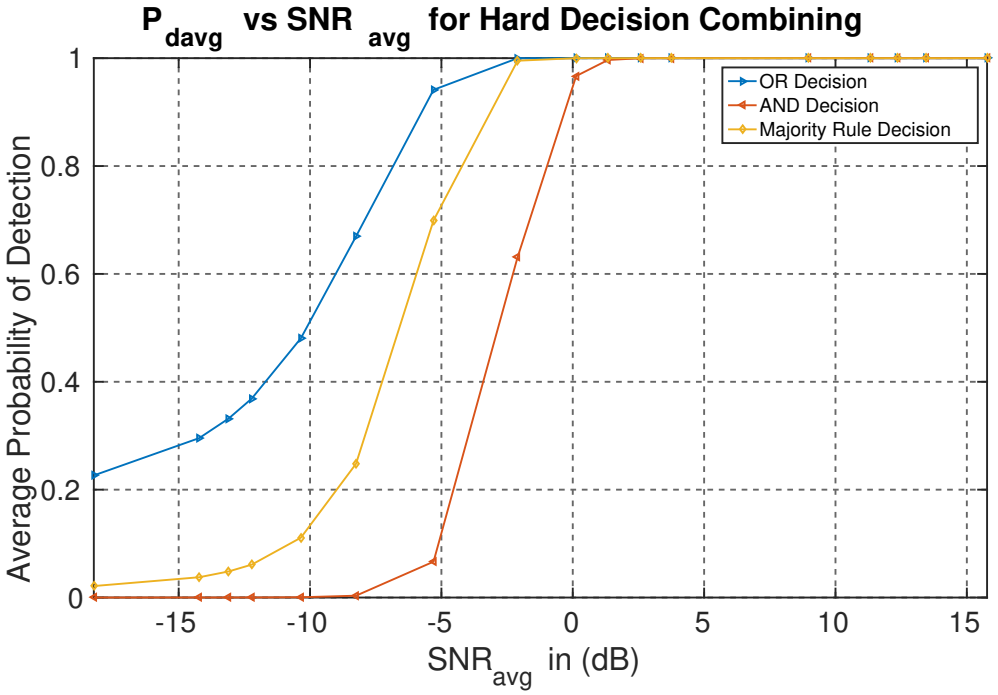

Figure 6.1: Probability of Detection versus SNR_avg For Hard Decision Combining.

accuracy requirements for physical equipment, primarily to reduce carrier frequency drift. Burst scenarios may also be considered to reduce bit error rate. Overall, for a completely non-existent field of study, these results point the possibility of operational success. Future work will we required, especially during the implementation phase of designs.

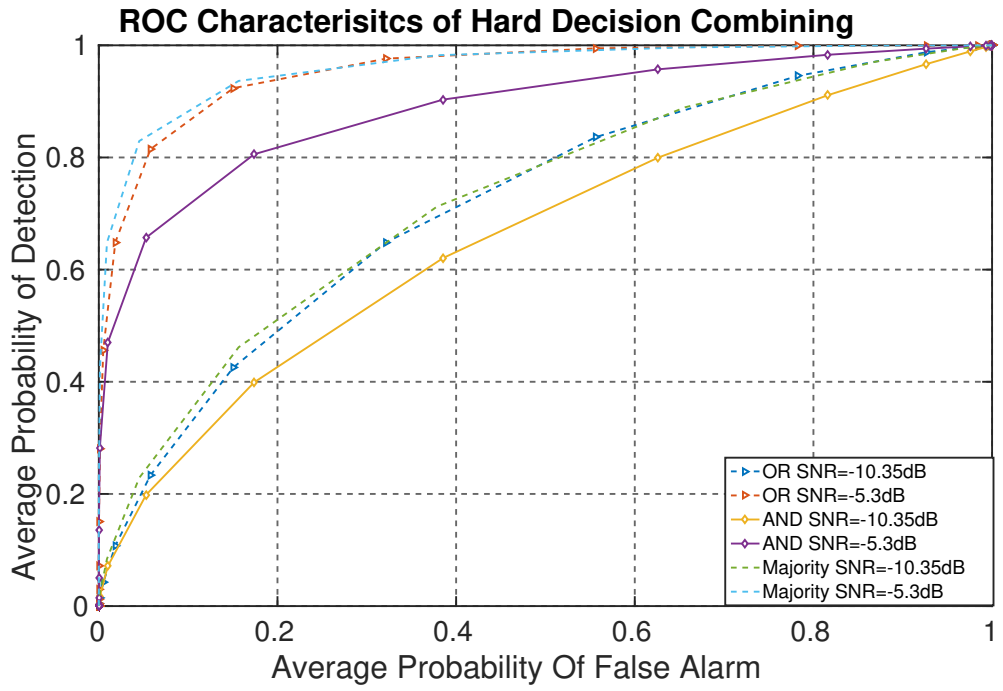

Figure 6.2: ROC Characteristics for Hard Decision Combining with Different SNRs.

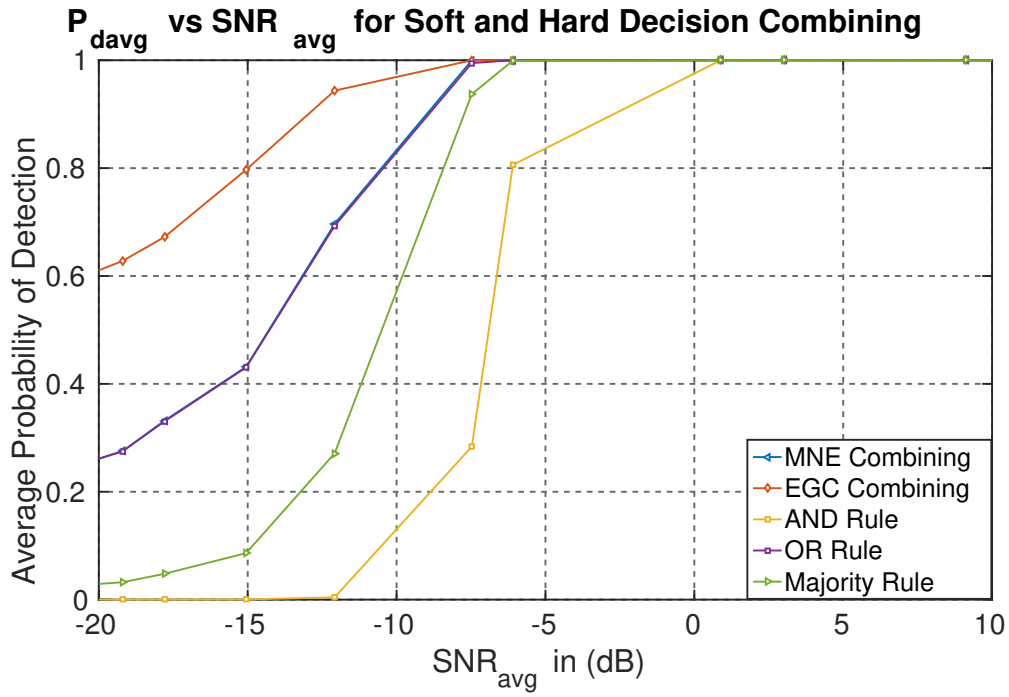

Figure 6.3: Probability of Detection versus SNR_avg For Soft and Hard Decision Combining.

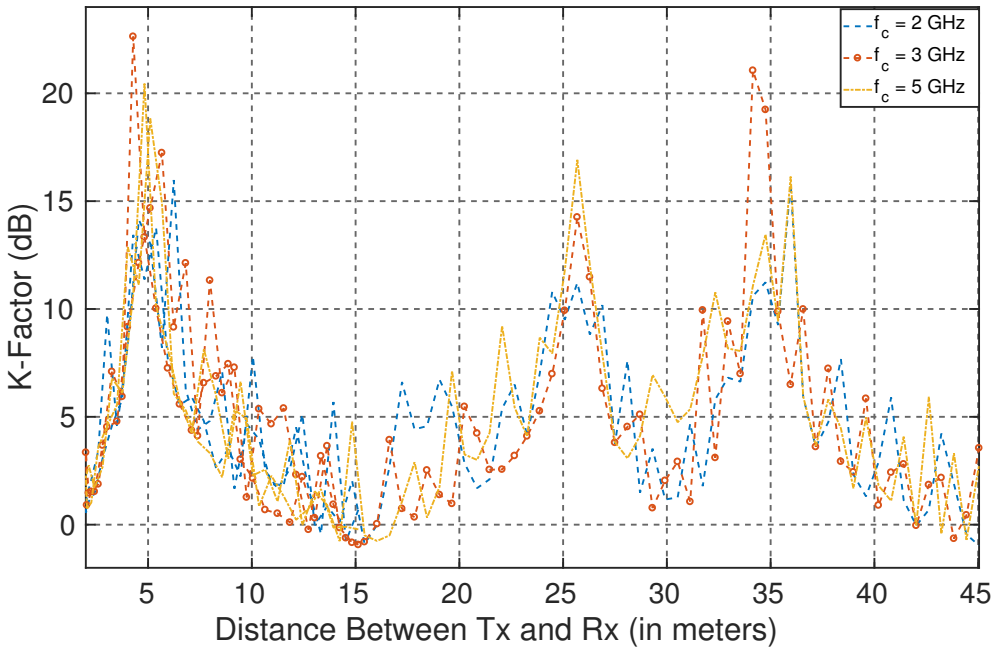

Figure 6.4: K-factor versus D_{LOS} for different center frequencies f_c = 2, 3 and 5 GHz.

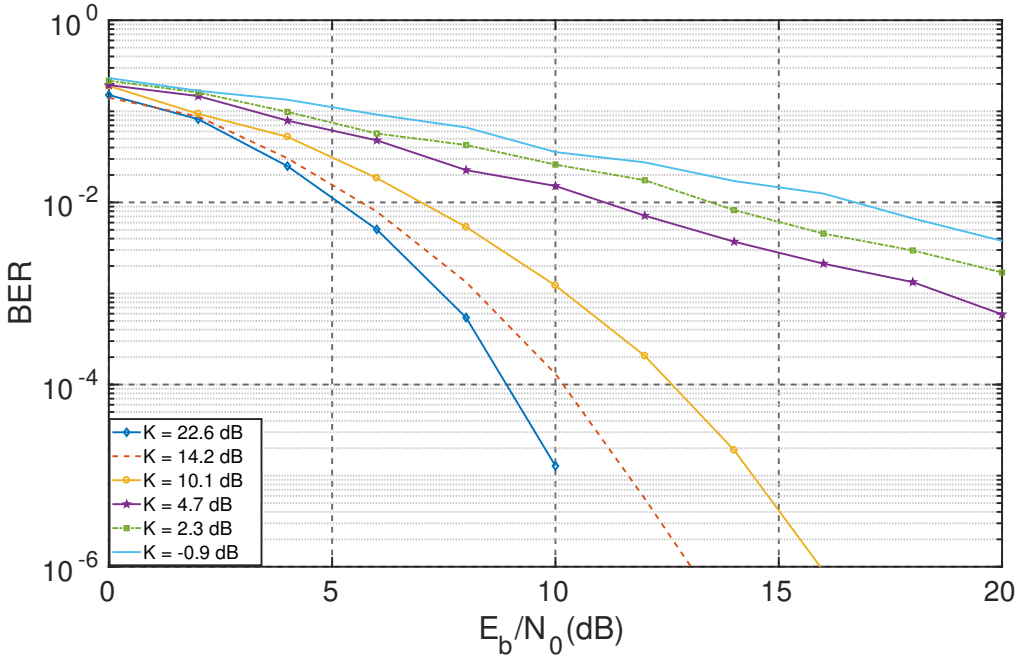

Figure 6.5: Comparison of E_b/N_0 verus BER for LTE-R OFDM-QPSK modulation with different K-factors.

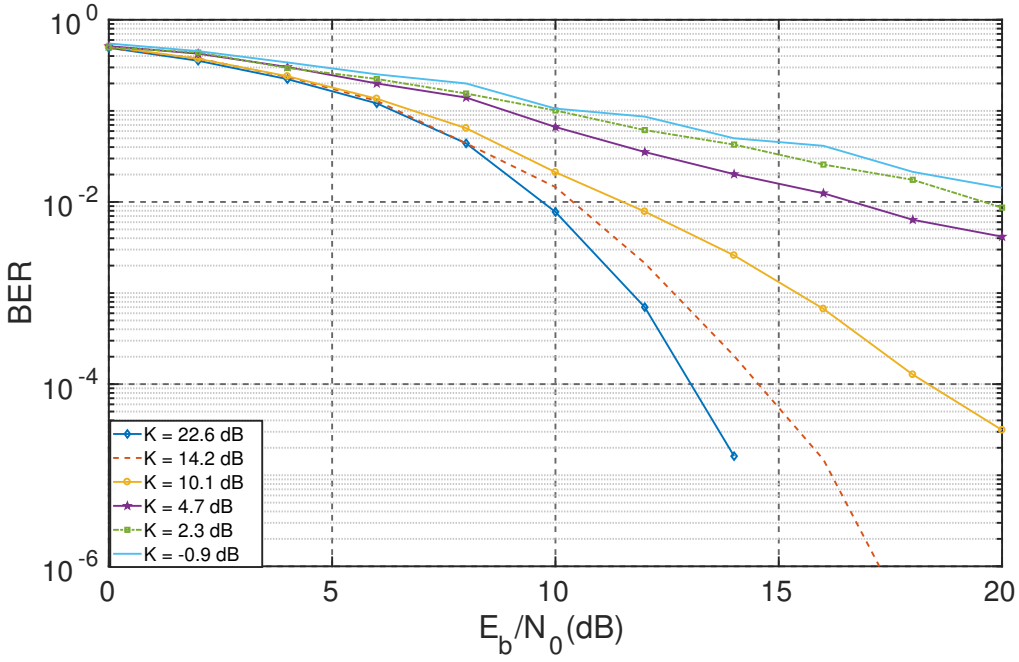

Figure 6.6: Comparison of E_b/N_0 verus BER for LTE-R OFDM-16QAM modulation with different K-factors.

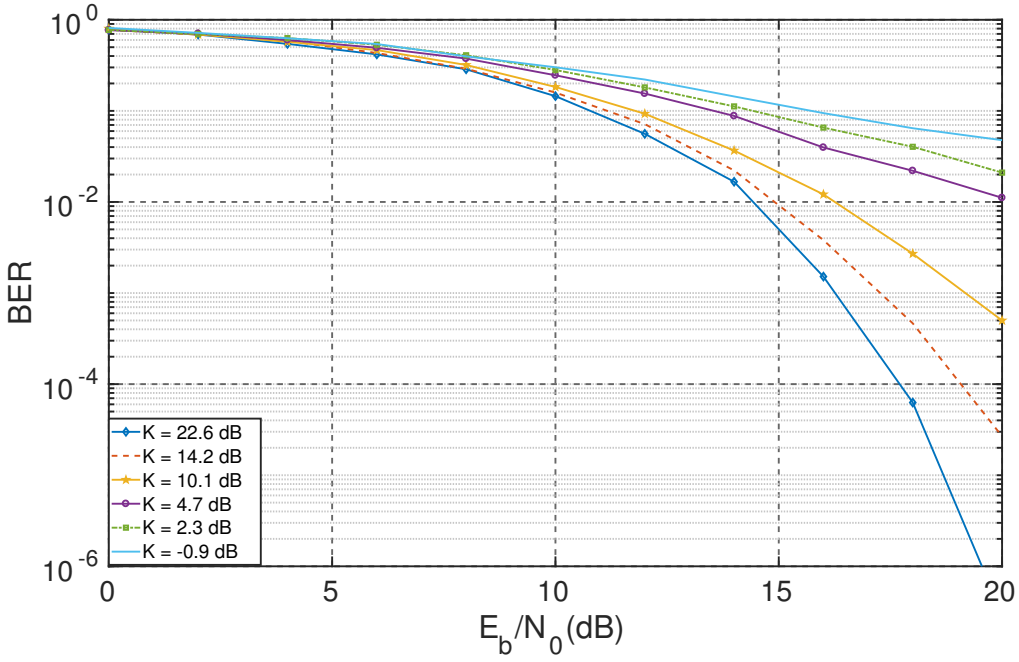

Figure 6.7: Comparison of E_b/N_0 verus BER for LTE-R OFDM-64QAM modulation with different K-factors.

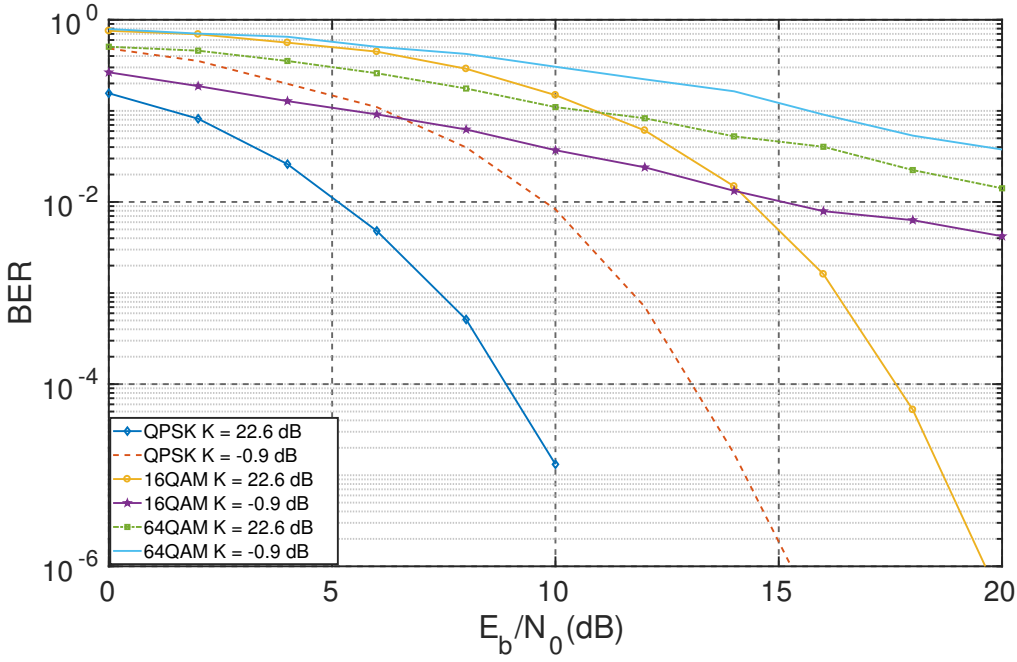

Figure 6.8: Comparison of E_b/N_0 verus BER for LTE-R OFDM modulation with different K-factors.

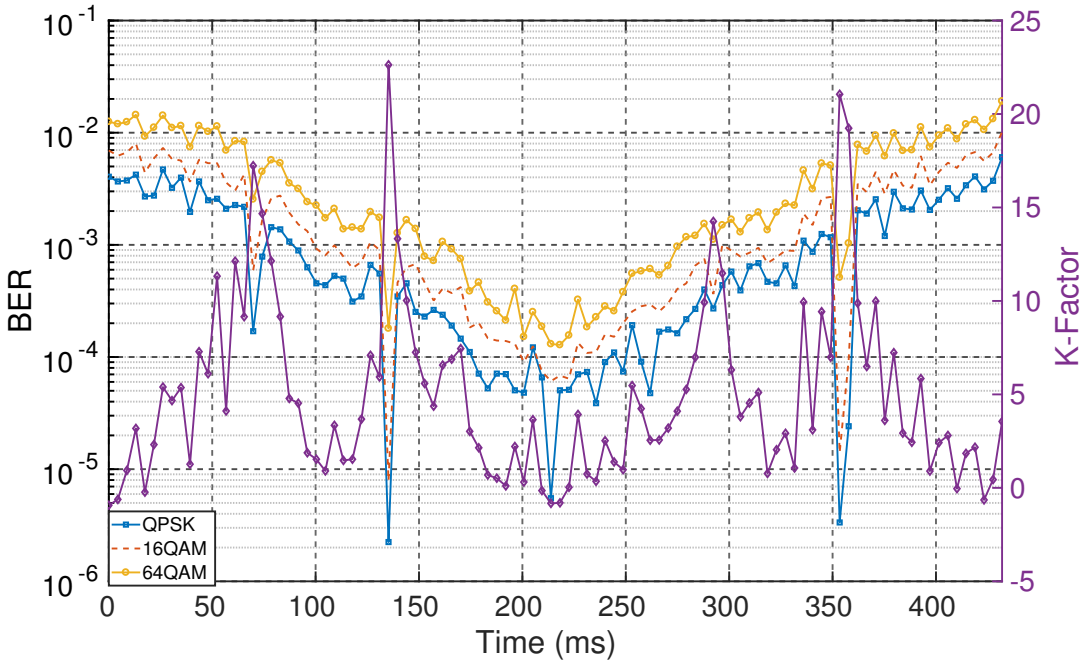

Figure 6.9: BER variation with time for HST with different modulation schemes of LTE-R. As the train moves towards the antenna the general trend of BER goes down with small-scale fluctuations due to varying K-factor.

Chapter 7

Conclusions

7.1 Research Outcomes

7.2 Future Work

Bibliography

[1] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, "Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey," *Computer networks*, vol. 50, no. 13, pp. 2127–2159, 2006.

[2] Q. Zhao and B. M. Sadler, "A survey of dynamic spectrum access," *IEEE signal processing magazine*, vol. 24, no. 3, pp. 79–89, 2007.

[3] A. M. Wyglinski and D. Pu, *Digital communication systems engineering with software-defined radio*. Artech House, 2013.

[4] H. Urkowitz, "Energy detection of unknown deterministic signals," *Proceedings of the IEEE*, vol. 55, no. 4, pp. 523–531, 1967.

[5] D. Cabric, S. M. Mishra, and R. W. Brodersen, "Implementation issues in spectrum sensing for cognitive radios," in *Signals, systems and computers, 2004. Conference record of the thirty-eighth Asilomar conference on*, vol. 1. Ieee, 2004, pp. 772–776.

[6] R. Tandra and A. Sahai, "Fundamental limits on detection in low snr under noise uncertainty," in *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, vol. 1. IEEE, 2005, pp. 464–469.

[7] S. M. Mishra, A. Sahai, and R. W. Brodersen, "Cooperative sensing among cognitive radios," in *Communications, 2006. ICC'06. IEEE International conference on*, vol. 4. IEEE, 2006, pp. 1658–1663.

[8] A. Ghasemi and E. S. Sousa, "Impact of user collaboration on the performance of

sensing-based opportunistic spectrum access," in *Vehicular Technology Conference, 2006. VTC-2006 Fall. 2006 IEEE 64th.* IEEE, 2006, pp. 1–6.

[9] U. G.-R. F. Group, "Gsm-r functional requirement specification (FRS)," UIC, Paris, France UIC EIRENE Technology Report, Tech. Rep. UIC Code 950, Version 7.3.0, 2012.

[10] G. T. 36.201, "Evolved universal terrestrial radio access (E-UTRA); lte physical layer; general description," 3GPP, Sophia Antipolis, France, Tech. Rep. version 12.2.0, Release 12, 2015.

[11] K. Masur and D. Mandoc, "Lte/saethe future railway mobile radio system? long term visions on railway mobile radio technologies," *International Union of Railways (UIC), Technical Report*, vol. 1, 2009.

[12] G. Tingting and S. Bin, "A high-speed railway mobile communication system based on lte," in *Electronics and Information Engineering (ICEIE), 2010 International Conference On*, vol. 1. IEEE, 2010, pp. V1–414.

[13] K. Guan, Z. Zhong, and B. Ai, "Assessment of lte-r using high speed railway channel model," in *Communications and Mobile Computing (CMC), 2011 Third International Conference on.* IEEE, 2011, pp. 461–464.

[14] H. Wei, Z. Zhong, K. Guan, and B. Ai, "Path loss models in viaduct and plain scenarios of the high-speed railway," in *Communications and Networking in China (CHINACOM), 2010 5th International ICST Conference on.* IEEE, 2010, pp. 1–5.

[15] D. G. Dudley, M. Lienard, S. F. Mahmoud, and P. Degauque, "Wireless propagation in tunnels," *IEEE Antennas and Propagation Magazine*, vol. 49, no. 2, pp. 11–26, 2007.

[16] J. Ma, G. Zhao, and Y. Li, "Soft combination and detection for cooperative spectrum sensing in cognitive radio networks," *IEEE Transactions on Wireless Communications*, vol. 7, no. 11, pp. 4502–4507, 2008.

[17] B. Shen, T. Cui, K. Kwak, C. Zhao, and Z. Zhou, "An optimal soft fusion scheme for cooperative spectrum sensing in cognitive radio network," in *Wireless Communications and Networking Conference, 2009. WCNC 2009. IEEE.* IEEE, 2009, pp. 1–5.

[18] H. Rifà-Pous, M. J. Blasco, and C. Garrigues, "Review of robust cooperative spectrum sensing techniques for cognitive radio networks," *Wireless Personal Communications*, vol. 67, no. 2, pp. 175–198, 2012.

[19] Y. Zeng, Y.-C. Liang, S. Zheng, and E. C. Peh, "Optimal cooperative sensing and its robustness to decoding errors," in *Communications (ICC), 2011 IEEE International Conference on.* IEEE, 2011, pp. 1–5.

[20] J.-M. Molina-Garcia-Pardo, M. Lienard, A. Nasr, and P. Degauque, "Wideband analysis of large scale and small scale fading in tunnels," in *ITS Telecommunications, 2008. ITST 2008. 8th International Conference on.* IEEE, 2008, pp. 270–273.

[21] T. Liu, X. Ma, R. Zhao, H. Dong, and L. Jia, "Doppler shift estimation for high-speed railway scenario," in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, May 2016, pp. 1–5.

[22] Usrp. [Online]. Available: https://www.ettus.com/product/category/USRP-Embedded-Series/

[23] Rtl-sdr. [Online]. Available: http://www.rtl-sdr.com/

[24] T. S. Rappaport *et al.*, *Wireless communications: principles and practice.* prentice hall PTR New Jersey, 1996, vol. 2.

[25] J. Mitola and G. Q. Maguire, "Cognitive radio: making software radios more personal," *IEEE Personal Communications*, vol. 6, no. 4, pp. 13–18, Aug 1999.

[26] B. Barker, A. Agah, and A. M. Wyglinski, "Mission-oriented communications properties for software-defined radio configuration," *Cognitive Radio Networks*, p. 435, 2008.

[27] S. Haykin, "Cognitive radio: brain-empowered wireless communications," *IEEE journal on selected areas in communications*, vol. 23, no. 2, 2005.

[28] T. R. Newman, B. A. Barker, A. M. Wyglinski, A. Agah, J. B. Evans, and G. J. Minden, "Cognitive engine implementation for wireless multicarrier transceivers," *Wireless communications and mobile computing*, vol. 7, no. 9, pp. 1129–1142, 2007.

[29] T. R. Newman, R. Rajbanshi, A. M. Wyglinski, J. B. Evans, and G. J. Minden, "Population adaptation for genetic algorithm-based cognitive radios," *Mobile networks and applications*, vol. 13, no. 5, pp. 442–451, 2008.

[30] G. Yang, J. Wang, J. Luo, O. Y. Wen, H. Li, Q. Li, and S. Li, "Cooperative spectrum sensing in heterogeneous cognitive radio networks based on normalized energy detection," *IEEE Transactions on vehicular technology*, vol. 65, no. 3, pp. 1452–1463, 2016.

[31] Y. Zeng and Y.-C. Liang, "Eigenvalue-based spectrum sensing algorithms for cognitive radio," *IEEE transactions on communications*, vol. 57, no. 6, 2009.

[32] J. Mitola, "Software radios-survey, critical evaluation and future directions," in *[Proceedings] NTC-92: National Telesystems Conference*, May 1992, pp. 13/15–13/23.

[33] R. I. Lackey and D. W. Upmal, "Speakeasy: the military software radio," *IEEE Communications Magazine*, vol. 33, no. 5, pp. 56–61, May 1995.

[34] Xilinx spartan 3a. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/xa-spartan-3a.html

Appendix A

Heterogeneous Cooperative Spectrum Sensing Code

A.1 harddecisionpdroc.m

```
% Hard-Decision Combining Results For Sensor Nodes
clc;
close all;
clear all;
%% Parameter Initialization
N = 32;
k=4;%sensor nodes..
variance = 24.32e-9;
pfa = 0.05;
threshold = (qfuncinv(pfa)+sqrt(N)).*sqrt(N)*2*variance;
snrthreotical = -18:0.5:20;
snrlinear = 10.^(snrthreotical/10);
%% SNR values from USRP and RTL-SDR
snrpracticalavg =
    [-18.23,-14.22,-13.1,-12.22,-10.35,-8.25,-5.3,-2.1,0.13,...
```

```
    1.34,2.58,3.76,8.98,11.33,12.35,13.45,15.77];
snrlinearprac = 10.^(snrpracticalavg/10);
%% Computing Detection Probability and ROC Characteristics
pdprac = qfunc((threshold-2*N*variance.*(1+snrlinearprac))./...
    (sqrt(N.*(1+2*snrlinearprac))*(2*variance)));
pdpracor = 1-(1-pdprac).^4;
pdpracand = pdprac.^4;
tmp1 =  (1-pdprac).^2;
tmp2 = (1-pdprac);
pdpracmjr = (6*pdprac.^2).*tmp1+(4*pdprac.^3).*tmp2+pdprac.^4;

pfapracor = 1-(1-pfa).^4;
pfapracand = pfa.^4;
tmp1 =  (1-pfa).^2;
tmp2 = (1-pfa);
pfapracmjr = (6*pfa.^2).*tmp1+(4*pfa.^3).*tmp2+pfa.^4;

%% ROC Characteristics...
figure(1)
hold on;
grid on;
plot(pfapracor,pdpracor(:,5),'-->','LineWidth',2,'MarkerFaceColor
    ','auto');
plot(pfapracor,pdpracor(:,7),'-->','LineWidth',2,'MarkerFaceColor
    ','auto');
plot(pfapracand,pdpracand(:,5),'-d','LineWidth',2,'
    MarkerFaceColor','auto');
plot(pfapracand,pdpracand(:,7),'-d','LineWidth',2,'
    MarkerFaceColor','auto');
```

```
plot(pfapracmjr,pdpracmjr(:,5),'--','LineWidth',2,'
    MarkerFaceColor','auto');
plot(pfapracmjr,pdpracmjr(:,7),'--','LineWidth',2,'
    MarkerFaceColor','auto');
xlabel('Average Probability Of False Alarm');
ylabel('Average Probability of Detection');
title('ROC Characterisitcs of Hard Decision Combining');
hold off;
set(gca,'fontsize',30,'box','on','LineWidth',2,'GridLineStyle
    ','--','GridAlpha',0.7);
lgd = legend('OR SNR=-10.35dB','OR SNR=-5.3dB','AND SNR=-10.35dB
    ',...
     'AND SNR=-5.3dB','Majority SNR=-10.35dB','Majority SNR=-5.3dB
        ');
lgd.FontSize=20;

%% Probability of detection
figure(2)
hold on;
grid on;
plot(snrpracticalavg,pdpracor,'->','LineWidth',2);
plot(snrpracticalavg,pdpracand,'-<','LineWidth',2);
plot(snrpracticalavg,pdpracmjr,'-d','LineWidth',2);
xlabel('SNR_{avg} in (dB)');
ylabel('Average Probability of Detection');
title('P_{davg} vs SNR_{avg} for Hard Decision Combining');
hold off;
set(gca,'fontsize',30,'box','on','LineWidth',2,'GridLineStyle
    ','--','GridAlpha',0.7);
```

```
lgd = legend('OR Decision','AND Decision','Majority Rule Decision
    ');
lgd.FontSize=20;
axis([-18.23 15.77 0 1])
```

A.2 softharddecisionpd.m

```
% Soft Decision Combining for sensor nodes...
%% Initializing parameters..
close all;
clear all;
N = [100,200,300,400];% Different sum factor
k=4;% Number of Sensor Nodes
variance = [24.025e-9,23.695e-9,25.678e-9,0.0323e-9];
pfa = 0.05;%Probability of false alarm
for i=1:4
threshold(i) = (qfuncinv(pfa)+sqrt(N(i)))*sqrt(N(i))*2*variance(i
    );
end
% SNR Values from four sensor nodes
snrpractical =
    [-21.45,-18.23,-15.45,-13.3,-12.67,-9.35,-2.23,-4.32,2.98,6.95,13.57,21.7

                    -22.23,-20.22,-17.34,-15.32,-13.45,-11.27,-7.75,-5.67,2.53,

                    -25.34,-23.34,-21.67,-20.33,-16.76,-13.38,-8.56,-6.53,0.38,

                    -27.32,-24.97,-22.34,-22.23,-17.34,-14.32,-11.35,-7.85,-2.35

for i=1:4
```

```
    snrlinearprac(i,:) = 10.^(snrpractical(i,:)/10);
end
for i=1:4
pdprac(i,:) = qfunc((threshold(i)-2*N(i)*variance(i).*(1+
    snrlinearprac(i,:))))./...
    (sqrt(N(i)*(1+2*snrlinearprac(i,:)))*(2*variance(i))));
end
for i=1:12
snravg(i) = mean(snrpractical(:,i));
end
%% MNE based CS..
pdpracmne = 1-(1-pdprac(1,:)).*(1-pdprac(2,:)).*(1-pdprac(3,:))
    .*(1-pdprac(4,:));
pdpracand = mean(pdprac).^4;
pdpracm = mean(pdprac);
pdpracor = 1-(1-pdpracm).^4;
tmp1 =  (1-pdpracm).^(k-2);
tmp2 = (1-pdpracm);
pdpracmjr = (6*pdpracm.^(k-2)).*tmp1+(4*pdpracm.^(k-1)).*tmp2+
    pdpracm.^k;
figure(1)
hold on;
grid on;
plot(snravg,pdpracmne,'-<','LineWidth',2,'MarkerFaceColor','auto
    ');
axis([-20 10 0 1])
%% EGC based CS..
snrlinearmean = 10.^(snravg/10);
snrlinear = 10.^(snrpractical/10);
```

```
pfa = 0.01;
M= mean(N);
threshold = mean(threshold);
for i=1:length(snravg)
    num(i) = threshold-snrlinearmean(i);
    den(i) = (1/16)*((1+2*snrlinear(1,i))/N(1)+variance(1)+(1+2*
        snrlinear(2,i))/N(2)+variance(2)+(1+2*snrlinear(3,i))/N(3)
        +variance(3)+(1+2*snrlinear(4,i))/N(4)+variance(4));
    pdegc(i) = qfunc(num(i)/sqrt(den(i)));
end

%% Plotting the data..
plot(snravg,pdegc,'-d','LineWidth',2,'MarkerFaceColor','auto');
plot(snravg,pdpracand,'-s','LineWidth',2,'MarkerFaceColor','auto
    ');
plot(snravg,pdpracor,'-s','LineWidth',2,'MarkerFaceColor','auto')
    ;
plot(snravg,pdpracmjr,'->','LineWidth',2,'MarkerFaceColor','auto
    ');
title('P_{davg} vs SNR_{avg} for Soft and Hard Decision Combining
    ');
xlabel('SNR_{avg} in (dB)');
ylabel('Average Probability of Detection');
set(gca,'fontsize',30,'box','on','LineWidth',2,'GridLineStyle
    ','--','GridAlpha',0.7);
legend('MNE Combining','EGC Combining','AND Rule','OR Rule','
    Majority Rule');
```

A.3 spectrumsenseusrp.py

```
#!/usr/bin/env python
#
# Copyright 2005,2007,2011 Free Software Foundation, Inc.
#
# This file is part of GNU Radio
#
# GNU Radio is free software; you can redistribute it and/or
    modify
# it under the terms of the GNU General Public License as
    published by
# the Free Software Foundation; either version 3, or (at your
    option)
# any later version.
#
# GNU Radio is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public
    License
# along with GNU Radio; see the file COPYING.  If not, write to
# the Free Software Foundation, Inc., 51 Franklin Street,
# Boston, MA 02110-1301, USA.
#

from gnuradio import gr, eng_notation
from gnuradio import blocks
from gnuradio import audio
```

```
from gnuradio import filter
from gnuradio import fft
from gnuradio import uhd
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import sys
import math
import struct
import threading
from datetime import datetime
import time
from gnuradio.wxgui import stdgui2, fftsink2, form
import wx

sys.stderr.write("Warning: this may have issues on some machines+
   Python version combinations to seg fault due to the callback
   in bin_statitics.\n\n")

class ThreadClass(threading.Thread):
    def run(self):
        return

class tune(gr.feval_dd):
    """
    This class allows C++ code to callback into python.
    """
    def __init__(self, tb):
        gr.feval_dd.__init__(self)
        self.tb = tb
```

```
    def eval(self, ignore):
        """
        This method is called from blocks.bin_statistics_f when
            it wants
        to change the center frequency.  This method tunes the
            front
        end to the new center frequency, and returns the new
            frequency
        as its result.
        """

        try:
            new_freq = self.tb.set_next_freq()
            while(self.tb.msgq.full_p()):
                time.sleep(0.1)
            return new_freq

        except Exception, e:
            print "tune: Exception: ", e


class parse_msg(object):
    def __init__(self, msg):
        self.center_freq = msg.arg1()
        self.vlen = int(msg.arg2())
        assert(msg.length() == self.vlen * gr.sizeof_float)
        t = msg.to_string()
        self.raw_data = t
```

```
        self.data = struct.unpack('%df' % (self.vlen,), t)


class my_top_block(gr.top_block):

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

        usage = "usage: %prog [options] min_freq max_freq"
        parser = OptionParser(option_class=eng_option, usage=
            usage)
        parser.add_option("-a", "--args", type="string", default
            ="",
                          help="UHD device device address args [
                              default=%default]")
        parser.add_option("", "--spec", type="string", default=
            None,
                          help="Subdevice of UHD device where
                              appropriate")
        parser.add_option("-A", "--antenna", type="string",
            default=None,
                          help="select Rx Antenna where
                              appropriate")
        parser.add_option("-s", "--samp-rate", type="eng_float",
            default=1e6,
                          help="set sample rate [default=%default
                              ]")
        parser.add_option("-g", "--gain", type="eng_float",
            default=None,
```

```
                    help="set gain in dB (default is
                        midpoint)")
parser.add_option("", "--tune-delay", type="eng_float",
                    default=0.25, metavar="SECS",
                    help="time to delay (in seconds) after
                        changing frequency [default=%default
                        ]")
parser.add_option("", "--dwell-delay", type="eng_float",
                    default=0.25, metavar="SECS",
                    help="time to dwell (in seconds) at a
                        given frequency [default=%default]")
parser.add_option("-b", "--channel-bandwidth", type="
    eng_float",
                    default=6.25e3, metavar="Hz",
                    help="channel bandwidth of fft bins in
                        Hz [default=%default]")
parser.add_option("-l", "--lo-offset", type="eng_float",
                    default=0, metavar="Hz",
                    help="lo_offset in Hz [default=%default
                        ]")
parser.add_option("-q", "--squelch-threshold", type="
    eng_float",
                    default=None, metavar="dB",
                    help="squelch threshold in dB [default
                        =%default]")
parser.add_option("-F", "--fft-size", type="int", default
    =None,
                    help="specify number of FFT bins [
                        default=samp_rate/channel_bw]")
```

```
parser.add_option("", "--real-time", action="store_true",
    default=False,
                  help="Attempt to enable real-time
                      scheduling")

(options, args) = parser.parse_args()
if len(args) != 2:
    parser.print_help()
    sys.exit(1)

self.channel_bandwidth = options.channel_bandwidth

self.min_freq = eng_notation.str_to_num(args[0])
self.max_freq = eng_notation.str_to_num(args[1])

if self.min_freq > self.max_freq:
    # swap them
    self.min_freq, self.max_freq = self.max_freq, self.
        min_freq

if not options.real_time:
    realtime = False
else:
    # Attempt to enable realtime scheduling
    r = gr.enable_realtime_scheduling()
    if r == gr.RT_OK:
        realtime = True
    else:
        realtime = False
```

```
            print "Note: failed to enable realtime scheduling
                "

        # build graph
        self.u = uhd.usrp_source(device_addr=options.args,
                                 stream_args=uhd.stream_args('
                                     fc32'))

        # Set the subdevice spec
        if(options.spec):
            self.u.set_subdev_spec(options.spec, 0)

        # Set the antenna
        if(options.antenna):
            self.u.set_antenna(options.antenna, 0)

        self.u.set_samp_rate(options.samp_rate)
        self.usrp_rate = usrp_rate = self.u.get_samp_rate()

        self.lo_offset = options.lo_offset

        if options.fft_size is None:
            self.fft_size = int(self.usrp_rate/self.
                channel_bandwidth)
        else:
            self.fft_size = options.fft_size

        self.squelch_threshold = options.squelch_threshold
```

```
s2v = blocks.stream_to_vector(gr.sizeof_gr_complex, self.
    fft_size)

mywindow = filter.window.blackmanharris(self.fft_size)
ffter = fft.fft_vcc(self.fft_size, True, mywindow, True)
power = 0
for tap in mywindow:
     power += tap*tap
c2mag = blocks.complex_to_mag_squared(self.fft_size)
self.freq_step = self.nearest_freq((0.75 * self.usrp_rate
    ), self.channel_bandwidth)
self.min_center_freq = self.min_freq + (self.freq_step/2)
nsteps = math.ceil((self.max_freq - self.min_freq) / self
    .freq_step)
self.max_center_freq = self.min_center_freq + (nsteps *
    self.freq_step)
self.next_freq = self.min_center_freq
tune_delay  = max(0, int(round(options.tune_delay *
    usrp_rate / self.fft_size)))  # in fft_frames
dwell_delay = max(1, int(round(options.dwell_delay *
    usrp_rate / self.fft_size))) # in fft_frames
self.msgq = gr.msg_queue(1)
self._tune_callback = tune(self)        # hang on to this
     to keep it from being GC'd
stats = blocks.bin_statistics_f(self.fft_size, self.msgq,
                                self._tune_callback,
                                   tune_delay,
                                dwell_delay)
self.connect(self.u, s2v, ffter, c2mag, stats)
```

```
        if options.gain is None:

            g = self.u.get_gain_range()
            options.gain = float(g.start()+g.stop())/2.0

        self.set_gain(options.gain)
        print "gain =", options.gain

    def set_next_freq(self):
        target_freq = self.next_freq
        self.next_freq = self.next_freq + self.freq_step
        if self.next_freq >= self.max_center_freq:
            self.next_freq = self.min_center_freq

        if not self.set_freq(target_freq):
            print "Failed to set frequency to", target_freq
            sys.exit(1)

        return target_freq


    def set_freq(self, target_freq):
        """
        Set the center frequency we're interested in.

        Args:
            target_freq: frequency in Hz
        @rypte: bool
```

```
        """

        r = self.u.set_center_freq(uhd.tune_request(target_freq,
            rf_freq=(target_freq + self.lo_offset),rf_freq_policy=
            uhd.tune_request.POLICY_MANUAL))
        if r:
            return True

        return False

    def set_gain(self, gain):
        self.u.set_gain(gain)

    def nearest_freq(self, freq, channel_bandwidth):
        freq = round(freq / channel_bandwidth, 0) *
            channel_bandwidth
        return freq

def main_loop(tb):

    def bin_freq(i_bin, center_freq):
        freq = center_freq - (tb.usrp_rate / 2) + (tb.
            channel_bandwidth * i_bin)
        return freq

    bin_start = int(tb.fft_size * ((1 - 0.25) / 2))
    bin_stop = int(tb.fft_size - bin_start)
    fid = open("./usrp.dat","wb")
    while 1:
```

```
            m = parse_msg(tb.msgq.delete_head())
            for i_bin in range(bin_start, bin_stop):
                center_freq = m.center_freq
                freq = bin_freq(i_bin, center_freq)
                power_db = 10*math.log10(m.data[i_bin]/tb.usrp_rate)
                signal = m.data[i_bin]/(tb.usrp_rate)

                if (power_db > tb.squelch_threshold) and (freq >= tb.
                    min_freq) and (freq <= tb.max_freq):
                    print freq,signal,power_db
                    fid.write(struct.pack('<f',signal))
        fid.close()#closing the file

if __name__ == '__main__':
    t = ThreadClass()
    t.start()

    tb = my_top_block()
    try:
        tb.start()
        main_loop(tb)

    except KeyboardInterrupt:
        pass
```

A.4 gnuradiortlsdrsense.py

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
##################################################
```

```
# GNU Radio Python Flow Graph
# Title: DTv Spectrum Sensing
# Author: Gill
# Description: Frequency Sweep for UHF White Spaces
# Generated: Fri Mar 10 14:30:20 2017
##################################################

if __name__ == '__main__':
    import ctypes
    import sys
    if sys.platform.startswith('linux'):
        try:
            x11 = ctypes.cdll.LoadLibrary('libX11.so')
            x11.XInitThreads()
        except:
            print "Warning: failed to XInitThreads()"

from PyQt4 import Qt
from gnuradio import blocks
from gnuradio import eng_notation
from gnuradio import fft
from gnuradio import gr
from gnuradio import qtgui
from gnuradio.eng_option import eng_option
from gnuradio.fft import window
from gnuradio.filter import firdes
from optparse import OptionParser
import numpy as np
import osmosdr
```

```
import sip
import sys
import time


class spectrum_sensing(gr.top_block, Qt.QWidget):

    def __init__(self):
        gr.top_block.__init__(self, "DTv Spectrum Sensing")
        Qt.QWidget.__init__(self)
        self.setWindowTitle("DTv Spectrum Sensing")
        try:
            self.setWindowIcon(Qt.QIcon.fromTheme('gnuradio-grc')
                )
        except:
            pass
        self.top_scroll_layout = Qt.QVBoxLayout()
        self.setLayout(self.top_scroll_layout)
        self.top_scroll = Qt.QScrollArea()
        self.top_scroll.setFrameStyle(Qt.QFrame.NoFrame)
        self.top_scroll_layout.addWidget(self.top_scroll)
        self.top_scroll.setWidgetResizable(True)
        self.top_widget = Qt.QWidget()
        self.top_scroll.setWidget(self.top_widget)
        self.top_layout = Qt.QVBoxLayout(self.top_widget)
        self.top_grid_layout = Qt.QGridLayout()
        self.top_layout.addLayout(self.top_grid_layout)

        self.settings = Qt.QSettings("GNU Radio", "
```

```
    spectrum_sensing")
self.restoreGeometry(self.settings.value("geometry").
    toByteArray())

##################################################
# Variables
##################################################
self.samp_rate = samp_rate = int(2e6)
self.freq = freq = 450e6
self.N = N = 1000

##################################################
# Blocks
##################################################
self.rtlsdr_source_0 = osmosdr.source( args="numchan=" +
    str(1) + " " + '' )
self.rtlsdr_source_0.set_time_source('external', 0)
self.rtlsdr_source_0.set_sample_rate(samp_rate)
self.rtlsdr_source_0.set_center_freq(freq, 0)
self.rtlsdr_source_0.set_freq_corr(0, 0)
self.rtlsdr_source_0.set_dc_offset_mode(2, 0)
self.rtlsdr_source_0.set_iq_balance_mode(0, 0)
self.rtlsdr_source_0.set_gain_mode(True, 0)
self.rtlsdr_source_0.set_gain(15, 0)
self.rtlsdr_source_0.set_if_gain(15, 0)
self.rtlsdr_source_0.set_bb_gain(15, 0)
self.rtlsdr_source_0.set_antenna('', 0)
self.rtlsdr_source_0.set_bandwidth(0, 0)
```

```
self.qtgui_freq_sink_x_0 = qtgui.freq_sink_c(
        1024, #size
        firdes.WIN_BLACKMAN_hARRIS, #wintype
        0, #fc
        samp_rate, #bw
        "Recieved Signal", #name
        1 #number of inputs
)
self.qtgui_freq_sink_x_0.set_update_time(0.10)
self.qtgui_freq_sink_x_0.set_y_axis(-120, 0)
self.qtgui_freq_sink_x_0.set_y_label('Relative Gain', 'dB
    ')
self.qtgui_freq_sink_x_0.set_trigger_mode(qtgui.
    TRIG_MODE_FREE, 0.0, 0, "")
self.qtgui_freq_sink_x_0.enable_autoscale(True)
self.qtgui_freq_sink_x_0.enable_grid(True)
self.qtgui_freq_sink_x_0.set_fft_average(1.0)
self.qtgui_freq_sink_x_0.enable_axis_labels(True)
self.qtgui_freq_sink_x_0.enable_control_panel(False)

if not True:
  self.qtgui_freq_sink_x_0.disable_legend()

if "complex" == "float" or "complex" == "msg_float":
  self.qtgui_freq_sink_x_0.set_plot_pos_half(not True)

labels = ['', '', '', '', '',
          '', '', '', '', '']
widths = [2, 1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1]
colors = ["blue", "red", "green", "black", "cyan",
            "magenta", "yellow", "dark red", "dark green",
               "dark blue"]
alphas = [1.0, 1.0, 1.0, 1.0, 1.0,
            1.0, 1.0, 1.0, 1.0, 1.0]
for i in xrange(1):
    if len(labels[i]) == 0:
        self.qtgui_freq_sink_x_0.set_line_label(i, "Data
            {0}".format(i))
    else:
        self.qtgui_freq_sink_x_0.set_line_label(i, labels
            [i])
    self.qtgui_freq_sink_x_0.set_line_width(i, widths[i])
    self.qtgui_freq_sink_x_0.set_line_color(i, colors[i])
    self.qtgui_freq_sink_x_0.set_line_alpha(i, alphas[i])

self._qtgui_freq_sink_x_0_win = sip.wrapinstance(self.
    qtgui_freq_sink_x_0.pyqwidget(), Qt.QWidget)
self.top_layout.addWidget(self._qtgui_freq_sink_x_0_win)
self.fft_vxx_0 = fft.fft_vcc(1024, True, (window.
    blackmanharris(1024)), True, 1)
self.blocks_vector_to_stream_0 = blocks.vector_to_stream(
    gr.sizeof_float*1, 1024)
self.blocks_stream_to_vector_0 = blocks.stream_to_vector(
    gr.sizeof_gr_complex*1, 1024)
self.blocks_moving_average_xx_0 = blocks.
    moving_average_ff(N, 1, 4000)
self.blocks_file_sink_2_0 = blocks.file_sink(gr.
```

```
            sizeof_float*1, '/home/gill/Desktop/ms-thesis/gr-
                spectrumsensing/grc/rtl-sdr_sensing/Results/snr_check.
                dat', False)
        self.blocks_file_sink_2_0.set_unbuffered(False)
        self.blocks_complex_to_mag_squared_0 = blocks.
            complex_to_mag_squared(1024)

        ##################################################
        # Connections
        ##################################################
        self.connect((self.blocks_complex_to_mag_squared_0, 0), (
            self.blocks_vector_to_stream_0, 0))
        self.connect((self.blocks_moving_average_xx_0, 0), (self.
            blocks_file_sink_2_0, 0))
        self.connect((self.blocks_stream_to_vector_0, 0), (self.
            fft_vxx_0, 0))
        self.connect((self.blocks_vector_to_stream_0, 0), (self.
            blocks_moving_average_xx_0, 0))
        self.connect((self.fft_vxx_0, 0), (self.
            blocks_complex_to_mag_squared_0, 0))
        self.connect((self.rtlsdr_source_0, 0), (self.
            blocks_stream_to_vector_0, 0))
        self.connect((self.rtlsdr_source_0, 0), (self.
            qtgui_freq_sink_x_0, 0))

    def closeEvent(self, event):
        self.settings = Qt.QSettings("GNU Radio", "
            spectrum_sensing")
        self.settings.setValue("geometry", self.saveGeometry())
```

```
        event.accept()

    def get_samp_rate(self):
        return self.samp_rate

    def set_samp_rate(self, samp_rate):
        self.samp_rate = samp_rate
        self.rtlsdr_source_0.set_sample_rate(self.samp_rate)
        self.qtgui_freq_sink_x_0.set_frequency_range(0, self.
            samp_rate)

    def get_freq(self):
        return self.freq

    def set_freq(self, freq):
        self.freq = freq
        self.rtlsdr_source_0.set_center_freq(self.freq, 0)

    def get_N(self):
        return self.N

    def set_N(self, N):
        self.N = N
        self.blocks_moving_average_xx_0.set_length_and_scale(self
            .N, 1)


def main(top_block_cls=spectrum_sensing, options=None):
```

```
    from distutils.version import StrictVersion
    if StrictVersion(Qt.qVersion()) >= StrictVersion("4.5.0"):
        style = gr.prefs().get_string('qtgui', 'style', 'raster')
        Qt.QApplication.setGraphicsSystem(style)
    qapp = Qt.QApplication(sys.argv)

    tb = top_block_cls()
    tb.start()
    tb.show()

    def quitting():
        tb.stop()
        tb.wait()
    qapp.connect(qapp, Qt.SIGNAL("aboutToQuit()"), quitting)
    qapp.exec_()


if __name__ == '__main__':
    main()
```

Appendix B

LTE-R Analysis Code

B.1 kfactordist.m

```
% Calculating K-factor for the tunnel environment for HST
clear all;
close all;
clc;
%% Creating a doppler profile for high speed raiway scenario..
Ds = 30;% Initial Distance between tx and rx times 2..
Dmin = 2;% Distance between raiway tracks and leaky feeder cables
    ...
Kf = [];
fc = 3e9;%center frequency..
c = 3e8;
v = 138.9;%300;
t = linspace(0,(2*Ds)/v(1),100);
fd = (v*fc)/3e8;%maximum doppler frequency...
costheta = zeros(size(t));%angle between BS and MS
d1 = [];
for i=1:length(t)
```

```
    d1(i) = sqrt(2^2+(Ds/2-v(1)*t(i))^2);%distance between tx and
        rx..
    if t(i) >=0 && t(i)<= (Ds/v)
        costheta(i) = ((Ds/2)-v*t(i))./sqrt(Dmin^2+(Ds/2-v*t(i))
            ^2);

    elseif t(i) > (Ds/v) && t(i)<=(2*Ds)/v
        costheta(i) = (-1.5*Ds+v*t(i))./sqrt(Dmin^2+(-1.5*Ds+v*t(
            i))^2);
    end
end
fs  = fd*costheta;
thetadeg = acosd(costheta);
fc_wds = fc-fs;
lambda = c./fc_wds;
Cin = 5-((0.1*1.8e10)./fc_wds)*1j;
C = Cin;
gammanum = C.*sind(thetadeg)-sqrt(Cin-(cosd(thetadeg)).^2);
gammaden = C.*sind(thetadeg)+sqrt(Cin-(cosd(thetadeg)).^2);
gamma = gammanum./gammaden;
ht = 6.1;%height of feeder cable
hr = 4.2;%height of the train
var1 = sqrt(d1.^2+(ht+hr)^2);
var2 = sqrt(d1.^2+(ht-hr)^2);
phase = ((((2*pi)./lambda).*(var1-var2))*180)/pi;
gammad = atan2d(imag(gamma),real(gamma));
phasegamma = abs(cosd(gammad-phase));
K = abs(gamma).^2+2*abs(gamma).*phasegamma;
Kf = 10*log10(1./K);
```

B.2 bercalculation.m

```
% Demonstration of Eb/N0 Vs SER for M-QAM modulation scheme
clc;
load Kf;
load t;
%----------Input Fields------------------------
%% QPSk
bitsperframe=1e3; %Number of input symbols
EbN0dB = [linspace(0,20,50) fliplr(linspace(0,20,50))]; %Define
    EbN0dB range for simulation
M=4; %for QPSk modulation.
hMod = comm.RectangularQAMModulator('ModulationOrder',M);
const = step(hMod,(0:3)');
%-----------------------------------------------
refArray =1/sqrt(2)*const';
k=log2(M);
totPower=15; %Total power of LOS path & scattered paths

EsN0dB = EbN0dB + 10*log10(k);
biterrsim = zeros(size(EsN0dB));
%---Generating a uniformly distributed random numbers in the set
    [0,1,2,..,M-1]
data=ceil(M.*rand(bitsperframe,1))-1;
s=refArray(data+1); %QPSK Constellation mapping with Gray coding
%--- Reference Constellation for demodulation and Error rate
    computation---
refI = real(refArray);
refQ = imag(refArray);
%---Place holder for Symbol Error values for each Es/N0 for
```

```
    particular M value---
index=1;
u=1;
% Kf = 4.9;
K = 10.^(Kf/10);
for x=EsN0dB
    sn=sqrt(K(u)/(K(u)+1)*totPower); %Non-Centrality Parameter
    sigma=totPower/sqrt(2*(K(u)+1));
    h=((sigma*randn(1,bitsperframe)+sn)+1i*(randn(1,bitsperframe)
        *sigma+0));
    numerr = 0;
    numBits = 0;
    while numerr < 100 && numBits < 1e7
        %---------------------------------------------
        %Channel Noise for various Es/N0
        %---------------------------------------------
        %Adding noise with variance according to the required Es/
            N0
        noiseVariance = 1/(10.^(x/10));%Standard deviation for
            AWGN Noise
        noiseSigma = sqrt(noiseVariance/2);
        %Creating a complex noise for adding with M-QAM modulated
             signal
        %Noise is complex since M-QAM is in complex
            representation
        noise = noiseSigma*(randn(size(s))+1i*randn(size(s)));
        received = s.*h + noise;
        %--------------I-Q Branching----------------
        received = received./h;
```

```
        r_i = real(received);
        r_q = imag(received);
        %---Decision Maker-Compute (r_i-s_i)^2+(r_q-s_q)^2 and
            choose the smallest
        r_i_repmat = repmat(r_i,M,1);
        r_q_repmat = repmat(r_q,M,1);
        distance = zeros(M,bitsperframe); %place holder for
            distance metric
        minDistIndex=zeros(bitsperframe,1);
            for j=1:bitsperframe
            %---Distance computation - (r_i-s_i)^2+(r_q-s_q)^2
                ----------------
            distance(:,j) = (r_i_repmat(:,j)-refI').^2+(
                r_q_repmat(:,j)-refQ').^2;
            %---capture the index in the array where the minimum
                distance occurs
            [dummy,minDistIndex(j)]=min(distance(:,j));
            end
        y = minDistIndex - 1;
        %----------------Symbol Error Rate Calculation
            ------------------------------------
        dataCap = y;
        numerr = sum(dataCap~=data)+numerr;
        numBits = numBits+bitsperframe;
        disp(numerr);
    end
symErrSimulatedqpsk(1,index) = numerr/numBits;
biterrsim(1,index) = symErrSimulatedqpsk(1,index)/k;
index=index+1;
```

```
% u=u+1;
end

%% 16 QAM
bitsperframe=1e3; %Number of input symbols
EbN0dB = [linspace(0,10,50) fliplr(linspace(0,10,50))]; %Define
    EbN0dB range for simulation
M=16; %for QPSk modulation.
hMod = comm.RectangularQAMModulator('ModulationOrder',M);
const = step(hMod,(0:M-1)');
%-------------------------------------------------
refArray =1/sqrt(10)*const';
k=log2(M);
totPower=10; %Total power of LOS path & scattered paths


EsN0dB = EbN0dB + 10*log10(k);
biterrsim = zeros(size(EsN0dB));
%---Generating a uniformly distributed random numbers in the set
    [0,1,2,..,M-1]
data=ceil(M.*rand(bitsperframe,1))-1;
s=refArray(data+1); %QPSK Constellation mapping with Gray coding
%--- Reference Constellation for demodulation and Error rate
    computation---
refI = real(refArray);
refQ = imag(refArray);
%---Place holder for Symbol Error values for each Es/N0 for
    particular M value---
index=1;
```

```
u=1;
K = 10.^(Kf/10);
for x=EsN0dB
    numerr = 0;
    numBits = 0;
    while numerr < 100 && numBits < 1e7
        sn=sqrt(K(u)/(K(u)+1)*totPower); %Non-Centrality
            Parameter
        sigma=totPower/sqrt(2*(K(u)+1));
        h=((sigma*randn(1,bitsperframe)+sn)+1i*(randn(1,
            bitsperframe)*sigma+0));
        %-----------------------------------------------
        %Channel Noise for various Es/N0
        %-----------------------------------------------
        %Adding noise with variance according to the required Es/
            N0
        noiseVariance = 1/(10.^(x/10));%Standard deviation for
            AWGN Noise
        noiseSigma = sqrt(noiseVariance/2);
        %Creating a complex noise for adding with M-QAM modulated
             signal
        %Noise is complex since M-QAM is in complex
            representation
        noise = noiseSigma*(randn(size(s))+1i*randn(size(s)));
        received = s.*h + noise;
        %--------------I-Q Branching-----------------
        received = received./h;
        r_i = real(received);
        r_q = imag(received);
```

```
        %——Decision Maker–Compute (r_i-s_i)^2+(r_q-s_q)^2 and
            choose the smallest
        r_i_repmat = repmat(r_i,M,1);
        r_q_repmat = repmat(r_q,M,1);
        distance = zeros(M, bitsperframe); %place holder for
            distance metric
        minDistIndex=zeros(bitsperframe,1);
             for j=1:bitsperframe
             %——Distance computation - (r_i-s_i)^2+(r_q-s_q)^2
                 ————————————
             distance(:,j) = (r_i_repmat(:,j)-refI').^2+(
                 r_q_repmat(:,j)-refQ').^2;
             %——capture the index in the array where the minimum
                 distance occurs
             [dummy, minDistIndex(j)]=min(distance(:,j));
             end
        y = minDistIndex - 1;
        %————————————Symbol Error Rate Calculation
            ————————————————————————————————
        dataCap = y;
        numerr = sum(dataCap~=data)+numerr;
        numBits = numBits+bitsperframe;
        disp(numerr);
    end
symErrSimulatedqam(1,index) = numerr/numBits;
biterrsim(1,index) = symErrSimulatedqam(1,index)/k;
index=index+1;
u=u+1;
end
```

```
%% 64 QAM Modulation...
bitsperframe=1e3; %Number of input symbols
EbN0dB = [linspace(0,10,50) fliplr(linspace(0,10,50))]; %Define
    EbN0dB range for simulation
M=64; %for QPSk modulation.
hMod = comm.RectangularQAMModulator('ModulationOrder',M);
const = step(hMod,(0:M-1)');
%-------------------------------------------------
refArray =1/sqrt(42)*const';
k=log2(M);
totPower=10; %Total power of LOS path & scattered paths
EsN0dB = EbN0dB + 10*log10(k);
biterrsim = zeros(size(EsN0dB));
%---Generating a uniformly distributed random numbers in the set
    [0,1,2,..,M-1]
data=ceil(M.*rand(bitsperframe,1))-1;
s=refArray(data+1); %QPSK Constellation mapping with Gray coding
%--- Reference Constellation for demodulation and Error rate
    computation---
refI = real(refArray);
refQ = imag(refArray);
%---Place holder for Symbol Error values for each Es/N0 for
    particular M value---
index=1;
u=1;
K = 10.^(Kf/10);
for x=EsN0dB
      sn=sqrt(K(u)/(K(u)+1)*totPower); %Non-Centrality Parameter
```

```
 sigma=totPower/sqrt(2*(K(u)+1));
 h=((sigma*randn(1,bitsperframe)+sn)+1i*(randn(1,bitsperframe
     )*sigma+0));
 numerr = 0;
 numBits = 0;
while numerr < 100 && numBits < 1e7
    %-------------------------------------------------
    %Channel Noise for various Es/N0
    %-------------------------------------------------
    %Adding noise with variance according to the required Es/
        N0
    noiseVariance = 1/(10.^(x/10));%Standard deviation for
       AWGN Noise
    noiseSigma = sqrt(noiseVariance/2);
    %Creating a complex noise for adding with M-QAM modulated
         signal
    %Noise is complex since M-QAM is in complex
        representation
    noise = noiseSigma*(randn(size(s))+1i*randn(size(s)));
    received = s.*h + noise;
    %--------------I-Q Branching-----------------
    received = received./h;
    r_i = real(received);
    r_q = imag(received);
    %---Decision Maker-Compute (r_i-s_i)^2+(r_q-s_q)^2 and
        choose the smallest
    r_i_repmat = repmat(r_i,M,1);
    r_q_repmat = repmat(r_q,M,1);
    distance = zeros(M,bitsperframe); %place holder for
```

```
            distance metric
        minDistIndex=zeros(bitsperframe,1);
            for j=1:bitsperframe
            %---Distance computation - (r_i-s_i)^2+(r_q-s_q)^2
                ---------------
            distance(:,j) = (r_i_repmat(:,j)-refI').^2+(
                r_q_repmat(:,j)-refQ').^2;
            %---capture the index in the array where the minimum
                distance occurs
            [dummy, minDistIndex(j)]=min(distance(:,j));
            end
        y = minDistIndex - 1;
        %---------------Symbol Error Rate Calculation
            ---------------------------------
        dataCap = y;
        numerr = sum(dataCap~=data)+numerr;
        numBits = numBits+bitsperframe;
        disp(numerr);
    end
symErrSimulatedqam64(1,index) = numerr/numBits;
biterrsim(1,index) = symErrSimulatedqam64(1,index)/k;
index=index+1;
u=u+1;
end

%%
fig = figure;
semilogy(t*1e3,symErrSimulatedqpsk(1,:),'-d','LineWidth',2);
hold on;
```

```
grid on;
semilogy(t*1e3,symErrSimulatedqam(1,:),'-d','LineWidth',2);
semilogy(t*1e3,symErrSimulatedqam64(1,:),'-d','LineWidth',2);
xlabel('Time (ms)');
ylabel('Bit Error Rate (Pb)');
title(['BER For OFDM Under Rician Fading Environment Inside
    Tunnel']);
set(gca,'fontsize',30,'box','on','LineWidth',2,'GridLineStyle
    ','--','GridAlpha',0.7);
axis([0 max(t)*1e3 10e-7 0])
lgd = legend('QPSK','16QAM','64QAM');
lgd.FontSize=20;
```