April 2012

Aerial Networking: Creating a Resilient Wireless Network for Multiple Unmanned Aerial Vehicles

Anastasios Vafeiadis
Worcester Polytechnic Institute

Archibald Alexander Owen
Worcester Polytechnic Institute

Aerial Networking:
Creating a Resilient Wireless Network for Multiple Unmanned Aerial Vehicles

A Major Qualifying Project

Submitted to the Faculty of

Worcester Polytechnic Institute

In Partial Fulfillment of the Requirements for the Degree of Bachelor of Science

Prepared by:
Archibald Owen
Anastasios Vafeiadis

Advisors:
Professor Alexander Wyglinski
Professor Taskin Padir

Sponsor:
The MathWorks Inc.

April 26, 2012

AW1-MQP-WND3

Unmanned Aerial Vehicles, Software-Defined Radio, Ad Hoc Wireless Networks

Abstract

In this report, we present the foundations for a wireless communications system between several Unmanned Aerial Vehicles (UAVs) that will facilitate their mission of Search and Rescue (SAR). This goal was chosen because the need for capable SAR crews is an ever-present requirement in the modern world. To accomplish their mission, these crews need knowledge of a disaster area or the locations of missing people. UAVs possess the ability to acquire this knowledge safely from above the search area and relay it to a user. In order to increase efficiency, a group of UAVs equipped with cameras (drones) can be used and relay their data through a central UAV called a "mothership." Such a system increases the search area while minimizing the risk to the rescuers. It could also be adapted to many other functions, such as law enforcement or research. For this project, we propose creating the initial stages of a mobile ad hoc network that enables the separate UAVs to interact, coordinate SAR efforts, and transmit information to the user. Our specific goals are to demonstrate the feasibility of such a network in the laboratory, simulating transmission between different nodes while accounting for possible errors and interference. We also define the groundwork for the physical implementation of the system, including the assembly of a motherboard and Wi-Fi transmitters that will perform the eventual communication between the mothership, drones, and user. The long term vision for our wireless network will be one that can handle many of the problems encountered on multiple mobile platforms moving in various formations. These problems include implementing a medium access control (MAC) protocol, the ability to add drones in real-time or account for ones that go out of range, and manage spectrum allocation for the different users. Thus, our work is the first step towards implementing a fully-functional UAV ad hoc network that utilizes the flexibility of software defined radio to improve efficiency and safety while performing a desired mission.

Executive Summary

The need for capable SAR crews is an ever-present requirement in the modern world. From large-scale events like earthquakes to small-scale events like missing hikers, there is always the possibility of a disaster or emergency that will require search and rescue efforts to help save lives and recover from the event. The number of catastrophes – both man-made and natural – where the efforts of first responders were stunted by physical obstacles is as bleak as it is long. Hurricanes, bombings, fires, natural gas explosions, flooding, or large-scale construction failure; the list is endless, and all can pose unique and cumbersome restrictions on the progress of the attending personnel.

While techniques for search and rescue have improved over the years, new technology in many instances could have aided the recovery efforts of the affected society. Among those the most promising of these is the use of Unmanned Aerial Vehicles (UAVs). UAVs have the potential to expand the rescuers' ability to assess the situation and search for individuals. Like helicopters or planes, UAVs provide aerial reconnaissance that can help create an overall picture of the situation. However, UAVs are cheaper than manned aircraft, they can stay airborne for hours, and they often have high resolution cameras, making them useful tools. The application of UAVs to SAR is thus of vital importance to saving more lives.

The goal of this project was to develop resilient wireless communications between a network of UAVs as seen in Figure 1. Using the concepts from software defined radio, the problem of varying link quality will be resolved by devising a mechanism that would continuously operate onboard the wireless node, sensing the radio environment and making decisions that would allow the wireless node to maintain connectivity with the rest of the network at all costs. Knowing the approximate trajectory and speed of each wireless node will allow for the network to be prepared when the wireless node is finally out of range and loses connectivity with the rest of the network.

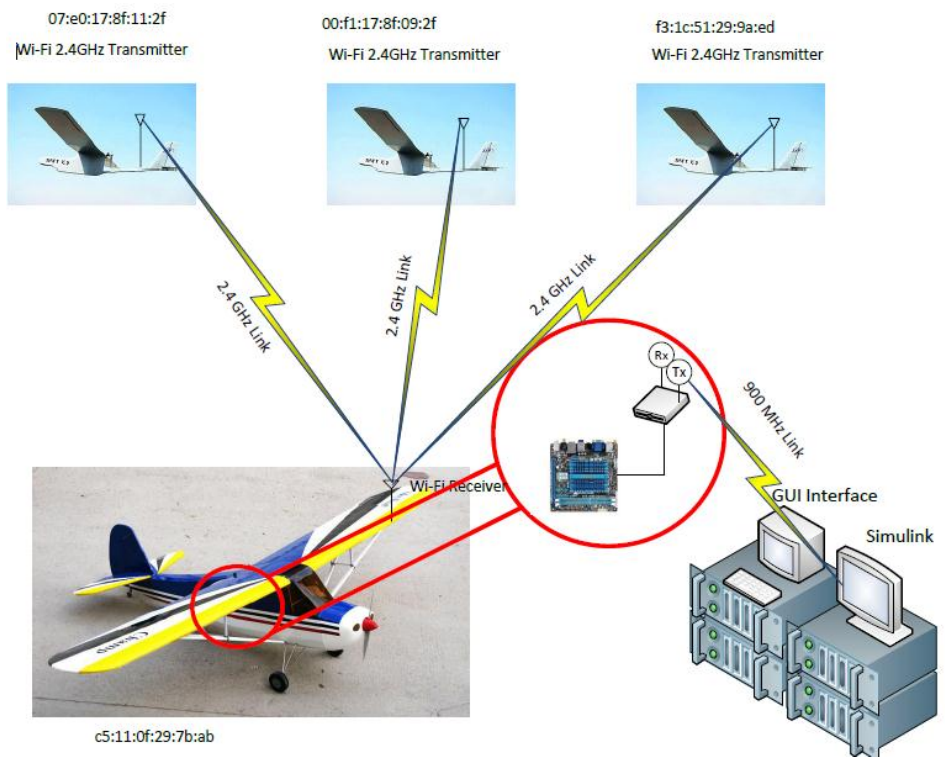
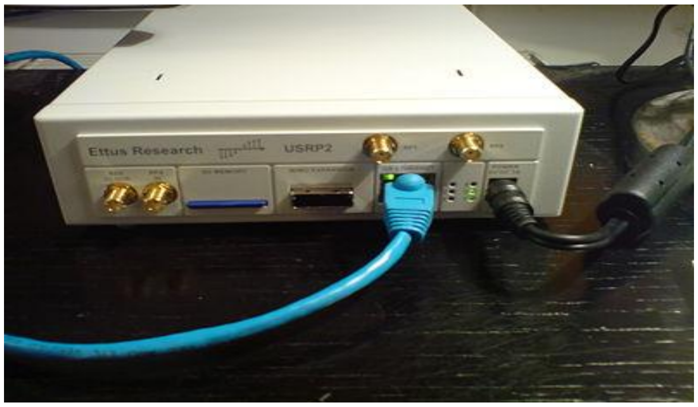


Figure 1: Proposed UAV Network Architecture

Software defined radio (SDR) has revolutionized the communications industry by providing unprecedented levels of flexibility. SDRs implement nearly all radio functionality in programmable components such as field programmable gate arrays (FPGA) and computers, allowing them to be reconfigured without any changes to the radio hardware. This flexibility makes SDR popular for applications such as rapid prototyping, scientific experimentation, limited-production devices and cognitive radio. The software defined radio platform that was used for this project was the Universal Software Radio Peripheral 2 (USRP2).

In order to achieve our goal, we studied and evaluated our proposed implementations using computer simulations. We used SDRs under lab conditions with the help of programming tools such as MATLAB/Simulink as seen in Figure 2. Since the radio that was used had inexpensive hardware and also due to the Doppler Effect on moving objects, the frequency offset between the transmitter and the receiver had to be fixed. Techniques such as observing the Fast Fourier Transform graph of the signal, squaring the signal and also locating its peaks were used to address this issue.



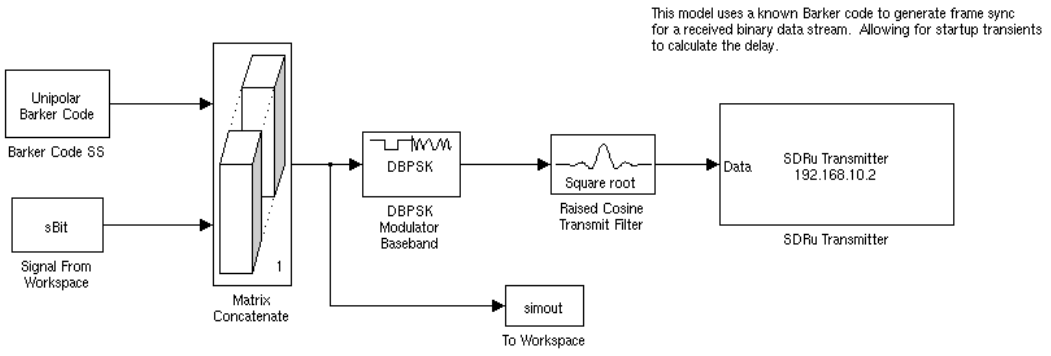

This model uses a known Barker code to generate frame sync for a received binary data stream. Allowing for startup transients to calculate the delay.
Unipolar Barker Code
Barker Code SS
sBit
Signal From Workspace
1
Matrix Concatenate
DBPSK
DBPSK Modulator Baseband
Square root
Raised Cosine Transmit Filter
Data
SDRu Transmitter 192.168.10.2
SDRu Transmitter
simout
To Workspace

Figure 2: A USRP2 in AK318 and its related Simulink block

Secondly, the process of frame synchronization had to be achieved. Frame synchronization is when the receiving radio finds the beginning and end of each frame, or segment of information, in the incoming binary message. This task was accomplished by using purpose-built blocks in Simulink that located a unique marker sequence in each frame called a Barker Code. Once this segment was found, the receiver could then begin decoding the received binary message at the correct point. Decoding the message at the wrong point would create errors, making correct frame synchronization very important.

Lastly, the entire code had to be implemented on the Intel Atom Pico ITX motherboard as seen in Figure 3 and tested in a controlled environment such as Harrington Auditorium, the results of which can be seen in Table 1. Power and heating issues were faced and solved during this step. Our results proved that the code could provide sufficient results within the constraints of this project and that the default amplification of the radio could transmit any file type within a range of 40+ yards.

Table 1: USRP2 Range Test Results

Transmitter Gain	**40 Yards**	**25 Yards**
32 dB	Successfully received	Successfully received
16 dB	Successfully received	Successfully received
8 dB	Received "? ? @? @P ? @? ? ? A? ? A E@ ???"	Received "? ?? ?? @? ?"
0 dB	Received nothing	Received nothing

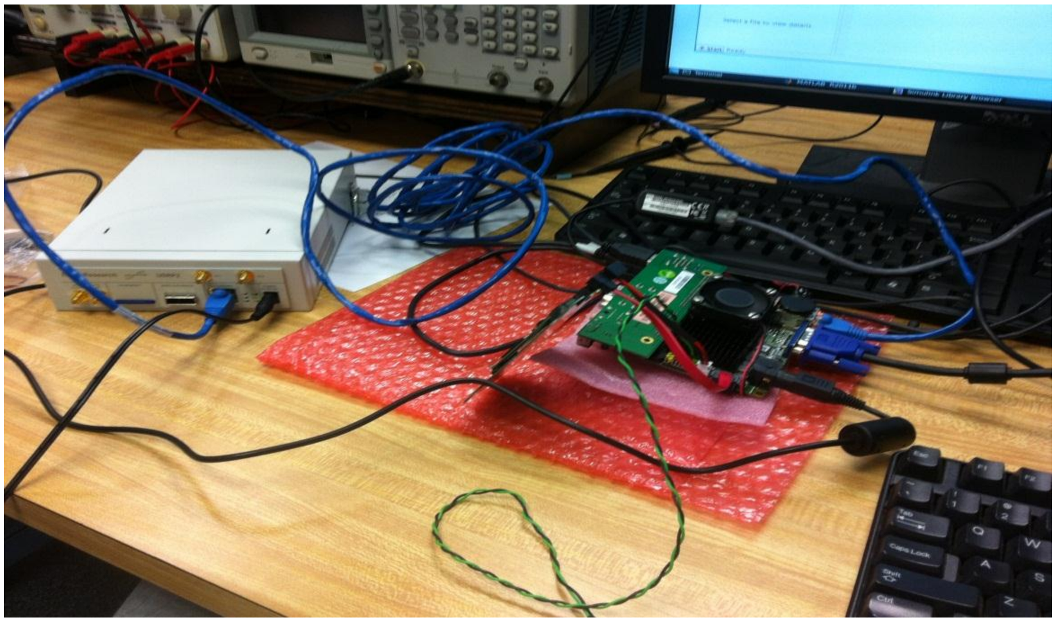


Figure 3: Final setup of the motherboard interfacing with a USRP2

Acknowledgements

This project would not have been possible without the help of certain individuals and organizations.

First, we would like to thank professors Alexander Wyglinski and Taskin Padir. Professor Wyglinski was the primary advisor for this project, providing valuable advice, references, knowledge, and technical expertise. His guidance was vital to the team as it carried out its work. Professor Padir also played an important role as co-advisor, providing the team with additional help and experience, especially from the robotics perspective.

In addition, we would like to thank Worcester Polytechnic Institute and the Electrical and Computer Engineering Department. WPI providing the opportunity to work on such an important project; the ECE Department provided the hardware and software resources to carry it out.

Lastly, we would like to thank The MathWorks Inc. Their support through a grant proved helpful in purchasing most of our equipment. Their software MATLAB and Simulink also powered our design, making them vital components of our project.

Contents

List of Figures

List of Tables

Chapter 1 – Introduction

1.1. The Importance of Search and Rescue

The need for capable SAR crews is an ever-present requirement in the modern world. From large-scale events like earthquakes to small-scale events like missing hikers, there is always the possibility of a disaster or emergency that will require search and rescue efforts to help save lives and recover from the event. SAR crews are uniquely suited to these incidents because of their training and equipment [56]. Normal police or fire departments are often not capable of performing these missions as effectively, making specially trained crews a vital tool in preparing for any potential emergency.

SAR missions span a wide variety of possible disaster scenarios. The general categories are mountain, ground, urban, combat, and air-sea search and rescue. Each of these scenarios poses different requirements and challenges for the rescue crews. For example, a mountain SAR mission would often involve looking for a missing hiker [57]. This situation requires knowledge of the terrain and conditions, the description of the missing hiker, and the last known location. Such information can help the crew find the hiker faster and get him to safety. For example, the National Park Service carried out an annual average of 4,090 SAR operations between 1992 and 2007 at a hefty price of $3.66 million each year [38]. The picture below demonstrates an NPS SAR team in action.

Figure 4: A Helicopter Performing a Search and Rescue Mission in Grand Teton National Park [60]

In contrast, an urban SAR mission might involve responding to a large earthquake. An earthquake could devastate a city, topple numerous buildings, bury many people, and start fires. The SAR crew would need to assess the damage, determine the current state of the disaster area, determine the number of casualties, and rescue and treat the survivors [58]. This knowledge would hopefully help the crew save as many people as possible.

This urban scenario is much more resource intensive than a mountain rescue, but the two share many similarities with each other and the other types of rescue. In both cases, the SAR crews need basic knowledge of the area and their objective. This is often called situational awareness, and it is very valuable because it helps the rescuers minimize further hazards, coordinate their efforts, and more efficiently save lives. The end objective, whether it be one of these scenarios or something else like finding a lost boater or looking for a downed fighter pilot, is based upon the knowledge that rescuers have of the disaster. In addition, proper training and equipment are also vital to a successful mission. Specialized equipment, along with the ability to use it, is important because it lets the crews look for that individual or move debris to get to the survivors [59]. Thus, training and knowledge of the situation are two of the chief requirements for crews to perform a successful SAR mission.

Ideally, every SAR mission would result in a successful rescue of the given individual. However, there are always cases where the rescue is not successful and the individual does not survive. For example, out of the US Coast Guard's average of 55,041 annual SAR responses from 1992 through 2007, they were able to save each year about 4,887 people, but another 781 lives were lost [27]. Sobering statistics like this are often due to unavoidable circumstances, such as if the person perishes before rescue arrives. Sometimes, though, the unsuccessful rescue is due to an inadequate response from the SAR crew. This failure would have to be attributed to one of the main aspects of a rescue, which are their training, planning, or knowledge. Training and planning can be improved through additional practice, but situational knowledge must be obtained when the crisis happens. Thus, increased knowledge of the disaster situation can better prepare the SAR crew and save more lives. The problem is that current methods of obtaining such information are limited to sources such as maps, personal accounts, and visually inspecting the area. These are often time consuming or unreliable, which limits the SAR crew's effectiveness at gathering information and saving as many lives as possible. Another option must be available to obtain the needed information.

Fortunately, such an additional means of gaining knowledge of a disaster area is available. Unmanned Aerial Vehicles (UAVs) have the potential to expand the rescuers' ability to assess the situation and search for individuals. Like helicopters or planes, UAVs provide aerial reconnaissance that can help create an overall picture of the situation. However, UAVs are cheaper than manned aircraft, they can stay airborne for hours, and they often have high resolution cameras, making them useful tools. The application of UAVs to SAR is thus of vital importance to saving more lives.

1.2. The Application of UAVs to SAR

UAVs are potentially a valuable tool for SAR missions. They possess the ability to acquire the knowledge rescue crews need from high above the disaster area and then relay it to a user at a ground station. They could use a combination of normal and infrared cameras in addition to other sensors to maximize the likelihood of detecting an individual. Because there is no one on board, UAVs are safe since they don't put anyone's life directly at risk. Their lack of a pilot also decreases their size and power requirements, making them relatively small and cheap and giving them longer endurance. This can increase their flight time in the search area, making each flight more efficient [63]. The lower power and weight requirements would also make it easier for them to carry supplies to affected individuals. These qualities make them the ideal tool for searching and observation, especially for organizations such as the National Park Service that spend significant amounts of money on SAR. An artist’s concept of a UAV performing such a SAR mission, equipped with camera, is shown here.

Figure 5: Theoretical Use of a UAV During a SAR Mission [62]

Furthermore, UAVs can be grouped together to more quickly search a given area and obtain needed data. This network of UAVs could coordinate its efforts to efficiently cover more ground and acquire a better picture of the situation in the disaster area. Such a network could include as many UAVs as desired, making it fast and easy to acquire information. The network could be modified and adapted to any disaster scenario. For example, if there were a missing individual the UAVs could carry out a detailed search pattern in the given area to cover as much ground as possible to find the person. If the search was unsuccessful, the rescue crew could add more UAVs to the network to expand the search area. On the other hand, the UAVs could be applied to an urban disaster scenario such as a bomb explosion to look in the most critically hit areas. The user at the base station could coordinate all this activity from a single console. The UAVs would send their data to this user, who could then direct this knowledge to the SAR coordinator to better allocate the rescuers’ manpower and resources.

1.3. Other Applications of UAV Networks

The potential applications of UAV networks to SAR are numerous: searching for missing hikers, assessing damage from a terrorist attack, looking for survivors in the most hard-hit areas after an earthquake, determining the size and strength of a forest fire, etc. These capabilities are useful for a wide variety of public service agencies. However, the applications of such a UAV network extend beyond SAR. Law enforcement, universities, and the military could all use such a network [61]. While most of these organizations already use UAVs to some degree, they have not applied a networked group of UAVs to a given problem. The table below lists a number of applications to which such a network could prove advantageous.

Table 2: Applications of UAV Networks [61]

Civilian	*Law Enforcement*	*Military*
• University project platform • Crop dusting • Forest fire monitoring	• Search for fugitives, stolen cars • Monitor demonstration • Border protection • Sovereignty patrols	• Improve situational awareness • Scout enemy positions • Attach or decoy operations

1.4. Current State of UAV SAR Systems

The current state of UAV search and rescue systems is largely in the theoretical and testing phase. This is best evidenced by providing several examples. One such project which helped inspire this one was the iSOAR UAV system developed at the University of Adelaide in South Australia from 2007-2009. Its purpose was to build a remotely controlled UAV that would use a camera to search an area for a missing hiker and then dropping emergency supplies. The user would be gathering the UAV's video data from a distance to help coordinate SAR efforts while the vehicle itself flew autonomously. This project was entered into the ARCAA Outback Challenge for UAV search and rescue systems. While it did not initially succeed, it set a strong example of the capabilities of such a system [29], [35].

Another more advanced example is the UAV Search and Rescue with Human Body Detection project undertaken at Linkoping University in Sweden from 2007-2008. Its purpose was, as the title indicates, to locate individual from the UAV and use this information to coordinate SAR efforts. The project used small remote-control helicopters flying autonomously and equipped with both infrared and regular cameras. During a test scenario, several of these UAVs flying together were able to locate individuals on the ground in a disaster zone and drop emergency supplies to them. The picture below shows two of these UAVs beginning the search phase of the mission. This demonstration of their system proved its functionality, but it has not progressed past this point [33].

Figure 6: Linkoping University UAVs in action

One project that is more closely related to this one was titled Flight Demonstrations of Cooperative Control for UAV Teams, and was worked on by a team of students at the Massachusetts Institute of Technology in 2004. The team used a fleet of eight UAVs as a test platform for evaluating autonomous coordination and control algorithms. The goal was to create a system that manages the simultaneous flight of these UAVs as they carried out a task, which is very similar to this project's goal. They focused on the use of task assignment and waypoint following to coordinate the separate platforms. They tested a number of the algorithms that they developed, but were not able to fully implement the system on the UAVs [64].

Lastly, another project that directly addresses the communications aspect of UAVs is the paper entitled Cross-Layer Routing and Dynamic Spectrum Allocation in Cognitive Radio Ad Hoc Networks, by a group at the State University of New York at Buffalo. Their project was largely theoretical and sought to create an algorithm that maximized throughput in an ad hoc aerial network. They produced an algorithm called ROSA (routing and spectrum allocation algorithm), which performed multi-hop routing, dynamic spectrum allocation, and maintained a bounded bit error rate (BER). They successfully tested the algorithm in the lab but did not implement it on a platform such as a UAV [65].

1.5. Issues with Current Systems

These examples of UAV search and rescue systems demonstrate how this specific area of research has made significant progress over the last decade. All the projects were able to make advanced communications algorithms or functioning UAVs capable of flight and, for some, video transmission. Such previous demonstrations provide an excellent example of what these systems can accomplish.

However, despite their success these projects suffered from one main flaw: lack of scalability in a functioning UAV system. The Linkoping and Adelaide University projects focused mainly on one UAV as it performs search and rescue missions; the MIT project was not fully operational; and the Buffalo project was largely theoretical with no UAVs used. Even the best precedents such as the project at Linkoping University are only able to incorporate a couple UAVs functioning at a time, and even then their total efficiency at carrying out the search is limited.

The abilities of a UAV network with resilient communications would be useful in many applications, but these current systems have not taken steps to pursue this possibility. There is a clear need for a resilient and modular UAV SAR system with a scalable number of platforms that can coordinate their efforts to improve the probability of successfully completing the mission. In particular, the communications aspect of this needed network would be important because its resilient nature would provide the UAVs and user with the flexibility to deal with varying search conditions, multiple UAVs, and multiple data sources.

1.6. Proposed UAV System

To satisfy this need, this team proposes the creation of a resilient aerial wireless network among a group of UAVs to facilitate the gathering of information for rescue crews during a SAR operation. The full system will include a number of UAVs coordinating their efforts to cover a large area and obtain the desired information. There will be a variable number of drones equipped with cameras that perform the actual searching and relay their data to a mothership using Wi-Fi. The mothership will compile this information and then use software-defined radio in the form of the Simulink program and USRP2 radios to transmit the data to a user at the base station. Thus, the use of communications in this UAV system is the focus of this project. A concept diagram of the project's architecture is seen below.

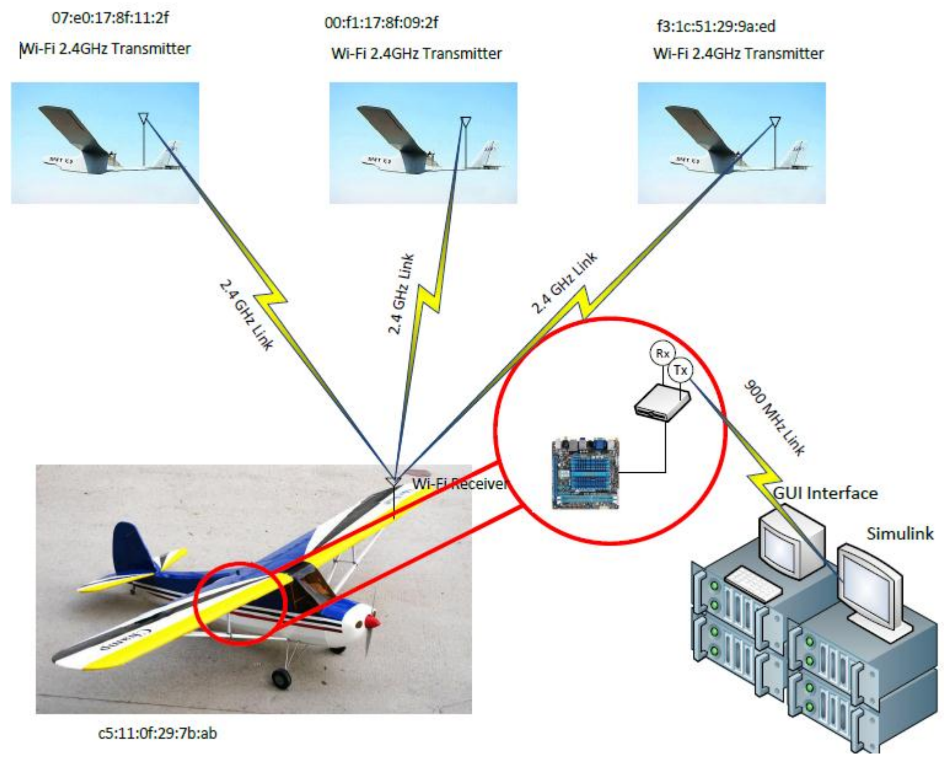


Figure 7: Proposed UAV Network Architecture

The novel aspects of the design that differentiate this project from similar precedents are:

- Scalability of UAV network due to ad hoc architecture: System will be able to incorporate more or fewer drones in real time during operation.
- Combining Wi-Fi and SDR communications links: Mothership will take data from drones over Wi-Fi and send to the base station using SDR.
- Original implementation of user-mothership link: Team will design the SDR protocols that link the user and mothership.

More specifically, this project will lay the groundwork for this communications network by designing its architecture and demonstrating proof of concept in lab. Such a system is very large and complex and cannot be completed in one MQP. Instead, the team will design the overall architecture for the network and how the UAVs will communicate, create a basic communications system to demonstrate feasibility, and put it on a small motherboard that can eventually be mounted on the mothership. These steps will create the basic elements of the desired system and allow the next group to begin the integration of the communications, software, and hardware components of the three separate MQP teams.

1.7. Report Organization

Now that the general motivation for the project has been introduced along with the proposed solution, the rest of the report will delve into the more technical aspects. Chapter 2 will provide relevant background information about SAR statistics and UAV history. It will also provide in great detail the basic theory behind communications systems with the protocols and analysis necessary to understand their functioning. Chapter 3 will then describe the proposed approach of the project. This will include the logistics involved, the way in which the scope of the project evolved, and the general manner in which it was carried out. Chapter 4 will discuss in detail the actual design, implementation, and testing of the groundwork of the communications systems. It will progress in a roughly chronological manner while describing the steps involved with completing each separate section. Lastly, chapter 5 will be a summary of the background topics covered, the structure of the project, the final results, and areas for further study.

Chapter 2 – Overview of Search and Rescue and Communications Methods

Prior to making any design decisions, it is crucial to understand the current state of the art. This chapter will explore the history of SAR UAVs, it will analyze how crucial is the use of more SAR techniques are according to statistics. Furthermore, it will elaborate on what software defined radio is, its architecture and the different tools that are available for implementation. Hence, it will include what is a mobile ad-hoc network (MANET), radio resource management techniques, existing protocols and multi-hop routing algorithms.

2.1. Overview of Search and Rescue

To understand the usefulness of SAR, it is helpful to look at some data. The US National Parks are a perfect example because they have millions of visitors a year. Since these people are often in the wilderness, they experience many possible dangers such as getting lost, injuring themselves, or numerous other hazards. As a result, the Park Service routinely undertakes many SAR missions each year. From 1992 to 2007, there were a total of 65,439 SAR incidents to help 78,488 individuals. The Park Service spent a total of $58,572,164 during this time frame on rescues. Hiking and boating accounted for most of the incidents. In addition, it is estimated that 1 out of 5 of these incidents would have resulted in fatalities. However, 2,659 people still died [38]. The chart below shows the number of lives lost and saved each year.

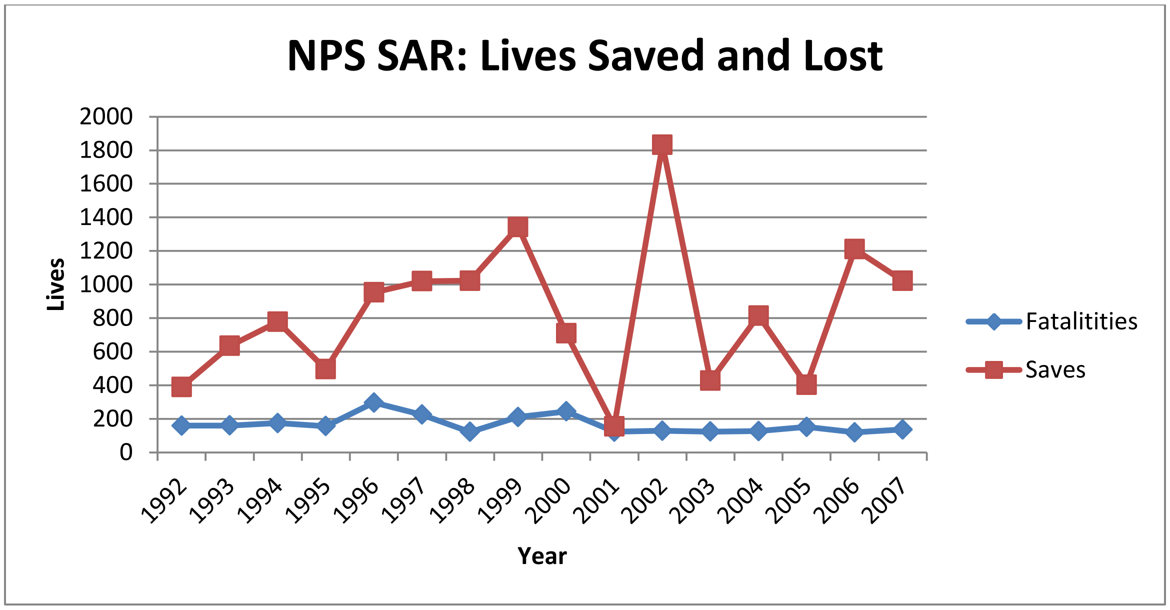


Figure 8: NPS Search and Rescue Success Statistics [38]

As the chart demonstrates, the number of lives saved and lost varies significantly from year to year, but has stayed somewhat constant over time. This is commendable because the ever-increasing number of visitors to the National Parks makes it difficult to keep the amount of lives lost constant. Still, the ideal number of annual fatalities would be zero.

Another branch of the government that is heavily involved with search and rescue is the US Coast Guard. They are responsible for the safety of all people and activities on or near the coast. This is a significant challenge because of the dangers involved, ranging from swimmers being swept away to sea to fishing ships sinking in violent storms. During the same timeframe from 1992 to 2007, the Coast Guard responded to 880654 incidents. They were able to save 78,194 individuals, while another 12,499 lives were lost [27].

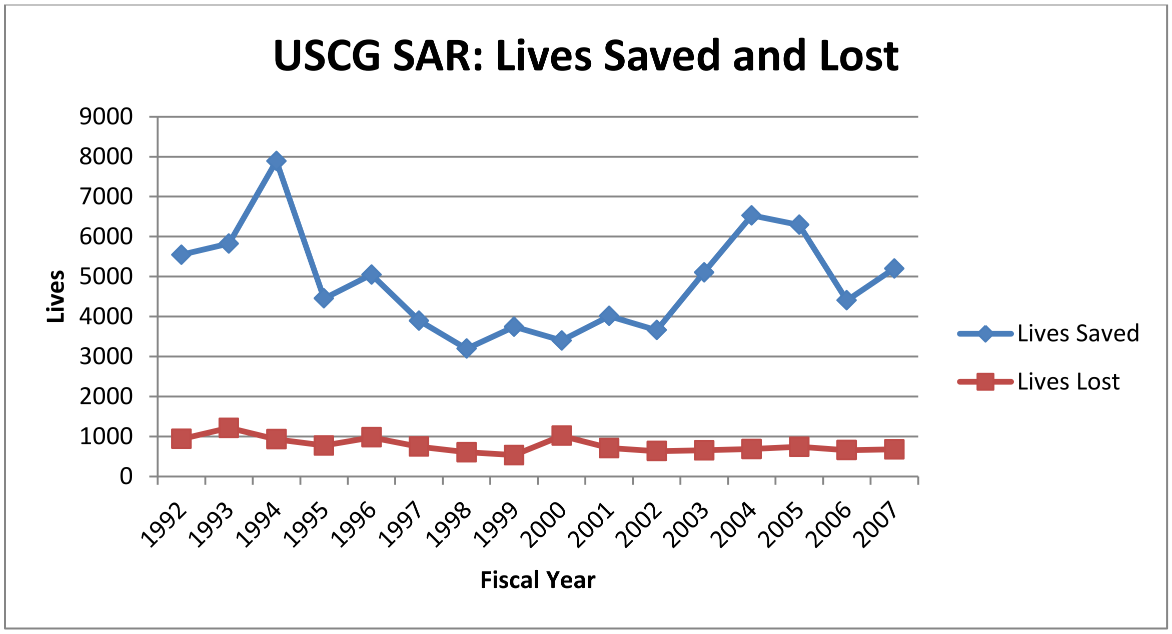


Figure 9: USCG Search and Rescue Success Statistics [27]

As the chart demonstrates, the USCG was also able to keep the number of lives lost fairly constant despite the increasing use of US coastal areas. Still, the fatality rate is much higher than it ideally should be.

2.2. History of UAVs

The history of UAVs has largely followed their growth as a military technology. While they do possess a wide range of capabilities that can be applied to missions such as SAR, the military was the first organization to fund and develop them. This process began as early as World War 1, with the Sperry Aerial Torpedo, when Peter Cooper and Elmer Sperry converted a US Navy biplane into the first radio-controlled UAV. The British improved on this design in the 1930s with the Queen Bee, the first returnable and reusable UAV that was used mainly for target practice. While these designs proved that making a radio-controlled, unmanned aircraft was possible, they were little more than modified airplanes used for simple experimentation [30].

UAV design made a large leap forward beginning with World War 2. In particular, the Germans launched thousands of their new V-1 and V-2 rockets at England with devastating effect. While they are more easily classified as missiles, they were still technically UAVs and thus demonstrated the warfighting capabilities of such a system. The US acquired and adapted this technology, which eventually became the Ryan Firebee drones of the Vietnam War. These UAVs were jet-powered vehicles that were widely used for various tasks such as surveillance and intelligence gathering [36], [41].

Following the Vietnam War, the development of UAVs progressed in leaps and bounds. In the 1970s and 1980s, Israel developed smaller reconnaissance aircraft such as the Scout and Pioneer. By the late 1990s, the US had developed one of the most famous UAVs, the Predator. Similar to the Scout and Pioneer, it was a slightly larger aircraft that could provide up to 16 hours of onsite surveillance with a range of 450 miles. Hellfire anti-tank missiles were fitted to it and provided it with an attack capability that has been extensively used. Other, more advanced UAVs have since been developed, such as the US's high endurance Global Hawk surveillance aircraft. The use of such vehicles has grown exponentially with the wars in Iraq and Afghanistan as UAVs have been used for everything from intelligence to attack missions [30].

Figure 10: A Predator Drone, One of the Most Famous UAVs

All of these historical examples of UAVs are from the military, and for good reason. The development of these systems is time consuming and expensive, which requires resources that usually only the military can supply. In addition, the military applications of UAVs were much more evident than the civilian ones. As a result, there have been few uses of UAVs in non-military fields over the past several decades. However, that is starting to change. They are beginning to be tested in applications such as scientific research, education, agriculture, law enforcement, and especially search and rescue.

Over the past several years, UAVs have begun to be applied to search and rescue. For example, many state National Guards in the US own the Predator UAV and can apply it to SAR missions. Specific instances when UAVs have been used for this purpose include Hurricanes Wilma, Rita, and Katrina in the US in 2005 and the Niigati Chuetsu earthquake in Japan. The specific systems that were used during these disasters were man-portable fixed and rotary wing UAVs, meaning that they were quite small and had limited capabilities [63]. These disasters show that UAVs are beginning to be used for SAR, but the small number of such instances indicates that the field is small and has yet to gain wider acceptance.

2.3. Software Defined Radio (SDR)

Traditional radios often consist of a super-heterodyne or integrated circuit transceiver implemented using dedicated hardware, in contrast with software defined radio. Since software defined radio is such a relatively new concept, it is difficult to find a consensus on a single definition. The basic concept of the SDR software radio is that the radio can be totally configured or defined by the software so that a common platform can be used across a number of areas and the software used to change the configuration of the radio for the function required at a given time. There is also the possibility that it can then be re-configured as upgrades to standards arrive, or if it is required to meet another role, or if the scope of its operation is changed. The SDR Forum, in collaboration with IEEE, has defined it as "radio in which some or the entire physical layer functions are software defined" [3], [4].

The concept of software radio was first published in the early nineties by Joseph Mitola III in a paper on radio architectures at the National Telesystems Conference, New York, in May 1992 [70]. This was followed in May 1995 by a special issue of the IEEE Communication Magazine describing the architecture, ADC, DSP, systems, smart antennas technology and the economy of SDR Technology. At the same time the US DoD initiated SPEAKeasy as the first publicly announced military software radio, then DARPA continued with the SPEAKeasy II program. The interest was further spurred on by the formation of the MMITS (Modular Multifunction Information Transmission System) Forum in 1996 (later transformed into the SDR Forum) [3].

The SPEAKeasy program started with a phase where functions such as programmability, flexibility, reconfigurability, and the use of signal processors were illustrated. It showed the capability of being able to communicate with multiple legacy systems simultaneously at demonstrations. The demonstrations in 1994 were conducted with over-the-air transmission and reception using standard HF (high frequency), VHF (very high frequency), and UHF (ultra high frequency) antennas covering the 90-200MHz band.

The successes of the initial phase lead to a continuation in 1995 where the objective was set to develop field capable prototypes with full RF capability. The implementation had to include commercial off the shelf (COTS) components, the use of non-proprietary buses, open architecture, INFOSEC (information security) and wideband data waveforms. Lacking additional funding the SPEAKeasy program was restructured in 1997 as all the tasks related to the wideband capability were eliminated [3]. However, there was sufficient interest to initiate a new program and the Joint Tactical Radio Systems (JTRS) program was established to investigate the requirements for scalability, the portability of waveforms, and the development of a common software communications architecture (SCA) that would facilitate the simple exchange of waveforms.

This unique radio technology works much like personal computing, where a single hardware platform can carry out many functions based on the software applications loaded. SDR uses software to perform radio-signal processing functions instead of using discrete electronic components, or application-specific integrated circuits. Frequency tuning, filtering, synchronization, encoding and modulation are now functions performed in software on high-speed reprogrammable devices such as digital signal processors (DSP), field programmable field arrays (FPGA), or general purpose processors (GPP). RF components are still needed for generation of high frequencies or for signal amplifications and radiation but SDR aims at reducing their usage to a minimum [3].

One major initiative that uses the SDR, software defined radio, is a military venture known as the Joint Tactical Radio System (JTRS). Using this, a single hardware platform could be used and it could communicate using one of a variety of waveforms simply by reloading or reconfiguring the software for the particular application required. This is a particularly attractive proposition, especially for coalition style operations where forces from different countries may operate together. An application of those radios is that they could be re-configured to enable communications to occur between troops from different countries.

SDR technology supports over-the-air upload of software modules to subscriber handsets. This helps both network operators as well as handset manufacturers. Network operators can perform mass customizations on subscriber's handsets by just uploading appropriate software modules resulting in faster deployment of new services. Manufacturers can perform remote diagnostics and provide defect fixes by just uploading a newer version of the software module to consumers' handsets as well as network infrastructure equipment [3].

The SDR software radio concept is equally applicable for the commercial world as well. One application may be for cellular base stations where standard upgrades frequently occur. In this project we were able to use a Software Defined Radio platform in order to achieve reliable communications between Unmanned Aerial Vehicles (UAVs). Our goal was to implement an algorithm that could be able to exchange information (e.g. global position status for rescuing a person) between two or more SDR platforms. Using concepts such as spectrum sensing and error detection, our initial objective was achieved.

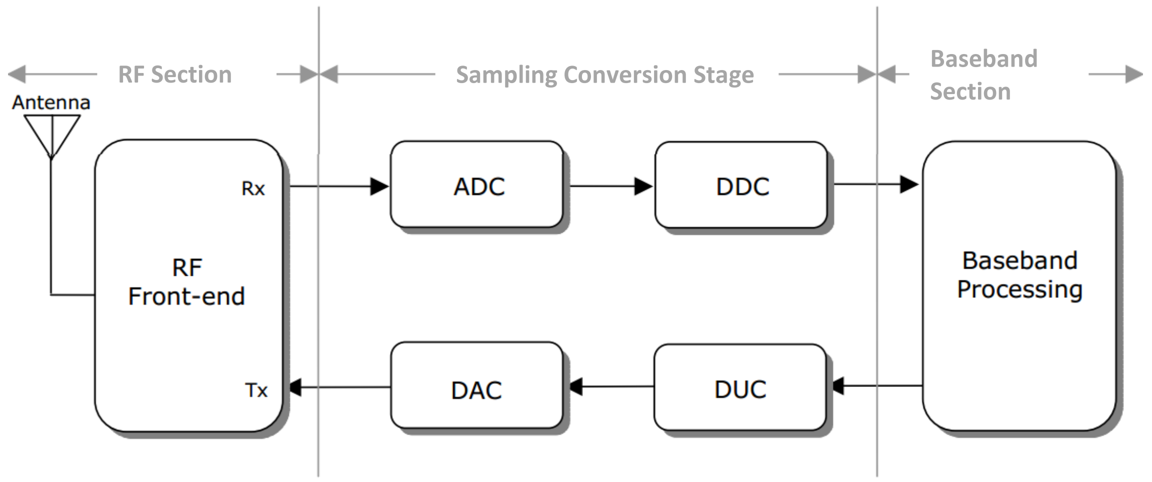


Figure 11: Block diagram of a generic Software Defined Radio Transceiver

Figure 11 provides a demonstration of a digital transceiver. The RF section (also called as RF front-end) is responsible for transmitting/receiving the radio frequency (RF) signal from the antenna via a coupler and converting the RF signal to an intermediate frequency (IF) signal. The RF front-end on the receive path performs RF amplification and analog down conversion from RF to IF. On the transmit path, RF front-end performs analog up conversion and RF power amplification. The Sampling Conversion Stage consisting of the ADC/DAC blocks perform analog-to-digital conversion (on receive path) and digital-to analog conversion (on transmit path), respectively. ADC/DAC blocks interface between the analog and digital sections of the radio system. DDC/DUC blocks perform digital down conversion (on receive path) and digital-up-conversion (on transmit path), respectively. DUC/DDC blocks essentially perform modem operations, i.e., modulation of the signal on transmit path and demodulation (also called digital tuning) of the signal on receive path. The baseband section performs baseband operations (connection setup, equalization, frequency hopping, timing recovery, correlation) and also implements the link layer protocol.

Some of the advantages of a software defined radio relative to an analog radio are [45]**:**

i. “The ability to receive and transmit various modulation methods using a common set of hardware; ”
ii. “The ability to alter functionality by downloading and running new software at will.”
iii. “The possibility of adaptively choosing an operating frequency and a mode best suited for prevailing conditions;”
iv. “The opportunity to recognize and avoid interference with other communications channels;”
v. “Elimination of analog hardware and its cost, resulting in simplification of radio architectures and improved performance;” and
vi. “The chance for new experimentation.”

However, a few obstacles remain to their universal acceptance. Those include [7]:

i. “The difficulty of writing software for various target systems”,
ii. “The need for interfaces to digital signals and algorithms”,
iii. “Poor dynamic range in some SDR designs” and
iv. “A lack of understanding among designers as to what is required”.

2.3.1 Universal Software Defined Radio 2 (USRP2)

The USRP2 is a low-cost software defined radio platform produced by Ettus Research. The device consists of a Gigabit Ethernet host computer interface, a Xilinx Spartan FPGA, and compatibility with all USRP RF modules. Two input channels and two output channels are provided, and MIMO capability is supported when multiple USRP2s are connected together. Only one full transceiver is supported, but up to 50MHz of signal bandwidth can be used due to the 100MS/s ADC sampling rate [13]. The USRP2 is fully supported by GNU Radio and Simulink, which are discussed in Section 2.4.1 and Section 2.4.2 respectively. The specifications of the USRP2 comparing to USRP, which uses a USB 2.0 computer interface in contrast with the Gigabit Ethernet, resulting in slower data transmission

Table 3: Comparison chart between USRP and USRP2 [3]

USRP* SPECIFICATIONS		
	USRP(1)	**USRP(2)**
Interface	USB 2.0	Gigabit Ethernet
FPGA	Altera EP1C12	Xilinx Spartan 3 2000
RF BW (to/from host)	8 MHz @ 16bits	25 MHz @ 16bits
ADC	12-bit, 64 MS/s	14-bit, 100 MS/s
DAC	14-bit, 128 MS/s	16-bit, 400 MS/s
Daughter boards (capacity)	2 TX, 2 RX	1 TX, 1 RX
SRAM	None	1 Megabyte
Other	-	MIMO support

At time of writing, the USRP2 is being discontinued in favor of the USRP N200 and USRP N210. These radios offer the same daughter card and transceiver capabilities along with a 100MS/s ADC sampling rate, but have different Xilinx FPGAs and several more subtle differences. Ettus Research has stated that the USRP N200-series is code-compatible with the USRP2, allowing code written for the USRP2 to be used seamlessly with a newer radio [13]. The most important advantage of an N210 USRP is that it requires no flash card for configuration of UHD or UDP packets as discussed in section 2.4.3.

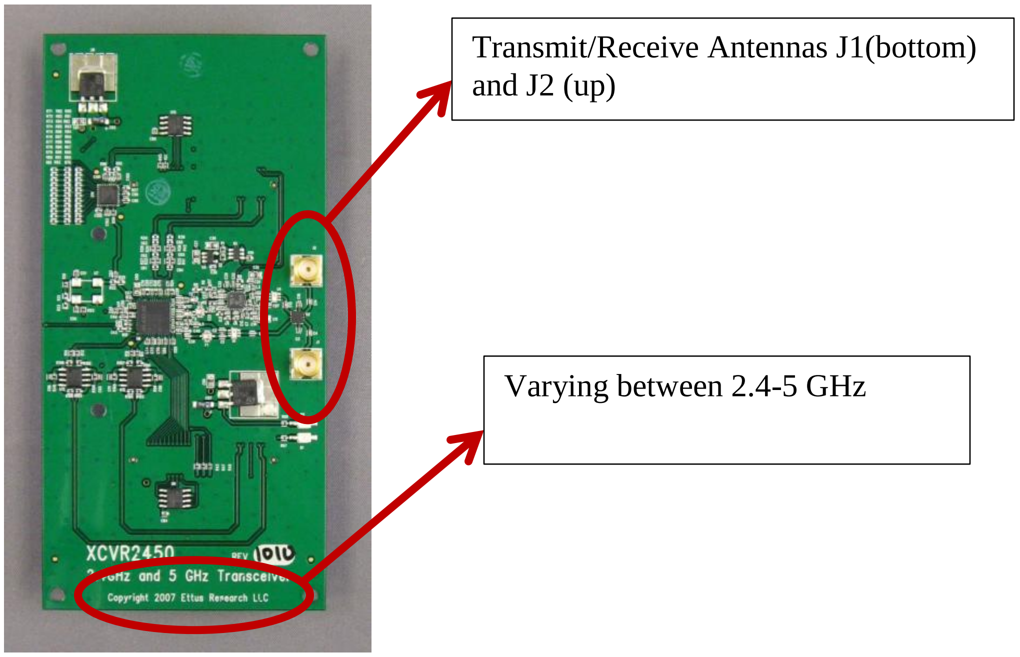

Transmit/Receive Antennas J1(bottom) and J2 (up)
Varying between 2.4-5 GHz
XCVR2450
REV 1010
2.4GHz and 5 GHz Transceiver
Copyright 2007 Ettus Research LLC

Figure 12: XCVR 2450 Daughter Card used for transmission and reception, ranging between 2.4-2.5 GHz and 4.9-6.0 GHz

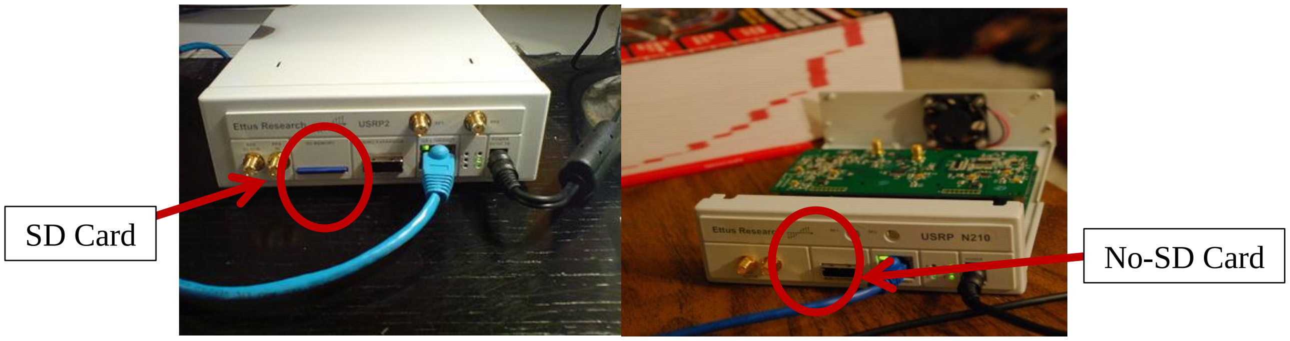


Figure 13: A USRP2 (left) VS USRPN210 located in Atwater Kent Laboratories

Figure 12 shows the PCB layout of the XCVR 2450 Daughter card that was used in this project for transmission and reception. Filtering on the XCVR2450 provides exceptional selectivity and dynamic range in the intended bands of operation. The typical power output and noise figure of the XCVR2450 is 100 mW and 8 dB, respectively. Notice the two RF antennas that they were connected to the front end of the USRP2.

Figure 13 shows one of the USRP2s that was used for this project VS the USRP N210 which will replace USRP2. The main advantage of the USRP N210 is that it does not use an SD card in order to configure its hardware. In fact, it can automatically adjust the software that controls its hardware to any configuration of the radio.

2.3. Data Transmission

Data transmission is related to the software defined radio parameters are also known as decision variables since their purpose is to provide the information needed to a cognitive algorithm that will optimize the system. The cognitive parameters presented here are RSSI, and BER. This section will serve as a tutorial for communication parameters.

2.3.1. Received Signal Strength Indicator (RSSI)

The received signal strength indicator is one of the most common cognitive parameters to be measured in wireless systems. The principle concept of RSSI is that the transmitted power is proportionately related to the received power. The received power decreases quadratically with the propagation distance. This can be modeled by [7]:

$$P_R = P_T * G_T * G_R * \left(\frac{\lambda}{4\pi}\right) * \left(\frac{1}{d}\right)^n \qquad (1)$$

In this equation (Friis' free space equation), P_R and P_T are the received and transmitted powers respectively. Likewise G_T and G_R are the gains of both the receiver and transmitter antennas. The wavelength is represented with λ. The distance between transmitter and receiver is d. As the distance from the transmitter increases there is a quadratic decrease in signal strength. The n in the system for free space is 2, but can be higher in different medium. It can be seen that the larger the wavelength of the propagating wave the less susceptible it is to path loss. Therefore at higher frequencies, radio waves cannot travel as far with the same transmission power [7]. Friis' free space equation is a generalization of the signal strength that will be received [7]. However, when interference is considered the signal strength may no longer follow the equation. This interference may be due to multi-path signals or other devices in the frequency band. This interference may be constructive, which will appear to be higher signal strength than the power received from the target. On the other hand, destructive interference will cause the device to read signal strength lower than the equation would predict. RSSI is a widely used cognitive parameter that is readily accessible on most devices. Though the measured RSSI may not correspond exactly to the power of the desired signal it is a fairly reliable guide to the general performance of the system [7].

2.3.2. Error Detection/ Repetition Coding

Repetition coding is a relatively simple coding scheme in which bits are repeated before being sent across a communications channel. For a repetition factor of k, each bit is repeated k times. Repetition coding effectively increases the bit-rate requirements for the channel capacity. Thus for a given channel capacity, increasing the repetition rate effectively reduces the rate at which information can be sent (i.e. it limits the coder input rate). This is illustrated in Figure 14. Note that the increased bit rate assumes that the frame/sample rate remains the same, and only the frame size increases by a factor of k.

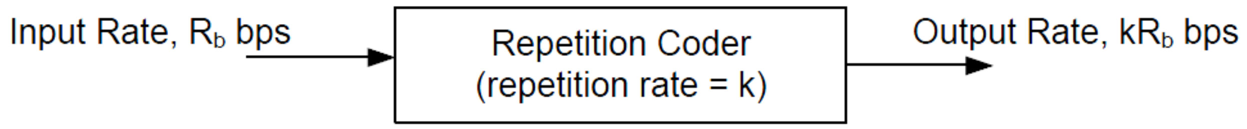


Figure 14: Relating repetition coder input and output bit rates for fixed sample rate

After passing through the channel, the repetition decoder removes the redundant bits. This was done by the use of a 'majority bit approach' as follows. In general for a received bit at time nT_s, where T_s is the sample time, and n is the integer-valued time index,
$r(nT_s) = s(nT_s) + v(nT_s)$ In our representation channel delay is ignored, $s(nT_s)$ represents the transmitted bit and $v(nT_s)$ represents the additive white Gaussian noise added to the bit. Using this model for the k copies of the received bit, it is clear to see that each one may be independently corrupted by noise. To determine the value of a particular bit, we calculate the average of the k values and then use the following decision rule,

$$d(\mathrm{n}\mathrm{T}_S) = \begin{cases} 0, & mean\ value < \frac{1}{2} \\ 1, & mean\ value \geq \frac{1}{2} \end{cases} \qquad (2)$$

By using repetition within transmission it can be easy to detect corruption in the data. It also provides clues as to what the correct sequence should be. By repeating bits or blocks a predetermined number of times then they can be compared with each other for continuity. If all of the blocks or bits are the same in the sequence it can be determined that an error did not occur. If there is a difference in bits then it is common to quantize the bit to the high mode of the string. Therefore if an odd number of repetitions are sent in a binary decision, whichever of the two choices was transmitted more times is more likely to be the original transmission. This technique is much more effective when the blocks of data repeated are small or are repeated many times. This gives a clearer mode of the data or can make it such that there are less ways for it to fail and therefore makes it easier to determine 10 which sequence is the correct one. The negative aspect to this technique is that it is very inefficient. By transmitting the same information with much redundancy the overall transmission will be longer and at a given symbol period, it will appear that there is a slower bit rate or a delay. In Figure 15, there is an example of an eight bit to redundancy scheme

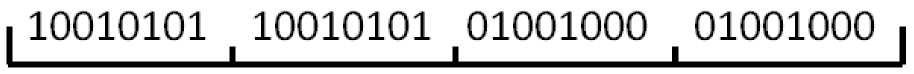


Figure 15: Example of an eight bit to redundancy scheme

Parity can be used for a more efficient practice. In this technique a parity bit is assigned for a known number of bits to be determined. The parity scheme can either be even or odd as decided by the programmer. The parity algorithm counts how many ones are in the stream behind it. If the parity scheme is odd then the parity bit will assign a one if there are an odd number of ones in the stream behind it. It will assign a zero for an even number of ones. An even parity scheme would work in the opposite way. This means that there only needs to be one bit every sequence to determine error rather than redundancy of the transmission. This makes the process faster by making the total data shorter. There are several issues with the reliability of this method. For instance, there is a problem if the number of corrupted bits is a multiple of two. This will not change if there is an even or an odd number of ones in the sequence. Therefore, the parity bit will return a "no error" when one has occurred. The other problem is when the parity bit itself is corrupted while the data is correct. This will return an "error" even when one has not occurred. Calling for parity bits more often will reduce the possibility of the first error happening while increasing the possibility of the second error happening. This will be a design decision implemented into any forward error correction that may be used. An example of a parity scheme can be seen in Figure 16. In each byte in the figure, the number of bits set to '1' is made even by setting the last bit, known as the parity bit, to '1' or '0'. If the number of bits set to '1' is not even in each byte when the signal is received, then there has been a bit error in the byte checked.

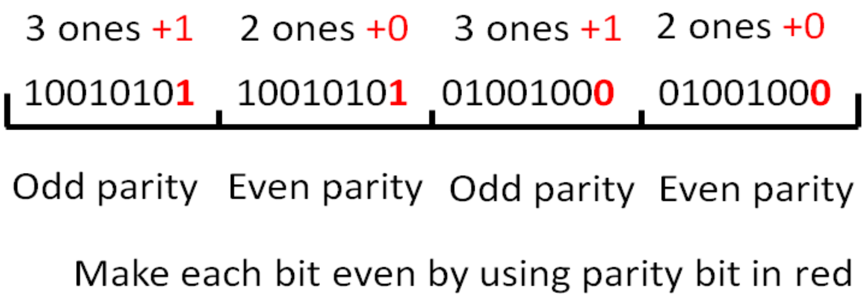


Figure 16: Example of a parity scheme where the eighth bit is an odd parity bit

The technique of checksum is used for overall transmission. It assigns a sequence at the end of the transmission that can be decoded to an equivalent of the number of bits that should have been received. This is so that the system knows if it has received the entire sequence or if packets had been dropped. The system counts the received bits and compares its checksum with the equivalent sequence that is generated for the number of bits received. If the checksum and the counted number of bits correspond, the entire sequence is deemed to have been transmitted.

2.3.2.1. Bit Error Rate

One of the most important ways to determine the quality of a digital transmission system is to measure its Bit Error Ratio (BER). The BER is calculated by comparing the transmitted sequence of bits to the received bits and counting the number of errors. The ratio of how many bits received in error over the number of total bits received is the BER. This measured ratio is affected by many factors including: signal to noise, distortion, and jitter. The most obvious method of measuring BER is to brute force send bits through the system and calculate the BER. Since this is a statistical process, the measured BER only approaches the actual BER as the number of bits tested approaches infinity. Fortunately, in most cases we need only to test that the BER is less than a predefined threshold. The number of bits required to accomplish this will only depend on the required confidence level and BER threshold.[46] The confidence level is the percentage of tests that the system's true BER is less than the specified BER. Since we cannot measure an infinite number of bits and it is impossible to predict with certainty when errors will occur, the confidence level will never reach 100%. The test time will be determined by how often the software defined radio receiver will want the value to be refreshed. This is an important decision as BER will be an important cognitive value in the system. It will be necessary for knowing how poorly the filtering in the system is working and how much noise is present in the system. The test time can be determined with:

$$t = -\frac{\ln(1-c)}{b*r}, \tag{3}$$

where t is the test time, c the degree of confidence level desired, b the upper bound of the BER and r the data rate.

For this project, we had to follow a procedure named as bit error rate test or tester (BERT) in order to measure the BER for a given transmission. The formula that helped us plot our results is the energy per bit to noise power spectral density ratio:

$$\frac{E_b}{N_o}(dB) = C - N_o - 10 \times \log_{10}(f_b) \tag{4}$$

where f_b is the transmission rate; N_o is the noise spectral density (dBm/Hz); E_b is the energy per bit (dBm/Hz) and C is the carrier power (dBm). Figure 17 shows the bit error rate plot versus the energy per bit to noise power spectral density ratio between different modulation schemes. Our results for the calculation of the bit error rate can be seen in chapter 4.

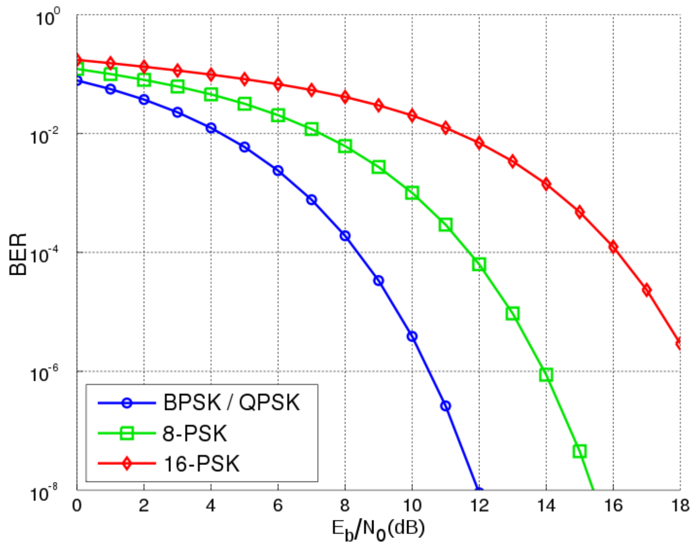


Figure 17: PSK BER curve (from [46])

2.3.3. Spectrum Sensing

Spectrum sensing is where the focus of cognitive radio research is starting to move towards. It is the capability of a device being aware of the frequency domain, or the radio frequency spectrum, of its surroundings. This can detect various forms of interference. It can be as simple as evaluating where there is high energy in the system or as complicated as detecting unknown modulated signals over random noise.

2.3.3.1. Energy Detection

Energy detection is a signal detection mechanism based on Neyman-Pearson approach [9] [10]. The concept of energy detection mechanism is quite simple. The detector computes the energy of the received signal and compares it to certain threshold value to decide whether the desired signal is present or not. The energy of the signal is preserved in both time domain and frequency domain. The time domain representation of this mechanism is shown in Figure 18. The frequency domain representation of this mechanism is shown in Figure 19. Theoretically, whichever representation is used for signal detection and analysis makes no difference in result. However in the former representation a pre-filter matched to the bandwidth of the signal is required. This need makes this representation quite inflexible compared to the frequency domain representation. So, it is intended to use the second representation in near future for analyzing the received signal via simulation.

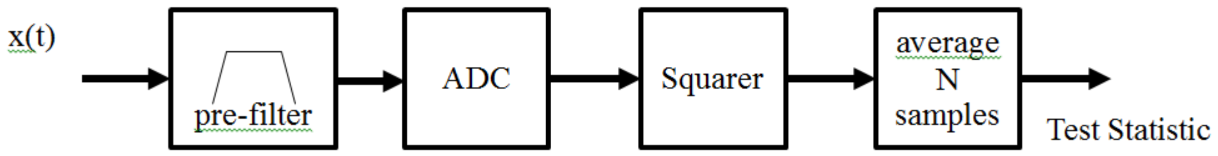


Figure 18: Time representation of energy detection mechanism

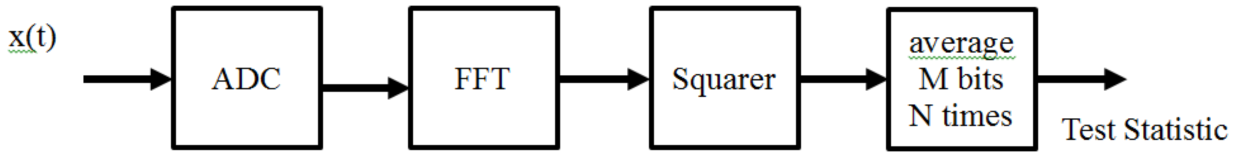


Figure 19: Frequency domain representation of energy detection mechanism

In order to measure the signal energy, the received signal is first sampled, then converted to frequency domain taking FFT followed by squaring the coefficients and then taking the average.

2.3.3.2. Cyclostationary Feature Detection

Cyclostationary feature detection is a method of differentiating primary user signals from noise without prior knowledge of their modulation schemes or protocols. All signals can be modeled as stochastic processes which are probability functions with random variables through time. Stochastic processes are broken up into subcategories based on how random they are. Wide sense stationary processes are stochastic processes with a constant mean such as noise which has zero mean. Cyclostationary processes are stochastic processes with statistical properties that vary cyclically. Modulated signals are cyclostationary processes because they are double sided with sine wave carriers, they have a fixed symbol period, and each modulation type has its own unique cyclostationary features [47].

In a more mathematized definition, cyclostationary processes have periodic autocorrelation functions where wide sense stationary signals do not. This means that the autocorrelation of cyclostationary processes can be written as a Fourier coefficient. The Fourier coefficient form of autocorrelation for cyclostationary processes is expressed in [47];

$$R_x^{\alpha}(\tau) = \lim_{T\to\infty} \frac{1}{T}\int_T x(t+\tfrac{\tau}{2}) * x * \left(t-\tfrac{\tau}{2}\right) * e^{-j2\pi\alpha t}dt \qquad (5)$$

This is called the cycle autocorrelation. If the statistically correlated periodic features in a cyclostationary process repeat every T, then the cycle autocorrelation has a cycle of α. Since the autocorrelation function is a quadratic transform the features of modulated signals that are functions of symbol rate and carrier can be detected.

The cycle autocorrelation is only a time domain transform. This means it falls short in measuring power spectral density of modulated signals. This is because the system is demodulating many different signals. The maximum energy of all the modulated signals is unlikely to be equivalent. Therefore a threshold would only be set by the highest energy signal. The frequency domain equivalent is called the spectral correlation function, which is expressed in [47]:

$$S_x^{\alpha}(f) = \lim_{\Delta t\to\infty} \frac{1}{\Delta t}\frac{1}{T}\int_{-\Delta t/2}^{\Delta t/2} X_T\left(t, f+\frac{\alpha}{2}\right) X_T^{*}\left(t, f-\frac{\alpha}{2}\right)dt \qquad (6)$$

$$X_T(t,f) = \int_{t-\frac{T}{2}}^{t+\frac{T}{2}} x(u)e^{-j2\pi f u}du \qquad (7)$$

Through Weiner's relationship, the Fourier transform of cycle autocorrelation is the spectral correlation function. The spectral correlation function is a two dimensional complex transform with to frequency based axis cycle α and frequency f that is used for feature detection [47].

To measure spectral correlation of a function, the received signal can be frequency shifted α Hz and –α Hz in two parallel mixers. This searches many different frequencies for possible carriers. After each frequency shifted signal is passed through identical band pass filters, the signal that was frequency shifted –α Hz is then complex conjugated. After the complex conjugation, the two signals match in carriers of both the real and imaginary plane for attempted demodulation. The two signals are then frequency multiplied back together and averaged over a period of time. If the final signal possess a high energy level, this means that the cycle α corresponds to a carrier frequency of a modulated signal [47]. The block diagram is depicted in Figure 20.

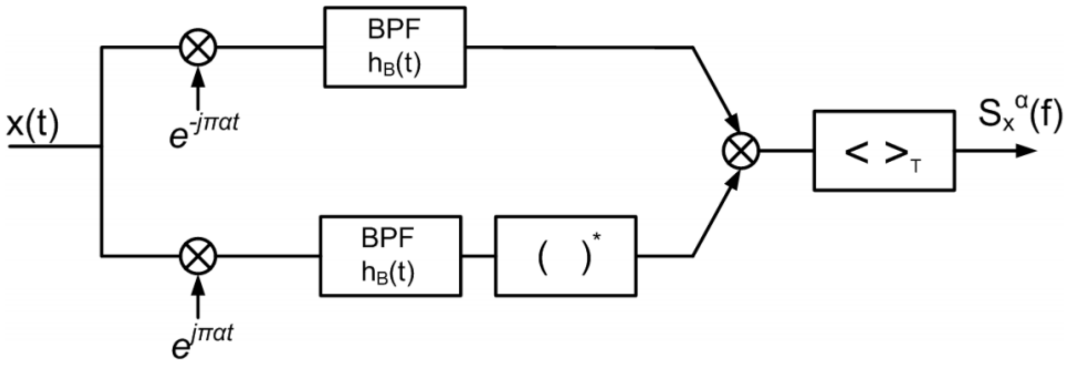


Figure 20: Spectral correlation block diagram [5]

The frequency shifting, bandpass filtering and the complex conjugation can all be implemented using a fast Fourier transform for any f and α. To implement the cyclostationary feature detector the received analog signal must be converted to digital. The digital signal is then input into an N point FFT. The conjugate outputs are then correlated and averaged over a period T. The feature detection then senses the peaks for α>0 and can even distinguish between different modulation schemes and protocols with simplified matched filtering [47]. The implementation is depicted in Figure 21.

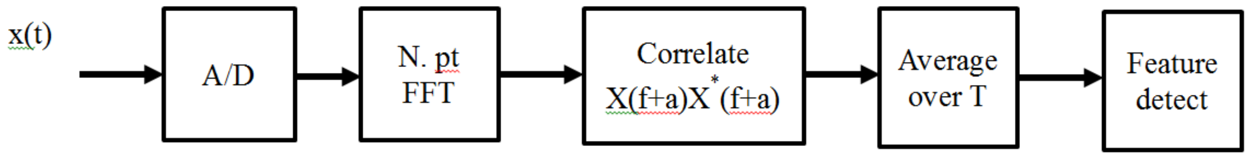


Figure 21: Implementation of Cyclostationary Feature Detector [5]

2.3.3.3. Matched Filtering

Matched filtering is the optimal method of signal detection since it effectively maximizes received signal to noise ratio. It follows the full cognitive radio model described earlier in that a prior knowledge of the primary user signal at both the physical and medium access control layers is required. This knowledge is the pulse shape that is used for the transmission. The radio has stored memory effectively describing the modulation type, order, pulse shaping and packet format to demodulate the signal. It is also required to synchronize with the carrier and time scheme as well as perform channel equalization [48].

Match filtering takes advantage of the fact that a filter that is the time reverse of the pulse shape used, when convolved with the signal, optimizes the energy of the signal. This is due to the fact that the filter is time reversed again and passed over the system when the convolution occurs. If the filters match, it will be a perfectly constructive superposition. However, a major drawback is the programming of all the different standards into the memory of the radio and dedicating enough receivers to detect all the primary users [48].

2.4. Software Selection for SDR Design

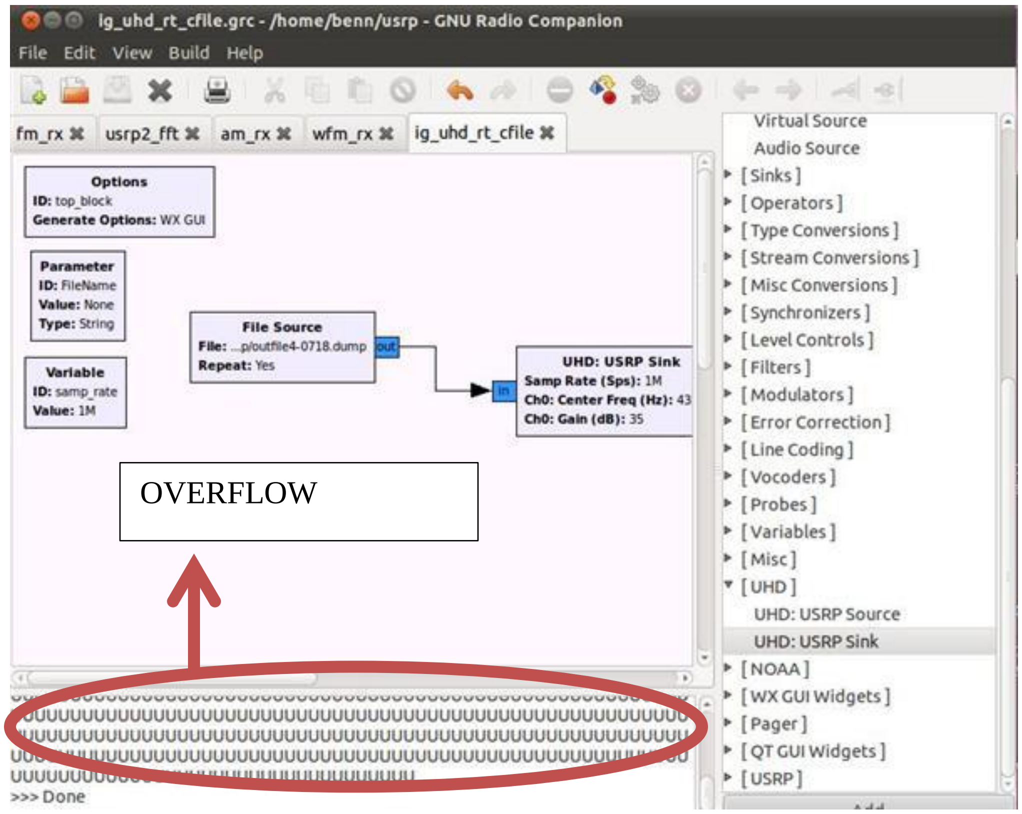
In order to use the USRP2 we had to find the appropriate software. This software could help us program the device and achieve reliable communications between the radios. The two main software platforms that could achieve that are GNU Radio Companion and MATLAB/Simulink. In this section we will elaborate on how each of these two platforms operates.

2.4.1. GNU Radio Companion (GRC)

GNU Radio is a free, open-source software environment for the implementation of low-cost software defined radio systems. The GNU Radio project was created so the general public could experiment with radio hardware and the wireless spectrum, and does so by providing free software to complement low-cost hardware [49]. The project is used by a variety of academic, government, and commercial researchers as well as some amateur radio enthusiasts, and supports both simulation environments and Ettus Research software-defined radio hardware products [49][50][51].

As of GNU Radio 3.2.2 and 3.3.0, the software provides a framework for combining signal processing blocks into flowgraphs. Each block, written in the C++ programming language for performance reasons, implements a single data processing step such as matched filtering [6][52]. GNU Radio provides numerous commonly-used blocks, though users are encouraged to create their own when a block they need is unavailable [49]. Each block executes in a separate operating system thread, allowing for pipelined processing that efficiently exploits multicore computer architectures [53]. Flowgraphs are typically written in the Python programming language and represent a data stream between the radio hardware and the user‘s choice of a data source or output. Additionally, flowgraphs can be produced automatically through a graphical user interface (GUI) tool called GNU Radio Companion (GRC). This implementation permits easy radio system design and development without compromising runtime performance [6], [49].

While GNU Radio is fully open source, has a strong user base, and is a well-established software-defined radio platform, the code remains in active development and is generally lacking in documentation. Significant bugs are routinely discovered and fixed, while backwards compatibility is often broken as the software attempts to benefit from the latest developments in the many libraries it depends on. Documentation consists of several tutorials, an API, email archives and code comments while end-user support is frequently performed by volunteers [49]. GNU Radio addresses the host computer software used with a USRP or USRP2, and when combined with either Ettus Research product a complete software-defined radio is formed.



Options
ID: top_block
Generate Options: WX GUI

Parameter
ID: FileName
Value: None
Type: String

Variable
ID: samp_rate
Value: 1M

File Source
File: ...p/outfile4-0718.dump
Repeat: Yes

UHD: USRP Sink
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 43
Ch0: Gain (dB): 35

```
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
>>> Done
```

Figure 22: GNU Radio Companion Overflow Example

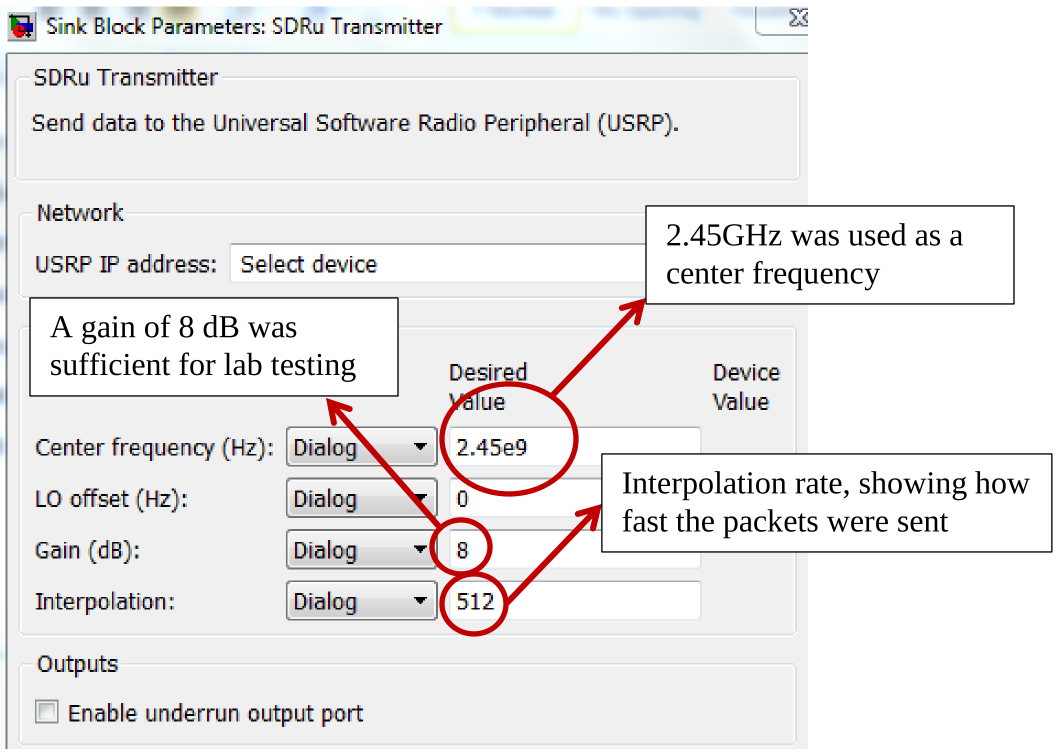
Figure 22 shows a screen capture of the Graphical User Interface (GUI) of then GRC. This block consists of a message in a text format that connects to the USRP Sink in order to get transmitted. However, we get an overflow since this message had to be formatted correctly using Python scripts in order to get transmitted properly.

2.4.2. Simulink Communications Blockset

The MathWorks Inc. chose to extend their MATLAB and Simulink software with a real-time communications processing toolbox, including USRP2 hardware blocks [51]. Simulink has long been established as a simulation toolbox, and can be used to implement flowgraphs in a visual form. This functionality is similar to GNU Radio Companion, except fully integrated with the Simulink product line. This provides easy access to, for example, common digital signal processing and logic functions. Additionally, this software interfaces with the well-established software package MATLAB. MATLAB focuses on offline vector and matrix manipulation as well as data visualization [52]. At the time of writing, numerous limitations still exist, such as comparatively weak USRP2 support and an implementation that was poorly optimized for real time data processing on multicore processors. The authors were exposed to this product and its limitations during an academic course on software defined radio during which the MathWorks Inc. gathered feedback about its performance. It is expected that many of these issues will be resolved in future releases.

We started completing our project using the R2011a version of MATLAB. However, as the project was on going The MathWorks released the version R2011b. The significant difference between those two versions was that the USRP2 hardware blocks no longer required the use of UDP packages. The new configuration was called a UHD package. In section 2.4.3 we explain the difference between UDP and UHD.

One of the most important advantages of Simulink in contrast to GNU Radio Companion is that it is friendlier to the user. One can customize the properties of each block that is essential for the reliable communication between the radios without having a strong background in programming languages such as Python and XML. Figure 23 shows the block properties of the SDRu Transmitter that was used for this project.


Sink Block Parameters: SDRu Transmitter

SDRu Transmitter

Send data to the Universal Software Radio Peripheral (USRP).

Network

USRP IP address: Select device

		Desired Value	Device Value
Center frequency (Hz):	Dialog	2.45e9	
LO offset (Hz):	Dialog	0	
Gain (dB):	Dialog	8	
Interpolation:	Dialog	512	

Outputs

☐ Enable underrun output port

2.45GHz was used as a center frequency

A gain of 8 dB was sufficient for lab testing

Interpolation rate, showing how fast the packets were sent

Figure 23: Block Properties of the SDRu Transmitter

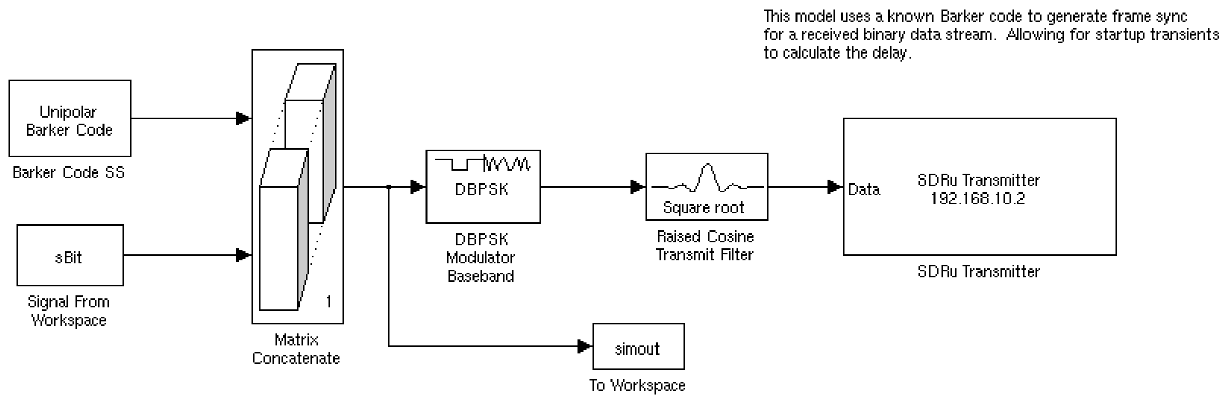

This model uses a known Barker code to generate frame sync for a received binary data stream. Allowing for startup transients to calculate the delay.
Unipolar Barker Code
Barker Code SS
sBit
Signal From Workspace
1
Matrix Concatenate
DBPSK
DBPSK Modulator Baseband
Square root
Raised Cosine Transmit Filter
Data
SDRu Transmitter 192.168.10.2
SDRu Transmitter
simout
To Workspace

Figure 24: Simulink Transmitter Block for frame synchronization

In figure 24 we provide the block diagram of the transmitter that we used in our project in order to perform frame synchronization. In Chapter 4 we will explain in further detail the functionality of each block.

2.5. Chapter Summary

In this chapter, search and rescue statistics, UAV history, and the principles of communications were introduced and discussed. The SAR data showed the importance of this function for both the US National Park Service and the US Coast Guard. However, the need for SAR extends far beyond these two organizations to almost every conceivable disaster. The history of UAVs and their evolution over the past century was discussed, including their use in the military and their recent application to SAR. In addition, a number of topics in communications systems were covered. These topics included the theoretical background of software-defined radio along with specific details of error detection, bit error rate, spectrum sensing, and matched filtering. Lastly, two important tools for the project – the USRP2 radios and the Simulink programming environment – were introduced. How they function was discussed, along with their importance in fulfilling the project's goal. With this theoretical background, the detailed analysis of the project's progression is now possible.

Chapter 3 – Proposed Approach

3.1. Introduction

Before delving into the technical details of how the project was completed, it will be helpful to understand the structure of the overall approach to designing and completing the stated goals. Analyzing the evolution of the design, logistics, problems encountered, and final deliverables of the project provide a broad overview of how it progressed. This chapter will discuss these issues to give a better understanding of the project's course.

3.2. Evolution of Design

Like many projects, this one went through several design stages. The initial goals and timeline at the beginning of A term were very different from the revised objectives midway through C term. This evolution of the overall goal and design of the project was a result of several variables and the realization of the project's scope. In addition, several problems that were encountered throughout the project also forced the readjustment of the goals.

3.2.1. Initial Plans

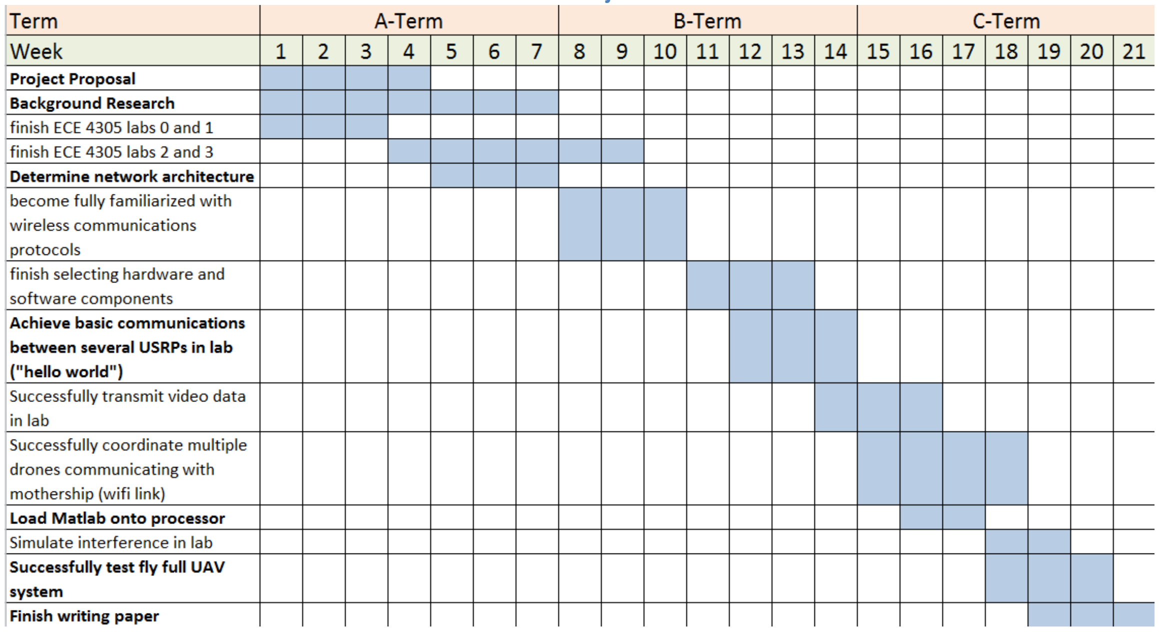
The initial goal of the project was to create a functional wireless network among several UAVs to aid in data collection during SAR operations. This goal seemed plausible because the design was split up among the three MQP teams who would work on communications, hardware, and software. It also matched the initial vision for the project of creating a small fleet of UAVs that were communicating with each other and the user as they carried out a SAR mission. That left this group with ensuring the UAVs had a working communications link between themselves and the base station. This goal seemed feasible over a three term timeframe, so the preliminary timeline was created around this objective. The group’s goal was split up into the separate chronological objectives shown below.

Table 4: Initial Chronological Objectives

First Objective	Second Objective	Third Objective
Theoretical Preparation – become familiarized with communications protocols and software-defined radio	In-Lab Text and Video Communications – use Simulink and USRP2s to send text and video messages	Testing Motherboard on UAV – put Simulink on motherboard, test on mothership

The initial timeline in table 5 had the group learning about SDR prototyping using Simulink in A term for theoretical preparation, achieving radio to radio text and video communications in lab by the end of B term, and then finally loading the system onto a motherboard in a UAV for flight tests by the end of C term.

Table 5: Initial Project Timeline



Term	A-Term							B-Term							C-Term						
Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Project Proposal	▨	▨	▨	▨																	
Background Research	▨	▨	▨	▨	▨	▨	▨														
finish ECE 4305 labs 0 and 1	▨	▨	▨																		
finish ECE 4305 labs 2 and 3				▨	▨	▨	▨	▨	▨												
Determine network architecture					▨	▨	▨														
become fully familiarized with wireless communications protocols								▨	▨	▨											
finish selecting hardware and software components											▨	▨	▨								
Achieve basic communications between several USRPs in lab ("hello world")												▨	▨	▨							
Successfully transmit video data in lab														▨	▨	▨					
Successfully coordinate multiple drones communicating with mothership (wifi link)															▨	▨	▨	▨			
Load Matlab onto processor																▨	▨				
Simulate interference in lab																		▨	▨		
Successfully test fly full UAV system																		▨	▨	▨	
Finish writing paper																			▨	▨	▨

3.2.2. Revised Plans

However, the group encountered a number of issues. First and foremost, it was quickly discovered that completing the entire stated project goal was not feasible within the given timeframe. Designing, building, and testing a functioning communications network between several UAVs and a user is a significant challenge that simply could not be completed within the given amount of time. This problem arose because the design and implementation of each separate section, such as achieving in-lab text and video communications, was much more time intensive than originally anticipated. Issues such as certain programs not working, needing to order additional hardware, and basic troubleshooting contributed to this larger problem. The realization of this unfeasible goal was further enforced once the inherent difficulties of the hardware and software elements of the UAV network were also considered. They encountered similar problems, requiring much of the project to be adjusted. As a result, the end goals had to be scaled back.

Although the group realized it had to change the project goals, it was not immediately apparent what the new goals should be. Instead, the group decided to work in the same direction as planned and attempt to achieve as much of the original objective as possible. By around the end of B term, it was determined that a more realistic goal was to test the Simulink USRP2 communications program in lab and get the motherboard functioning when connected to the radio. An additional "nice to have" objective was having the motherboard take in a file from another computer through Wi-Fi and transmit it to the simulated base station through the created radio link. However, this step was not deemed immediately essential. Once this goal was solidified, working towards it became much easier. The chronological objectives were also updated to reflect the new goal.

Table 6: Revised Chronological Objectives

First Objective	Second Objective	Third Objective
Frame Synchronization – successfully transmit and receive "hello world" text message	Purchase Components – select and buy motherboard, hard drive, Wi-Fi transmitters	Run Simulink on Motherboard – load Simulink on Pico ITX, use it to run USRP2s in lab for proof of concept

The accompanying revised timeline is shown in table 7. While the revised objectives were not as far down the design path as initially hoped, they were much more realistic. The group realized and understood that continuing the design of the communications system would have to be continued by future project teams.

Table 7: Revised Project Timeline

Term	A-Term							B-Term							C-Term							DC-Term						
Week	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
Short Project Proposal	■	■	■																									
Background Research	■	■	■	■	■	■	■																					
Setup Matlab, USRP2 interface	■	■																										
Determine network architecture			■	■																								
finish ECE 4305 labs 0 and 1					■	■																						
finish ECE 4305 labs 2 and 3							■	■																				
become fully familiarized with wireless communications methodology									■	■																		
finish selecting and purchasing hardware components											■	■																
Achieve basic communications between several USRPs in lab ("hello world")												■	■	■														
Improve resiliency, flexibility of USRP2 program														■	■	■												
Setup and troubleshoot hardware																■	■	■										
Load Matlab onto processor																		■	■									
Test USRP2 range in Harrington																				■	■							
Test Matlab on processor																					■							
Write paper																						■	■	■	■	■	■	■

3.3. Project Logistics

In the process of designing this project, the team encountered a number of logistical issues. These issues mainly included the design decisions that had to be made about which components to use and how to pay for them with the given budget.

3.3.1. Design Decisions

The group had to make a number of decisions when designing the groundwork for the UAV communications system. First and foremost was deciding how to create the programs that would actually send a receive data with a radio. MATLAB with the add-on Simulink was chosen for this purpose because of its availability and relative ease of use. Simulink allows the user to create almost any system desirable simply by placing blocks on a model. Its inclusion of a Communications System Toolbox with various relevant blocks was very useful. In addition, the flexibility of the MATLAB programming environment and its ability to work with Simulink provided further use. As for the specific radios themselves, the Universal Software Radio Peripheral (USRP2) was chosen. It was selected for similar reasons: it was readily available and fairly straightforward to use. It also had a specially designed block in Simulink that allowed the radio to be operated from the program over a gigabit Ethernet cable. The program and radio fuse well together, and the fact that they were already provided by the school meant that no funds had to be expended on purchasing them.

The process of choosing the motherboard and accompanying hardware was more challenging. This decision had to be based on its ability to interface with and run the USRP2. In addition, the motherboard needed an accompanying hard drive to store information. The team also had to select a Wi-Fi amplifier to meet the requirement that the drones and mothership be able to communicate at longer range. Lastly, a small amplifier was needed to attach to the USRP2 to boost its transmission range. All of these components are listed in the table below, along with their prices and specific reasons for purchase.

Table 8: Hardware Purchases

Item	Price	Reasons for Purchase
Axiomtek Pico820 Motherboard with 1.5GHz Atom Z530 Processor	$550	• Small footprint: 100x72 mm • Low power consumption: 20 W • Gigabit Ethernet • Sufficiently powerful processor • mSATA connector
ADATA 30 GB SATA Solid State Drive	$65	• Small footprint: 51x30x4 mm • Sufficient memory size • Low price
Ubiquiti Bullet M Wi-Fi Amplifier	$80	• Large range: 3-5 km • Few comparable alternatives • Decent prince
MiniCircuits 13 dB Low Noise Amplifier	$0 (already available)	• Already available • Small and efficient • Sufficient for testing purposes
Aleratec mSATA to SATA SSD Adapter	$25	• Required adapter to connect hard drive to motherboard

3.3.2. Project Costs

Due to the nature of the project, overall costs were relatively low. The majority of the design and testing was performed on available school computers. The Electrical and Computer Engineering Department also provided the MATLAB and Simulink software and the USRP2 radios, resulting in no required spending for these items. The main expenditure for the project was the Pico ITX Motherboard, which was actually quite expensive because of its relatively powerful processor and capabilities. The additional items cost relatively little, and added up to a total of $720. About $700 of this cost was covered by a $6000 grant from the MathWorks for the project and the rest was covered with personal funds. The additional money from the grant was used by the other teams, which required significantly more money for items such as the UAVs themselves.

3.4. Final Project Deliverables

At the end of C term, the deliverables that the team produced were largely in line with the goals set out in the revised timeline. These items were as follows:

- Two functioning Simulink programs that send a receive a text message via USRP2
- MATLAB program that converts given file type (e.g. XML) to bits for transmission and rebuilds file upon reception
- Pico ITX motherboard running Ubuntu with MATLAB and Simulink connected to USRP2
- Long range test of USRP2s in Harrington Gymnasium to demonstrate proof of concept

These deliverables are the basic elements for a UAV communications system. Further testing and design will be required before it can be fully implemented onboard the UAV network, but these items are a large step towards this final goal.

Unfortunately, not all of the originally desired capabilities were included. Upon realization that the initial project goal was not feasible, the objective had to change along with the planned functionality. The main dropped capabilities were video transmission with the USRP2s and Wi-Fi to Simulink program data conversion. The USRP2s, while very flexible in their programming, simply could not sustain a high enough bit rate to transmit video. Once repetition coding and the decimation factor were taken into account, the sample rate dropped from 100 Ms/s to less than 100 ks/s, which is enough for only text and possibly low quality pictures. In addition, the ability to receive a file via Wi-Fi and then route it to the simulated user over the USRP2s also proved unrealistic. This step would require a separate non-MATLAB program to receive the file and put it in the proper directory, all before the Simulink program ran.

This degree of integration involves significant programming beyond the capabilities of the team within the given timeframe. Thus, some of these desired goals must be addressed with any further work on the project.

3.5. Summary

Like many projects, this one evolved over time. The initial goals were unrealistic and had to be scaled back to meet time and resource constraints. As the project progressed, the final objectives became more solidified and feasible. They were accomplished successfully by the end of the project, leaving additional desired objectives as future work. In addition, the overall cost of the equipment was fairly low because much of it was already available and most of the rest was quite cheap. Funding from The MathWorks was able to cover most of the expenses. Thus, team was able to achieve its modified goals while staying within a fairly small budget.

Chapter 4 – Implementation

In this chapter our implementation and the results of the project are elaborated. More precisely the setup of the two main Simulink blocks that were used regarding the auto-frequency offset and the frame synchronization are analyzed. The transition from the simple messages to XML files and the Wi-Fi protocol use is also elaborated. Lastly, the chapter ends with the implementation of the project into the motherboard and the success in transmission and reception of any type message.

4.1. Offset calculation and automatic detection

As in any radio, USRP2's hardware design is characterized by the disadvantage of having offset between the transmitter radio and the receiver radio. The main reason of the frequency offset between the two radios is the relatively inexpensive hardware. In our case, the offset frequency was first measured manually by using the FFT plot from observeFFT.mdl and then we were able to implement an algorithm in order to automatically correct this offset. The Doppler Effect was another issue for the frequency offset between the radios that would be placed in the UAV. Therefore, our algorithm had to be able to detect and automatically correct any offset. An example is illustrated in figures 25 and 26.

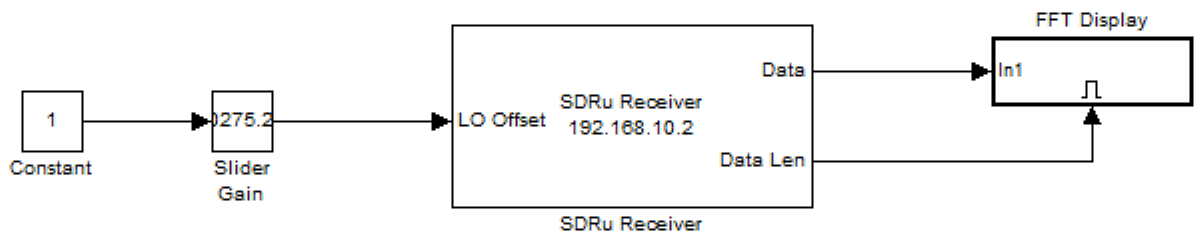


Figure 25: The observeFFT.mdl consisting of the SDRu block receiver and the FFT display block

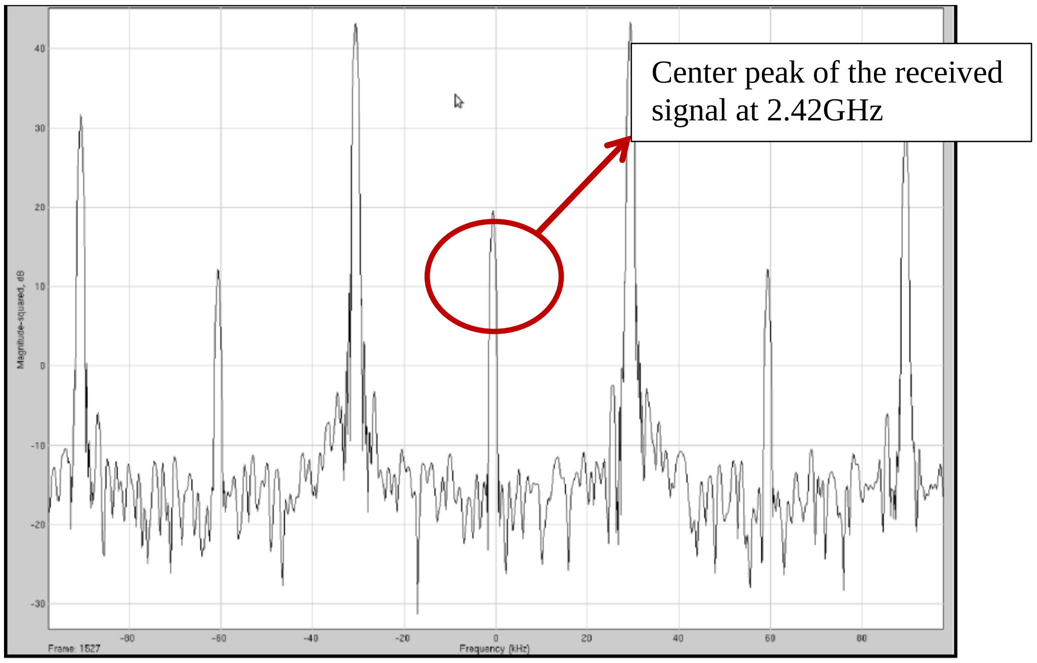


Figure 26: Example FFT plot at 2.42GHz

Another USRP2 was used to run siggen.mdl at 2.42GHz as shown in Figure 27. A 30kHz baseband signal was transmitted, and the baseband FFT was observed on the receiver end. The peak at 30kHz was magnified, and visually inspected to determine the value at which the peak occurred. This was then compared to the expected 30kHz transmitted baseband frequency. Thus using the results of the previous two sections, it was then determined that the offset is effectively given by the formula: $f_{offset} = f_{observed} - 30\text{kHz}$.

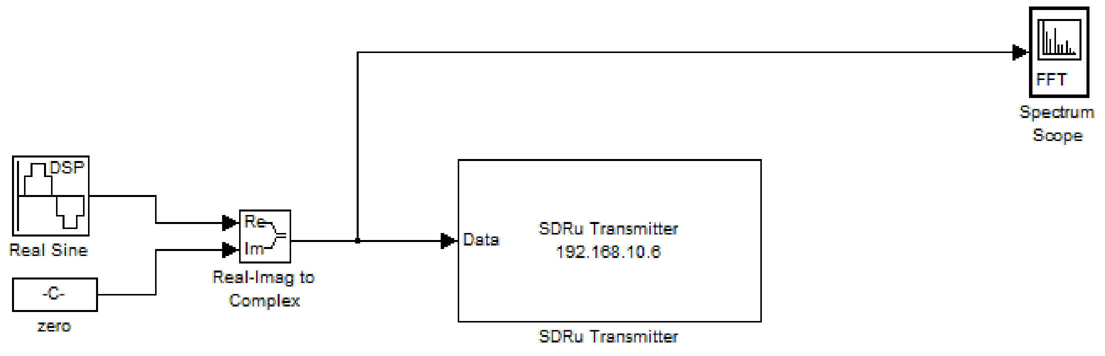


Figure 27: Simulink model of siggen.mdl for a generation and transmission of a signal

Initially in the offset was determined to be 8kHz. This was repeated for several carrier frequencies and a similar offset was observed. Then the entire process was repeated by swapping the roles of the transmitter and receiver. The offset was determined to be -7.8kHz. It was interesting to note that when other devices were paired the offset was different as expected. However, upon powering down and then repeating the experiment with the same pairs, the offset was significantly different. For example offsets below 1kHz were observed using the same pair for which the offset was previously measured as being around 7kHz. This was noted as a practical issue to be wary of, and it was concluded that frequency offsets are inevitable even if transmitter-receiver pairs are compensated at a specific time, as offsets would be time-varying with the same pairs. Also, noting the effects of frequency and phase offsets on the system operation, this must be compensated for.

Using the Simulink models of DBSKTx.mdl and DBPSKRx.mdl as shown in Figure 28, the effect of frequency offset on digital communications was investigated. Frequencies ranging from -30kHz to +30kHz were added to the transmitter center frequency, to simulate the transmitter-receiver carrier frequency offset. These were stepped in increments of 10kHz. For ease of pattern recognition, alternating 1's and 0's were sent at the transmitter, and the demodulated receiver output was plotted. In theory these could be used to determine average error rates, since it would be expected that there be a series of alternating ones and zeros in the receiver output. However instead of calculating approximate values for this, visual inspection of the stem plots was used to gauge the impact qualitatively. The rationale for this was firstly because it was not possible to know which bit at the transmitter mapped back to the receiver, in terms of the time offset. Also, even if an error rate was calculated for this exercise, it would only be for limited transmitted patters, and may not necessarily be used for generalization. However, the stem plots provide a quick visual approach to qualitatively observe the impact of frequency offset and simulating device mobility.

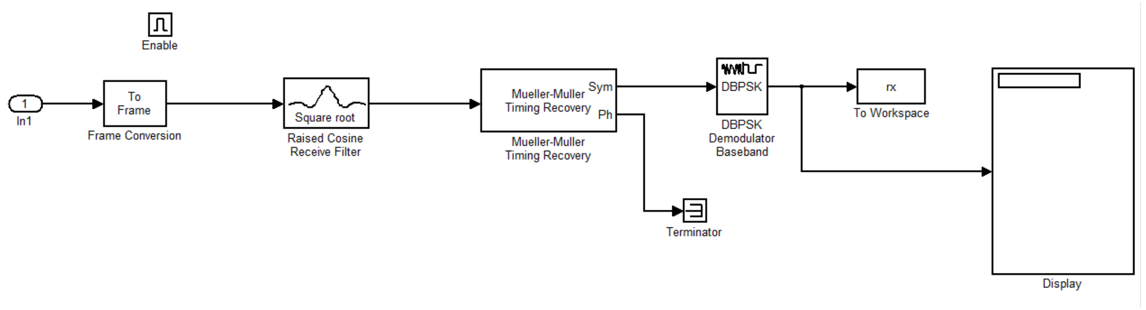

Enable
1
In1
To Frame
Frame Conversion
Square root
Raised Cosine Receive Filter
Mueller-Muller Timing Recovery
Sym
Ph
Mueller-Muller Timing Recovery
DBPSK
DBPSK Demodulator Baseband
rx
To Workspace
Terminator
Display

Figure 28: DBPSKRx.mdl; DBPSK protocol implementation on the receiver side

Figures 29-31 provide examples of the plots obtained, taking consideration of the comments above.

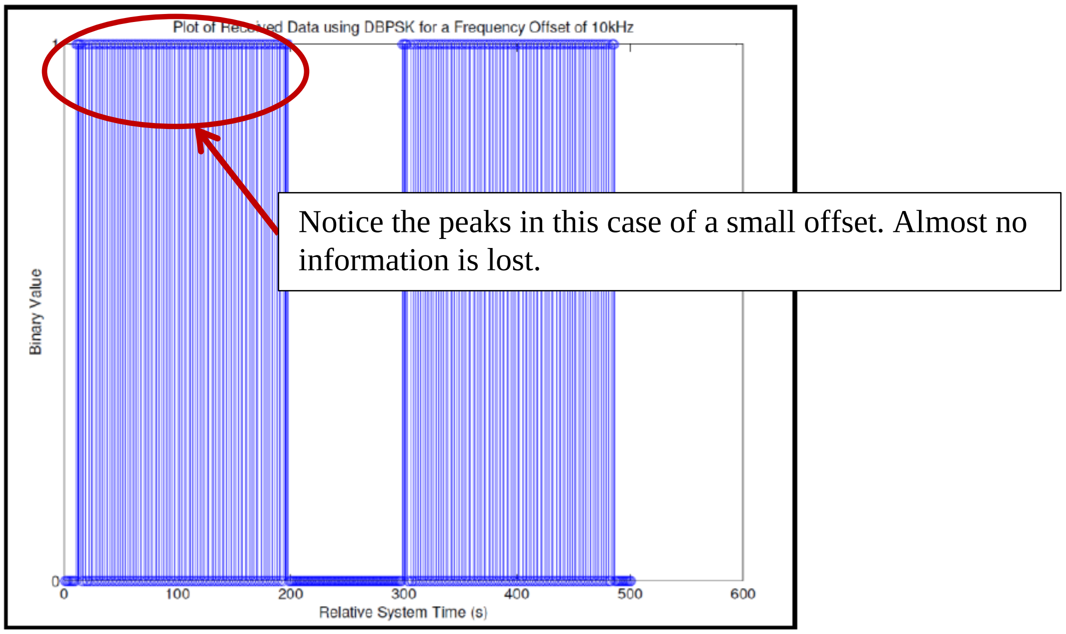
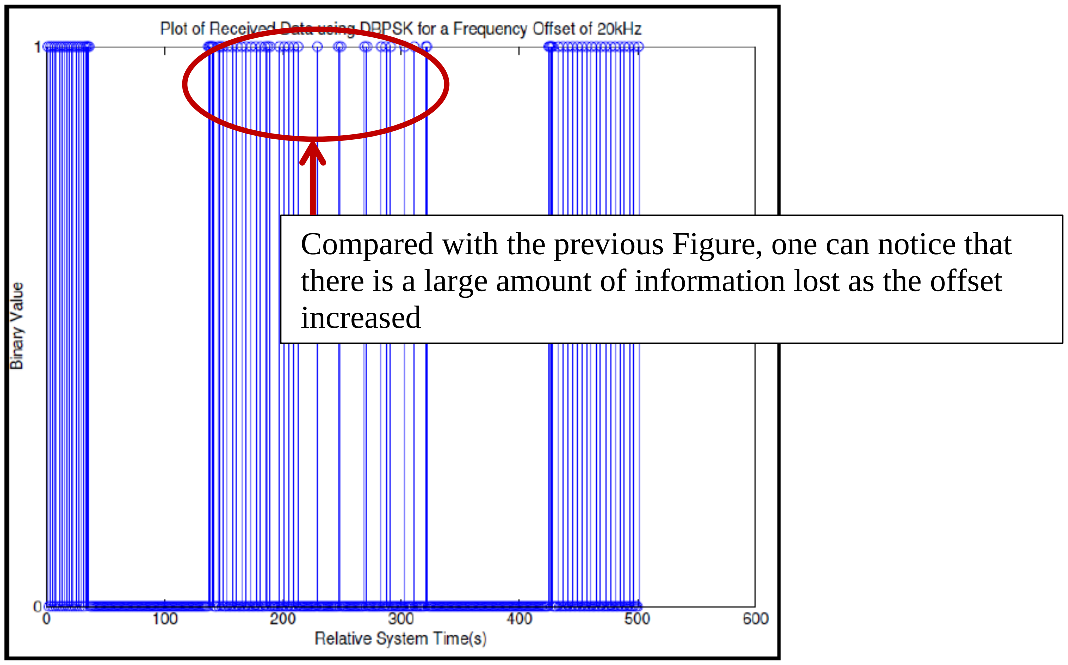
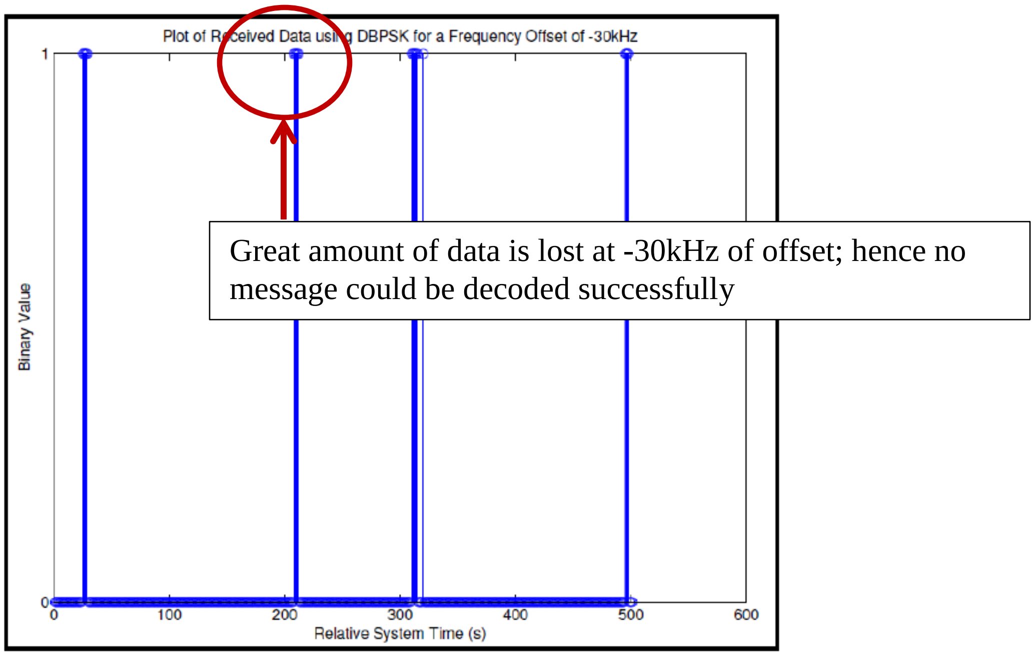


Figure 29: Example Plot of Received Data for DBPSK receiver with 10kHz offset



Compared with the previous Figure, one can notice that there is a large amount of information lost as the offset increased

Figure 30: Example Plot of Received Data for DBPSK receiver with 20kHz offset.



Great amount of data is lost at -30kHz of offset; hence no message could be decoded successfully

Figure 31: Example Plot of Received Data for DBPSK receiver with -30kHz offset

The general trend noticed was that increasing the frequency offset generally caused increasing lost information. This was the case since whenever a window was opened the overrun scope went high (i.e. from 0 to 1). Even though the overrun scope was mostly in the low state indicating no overloads, there was still much lost information. There was not much variation due to simulating limited mobility within the lab located in AK227, but it was also noted that when the lab was busy (C term 2012 when ECE4305 was offered), the errors were higher. This might have been the result of interference between the devices.

Our final design can be seen in Figure 32 and also in tables 9 and 10, where we highlight its key features and the initialization parameters.

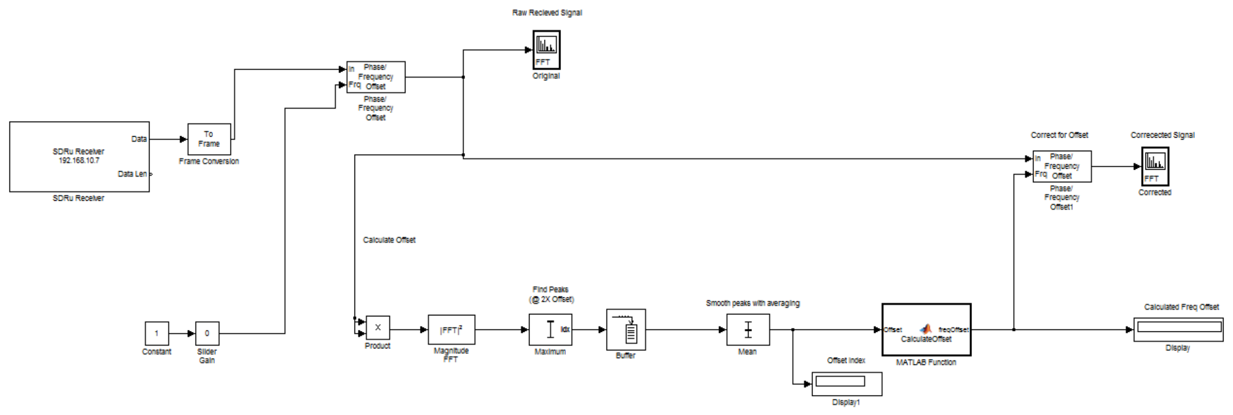
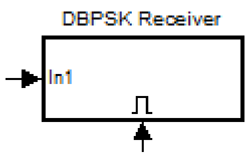
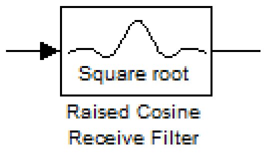
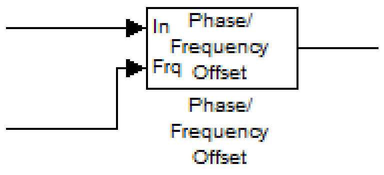


Figure 32: Complete auto-offset Simulink block that was designed for this project

Table 9: Functionality of Simulink blocks in the auto-frequency model

Main Blocks	Function
DBPSK Modulator Baseband 	Modulate the input signal using the differential binary phase shift keying method.
Raised Cosine Transmit Filter 	Upsample and filter the input signal using a square root raised cosine FIR filter.
Insert Frequency Offset  In Frq Phase/ Frequency Offset Phase/ Frequency Offset	Apply a frequency offset to the input signal with the purpose of simulating frequency offset caused by inaccuracies in the oscillator circuits in the RF end.
Square of the signal	Take the square of a signal, so the FFT of the

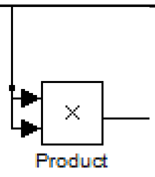
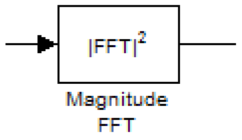
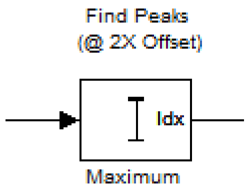
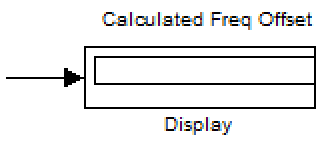
	received signal will be shifted double of the frequency offset.
Magnitude FFT 	Compute magnitude-squared FFT of the signal with offset after taking a square.
Maximum 	Returns the value and index of the maximum elements of the input signal coming out from Magnitude FFT block. Indices are the locations of maximums, which should be equal to double of the offset in this system.
Offset Display 	Numeric display of input value which is the estimated offset we calculate from the receiving side.

Table 10: Simulink initialization parameters for the auto-frequency offset model

Parameter	Value
Interpolation	500
Oversampling	2
Fs1 (Relevant sampling frequency)	10^8/interpolation
Fs2 (Symbol rate)	Fs1/oversampling
Bitrate	Fs2/bitsPerSym
Center frequency of USRP2 Tx/Rx	Set to 2.42 GHz
Frequency scaling factor (1024 point FFT)	$2048/F_S$

In addition to the initialization parameters, the key to the correct frequency representation was the conversion of the FFT indices into actual frequency units. To illustrate the issue, a plot of the MagFFT block output is provided in Figure 33. As seen there is no link in the figure, relating the index to a frequency value as yet. Also the Max block output should start counting from 0, not 1; and the frequency scaling used for division in the last block, must be adjusted. Since a 1024 point FFT was used, the indices of the FFT vector represent increments of Fs /1024. Thus the constant used for frequency calculation was changed to 2048/ Fs.

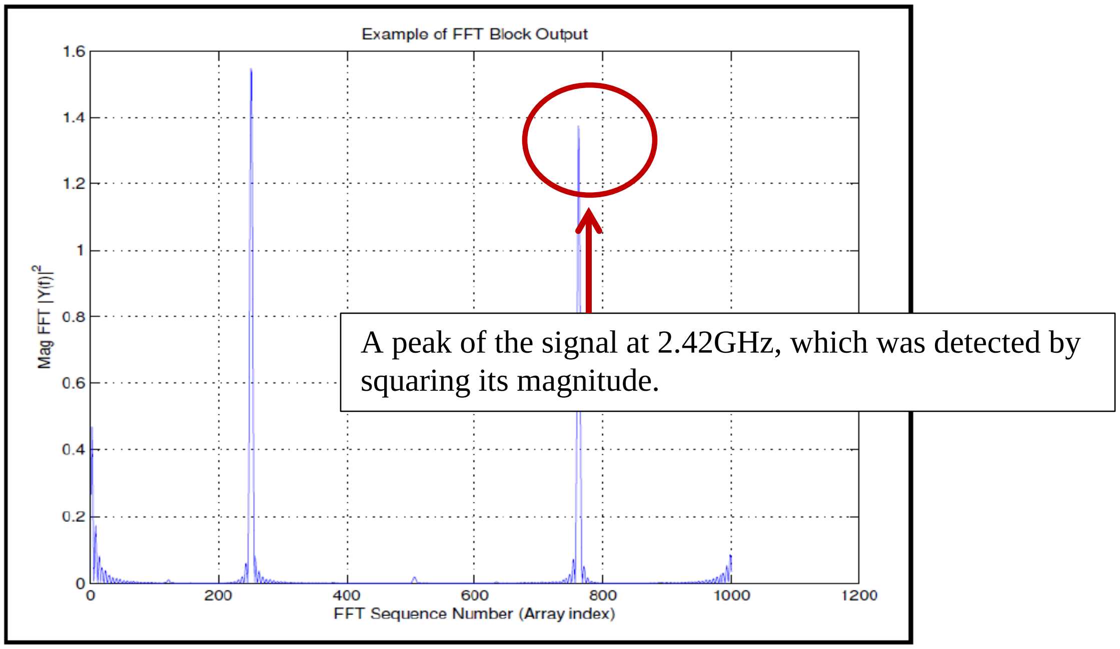


Figure 33: Example of MagFFT block output, not adjusted to actual values.

Using siggen.mdl, various frequency offsets were simulated on the transmitter end. It must be noted that the transmitter offsets are the values input into the software to offset the transmitter center frequency from the receiver center frequency. The actual offset from the receiver must also be incorporated into these in the estimate. The AFOC was then manually observed to determining the frequency offset. This was recorded in addition to the displayed estimate. These are all plotted in Figure 34.

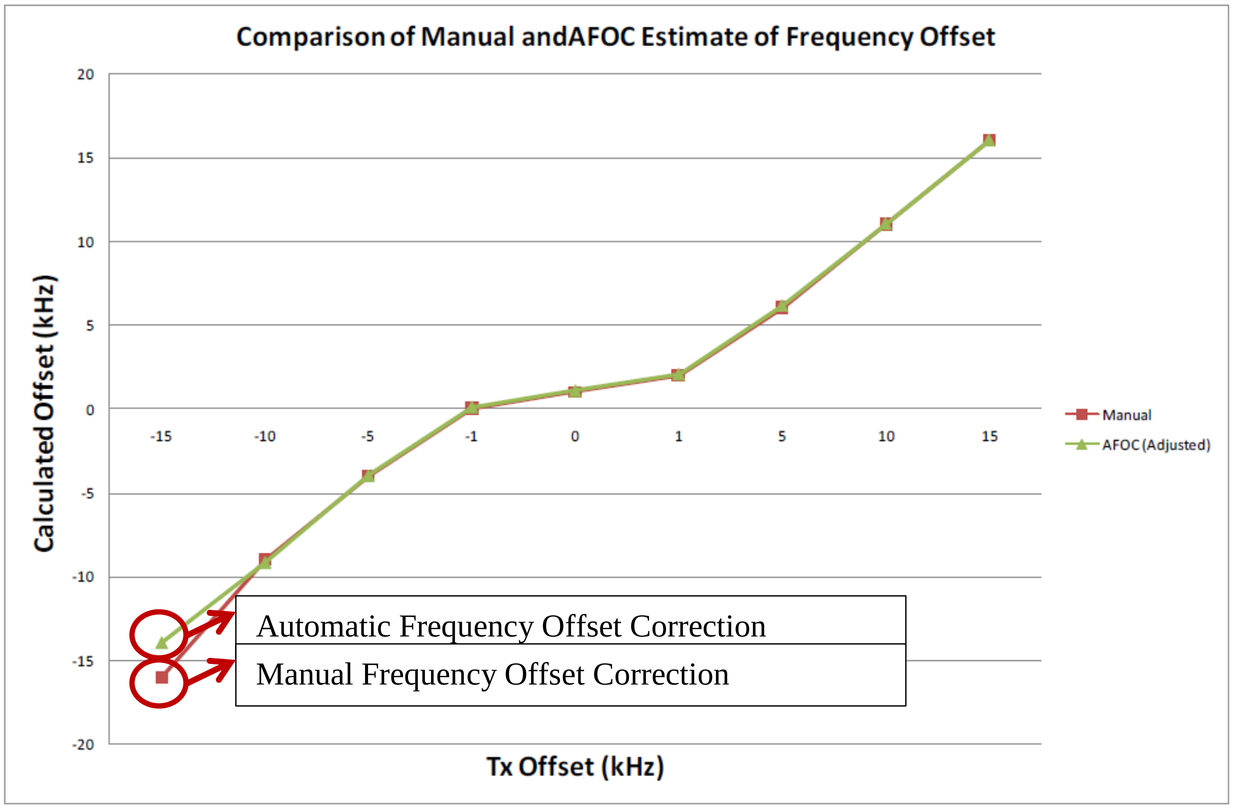


Figure 34: Auto-frequency offset error performance for various offsets

4.2. Performing Frame Synchronization

Frame synchronization is the process in the telecommunications transmission system to align the digital channel (time slot) at the receiving end with the corresponding time slot at the transmission end as it occurs [55]. For example, one transmits a packet which contains numerous frames. At the receiver side, you want to know where a specific frame actually starts, and then one needs to implement frame synchronization.

Frame synchronization involves the following steps: In the first step, the transmitter injects a fixed length symbol pattern, called a marker, into the beginning of each frame to form a marker and frame pair, which is known as a packet. Packets are then converted from symbols into a waveform and transmitted through the channel [55]. The receiver detects the arrival of packets by searching for the marker, removes the markers from the data stream, and recovers the transmitted messages. Marker detection is the most important step for frame synchronization.

The USRP2 solution consists of two aspects: the transmitting node and the receiving node. Figure 35 illustrates the model for the transmitter.

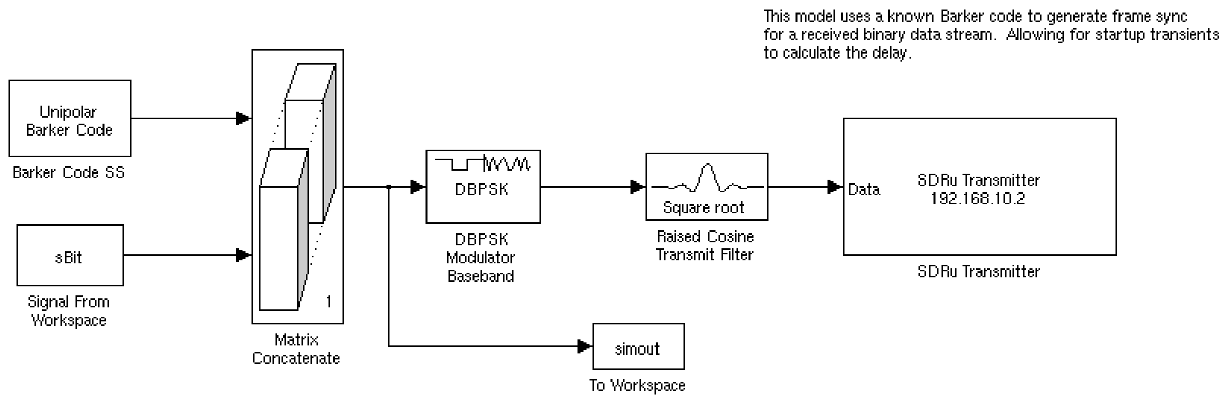

This model uses a known Barker code to generate frame sync for a received binary data stream. Allowing for startup transients to calculate the delay.
Unipolar Barker Code
Barker Code SS
sBit
Signal From Workspace
1
Matrix Concatenate
DBPSK
DBPSK Modulator Baseband
Square root
Raised Cosine Transmit Filter
Data
SDRu Transmitter 192.168.10.2
SDRu Transmitter
simout
To Workspace

Figure 35: Simulink model for frame sync (transmitter)

The system consisted of six blocks as shown. The Unipolar Barker Code subsystem generated the synchronization bits at a rate of 13 bits in the entire frame sampling period that is sent to the concatenation and padding block to the right. The sBit block accepted any messages which were to be sent. Since we wanted the input of the DBPSK modulator block to be a 179x1 frame each sample time, the concatenation and padding block was coded to essentially generate the required frame. The code for this block is presented in the Appendix. The intent for this block is to expand it such that all required frame construction for the model is done here. Thus we also decided that in future work we would also consider implementing the other framing functions (e.g. adding addresses, framing and error detection/correction codes, additional synchronization information, as well as fragmentation and reassembly information if necessary). Table 11 presents a listing of the blocks and summarizes their functionality.

Table 11: Functionality of blocks used in the Frame Sync model

Main Blocks	Function
Unipolar Barker Code Generator	Generate Barker code of length 13, used for frame synchronization. The sample rate was set to 1/13 samples per frame. Note that this is subsequently normalized based upon the USRP2 Tx sample rate. Note that this should not be confused with the digital sampling rate. In fact the rate specified here is the rate at which Simulink blocks' states and outputs are updated.

DBPSK Modulator Baseband	Modulate the input signal using the differential binary phase shift keying method.
Raised Cosine Transmit Filter	Upsample and filter the input signal using a square root raised cosine FIR filter.
Signal from workspace	This was used to generate the data required for transmission.
Concatenation and Padding	Add the 13bit Barker code to the start of the frame, and pad to the end of the frame with zeros if it is less than the frame samples frame-size.

The next aspect was the receiver. The receiver model is illustrated in Figure 36. The main blocks included the auto-frequency offset which compensated for frequency offset. This has been discussed previously. In this case, it was not known when the first piece of data received actually began in the frame. When the receiver code was initiated, it could pick up data starting in any part of the frame. Thus in our case we used the correlation against the barker code to determine the instants at which the synchronization sequences existed, and ignored the data before the first case. Model initialization parameters for both transmitter and receiver are given in Table 12.

Table 12: Initialization parameters for the frame synchronization model

Parameter	Value
Interpolation	500
Oversampling	2
Fs1 (Relevant sampling frequency)	10^8/interpolation
Fs2 (Symbol rate)	Fs1/oversampling
Bitrate	Fs2/bitsPerSym
Center frequency of USRP2 Tx/Rx	Set to 2.42 GHz
Frequency scaling factor (1024 point FFT)	2048/F_S

.

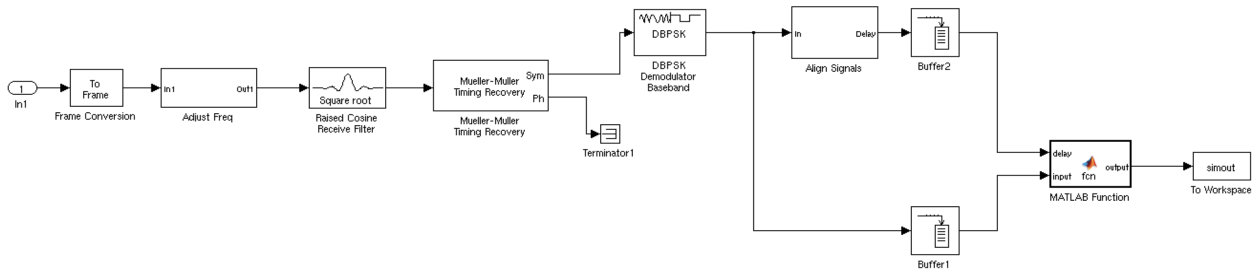


Figure 36: Frame Sync Simulink model (Receiver).

4.2.1. Performance Evaluation

According to our results of successfully decoding the message, the success rate was generally below 5% without frequency offset compensation. We thus thought we should examine the reasons behind the rates being achieved by our other class mates as well as ourselves. There was a 2-4% increase in the performance when we tried varying the various parameters from Table 12, with the exception of the data rate.

4.2.1.1. Unexpected Frame Lengths

We decided to investigate this further since we were getting very low success rates. We realized that the max function only retrieves the first instance of the maximum occurring in the argument for the function. Since we buffered a few received frames and visualized them, we noted that the correlation was 13 at the frame start points, and this was repeated almost periodically, except in a few instances where perhaps there was noise and a barker code got garbled. We essentially detected all frame starts in our received sequence, and found the difference between successive start points. An example of this is presented in Figure 37. As seen in the figure, on average there were about 230 bit samples between each successive detected frame in the data stream. The higher spikes corresponded to instances where perhaps the frame synchronization was lost and a frame thus seemed longer than it really was. We expected the number of bits between frames to be about 179. We double-checked our models, and they did not suggest an alternate frame length was erroneously input. Thus we decided to try other USRP2 devices with our code. We got different frame lengths, but the same result as with our own. Essentially, the frame lengths seemed to spread out at the receiver. We thought that perhaps we could resample, by low-pass filtering followed by sampling to obtain the required frame sizes, in case there was some sort of temporal spreading.

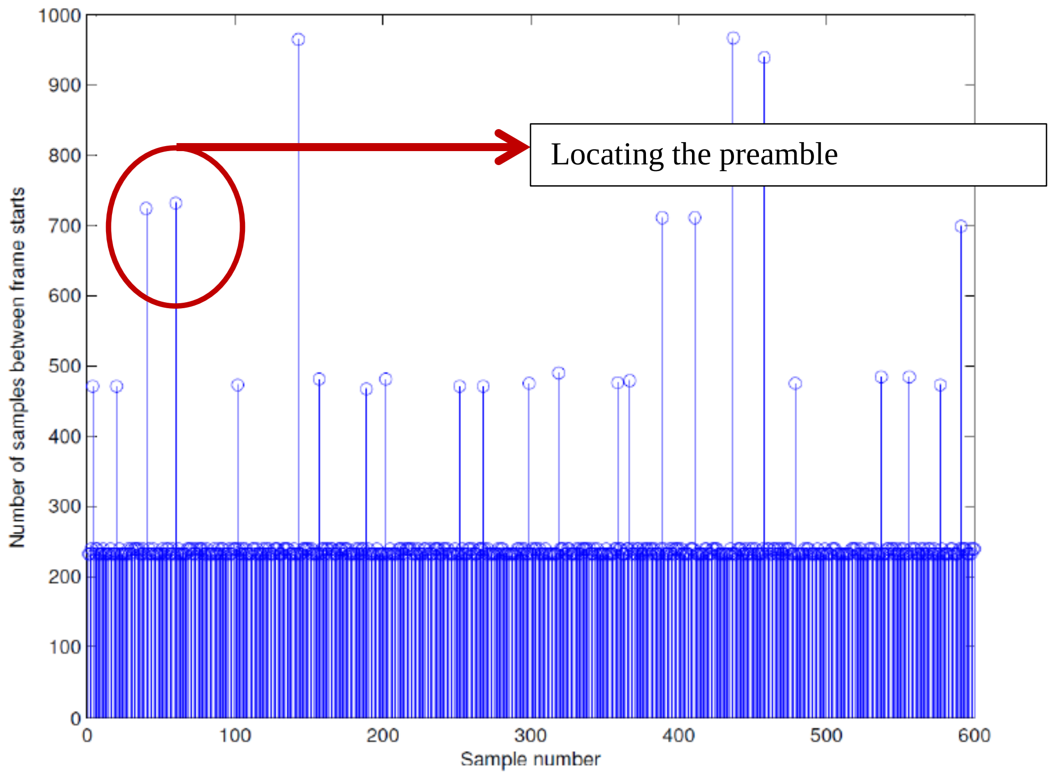


Figure 37: Number of samples between detected frame synchronization points

We then thought that maybe that would not work because we were able to pick up the frame starts, and the values for the correlation were 13 which corresponded to what we expected for the 13 bit barker code. Thus if the barker code was exactly 13 indicating that we were able to pick it up, then we assumed the remainder of the frame was also not spread temporally. Thus we wondered what could possibly cause the spacing between received frames to be more than 179.

4.2.1.2. Adjusting Data Rates

To investigate this further, we looked at the transmitting data rates. What we observed was that we got better performance when we transmitted at slower data rates. To better understand this we referred to the transmitter and receiver models from Figures 27 and 28 to determine what was going on. The USRP2 DAC samples the data at 100Msamples/second. By adjusting the interpolation we can adjust the rate at which the USRP2 samples the data at the input of the USRP2 Transmitter block. It must be noted that the definition of sample time in Simulink refers to the times at which the block outputs and their states change. By looking at the sample times we realized that at the transmitter, the typical rate was around 0.00179s. This meant that the blocks changed their outputs at this rate.

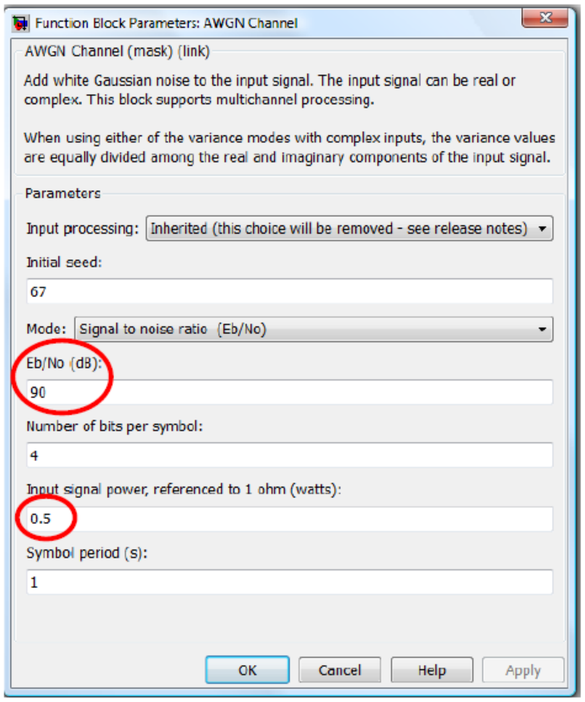
Now after passing through the Raised Cosine Transmit Filter block the system essentially increases the frame size from 179 to 358 elements. This is held at the USRP2 block input for 0.00179s before it changes again. Since the USRP2 interpolation was set to 500, the sampling rate at the DAC essentially transmitted the data at 200,000 samples per second, where each sample was a frame of 179 bits. This means that the frame was transmitted roughly 200,000 times in one second. Hence, in theory it may be possible to get a raw rate of 35.8Mbps ignoring the other practical issues. However, realistically we know that this is not the case, since details of the USRP implementation will determine the threshold for the rates. Furthermore, since the data at the input of the USRP block does not change for 0.00179 seconds, what we in fact get is that we can send the a maximum of around 200,000 frames for each change in the Simulink sample time. Since between Simulink sample times the data is latched at the USRP block input, this restricts the data transmission to the times at which the data changes.

When we adjusted the rates via the block, we were able to get varying rates of success at the receiver. We were able to have a form of repeater coding inadvertently achieved due to the sampling rate of the USRP2 device sending multiple samples of the message between Simulink sample instants. It must be noted that this benefit would also depend upon the rate at which samples from the USRP2 receiver block are input into the receiver model in Figure 28. This depends upon the sample time for this block, which determines the rate at which samples were logged. Thus there is a pairing between the interpolation and decimation of the USRP2 blocks which in theory can be used to adjust the data rates.

4.2.1.3. Bit Error Rate Results

We examined the typical performance of the system under different SNR values, in 5dB increments from 0 to 75dB. First we set the interleaver so that its output mirrored the input, using the statement y = u, where y was the output and u was the input. This was then done for the de-interleaver. These were plotted for various repetition rates to compare, as shown in Figure 40.

To change the SNR it was noted that the Bernoulli generator produced equally probable 0’s and 1’s. Thus the input symbol power was calculated from (Wyglinski 2011) as 0.5 for 16-QAM, when normalized for an amplitude of 1. Then the AWGN channel parameters were adjusted to change the SNR values. Since 16-QAM was used, the number of bits per symbol was let at 4. Also with no loss of generality the symbol period was left at 1. Note that Simulink would also determine the noise variance based upon our specification of the E_b/N_o ratio. An example of the configuration is shown in Figure 38 below. Figure 39 shows the port properties when a repetition code rate of 2 was used.



Function Block Parameters: AWGN Channel

AWGN Channel (mask) (link)

Add white Gaussian noise to the input signal. The input signal can be real or complex. This block supports multichannel processing.

When using either of the variance modes with complex inputs, the variance values are equally divided among the real and imaginary components of the input signal.

Parameters

Input processing: Inherited (this choice will be removed - see release notes)

Initial seed:
67

Mode: Signal to noise ratio (Eb/No)

Eb/No (dB):
90

Number of bits per symbol:
4

Input signal power, referenced to 1 ohm (watts):
0.5

Symbol period (s):
1

Figure 38: Example configuration of AWGN block to test repetition coder/decoder performance

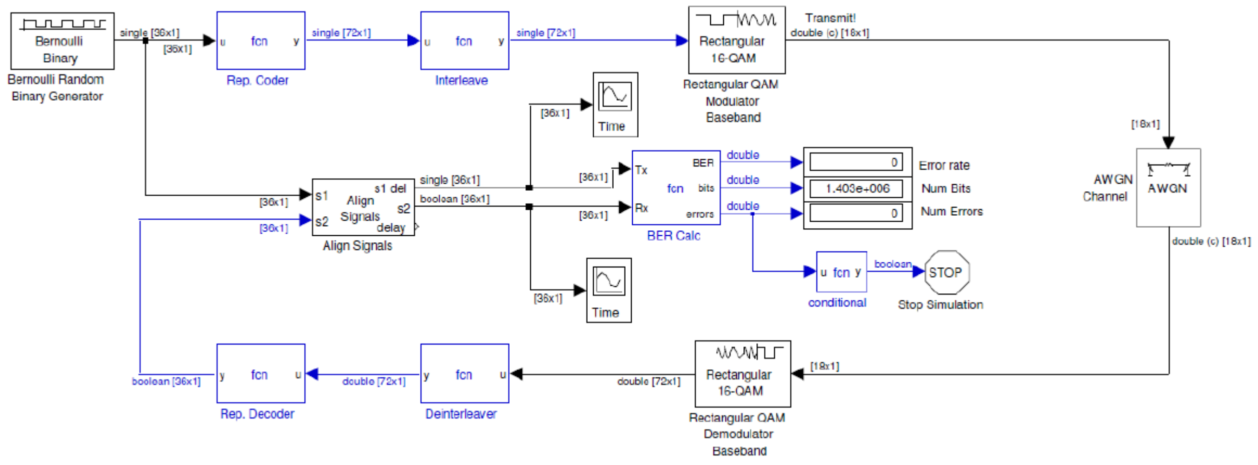


Figure 39: Simulink model showing port properties for a repetition rate, k = 2

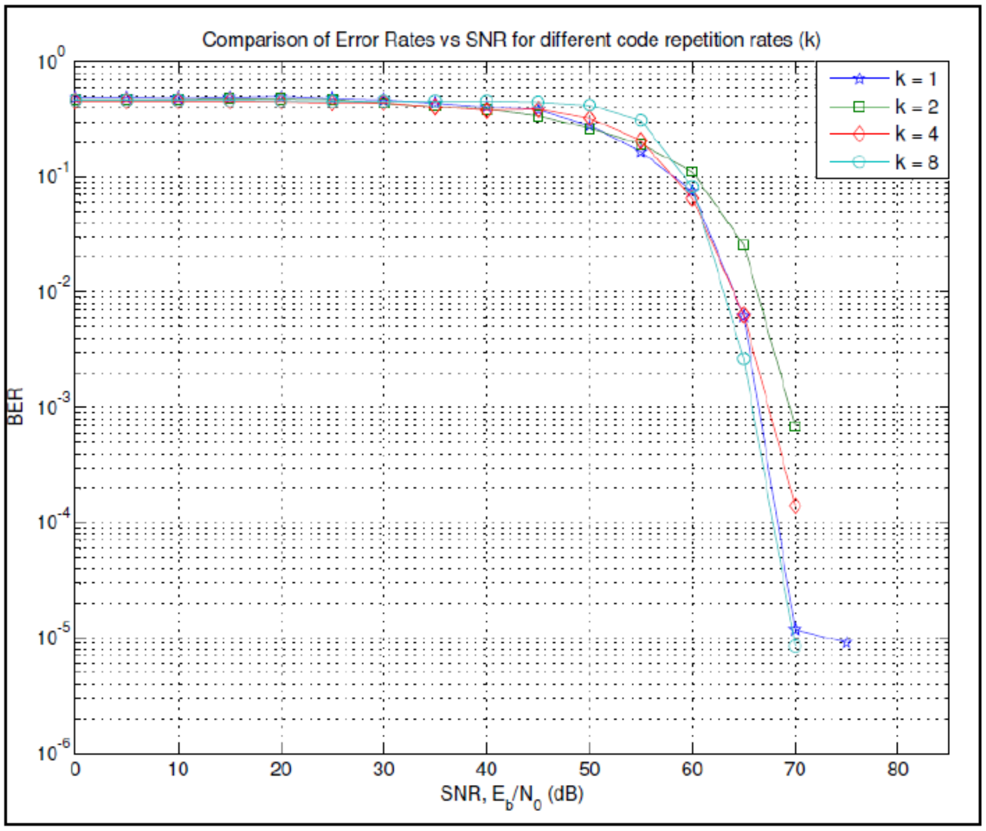


Figure 40: Error performance of different code repetition rates in AWGN channel

Observing Figure 40, there was not a significant improvement in the error performance (less than 1dB of error improvement) for a given SNR, by increasing the code repetition rate. Without code repetition (i.e. k = 1), for a given SNR, in general the performance was better than with code repetition. However for k > 1, it was seen that for SNR's above 60dB increasing k improved the performance. However for lower SNRs (i.e. when better error performance would be desirable), the performance was comparable for the three cases examined. This did not fit well with the expected calculations, but can be explained by first noting that the performance would tend to increase for much larger code repetition rates. However, this would be impractical and a waste of channel bandwidth in transmitting so much redundancy.

Additionally, the trend can be explained by noting that if there were noise bursts greater than the code repetition rate, it can affect the error performance gained by code repetition. Thus, again it seems that only by increasing k to impractical values can significantly improve the performance. In fact, if the noise burst durations could be characterized, then the value for k should be at least twice the expected duration of bursts, to counteract them by averaging out the corrupted bits. This is further supported by considering the minimum Hamming distance for a k-rate repetition code as discussed in [71]. Since there are two valid code-words (i.e. all ones and all zeros), then the minimum distance will be k, and the maximum number of bit errors which it can correct, will be approximately 1/2 r.

Another thing to note would be that the performance of the coder may actually be different in different channel models. Intuitively, repetition coding may work better to counteract channel fading as opposed to the effect of AWGN. Clearly the extra redundancy can assist with errors of certain types (more on this in the next section). At the receiver end, the decoder essentially reduces the rate by a factor of k, removing the added redundancy. Compared to other approaches such as convolutional coding and block coding, this method is extremely simple, and its error performance is not as good. Thus it can be concluded that repetition coding presents a mediocre compromise between error performance and data rate compared to other coding schemes, and the main advantage over these alternatives, is its ease and simplicity of implementation. The above thus summarizes the tradeoffs in choosing a repetition rate, as well as repetition coding itself.

As discussed previously the repetition coder does not provide great performance by itself, and in AWGN channels the presence of burst errors limits the error performance of simple code repetition. Errors usually occur in bursts and are not independent. As shown above, if the number of errors exceeds 1/2 the repetition rate, k, then the error-correcting capability is compromised and it may not be possible to recover the original code word. Interleaving permutes the bits, by some pre-determined form of systematic shuffling (i.e. depending on the interleaving algorithm) so if burst errors occur, the errors are more uniformly distributed and can thus improve the error performance. The interleaver and repetition coding effectively repeated each frame k times. For example 101 produces the 12-bit sequence 101101101101, for k = 4. Figure 41 presents a comparison of k=4 plus interleaving versus no interleaving for k = 4 and k = 8.

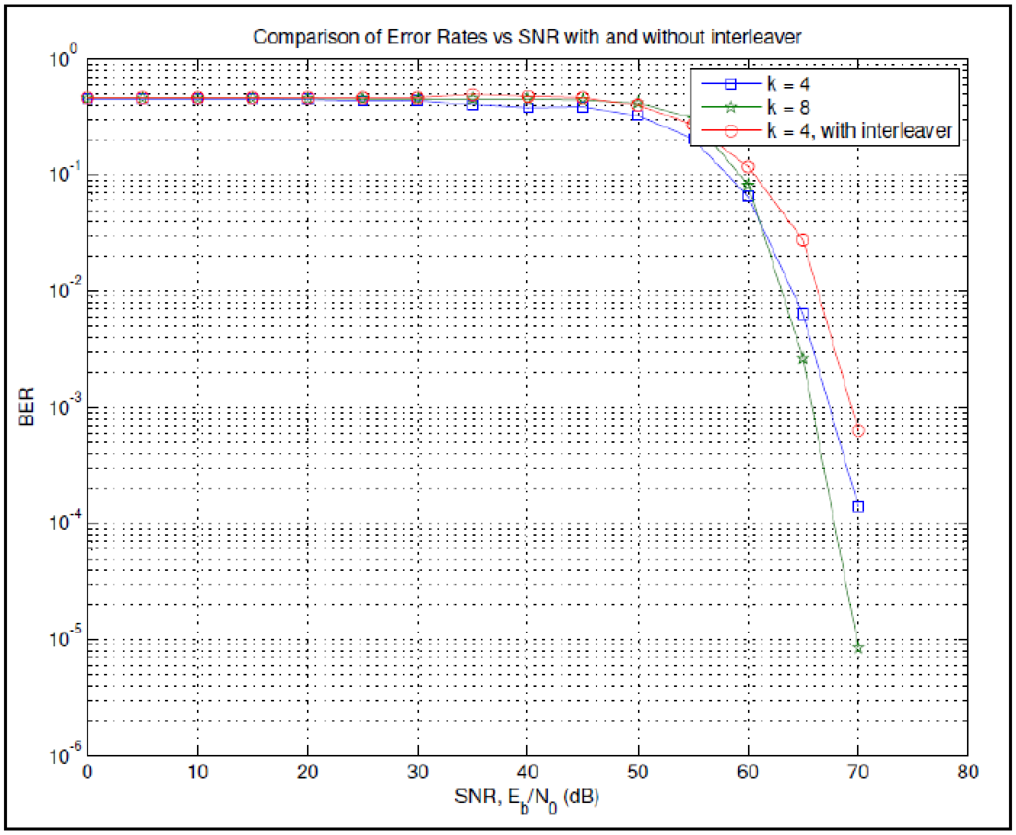


Figure 41: Error performance of different code repetition rates compared to interleaver addition in AWGN channel

As seen from the graph, there was also no significant improvement in the error performance when the interleaver was added. Observing Figure 41, there was not a significant improvement in the error performance (less than 1dB of error improvement) for a given SNR, by increasing the code repetition rate to 8 or by adding interleaving to the k = 4 coding. It was seen that for SNR's above 60dB increasing k improved the performance, while interestingly interleaving reduced it. However for lower SNRs (i.e. when better error performance would be desirable), the performance was comparable for the three cases examined. This did not fit well with the expected calculations, but can be explained by first noting that the interleaver used was very simple. More complex interleavers or decoders can capitalize upon knowledge of the error structure improving error performance. As before, it must be noted that in other non-AWGN channels interleaving may work better. These issues can be further examined, but due to the limited course time, are just noted as areas for further investigation, perhaps in the project. Because of the need to store the bits of a frame for interleaving and de-interleaving, it can be seen that these techniques can potentially improve error performance at the expense of added latency. This represents a tradeoff of this technique. However this would hold for many other coding schemes which all rely upon some form of framing or are block-based (e.g. convolutional coding and block coding schemes).

Another interesting observation was that whenever the simulation was run over, the results were generally the same unless the seed was changed. Thus for each run the seed was changed. When this was done, it was observed that successive runs for the same SNR value produced significantly different results. This hinted that perhaps the simulation cut-off was too short, or that the variation in the performance would change for different runs. Thus insight was actually incorporated into the process for data collected for each scenario at each SNR value. Hence, average values were used for several runs, and it was these that were plotted in the graphs in Figures 40 and 41 above.

When the input was connected to both inputs of the BER calculator, the BER was always 0, no matter how long the simulation was run. With reference to the MATLAB documentation persistent variables store data even after functions are exited. This provides a way to accumulate data and track states, at periodic function calls each sample instant or asynchronous function call.

If one of the inputs is inverted, and the AWGN is removed, then the BER is 1 (i.e. all bits are in error). As seen in [71] and [72], different constellations correspond to different modulation schemes, and this may translate into different error performances. As seen in [73], 16-QAM outperforms 16-PSK, and 16-PAM for a given SNR in terms of E_b/N_o ratio. Thus if the constellation diagrams were changed (i.e. different modulation schemes) it is expected that the error performance would be different.

4.3. Hardware Selection

After progress had been made in transmitting a text message from one computer to another via the Simulink-USRP2 interface, the actual hardware on which this system would be implemented needed to be selected. Work on the Simulink program and improving its resiliency would continue, but the team had to now also work on selecting, purchasing, and testing the equipment. This was a vital part of the project because it is this hardware that would eventually be placed on the UAVs and run the overall communications system.

First, the team had to decide upon which hardware components would be needed for the system. Because the team was only responsible for the communications, this included all equipment that would be involved with this process. Overall, the system would be required to transmit data from the drones to the mothership, have the mothership compile and prioritize the information, and then transmit it to the base station. This process would thus need hardware capable of a drone-mothership link, signal processing, and a mothership-base station link. The separate hardware components were selected to meet these specific needs.

To enable the drones to communicate with the mothership, a basic Wi-Fi link was used. Because Wi-Fi has a limited range, an amplifier and antenna was needed to increase the transmission distance. The Bullet M Wi-Fi radio amplifier was selected for this application. An example of the Bullet M can be seen below.

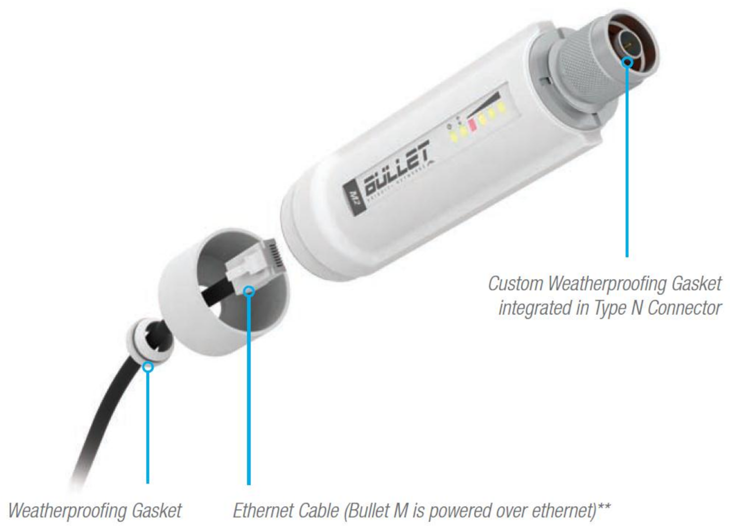

Custom Weatherproofing Gasket integrated in Type N Connector
Weatherproofing Gasket
Ethernet Cable (Bullet M is powered over ethernet)**

Figure 42: The Ubiquiti Wi-Fi Bullet M [69]

The Bullet M weighs a light 0.18 kg, which is important when trying to cut weight on a UAV. It consumes up to 600 mW of power, has a data rate of 100+ Mbps, and a range of several kilometers [69]. These specifications made the Bullet M unique among Wi-Fi amplifiers, enabling it to have a relatively large range with low power and weight requirements. The Bullet M was the ideal choice for the drone-mothership communications. However, due to budget constraints, the team was only able to purchase one unit for $80. It is sufficient for testing, but further funding will need to be acquired in order to purchase more units and install them on all the UAVs.

The next important purchase item was the motherboard. This component was required for compiling all the incoming data from the other drones into one data stream that could be sent to the base station using software-defined radio. Specifically, the Simulink program the team wrote would carry out much of this task, which meant that the motherboard would need sufficient processing power to run MATLAB and Simulink and a gigabit Ethernet port to connect to the USRP2. After much searching, the team settled on the Axiomtek Pico820 motherboard, as seen below.

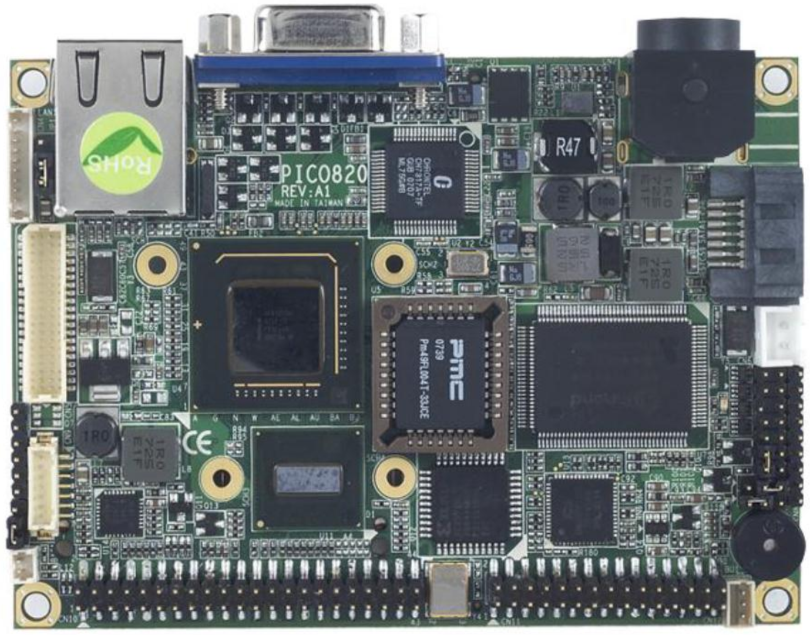


Figure 43: Pico ITX Motherboard [66]

The board was a small 100mm×72mm requiring a standard 5V power supply. It possessed a 1.6 GHz Intel Atom Z-series processor and 2 GB of RAM. These specifications meant that it would be capable of adequately running Simulink for the desired application. In addition, it had numerous input and output ports, including 4 USB 2.0 slots and a gigabit Ethernet port [66]. This Ethernet port was an important requirement in order for it to interface with the USRP2. The board was purchased for $550, making it the largest expense. However, only one was needed to place on the mothership; separate computers would perform any required computation on the drones and base station. Thus, this Pico ITX would meet all of the required processing needs for the communications system.

Two additional components were the solid state drive and the radio amplifier. The drive was needed as storage for the motherboard and to complete the fully-functioning computer that would run Simulink. The team selected the ADATA 30 GB mSATA Internal Solid State Drive. It was chosen because of its large storage capacity, small 51mm×30mm×4mm footprint, high 280 MB/s data transfer rate, minimal 1.5 W power consumption, and reasonable $65 price [68]. A picture of it can be seen below.

Figure 44: ADATA 30 GB Solid State Drive [68]

The other additional component, the radio amplifier, was needed to provide additional power to the USRP2 and increase its transmission range. The ECE Wireless Laboratory already possessed such an amplifier, the MiniCircuits ZX60-33LN Low Noise Amplifier, so that is the one the team selected. It had a bandwidth of 50 to 3000 MHz, a gain of 16.5 dB, and a required DC voltage of 5.5 V [67]. These specifications, along with its small size, made it ideal for increasing the USRP2 range. In addition, using the one already supplied by the ECE department saved the team the $80 on the price. The amplifier can be seen below.

Figure 45: MiniCircuits ZX60-33LN Amplifier [67]

A table detailing all of these specifications for the different components can be seen in table 13.

Table 13: Hardware Specifications [66], [67], [68], [69]

Ubiquiti Bullet M	• **Weight: 0.18 kg** • **Power: 600 mW** • **Data rate: 100+ Mbps** • **Range: several kilometers**
Axiomtek Pico820	• Size: 100mm×72mm • Power supply: 5 V • Processor: 1.6 GHz Intel Atom • RAM: 2 GB • Ports: gigabit Ethernet, 4 USB 2.0
ADATA mSATA Solid State Drive	• Capacity: 30 GB • Size: 51mm×30mm×4mm • Transfer rate: 280 MB/s • Power: 1.5 W
MiniCircuits zx60-33LN Amplifier	• Bandwidth: 50 to 3000 MHz • Gain: 16.5 dB • Power supply: 5.5 V

4.4. Issue with Conversion from UDP to UHD

A major issue was encountered at the beginning of C term with the progression of the project. By the end of B term, the team had begun to successfully transmit and receive basic string messages such as “hello world” from one USRP2 to the other. This was a significant step and signaled that the project was making good progress. However, with the start of C term, the class ECE 4305: Software Defined Radio began to use the USRP2s and upgraded all of the MATLAB copies from version 2010b to 2011a. Along with this change was a switch from the driver software for the USRP2s from UDP to UHD. This seemingly minor change actually caused several weeks of problems as the Simulink models ceased to work and the team had to make numerous changes to ensure that it functioned properly again.

UDP, or User Datagram Protocol, is the driver software that Simulink used to interface with the USRP2 in the 2010b and prior versions of MATLAB. UDP’s functioning is not visible to the user at all because it defines the lower level interactions between the radio and the computer, but it is for this very reason that it is important. With UDP, Simulink programs using the USRP2 interface block must run with the frame size set at 358 samples. This is a set parameter that cannot change, making the user define their programs to meet this requirement. The team went through this process during B term, designing the text transmission program to fit the message within the 358 samples of the frame. In addition, the sample time of each program had to be tailored to match the frame size and message length. While this parameter limits how data may be transmitted, the user can modify the program to separate a larger message into separate frames. The team was planning on doing this very step at the beginning of C term to send larger and more complex messages over the USRP2s.

However, when C term started ECE 4305 updated all of the computers to run MATLAB 2011a with the UHD driver software. UHD stands for Universal Software Radio Peripheral Hardware Driver, and operates very differently from UDP. It can have a variable frame length, letting the user decide to incorporate any number of samples ranging from, for example, 50 to over 10,000. The sample time could then be matched to this new frame length instead of a set length of 358. These two different parameters allow the user much greater flexibility in creating a program by not limiting how the radio transmits data. Furthermore, there were even changes to the Simulink blocks themselves, requiring the USRP2 blocks from the original programs to be updated to the new USRP2 blocks. The different blocks can be seen in Figures 46 and 47.

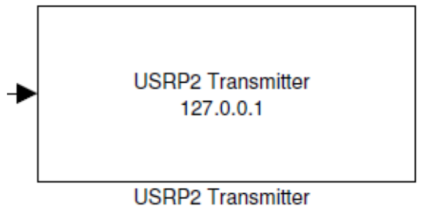


Figure 46: MATLAB Version 2010b USRP2 Transmitter Block

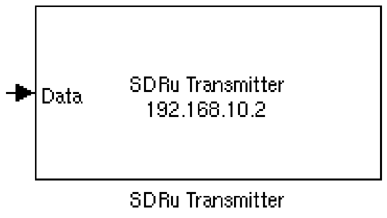


Figure 47: MATLAB Version 2011a USRP2 Transmitter Block

While these updates may seem relatively inconsequential, they had a large impact on the progress of the project. Several weeks at the beginning of C term had to be devoted to including the changes and ensuring that the original Simulink transmit and receive programs were again working. Because all of the settings and blocks in Simulink are so closely intertwined and dependent, changing one number such as the sample time in the setting for one block could create a cascade of errors in the following blocks. The only blocks that required visible modification were the USRP2 transmitter and receiver blocks; all the other changes were in the settings. As a result, there is no way to document the work done to update the models. Significant time was devoted in lab to sorting through the errors resulting from these updates and modifying the programs to work with UHD. After much troubleshooting to determine and fix the source of these errors, the team was able to modify the Simulink programs to work with UHD at any chosen frame length.

4.5. Increasing Flexibility of Transmitter and Receiver Models

After modifying the basic Simulink transmitter and receiver models to operate with UHD, the group continued on to improve the flexibility and resiliency of these models. This was an important next step because the radios needed to be able to transmit different messages under different scenarios. To account for this possibility, the team implemented the option to send variable length messages and different file types.

4.5.1. Variable Length Messages

First, the group needed to be able to modify the frame length as desired. This ability would enable messages of any length to be transmitted efficiently using as few frames as needed. Using UHD, this proved to be a relatively straightforward task. The first step was to select the message; in this example, the string “hello world” was chosen. The MATLAB m-file charToBitsAndBack.m (as seen in the appendix) was used to convert the string to a binary array where each character was represented by seven bits. Thus, this string would be 77 bits long. It was defined as the variable sBit and sent to the model mFindFrameStartTx.mdl, which then transmitted the message to the receiver through the USRP2. The model is shown below.

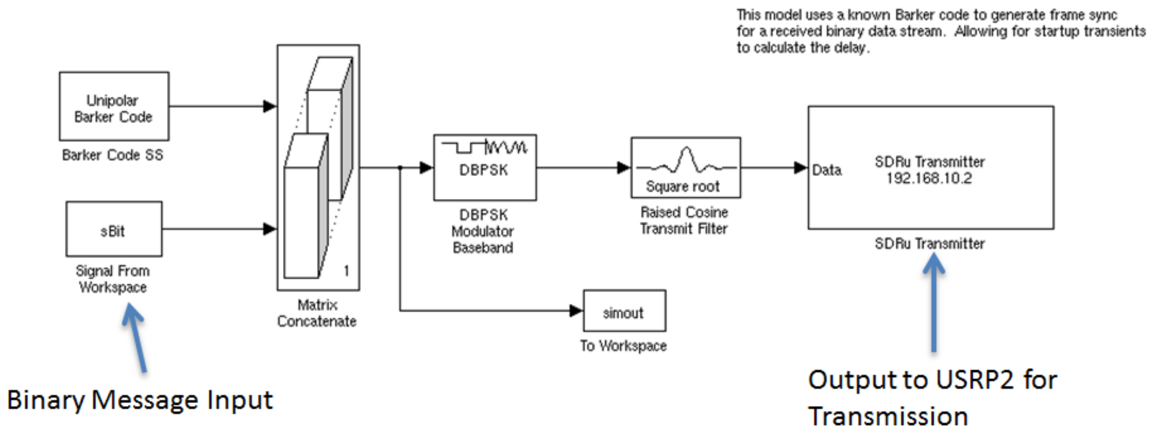
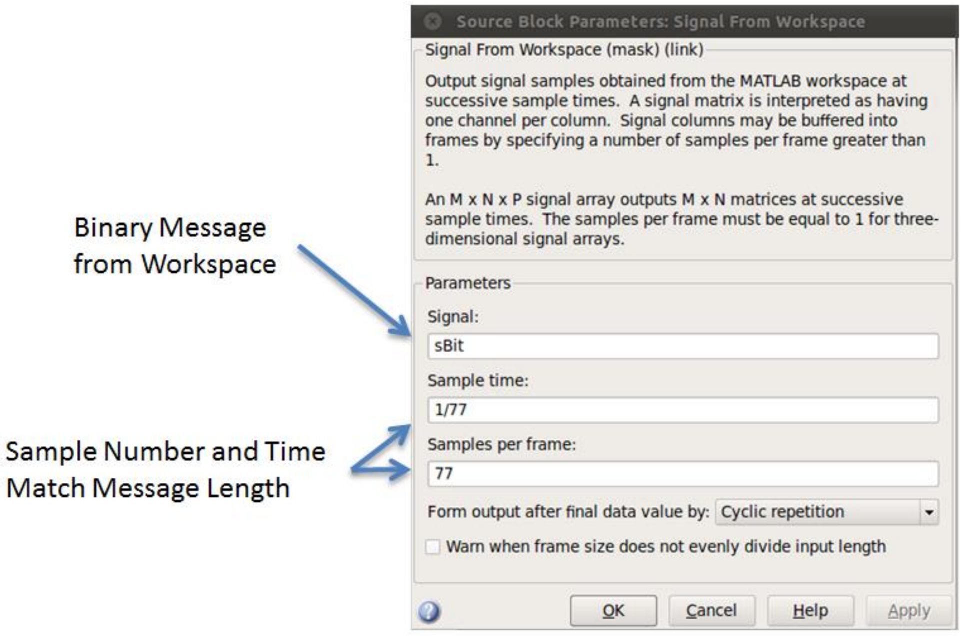

This model uses a known Barker code to generate frame sync for a received binary data stream. Allowing for startup transients to calculate the delay.
Unipolar Barker Code
Barker Code SS
sBit
Signal From Workspace
1
Matrix Concatenate
DBPSK
DBPSK Modulator Baseband
Square root
Raised Cosine Transmit Filter
Data
SDRu Transmitter 192.168.10.2
SDRu Transmitter
simout
To Workspace
Binary Message Input
Output to USRP2 for Transmission

Figure 48: mFindFrameStartTx.mdl - Text Message Transmitter Model

In addition to modifying the input string, the settings for the model had to also be modified to accommodate different frame lengths. Specifically, the sample time in the "Signal From Workspace" block had to be set to 1/77 with 77 samples per frame to match the length of the message, as seen below. These numbers would be different if the message was a different length.


Source Block Parameters: Signal From Workspace
Binary Message from Workspace
Sample Number and Time Match Message Length
OK
Cancel
Help
Apply

Signal From Workspace (mask) (link)

Output signal samples obtained from the MATLAB workspace at successive sample times. A signal matrix is interpreted as having one channel per column. Signal columns may be buffered into frames by specifying a number of samples per frame greater than 1.

An M x N x P signal array outputs M x N matrices at successive sample times. The samples per frame must be equal to 1 for three-dimensional signal arrays.

Parameters

Signal: sBit

Sample time: 1/77

Samples per frame: 77

Form output after final data value by: Cyclic repetition

☐ Warn when frame size does not evenly divide input length

Figure 49: "Signal From Workspace" Block Settings

The other blocks did not depend on the frame length, so their settings did not need to be changed. In this manner – selecting the message, entering it into charToBitsAndBack.m, and then modifying the sample time and samples per frame in the “Signal From Workspace” block – the Simulink transmitter program was made to accommodate messages of any desired length.

To receive the message with its variable length, the receiver model required similarly straightforward modifications. The full model and its “FrameSync” subsystem, as seen in the two figures below, did not require any visible changes.

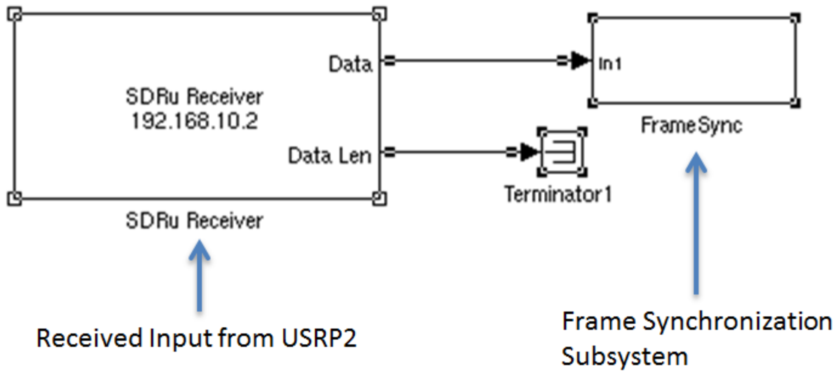


Figure 50: FrameSyncFinalv2.mdl – Text Message Receiver Model

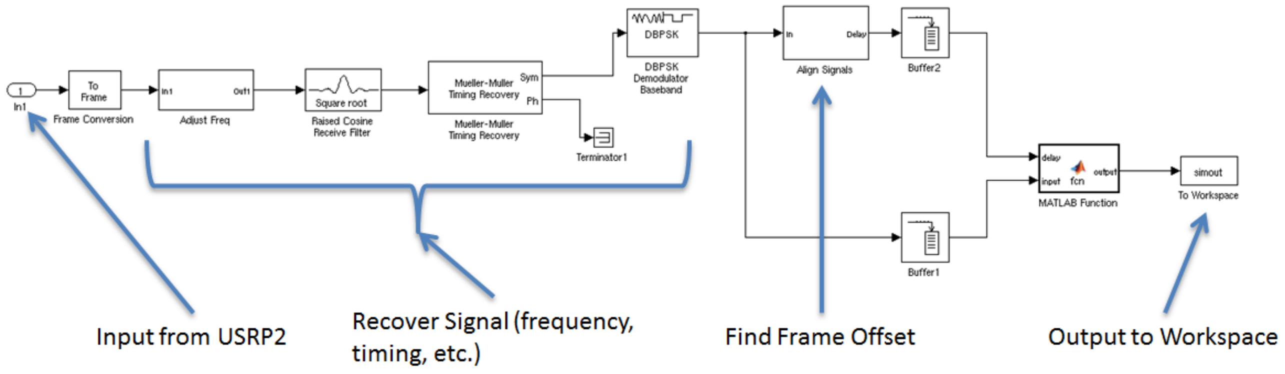


Figure 51: FrameSync Subsystem

Instead, parameters in the settings of the blocks had to be modified. Specifically, the MATLAB function in the subsystem had a variable mlength that needed to be set to the desired frame length of 77. In addition, the m-file that decoded the received message from bits into a string, CharToBitsAndBackRxv2.m, also had a variable mlength that needed to be set to the desired frame length. Both these functions can be seen in the appendix. Once these changes were made, the receiver model was able to work in unison to send and receive any message put into a frame of variable length, thus increasing the overall flexibility of the system.

4.5.2. XML to Bits Conversion

Although the system could now handle variable length messages and frames, it still had room for further improvement. The messages that it was sending were simple strings defined in a MATLAB script, which are not very challenging to decode and transmit. The next step needed to be the addition of a feature that allowed the system to take in a file of any type and convert it to a binary vector which could then be transmitted. This would be much more realistic for when the system is fully implemented on the mothership and taking in various types of files such as video.

In order to meet this need for interoperability with different file types, the m-file inputXML.m was created, which can be seen in the appendix. It began by calling any desired file of any type, in this case the XML document CommsAbstract.xml. It used the functions fopen() and fread() to take in the file and convert it into an array of bits that represent the ASCII values in the original file. The array was defined so that each column was one frame with the frame length being 987, resulting in 987 rows. The number of columns was determined by the size of the original file.

At this point, the transmitter program was able to transmit the frames in the array one at a time, allowing the entire file to be transmitted in separate packets. The receiver would then run the last several lines of code in inputXML.m, which used the functions fopen() and fwrite() to create a new blank file, convert the incoming binary frames into ASCII characters, and then write these characters into the file. This last step thus completed the process of transmitting and receiving the XML file. A flow diagram demonstrating this process can be seen below.

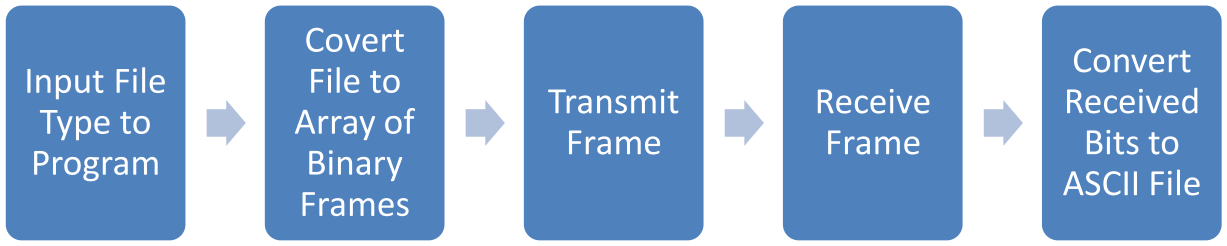

Input File Type to Program
Covert File to Array of Binary Frames
Transmit Frame
Receive Frame
Convert Received Bits to ASCII File

Figure 52: XML Conversion Flow Diagram

This process using inputXML.m is fairly simple yet very important. The three main functions used in the script– fopen(), fread(), and fwrite() – are all predefined MATLAB functions designed for this type of file manipulation. When they are combined and applied in this manner, they make a script that is capable of taking in any file type of any size and successfully transmitting it with the USRP2s. This is an important step for the large video files that will need to be sent to the base station. The one problem encountered with this method was that the Simulink program could not switch quickly from sending one frame to the next. There was no clear way to address this problem using blocks in the transmitter model to switch between the columns of the array, so a separate test script was written that called the model repeatedly and sent a separate frame to it on each call. This method was slow and inefficient, but there was no better alternative. A more effective means of transmitting multiple frames in a row should be developed in further work.

4.6. Testing USRP2 Transmission Range

Now that the system could handle transmitting messages with multiple frames originating from different file types, the next step was to determine the effective range of the USRP2s. Since the UAVs will be moving around while carrying out the search, they will be at varying distances from the user. The communications system will need to handle such a situation and ensure that contact is always maintained between the mothership and base station. Knowing the effective range of the radios is imperative while designing the communications system to ensure that this requirement is met.

To determine the range of the USRP2s, a field test needed to be carried out in a controlled environment. This test would require a large open space to provide the radios an unrestricted line of sight and thus maximize the likelihood of transmission. In addition, the site needed electrical outlets available to power the computers and radios. Lastly, Wi-Fi was required to connect the laptops to the WPI network and obtain the license for MATLAB, which would run with Simulink to control the USRP2s. These requirements logically led to doing the experiment in a large indoor space, which is why Harrington Auditorium was ultimately selected.

To conduct the test, the team set up a USRP2 connected to a laptop at either end of the basketball courts in Harrington, with each pair operated by a team member. There was a clear line of sight between both radios. A picture of this setup is seen below.

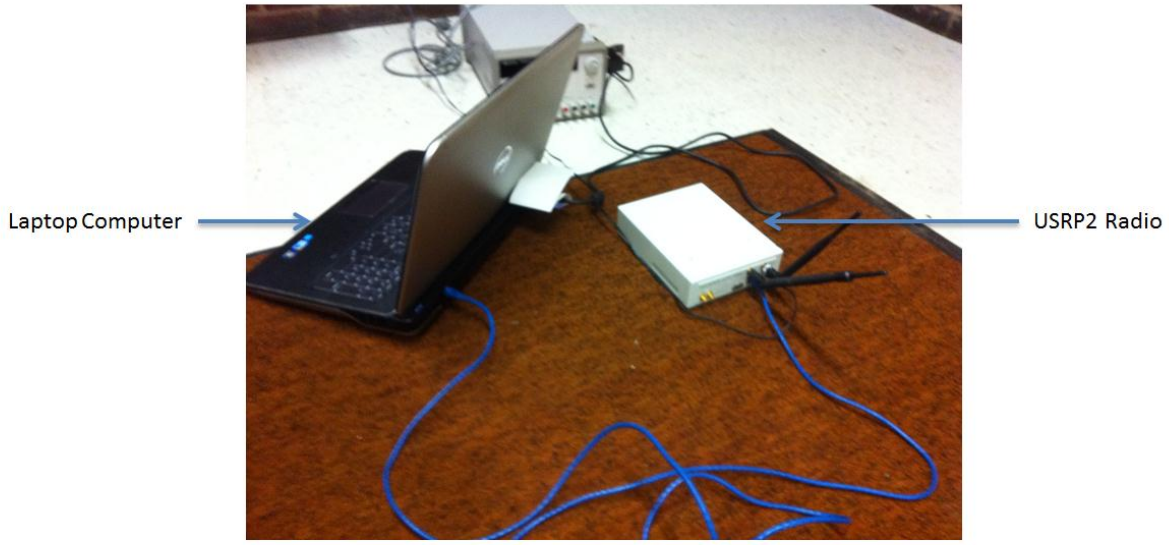


Figure 53: USRP2 Range Test Setup

The frequency offset between the two radios was first determined to be 80 kHz by using the model observeFFT.mdl, which permitted the receiver to observe the spectrum of the transmitter. This is a very large number because the normal offset is no more than 20 kHz, leading to speculation that there was a problem with the radios. Nonetheless, the offset value was incorporated into the receiver's frequency setting to ensure correct transmission. In addition, the transmitter used the Simulink model mFindFrameSyncTx.mdl to transmit the message and the m-file char2BitsAndBack.m to encode the message. The receiver used the model frameSyncFinalv2.mdl to receive and the m-file char2BitsAndBackRxv2.m to decode the message. These models are in figures 48, 50, and 51, and the m-files are in the appendix.

Next, the range and transmitter power were selected to be the independent variables. Modifying the range between the radios would simulate the varying range between the mothership and the base station; modifying the transmit power would simulate cases of path loss due to atmospheric interference or larger distances. The ranges were set at approximately 40 yards and 25 yards, and the power of the transmitting USRP2 was switched between 32 dB, 16 dB, 8 dB, and 0 dB at each distance. A map of the setup of the radios in Harrington is shown in Figure 54.

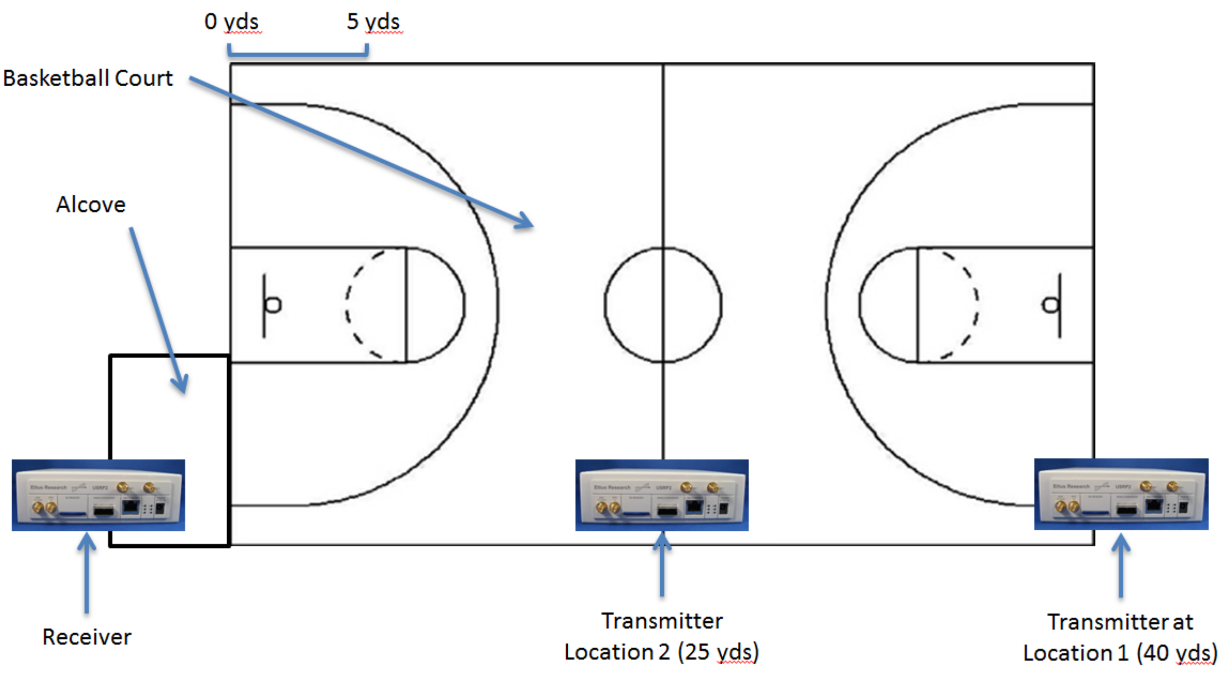


Figure 54: Harrington Range Test Map

Changing the receiving USRP2's power was deemed unnecessary since it simulated the base station, which would be stationary and possess a constant gain. Thus, the receiver's power was kept at a constant 32 dB, which was the maximum possible gain. The transmitted message was "Is it the Apocalypse… or is it just C term?" The results were recorded for each set of variables and are shown below.

Table 14: USRP2 Range Test Results

Transmitter Gain	40 Yards	25 Yards
32 dB	Successfully received	Successfully received
16 dB	Successfully received	Successfully received
8 dB	Received "? ? @? @P ? @? ? ? A? ? A E@ ???"	Received "? ?? ?? @? ?"
0 dB	Received nothing	Received nothing

As these results demonstrate, the USRP2s performed well at varying ranges. The received messages were nearly identical between 40 yards and 25 yards, which was not expected. Instead, the team anticipated that the message would be received better when the radios were closer. The message was received at both ranges if the transmitter gain was above 16 dB but not if it was below 8 dB, indicating that the cutoff gain between successful and unsuccessful transmission was somewhere in between the two. This result indicates that provided there is a clear line of sight between the radios, then the likelihood of successful reception is high for a sufficiently high gain. Furthermore, the radios could likely still work at a much greater range – such as 50, 70, 100 yards or more – since the message was received even at well under the transmitter's maximum gain of 32 dB. Further testing at longer ranges would have been ideal to confirm this expectation. Unfortunately, the group was pressed for time and there was no way to increase the range of the test in Harrington.

This USRP2 range test proved to be highly successful. It demonstrated the capability of the radios to successfully transmit and receive a message at ranges up to at least 40 yards, provided that the gain was sufficiently high. In addition, the results suggest that the radios could operate just as well at much greater ranges. The addition of an amplifier, such as the MiniCircuits ZX60-33LN, would increase the distance even more. These results bode well for the mothership, suggesting that it can operate well at a minimum of 40 yards and, with sufficient gain, probably at 100 yards or more. Thus, this field test provides proof-of-concept for the chosen architecture of a wireless UAV system.

4.7. Implementation on Motherboard

At this point our complete communications protocol had to be implemented on the Pico ITX motherboard. This step is highly important and innovative because it is the first time that such an SDR system has been implemented on a small processor with the intent of placing it on a UAV. This section elaborates the steps that were followed in order to have a fully functional motherboard, capable of performing reliable communications while operating in the sky.

4.7.1. Choosing the operating system

First of all, the appropriate operating system that would be loaded in the motherboard had to be chosen. The initial choice was the Debian operating system. Debian is a free and open source software. It is known for relatively strict adherence to the philosophies of UNIX and free software as well as using collaborative software development and testing processes [54]. In brief, this software would have been the most suitable for our purpose since it has the least CPU consumption comparing to other similar operating systems.

However, due to issues that were faced during the installation of Debian, we chose a similar operating system, the XUBUNTU. The installation error of Debian was caused by the graphics card that was installed in the motherboard. Since it was a built-in feature we could not fix it. Hence, XUBUNTU was installed in the motherboard. This operating system is based on the same concept as Debian and it also has a similar friendly Graphic User Interface to Debian. Figure 55 shows are successful installation of MATLAB/Simulink on the Pico ITX motherboard

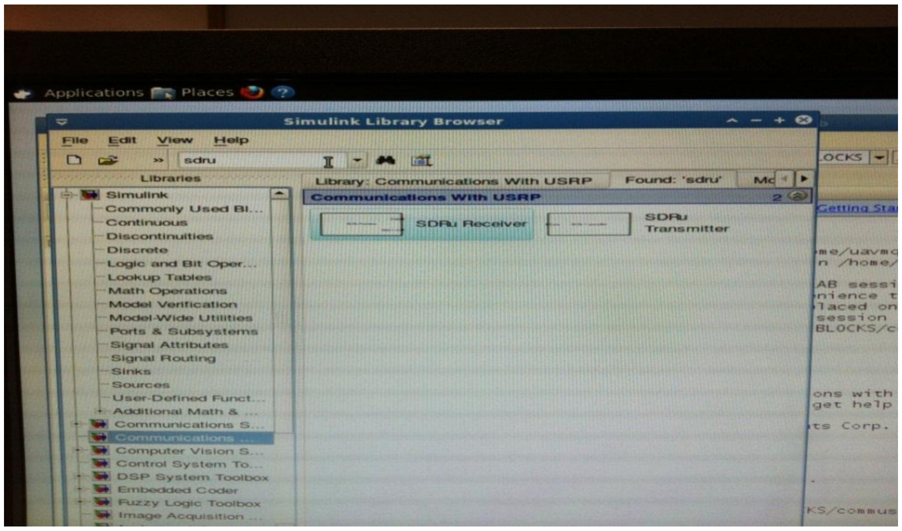


Figure 55: MATLAB running on the Pico ITX motherboard

Several tests regarding the CPU power consumption between the two operating systems were performed. The results can be summarized in Figures 56 and 57 for XUBUNTU and the Debian respectively.

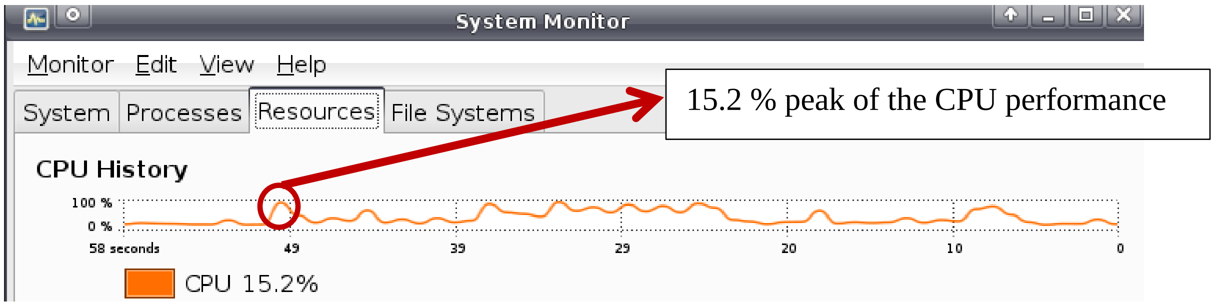


Figure 56: CPU power consumption using the XUBUNTU's system monitor

Figure 56 shows that after 1 minute of monitoring the CPU power usage while installing MATLAB in XUBUNTU, 15.2 % of the total CPU was being used.

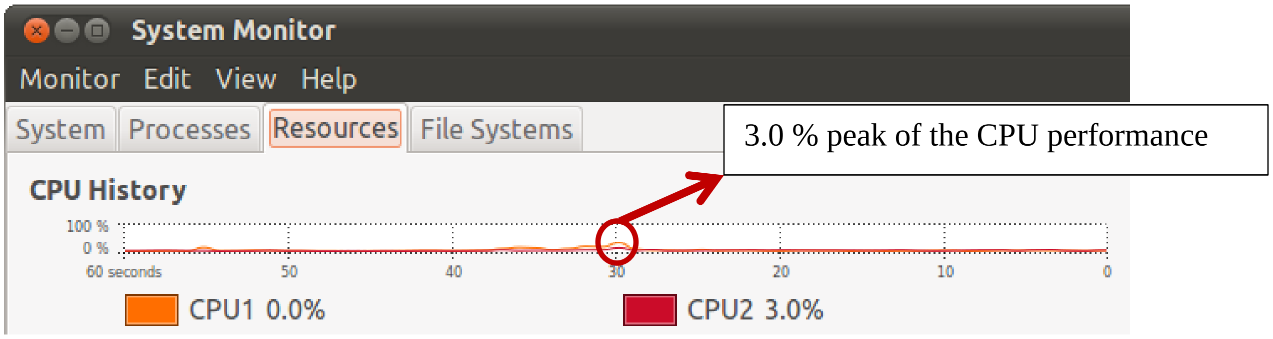


Figure 57: CPU power consumption using the Debian system monitor

Figure 57 verifies the significant difference in CPU power usage while installing MATLAB in Debian. Looking at the CPU2, only 3.0 % was being used after 1 minute of monitoring.

4.7.2. Power and Heat Issues

As mentioned in section 4.7.1 XUBUNTU consumed more CPU power while MATLAB was being installed. The CPU usage could go up to 85.7% that caused our processor to overheat (over 60^{o}C) and therefore to get our motherboard "frozen". We also realized that our initial assumption of the 1.5A current that was needed to power the motherboard was not sufficient; therefore we had to increase it to 3A.

In order to fix the overheating issues we had to install a fan on the top of the heat sink of the microprocessor. The result of this installation was that the temperature was stable at 35^{o}C and hence the performance of the motherboard was significantly improved.

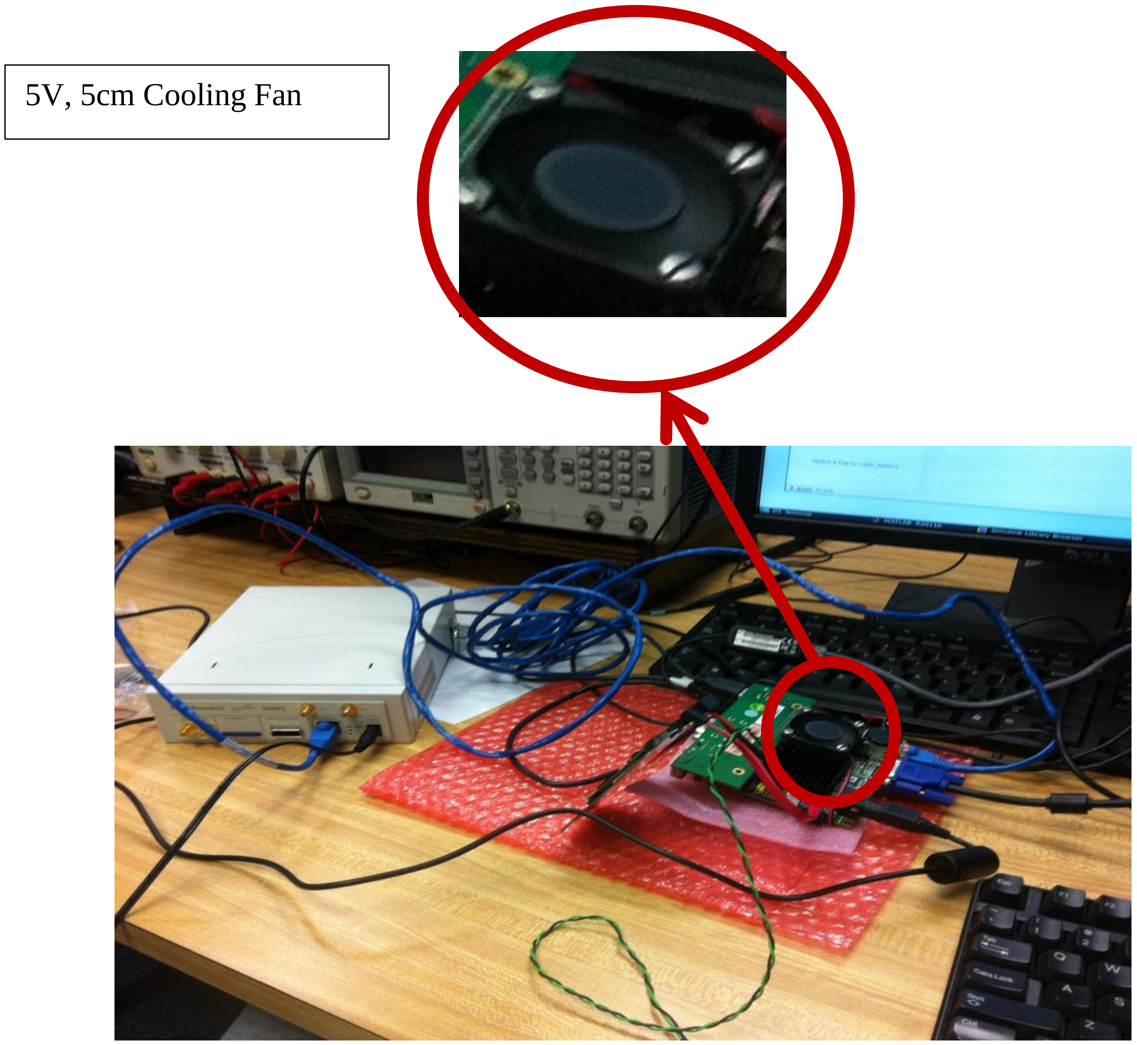


Figure 58: Final setup of the motherboard with the cooling fan installed and the USRP2 connected

Figure 58 shows the final setup of the motherboard. Notice that the USRP2 is also connected to it via the Ethernet interface. The cooling fan uses the 5V that was needed to power the Pico ITX. This picture was taken in AK 227 on March 2nd 2012 at 20:35 during the final successful transmission test of an XML file that can be found on Appendix E.

At this point all the components were successfully installed. Therefore we run several tests using a decimation of 512 while transmitting any message. As expected the transmission was successful since we were able to receive any message that was generated using the Pico ITX motherboard.

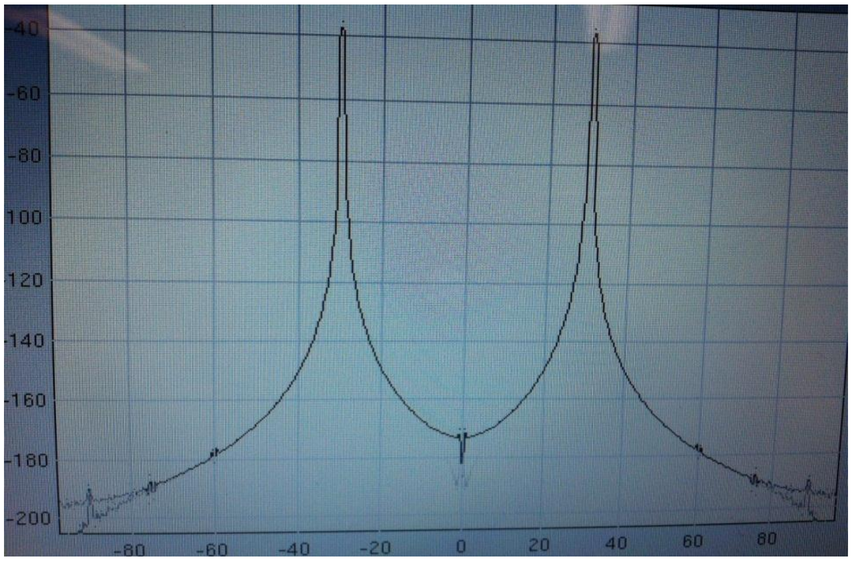
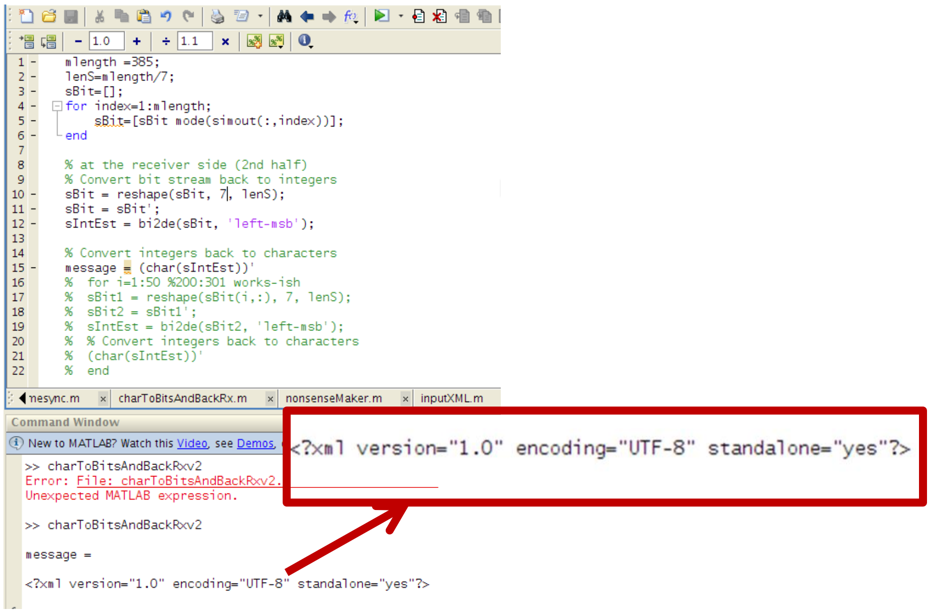


Figure 59: Successful transmission of the XML file using an interpolation of 512 at 2.42GHz.

Figure 59 and 60 shows the successful transmission of the XML file that can be found on Appendix E using a decimation of 512 at 2.42 GHz.



```
mlength =385;
lenS=mlength/7;
sBit=[];
for index=1:mlength;
    sBit=[sBit mode(simout(:,index))];
end

% at the receiver side (2nd half)
% Convert bit stream back to integers
sBit = reshape(sBit, 7, lenS);
sBit = sBit';
sIntEst = bi2de(sBit, 'left-msb');

% Convert integers back to characters
message = (char(sIntEst))'
%  for i=1:50 %200:301 works-ish
%  sBit1 = reshape(sBit(i,:), 7, lenS);
%  sBit2 = sBit1';
%  sIntEst = bi2de(sBit2, 'left-msb');
%  % Convert integers back to characters
%  (char(sIntEst))'
%  end
```

```
>> charToBitsAndBackRxv2
Error: File: charToBitsAndBackRxv2.
Unexpected MATLAB expression.

>> charToBitsAndBackRxv2

message =

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

Figure 60: Successful reception of the XML file using a decimation of 512 at 2.42GHz on the picoITX motherboard

4.8. Summary

This project was able to achieve most of its final goals during the implementation phase. The team's achievements were as follows:

- Successfully created a functioning Simulink system using the USRP2s that could transmit a text message.
- Improved system flexibility by modifying the length of the message and the frames.
- Applied script that allowed the program to read in and transmit any file type of any length and then decode it at the receiver.
- Carried out range test of USRP2s in Harrington Auditorium to determine that they can transmit a minimum of 40 yards and likely much further.
- Loaded MATLAB and Simulink onto Pico ITX motherboard and used them to operate a USRP2.

Several issues were encountered along the way, particularly the conversion from UDP to UHD, but they were eventually overcome. These accomplishments have successfully provided proof-of-concept for the architecture of the desired wireless UAV search network and laid the groundwork for future work on the system.

Chapter 5 – Conclusion and Future Work

5.1. Conclusions

Overall, this project was successful. The group's goal was to lay the groundwork and provide proof-of-concept for an unmanned aerial vehicle communications system that would aid in search and rescue missions. The specific objectives for the end of C term were to perform a range test of the USRP2 radios in Harrington Auditorium and operate the radios using Simulink installed on the motherboard. These objectives were successfully met, and many conclusions can be drawn from the results. However, time and budget constraints meant that the team was unable to progress further and implement a fully functional communications system on the UAVs, leaving room for future work.

5.1.1. USRP2 and Simulink Communications System

One of the main objectives of the project was to create a communications system using Simulink that would simulate the mothership-base station link. The team was able to successfully achieve this objective by sending text messages between two USRP2s in lab. The Communications Systems Toolbox in Simulink was used to design a system that would take in a predetermined message, encode it, transmit it with the radios, and decode it at the receiver. This process was accomplished using the mFindFrameSyncTx.mdl and frameSyncFinalv2.mdl models as the transmitter and receiver, respectively. However, transmitting video data was deemed too difficult with the limitations of a 200 kbit/s sampling rate and 512 decimation rate for the radios. In addition, Simulink was not optimized to transmit video. Instead, the team focused on improving the flexibility of this system and testing the radio transmission range.

The team sought to improve the Simulink program's flexibility by modifying the frame length and the file type that could be sent. Modifying the frame length proved to be fairly straightforward with UHD and was accomplished by changing a number of settings in the transmitter and receiver files. The frame length could thus be set to any desired number from about 50 to 10,000 samples, although a length of 77 was often used for testing. Transmitting different file types was accomplished by writing a script that used the fopen() fread(), and fwrite() functions. It converted any given file to bits at the transmitter and reconverted them to a file at the receiver. Although the script successfully accomplished this conversion, the team was unable to find a method to quickly transmit multiple frames, which led to a slower data rate.

Once the Simulink system was improved in this manner, the team then focused on testing the range of the USRP2s. A large open area was needed to provide a clear line of sight, and it needed both Wi-Fi and power outlets for the computer and radios. Thus, Harrington Auditorium was chosen for the test. The radios were set at 40 then 25 yards apart, with the transmitter varying the gain between 32 dB, 16 dB, 8 dB, and 0 dB. Surprisingly, the results were largely the same for the two ranges, with the message being correctly received at 32 dB and 16 dB, a garbled message received at 16 dB, and nothing received at 0 dB. These results show that the USRP2s can function at a range of at least 40 yards, and likely much further, especially with amplification. This test proved that when the radios are mounted on the UAVs, they will be able to maintain connectivity over varying distances.

5.1.2. Motherboard Implementation

The other main objective of this project was the implementation of the Simulink system on a Pico ITX motherboard. The purpose of this objective was to demonstrate that the communications system could operate on a small processor located on the UAV during flight, which is a requirement for the UAVs to be able to communicate with the user. The Axiomtek Pico820 was chosen for this application for its small size, sufficient 1.6 GHz processor, and sizable 2 GB of RAM.

To achieve this objective, the team first attempted to install Xubuntu onto the motherboard. However, the board overheated and the installation could not be completed. A fan was fixed onto the heat sink to address the issue. With the fan, the overheating was dissipated and Xubuntu was successfully installed. MATLAB and Simulink were then installed without error. Lastly, the USRP2 was connected to the Pico ITX's gigabit Ethernet port and the model mFindFrameSyncTx.mdl was run, which used the radio to transmit a basic text message. This test proved that a small motherboard could operate the USRP2, which is proof-of-concept that it could also function in this manner on the UAV to provide a link between the mothership and base station.

5.2. Future Work

There is a large amount of future work that can be done on this project. Since the team was only able to lay the groundwork for an aerial wireless ad hoc network, future MQP groups can continue the progress and full implementation of the system on the UAVs. Specific areas of potential work include:

- Further improving resiliency of Simulink programs – incorporate energy detection, bidirectional communications, and handshaking with USRP2s.
- Increasing effective data rate of USRP2s – implement transmission of videos and dual-direction control commands over faster network.
- Integrate Wi-Fi and USRP2 – will allow conversion of data between the drones to mothership Wi-Fi link and the mothership to base station USRP2 link.
- Load motherboard onto mothership – test performance of Simulink system and ability to interface with Wi-Fi.

The first two bullet points are the areas of work that are most feasible in the near term. They are the next logical steps from the products of this project and are required in order for the communications system to have the improved resiliency needed for actual implementation. The second two bullet points are longer term goals that will prove more difficult because they require the integration of the hardware, software, and communications aspect of the project. However, they are the important final steps for testing the full implementation of the system. All of these opportunities for future work will extend the capabilities of the products that the team created. In addition, they will bring the project further towards the final vision of a functional UAV search and rescue system.

References

[1] J. I. Mitola and G. Q. J. Maguire, "Cognitive Radio: Making Software Radios More Personal," IEEE Personal Communications Magazine, pp. 13-18, Aug. 1999.

[2] P. Kolodzy, "Spectrum Policy Task Force," Federal Communications Commission, 2002.

[3] SDR Forum. (2007, Nov.) SDR Forum. [Online]. http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-R-0011-V1_0_0.pdf

[4] Ansari, J.; Xi Zhang; Achtzehn, A.; Petrova, M.; Mahonen, P.; , "A flexible MAC development framework for cognitive radio systems," Wireless Communications and Networking Conference (WCNC), 2011 IEEE , vol., no., pp.156-161, 28-31 March 2011.

[5] Mitola, J.; , "The software radio architecture," Communications Magazine, IEEE , vol.33, no.5, pp.26-38, May 1995

[6] D. Valerio, ―Open Source Software-Defined Radio: A survey on GNUradio and its applications, Forschungszentrum Telekommunikation Wien, Vienna, Austria, Rep. FTWTR-2008-002, 2008.

[7] Buracchini, E.; "The software radio concept," Communications Magazine, IEEE , vol.38, no.9, pp.138-143, Sep 2000.

[8] M. Srbinovska, C. Gavrovski, and V. Dimcev. (2008, Sep.) Localization Estimation System Using Measurment of RSSI Based on ZigBee Standard. [Online]. http://fett.tu-sofia.bg

[9] S. Kay, “Statistical decision theory I (Ch. 3)” and “Deterministic signals (Ch. 4),” in Fundamentals of Statistical Signal Processing, Detection Theory, pp.60-140, Prentice Hall, 1998.

[10] H. Tang, “Some physical layer issues of wide-band cognitive radio systems,” 2005 First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN 2005), pp.151-159, Nov. 2005.

[11] K. Po and J. Takada, “Signal detection for analog and digital TV signals for cognitive radio,” IEICE Technical Report, SR2006-54(2006-11).

[12] Ettus Research. (2009, Nov. 18). USRP2: The Next Generation of Software Radio Systems [Online]. Available: http://www.ettus.com/downloads/ettus_ds_usrp2_v5.pdf.

[13] Ettus Research, http://www.ettus.com/.

[14] Sahin, G.; Chindapol, A.; , "Adaptive Spectrum Management in UAV-Aided Ad-Hoc Wireless Networks in Single-area Theater," 2007 Mobile Networking for Vehicular Environments , vol., no., pp.133-138, 11-11 May 2007.

[15] Alshbatat, A.I.; Liang Dong; , "Cross layer design for mobile Ad-Hoc Unmanned Aerial Vehicle communication networks," Networking, Sensing and Control (ICNSC), 2010 International Conference on , vol., no., pp.331-336, 10-12 April 2010.

[16] Alshbatat, A.I.; Liang Dong; , "Adaptive MAC protocol for UAV communication networks using directional antennas," Networking, Sensing and Control (ICNSC), 2010 International Conference on , vol., no., pp.598-603, 10-12 April 2010.

[17] Song Lu; Huang Ting-lei; , "A summary of key technologies of ad hoc networks with UAV node," Intelligent Computing and Integrated Systems (ICISS), 2010 International Conference on , vol., no., pp.944-949, 22-24 Oct. 2010.

[18] Ke Zhang; Wei Zhang; Jia-Zhi Zeng; , "Preliminary Study of Routing and Date Integrity in Mobile Ad Hoc UAV Network," Apperceiving Computing and Intelligence Analysis, 2008. ICACIA 2008. International Conference on , vol., no., pp.347-350, 13-15 Dec. 2008.

[19] Dawei Ma; Shizhong Yang; , "UAV image transmission system based on satellite relay," Microwave and Millimeter Wave Technology, 2004. ICMMT 4th International Conference on, Proceedings , vol., no., pp. 874- 878, 18-21 Aug. 2004.

[20] Bennett, R.R.; , "Satellite relay for unmanned air vehicle data," Military Communications Conference, 1992. MILCOM '92, Conference Record. 'Communications - Fusing Command, Control and Intelligence'., IEEE , vol., no., pp.1146-1150 vol.3, 11-14 Oct 1992.

[21] Lei Ding; Melodia, T.; Batalama, S.N.; Matyjas, J.D.; Medley, M.J.; , "Cross-Layer Routing and Dynamic Spectrum Allocation in Cognitive Radio Ad Hoc Networks," Vehicular Technology, IEEE Transactions on , vol.59, no.4, pp.1969-1979, May 2010.

[22] Tisdale, J.; Ryan, A.; Zu Kim; Tornqvist, D.; Hedrick, J.K.; , "A multiple UAV system for vision-based search and localization," American Control Conference, 2008 , vol., no., pp.1985-1990, 11-13 June 2008.

[23] Wood, L.; Eddy, W.M.; Ivancic, W.; McKim, J.; Jackson, C.; , "Saratoga: a Delay-Tolerant Networking convergence layer with efficient link utilization," Satellite and Space Communications, 2007. IWSSC '07. International Workshop on , vol., no., pp.168-172, 13-14 Sept. 2007.

[24] Yucek, T.; Arslan, H.; , "A survey of spectrum sensing algorithms for cognitive radio applications," Communications Surveys & Tutorials, IEEE , vol.11, no.1, pp.116-130, First Quarter 2009.

[25] Tisdale, J.; Ryan, A.; Zu Kim; Tornqvist, D.; Hedrick, J.K.; , "A multiple UAV system for vision-based search and localization," American Control Conference, 2008 , vol., no., pp.1985-1990, 11-13 June 2008.

[26] Morgenthaler,S; "Prototype of a Highly Adaptive and Mobile Communication Network using Unmanned Arial Vehicules (UAVs) ", University of Barne, 18 November 2009.

[27] United States Coast Guard; "U.S. Coast Guard SAR Statistics," *United States Coast Guard,* vol. 2012, 6 February 2012, 2012.

[28] Homeland Security; "UAVs Perform Autonomous Search-and-Rescue Operations," *Homeland Security News Wire,* vol. 2012, 19 July, 2010.

[29] University of Adelaide; "Design and Build of a Search and Rescue UAV," *YouTube,* vol. 2012, 26 October, 2009.

[30] PBS; "Time Line of UAVs," *NOVA,* vol. 2012, pp. 1, November 2002, 2002.

[31] S. Churchill, "Mountain Rescue UAVs, E911 and Triangulation," *DailyWireless. Org,* vol. 2012, 13 December, 2006.

[32] C. Danilov, T. R. Henderson, T. Goff, J. H. Kim, J. Macker, J. Weston, N. Neogi, A. Ortiz and D. Uhlig, "Experiment and Field Demonstration of a 802.11-Based Ground-

UAV Mobile Ad-Hoc Network," *Military Communications Conference, 2009. MILCOM 2009. IEEE,* pp. 1, 18 October, 2009.

[33] P. Doherty, "Autonomous UAV Search andRescue," *VideoLectures. Net,* vol. 2012, 3 July, 2007.

[34] P. Doherty and P. Rudol, "A UAV Search and Rescue Scenario wit Human Body Detection and Geolocalization," *Lecture Notes in Computer Science,* 2007.

[35] M. Eldridge, J. Harvey, T. Sandercock and A. Smith, "Design and Build a Search and Rescue UAV," *University of Adelaide,* vol. 2012, pp. 298, 30 October, 2009.

[36] J. Garamone, "From U.S. Covil War to Afghanistan: A Short Hisotry of UAVs," *US Department of Defense,* vol. 2012, pp. 1, 16 April 2002, 2002.

[37] M. Goodrich, "UAV-Enabled Wilderness Search and Rescue," *Brigham Young University,* vol. 2012, 12 October`, 2006.

[38] T. W. Heggie and M. E. Amundson, "Dead Men Walking: Search and Rescue in US National Parks " *Sierra Nature Notes,* vol. 2012, pp. 6, 2009.

[39] J. Huo, Z. Xu, Y. Zhang and X. Shan, "A UAV Mobile Strategy in Mobile Ad Hoc Networks," *Electronics, Communications and Control (ICECC), 2011 International Conference on,* pp. 686, 9 September, 20122.

[40] P. Rudol and P. Doherty, "Human body detection and geolocation for UAV search and rescue missions using color and thermal imagery," in *Aerospace Conference, 2008 IEEE,* Big Sky, MT, 2008, pp. 1.

[41] T. Scheve, "How the MQ-9 Reaper Works," *How Stuff Works,* vol. 2012, pp. 1, 2012.

[42] S. Waharte and N. Trigoni, "Supporting Search and Resuce Operations with UAVs," *University of Oxford,* vol. 2012, pp. 6, .

[43] Z. Xu, J. Huo, Y. Wang, J. Yuan, X. Shan and Z. Feng, "Analyzing Two Connectivities in UAV-Ground Mobile Ad Hoc Networks," *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on,* vol. 2, pp. 158, 10 June, 2011.

[44] K. Zhang, W. Zhang and J. Zeng, "Preliminary Study of Routing and Date Integrity in Mobile Ad Hoc UAV Network," *Apperceiving Computing and Intelligence Analysis, 2008. ICACIA 2008. International Conference on,* pp. 347, 13 December, 2008.

[45] J.R. MacLeod, M.A. Beach, P.A. Warr, T Nesimoglu, Filter Considerations In The Design Of A Software-Defined Radio, IST Mobile Communications Summit, Barcelona, Spain, September 2001.

[46] Jeruchim, M.; , "Techniques for Estimating the Bit Error Rate in the Simulation of Digital Communication Systems," Selected Areas in Communications, IEEE Journal on , vol.2, no.1, pp. 153- 170, Jan 1984.

[47] A. Sahai and D. Cabric. (2005, Nov.) Electrical Engineering and Computer Science UC Berkeley. [Online]. http://www.eecs.berkeley.edu/~sahai/Presentations/DySPAN05_part2.ppt

[48] D. Cabric, S. M. Mishra, and R. W. Brodersen, "Implementation Issues in Spectrum Sensing for Cognitive Radios," Berkeley Wireless Research Center, Berkeley, Lab Research Report, 2005.

[49] Introducing Wideband RF Transceiver Board U-RFX, http://osdir.com/ml/discussgnuradio-gnu/2011-02/msg00017.html

[50] Ettus Research. (2009, Nov. 18). Universal Software Radio Peripheral: The Foundation for Complete Software Radio Systems [Online]. Available: http://www.ettus.com/downloads/ettus_ds_usrp_v7.pdf.

[51] Ettus Research. (2009, Nov. 18). USRP2: The Next Generation of Software Radio Systems [Online]. Available: http://www.ettus.com/downloads/ettus_ds_usrp2_v5.pdf.

[52] Scaperoth, D.; —Cognitive Software Defined Radio, May,2005.

[53] GNU Radio and multi core processors, http://www.ruby-forum.com/topic/217088.

[54] Debian Social Contract, http://www.debian.org/social_contract

[55] A. Wyglinski, "Software Defined Radio Systems and Analysis Laboratory Assignment 2", http://dot.ece.wpi.edu/wp-content/uploads/2011/11/ece4305_lab2.pdf

[56] Federal Emergency Management Agency, "About US&R," vol. 2012, 2012.

[57] D. Ferguson, "GIS For Wilderness Search and Rescue," *ESRI Federal User Conference*, vol. 2012, pp. 10, 2008.

[58] Indiana Department of Homeland Security, "Wilderness Search and Rescue," vol. 2012, 2008. .

[59] New Hampshire Fish and Game, "Specialized Search and Recue Team," vol. 2012, pp. 1, 2012.

[60] J. Stanford, "Crash Highlights Rescue Risks," *JH Underground,* vol. 2012, 2012.

[61] C. Frederick-Recascino, "Strata: New Uses for UAVs," *Embry-Riddle Aeronautical University,* vol. 2012, pp. 8, 2006.

[62] A. Fournier, "Unmanned Aerial Vehicle Competition Focuses on SAR," *SAR Scene,* vol. 2012, 2006.

[63] R. Murphy, S. Tadokoro, D. Nardi, A. Jacoff, P. Fiorini, H. Choset and A. Erkmen, "Search and Rescue Robotics," *Springer Handbook of Robotics*, vol. 2012, 2008.

[64] J. How, E. King and Y. Kuwata, "Flight Demonstrations of Cooperative Control for UAV Teams," *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit,* vol. 2012, pp. 9, 2004.

[65] L. Ding, T. Melodia, S. Batalama, J. Matyjas and M. Medley, "Cross-Layer Routing and Dynamic Spectrum Allocation in Cognitive Radio Ad Hoc Netowrks," *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit,* vol. 2012, pp. 10, 2010.

[66] Axiomtech, "Pico820," *E-ITX,* vol. 2012, pp. 1, 2012.

[67] MiniCircuits, "Coaxial Low Noise Amplifier ZX60-33LN," vol. 2012, pp. 2, 2012.

[68] Newegg, "ADATA XM13 mSATA 30 GB SATA II Internal Solid State Drive," vol. 2012, 2012.

[69] Ubiquiti Networks, "Bullet M: Zero-Variable Wireless Infrastructure Deployment," vol. 2012, pp. 8, 2011.

[70] Mitola J, "Software Radio Architecture–Object Oriented Approaches to Wireless Systems Engineering", Wiley-Interscience, ISBN 0-471-38492-5, 2000

[71] Wyglinski, Alexander M. ECE4305: Software Defined Radio Systems and Analysis[C-Term, 2011]. 12 9, 2011. http://www.wireless.wpi.edu/courses/ece4305c11/index.html (accessed January 17, 2011).

[72] Rice, M; Digital communications: a discrete-time approach. Upper Saddle River, New Jersey: Prentice Hall, 2009.

[73] Proakis J; Digital Communications: McGraw-Hill, 1995

Appendices

Appendix A

Characters to bits and back MATLAB file (Transmitter Side)

The following MATLAB code was generated before running the transmitter for the framesync (sending hello world). The main purpose of this code is to transform a string to a binary stream and back again. All the characters from the ASCII table are defined in the variable ***ASCIITable***.

```
% Transform a string to a binary stream and back again.
%MQP version
s = 'Hello, world!';
ASCIITable = [32  ' ';...
              33  '!';...
              34  '"';...
              35  '#';...
              36  '$';...
              37  '%';...
              38  '&';...
              39  '''';...
              40  '(';...
              41  ')';...
              42  '*';...
              43  '+';...
              44  ',';...
              45  '-';...
              46  '.';...
              47  '/';...
              48  '0';...
              49  '1';...
              50  '2';...
              51  '3';...
              52  '4';...
              53  '5';...
              54  '6';...
              55  '7';...
              56  '8';...
              57  '9';...
              58  ':';...
              59  ';';...
              60  '<';...
              61  '=';...
              62  '>';...
              63  '?';...
              64  '@';...
              65  'A';...
```

```
66  'B';...
67  'C';...
68  'D';...
69  'E';...
70  'F';...
71  'G';...
72  'H';...
73  'I';...
74  'J';...
75  'K';...
76  'L';...
77  'M';...
78  'N';...
79  'O';...
80  'P';...
81  'Q';...
82  'R';...
83  'S';...
84  'T';...
85  'U';...
86  'V';...
87  'W';...
88  'X';...
89  'Y';...
90  'Z';...
91  '[';...
92  '\';...
93  ']';...
94  '^';...
95  '_';...
96  '`';...
97  'a';...
98  'b';...
99  'c';...
100 'd';...
101 'e';...
102 'f';...
103 'g';...
104 'h';...
105 'i';...
106 'j';...
107 'k';...
108 'l';...
109 'm';...
110 'n';...
111 'o';...
112 'p';...
113 'q';...
114 'r';...
115 's';...
116 't';...
117 'u';...
118 'v';...
119 'w';...
120 'x';...
121 'y';...
122 'z';...
```

```
              123 '{';...
              124 '|';...
              125 '}';...
              126 '~'];

% Cycle through the string and convert characters to integers
lenS = length(s);
sInt = zeros(lenS,1);  % Preallocate for speed

% at the transmitter side (1st half) generates sBit for transmission
% If you need this part, comment the 2nd half
% Convert integers to bit stream
for idx = 1 : lenS
    ASCIIIdx = find(ASCIITable(:,2) == s(idx));
    sInt(idx) = ASCIITable(ASCIIIdx,1);
end

% Convert integers to 7-bit words and columnize
sBit = de2bi(sInt, 7, 'left-msb')';
sBit = sBit(:); % This is the sBit for transmission [77*1]

% % at the receiver side (2nd half)
% % Convert bit stream back to integers
% rx=received(1,:); % you only need the first 77 bits
% sBit1 = reshape(rx, 7, lenS);
% sBit2 = sBit1';
% sIntEst = bi2de(sBit2, 'left-msb');
%
% % Convert integers back to characters
% sEst = (char(sIntEst))';
%
% % [EOF]
```

Appendix B.

Characters to bits and back MATLAB file (Receiver Side)

The following MATLAB code was generated before running the receiver for the framesync (receiving and decoding hello world). The main purpose of this code is to convert integers to 7-bit words and columnize and then convert a bit stream back to integers. All the characters from the ASCII table are defined in the variable ***ASCIITable***.

```
mlength =245; %need to change in matlab function block, too
lenS=mlength/7;
sBit=[];
for index=1:mlength;
    sBit=[sBit mode(simout(:,index))];
end

% at the receiver side (2nd half)
% Convert bit stream back to integers
sBit = reshape(sBit, 7, lenS);
sBit = sBit';
sIntEst = bi2de(sBit, 'left-msb');

% Convert integers back to characters
message = (char(sIntEst))'
%  for i=1:50 %200:301 works-ish
%  sBit1 = reshape(sBit(i,:), 7, lenS);
%  sBit2 = sBit1';
%  sIntEst = bi2de(sBit2, 'left-msb');
%  % Convert integers back to characters
%  (char(sIntEst))'
%  end
```

Appendix C

Alternative FramSync – Receiver Side

The MATLAB script below takes care of the entire frame sync, without using the Simulink blocks as described in the Results Section. In general, we wrote a MATLAB script for frame synchronization on the receiver side.

```
%% Tasos Code for frame sync 2nd attempt
function [sEst_1 sEst_2] = fcn(u)
s = 'Hello world';
lenS = length(s);
length_of_ASC = 77;

% Barker code
barker = [0 0 0 0 0 1 1 0 0 1 0 1 0]';
barker_bipolar = strrep(barker,0,-1)';

% Correlation
data_bipolar = strrep(u,0,-1)';
corr_temp = xcorr(data_bipolar,barker_bipolar);
corr = corr_temp(length(data_origin)-length(barker_bipolar)+1:end,1);

% Peak detection
peak_pos = [];
peak_val = []'
valid_data = [];
sBit1 = [];
m = 1;
valid_data_output = [];

max_num = 20;
corr_change = corr;
peak_pos = find(corr > 12.5);

% Find start and end of each frame
start_frame = peak_pos+1;
end_frame = peak_pos+77;

% Difference between frame lengths
diff = peak_pos(2:end)-peak_pos(1:end-1);

% Extract frames
for i=1:length(peak_pos)
    valid_data(:,1) = data_origin(start_frame(i):end_frame(i));
end

% Find the "correct data" (1 way)
valid_data_output_1 = mode(valid_data,2);

% Find the "correct data" (2 way)
valid_data_output_2 = mean(valid_data,2) > 0.5;
```

```
% Extract data
sBit1 = reshape(valid_data_output_1(14:90), 7, lenS);
sBit1 = sBit1';
sIntEst_1 = bi2de(sBit1, 'left-msb');

sBit2 = reshape(valid_data_output_2(14:90), 7, lenS);
sBit2 = sBit2';
sIntEst_2 = bi2de(sBit2, 'left-msb');
```

Appendix D

Encoding an ASCII file into a binary file

This script converts ASCII text to Binary numbers

```
function encode(File,File2)
%ENCODE  Converting ASCII text to Binary numbers
%   DECODE(FILE,FILE2) where FILE is an input text file consisting of ASCII
text and
%       FILE2 is the output file.
%       The code key file (code.m) should be included in the same directory
in order
%       for this program to run.

wpl=0;
[spc_ent,C0]=textread('code.m','%s %s',2);
[L,C]=textread('code.m','%s %s','headerlines',2);
L=char(L);
FID = fopen(File,'r');
OUT = fopen(File2,'w');
while 1
    tline = fgetl(FID);
    if ~ischar(tline), break, end
    for i=1:size(tline,2)
        x=find(L==tline(i));
        if isempty(x)==1
            fprintf(OUT,'%s',char(C0(1)));
            wpl=wpl+1;
        else
            fprintf(OUT,'%s',char(C(x)));
            wpl=wpl+1;
        end
        if wpl==10, fprintf(OUT,'\n');, wpl=0;, end
    end
    fprintf(OUT,char(C0(2)));
    wpl=wpl+1;
end
fclose('all');
```

Decoding a *.txt file using ASCIITable

This script converts binary numbers to ASCII text

```
function decode(File,File2)
%DECODE  Converting Binary numbers to ASCII text
%   DECODE(FILE,FILE2) where FILE is an input text file consisting of binary
numbers
%       and FILE2 is the output ASCII text file.
%       The code key file (code.m) should be included in the same directory
in order
%       for this program to run.
```

```
t=0;
[spc_ent,C0]=textread('code.m','%s %s',2);
[L,C]=textread('code.m','%s %s','headerlines',2);
L=char(L);
bit = length(char(C(1)));
FID = fopen('matlab.txt','r');
OUT = fopen('matlab1.txt','w');
while 1
    tline = fgetl(FID);
    if ~ischar(tline), break, end
    for i=1:size(tline,2)/bit
        for j=1:93
            x=isequal(char(C(j,:)),tline(8*i-7:8*i));
            if x==1
                t=j;
            end
        end
        if t~=0
            fprintf(OUT,'%s',L(t));
        elseif tline(8*i-7:8*i)==char(C0(2))
            fprintf(OUT,'\n');
        elseif tline(8*i-7:8*i)==char(C0(1))
            fprintf(OUT,' ');
        end
        t=0;
    end
end
fclose('all');
```

Appendix E

Input an XML file for transmission (multiple frames)

The following MATLAB script was generated prior to the start of the transmitter. This scripts takes in an XML file and converts it to a binary array so that Simulink can then transmit.

```
statsXML=dir('CommsAbstract.xml'); %make a struct with info of file,
including length
numBits=statsXML.bytes*8; %total number of bits in file
lenFrame=987; %define frame length - weird thing: later functions don't work
well when lenFrame is over 57769
numFrame=ceil(numBits/lenFrame); %number of frames required to fully transmit
message

idXML=fopen('CommsAbstract.xml', 'r+b'); %open the desired file and return
the file identifier
%charXML=fread(idXML, lenFrame, 'ubit2=>char'); %convert first N values to
%a char vector - for some reason, the script DOES NOT write to the new xml
%file properly when this line runs, so DON'T RUN IT

bitsXML=zeros(ceil(numBits/lenFrame)*lenFrame, 1); %initialize bits vector
bitsXML(1:numBits)=fread(idXML, numBits, 'ubit1'); %convert file to a bits
vector
TxXML=reshape(bitsXML, lenFrame, numFrame); %reshape bits vector into frames
in transmission array
%NOTE: there will probably be extra zeros padded on the end

%now pretend that bitsXML was transmitted and received perfectly...

blankXML=fopen('blankXML.xml', 'r+b'); %open XML document to write to
printXML=fwrite(blankXML, bitsXML, 'ubit1'); %write bits to chars in document
%fprintXML=fprintf(bitsXML, 'ubit1')
```

CommsAbstract.xml file

Below, we provide the sample CommsAbstract.xml file that was used as an input for the ***statsXML*** variable in the previous code

Comms group
Abstract
Wednesday, September 14, 2011
The need for capable search and rescue (SAR) crews is an ever-present need in the modern world. To accomplish their mission, these crews need knowledge of a disaster area or the locations of missing people. Unmanned Aerial Vehicles (UAVs) possess the ability to acquire this knowledge safely from above the search area and relay it to a user. In order to increase efficiency, a group of UAVs equipped with cameras (drones) can be used and relay their data through a central UAV called a mothership. This presents a number of challenges about how the separate UAVs will communicate with each other and the user. For this project, we will propose creating a mobile ad hoc network that enables the separate UAVs to interact, coordinate SAR

efforts, and transmit information to the user. Two additional groups will work on the UAV platform and systems integration, thus dividing the work for the overall UAV system. Our wireless network will be designed to handle many of the problems encountered on multiple mobile platforms moving in various formations. As link quality will not be consistent between nodes, drones may suddenly drop out or join the network. These factors are based largely on the relative movement of each platform, and can thus be measured and accounted for. The primary goal of this project will be to address these problems by creating a resilient communications system in which the UAVs beam their data to the user through the mothership. The mothership will coordinate most user-UAV interaction and manage spectrum allocation and monitoring the other UAVs. This will increase overall network resiliency by centralizing system management. By utilizing concepts from cognitive radio and predictive models based on the motion of each node, our network will attempt to maintain connectivity at all costs and compensate in the event of a disruption. The network can be optimized based on current node states to preserve connectivity, and still recover from entry or exit events when they happen. Numerous lab and field tests will be conducted to achieve this goal. The end product will be a wireless system encoded in the onboard processors of each UAV that successfully enables the user to use the UAV system to carry of their mission of search and rescue.

APPENDIX F

MATLAB/Simulink Installation and Testing

The final step after setting up the motherboard was to install MATLAB with Simulink in order to perform tests to check the capabilities of the motherboard. The steps for the installation are the following:

```
1. sudo wget
   'ece.wpi.edu/linux/MATLAB?action=AttachFile&do=get&target=matlab-
   r2011b.desktop' -O /usr/share/applications/matlab.desktop
```

However this command gave an error regarding an outdated version of the GNU Compiler Collection. The way to fix it is the following:

1. Install The GNU Compiler Collection 4.3 and The GNU Standard C++ Library

```
sudo apt-get install gcc-4.3-multilib libstdc++6-4.3-dev
```

2. Make a MATLAB specific 'bin' directory for gcc symlink.

```
mkdir ~/.matlab/bin
```

3. Symlink gcc to gcc-4.3 via user MATLAB specific 'bin' directory.

```
ln -s /usr/bin/gcc-4.3 ~/.matlab/bin/gcc
```

4. Add MATLAB specific 'bin' directory to the front of your system $PATH within your local startup.m file.

```
printf
"setenv('PATH',sprintf('/home/%%s/.matlab/bin:%%s',getenv('USER'),getenv('
PATH')));\n" >> ~/Documents/MATLAB/startup.m
```

5. Run MATLAB

```
root@UAVMqp - Xubuntu:~$ matlab &
```