Principles of Wireless Communications Lab 3: OFDM

Mark Goldwater and Anusha Datar

April 7, 2021

1 Introduction

Orthogonal frequency-division multiplexing (OFDM) is a type of digital transmission that encodes data on multiple carrier frequencies. In other common modulation schemes such as binary phase-shift keying (BPSK), quadrature amplitude modulation (QAM), and $N \times M$ multiple input multiple output (MIMO) one assumes a flat-fading channel model. One can assume they can model a channel as flat-fading when the coherence bandwidth of the channel (where the channel itself is assumed to be essentially flat) is significantly larger than the bandwidth of the transmitted signal. This assumption allows one to represent the effect of the channel as the multiplication of a single complex coefficient. Figure 1 visually illustrates the comparison between a clean and messy channel.

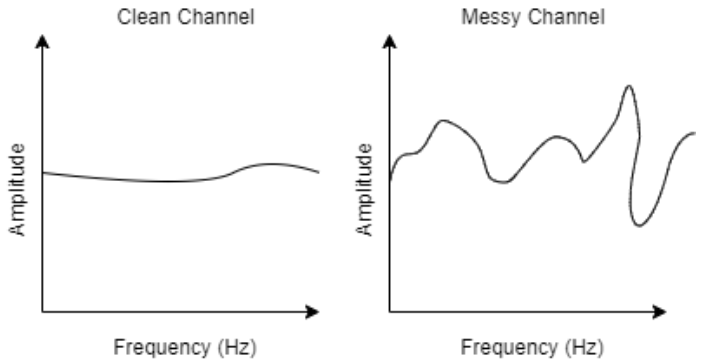

Figure 1: Comparison of a clean channel to a messy channel.

The frequency response of the channel shown in the left side of the diagram above would cooperate nicely with a flat-fading model. Because the amplitude of the channel is essentially flat, its effects can be well-approximated with a single complex coefficient. However, the channel shown on the right in Figure 1 is significantly more ”messy” in that it is not flat for any meaningfully long duration of time. As a result it is difficult to accurately approximate this channel using a single complex coefficient with the flat-fading channel model.

This issue is what OFDM tries to address by splitting up the channel frequency band into many smaller (effectively flat) bands for which the flat-fading model applies. The exact details of how this is implemented are described in subsequent sections of the report. This lab is based on achieving three distinct milestones:

1. Simulate OFDM assuming frequency correction
2. Simulate OFDM, and perform frequency correction
3. Imlement OFDM and frequency correction on Universal Software-defined Radio Peripheral (USRP) B210 radios

1.1 Software Overview

1.1.1 a.) Without Frequency Correction

The simulation uses a single main function - **simulate_with_synchronized_clocks.m**. As illustrated in Figure 2, this function calls modular helpers to estimate the channel matrix, encode the data, send data through the channel, correct for lag, and compute the error. All project code and associated documentation is available at `https://github.com/anushadatar/ofdm-implementation/releases/tag/v.3.a/`. Each of the bolded functions in Figure 2 has its own file in this repository. Figure 2 below shows the flow of operations for channel estimation and data transmission.

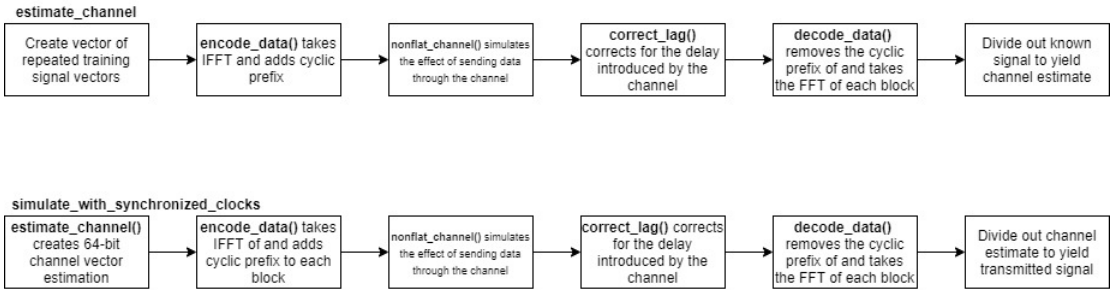

Figure 2: Software flowchart.

Below are the declarations for and a short description of each of the functions used in the diagram above. Note that the **estimate_channel** function calls many of the same helper functions as **simulate_with_synchronized_clocks**.

1. **[output_data, error_rate] = simulate_with_synchronized_clocks(x_train, x_data)** prepares OFDM data, simulates sending it across the channel, corrects and decodes it, and calculates the percent difference between the original data and the decoded data.
2. **[H_k] = estimate_channel(x_train, block_size, prefix_size)** calculates a 64-bit estimate of the channel based using a transmission vector.
3. **[x_cyclic] = encode_data(x_data, block_size, prefix_size)** takes the IFFT of and adds the cyclic prefix to each block.
4. **[y_time] = nonflat_channel(x_cyclic)** simulates the effect of sending data through the channel.
5. **[y_time] = correct_lag(x_cyclic, y_time)** corrects for the delay introduced by the channel.
6. **[y_decoded] = decode_data(y_time, number_of_blocks, block_size, prefix_size)** removes the cyclic prefix of and takes the FFT of each block.
7. **[err] = compute_error(rx_data, tx_data)** computes the percent error between the transmitted and decoded signals.

1.1.2 b.) With Frequency Correction

To more closely reflect the way that OFDM implementations manifest in actual hardware (and to simplify the workflow for the third section of this lab), the simulation that includes frequency correction uses a single function for preparing transmit data (called package_data.m) and another main function for decoding received data (called process_received_data.m). The function simulate_without_synchronized_clocks.m) simulates preprocessing data, sending it through the channel, and then decoding it. As illustrated in Figure 3, each of these functions calls modular helpers. All project code and associated documentation is available at `https://github.com/anushadatar/ofdm-implementation/releases/tag/v.3.b`. Each of the bolded functions in Figure 3 has its own file in this repository. Figure 3 below shows the flow of operations for channel estimation and data transmission.

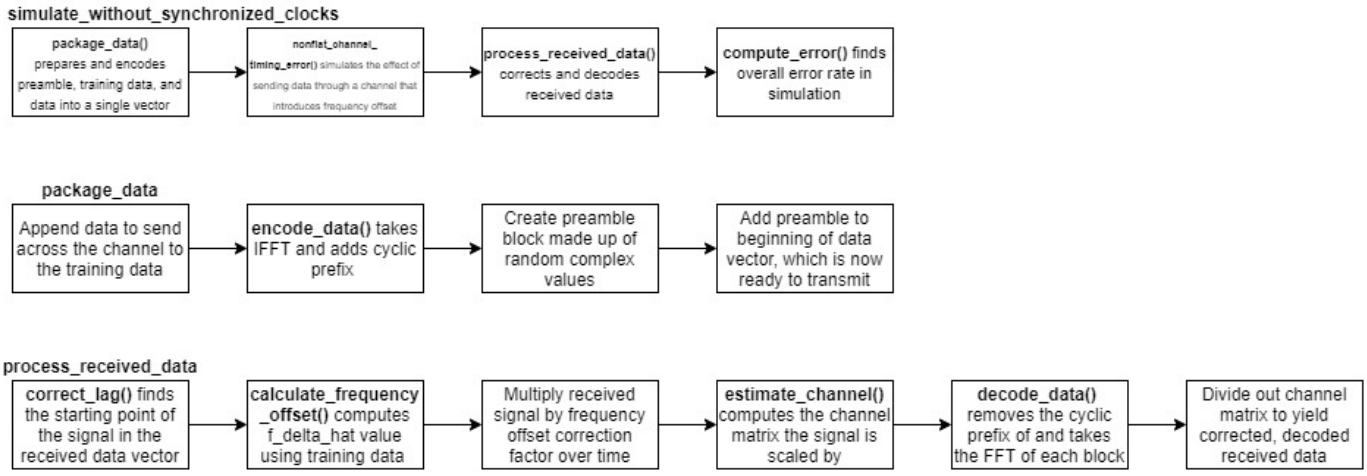

Figure 3: Software flowchart.

Below are the declarations for and a short description of each of the functions used in the diagram above. Note that the **estimate_channel** function calls many of the same helper functions as **process_received_data** (see Figure 3 for additional context).

1. **[output_data, error_rate] = simulate_without_synchronized_clocks(x_train, x_data)** prepares OFDM data, simulates sending it across the channel, corrects and decodes it, and calculates the percent difference between the original data and the decoded data.
2. **[tx_cylic] = package_data_(x_train, x_data, block_size, prefix_size, num_preamb_blocks)** creates and encodes a vector including the preamble, the training data, and the actual data such that it is ready to transmit across the channel.
3. **[y_time] = nonflat_channel_timing_error(x_cyclic)** simulates the effect of sending data through a channel that introduces a timing error.
4. **[output_data] = process_received_data(x_train, tx_cyclic, y_time, block_size, prefix_size, number_of_blocks, num_train, num_preamb_blocks)** calls several helper functions to correct and decode the received data.
5. **[f_delta_hat] = calculate_frequency_offset(y, block_size)** calculates the frequency offset value using the second and third blocks of the data.
6. **[H_k] = estimate_channel(x_train, block_size, prefix_size)** calculates a 64-bit estimate of the channel based using a transmission vector.
7. **[x_cyclic] = encode_data(x_data, block_size, prefix_size)** takes the IFFT of and adds the cyclic prefix to each block.
8. **[y_time] = correct_lag((x_cyclic, y_time)** corrects for the delay introduced by the channel.
9. **[y_decoded] = decode_data(y_time, number_of_blocks, block_size, prefix_size)** removes the cyclic prefix of and takes the FFT of each block.
10. **[err] = compute_error(rx_data, tx_data)** computes the percent error between the transmitted and decoded signals.

1.1.3 c.) In Hardware

The code associated with sending the OFDM signals in hardware includes additional functions for writing to and reading from files for the USRP BS210 radios, the addition of pilot signals for phase offset correction, and the addition of scale factors to prevent clipping. The function for preparing transmit data is called package_data.m), and the function for decoding received data is called process_received_data.m. As illustrated in Figure 4, each of these functions calls modular helpers . All project code and associated documentation is available at `https://github.com/anushadatar/ofdm-implementation/`. Each of the bolded functions in Figure 4 has its own file in this repository. Figure 4 below shows the flow of operations for channel estimation and data transmission.

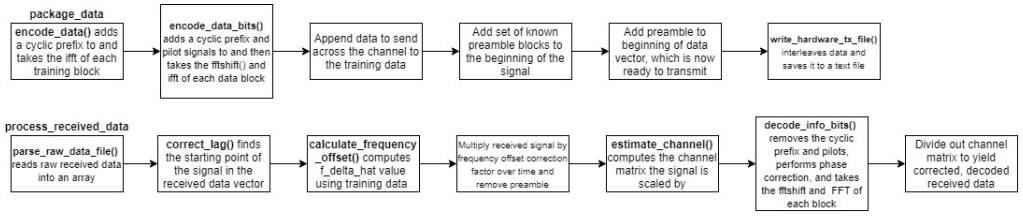

Figure 4: Software flowchart.

Below are the declarations for and a short description of each of the functions used in the diagram above. Note that the **estimate_channel** function calls many of the same helper functions as **process_received_data** (see Figure 3 for additional context).

1. **[tx_cylic] = package_data_(x_train, x_data, block_size, prefix_size, num_preamb_blocks)** creates and encodes a vector including the preamble, the training data, and the actual data such that it is ready to transmit across the channel. It then writes it to a file called "tx.dat."
2. [err, output_data] = process_received_data_hardware(rx_path, tx_path, x_train, x_data, block_size, prefix_size, number_of_blocks, num_train, num_preamb_blocks) decodes raw data from "rx.dat," decodes and correct it, and computes the error rate.
3. **[output_data] = process_received_data(x_train, tx_cyclic, y_time, block_size, prefix_size, number_of_blocks, num_train, num_preamb_blocks)** calls several helper functions to correct and decode the received data.
4. **[**
5. **[rx] = process_raw_data_file(rx_fname)** reads raw USRP BS210 received data and writes it to a MATLAB vector.
6. **write_hardware_tx_file(tx_data)** writes the MATLAB vector signal to a file called "tx.dat."
7. **[f_delta_hat] = calculate_frequency_offset(y, block_size)** calculates the frequency offset value using the second and third blocks of the data.
8. **[H_k] = estimate_channel(x_train, block_size, prefix_size)** calculates a 64-bit estimate of the channel based using a transmission vector.
9. **[x_cyclic] = encode_data(x_data, block_size, prefix_size)** takes the IFFT of and adds the cyclic prefix to each training block.
10. **[y_decoded] = encode_info_bits(x_data, block_size, prefix_size)** adds pilots and a cyclic prefix to and takes the IFFT and fftshift of each data block.
11. **[y_time] = correct_lag((x_cyclic, y_time)** corrects for the delay introduced by the channel.
12. **[y_decoded] = decode_data(y_time, number_of_blocks, block_size, prefix_size)** removes the cyclic prefix of and takes the FFT of each training block.

13. **[y_decoded] = decode_info_bits(y_time, block_size, prefix_size, number_of_blocks, H_k)** corrects for phase offset in, removes pilots of, removes the cyclic prefix of, and takes the FFT and ifftshift of each data block.

14. **[err] = compute_error(rx_data, tx_data)** computes the bit error rate (BER).

1.2 Hardware Setup

The image below in Figure 5 depicts the hardware setup used for part c, which is made up of two USRP BS210 radios facing each other approximately two feet apart.

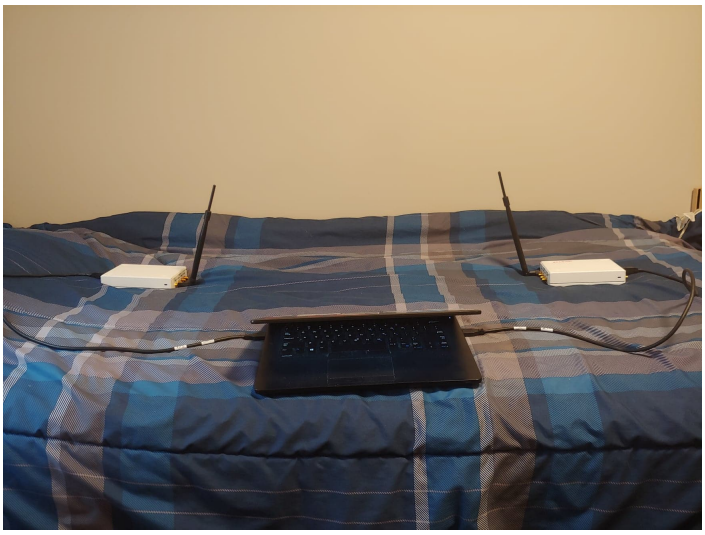

Figure 5: Hardware setup image.

A gain of 60 dB was used on each radio to amplify the signal on both the transmit and receiving end.

2 Mathematical Background

2.1 Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT)

2.1.1 Transformation Background

The Fourier transform is a mathematical transformation that decomposes functions of time into functions that depend on frequency, and does so by projecting the function onto different complex exponentials. For example, consider the function $f(t)$ in the time domain, that is projected onto a complex exponential that is oscillating at a frequency of 50 Hz. To accomplish this, one must use the inner product of $f(t)$ with the specified complex exponential (in the form $e^{-j2\pi ft}$) as follows.

$$< f, e^{-j2\pi(50)t} >= \int_{-\infty}^{\infty} f(t)e^{-j2\pi(50)t}dt \tag{1}$$

Note that the output of the above equation will be a complex number. However, in order to compute the full continuous time Fourier transform (CTFT), one must project $f(t)$ onto complex exponentials of every frequency. To accomplish this one can take the inner product of functions above and parameterize it by frequency by substituting an f for 50. This produces the following equation

$$F(f) = \int_{-\infty}^{\infty} f(t)e^{-j2\pi ft}dt \tag{2}$$

which is the continuous time Fourier transform of $f(t)$ and is denoted by $F(f)$ as it is now a function of frequency.

However, to practically apply the Fourier transform in this lab and in general, it needs to be discretized so that computers (whose digital hardware is fundamentally discrete) are able to use it to process data. In undergraduate courses, it is common to learn about the discrete-time Fourier transform (DTFT) which transforms the discrete-time signal $x[m]$ of length L as follows where Ω is normalized radial frequency.

$$X(e^{j\Omega}) = \sum_{l=0}^{L-1} x[l]e^{-j\Omega l} \tag{3}$$

Note that taking the L-point DTFT is permitted due to the assumption that $x[m] = 0$ for $m < 0$ and $m \geq L-1$. In the above equation, the input parameter specifies a complex exponential that is **continuous in frequency and discrete in time**, hence the name discrete-time Fourier transform. Another important note is that because time is discrete in the DTFT, a sum, which is effectively the discrete version of an integral, is used.

Finally, to fully discretize the CTFT, one can can sample the DTFT to produce the discrete Fourier transform which is shown below and can be calculated using the $fft()$ function in MATLAB.

$$X_k = \sum_{l=0}^{L-1} x[l]e^{-2\pi j\frac{lk}{L}} \tag{4}$$

One can calculate the inverse discrete Fourier transform (IDFT) by switching the positions of $x[m]$ and X_k, flipping the sign on the complex exponential, and dividing the summation by the length of the signal as shown below.

$$x[m] = \frac{1}{L}\sum_{k=0}^{L-1} X_k e^{2\pi j\frac{mk}{L}} \tag{5}$$

In the IDFT, the frequency domain signal is essentially projected onto complex exponentials with frequencies of opposite signs (which means that they are rotating the opposite direction of the complex plane) and normalized by the length of the signal. The DFT and IDFT are crucial to the OFDM modulation scheme. In addition to being discrete in frequency (which is necessary given that one signal is in the frequency domain when using OFDM), these transformations exhibit properties useful for OFDM which are explained in the subsequent section.

2.1.2 Properties

The DFT has several properties that are especially useful in the context of implementing OFDM systems.

1. **Linearity**
   In the case that $y[m] = \alpha p[m] + \beta q[m]$, the DFT of $y[m]$ is related to the DFTs of $p[m]$ and $q[m]$ by:
   $$Y_k = \alpha P_k + \beta Q_k \tag{6}$$

2. **Periodicity**
   The L-point DFT of a signal is periodic with period L. In other words, If X_k is the L-point DFT of $x[m]$, then:
   $$X_{k+L} = X_k \tag{7}$$

3. **Circular convolution in time becomes multiplication in frequency**
   Circular convolution of two discrete signals with period L corresponds to a multiplication of those signals' DFTs. More specifically, one can define the L-point periodic extension of $x[m]$ as follows:
   $$\widetilde{x}[m] = x[m \ mod \ L] \tag{8}$$

If X_k and H_k are the DFTs of $x[m]$ and $h[m]$ respectively (assuming that both DT signals are zero when $m < 0$ and $m \geq L-1$), then $y[m] = \widetilde{x} * h[m]$ and Y_k is the L-point DFT of $y[m]$. This allows one to model the DFT of the received signal as the product of the DFT of the channel and the DFT of the input signal.

$$Y_k = H_k X_k \tag{9}$$

This implies that in the case of OFDM, one can model the channel as a discrete signal and then divide out its value to find X_k if the channel performs circular convolution on the transmitted signal. How this is carried out is discussed later.

2.2 Channel Estimation

2.2.1 Continuous-Time Introduction

Just like with other modulation schemes used in communications systems, one needs to perform some sort of channel estimation. To estimate the channel for OFDM, a similar approach to MIMO channel estimation is used where a training signal (which is known by both the transmitter and receiver) is sent over the channel. However, OFDM is used in cases when the channel is "messy" as previously discussed, so it is necessary to calculate a higher-resolution estimate of the channel rather than a single complex coefficient to use under the assumption of the flat-fading channel model. In the implementation of OFDM for this lab, the "messy" channel is split into 64 small sub-carriers which are constant enough in magnitude to apply the flat-fading channel model.

First, one can assume that the data to be transmitted over the OFDM channel (± 1 for the BPSK data streams in this lab) are signaling in the frequency domain. An example of "1" encoded as a pulse in the frequency domain is shown below in Figure 6 where f_Δ is the spacing between sub-carriers which target different locally flat regions of the overall non-flat channel.

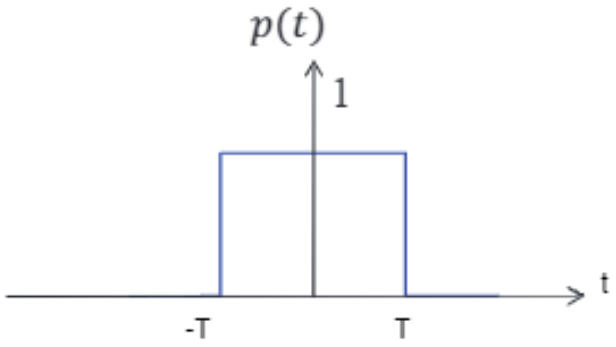

Figure 6: One bit of data signaled in the time domain [1].

Assuming the training signal to be sent is x_{train}, k is the discrete-time index, l denotes which sub-carrier is being used, p is a pulse function, and a_{kl} is the bit encoded at time k on sub-carrier l, the training signal can be represented in continuous time with the following equation.

$$x_{\text{train}}(t) = \sum_{k=-\infty}^{\infty} \sum_{l=0}^{63} a_{kl} p(t - kT) e^{j2\pi l t f_\Delta} \tag{10}$$

Note that this setup requires sending data over the channel in a 64-bit block. The next step is where the magic of OFDM happens, is why one encodes data used in OFDM in the frequency domain, and is the core of what inspires its name. To prepare the encoded data to send over the channel one must take its IDFT when implemented digitally to transform it into the time domain. For the sake of this explanation, consider the CTFT of the 64 sub-carriers at the zeroth symbol (where $k = 0$ in Equation 10). First note that Equation 10 when $k = 0$ is as follows

$$x_{\text{train},0}(t) = \sum_{l=0}^{63} a_{0l} p(t) e^{j2\pi l t f_\Delta} = p(t) \sum_{l=0}^{63} a_{0l} e^{j2\pi l t f_\Delta} \tag{11}$$

First, note that the summation in Equation 11 closely resembles the IDFT. Thus what is transmitted is the IDFT of the discrete pulses. However, to discover how the channel "sees" the data, one must take the Fourier transform of the transmitted data. After taking the CTFT of Equation 11, the following (as shown in 7) is obtained. This equation includes a summation of sinc functions lined up as shown in Figure 11 below.

$$X_{\text{train},0}(f) = \left(\frac{sin(\pi f \frac{1}{f_\Delta})}{\frac{\pi f}{f_\Delta}} \right) * \left(\sum_{l=0}^{63} a_{0l} \delta(f - l f_\Delta) \right) \tag{12}$$

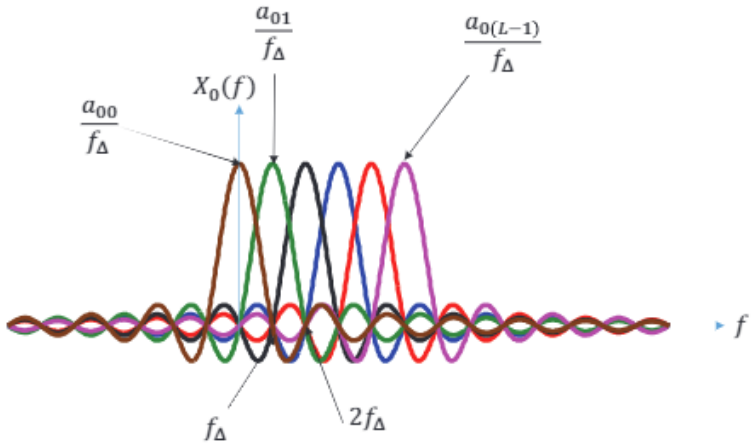

Figure 7: A symbol of the OFDM signal prepared to be sent across the channel [1].

Note that $L = 63$ in the Figure 7 above. The main peak of each sinc function aligns with the zeros of all the others. This is where the "orthogonal" comes from in the name OFDM. If f_Δ is small enough, than each of the sinc sub-carriers can be considered a narrow-band signal (because the main lobe of each is where the majority of the power is of each sinc function).

At this point it may seem logical to send the above signal over the channel and use the fact that the receiver also knows x_{train} to solve for the channel coefficient that is applied to each sub-carrier. However, when implementing OFDM digitally, because signaling occurs in the frequency domain, the use of the DFT and IDFT requiring accounting for the circular convolution discussed above. In other words, one must account for the fact that, when using the DFT/IDFT, multiplication in the frequency domain is not convolution in the time domain, but rather circular convolution.

2.2.2 Discrete Time Implementation

Starting from the beginning, this time in discrete time, the steps to carrying out OFDM are as follows:

1. Create a 64-bit training vector to transmit across the channel. Each bit here translates to one subcarrier in an OFDM symbol.

$$X_{k,\text{train}} = \begin{bmatrix} X_0 & X_1 & X_2 & \dots & X_{63} \end{bmatrix} \tag{13}$$

2. The training data vector created above is signaling in the frequency domain. One must take the 64-point IDFT of the x_{train} to transform it into the time domain so that orthogonal sinc functions can divide the frequencies of the channel into small flat bands so that one can derive a channel coefficient for each of the narrow-band sub-carriers.

$$x_{k,\text{train}}[m] = IDFT\{X_{k,\text{train}}\} = \begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_{63} \end{bmatrix} \tag{14}$$

3. Here is where the discrete implementation differs greatly from the continuous, for **circular convolution** in the time domain is multiplication in the frequency domain. Mathematically, to perform a circular convolution in discrete time, one takes the 64-bit vector and lines them side-by-side infinitely which is expressed as follows

$$\bar{x}_{k,\text{train}}[m] = x_{k,\text{train}}[m \ mod \ 64] \tag{15}$$

Given the periodic extension of the time-domain 64-bit block in Equation 15, the equation $y[m] = \bar{x}[m] * h[m]$ describes the periodic extension being transmitted through a channel via circular convolution. Given that Y_k, X_k, and H_k are the DFTs of their respective signals, the following equation is also true.

$$Y_k = H_k X_{k,\text{train}} \tag{16}$$

However, the caveat here is that when sending an infinite signal over the channel, it will take an infinite amount of time to send. Thus, the transmission rate is effectively $0 \frac{\text{bits}}{\text{second}}$.

The way to solve this problem is to append just enough of the end of the signal to the front of the data to "trick" the channel into performing circular convolution. Thus, one must define $\bar{x}_{k,\text{train}}$ not to be an infinite periodic extension, but a periodic extension that adds the last 16 samples of a 64-bit block (known as the cyclic prefix) to its front before sending. This makes the transmitted data 80 samples.

$$x_{k,\text{train}} = \begin{bmatrix} x_{48} & x_{49} & \dots & x_{63} & x_0 & x_1 & x_2 & \dots & x_{63} \end{bmatrix} \tag{17}$$

Additionally, the OFDM standard assumes that the impulse response of the channel is 16 samples long, meaning that if one appends the last 16 samples of a 64-bit block to its beginning, one can line up many blocks with a cyclic prefix to send in one large block. The reason for this is that there will be a point, during the convolution with the channel, when the impulse response of the channel is completely within the cyclic prefix. As a result, there will be no cross-pollination between OFDM data blocks. So as a final step, 100 of these cyclically prefixed training signals are lined up to send at once so that the final channel estimation can be an average of the estimation calculated from each of the known "blocks". However, note that, the rest of the steps here are explained as if there is only one block for simplicity. In Figure 8 below there are two 64-bit OFDM blocks with 16-bit cyclic prefixes (80 samples each in total).

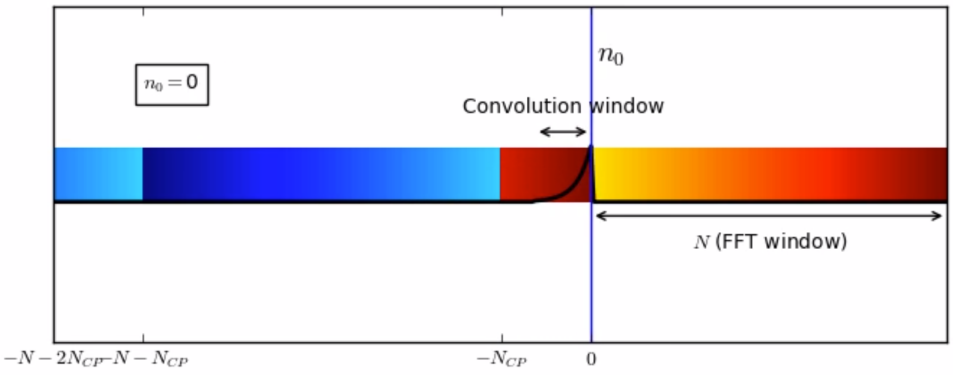

Figure 8: Two 64-bit OFDM blocks with cyclic prefixes (80 samples each in total) being convolved with the channel [2].

Note that, at the depicted part of the convolution, the channel impulse response is completely within the cyclic prefix. As a result, the OFDM data block from the first 80-sample chunk will not affect the next because of the cyclic prefix "barrier". This is a nice additional benefit.

4. Next, one must remove the cyclic prefix from the received data, $y[m]$, which entails removing the first 16 elements of the received signal to obtain the following:

$$y[m] = \begin{bmatrix} y_0 & y_1 & y_2 & \dots & y_{63} \end{bmatrix} \tag{18}$$

5. After removing the cyclic prefix, one must compute the DFT of the received signal because the transmitted bits were encoded in the frequency domain.

$$Y_k = DFT\{y[m]\} \tag{19}$$

6. At this point in the channel estimation process, $X_{k,\text{train}}$ and Y_k are known by the receiver. Thus one can use the relationship $Y_k = H_k . * X_{k,\text{train}}$ to solve for H_k as follows.

$$H_k = \frac{Y_k}{X_{k,\text{train}}} \tag{20}$$

Note that the above division is element-wise. H_k is a 64-bit vector which contains coefficients to equalize the portion of the channel by which each sub-carrier is processed. Here, it is important to note that when sending a long string of 100 "blocks" each must separately be divided by H_k and then averaged. A channel estimation is now obtained!

2.3 Data Transmission

Generating a channel estimate allows for sending a data vector (i.e., a vector of data not known by the receiver) across the channel. To do so, the transmitter divides the data vector into 64-bit blocks. Then, it leverages the same strategy used in channel estimation of taking the IFFT of each block and then inserting a cyclic prefix containing the last 16 bits of each block to the beginning of the data block. This step facilitates the circular convolution of the block. The receiver then can remove the cyclic prefix (i.e. the first sixteen bits) from each block of the received data and compute the DFT to yield Y_k, which is equal to the product of the channel and the transmit data. To recover the original signal, the receiver divides out the channel estimate for each block to perform channel equalization.

$$X_{k,\text{data}} = \frac{Y_k}{H_k} \tag{21}$$

The diagram in Figure 9 below illustrates this process for a single block. A data transmission may consist of many side-by-side blocks, but the overall process should be conducted on each individual block (and not on the entire vector).

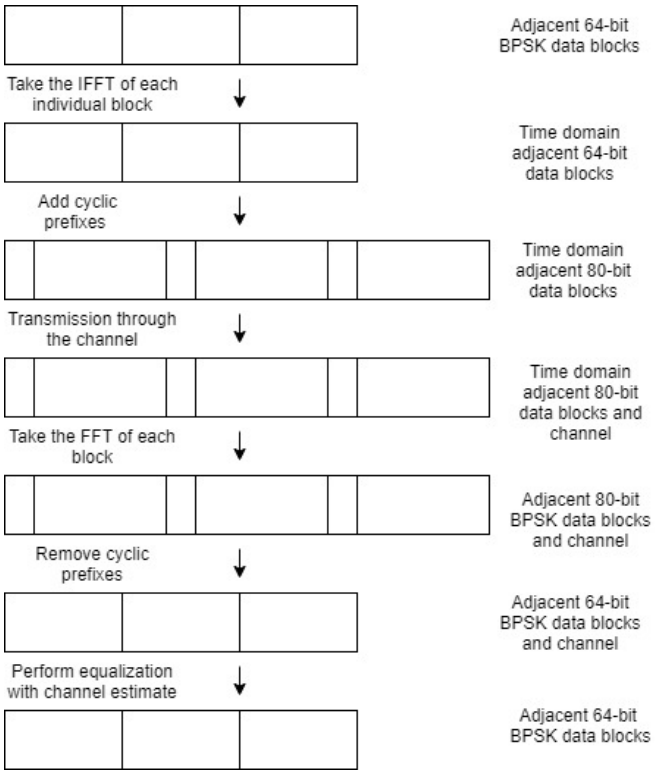

Figure 9: Data Transmission Diagram.

2.4 Frequency Offset Correction and the Schmidl-Cox Algorithm

The second milestone for this lab is to once again simulate transmitting data using OFDM modulation; however, in this iteration the modeled channel includes a frequency offset. To begin, note that the system channel with a frequency offset can be modeled as follows

$$y[k] = h * x[k]e^{jf_\Delta k} + n[k] \tag{22}$$

where h is the impulse response of the channel, x is the transmitted signal, n is any additive noise, and f_Δ is the frequency offset imposed by the channel. To account for the frequency offset imposed by the channel, the solution is clearer if Equation 22 is rewritten with the substitution $x'[k] = h * x[k]$ as shown below.

$$y[k] = x'[k]e^{jf_\Delta k} + n[k] \tag{23}$$

In order to exploit this model to extract and correct for the frequency offset, one transmits a data sequence called the **long training sequence (LTS)** appended to the front of the training sequences used for channel estimation and the actual data itself. This sequence typically consists of two identical 64-bit blocks of data along with the last 32 bits of the 64 bits appended to the front. However, for simplicity, the LTS used in this lab was three identical 64-bit blocks of data as shown below where the subscripts follow the format $x_{block,index}$.

$$\text{LTS} = \begin{bmatrix} x_{1,0} & \dots & x_{1,63} & x_{2,0} & \dots & x_{2,63} & x_{3,0} & \dots & x_{3,63} \end{bmatrix} \tag{24}$$

Note that the LTS is created in the time domain and attached to the front of the already encoded channel estimation training sequences and data. Each $x_{block,index}$ is a pseudo-random complex number $\pm 1 \pm j$

because when one signals the data in the frequency domain (with ± 1 for BPSK) and takes the IDFT to transform it into the time domain, the result is complex numbers in the time domain.

Upon further analysis of the component of $x'[k]$ that contains the LTS (i.e. the LTS convolved with the impulse response of the channel) blocks two and three will contain identical samples on the transmission end as well as the receiving end of the system. In other words, if the first 192 bits of $x'[k]$ are the LTS, the following equations hold.

$$\begin{aligned} x'[65] &= x'[129] \\ x'[66] &= x'[130] \\ &\vdots \\ x'[128] &= x'[192] \end{aligned} \tag{25}$$

In order to exploit this fact, one must take the corresponding outputs of the overall system $y[k]$ and divide the one with the larger index by the one with the smaller index. To understand why, take $y[129]$ and $y[65]$ as an example. When these are divide in the way described, it yields the following equation

$$\frac{y[129]}{y[65]} = \frac{x'[129]e^{jf_\Delta 129} + n[129]}{x'[65]e^{jf_\Delta 65} + n[65]} \tag{26}$$

assuming that **n = 0** and using the fact that $x'[65] = x'[129]$, Equation 26 above can simplify to the following.

$$\frac{y[129]}{y[65]} = e^{jf_\Delta 64} \tag{27}$$

Now, calculating the phase of the exponential in Equation 27 above and dividing by 64 would yield an estimate of f_Δ; however, one is able to perform 63 more estimates in this same manner using all the equalities listed in Equation 25. Once all of these estimates are calculated one can simply average them in order to account for random variability due to noise and produce an accurate final estimate. This whole operation is carried out according to the following equation.

$$\hat{f_\Delta} = \frac{1}{64}\left(\frac{1}{64}\left(\angle\frac{y[129]}{y[65]} + \angle\frac{y[130]}{y[66]} + \cdots + \angle\frac{y[192]}{y[128]}\right)\right) \tag{28}$$

Note that the first factor of $\frac{1}{64}$ is a part of calculating the mean of all the calculated phases, and the second factor of $\frac{1}{64}$ is to divide out the 64 coefficient of f_Δ in the exponent of e. Finally, one can use the estimated frequency offset to correct the whole received signal by multiplying by a factor of $e^{-jf_\Delta k}$ as shown below in Equation 29.

$$y_{\text{corrected}}[k] = y_k e^{-jf_\Delta k} \tag{29}$$

An added benefit of an LTS which has relatively high signal power when compared to the channel estimation training sequences and data is that one can use it to find the start of the transmission in the received data using the cross-correlation. The below figure shows the absolute value of the raw received signal.

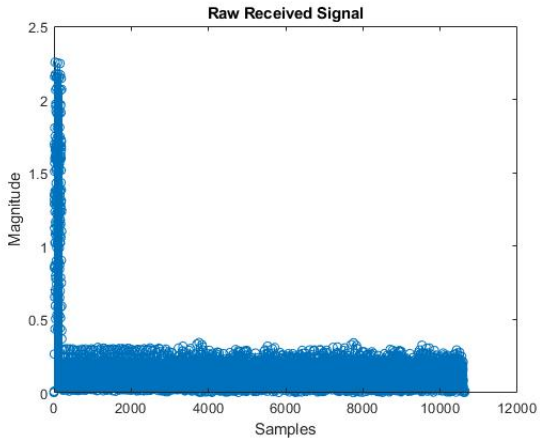

Figure 10: Plot of the absolute value of the raw received signal. The large spike at the front is the LTS.

The large spike at the front of the raw received signal shown above in Figure 10 is the LTS whose relatively high power is effective in measureing the signal delay with the cross-correlation and correcting for it.

2.5 Hardware Transmission

2.5.1 Phase Offset Correction

Equation 30 describes the signal following lag correction. Here, h corresponds to the channel estimate and $n[k]$ corresponds to the noise introduced by the channel, and $e^{j*f_\Delta k+\theta_k}$ corresponds to the frequency and phase offset.

$$y[k] = (h * x[k])e^{j*f_\Delta k+\theta_k}) + n[k] \tag{30}$$

The Schmidl-Cox algorithm described in section 2.4 allows for the estimating and accounting for f_Δ, but it does not provide a means by which to account for the phase offset, θ_k. This phase offset corresponds to the drift in the carrier frequency offset over time within the course of the block.

To account for the drift in the carrier frequency offset, one must first assume that θ_k is approximately constant over the course of a single block. Then, one can add known pilot signals to the transmit data that are regularly spaced in the frequency domain. By comparing the received value of these known pilot signals to the known value, one can find the mean value of the phase offset associated with the particular block. This simple estimation strategy accounts for block-by-block fluctuations. Assuming that one has already accounted for the effects of the channel magnitude and frequency offset, only the effects of a phase offset remains. Thus, if one calls θ_k the constant phase offset assumed for a given 80-bit block, one can represent the system at this point with the following equation.

$$Y_k = X_k e^{j\theta_b} \tag{31}$$

Thus, one can solve for this phase offset through the following calculation

$$\theta_b = \angle \frac{Y_k}{X_k} \tag{32}$$

If a few of these pilot signals are scattered throughout a block, one can derive a phase offset estimation from each and average to obtain a more accurate estimation. For example, if an implementation includes pilot symbols at indices 7, 26, 40, and 59 within each block, finding this average requires the use of equation 33 (assuming that Y corresponds to the received signal block and X corresponds to the known value of the pilot at that index).

$$\hat{\theta}_b = \frac{1}{4}(\angle \frac{Y_7}{X_7} + \angle \frac{Y_{26}}{X_{26}} + \angle \frac{Y_{40}}{X_{40}} + \angle \frac{Y_{59}}{X_{59}}) \tag{33}$$

After finding that average, one can account for it by multiplying each block by $e^{-j\theta_b}$ to cancel out the existing phase offset.

2.5.2 Guardbands

Preparing the data for transmission using hardware also requires the addition of guardbands to each block. This prevents individual blocks from interfering with each other and accounts for the roll-off associated with the low-pass filter in the radio (as it would attenuate the data bits on the most positive and negative frequencies in the data). In the case of this implementation, the first six bits and the final five bits of each block (prior to the addition of the cyclic prefix) serve as guardbands. Because the system can simply account for any interference or attenuation when the transmitted data is known, neither the training data nor the preamble contain guardbands.

2.5.3 DC Offset

When the radio modulates the signal for transmission across the air, it convolves the original signal with a sinusoid centered at the center index of the data block its transmitting. The result of this convolution interferes with the value of the original signal, for it aligns the impulses created by the sin or cos carrier signals at f_c with the 0 Hz frequency bin of the original signal. For that reason, the signal does not include any data at the center index of the block.

2.5.4 Block Diagram

The diagram in figure 11 shows the layout of a single data block.

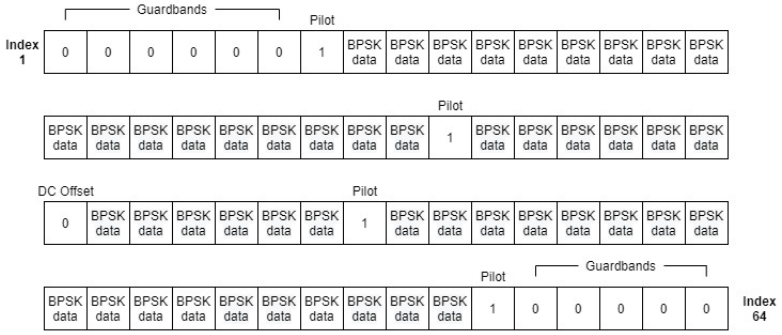

Figure 11: Block diagram of data block prior to cyclic prefix encoding and transmission.

This configuration accounts for the need for guardbands, pilot signals, and the DC offset.

3 Results

3.1 Simulation - Assuming Frequency Correction

In Figure 12 below is a comparison of the actual frequency response of the channel (left) with the estimated frequency response.

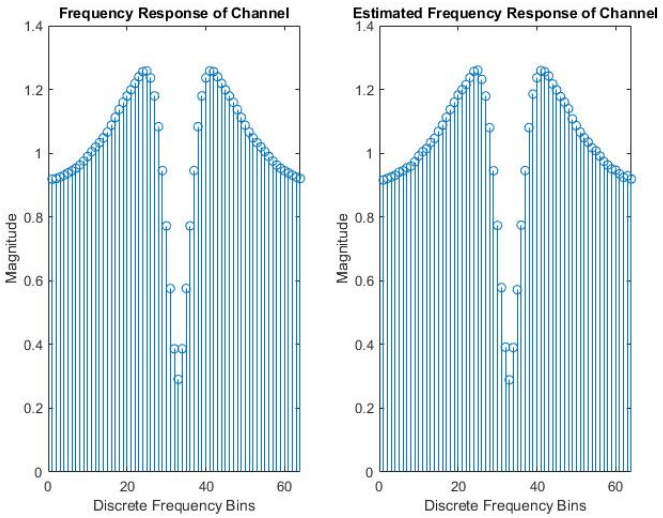

Figure 12: Comparison of the frequency response of the channel to the estimated frequency response of the channel.

It is clear that they are almost identical, so the method of using 100 64-bit training signals to calculate individual channel estimates which are averaged proved successful.

Figure 13 below is a comparison of the first ten bits of the transmitted BPSK signal with the first ten bits of the received BPSK signal.

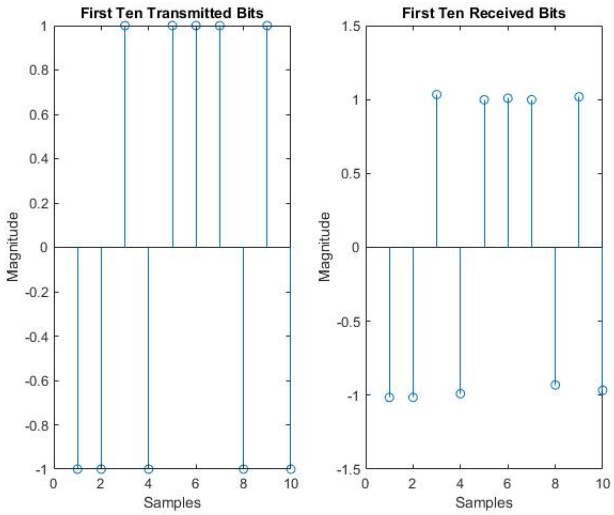

Figure 13: Comparison of the transmitted and received signals using OFDM modulation.

It is also clear in this comparison that the two are almost identical with the exception of small magnitude differences in the received data which is most likely due to noise introduced in the channel. These results are comparable to the level of magnitude accuracy achieved in the first section of this lab.

In the constellation plot of the received data in Figure 14 below, there is clear separation between the ± 1 BPSK values.

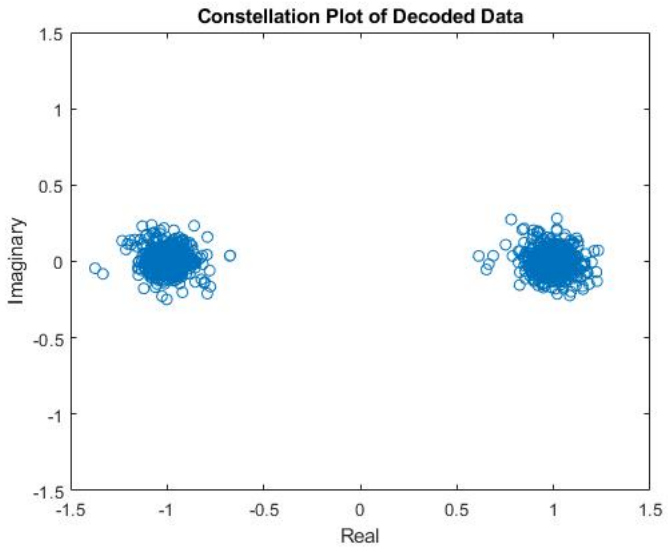

Figure 14: Constellation plot of received data.

Additionally, they variance among the data points along the real and imaginary components of the signal are quite small as one can observe in Figure 13, where the magnitudes of the shown bits from the received signal are close to ± 1.

3.2 Simulation - With Frequency Offset

Figure 15 below shows the actual frequency response of the channel the estimated frequency response.

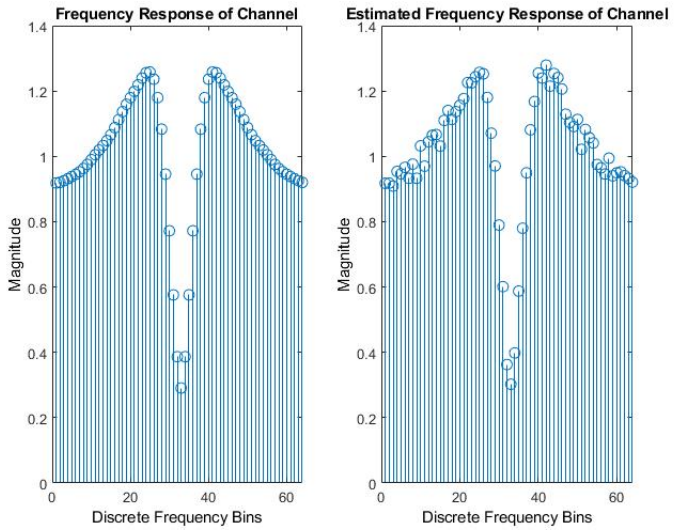

Figure 15: Comparison of the frequency response of the channel to the estimated frequency response of the channel.

The similarity of the two figures indicates that the channel estimation method accurately characterizes the channel and allows for dividing out its effects. This matches the accuracy of the channel estimation

performed in the simulation without the frequency offset.

Figure 16 below is a comparison of the first ten bits of the transmitted BPSK signal with the first ten bits of the received BPSK signal.

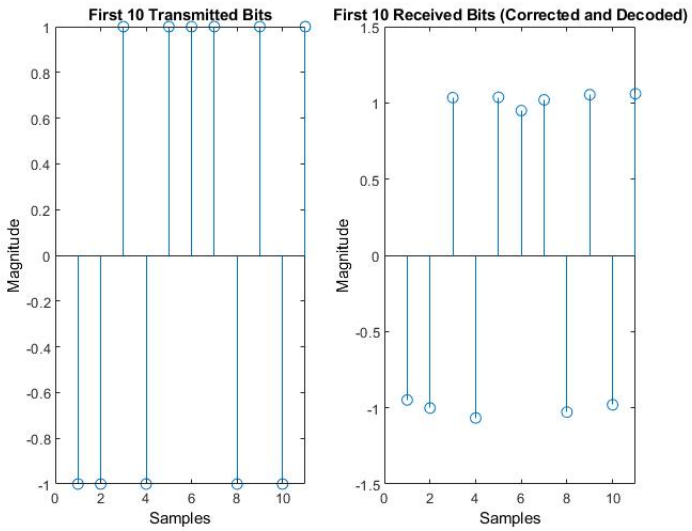

Figure 16: Comparison of the transmitted and received signals using OFDM modulation following correction for frequency offset.

This comparison illustrates that the signals themselves are almost identical in value with the exception of small magnitude differences likely due to slight inaccuracy of the channel estimation and additive noise.

The constellation plot of data uncorrected for frequency offset is shown in Figure 17 below, and illustrates the unacceptable error introduced by the channel when the frequency offset is not corrected.

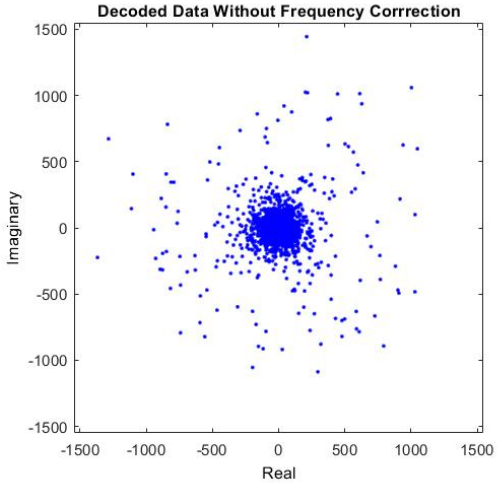

Figure 17: Constellation plot of received data.

This error occurs largely because of the frequency offset skews the received data in the complex plane. This

diverges from the initial uncorrected data in the first section of the lab, as As illustrated below in Figure 18, correction and decoding yields two separate BSPK signal values.

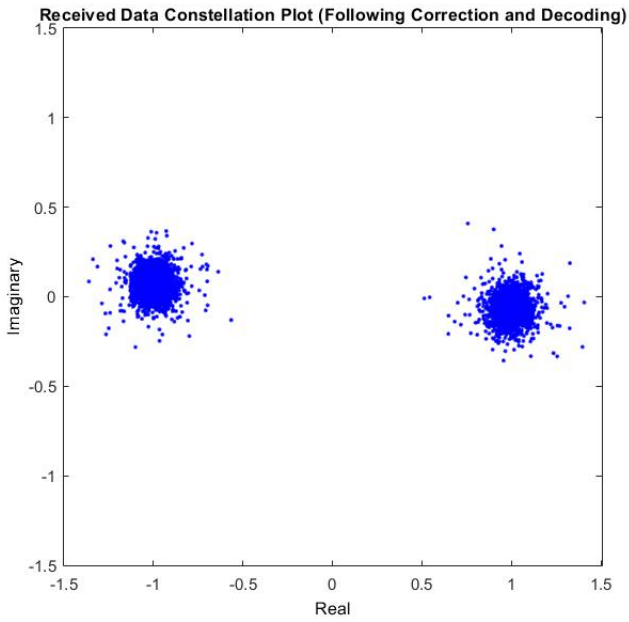

Figure 18: Constellation plot of received data.

This constellation plot shows that there is nontrivial variance in both the real and imaginary components of each of the signals; while they certainly represent two separate points, the effects of the channel remain apparent. It is also notable here that a small frequency offset remains, as the BPSK clusters are slightly rotated. This phase offset could likely be due to inaccuracy in the estimation of the channel or slight inaccuracy in timing lag estimation. Using a slightly inaccurate channel estimation to equalize the effects of the channel in the received data could have inadvertently introduced unwanted phase offset into the simulation.

3.3 Hardware Implementation

When implementing phase offset correction and adding guard bands to the received, there are a few "gotchas" that one should keep in mind.

1. One should complete phase correction **AFTER** both frequency offset correction and channel equalization have been performed.

2. If one uses the function **fftshift** in the implementation, ensure that the channel estimation vector is appropriately shifted such that corresponding elements are divided.

For this lab, the transmit vector consisted of three identical 64-bit blocks to use with the Schmidl-Cox frequency correction algorithm, ten identical 64-bit blocks to use in an estimation of the channel frequency response, and then 100 64-bit blocks which each contained 48 bits of actual raw data bits as shown above in Figure 11. The other 12 were zeros inserted to create guard bands to protect the precious data bits from the roll off of the non-ideal band-pass filter in the B210 radio, a zero at the DC frequency component to protect from spillover from the carrier signals, or known pilot signals used to correct for a phase offset.

Below in Figure 19 is the section of the signal received by the B210 radio which contains the actual transmitted data as described in the above paragraph.

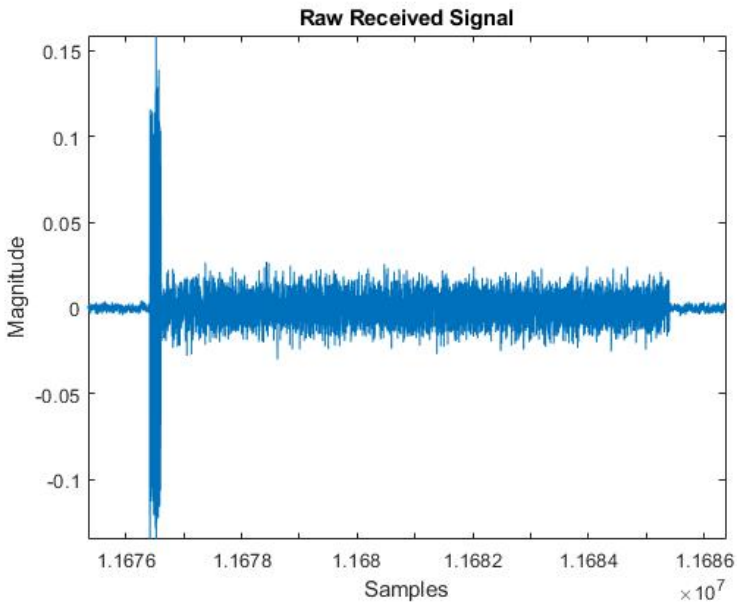

Figure 19: Section of the data received by the B210 radio that contains the transmitted data. Cross correlation was used to detect this subset of samples.

It is clear that in Figure 19 above, the signal power is less than what was simulated in Figure 10. This indicates that the Tx/Rx gain could be further increased from the 60 dB setting that was used; however, the SNR in this transmission was sufficient enough for extracting the data.

Below in Figure 20 is an estimate of the frequency response of the channel between the B210 radios which transmitted and received the data.

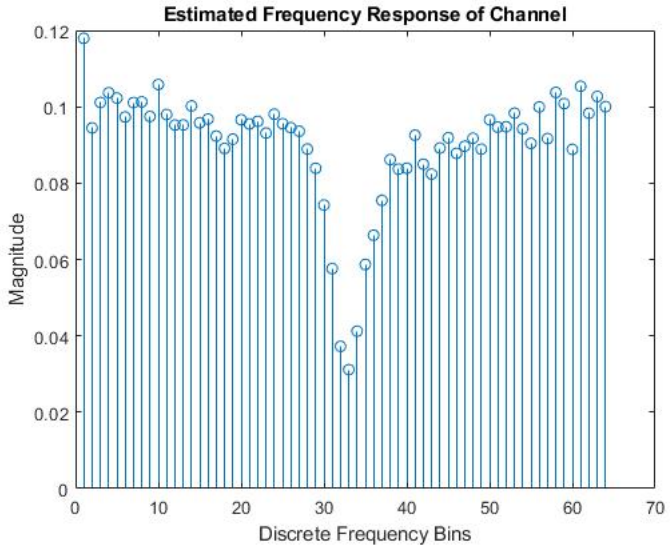

Figure 20: Estimated channel calculated by sending ten training signals across the "real" hardware channel.

The estimated channel impulse response resembles the form of the simulated channel as they both have

a sharp dip in their centers (which corresponds to the expected roll-off of the band-pass filter). However, while the simulated channel curves downward on either side of the dip as seen in Figure 13 or Figure 16, the channel frequency response here slopes gently upwards on either side of this regime. This is understandable, as the actual channel the data was sent through is not going to perfectly match that of the simulation.

Below in Figure 21 are the fist ten bits of both the transmitted data and the received data after having been fully corrected for frequency/phase offsets and decoded.

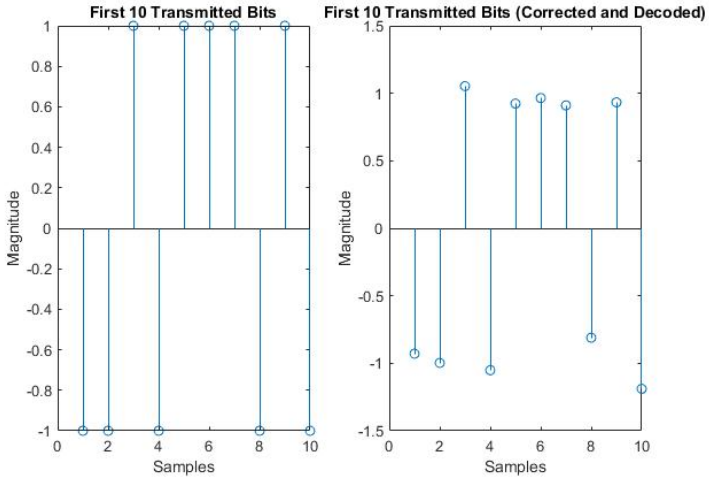

Figure 21: The first ten transmitted data bits and the first ten received data bits after frequency/phase correction and decoding.

The signs of each of the corresponding bits are identical; however, the magnitudes in the bits received by the hardware are not perfectly valued at ± 1, for there is noise introduced by the channel. Additionally, the corrections for phase, frequency offsets, and channel equalization were not perfect.

Below in Figure 22 is the final constellation plot for the data transmitted and received using the BPSK modulation scheme with hardware.

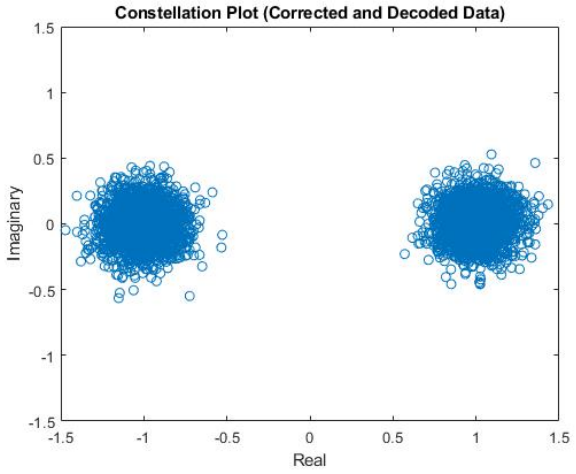

Figure 22: Constellation plot of the data received, corrected, and decoded by the B210 USRP hardware.

Each of the BPSK constellation points in Figure 22 seem symmetric and similar in their variance (indicating that the frequency correction was effective) and approximately even with the real axis (indicating the the phase correction was also effective). The bit error rate for this transmission was 0%.

To calculate the transmit rate of this system, one must account for the fact that the B210 radio is transmitting/receiving at a rate of $2 \times 10^6 \frac{\text{bits}}{\text{second}}$, and, after the training sequences for frequency correction and channel estimation, there are only 48 actual data bits sent in an 80-bit block. Using these values one can calculate the amount of time it takes to send an 80-bit block via the following equation.

$$\frac{1}{2 \times 10^6} \frac{\text{seconds}}{\text{bit}} \times 80 \text{ bits} = 4 \times 10^{-5} \text{ seconds} \tag{34}$$

Next, since in this amount of time, only 48 of those 80 bits are actually raw data (i.e. not guard bands or pilot bits), one can calculate the effective transmit rate as follows.

$$\frac{48}{4 \times 10^{-5}} \frac{\text{bits}}{\text{second}} = 1.2 \text{ Mbps} \tag{35}$$

4 References

[1] Govindasamy S., "OFDM I: Intro and The Discrete Fourier Transform".

[2] "The Cyclic Prefix in OFDM", https://dspillustrations.com/pages/posts/misc/the-cyclic-prefix-cp-in-ofdm.html