Using MATLAB to Understand Modulation of Radio Signals

Sumanth Mallya

Revision 10/05/2015

Table of Contents

Table of Figures

Introduction

The final report for Experiment 2 will cover an overview of the first two parts of the experiments, previously completed, as well as the third part. The first part of the experiment studied the process of phase modulation, specifically focusing on the behavior of Binary Phase Shift Keying (BPSK). Figure 1 displays the natural behavior of BPSK, switching between its two phases. Building on the knowledge previously obtained, the first part of the experiment transmitted and received the data message "Kilroy was here". The tools utilized as transmitter and receiver were a set of speakers and a microphone, respectively. In order to assure a successful communication it is vital that the transmitter and receiver should have equal: carrier frequency, type modulation, bit rate and training sequence. [1]

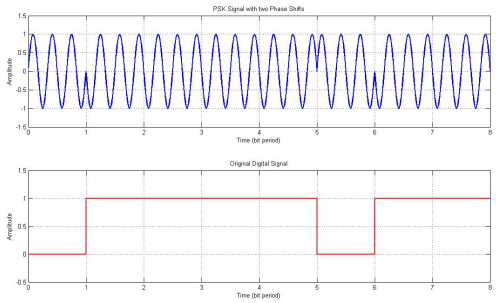

Figure 1 BPSK modulation example [2]

The second part of the experiment focused on receiving and interpreting the data, previously modulated and transmitted. Different frequencies were tested in order to understand the effects of the recording devices and the Doppler Effect. Finite Impulse Response (FIR) filtering was utilized as the filtering mechanism to remove the portion of the signal that is twice the original carrier frequency. Figure 2 bellow displays a typical setting for transmitting the data package using the speakers built in the computers that were used on this portion of the experiment.

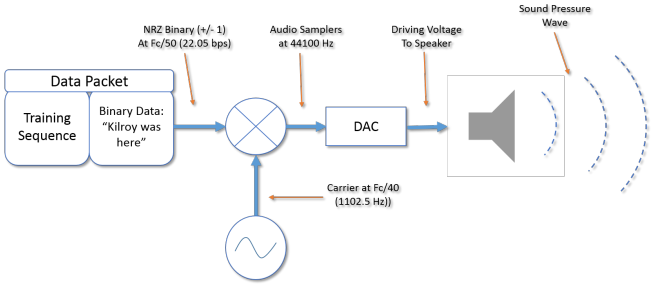

Figure 2 Block Diagram of the transmit task [1]

Receiving the data that has been transmitted is a more complicated task, because it requires more steps. Obtaining quadrature detection is desired in this step; therefore the receiver multiplies the data by a complex copy of its carrier. Figure 3 below displays how the receiver software looks like, and it can be easily recognized that it is more complex than the transmitter.

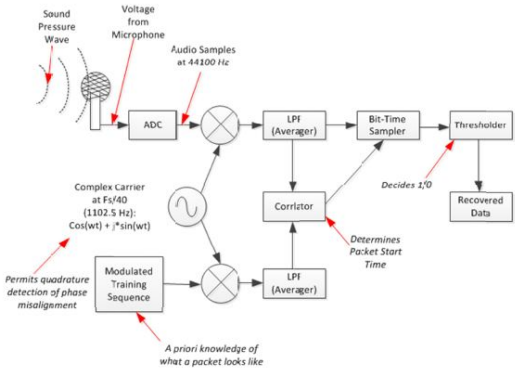

Figure 3 Block Diagram of the receiving software [1]

In order to detect and interpret the correct message from the data, a correlator is utilized in the last part of the experiment. The correlator's principal function is to detect the beginning of the packet transmitted. In order to detect its start, the correlator searches for the known training sequence of the data. Once the training sequence is obtained, the signal output that the correlator emits is high, after obtaining this output a packet of one audio sample per bit is extracted. This packet on the bit time sampler will determine if a 0 or a 1 is transmitted at that given time.

Experimental Procedure

Part 1

- The carrier frequency, sampling rate and bit rate were determined and kept in integral multiples in order to have ease of calculation
- The carrier wave, used to carry the data was generated. A sine wave was chosen for the task.
- The data that needs to be transmitted was converted from text to decimal characters and a binary sequence was generated from the decimal numbers. Similarly a random sequence of numbers was used to create a preamble, which was used as a header to the data in order to help with packet detection during the demodulation phase.

- The data to be transmitted was differentially modulated onto the carrier wave and the preamble was directly modulated onto the carrier wave and the two were combined to generate the modulated sequence.
- Record and play devices were created in MATLAB, the output and input audio devices on the computer were checked in order to eliminate normalization and auto noise reduction by the computer's operating system.

Part 2

- The data was played and recorded in different sequences namely
    - Play and record on the same system
    - Play and record on different systems
    - Play and record on different systems while the recording system was in motion.

- The recorded data was then extracted from the received wave and phase changes were analyzed to confirm good transmission and recording.
- In accordance with BPSK modulation, the recorded data was mixed down to baseband by multiplying the recorded data with an in phase component defined by a cosine wave and a quadrature component defined by a sine wave.
- The data was then run through a moving average filter in order to remove the data present at frequencies outside of the carrier frequency.

Part 3

- A correlator was generated by using the filtered preamble (which is known to the receiver) and the filtered data, based on the results the data packet was located in the recorded data at the maximum value of the correlator output.
- Required data was then extracted from the filtered data based on the number of bits and the number of samples per bit.
- The preamble was segregated from the data, and data was converted back to text.

Required Resources

- MATLAB R2013b
- High Definition Audio device
- Microphone device

Results

In part one of this experiment the data "Kilroy was here" was modulated onto a carrier wave, with a predetermined frequency of 44100Hz, using binary phase shift keying (BSPK) technique. This was accomplished by first converting the data into binary and then creating a preamble, also known as a training sequence, from a pseudorandom number generator. The preamble is used to determine if there is incoming data so that the demodulation code knows where to search for the data packet in the recorded data. Once the preamble and data were converted to binary, direct BPSK was applied to the preamble while Differential BPSK was applied to the data sequence. Using Direct BPSK on the preamble and Differential BPSK on the data helps distinguish between the preamble and data in the packet. With BPSK applied to the two sequences, the data sequence was concatenated onto the end of the preamble sequence to generate the full carrier waveform. Figures 4 and 5 below show the waveform that was generated in part one of the experiment.

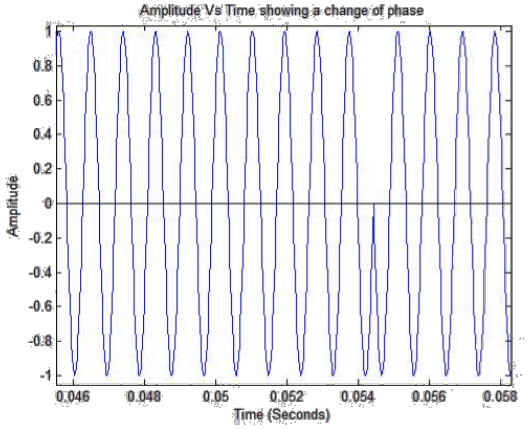

Figure 4: Phase Change in Data fc = 1102.5Hz, Rb= 18.75Hz

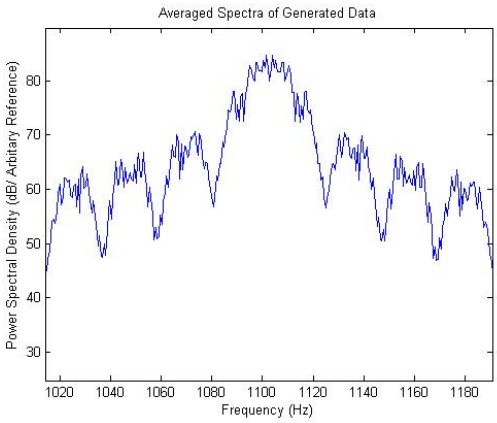

Figure 5: Averaged Data Spectra at fc = 1102.5Hz, Rb = 18.75Hz

Figure 4 shows a phase change in the generated wave that was sent from the code. BPSK works through changes in the phase of the wave corresponding to bit values. Without these phase changes no data, or at least no correct data could have been received. Figure 5 shows the average power spectral density (PSD) at a bit rate of 18.75Hz.

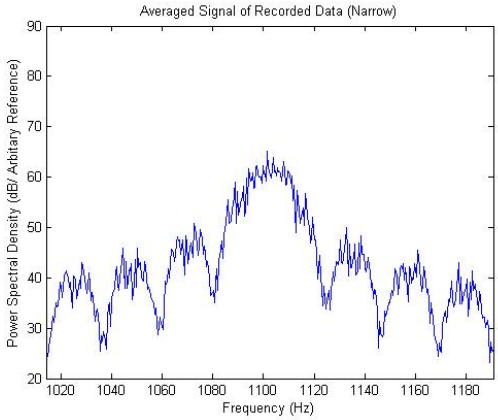

Figure 6: Averaged Data from Received Spectra at fc = 1102.5Hz, Rb = 18.75Hz

Figure 6 shows the PSD from the received signal. The received signal looks very similar to the generated signal when they are both centered about 1102.5 Hz and have nulls at every 13dB interval. The received signal however does have a lower peak power spectral density, which can be expected due to losses in the electronics and atmosphere.

In part two of the experiment, the data sequence generated in part one was sent on the carrier wave and recorded. The recorded packet was then filtered to allow for better visualization of the information being sent and to remove some of the encountered noise from the audio devices and other causes. Both the filtered and non-filtered recorded data were analyzed to find many characteristics of the waveform including the phase, magnitude, and PSD. Since BPSK deals with changes in phase to signal whether a given bit should be a one or a zero, the phase plots over time are the most important information to look at. Figures 7 and 8 show phase plots of recorded data on two separate computers.

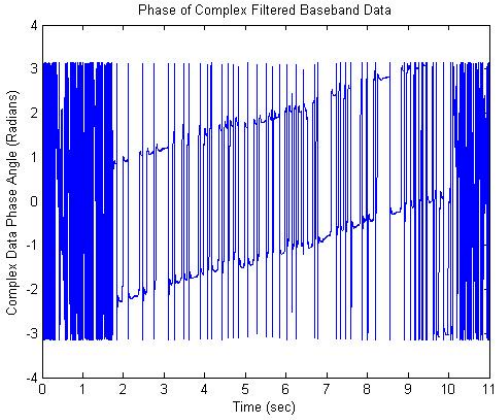

Figure 7: Phase of Filtered Baseband Data (Out of Sync)

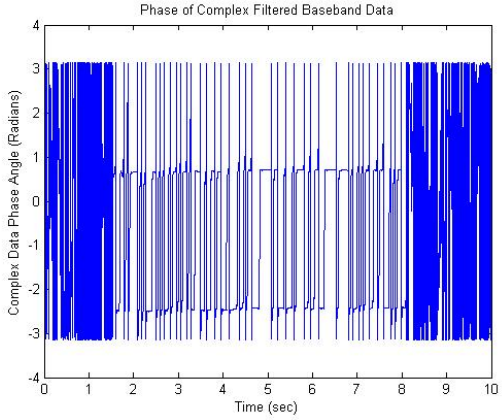

Figure 8: Phase of Filtered Baseband Data (Synced Clock)

In Figure 7, a constant upward slop in the phase change can be observed. This slope is cause by the clocks in the transmitting and recording devices being out of sync with one another. In Figure 8, the clock on the transmitter is in sync with the clock on the receiver causing no gradual change in phase over time. Since the data being sent was encoded using Differential BPSK, the absolute phase of the wave is not important as the *change* in phase from one bit to the next actually contains the data. This property of differential BPSK can be found helpful when clocks are out of sync.

Finally in part three of the experiment the recorded data was decoded and the sent data was extracted and displayed. This was achieved by first correlating the recorded data with the known preamble from part one so that the location of the preamble and therefore the start of the packet could be determined. Figure 9 shows the received data when correlated with the preamble. The large spike in magnitude at around 1.8 seconds corresponds to the time (location) when the preamble began. In Figure 10 two modulated carrier waves were transmitted simultaneously and the data was recorded. In this case, the spike in the magnitude is much harder to distinguish as compared to the previous figure. This means that determining where the preamble begins becomes difficult and the data received may skew because of an uncertain starting point.

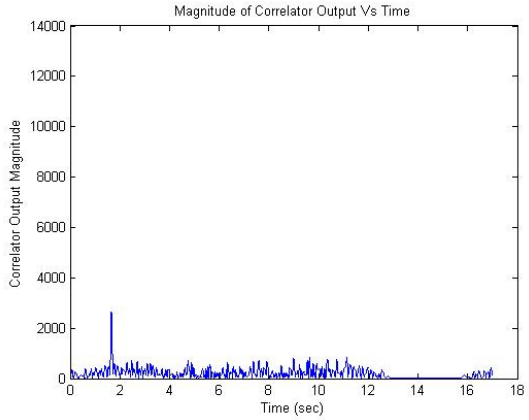

Figure 9: Correlated Data

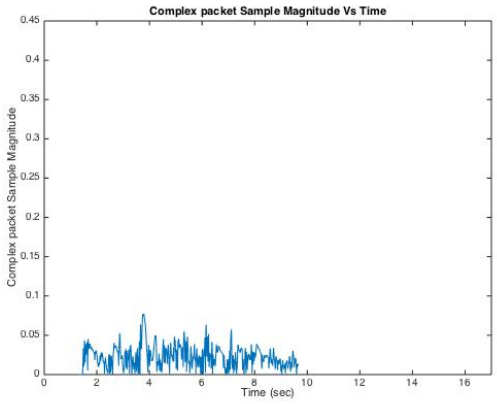

Figure 10: Correlated Data with Two Transmitted Sequences

The next step was to extract the samples from the filtered data and reduce the total number of samples to be analyzed. Reducing the total number of samples is important because with a sample rate of 44100 Hz running for approximately 10 seconds, a large amount of data was recorded. More relevantly, in order to decode the data, only one sample per bit is required. Once the recoded data was trimmed to contain only information from the packet and the total number of samples in the packet was reduced, the known preamble was separated from the packet leaving just the data samples. These samples were plotted in constellation, phase, and magnitude plots to help visualize the data and were then converted into bits containing the sent message in binary.

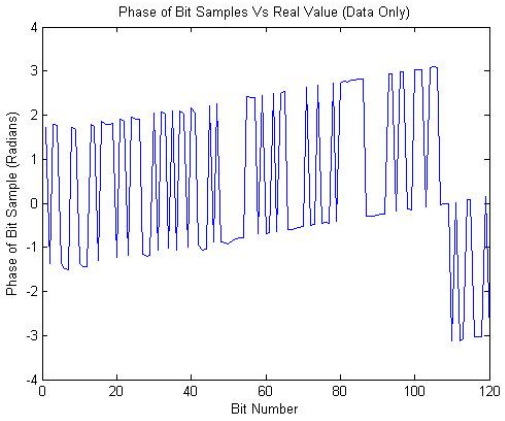

Figure 11: Phase Plot (Single Machine)

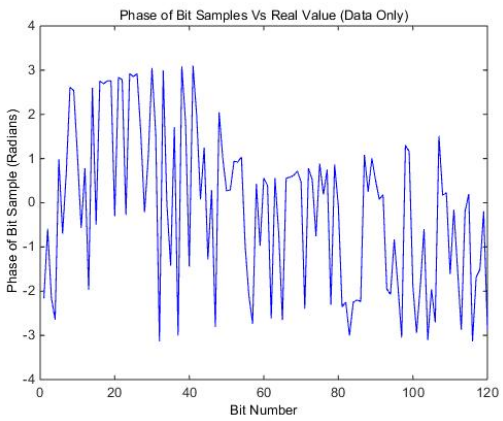

Figure 12: Phase Plot (Interference)

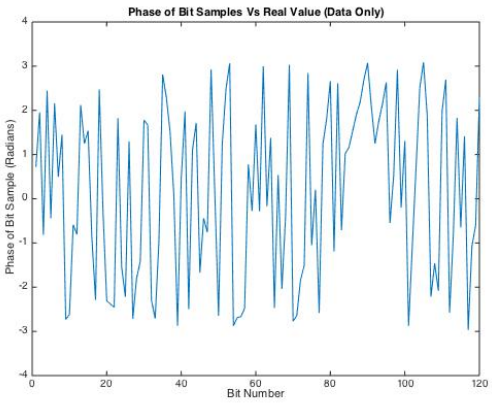

Figure 13: Phase Plot (Doppler)

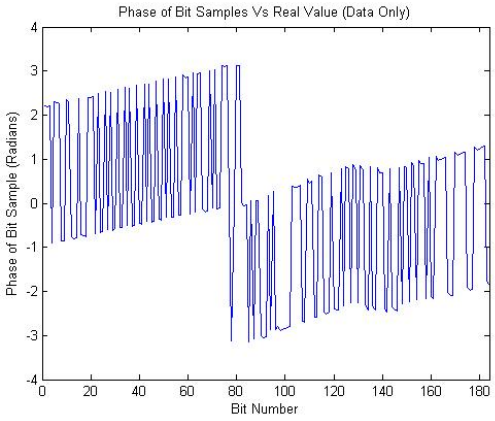

Figure 14: Phase Plot (New Data)

The plots above show the phase plots for sample data in a variety of conditions. In Figure 11, a single machine transmits and receives the data. The clocks on the transmitter and receiver are not synchronized but the phase changes are clear and easy to detect. In Figure 11 the same trend can be seen even though the transmitted data is changed from "Kilroy was here" to "Leroooooyyyyy Jenkins!!!" In Figure 12 two computers transmitted data simultaneously. The phase in this case is not clear and as expected, the interference between the two signals caused an incorrect output as show in Figure 21. Figure 13 also shows unclear phase changes, but this time caused by Doppler shift as the receiving machine moved first towards and then away from the transmitting machine.

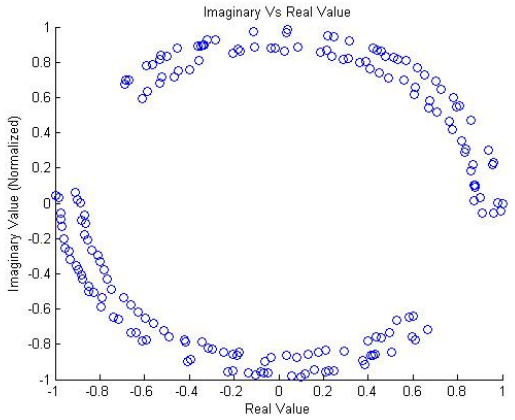

Figure 15: Constellation Plot (Single Machine)

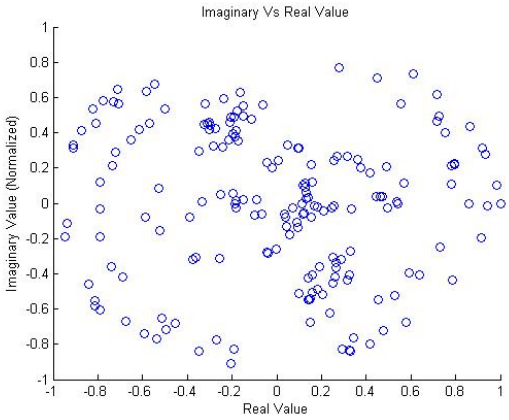

Figure 16: Constellation Plot (Interference)

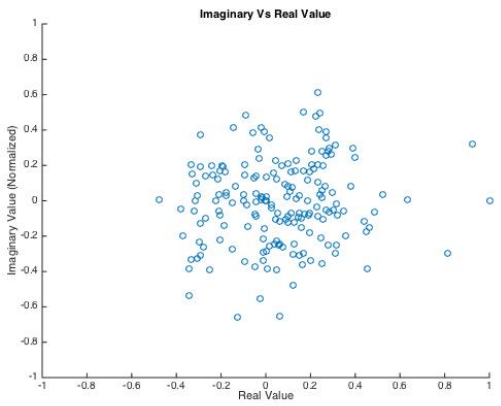

Figure 17: Constellation Plot (Doppler)

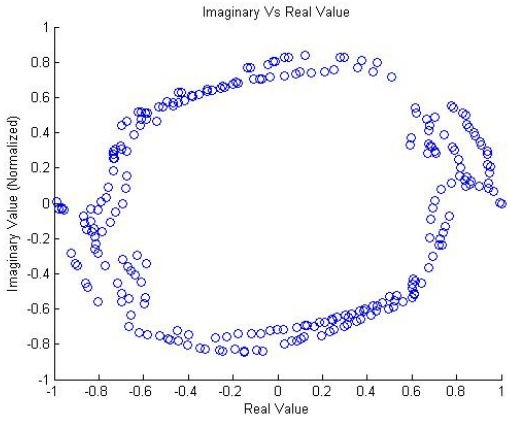

Figure 18: Constellation Plot (New Data)

Constellation Plots are another way to visually represent the phase of individual samples. Figures 15 through 18 present similar information to Figures 11 through 14 for the same conditions but perhaps show the information more clearly. It can again be observed that for the two working cases (Figures 15 and 18) there is a clear distinction in the phase of each sample. In Figure 18, since the clocks are not synchronized and the packet being sent is longer, the phase differences seem to merge together. This however is not a problem because BPSK relies on the change in phase and the change in phase from one sample to the next is clear. Like before, for the Interference and Doppler tests, shown in Figures 16 and 17 respectively, there is no clear distinction in phase change from sample to sample so the final results did not represent the data that was transmitted.

Finally, Figures 19 through 21 show the final output result for each of the test cases discussed above with the exception of the Doppler test.

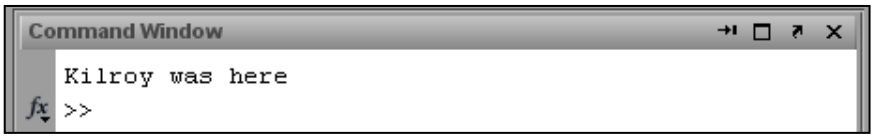

Figure 19: Output Data (Single Machine)

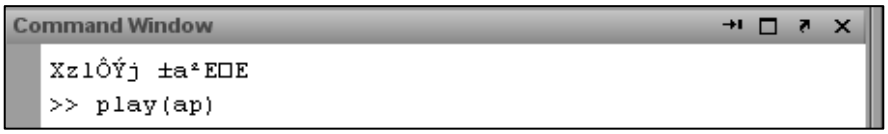

Figure 20: Output Data (Interference)

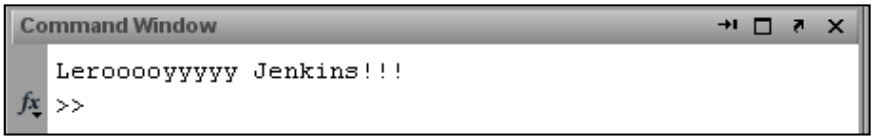

Figure 21: Output Data (New Data)

All the results are as expected, or at least as far as whether the results were correct or incorrect since it is impossible to predict what a wrong answer will actually look like. The limitations in the BPSK method that were discovered during this experiment will be discussed later in the Conclusion section of this report.

Conclusion

As the bit rate is increased, the sound being output is a heard to be a lot shriller and the change in amplitudes are perceived to be a lot faster by the human ear. At lower bit rates the sound is a lot coarser and is perceived to be played a lot slower.

During the experiment the data was found to be legible up to a bit rate of 55.125 Hz using a carrier frequency of 1102.5 Hz and a sampling frequency of 44100 Hz. Any further increase in the bit rate resulted in illegible data.

The recording was also tested at different volume levels, however this did not have significant effect on the data being output. Apart from the reduction in amplitudes for the recorded data, there was no significant alteration observed in the plots, the data once decoded was coherent with the expected results. However, one of the system limitations was found when the volume setting was too low. At a volume below 2 on the computer, the interference of background noise was found to be significant. The amplitude of the noise was approximately at the same value as that of the recorded sound and the data could not be recovered efficiently.

Another limitation on the system was encountered when a second computer was used. The demodulated data was not legible when the packet was transmitted from one computer and recorded on a different one. This can be attributed to the difference in clock speeds of the computer, errors in playback and recording timing during performance as well as ambient noise. The simulation of Doppler Effect was also found to yield illegible data.

During the course of the experiment, initially an error was made while converting the data to its equivalent binary bits due to a miss interpretation of the “char” and “double” variable types in MATLAB, this led to the data not being modulated onto the carrier at all, this was detected when a phase change was not observed once the data was modulated onto the carrier wave. Furthermore the demodulated wave phase change was not initially apparent, however upon close examination the phase change was found. The pattern of phase change however was different from that seen when the wave was modulated, the pattern seemed to change from a continuous sinusoidal variation to a wedge shaped, sharp change in phase.

The filtering process as described in the tutorial was not found to be adequate to filter out the data in undesired frequencies during this experiment. Thus a factor was added onto the formula which helped in better filtering of data and getting an accurate result.

At the final phase when the demodulated data was giving erroneous results it was discovered that the data was not modulated differentially, but was directly modulated. Once the code was changed to reflect the data being differentially modulated, the desired output was achieved.

To transmit the data by means of ASK modulation, the carrier wave would have had another variable to specify its amplitude. Data_Sequence = A sin ($2 \pi f_c t$), where A is the amplitude. Instead of changing the phase at the end of each bit, the amplitude term (A) would have been modified to be increased/decreased

based on the data bit and thus the wave would have been modified. However the data being demodulated would have had significant noise interference, especially during the bits with reduction in amplitude.

To modulate the data using the On/Off Keying modulation technique, the carrier wave would be multiplied by ‘1’ for the samples where the binary bit is expected to be ‘1’, keeping the amplitude constantly varying in a sinusoidal pattern in coherence with the carrier wave. However, for the samples where the bit is expected to be ‘0’, the amplitude of the sinusoidal wave would be multiplied by ‘0’. Thus the entire wave would stop to exist and the period can be described as a wave with amplitude ‘0’. It is expected that the sound being output from the speakers would cease to exist. It is also predicted that a bit rate of higher than 55.125 Hz could be achieved. The effect of noise would also be large, since the amplitude of the wave would be ‘0’ for samples where the data bit is ‘0’. This can be negated by either using a high amplitude value or reducing the difference threshold as described in step 13.

References

[1] W. Barott, "Tutorial Assignment (HW2) Using MATLAB to Understand Modulation of Radio Signals," Embry-Riddle Aeronautical University, Daytona Beach, Fall 2015.

[2] Y. Maheshwar, "Analog and Digital Modulation Techniques," Michigan, 2014.

[3] M. Wickert, "FIR Filters, Chapter 5," University of Colorado Colorado Springs, Colorado, 2010.

Appendix A: MATLAB Source Code

```
clc
clear

%% Step 1: Generate Data in Binary
data = 'Kilroy was here';
text = char(data);                    %Converts character to ASCII
y = zeros(length(data)*8,1);          %Converts  data from decimal to binary
for n = 1:1:length(data)
    a=abs(data(n));
    f = 8*(n-1)+1;
    y(f:f+7,1)=(de2bi(a,8));
end
data_binary = y';

%% Step 2: Generate Preamble
preamble = randi(10, 16, 1); % Creating the preamble
preamble_binary = de2bi(preamble); %Converts  preamble from decimal to binary
preamble_binary = reshape(preamble_binary, 1,
size(preamble_binary,1)*size(preamble_binary,2));

%% Step 3: Define Variables
fs = 44100; % Sample rate
fc = fs/40; % Carrier frequency
fb = fc/50; %Bit rate

Nsb = fs/fb; % Number of samples per bit
T = 10;

%% Step 4: Apply Direct BPSK to Preamble
t = (1/fs):(1/fs):T; % time for the carrier wave

Training_Sequence = sin(2*pi*fc*t); %carrier wave

for counter = 1 : size(preamble_binary,2)
    if preamble_binary(counter) == 1 %Modulation of the carrier wave with
preamble (Differential Modulation)
        Training_Sequence(counter*Nsb) = Training_Sequence(counter*Nsb)*1 ;
    elseif preamble_binary(counter) == 0
        Training_Sequence(counter*Nsb) = Training_Sequence(counter*Nsb)* -1;
        for count = 1:Nsb
            Training_Sequence(counter*Nsb + count) =
Training_Sequence(counter*Nsb + count)* -1;
        end
    end
end

Training_Sequence = Training_Sequence(1:size(preamble_binary,2)*Nsb);

%% Step 5: Apply Differential BSPK to Payload

Data_Sequence = sin(2*pi*fc*t); %Carrier wave
```

```
for counter = 1 : size(data_binary,2)
    if data_binary(counter) == 1 %Modulation of the carrier wave with data
(BPSK)
        Data_Sequence(counter*Nsb) = Data_Sequence(counter*Nsb) * -1;
        for count = Nsb*counter:size(Data_Sequence,2)
            Data_Sequence(count) = Data_Sequence(count)* -1;
        end
    elseif data_binary(counter) == 0
        for count = Nsb*counter:size(Data_Sequence,2)
            Data_Sequence(count) = Data_Sequence(count)* 1;
        end
    end
end

Data_Sequence = Data_Sequence(1:size(data_binary,2)*Nsb);

tm = t(1 : (size(preamble_binary,2) + size(data_binary,2))*Nsb); %Modulated
time function

mod_wave = [Training_Sequence, Data_Sequence]; % Modulated wave carrying
preamble and data

% figure(1)
% plot(tm, mod_wave)
% axis([2.035 2.044 -1.05 1.05])
% xlabel('Time (Seconds)')
% ylabel('Amplitude')
% title('Amplitude Vs Time showing a change of phase')

pkFFT = fft(mod_wave);
psd = 10*log10(pkFFT .* conj(pkFFT));
freqList = linspace(-fs/2,fs/2, length(pkFFT));

% figure(2)
% plot(freqList,fftshift(psd))
% axis([fc-fb*4 fc+fb*4 (max(psd)-60) max(psd)+5])
% grid on
% xlabel('Frequency (Hz)')
% ylabel('Power Spectral Density (dB/ Arbitary Reference)')
% title('Unaveraged Spectra of Generated Data')

nAv = 5;
pkFFT = fft(reshape(mod_wave, length(mod_wave)/nAv, nAv));
matpsd = (pkFFT .* conj(pkFFT));
psd = 10*log10(sum(matpsd,2));
freqList = linspace(-fs/2, fs/2, size(pkFFT,1));

% figure(3)
% plot(freqList,fftshift(sum(psd,2)))
% grid on
% axis([fc-fb*4 fc+fb*4 (max(psd)-60) max(psd)+5])
% xlabel('Frequency (Hz)')
% ylabel('Power Spectral Density (dB/ Arbitary Reference)')
% title('Averaged Spectra of Generated Data')
```

```
%% Step 6: Create Objects for Playing and Recording

ap = audioplayer(mod_wave, fs);
ar = audiorecorder(fs, 16, 1);

%% Step 7: Play and Record Modulated data

record(ar, T + 1)
pause(1)
play(ap)
pause(T + 1)
stop(ar)
% play(ar)

recorded_data = getaudiodata(ar,'double');
recorded_data = recorded_data';

tr = (1/fs):(1/fs):T + 1;

% figure(4)
% plot(tr,recorded_data)
% axis([0 T+2 -1.05 1.05]);
% xlabel('Time (Seconds)')
% ylabel('Amplitude')
% title('Recorded Sound')

% figure(5)
% plot(tr,recorded_data)
% axis([4.7 4.91 -1.05 1.05]);
% xlabel('Time (Seconds)')
% ylabel('Amplitude')
% title('Recorded Sound')

pkFFT = fft(recorded_data);
psd = 10*log10(pkFFT .* conj(pkFFT));
freqList = linspace(-fs/2,fs/2, length(pkFFT));

% figure(6)
% plot(freqList,fftshift(psd))
% axis([0 7000 30 90])
% grid on
% xlabel('Frequency (Hz)')
% ylabel('Power Spectral Density (dB/ Arbitary Reference)')
% title('Spectra of Recorded Data (Wide)')

nAv = 5;
pkFFT = fft(reshape(recorded_data, length(recorded_data)/nAv, nAv));
matpsd = (pkFFT .* conj(pkFFT));
psd = 10*log10(sum(matpsd,2));
freqList = linspace(-fs/2, fs/2, size(pkFFT,1));

% figure(7)
% plot(freqList,fftshift(sum(psd,2)))
% grid on
% axis([fc-fb*4 fc+fb*4 20 90])
```

```
% xlabel('Frequency (Hz)')
% ylabel('Power Spectral Density (dB/ Arbitary Reference)')
% title('Averaged Signal of Recorded Data (Narrow)')

%% Step 8: Mix Data Down to Baseband

pkFFT = fft(recorded_data);
psd = 10*log10(pkFFT .* conj(pkFFT));
freqList = linspace(-fs/2,fs/2, length(pkFFT));

% figure(8)
% plot(freqList,fftshift(psd))
% axis([-4000 4000 30 90])
% xlabel('Frequency (Hz)')
% ylabel('Power Spectral Density (dB/ Arbitary Reference)')
% title('Averaged Signal of Recorded Data (Double Sideband)')

ts = (1/fs):(1/fs): T + 1;

lo_inphase = cos(2*pi*fc*ts);
lo_quadrature = sin(2*pi*fc*ts);

baseband_data = lo_inphase .* recorded_data + 1i*lo_quadrature .*
recorded_data;

Training_Sequence_Zeros = [Training_Sequence , zeros(1, size(recorded_data,2)
- size(Training_Sequence,2))];
baseband_preamble = lo_inphase .* Training_Sequence_Zeros + 1i*lo_quadrature
.* Training_Sequence_Zeros;

pkFFT = fft(baseband_data);
psd = 10*log10(pkFFT.* conj(pkFFT));
freqList = linspace(-fs/2,fs/2, length(pkFFT));

% figure(9)
% plot(freqList,fftshift(psd))
% axis([-4000 4000 30 90])
% xlabel('Frequency (Hz)')
% ylabel('Power Spectral Density (dB/ Arbitary Reference)')
% title('Spectra After Mixing with Complex fc')

%% Step 9: Apply Filtering

Coeff = ones(1, round(0.5 * fs/fb))/round(0.5 * fs/fb);

filtered_baseband_data = filter(Coeff, 1, baseband_data);
filtered_baseband_preamble = filter(Coeff, 1, baseband_preamble);

pkFFT = fft(filtered_baseband_data);
psd = 10*log10(pkFFT.* conj(pkFFT));
freqList = linspace(-fs/2,fs/2, length(pkFFT));

% figure(10)
% plot(freqList,fftshift(psd))
```

```
% axis([-4000 4000 30 90])
% xlabel('Frequency (Hz)')
% ylabel('Power Spectral Density (dB/ Arbitary Reference)')
% title('Spectra After Filtering')

nAv = 5;
pkFFT = fft(reshape(filtered_baseband_data,
length(filtered_baseband_data)/nAv, nAv));
matpsd = (pkFFT .* conj(pkFFT));
psd = 10*log10(sum(matpsd,2));
freqList = linspace(-fs/2, fs/2, size(pkFFT,1));

% figure(11)
% plot(freqList,fftshift(psd))
% axis([-70 70 30 90])
% xlabel('Frequency (Hz)')
% ylabel(' Power Spectral Density (dB/ Arbitary Reference)')
% title('Spectra After Filtering')

% figure(12)
% plot(ts, abs(filtered_baseband_data))
% axis([0 10 0 .1])
% xlabel('Time (sec)')
% ylabel('Complex Data Magnitude')
% title('Magnitude of Complex Filtered Baseband Data')

% figure(13)
% plot(ts, angle(filtered_baseband_data))
% axis([0 T+2 -4 4])
% xlabel('Time (sec)')
% ylabel('Complex Data Phase Angle (Radians)')
% title('Phase of Complex Filtered Baseband Data')

%% Step 10: Correlate Recorded Data with Training Sequence

Correlatoroutput = ifft(conj(fft(filtered_baseband_preamble)) .*
fft(filtered_baseband_data));

figure(14)
plot(ts, abs(Correlatoroutput))
axis([0 T+2 0 14000])
xlabel('Time (sec)')
ylabel('Correlator Output Magnitude')
title('Magnitude of Correlator Output Vs Time')

[max_value, start_sample] = max(Correlatoroutput);

%% Step 11: Extract Samples From Recroded Data

end_sample = start_sample + (Nsb* (size(data_binary,2) +
size(preamble_binary,2))) - 1;
extracted_packet = filtered_baseband_data(1, start_sample : end_sample);

tn = ts(start_sample : end_sample);
```

```
figure(15)
plot(tn, abs(extracted_packet))
%axis([0 T+2 0 .45])
xlabel('Time (sec)')
ylabel('Complex packet Sample Magnitude')
title('Complex packet Sample Magnitude Vs Time')

figure(16)
plot(tn, angle(extracted_packet))
%axis([0 T+2 -4 4])
xlabel('Time (sec)')
ylabel('Complex Packet Data Phase Angle (Radians)')
title('Phase of Complex Packet Data Vs Time')

%% Step 12: Reduce Total Number of Samples

sample_bits = extracted_packet(1000:Nsb:end);
max_sample = max(sample_bits);
sample_bits_norm = sample_bits./max_sample;

figure(17)
scatter(real(sample_bits_norm), imag(sample_bits_norm))
axis([-1 1 -1 1])
xlabel('Real Value')
ylabel('Imaginary Value (Normalized)')
title('Imaginary Vs Real Value')

%% Step 13: Plot the Phase

angle_sample_data_bits =
angle(sample_bits(size(preamble_binary,2):1:end));%this will include the
first preamble bit so that diff() reamoves that
bit_number = 1:1:(size(data_binary,2));
difference = diff(angle_sample_data_bits);

for z = 1 : 1 : size(difference,2)
    if (difference(z) > pi/2 || difference(z) < -pi/2)
        demod_data(z) = 1;
    else%if (difference(z) < 0)
        demod_data(z) = 0;
    end
end
demod_data(1) = [];
demod_data(120) = 0;
figure(18)
plot(bit_number, angle_sample_data_bits(2:1:end))
axis([0 size(data_binary,2) -4 4])
xlabel('Bit Number')
ylabel('Phase of Bit Sample (Radians)')
title('Phase of Bit Samples Vs Real Value (Data Only)')

%% Step 14: Print the Received Data

out = char(bi2de(reshape(demod_data,8,15).'))
```