

Third Year

First Term

**Super-heterodyne Receiver  
Communication Project report**

|  |  |  |
| --- | --- | --- |
| Name | Sec | ID |
| Yousef Khaled Omar Mahmoud | 4 | 9220984 |

Presented by

**Instructors: Dr.** Ahmed Hesham

**Eng.** Said Kamel

# Abstract:

In this project, we aimed to simulate a multi-stage communication system, focusing on the transmission and reception of audio signals through various stages, including baseband, RF (Radio Frequency), IF (Intermediate Frequency), and AM (Amplitude Modulation). Our system processes audio signals by modulating them into AM DSB-SC (Double Sideband Suppressed Carrier) format.

We implemented a radio simulator device with MATLAB. we focused on emulating the receiver and its filtering process, using software to handle the audio files, and explored different signal processing techniques.

# Table of content:

[Abstract: 2](#_Toc183806012)

[Table of content: 3](#_Toc183806013)

[Introduction 5](#_Toc183806014)

[System Design and Implementation: 5](#_Toc183806015)

[ The modulator: 5](#_Toc183806016)

[ The receiver: 5](#_Toc183806017)

[System Implementation: 6](#_Toc183806018)

[System Components: 7](#_Toc183806019)

[1. AM Modulator: 7](#_Toc183806020)

[2. RF Stage: 8](#_Toc183806021)

[3. Mixer: 9](#_Toc183806022)

[4. IF Stage: 10](#_Toc183806023)

[5. Baseband Detection: 11](#_Toc183806024)

[Simulation Results 12](#_Toc183806025)

[1. Normal Simulation: 12](#_Toc183806026)

[2. Noise: 12](#_Toc183806027)

[3. No RF Filter: 12](#_Toc183806028)

[4. Frequency Offset (200 Hz): 12](#_Toc183806029)

[5. Frequency Offset (1200 Hz): 12](#_Toc183806030)

[Normal Simulation: 13](#_Toc183806031)

[Noise Simulation: 16](#_Toc183806032)

[NO RF Filter: 17](#_Toc183806033)

[Frequency Offset (200 Hz): 18](#_Toc183806034)

[Frequency Offset (1200 Hz): 19](#_Toc183806035)

[Conclusion: 20](#_Toc183806036)

[ Noise and Frequency Offsets: 20](#_Toc183806037)

[ The RF filter: 20](#_Toc183806038)

[References: 21](#_Toc183806039)

[Appendix: 21](#_Toc183806040)

[Main code: 23](#_Toc183806041)

[AM DSC SC MODULATION: 26](#_Toc183806042)

[Mixer: 27](#_Toc183806043)

# Introduction

The primary goal of this project is to simulate the fundamental components of an analog AM communication system using MATLAB. This involves the creation of an AM modulator to encode audio signals for transmission and a super-heterodyne receiver to recover the transmitted signal. The project uses provided audio signals as input messages, simulating a real-world scenario of radio transmission and reception.

# System Design and Implementation:

An AM communication system consists of two main stages:

* The modulator: encodes the message signal onto a higher-frequency carrier for efficient wireless transmission.
* The receiver: utilizes a super-heterodyne architecture, involving mixing, filtering, and amplification, to recover the original message.

## System Implementation:

In *fig[1]*, We describe how we implement the system in MATLAB.[1]

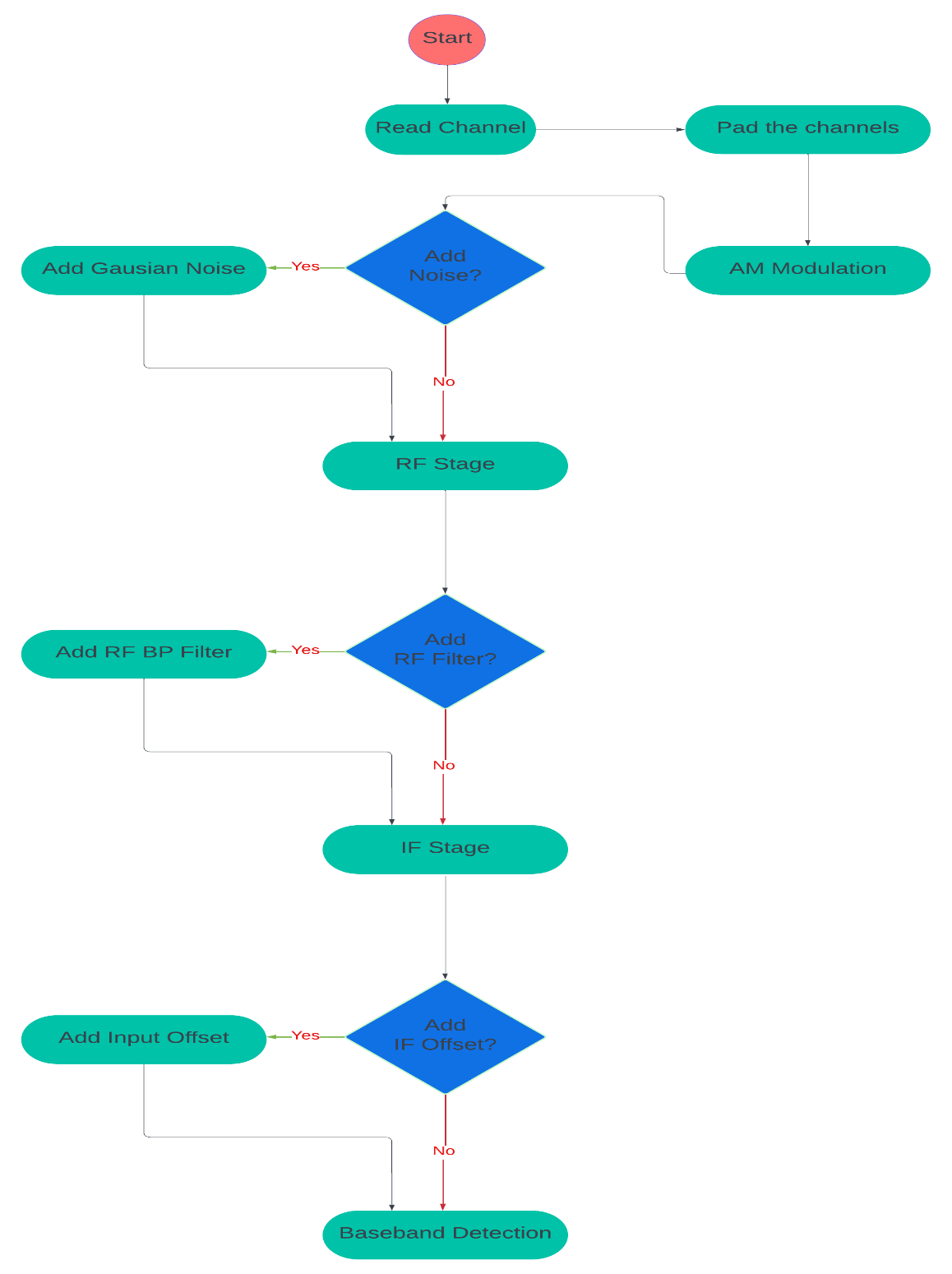


Figure Simulation Flow Chart

## System Components:

### AM Modulator:

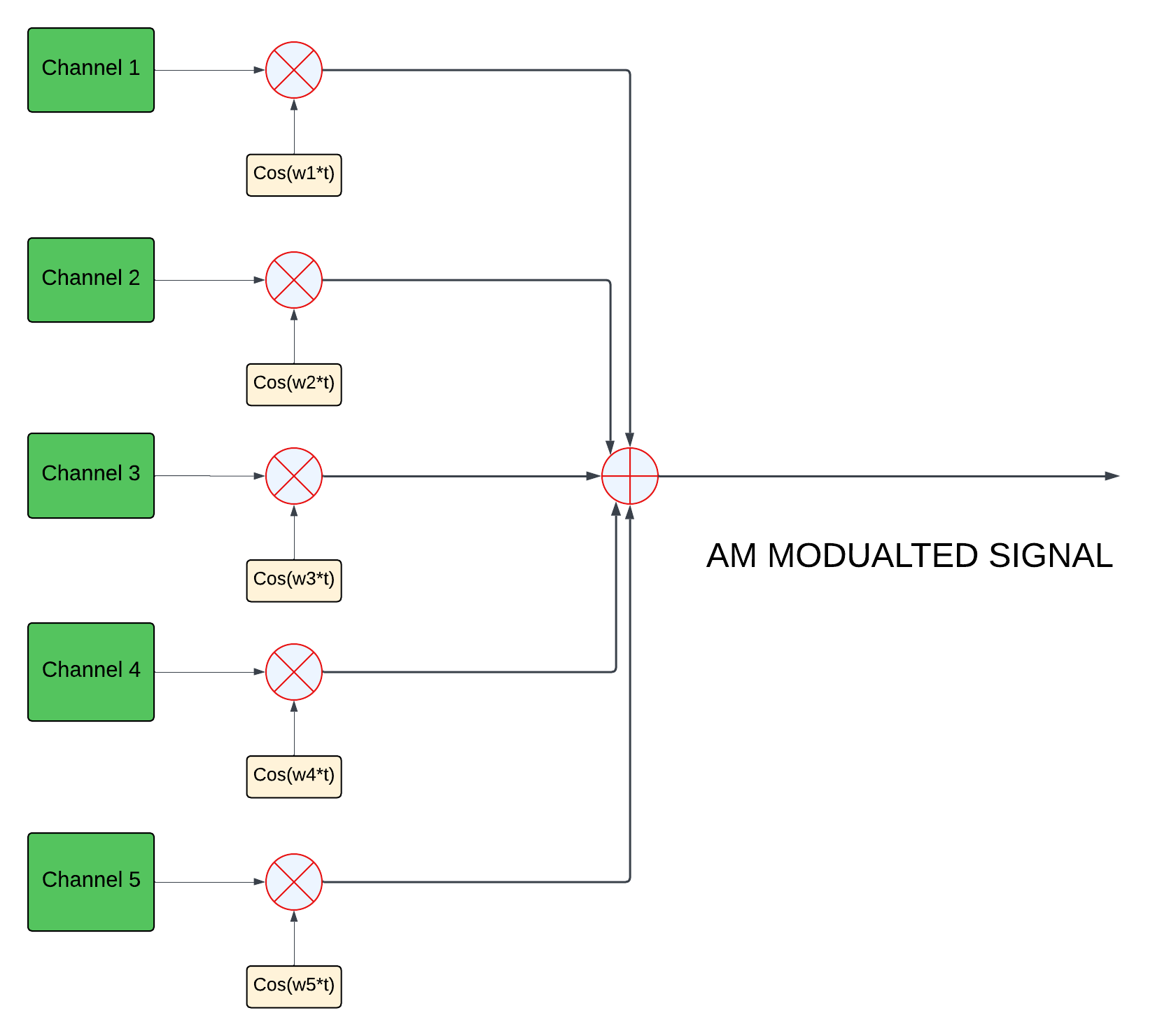


Figure AM Modulation Double Sideband Suppressed Carrier

Equation 1 Frequency Division Multiplexing

* *Where: Mn(t) is the n-th channel's message signal, Wn=100+nΔF, with ΔF=50 kHz and n as the channel index  
   (n=0 corresponds to the first signal modulated at 100 kHz)*

As shown in *fig[2]*,This project presents a MATLAB simulation of a multi-channel communication system using Frequency Division Multiplexing (FDM). The system processes five audio signals, each encoded using Amplitude Modulation (AM) onto carrier waves with unique frequencies.

The modulated signals are summed to form the FDM signal, defined mathematically in *Equ[1]*, and transmitted over RF channels and processed at the receiver.

### RF Stage:

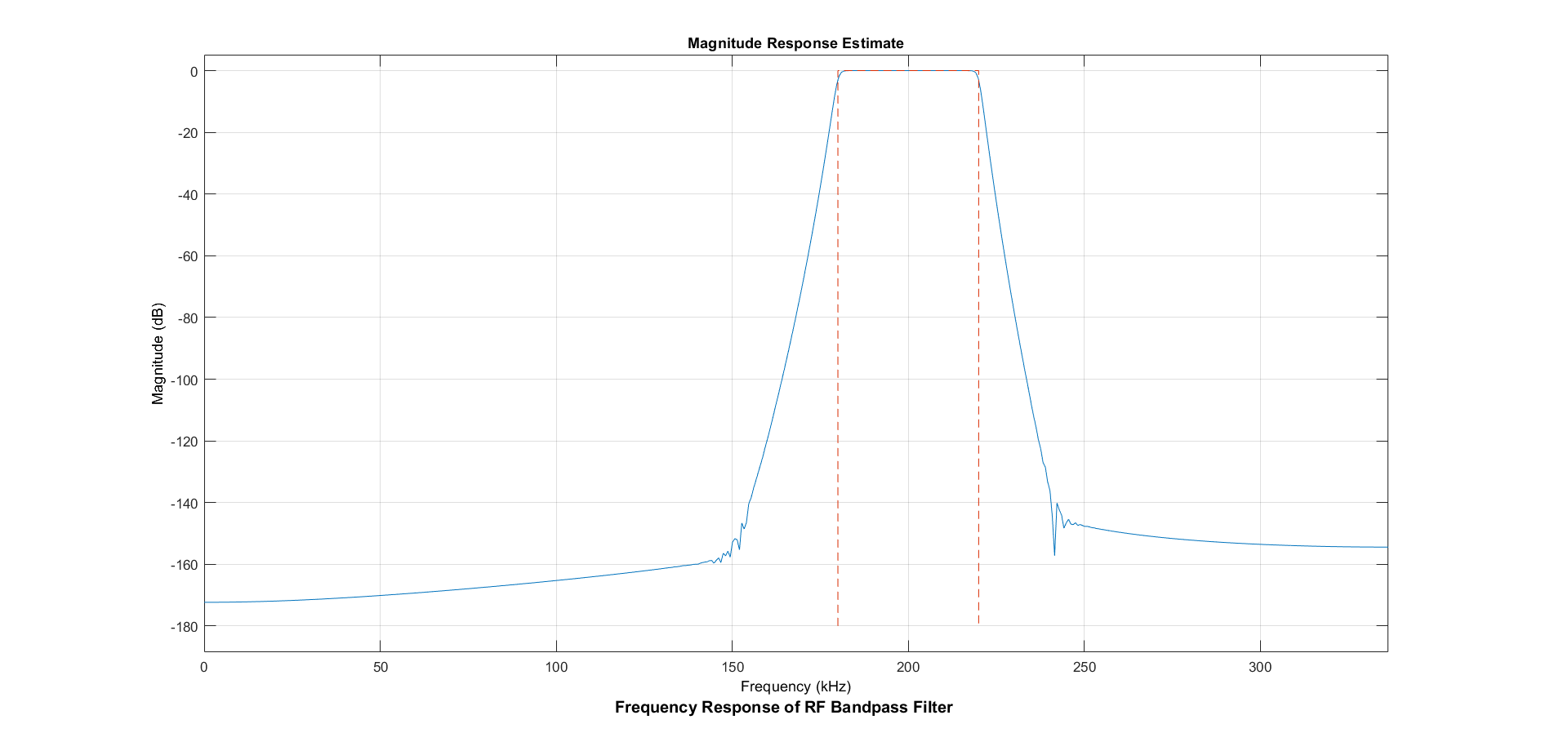


Figure Receiver Filter Frequency Response

As shown in *fig[3]*, RF filter is a band pass filter centered on the desired station’s carrier frequency ωn.

* **The purpose of the RF stage**: is to reject the intermediate frequency (IF) image and other unwanted signals outside the passband to extract the desired station.

### Mixer:

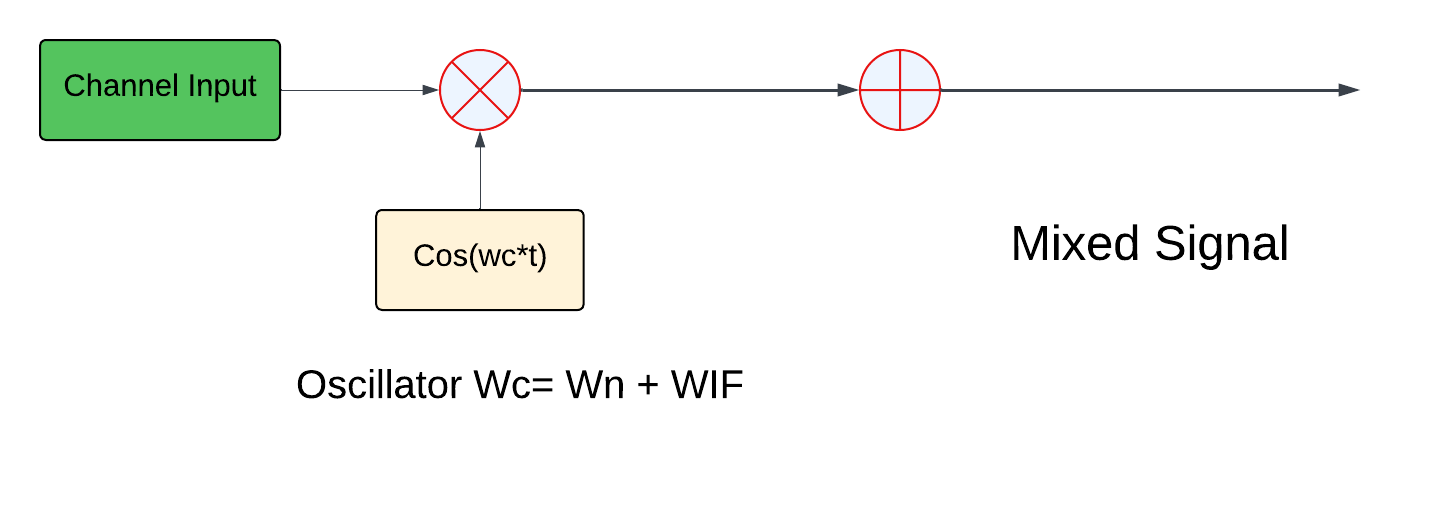


Figure Mixer with Oscillator Block Diagram

As shown in *fig[4]*, The Oscillator generates a carrier signal for mixing, shifting the desired signal to the Intermediate Frequency (IF) band.

Equation 2 Oscillator Frequency

* *Where: Wc is the oscillator frequency, Wn is the channel frequency and WIF is the intermediate frequency*

According to *Equ[2]*,The oscillator frequency is set to get the desired channel to IF stage.

### IF Stage:

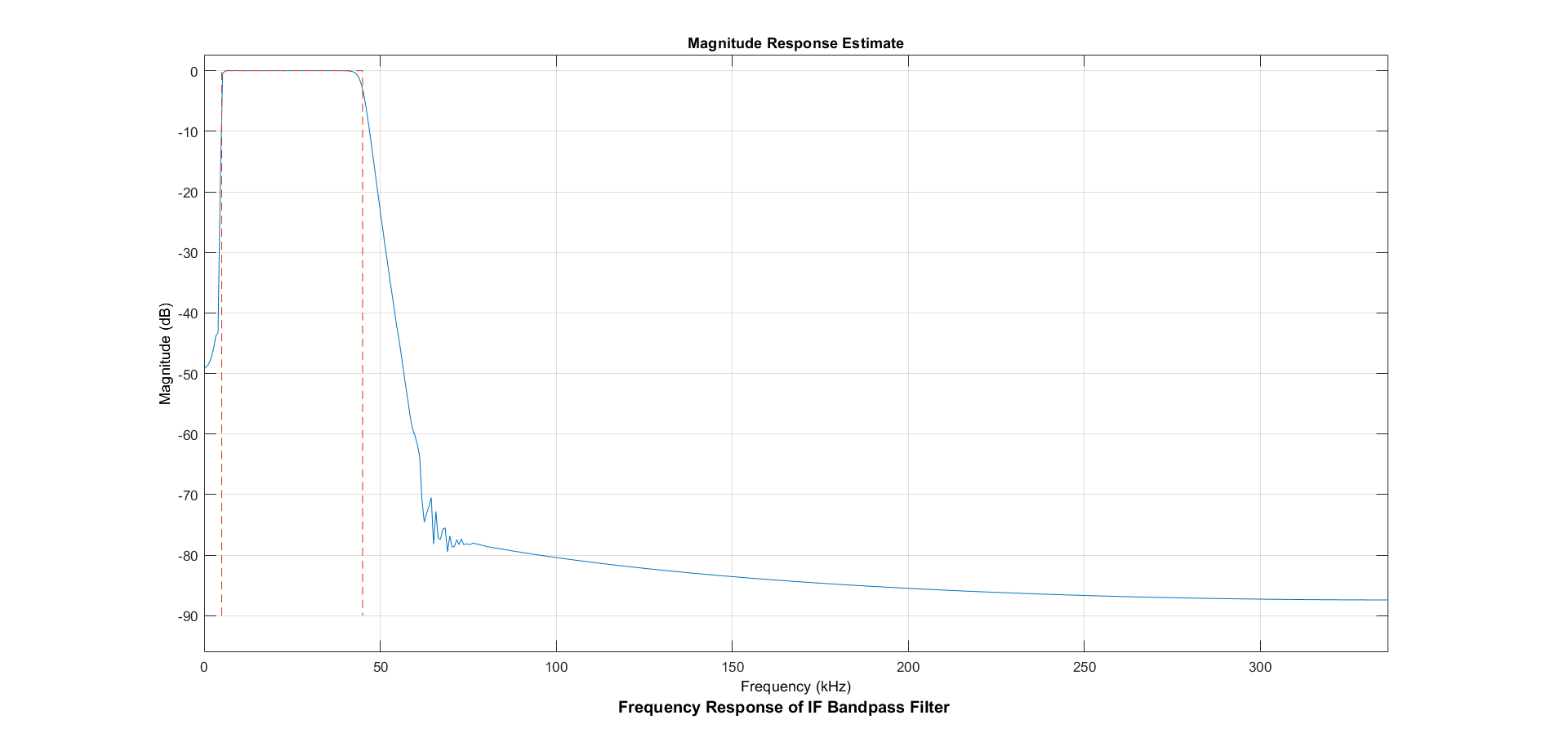


Figure Intermediate Filter Frequency Response

As shown in *fig[5]*, IF filter is a band pass filter centered on intermediate frequency (WIF).

* **The purpose of the IF stage**: filters the IF signal for further processing. By shifting the signal to a fixed WIF, it simplifies the filter design and improves selectivity.

### Baseband Detection:

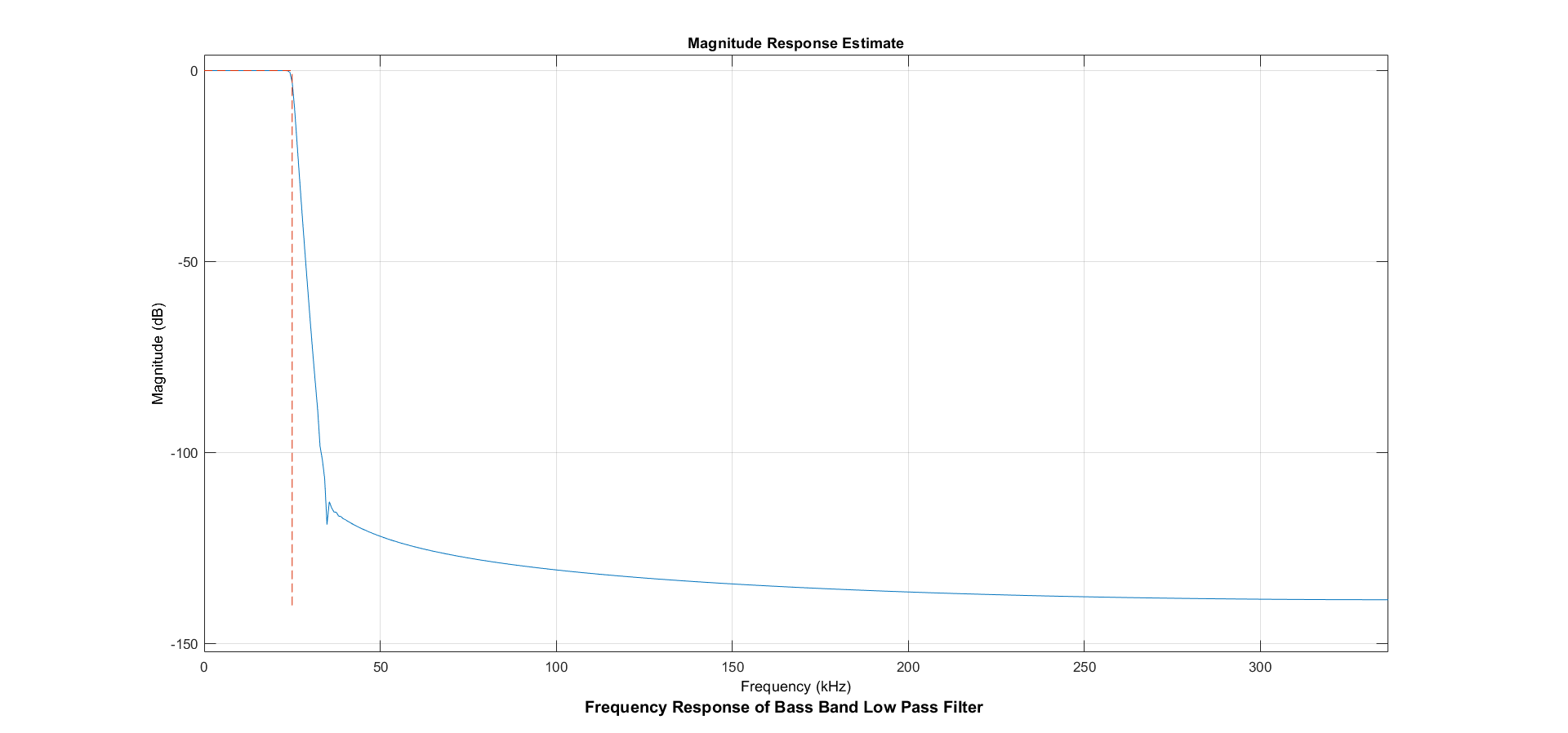


Figure Baseband Filter Frequency Response

As shown in *fig[6]*, The baseband filter a Low-Pass Filter (LPF) that removes high-frequency components after mixing the IF signal with the carrier, centered at a frequency little higher than the channel band width which is 10 kHz.

* **The purpose of the Baseband Detector:** recovering the original message by demodulating the IF signal.

The IF stage is crucial because it allows the receiver to use fixed components (filters and amplifiers) irrespective of the received carrier frequency.

# Simulation Results

In this project, we implemented and analyzed various simulations to evaluate the performance and reliability of our communication system under different conditions. The simulations included the following scenarios:

### Normal Simulation:

The system operates without any external disturbances, providing a baseline for performance analysis.

### Noise:

Additive noise was introduced to simulate real-world interference, demonstrating the system's ability to process and recover the original signal under noisy conditions.

### No RF Filter:

The impact of bypassing the RF stage was analyzed to highlight its role in filtering and isolating the desired signal.

### Frequency Offset (200 Hz):

A small offset was applied to the carrier frequency, simulating minor synchronization errors and testing the system's tolerance.

### Frequency Offset (1200 Hz):

A larger offset was introduced, representing more severe desynchronization, to analyze the limits of the system's error-handling capabilities

## Normal Simulation:

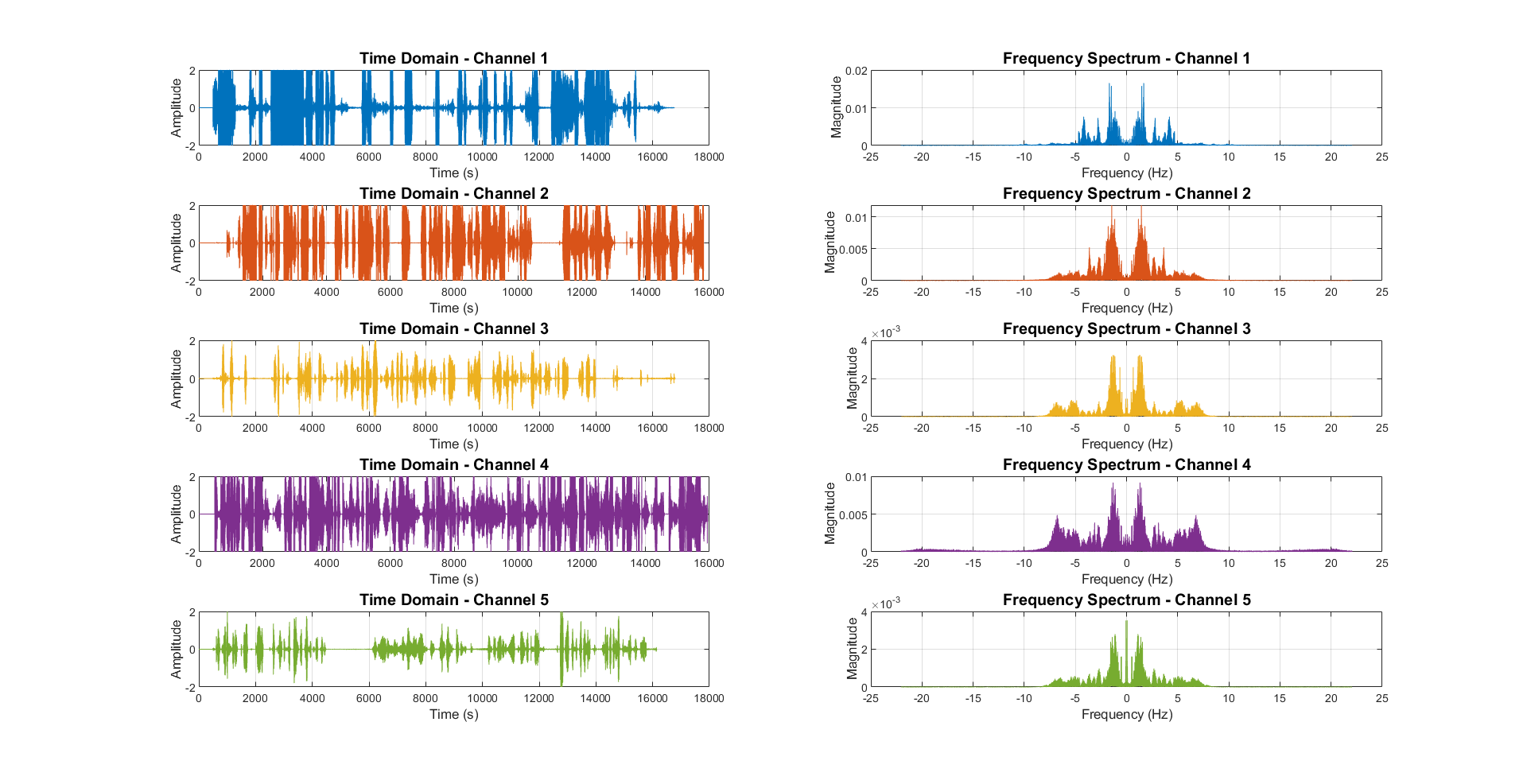
We’re going to run the simulation without any problems (Ideal Case) and going to choose channel 3.

Figure Channels plot in time and frequecny domain

As shown in *fig[7]*, All channels have a Band width approximately equal to 10 kHz.

Their length wasn’t equal so we need to pad them

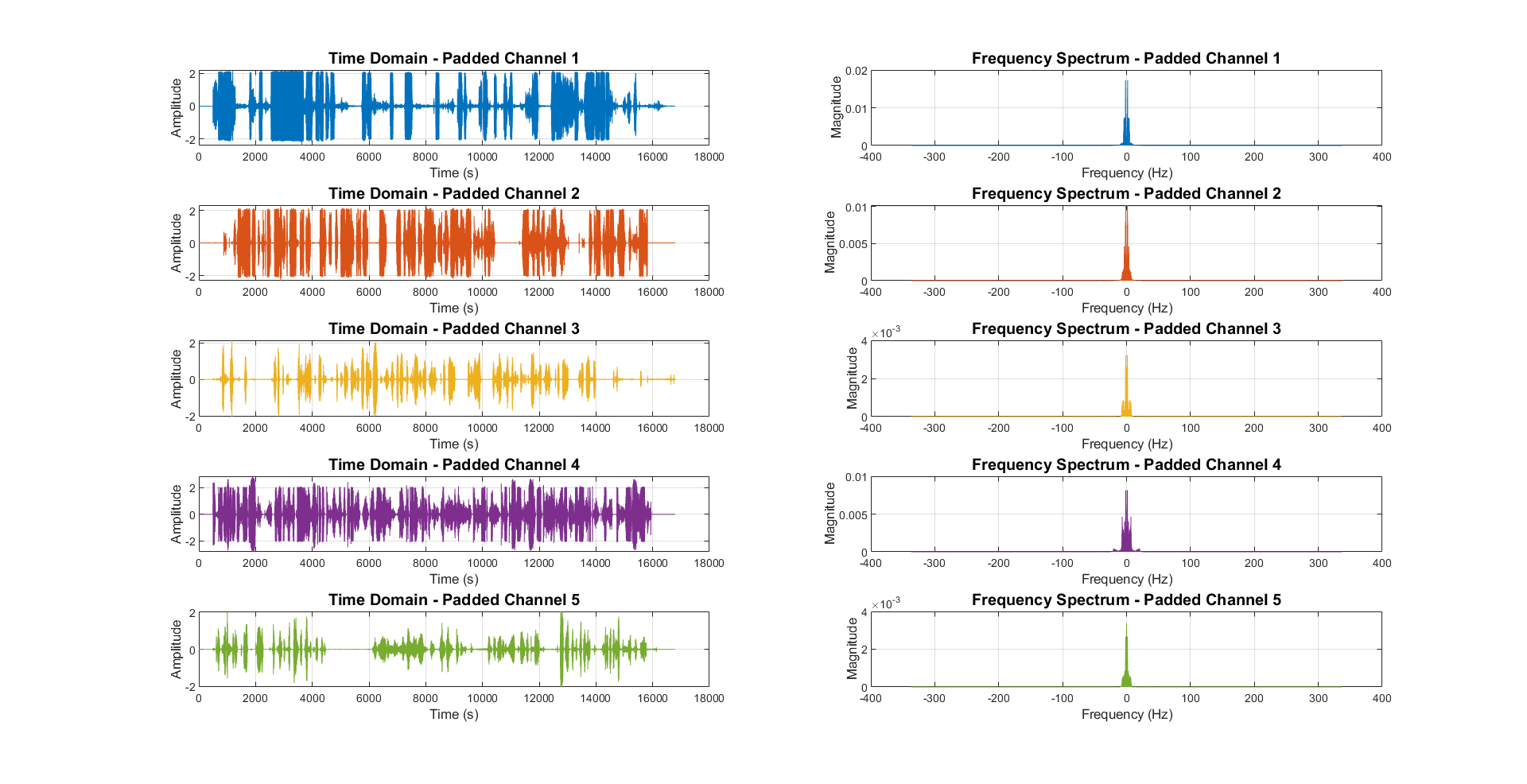


Figure Padded Channels

As shown in *fig[8]*, The channels are now equal in length, so they are padded.

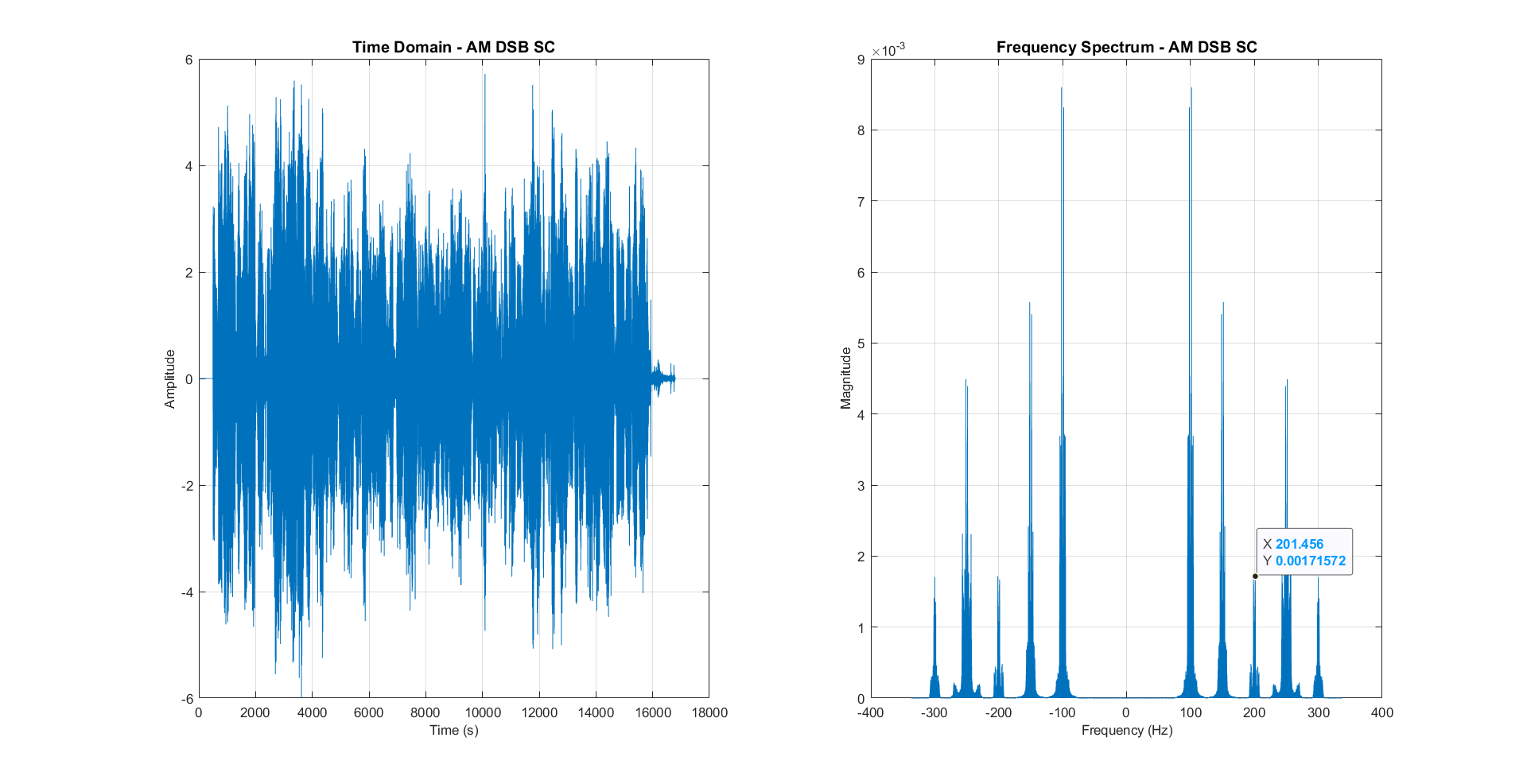


Figure frequency division multiplexing

As shown in *fig[9]*, We used AM DSB-SC modulation where the carrier and message signal are multiplied. The channel 3 that’s desired is at frequency equals to 200 kHz using *Equ[1]*.

We sum each channel modulated, as shown in *Equ[1]*, to transmit it into free space.

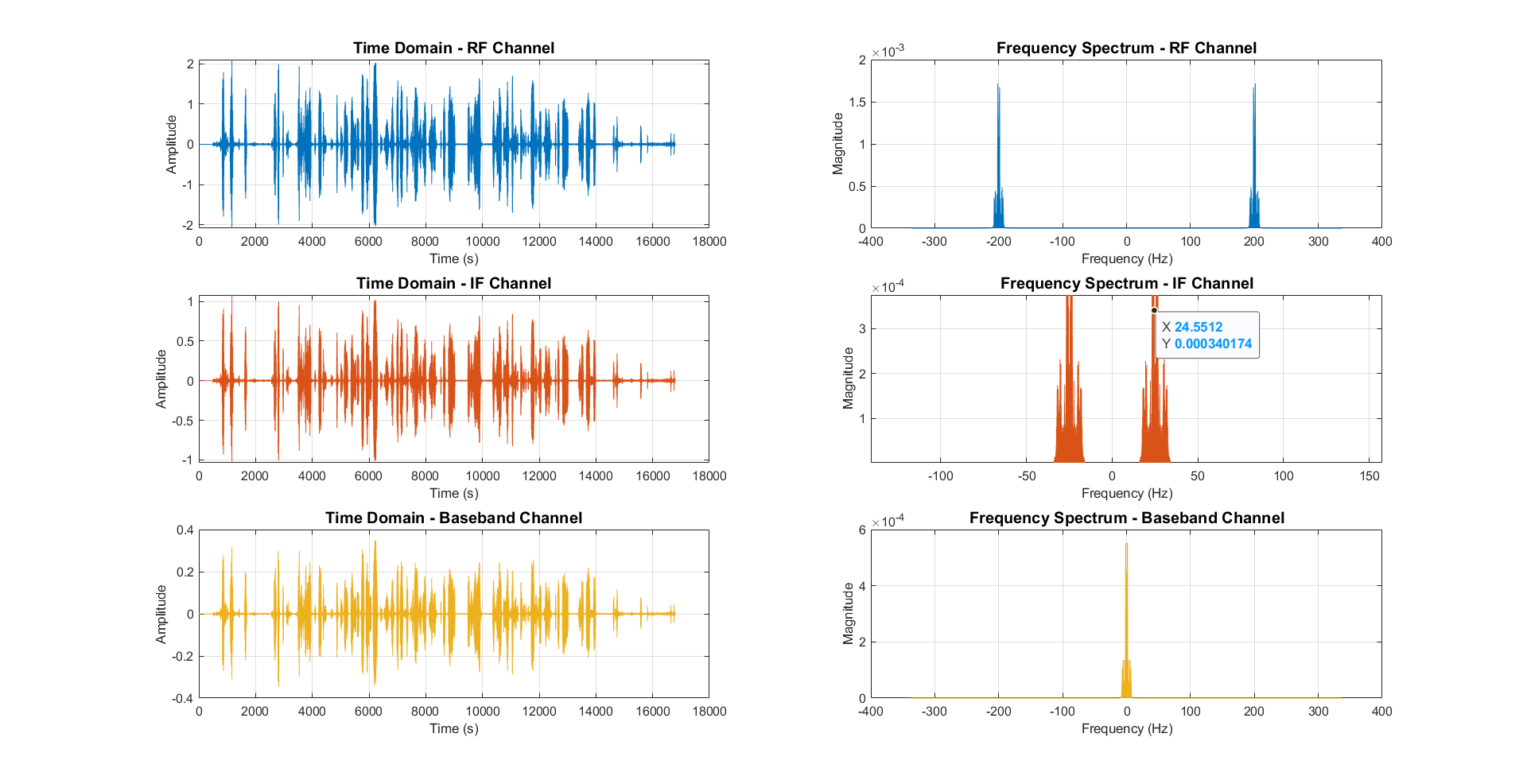


Figure Receiver Signals

As shown *fig[10]*, We first filter the signals using the RF filter in *fig[3]*.

After that we use the mixer with oscillator in *fig[4]* to get the signal to IF frequency using the *Equ[2]* then filter it using the IF filter *fig[5]*.

The baseband detection is the last stage to retreat the orignal message using the mixer with oscillator frequency at WIF.

The received message doesn’t have any problems as it’s **Ideal Case.**

We listened to it and it was identical to the orignal message.

## Noise Simulation:

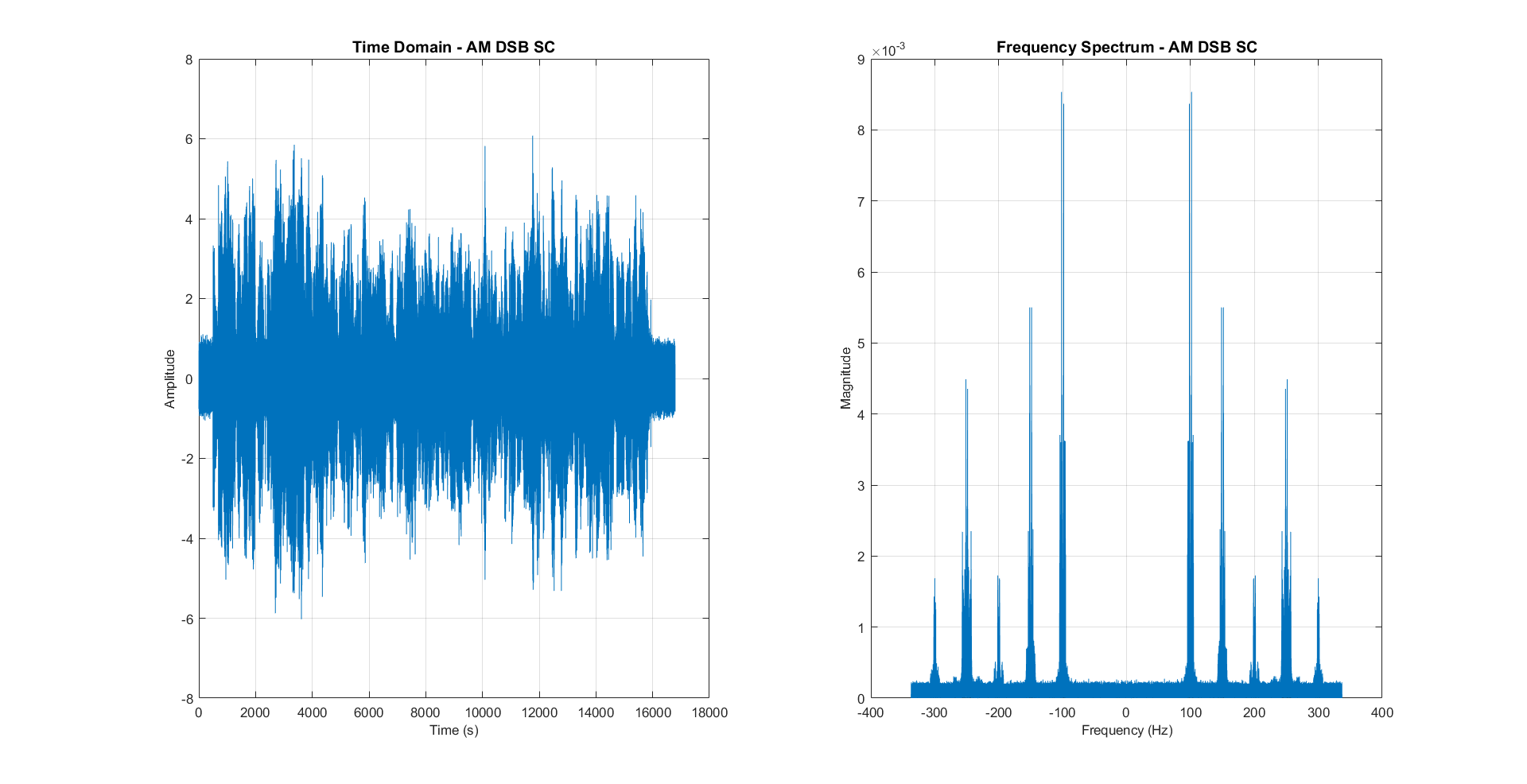


Figure frequency division multiplexing with noise

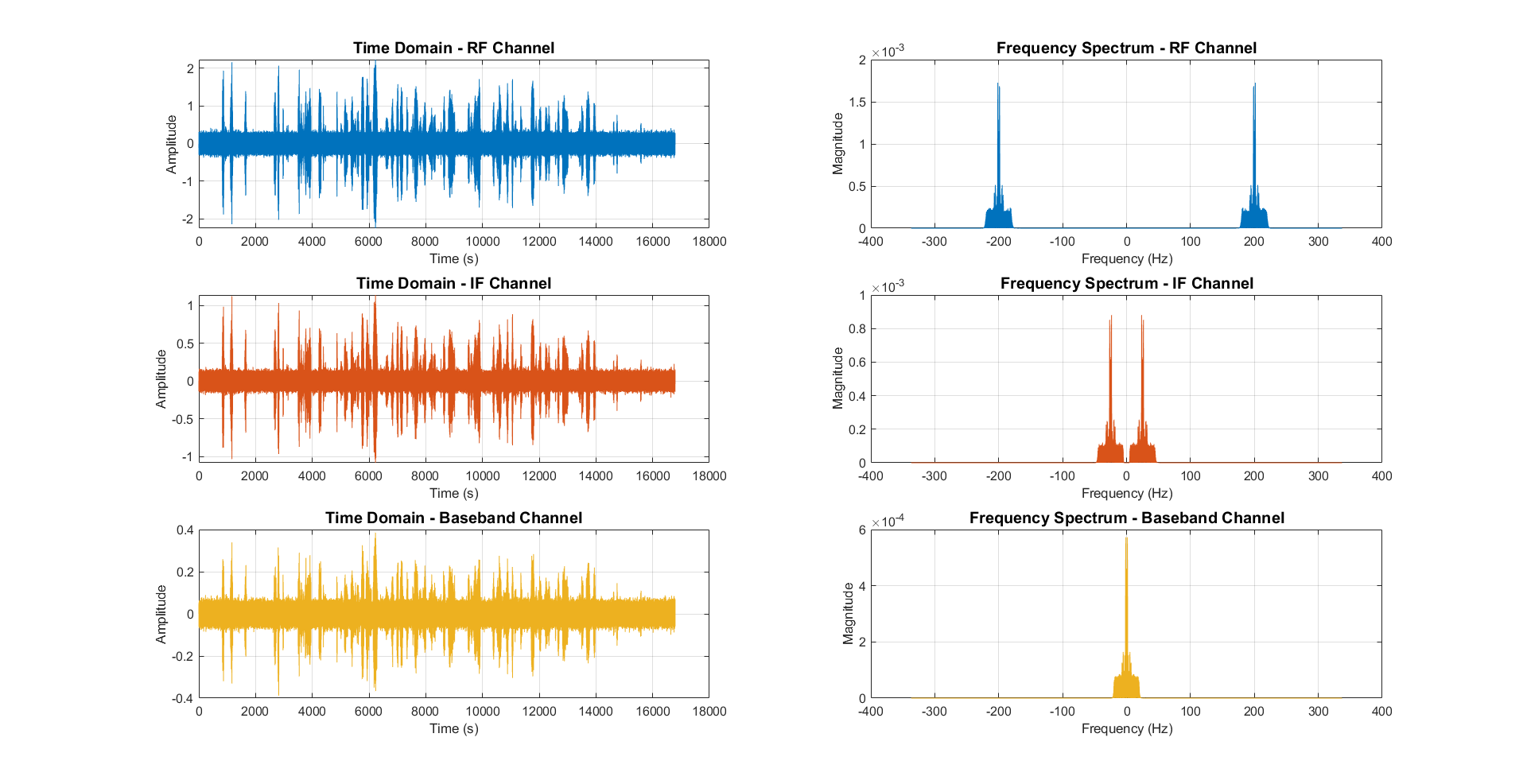
As shown in *fig[11]*, We added the gaussian noise to the AM Modulation mixer with SNR=10.

Figure Receiver Signals with Noise

As shown in *fig[12]*, The receiver did its job and got the desired channel but it didn’t filter the noise.

So listening to the desired channel we couldn’t understand the message because it was too noisy.

But if the SNR was about 80-100 dB It’s understandable.

## NO RF Filter:

Figure Receiver Signals without RF Filter

As shown in *fig[13]*, The receiver no longer isolates the desired signal from the spectrum. Instead, multiple signals, including unwanted neighboring channels, and IF image of the desired channel in the frequency domain.

The result is a mix of multiple modulated signals entering the demodulator, making it impossible to correctly reconstruct the original message.

So when listening to the bassband message we’re hearing two channels at the same time.

## Frequency Offset (200 Hz):

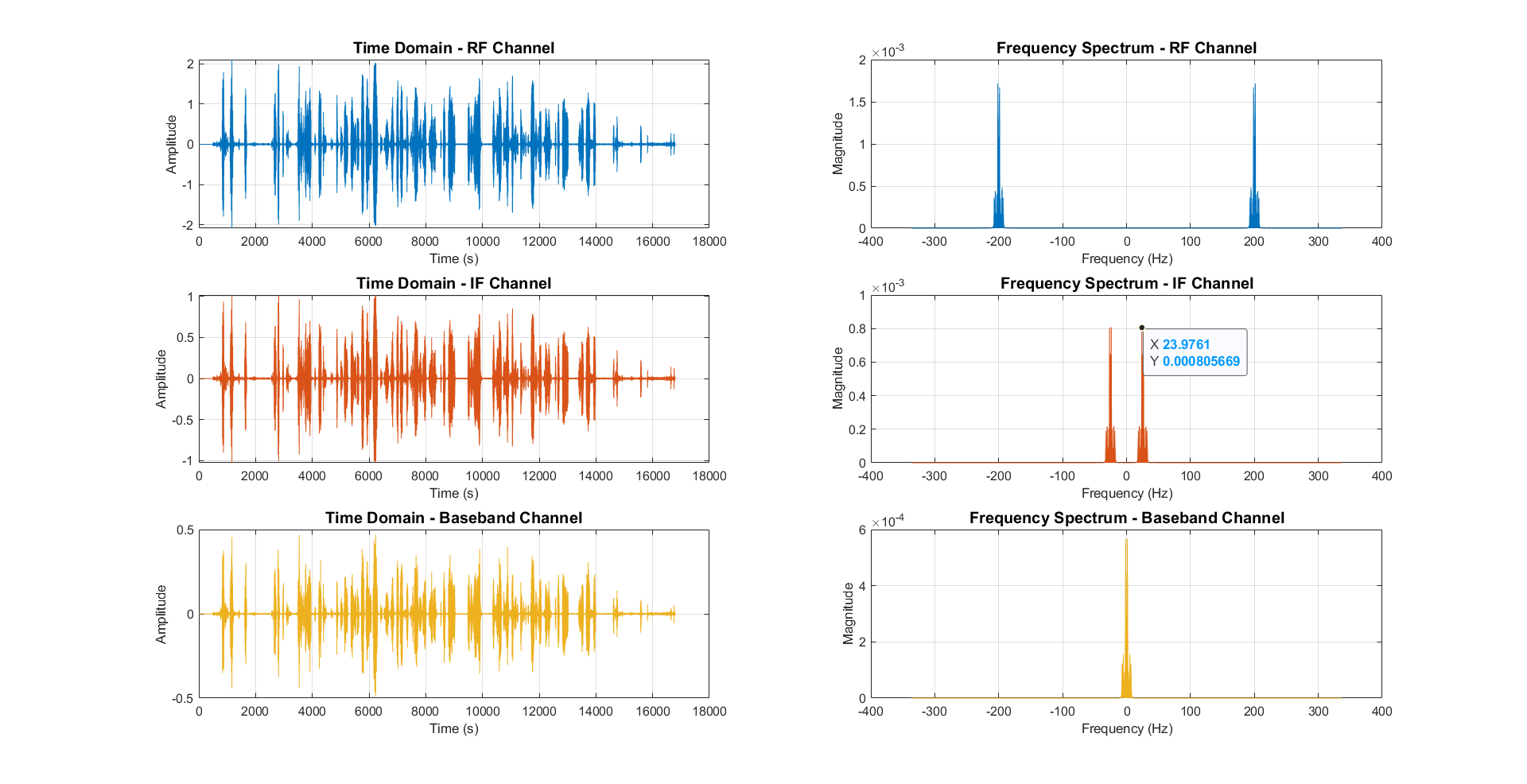


Figure Receiver Signals with Offset = 200 Hz

As shown in *fig[13]*, The RF mixer has offset = 200Hz causing a slight shift in the carrier frequency during modulation and demodulation. This offset results in a minor distortion in the reconstructed signal at the receiver.

While the original message remains somewhat understandable, the quality is reduced due to the frequency mismatch.

## Frequency Offset (1200 Hz):

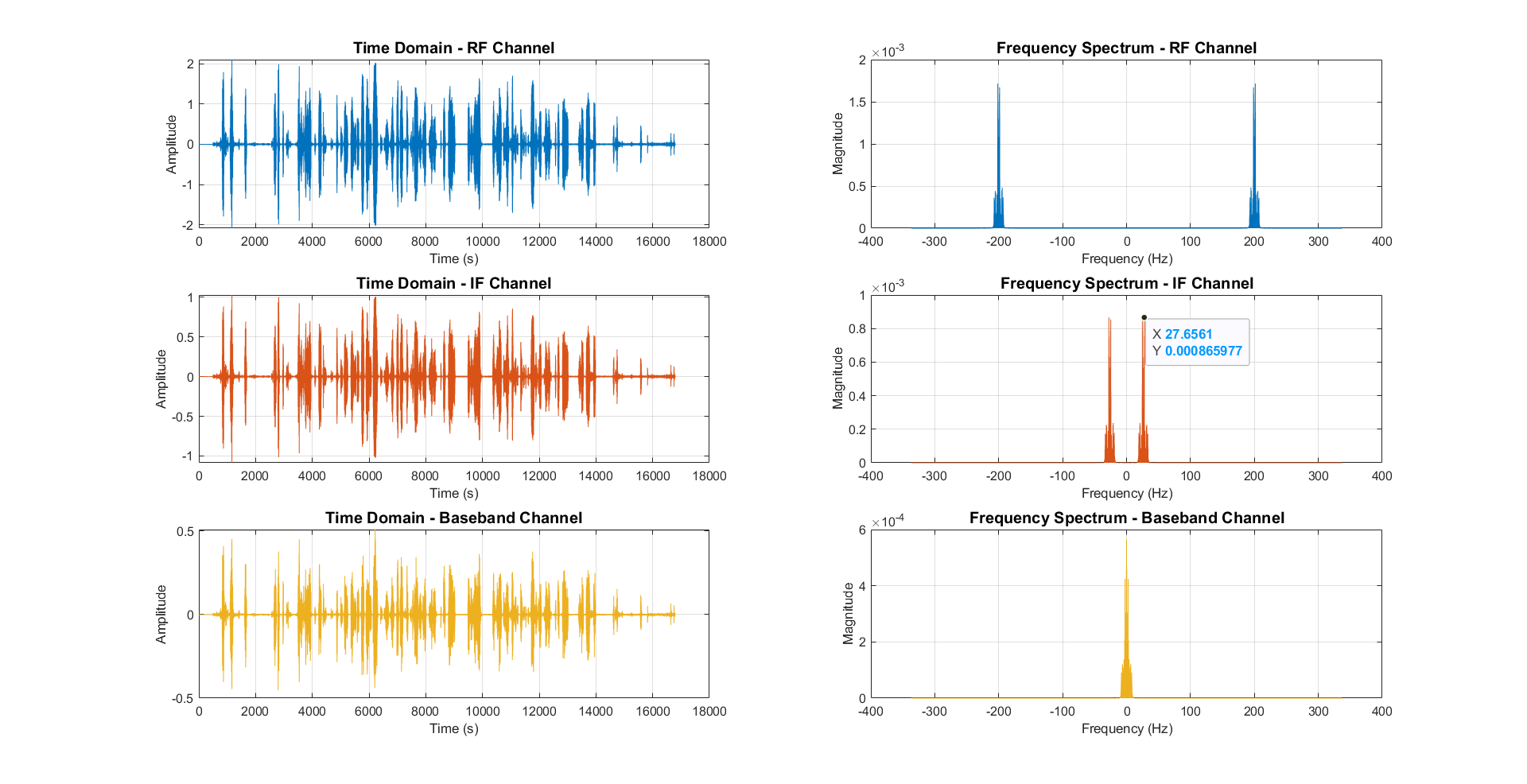


Figure Receiver Signals with Offset = 1200 Hz

As shown in *fig[13]*, The RF mixer has offset = 1200Hz resulting in a severe misalignment of the carrier frequency. This large offset distorts the modulated signal substantially, causing the demodulated message to become heavily corrupted.

# Conclusion:

In this project, we successfully designed, simulated, and analyzed a multi-stage AM communication system using MATLAB. By implementing key stages such as AM modulation, RF filtering, mixing, IF filtering, and baseband detection, we created a robust transceiver model capable of transmitting and receiving audio signals. The system was evaluated under various scenarios to understand its strengths and limitations.

Through this project, we can conclude that:[2]

* Noise and Frequency Offsets: are two major challenges in AM communication systems. These issues can significantly degrade signal quality and impair message recovery.
* The RF filter: is crucial for rejecting the IF image and other unwanted signals outside the passband, ensuring that only the desired channel is isolated and passed to the subsequent stages for accurate demodulation

# References:

[1] <https://github.com/youefkh05/Super-heterodyne-Receiver>

[2] Ziemer, R. E., Tranter, W. H., & Fannin, D. R. (2013). Digital and Analog Communication Systems. Pearson, 8th Edition, Chapter 5-2

# Appendix:

To have a good data structure to handle for the project we made AudioFile class to make it easier to code

classdef AudioFile

properties

Filename % Name of the file

SamplingFrequency % Sampling frequency of the audio (in kHz)

AudioData % Actual audio data

duration % File length

player % Object to play the sound file

end

methods

function obj = AudioFile(filename)

% Constructor method to initialize the object

if nargin > 0

[audioData, fs] = audioread(filename);

% Convert to mono if stereo

if size(audioData, 2) == 2

mono\_audioData = sum(audioData, 2); % Sum the two channels to get a mono signal

else

mono\_audioData = audioData;

end

obj.Filename = filename;

obj.SamplingFrequency = fs;

obj.AudioData = mono\_audioData;

obj.duration = length(obj.AudioData) / obj.SamplingFrequency;

obj.player = audioplayer(obj.AudioData, obj.SamplingFrequency);

obj.SamplingFrequency = fs/1000;

end

end

function playAudio(obj)

% Method to play the audio file

if ~isempty(obj.player)

play(obj.player);

else

disp('No audio data to play.');

end

end

function pauseAudio(obj)

% Method to pause the audio file

if ~isempty(obj.player)

pause(obj.player);

else

disp('No audio data to pause.');

end

end

function stopAudio(obj)

% Method to stop the audio file

if ~isempty(obj.player)

stop(obj.player);

else

disp('No audio data to stop.');

end

end

function printAudio(obj)

% Method to print all the audio file data

if ~isempty(obj.AudioData)

disp(['Filename: ', obj.Filename]);

disp(['Sampling Frequency: ', num2str(obj.SamplingFrequency), ' kHz']);

disp(['Audio Duration: ', num2str(obj.duration), ' seconds']);

% Optional: Display first few samples of audio data

%disp('First 10 samples of Audio Data:');

%disp(obj.AudioData(1:10));

else

disp('No audio data to print.');

end

end

end

end

We used functions that we made to make the code more readable. We will show the main functions and the other are in the reference [1].

## Main code:

clear all;

close all;

clc;

% Add the Functions and Filters folder to the MATLAB path temporarily

addpath('Functions');

addpath('Filters');

load RF\_Band\_Pass\_Filter; % Load predefined RF Band-Pass Filter

% List of channel audio file names

fileNames = [...

"Ch0\_Short\_QuranPalestine.wav", ...

"Ch1\_Short\_FM9090.wav", ...

"Ch2\_Short\_BBCArabic2.wav", ...

"Ch3\_Short\_RussianVoice.wav", ...

"Ch4\_Short\_SkyNewsArabia.wav"];

ChannelPath = "Channels\";

% Step 1: Load and visualize channel data

channels = read\_channels(fileNames, ChannelPath); % Read audio files into 'channels' structure

print\_channels\_data(channels); % Display details of each channel

plotChannelSpectrum(channels, "Channel", 1); % Plot spectrum of the channels

% Step 2: Determine max duration, sampling frequency, and length

[maxDuration, maxSamplingFreq, maxLength] = getMaxAudioInfo(channels);

% Display the results

fprintf('Max Duration: %.2f seconds\n', maxDuration);

fprintf('Max Sampling Frequency: %.2f kHz\n', maxSamplingFreq);

fprintf('Max Audio Data Length (number of samples): %d\n', maxLength);

% Step 3: Pad audio files to ensure equal lengths and uniform sampling rate

channels = padAudioFiles(channels, maxLength, maxSamplingFreq);

Total\_BW = 7 \* plotChannelSpectrum(channels, "Channel", 1); % Calculate total bandwidth

% Display total bandwidth across all channels

fprintf('Total Bandwidth: %.2f kHz\n', Total\_BW);

% Ensure the sampling frequency covers the required bandwidth

if Total\_BW>=maxSamplingFreq

%we gonna resample the audio files

maxSamplingFreq=Total\_BW;

fprintf('Max Sampling Frequency: %.2f kHz\n', maxSamplingFreq);

channels = padAudioFiles(channels, maxLength, maxSamplingFreq);

%check the new max length and frequency

[maxDuration, maxSamplingFreq, maxLength] = getMaxAudioInfo(channels);

end

% Save padded audio files

channels=saveChannelsAsWav(channels, "ch\_pad", "Channels\Padded");

plotChannelSpectrum(channels, "Padded Channel", 1); % Visualize padded signals

% Step 4: AM Modulation (DSB-SC)

AM\_Modulated\_Signal = AM\_Modulate\_DSB\_SC(channels, maxLength, maxSamplingFreq );

% Optional Step: Add Noise

Noise = input('Do you Want to add noise?\n Yes: y No: anything else \n', 's');

if Noise == "y"

% Get user input SNR

SNR\_dB = input(["How Much SNR (db) ?\n"]);

%add the noise

fprintf('Adding Noise ...\n');

Fs=ceil(1000\*maxSamplingFreq);

AM\_Modulated\_Signal = addNoiseToAudio(AM\_Modulated\_Signal, SNR\_dB, Fs);

end

% Save and plot the modulated signal

AM\_Modulated\_Signal=saveChannelsAsWav(AM\_Modulated\_Signal, "ch\_AM", "Channels\AM");

plotChannelSpectrum(AM\_Modulated\_Signal, "AM DSB SC", 0);

% Step 5: RF Stage - Select a Channel

% Filter parameters

Fc = 100e3 ; % Carrier frequency in Hz

fDelta=50e3; % Channel spacing in Hz

Fs=ceil(1000\*maxSamplingFreq); % Sampling frequency in Hz

Channel\_BandWidth = 40e3; % Bandwidth of each channel in Hz

% Signals Parameters

maxChannelNumber = length(channels);

% Filter to get the channel desired

[AM\_Modulated\_Signal\_RF\_Filter,ChannelNumber, Channel\_Frequency, RF\_BPF] = ...

choose\_channel(AM\_Modulated\_Signal,maxChannelNumber, Fc, fDelta, ...

Channel\_BandWidth,Fs);

% Optional Step: Remove RF Filter

RF\_Filter = input("Do you Want to add RF Filter?\n"+...

"Yes: y No: anything else \n", 's'); % Specify 's' for string input

if RF\_Filter == "y"

% Visualize the new filter

fprintf('You selected Channel %d with carrier frequency %.1f kHz.\n', ChannelNumber, Channel\_Frequency / 1e3);

%Plot the frequency response of RF Bandpass Filter

plotFilter(RF\_BPF, Fs, "Frequency Response of RF Bandpass Filter");

else

AM\_Modulated\_Signal\_RF\_Filter = AM\_Modulated\_Signal;

end

% Step 6: IF Stage

WIF = 25e3; % Intermediate Frequency (IF) in Hz

% Optional Step: Add offset to IF

offset = input('Do you Want to add offset?\n Yes: y No: anything else \n', 's');

if offset == "y"

% Get user input offset

offset\_frequency = input(["What's your offset (hz) ?\n"]);

%add the offset

fprintf('Adding Offset ...\n');

IF\_Channel = mixer(AM\_Modulated\_Signal\_RF\_Filter, Channel\_Frequency, WIF + offset\_frequency,Fs);

else

%no offset

IF\_Channel = mixer(AM\_Modulated\_Signal\_RF\_Filter, Channel\_Frequency, WIF ,Fs);

end

%IF Filter

[IF\_Channel\_Filtered,IF\_BPF] = ...

if\_stage(IF\_Channel, WIF,Channel\_BandWidth,Fs);

%Plot the frequency response of IF Bandpass Filter

plotFilter(IF\_BPF, Fs, "Frequency Response of IF Bandpass Filter");

% Step 7: Baseband Stage

[Bass\_Band\_Channel, Bass\_Band\_Filter] = baseband\_detection(IF\_Channel\_Filtered, WIF, Fs);

%Plot the frequency response of Bass\_Band Low Pass Fileter

plotFilter(Bass\_Band\_Filter, Fs, "Frequency Response of Bass Band Low Pass Filter");

% Step 8: Save and Visualize Receiver Signals

plotReceiver(AM\_Modulated\_Signal\_RF\_Filter, IF\_Channel\_Filtered, Bass\_Band\_Channel);

%Save as Wav

AM\_Modulated\_Signal\_RF\_Filter=saveChannelsAsWav(AM\_Modulated\_Signal\_RF\_Filter, "ch\_RF\_Filter", "Channels\RF");

IF\_Channel\_Filtered=saveChannelsAsWav(IF\_Channel\_Filtered, "IF\_Channel", "Channels\IF");

Bass\_Band\_Channel=saveChannelsAsWav(Bass\_Band\_Channel, "Bass\_Band\_Channel", "Channels\Bass\_Band");

% Optional Step: Listening to Bass Band

Bass\_Band\_play = input('Do you Want to listen the received channel?\n Yes: y No: anything else \n', 's');

if Bass\_Band\_play == "y"

playAudio(Bass\_Band\_Channel);

else

%nothing

end

% End of the code

fprintf('Processing complete. All stages executed and signals saved.\n');

## AM DSC SC MODULATION:

function multiplexed\_Audio\_Signal = AM\_Modulate\_DSB\_SC(channels, Length, sampleFreq)

% Function to perform AM modulation (DSB-SC) and return multiplexed signal

% Parameters:

% channels - Array of audio signals (cell array of vectors)

% sampleFreq - Sampling frequency of the audio signals (scalar)

% Returns:

% multiplexedSignal - Combined FDM signal

% Parameters for modulation

baseCarrierFreq = 100; % Base carrier frequency in kHz

deltaFreq = 50; % Frequency increment between carriers in kHz

% Time vector based on the sample frequency and signal length

%Length = max(cellfun(@length, channels)); % Ensure all signals are the same length

Ts=1/sampleFreq; %Sample Time

t=[0:Ts:Length\*Ts-Ts]; %Time vector

% Initialize the multiplexed signal

multiplexedSignal = zeros(1, Length);

% Loop through each channel and modulate it

for n = 1:length(channels)

% Get the current channel signal

signal = channels(n).AudioData;

% Carrier frequency for this channel

carrierFreq = baseCarrierFreq + (n-1) \* deltaFreq;

% Generate the carrier signal

carrier = cos(2 \* pi \* carrierFreq \* t)';

% Perform DSB-SC modulation

modulatedSignal = signal .\* carrier;

% Add the modulated signal to the multiplexed signal

multiplexedSignal = multiplexedSignal + modulatedSignal';

end

%same information

multiplexed\_Audio\_Signal = AudioFile("Channels\Padded\ch\_pad\_3.wav");

multiplexed\_Audio\_Signal.SamplingFrequency = sampleFreq;

multiplexedSignal=transpose(multiplexedSignal);

multiplexed\_Audio\_Signal.AudioData=multiplexedSignal;

end

## Mixer:

function Output\_Audio\_File = mixer(Input\_Audio\_File, Wc, WIF,Fs)

% Function to simulate a mixer

% Parameters:

% InputSignal - The input signal to be mixed

% omega\_c - Carrier frequency in radians/sec

% omega\_IF - Intermediate frequency (IF) in radians/sec

% Fs - Sampling frequency in Hz

%

% Output:

% OutputSignal - The mixed output signal

Output\_Audio\_File = Input\_Audio\_File; %to have the same information

% Time vector based on the input signal length and sampling frequency

Length=length(Input\_Audio\_File.AudioData);

Ts=1/Fs; %Sample Time

t=[0:Ts:Length\*Ts-Ts]; %Time vector

% Generate the carrier signal with frequency (omega\_c + omega\_IF)

CarrierSignal = cos(2 \* pi \* (Wc + WIF) \* t)';

% Perform the mixing (multiplication)

OutputSignal = Input\_Audio\_File.AudioData .\* CarrierSignal;

% Update the mixed object's properties

Output\_Audio\_File.AudioData = OutputSignal;

Output\_Audio\_File.player = audioplayer(OutputSignal, Fs); % Update player

end