Cairo University

Cairo University, Faculty of Engineering
Electronics and Electrical Communications
Department (EECE)

Analog Communications

MATLAB implementation of a superheterodyne receiver

By:

خالد أحمد حامد عبدالعزيز محمد كليب

ID	Section	BN
9220276	**2**	**8**

Contents

Figures

Transmitter

Preprocessing Block

Objective: Prepare audio signals for modulation and transmission.

Design Steps:

1. **Loading Signals**:
    - Audio signals were loaded using the “audioread” function.
2. **Stereo to Mono Conversion**:
    - Stereo signals were converted to mono using the average of the two channels (mean function), ensuring normalized representation.
3. **Padding**:
    - Signals were padded with zeros to equalize their lengths, ensuring compatibility during processing and multiplexing.

Preprocessing Block figures:

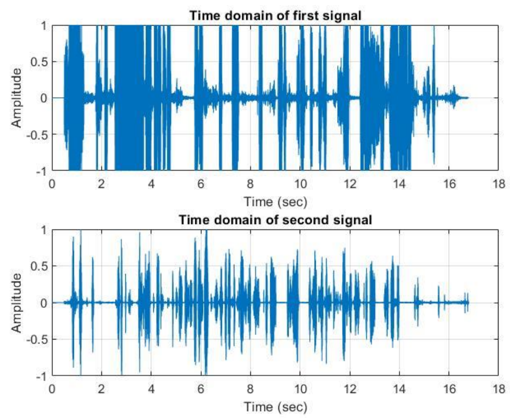

Figure 1: Time domain of the original signals

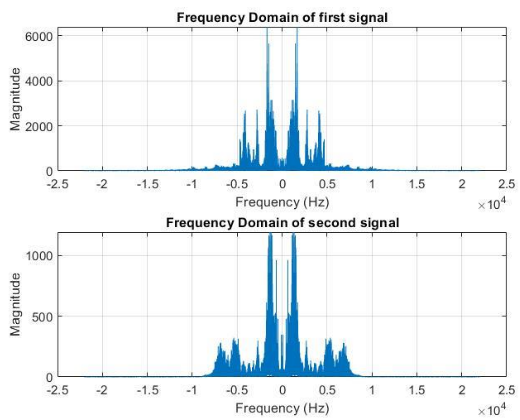

Figure 2: Frequency domain of the original signals

Modulator Block

Objective: Implement Double Sideband Suppressed Carrier (DSB-SC) AM modulation.
Design Steps:

1. **Carrier Generation**:
   - Carriers were generated for each signal using the equation: cos(2πfct) where fc is the carrier frequency.
2. **Interpolation**:
   - To avoid aliasing and meet the Nyquist criterion, signals were upsampled using the interp function.
3. **Modulation**:
   - Modulation was performed by multiplying each signal with its respective carrier.
4. **FDM Signal**:
   - Frequency-Division Multiplexing (FDM) was achieved by summing the modulated signals.

Amplitude Modulation Block figures:

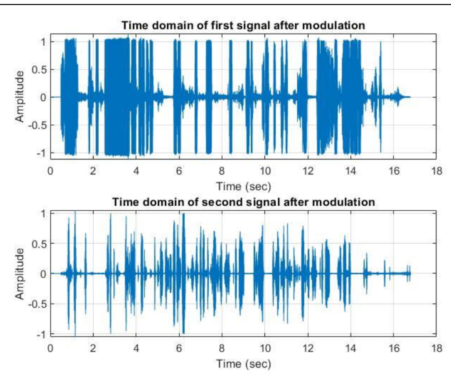

Figure 3: Time domain of the two signals after modulation

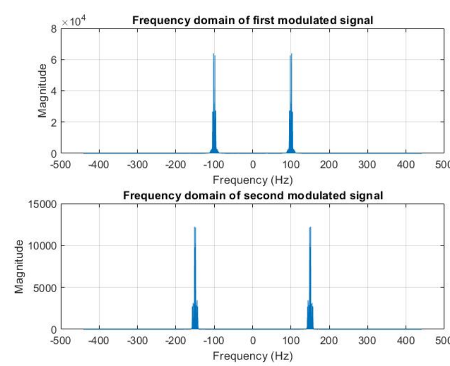

Figure 4: Frequency domain of the two signals after modulation

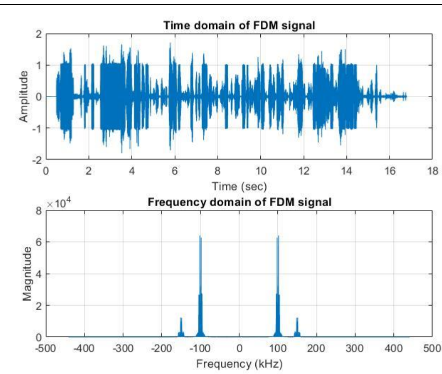

Figure 5: Time & Frequency domains of the FDM signal

Wireless Channel

Objective: Simulate a simple channel for signal transmission.

Design:

The channel is assumed to be clear of noise or distortion. No additional processing is applied during this stage. However, Additive White Gaussian Noise (AWGN) will be introduced later in **Question b-4** to simulate real-world interference and analyze its effect on the received signal.

RF Stage

Objective: Isolate the individual desired channels from the multiplexed signal.

Design Steps:

Band-Pass Filter Design:

- Tunable band-pass filters (BPFs) were designed using the “designfilt“.
- Filters were centered at each carrier frequency and had passbands defined as (fc ± bandwidth/2) and were applied to the FDM signal.

RF Stage figures:

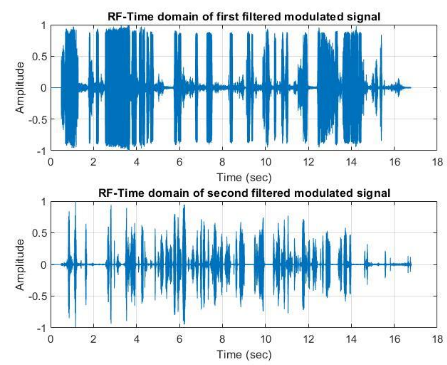

Figure 6: RF time domain of the two signals

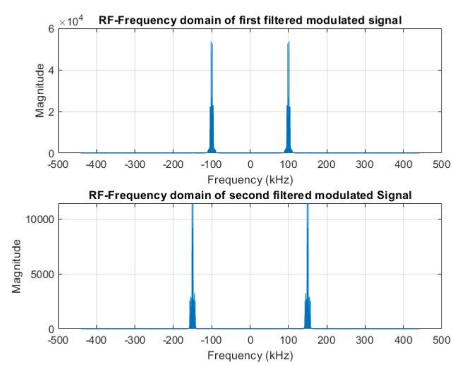

Figure 7: RF frequency domain of the two signals

Mixer Block

Objective: Shift the RF signal to an intermediate frequency (IF).

Design Steps:

1. **Local Oscillator Generation**:
   - A local oscillator signal of frequency fLO = fc + f_IF was generated using cos(2π*fLO*t).
2. **Mixing**:
   - The RF-filtered signal was multiplied by the local oscillator signal to shift the frequency spectrum to IF, This step prepares the signals for demodulation while maintaining their integrity.

Mixer Block figures:

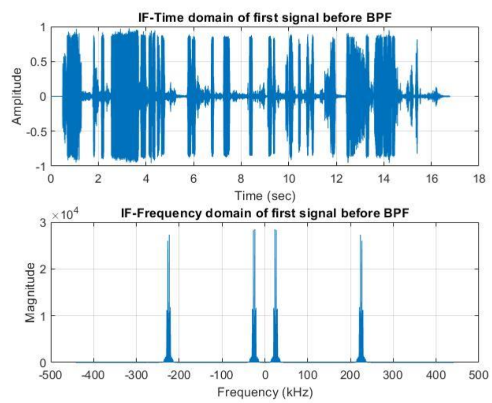

Figure 8: The first signal after mixing with 125 KHz source at time and frequency domains

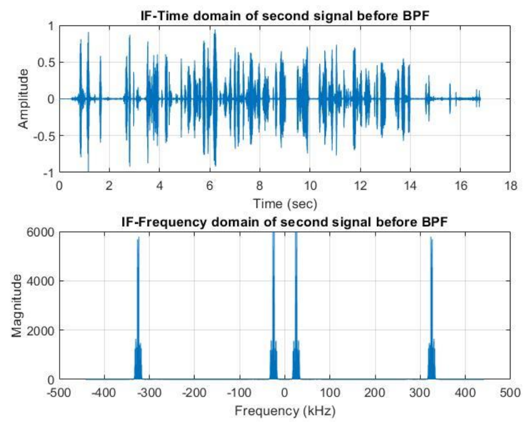

Figure 9: The second signal after mixing with 175 KHz source at time and frequency domains

IF Stage

Objective: Isolate the desired signal at the intermediate frequency.

Design Steps:

1. **Band-Pass Filter Design**:
    - A second set of band-pass filters, centered at fIF, was designed using "designfilt" to isolate the IF signal.
2. **Filtering**:
    - The mixed signals were passed through these filters to eliminate unwanted spectral components, retaining only the desired IF signal ensuring image rejection.

IF Stage figures:

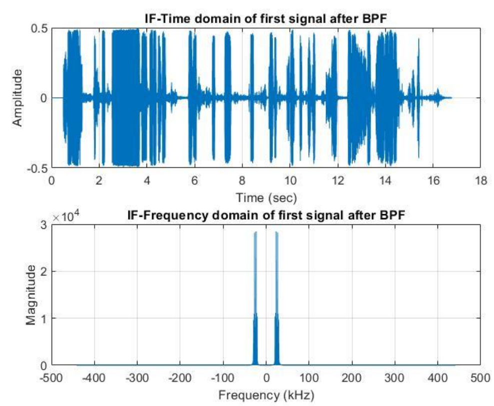

Figure 10: The first signal after passing the IF BPF

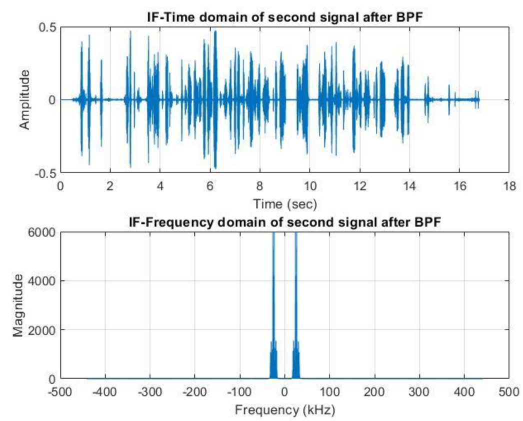

Figure 11: The second signal after after passing the IF BPF

Baseband Detection stage

Objective: Recover the original signal from the IF stage.

Design Steps:

1. **Local Oscillator for Downconversion**:
    - A new local oscillator signal of frequency f_IF was generated for down conversion.
2. **Mixing**:
    - The IF signal was mixed with the local oscillator signal to shift the spectrum back to baseband.
3. **Low-Pass Filtering**:
    - The resulting signal was passed through a low-pass filter to extract the baseband signal, eliminating high-frequency components.

Baseband Stage figures:

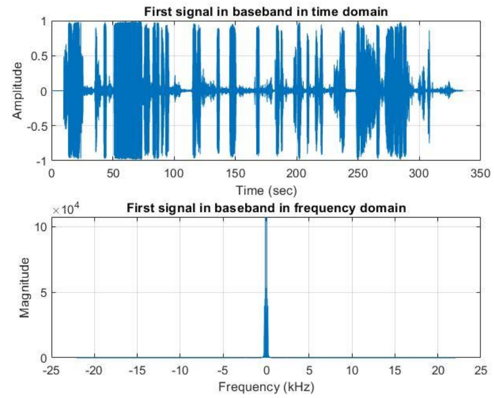

Figure 12: The first signal in basband in time domain

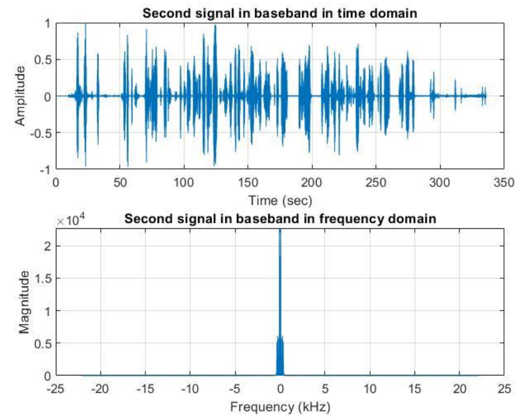

Figure 13: The second signal in baseband in frequency domain

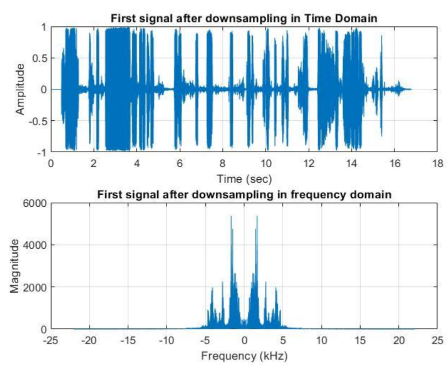

Figure 14: The first signal after reciption

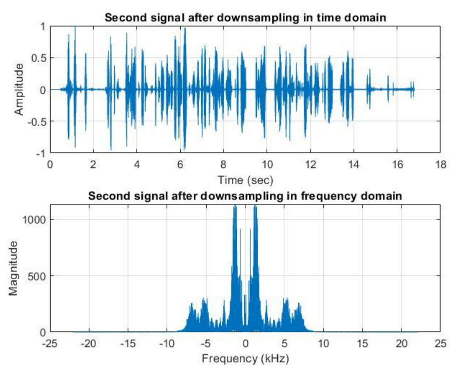

Figure 15: The second signal afer reciption

Discussion part

Comment on the output received sound

The “Sound” command was used to play the demodulated signals to verify the successful recovery of the transmitted audio. Upon listening, the signals were clear and closely resembled the original monophonic audio, confirming the correct operation of the super-heterodyne receiver.

Impact of adding noise

I have added noise to both signals after RF-Stage, During playback with the "`Sound`" command, the signals were audibly distorted, with the severity depending on the Signal-to-Noise Ratio (SNR) applied. Lower SNR values resulted in more noticeable noise and reduced clarity of the audio.

I have added a noise to both signals with SNR equals 15 dB for the first signal and -15 dB for the second signal

Noise effect figures:

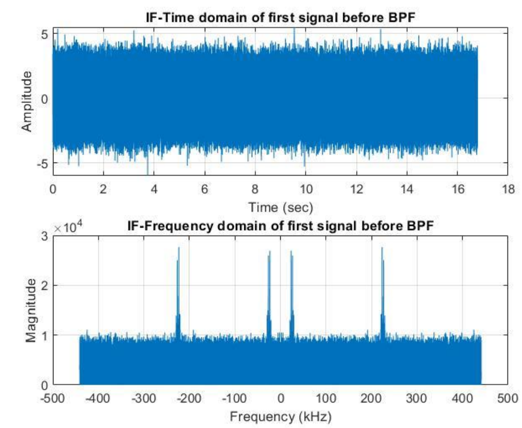

Figure 16: The first signal after mixing with 125 KHz source at time and frequency domains after adding noise

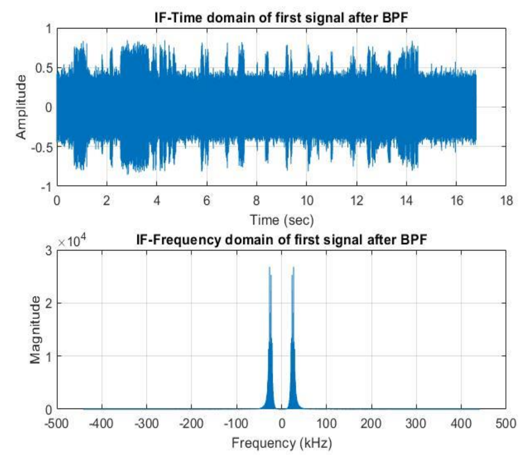

Figure 17: The first noisy signal after passing he IF BPF

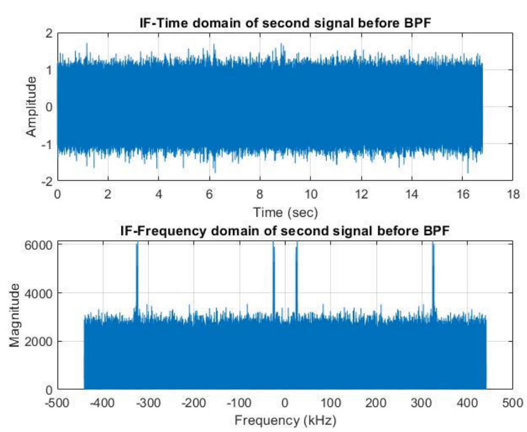

Figure 18: The first signal after mixing with 175 KHz source at time and frequency domains after adding noise

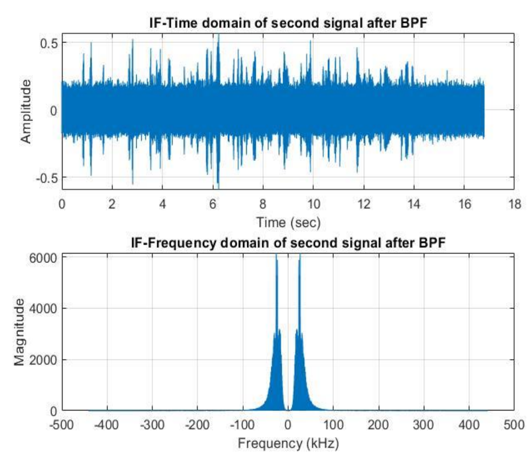

Figure 19: The second noisy signal after passing the IF BPF

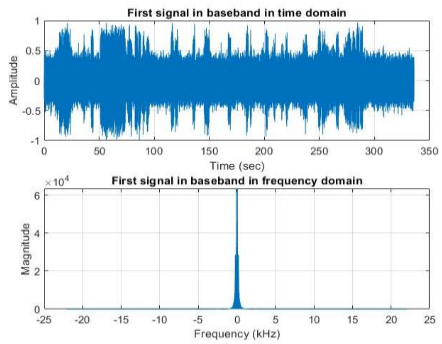

Figure 20: The first noisy signal at baseband

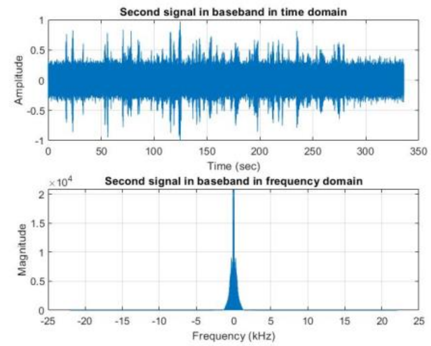

Figure 21: The Second noisy signal at baseband

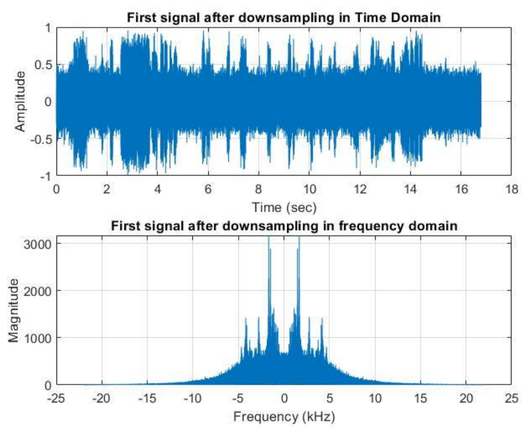

Figure 22: The first noisy signal after reception

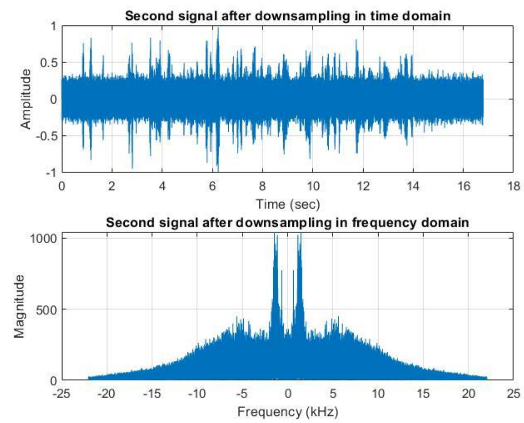

Figure 23: The second noisy signal after reception

Impact of removing RF-BPF

When the RF band-pass filter (BPF) was removed, the demodulation process for the station at ω0 was significantly affected. Without the RF stage, undesired signals and interference from other stations were not filtered out (Our two signals wil interfere with each other), resulting in overlapping frequency components. During demodulation, this caused distortion in the output signal, with multiple channels interfering, making it difficult to recover the desired station at ω0 . This highlights the critical role of the RF stage in isolating specific channels in frequency-division multiplexing systems.

Removing RF-LPF figures:

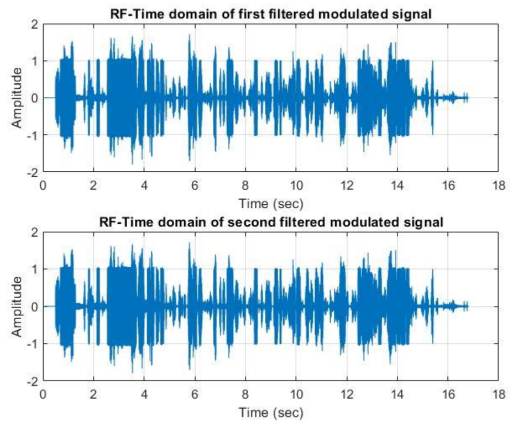

Figure 24: RF time domain of both two signals

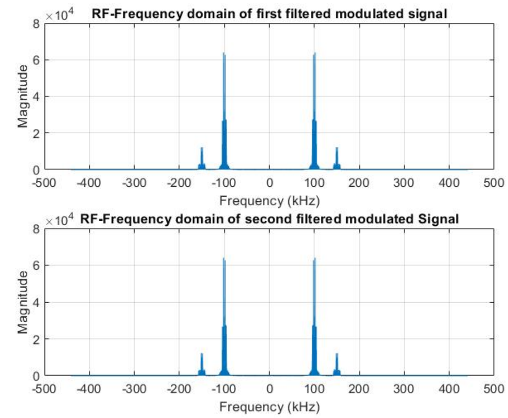

Figure 25: RF frequency domain of both two signals

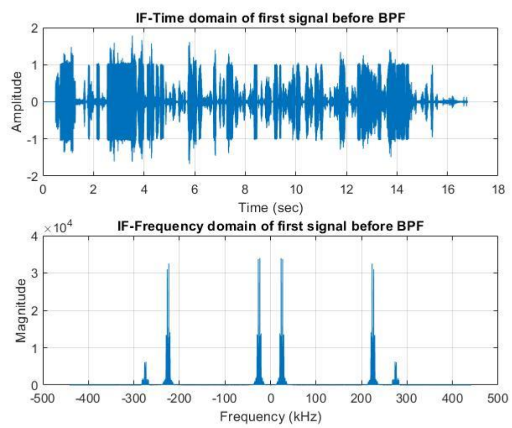

Figure 26: The first signal after mixing with 125 KHz source at time and frequency domains

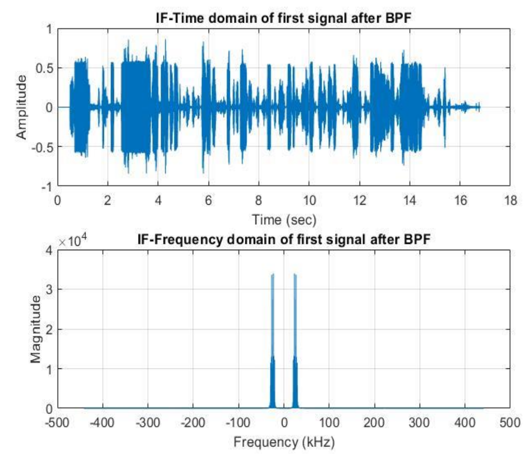

Figure 27: The first signal after passing the IF BPF

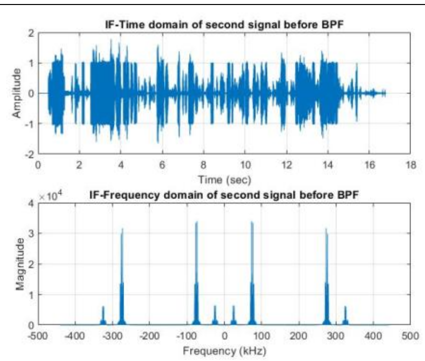

Figure 28: The second signal after mixing with 175 KHz source at time and frequency domains

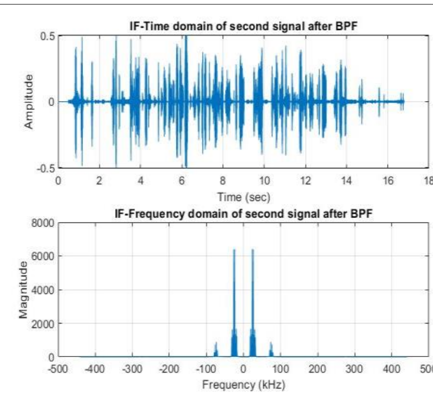

Figure 29: The Second signal after passing the IF BPF

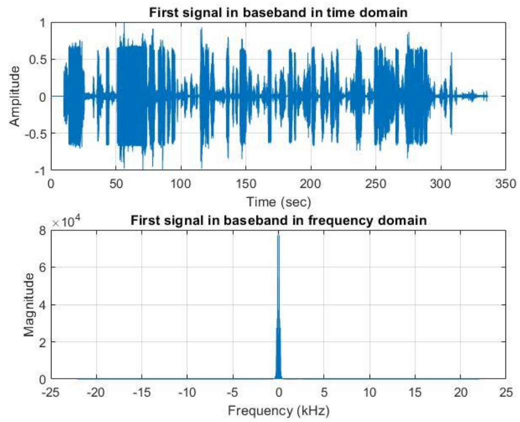

Figure 30: The first signal at baseband

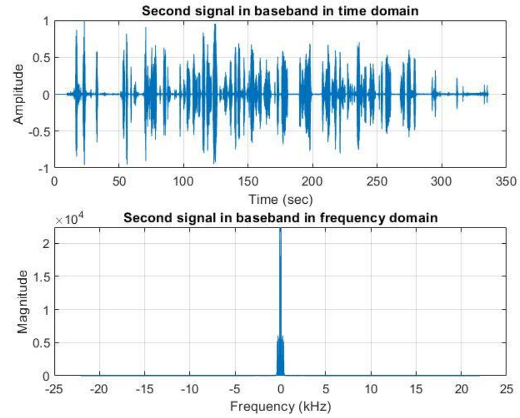

Figure 31: The Second signal at baseband

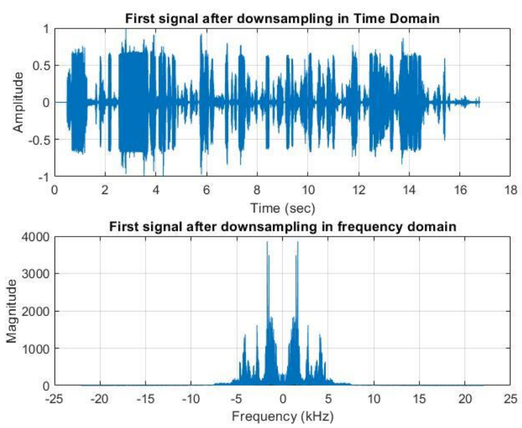

Figure 32: The first signal after reception

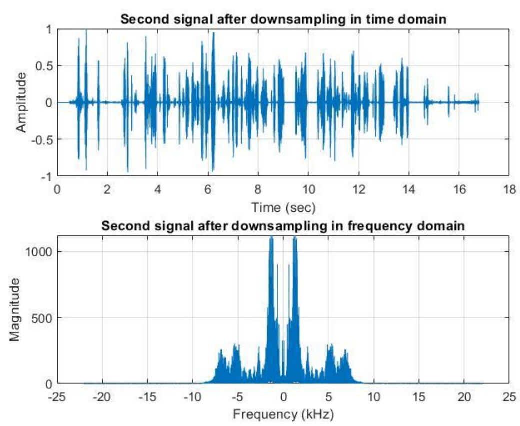

Figure 33: The second signal after reception

Impact of Receiver Oscillator Frequency Offset

When the receiver oscillator has a frequency offset:

1. **Spectrum Analysis**:
   - **0.2 kHz Offset**: A small frequency offset introduces a slight shift in the spectrum, causing imperfect alignment of the desired signal's carrier frequency with the oscillator. This results in minor distortion and incomplete signal demodulation, but the signal remains somewhat recognizable.
   - **1.2 kHz Offset**: A larger offset causes significant spectral misalignment, leading to severe distortion in the demodulated signal. The desired signal is mixed with unintended frequency components, resulting in substantial loss of signal integrity.

2. **Sound Quality**:
   - **0.2 kHz Offset**: The audio becomes slightly distorted, with a noticeable warping or "beating" effect in the sound, but it remains intelligible.
   - **1.2 kHz Offset**: The sound quality degrades drastically, often rendering the audio unintelligible as noise and interference dominate the signal.

These observations emphasize the importance of precise frequency synchronization in super-heterodyne receivers to ensure accurate signal demodulation and recovery.

I have applied two frequency offsets 200 Hz and 1200 Hz for both signals and these are the results:

Receiver Oscillator Frequency Offset figures:

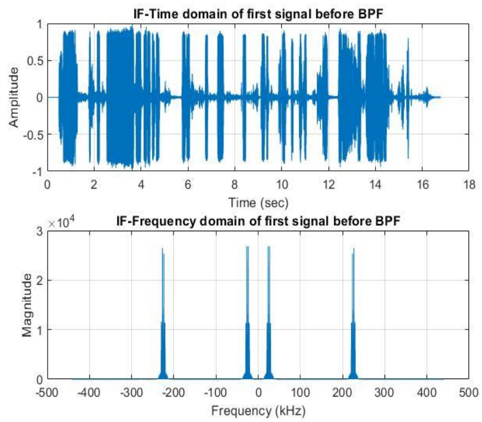

Figure 34: The first signal with offset .2 KHz after mixing with 125 KHz source at time and frequency domains

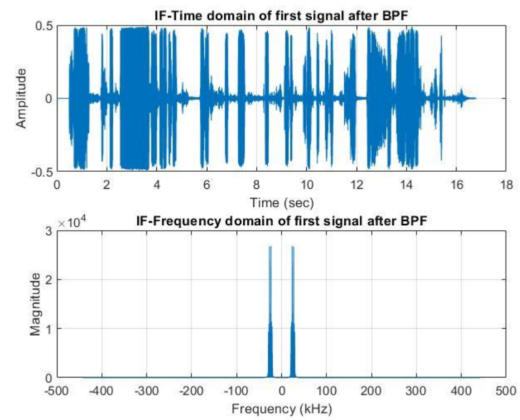

Figure 35: The first signal with offset .2 KHz after passing the IF BPF

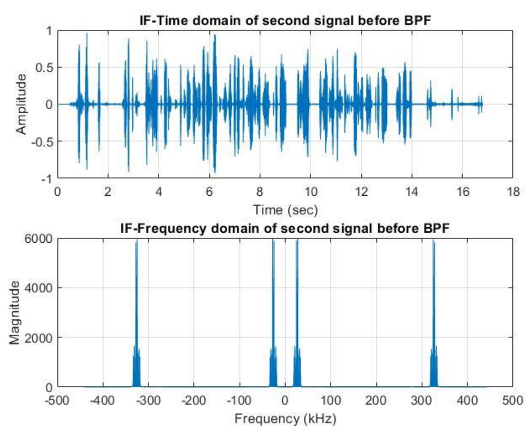

Figure 36: The second signal with offset 1.2 KHz after mixing with 175 KHz source at time and frequency domains

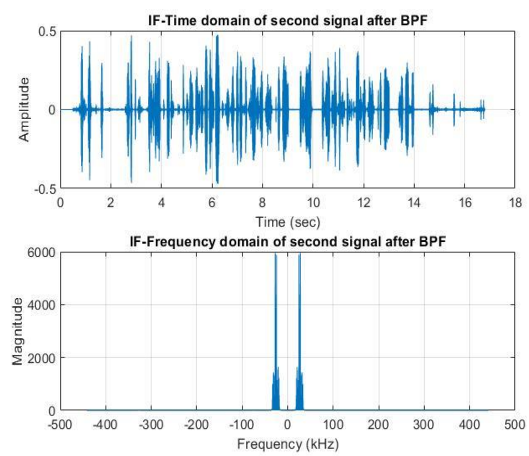

Figure 37: The Second signal with offset 1.2 KHz after passing the IF BPF

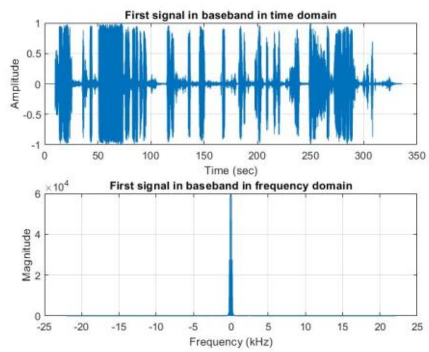

Figure 38: The first signal with offset .2 KHz at baseband

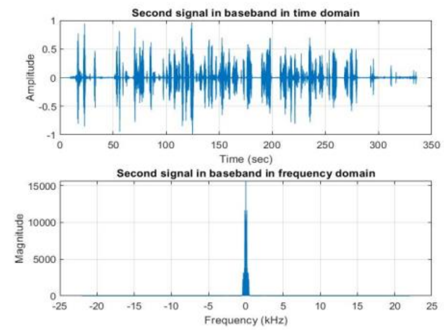

Figure 39: The Second signal with offset 1.2 KHz at baseband

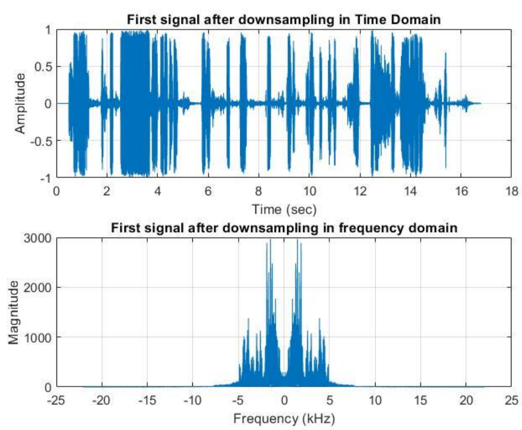

Figure 40: The first signal with offset .2 KHz after reception

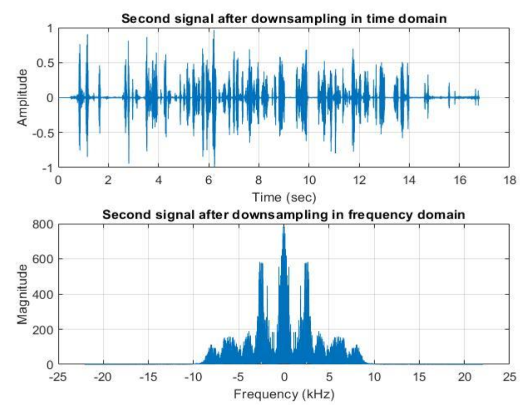

Figure 41: The Second signal with offset 1.2 KHz after reception

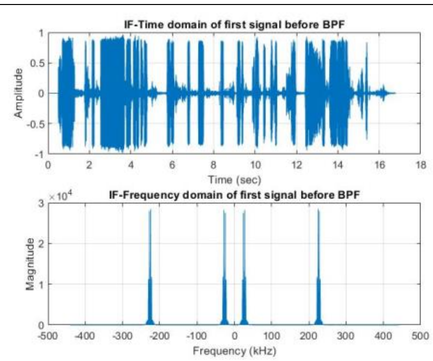

Figure 42: The first signal with offset 1.2 KHz after mixing with 125 KHz source at time and frequency domains

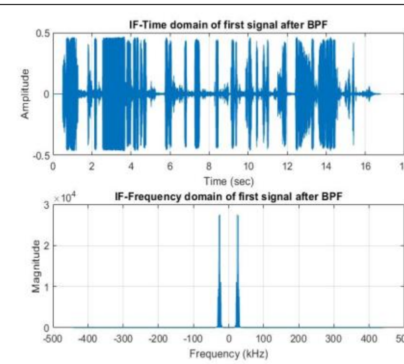

Figure 43: The first signal with offset 1.2 KHz after passing the IF BPF

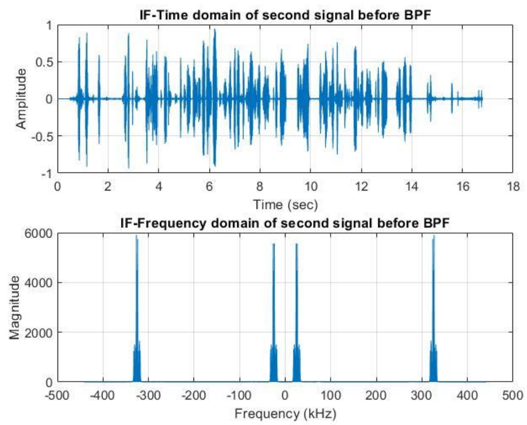

Figure 44: The second signal with offset .2 KHz after mixing with 175 KHz source at time and frequency domains

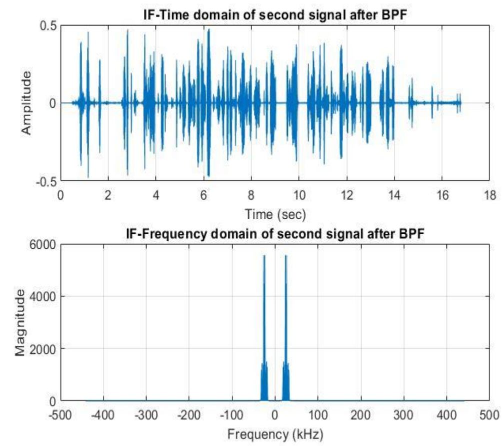

Figure 45: The Second signal with offset .2 KHz after passing IF BPF

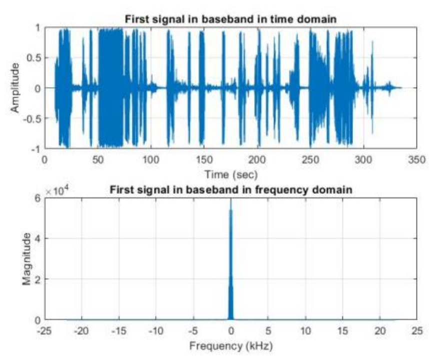

Figure 46: The first signal with offset 1.2 KHz at baseband

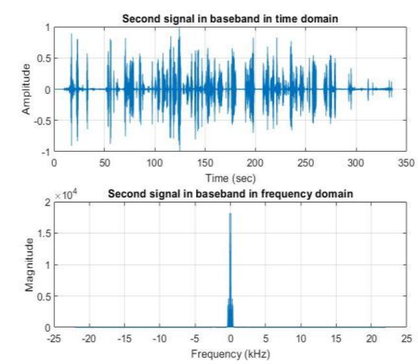

Figure 47: The second signal with offset .2 KHz at baseband

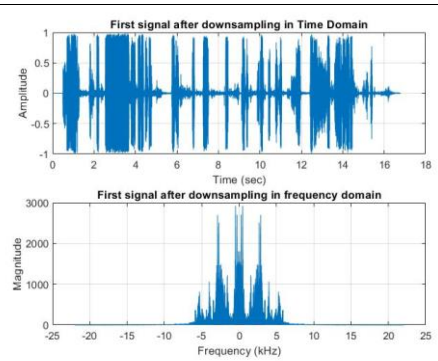

Figure 48: The first signal with offset 1.2 KHz after reception

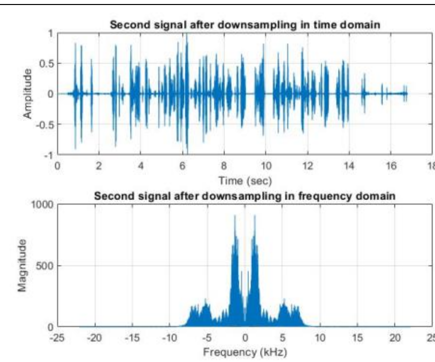

Figure 49: The second signal with offset .2 KHz after reception

Implementation on matlab

```matlab
%% Analog Modulation - Communication Project
% Super-Heterodyne Receiver Simulation
%% Clearing the workspace
clear; clc; close all;
%% i : Loading signals, Converting from stereo to mono & Padding
signals
% Using mean provides a normalized mono representation
[stereo1, fs1] = audioread("Short_QuranPalestine.wav");% Load the
first audio signal
mono1 = mean(stereo1, 2); % Convert first signal from stereo to mono
[stereo2, fs2] = audioread("Short_BBCArabic2.wav");% Load the second
audio signal
mono2 = mean(stereo2, 2); % Convert second signal from stereo to mono
% Ensure both signals have the same sampling frequency
if fs1 ~= fs2
    error('Sampling frequencies of the two audio files does not
match.');
end
fs = fs1;
% Pad the shorter signal to match the length of the longer signal
maxlength = max(length(mono1), length(mono2));
mono1 = [mono1; zeros(maxlength - length(mono1), 1)];
mono2 = [mono2; zeros(maxlength - length(mono2), 1)];
% Plotting signals in time before modulation
figure ;
% Plot the first signal in time domain
t1 = (0:maxlength-1)'/fs;
subplot(2, 1, 1);
plot(t1, mono1);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Time domain of first signal');
grid on;
% Plot the second signal in time domain before modulation
t2 = (0:maxlength-1)'/fs;
subplot(2, 1, 2);
plot(t2, mono2);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Time domain of second signal');
grid on;
% Plot signals in frequency domain
figure ;
% Plot the first signal in frequency domain
N1 = length(mono1);
f1 = (-N1/2:N1/2-1) * (fs/N1);
spectrum1 = fftshift(abs(fft(mono1)));
subplot(2, 1, 1);
plot(f1, spectrum1);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency Domain of first signal');
grid on;
% Plot the second signal in frequency domain
```

```
N2 = length(mono2);
f2 = (-N2/2:N2/2-1) * (fs/N2);
spectrum2 = fftshift(abs(fft(mono2)));
subplot(2,1, 2);
plot(f2, spectrum2);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency Domain of second signal');
grid on;
%% ii : AM Modulation (DSB-SC)
% Defining parameters
fc1 = 100e3; % Carrier frequency for first signal in Hz
df =50e3; % Frequency increment for subsequent signals
bandwidth = 25e3; % bandwidth in Hz
filterorder = 10;
f_IF = 25e3; % Intermediate Frequency (IF)
wc1 = fc1 + f_IF; % Carrier frequency
wc2 = fc1+ f_IF + df;
fsmultiplier = 20;
fsnew= fs * fsmultiplier; % New sampling frequency
ts = 1 / fsnew; % Smpling rate
% Resample the signals to increase Fs
mono1_interp = interp(mono1, fsmultiplier);
mono2_interp = interp(mono2, fsmultiplier);
% Generating carriers
N = max(length(mono1_interp), length(mono2_interp));% No need for
comparison as they are the same lenght after padding
t = (0:N-1)' * ts;
carrier1 = cos(2 * pi * fc1 * t);
carrier2 = cos(2 * pi * (fc1 + df) * t);
% Modulating signals (DSB-SC)
modulated1 = (mono1_interp .* carrier1);
modulated2 = (mono2_interp .* carrier2);
% Plotting signals in time after modulation
figure ;
% Plot the first signal in time domain
subplot(2, 1, 1);
plot(t, modulated1);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Time domain of first signal after modulation');
grid on;
% Plot the second signal in time domain after modulation
subplot(2, 1, 2);
plot(t, modulated2);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Time domain of second signal after modulation');
grid on;
% Plot signals in frequency domain after modulation
figure ;
% Plot the first signal in frequency domain
modulatedspectrum1 = fftshift(abs(fft(modulated1)));
fmod1 = linspace(-fsnew/2, fsnew/2, length(modulated1)) / 1e3;
subplot(2, 1, 1);
plot(fmod1, modulatedspectrum1);
xlabel('Frequency (Hz)');
```

```
ylabel('Magnitude');
title('Frequency domain of first modulated signal');
grid on;
% Plot the second signal in frequency domain
modulatedspectrum2 = fftshift(abs(fft(modulated2)));
fmod2 = linspace(-fsnew/2, fsnew/2, length(modulated2)) / 1e3;
subplot(2,1, 2);
plot(fmod2, modulatedspectrum2);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Frequency domain of second modulated signal');
grid on;
% Combine modulated signals (FDM)
FDMsignal = modulated1 + modulated2;
% Plot the FDM signal in the time domain
figure;
subplot(2, 1, 1);
plot(t, FDMsignal);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Time domain of FDM signal');
grid on;
% Plot the FDM signal spectrum
FDMspectrum = fftshift(fft(FDMsignal));
frequencies = linspace(-fsnew/2, fsnew/2, length(FDMsignal)) / 1e3; %
in kHz
subplot(2, 1, 2);
plot(frequencies, abs(FDMspectrum));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('Frequency domain of FDM signal');
grid on;
%% iii : Wireless Channel & RF stage
% Band-Pass Filter for the first modulated signal
fpass1 = [fc1 - bandwidth/2, fc1 + bandwidth/2]; % Passband range for
Signal 1
BPF_RF1 = designfilt('bandpassiir', 'FilterOrder', filterorder, ...
                     'HalfPowerFrequency1', fpass1(1),
'HalfPowerFrequency2', fpass1(2), ...
                     'SampleRate', fsnew);
filteredsignal1 = filter(BPF_RF1, FDMsignal);% Applying BPF to the
first modulated signal
filteredsignal1 = filteredsignal1 / max(abs(filteredsignal1));%
Normalizing filtered signals
snr1 = -15; % Signal-to-Noise Ratio (SNR) for first signal in dB
noisy_RF1 = awgn(filteredsignal1, snr1, 'measured'); % Add Gaussian
noise to Signal 1
%filteredsignal1 = FDMsignal;
% Band-Pass Filter for the second modulated signal
fpass2 = [fc1 + df - bandwidth/2, fc1 + df + bandwidth/2]; % Passband
range for Signal 2
BPF_RF2 = designfilt('bandpassiir', 'FilterOrder', filterorder, ...
                     'HalfPowerFrequency1', fpass2(1),
'HalfPowerFrequency2', fpass2(2), ...
                     'SampleRate', fsnew);
filteredsignal2 = filter(BPF_RF2, FDMsignal);% Apply BPF to the second
modulated signal
```

```
filteredsignal2 = filteredsignal2 / max(abs(filteredsignal2));%
Normalize filtered signals
snr2 = 15; % Signal-to-Noise Ratio (SNR) for second signal in dB
noisy_RF2 = awgn(filteredsignal2, snr2, 'measured');% Add Gaussian
noise to Signal 2
% filteredsignal2 = FDMsignal;
% Plot signals in time domain after RF-Stage
% Plot the first filtered signal in time domain
figure;
subplot(2, 1, 1);
plot(t, filteredsignal1);
xlabel('Time (sec)');
ylabel('Amplitude');
title('RF-Time domain of first filtered modulated signal');
grid on;
% Plot the second filtered signal in time domain
subplot(2, 1, 2);
plot(t, filteredsignal2);
xlabel('Time (sec)');
ylabel('Amplitude');
title('RF-Time domain of second filtered modulated signal');
grid on;
% Plot signals in frequency domain after RF-Stage
% Plot the spectrum of the first filtered signal
figure;
filteredspectrum1 = fftshift(fft(filteredsignal1));
freqfiltered = linspace(-fsnew/2, fsnew/2, length(filteredsignal1)) /
1e3; % Convert to kHz
subplot(2, 1, 1);
plot(freqfiltered, abs(filteredspectrum1));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('RF-Frequency domain of first filtered modulated signal');
grid on;
% Plot the spectrum of the second filtered signal
filteredspectrum2 = fftshift(fft(filteredsignal2));
freqfiltered = linspace(-fsnew/2, fsnew/2, length(filteredsignal2)) /
1e3; % Convert to kHz
subplot(2, 1, 2);
plot(freqfiltered, abs(filteredspectrum2));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('RF-Frequency domain of second filtered modulated Signal');
grid on;
%% iv : Mixer Stage & IF Stage
% Mixer for the first filtered signal
ifcarrier1 = cos(2 * pi * wc1 * t);
ifsignal1 = filteredsignal1 .* ifcarrier1;%shift to IF frequency
% Plot IF first signal before band-pass filter in time domain
figure;
subplot(2, 1, 1);
plot(t, ifsignal1);
xlabel('Time (sec)');
ylabel('Amplitude');
title('IF-Time domain of first signal before BPF');
grid on;
% Plot IF first signal before filtering in frequency domain
```

```
IFspectrum1before = fftshift(fft(ifsignal1));
subplot(2, 1, 2);
plot(frequencies, abs(IFspectrum1before));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('IF-Frequency domain of first signal before BPF');
grid on;
ifbw1 = 10e3; % Bandwidth of the IF filter in Hz
% Design the IF Band-Pass Filter
BPF_IF1 = designfilt('bandpassiir', ...
                     'FilterOrder', 6, ... % Example filter order
                     'HalfPowerFrequency1', f_IF - ifbw1/2, ...
                     'HalfPowerFrequency2', f_IF + ifbw1/2, ...
                     'SampleRate', fsnew);
% Apply IF Band-Pass Filter to the first mixed signal
IF_filtered_signal1 = filter(BPF_IF1, ifsignal1);
% Plot IF first signal after band-pass filtering in time domain
figure;
subplot(2, 1, 1);
plot(t, IF_filtered_signal1);
xlabel('Time (sec)');
ylabel('Amplitude');
title('IF-Time domain of first signal after BPF');
grid on;
% Plot IF first signal after filtering in frequency domain
IFspectrum1after = fftshift(fft(IF_filtered_signal1));
subplot(2, 1, 2);
plot(frequencies, abs(IFspectrum1after));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('IF-Frequency domain of first signal after BPF');
grid on;
% Mixer for the second filtered signal
ifcarrier2 = cos(2 * pi * wc2 * t);
IF_signal2 = filteredsignal2 .* ifcarrier2;%shift to IF frequency
% Plot IF Signal 2 before band-pass filtering in time domain
figure;
subplot(2, 1, 1);
plot(t, IF_signal2);
xlabel('Time (sec)');
ylabel('Amplitude');
title('IF-Time domain of second signal before BPF');
grid on;
% Plot IF Signal 2 before band-pass filtering in frequency domain
IF_spectrum2_before = fftshift(fft(IF_signal2));
subplot(2, 1, 2);
plot(frequencies, abs(IF_spectrum2_before));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('IF-Frequency domain of second signal before BPF');
grid on;
% Define parameters for the second IF Band-Pass Filter
ifbw2 = 20e3; % Bandwidth of the IF filter in Hz
% Design the IF Band-Pass Filter for the second signal
BPF_IF2 = designfilt('bandpassiir', ...
                     'FilterOrder', 6, ... % Example filter order
                     'HalfPowerFrequency1', f_IF - ifbw2/2, ...
```

```
                     'HalfPowerFrequency2', f_IF + ifbw2/2, ...
                     'SampleRate', fsnew);
% Apply IF Band-Pass Filter to the second mixed signal
IF_filteredsignal2 = filter(BPF_IF2, IF_signal2);
% Plot IF second Signal after band-pass filtering in time domain
figure;
subplot(2, 1, 1);
plot(t, IF_filteredsignal2);
xlabel('Time (sec)');
ylabel('Amplitude');
title('IF-Time domain of second signal after BPF');
grid on;
% Plot IF second Signal after band-pass filtering in frequency domain
IF_spectrum2_after = fftshift(fft(IF_filteredsignal2));
subplot(2, 1, 2);
plot(frequencies, abs(IF_spectrum2_after));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('IF-Frequency domain of second signal after BPF');
grid on;
%% Baseband Detection
% Define Intermediate Frequency (IF) for downconversion
f_IF = 25e3; % Example IF frequency for the downconversion
% Mixer for baseband detection for first signal
baseband_carrier1 = cos(2 * pi * (f_IF) * t); % Carrier for Signal 1
baseband_signal1 = IF_filtered_signal1 .* baseband_carrier1; % Mixing
to baseband
% Low-Pass Filter for first signal
LPF1 = designfilt('lowpassiir', ...
                  'filterorder', 8, ...
                  'HalfPowerFrequency', fs / 2, ... % Nyquist
frequency of original sampling rate
                  'SampleRate', fsnew);
% Apply LPF to extract baseband signal & normalize the first signal
baseband_signal1_filtered = filter(LPF1, baseband_signal1);
baseband_signal1_filtered = baseband_signal1_filtered /
max(abs(baseband_signal1_filtered));
% Mixer for Baseband Detection for second signal
baseband_carrier2 = cos(2 * pi * (f_IF) * t); % Carrier for second
Signal
baseband_signal2 = IF_filteredsignal2 .* baseband_carrier2; % Mixing
to baseband
% Low-Pass Filter for second Signal
LPF2 = designfilt('lowpassiir', ...
                  'filterorder', 8, ...
                  'HalfPowerFrequency', fs / 2, ... % Nyquist
frequency of original sampling rate
                  'SampleRate', fsnew);
% Apply LPF to extract baseband signal & normalize the first signal
baseband_signal2_filtered = filter(LPF2, baseband_signal2);
baseband_signal2_filtered = baseband_signal2_filtered /
max(abs(baseband_signal2_filtered));
% Plot first signal in Baseband in Time Domain
figure;
subplot(2, 1, 1);
x = (0:length(baseband_signal1_filtered)-1)' / fs; % Time vector for
original sampling rate
```

```
plot(x, baseband_signal1_filtered);
xlabel('Time (sec)');
ylabel('Amplitude');
title('First signal in baseband in time domain');
grid on;
% Plot first signal in Baseband in Frequency Domain
baseband_FDomain1 = fftshift(fft(baseband_signal1_filtered));
freq_base1 = linspace(-fs/2, fs/2, length(baseband_signal1_filtered))
/ 1e3; % in kHz
subplot(2, 1, 2);
plot(freq_base1, abs(baseband_FDomain1));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('First signal in baseband in frequency domain');
grid on;
% Plot second signal in Baseband in Time Domain
figure;
subplot(2, 1, 1);
plot(x, baseband_signal2_filtered);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Second signal in baseband in time domain');
grid on;
% Plot Second signal in Baseband in Frequency Domain
baseband_FDomain2 = fftshift(fft(baseband_signal2_filtered));
freq_base2 = linspace(-fs/2, fs/2, length(baseband_signal2_filtered))
/ 1e3; % in kHz
subplot(2, 1, 2);
plot(freq_base2, abs(baseband_FDomain2));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('Second signal in baseband in frequency domain');
grid on;
% Define Downsampling Factor
down_sampling_factor = fsmultiplier; % Ratio of original to new
sampling frequency
% Anti-Aliasing LPF Design
anti_aliasing_filter = designfilt('lowpassiir', ...
                             'Filterorder', 8, ...
                             'HalfPowerFrequency', fs / 2, ... %
Nyquist frequency of original sampling rate
                             'SampleRate', fsnew);
% Downsample Signals
anti_aliased_signal1 = filter(anti_aliasing_filter,
baseband_signal1_filtered); % Apply anti-aliasing LPF
anti_aliased_signal2 = filter(anti_aliasing_filter,
baseband_signal2_filtered); % Apply anti-aliasing LPF
down_sampled_signal1 = downsample(anti_aliased_signal1,
down_sampling_factor);% Downsample
down_sampled_signal2 = downsample(anti_aliased_signal2,
down_sampling_factor);% Downsample
% Time Vector for Downsampled Signals
t_down_sampled = (0:length(down_sampled_signal1)-1)' / fs; % Time
vector for downsampled signals
fs_down_sampled = 1 / t_down_sampled;
% Plot first signal after downsampling in Time Domain
figure;
```

```
subplot(2, 1, 1);
plot(t_down_sampled, down_sampled_signal1);
xlabel('Time (sec)');
ylabel('Amplitude');
title('First signal after downsampling in Time Domain');
grid on;
% Plot first signal after downsampling in frequency Domain
down_sampled_FDomain1 = fftshift(fft(down_sampled_signal1));
frequ_down_sampling1 = linspace(-fs/2, fs/2,
length(down_sampled_signal1)) / 1e3; % in kHz
subplot(2, 1, 2);
plot(frequ_down_sampling1, abs(down_sampled_FDomain1));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('First signal after downsampling in frequency domain');
grid on;
% Plot second signal after downsampling in Time Domain
figure;
subplot(2, 1, 1);
plot(t_down_sampled, down_sampled_signal2);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Second signal after downsampling in time domain');
grid on;
% Plot second signal after downsampling in frequency Domain
down_sampled_FDomain2 = fftshift(fft(down_sampled_signal2));
freq_down_sampling2 = linspace(-fs/2, fs/2,
length(down_sampled_signal2)) / 1e3; % in kHz
subplot(2, 1, 2);
plot(freq_down_sampling2, abs(down_sampled_FDomain2));
xlabel('Frequency (kHz)');
ylabel('Magnitude');
title('Second signal after downsampling in frequency domain');
grid on;
% Comparison between original and recieved signals
sound(mono1, fs);% Play the first original signal
pause(20);% Wait 20 seconds to let first signal to finish
sound(down_sampled_signal1,fs);% Play the first signal after
downsampling
pause(20);% Wait 20 seconds to let first dwnsampled signal to finish
sound(mono2, fs);% Play the second original signal
pause(20);% Wait 20 seconds to let second original signal to finish
sound(down_sampled_signal2, fs);% Play the second signal after
downsampling
```