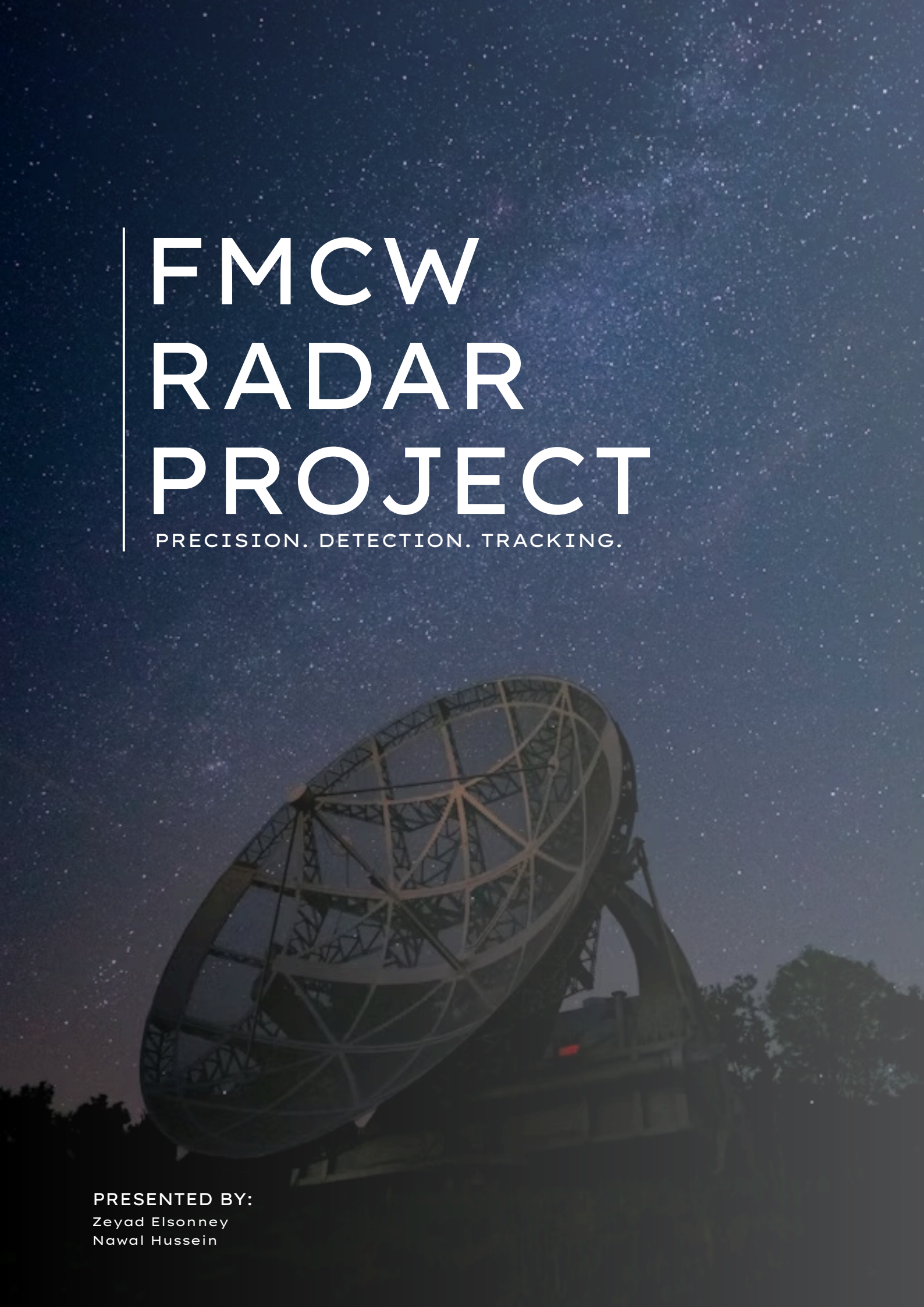


FMCW RADAR PROJECT

PRECISION. DETECTION. TRACKING.

PRESENTED BY:

Zeyad Elsonney

Nawal Hussein

TEAM MEMBERS

Name	Code
Zeyad Mohammed Abdullah Abdulfatah	91240328
Nawal Hussein Fawzy Hussein	91240819

TABLE OF CONTENTS

1. Introduction

Frequency-Modulated Continuous-Wave (FMCW) radar is a high-resolution sensing technology that transmits a continuous, frequency-sweeping signal—often called a "chirp"—to precisely measure the range, velocity, and angle of objects. Unlike traditional pulsed radar, FMCW systems operate continuously, making them ideal for short-to-medium range applications such as Advanced Driver Assistance Systems (ADAS), drone navigation, and industrial sensing. These systems frequently utilize millimeter-wave (mmWave) frequencies (e.g., 76–81 GHz), where the short wavelengths (approx. 4 mm) allow for compact hardware components and exceptional accuracy, capable of detecting sub-millimeter movements. Modern implementations have evolved from bulky discrete setups to highly integrated CMOS-based solutions that combine RF components, ADCs, and digital processors into single efficient chips, significantly reducing cost and power consumption while enhancing signal processing capabilities.

2. FMCW Working Principle

2.1 Fundamental Concept

Frequency Modulated Continuous Wave (FMCW) scientific base is that it transmits a signal continuously rather than in short bursts. To measure distance, the radar changes (modulates) the frequency of this signal over time.
The most common modulation scheme, and the one utilized in this project, is the Linear Frequency Modulation (LFM) or "Sawtooth" pattern. The radar generates a "chirp"—a sinusoid whose frequency increases linearly from a start frequency to a stop frequency over a defined duration.
When this chirp is transmitted, it travels to the target and is reflected back to the receiver. Because the signal travels at the speed of light, the received echo arrives with a time delay relative to the transmission. By the time the echo returns, the transmitter has already swept to a higher frequency.

2.2 Range Measurement (The Beat Frequency)

The core principle of FMCW detection lies in the "mixing" process. The receiver takes the instantaneous transmitted signal and multiplies (mixes) it with the received echo. Because the echo is a time-delayed version of the transmission, there is a constant frequency difference between the two signals.
This difference is known as the Beat Frequency (or Intermediate Frequency, IF).

- **Stationary Targets:** The beat frequency is directly proportional to the round-trip time delay. Since the time delay depends on the distance to the target, a higher beat frequency indicates a target that is further away.
- **Processing**: By performing a spectral analysis (Fast Fourier Transform) on this beat signal, the radar can identify distinct peaks. Each peak frequency corresponds to a specific target range.

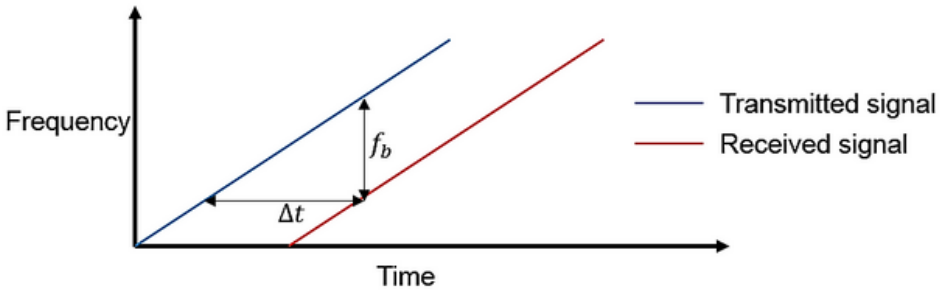


Fig 1: *The blue line represents the transmitted frequency, while the red line represents the received echo. The vertical gap between them is the beat frequency (fb)*

2. FMCW Working Principle

2.3 Velocity Measurement (Doppler Phase Shift)

While the beat frequency provides range information, FMCW radar extracts velocity information by analyzing the phase of the signal across multiple consecutive chirps.
When a target is moving, the distance changes slightly from one chirp to the next. This minute change in distance results in a phase rotation of the beat signal. By transmitting a sequence (or "frame") of chirps and performing a second FFT across the chirps (often called the Doppler FFT), the radar can measure this rate of phase change, which is directly proportional to the relative velocity of the target.

2.4 System Architecture

The hardware architecture typically consists of a waveform generator, a Voltage Controlled Oscillator (VCO) to create the high-frequency chirp, transmit and receive antennas, and a mixer. The output of the mixer is a low-frequency signal that is digitized by an Analog-to-Digital Converter (ADC) for processing.

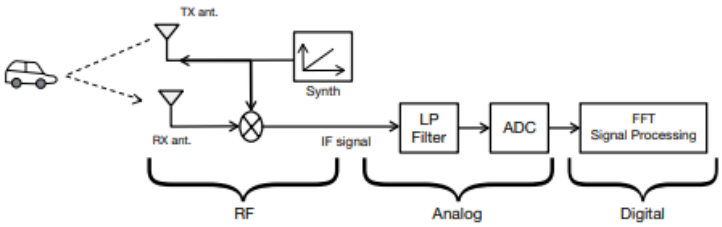


Fig 2: FMCW block diagram

2.5 The Sampling Process

Once the beat signal is isolated, it must be converted from an analog voltage into digital numbers for the computer (or DSP) to process. This is the job of the Analog-to-Digital Converter (ADC).
The ADC samples the beat signal at fixed intervals defined by the Sampling Frequency (fs).

- **The Limit:** The sampling rate imposes a physical limit on the system. According to the Nyquist criterion, the ADC must sample at least twice as fast as the highest frequency present in the beat signal.
- **Implication for Range:** Since distant targets produce higher beat frequencies, the sampling rate ultimately dictates the Maximum Range of the radar. If a target is too far, its beat frequency will exceed the ADC's capability and will be filtered out or aliased (appear as a ghost target at a closer range).

2. FMCW Working Principle

2.6 Fast Time vs. Slow Time

To measure both range and velocity simultaneously, FMCW radar organizes the sampled data into a two-dimensional matrix (often called the "Radar Data Cube"). This introduces two distinct time scales:

Fast Time (Intra-Chirp)
"Fast Time" refers to the time scale within a single chirp.
As the radar sweeps through one chirp, the ADC collects hundreds or thousands of samples (e.g., 1024 samples).
These samples are taken very rapidly (at the Sampling Frequency, fs).
Purpose: Processing data along the "Fast Time" axis (Range FFT) reveals the Range of the targets.

Slow Time (Inter-Chirp)
"Slow Time" refers to the time scale across multiple chirps.
The radar transmits a sequence of chirps (e.g., 128 chirps) to form a frame.
"Slow Time" tracks how the signal changes from the first chirp to the second, to the third, and so on. The time interval here is the Chirp Repetition Interval (Tc).
Purpose: Processing data along the "Slow Time" axis (Doppler FFT) reveals phase shifts caused by small movements of the target, which allows the calculation of Velocity.

Summary of Dimensions:
Row (Fast Time): Contains the beat frequency -> Range FFT.
Column (Slow Time): Contains the phase history -> Doppler FFT.

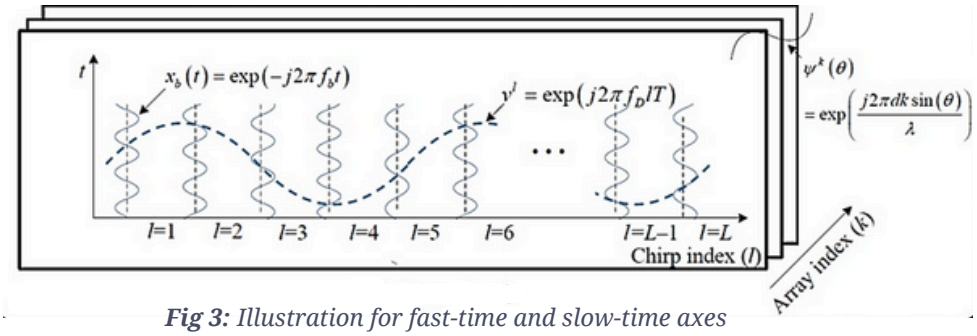


Fig 3: *Illustration for fast-time and slow-time axes*
(t-axis → Fast time)
(chirp index(l)→ Slow time)

2. FMCW Working Principle

2.7 Waveform Parameters

The performance of the FMCW radar is governed by several key parameters configured in the simulation. These parameters determine the radar's resolution, maximum range, and ability to detect moving objects.

- Carrier Frequency (fc): This is the operating frequency of the radar (e.g., 76.5 GHz). Higher carrier frequencies (millimeter-wave) allow for smaller antennas and better sensitivity to small movements compared to lower frequency radars.

- Bandwidth (BW): The bandwidth is the difference between the maximum and minimum frequency of the chirp. This is the most critical parameter for Range Resolution. A wider bandwidth allows the radar to distinguish between two targets that are close together.

- Chirp Duration (Tc): This is the time it takes to complete one full frequency sweep. The duration impacts the amount of energy on the target and influences the maximum unambiguous velocity. Shorter chirps allow for the detection of faster-moving objects but require faster sampling rates.

- Slope (s): The rate at which the frequency changes, defined by the Bandwidth divided by the Chirp Duration. The slope determines how the time delay is mapped to the beat frequency.

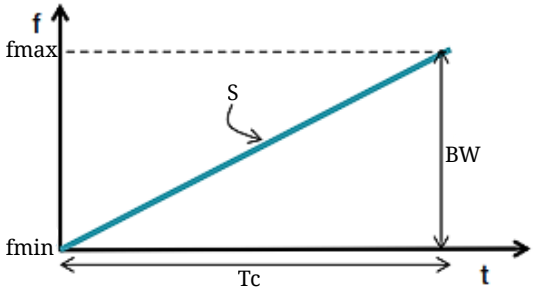


Fig 4: *The chirp is characterized by a start frequency (fmin), bandwidth (BW) and duration (Tc). The slope of the chirp (S) captures the rate of change of frequency*

3. Theory & Implementation

```
clear; clc; close all;
%% Constants
c = 3e8;            % Speed of light
fs = 2e9;           % Sampling frequency
fc = 76.5e9;        % Carrier frequency
BW = 1e9;           % Bandwidth
Tc = 3e-6;          % Chirp duration
N = 2048;           % Number of chirps
A_tx = 1;           % Amplitude
% FMCW Parameters
SNR_dB= 5;
s = BW/Tc;
fb_max = (s * 2 * 200) / c;   % important to satisfy nyquist
M = round(Tc*fs);             % No. of samples per chirp
R_max = c*fs*Tc/(4*BW);       % Max Range
v_max = c/(4*fc*Tc);          % Max Velocity
delta_v = c/(2*fc*(N*Tc));    % Velocity Resolution
delta_R = c/(2*BW);           % Range Resolution
fprintf('Range Res: %.2fm\n',delta_R);
fprintf('Max Range: %.2fm\n',R_max);
fprintf('Velocity Res: %.2fm/s\n', delta_v);
fprintf('Max Velocity: %.2fm/s\n',v_max);
```

3.1 Constants

The FMCW radar designed in the code works within the frequency range 76 GHz - 77 GHz, which means a bandwidth (**BW**) of 1 GHz. The carrier frequency (**fc**) must be 76.5 GHz to maximize the radar's performance (resolution) while strictly obeying legal regulations.

Tc: It is set to 5u sec to balance the required bandwidth, resolution. It also balances between the needed max; range and needed max; velocity

fs: is chosen such that we can capture the maximum expected beat frequency (f_b_max) with margin, adhering to the Nyquist Criterion (fs>=2f_b_max).
Also we can achieve a 750 meters range .

N: The count of identical chirps used in the slow-time domain for Doppler processing. Justification: A large N is required to achieve high Velocity Resolution (delta v), as it increases the total observation time (N*Tc). The value 2048 is chosen as we also have a very small Tch which decreases the resolution dramatically.

M: This is the length of the Fast-Time dimension (the range axis). It represents how many discrete data points are collected during the active transmission time of one chirp (Tc). It is critical because it dictates the size of the FFT in the range domain and though it is useful in the mixing process and calculating the true frequency bins.

3. Theory & Implementation

3.2 FMCW Parameters

Calculated parameters:

Range Resolution: 0.15 m
Maximum Range: 750m
Velocity Resolution: 0.19m/s
Maximum Velocity: 196m/s

$$* \ s = \frac{BW}{Tc} \qquad * \ M = Tc \times fs$$

$$* \ vmax = \frac{c}{4\, fc\, Tc} \qquad * \ Rmax = \frac{c\, fs\, Tc}{4\, BW}$$

$$* \ \Delta v = \frac{c}{2\, fc\, N\, Tc} \qquad * \ \Delta R = \frac{c}{2\, BW}$$

3.3 Signal Generation

```
% we should have a time vector to represent the samples on the time axis
% each chirp is of length=M, to represent it, we need to multiply it by Ts so
that we have 1Ts, 2Ts,...
% this represents the fast time axis, it is vertical so we take the transpose
of the samples matrix
t= (0:M-1)'/fs;
%Transmitted signal
% we use baseband of -0.5 to 0.5 GHz, as if we used 76 GHz to 77 GHz as it is,
it would require an impossible fs to achieve Nyquist rate
% fmin=-0.5GHz
Tx= A_tx*exp(2j*pi*(-BW*t/2 + s*t.^2/2));
% noise amp to be added to received signal
signal_power = mean(abs(Tx).^2);
noise_power = signal_power / 10^(-SNR_dB/10);
noise_amp = sqrt(noise_power / 2);
```

Time Vector and Signal Structure

The first line is the foundation for the digital signal by establishing the exact points in time where the continuous analog signal will be sampled.

This (M* N) matrix structure is the foundation of the processing grid, where the rows are Fast Time (M) and the columns are Slow Time (N).

Transmitted Pulse Definition

The transmitted signal equation $P = A_{tx} \exp\left(\pi j(-BWt + st^2)\right)$ is chosen to center the sweep at the radar's carrier frequency (fc). This ensures the instantaneous frequency sweeps linearly over the entire bandwidth BW, starting precisely at fc - BW/2 and ending at fc + BW/2. This symmetric design simplifies signal processing and is a standard configuration for baseband representation in FMCW systems.

3. Theory & Implementation

Noise Amplitude Calculation

To ensure the fidelity of the simulation, electronic noise is added such that the Signal-to-Noise Ratio (SNR) is precisely 10dB. This requires calculating the necessary noise amplitude (sigma) using the relationship:

$$SNR = \frac{Psignal}{Pnoise}$$

The equation first determines the signal power, divides it by the desired linear SNR ratio (10^1), and accounts for the complex nature of the noise by dividing the noise power budget by two. The square root then converts the resulting required noise power into the necessary standard deviation (sigma) used by the (randn) function to generate the complex additive noise.

```
% Targets
targets.R = [50; 180];
targets.v = [+100; -80];
% Received signal generation
% The received signal is only the trans. signal but shifted and delayed
% we need to find Rx for each chirp
mix= zeros(M,N); % allocate memory for mixing output
% this represesnts the slow time (rows) vs fast time (columns)
% each chirp has M samples, so column must have M rows to fit all the data
% we repeat this for N chirps, so we need N columns
for n= 1:N % we get Rx for each chirp
    % allocate memory for rx
    Rx= zeros(M,1); %reset rx for every new chirp
    % the sz of Rx is exactly the same as Tx (M)
for i= 1:length(targets.R) % loop on all targets
% 1st: get RTT (delay)
%we need to make account for moving targets too
    range = targets.R(i) + targets.v(i)*(n-1)*Tc;
    RTT = 2*range/c;
% 2nd: get doppler shift
fd= 2*targets.v(i)*fc/c;
dopp_shft= exp(2j*pi*fd*(n-1)*Tc);
% 3rd: we only want the overlap btn the Tx and Rx only
for k=1:length(t)
if t(k)>=RTT
Rx(k)= Rx(k) + A_tx*exp(2j*pi*(-BW*(t(k)-RTT)/2 + s*(t(k)-
RTT).^2/2))*dopp_shft;
else
        Rx(k) = 0;
    end
end
end
```

3. Theory & Implementation

In the part of (*t(k)>=RTT*) function, we are trying to calculate the range and velocity so we are trying to detect valid places for the overlapping between the echo and the transmitted signal which is the area at time (RTT:TC).

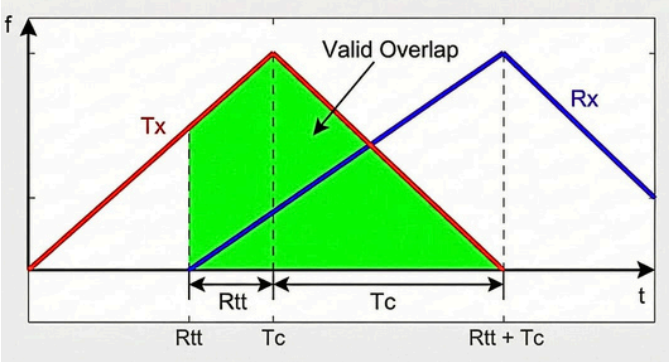


Fig 5: *Ilustrative graph for overlapping between Tx and Echo*

For the term (*targets.v(i)*(n-1)*Tc*), it is added just to process the rotation that has occurred in the beat signal and though it helps in processing the Doppler FFT. The term also calculates the cumulative physical distance the moving target covers between the start of the first chirp and the start of the current chirp n. Imagine we are inside the radar sitting still, repeatedly sending out chirps every Tc seconds. With each new chirp, any moving car in front of the radar has had time to travel a little closer or farther.

So for each chirp, we recalculate where the target is right now, figure out how long its echo will take to come back, generate the properly delayed and phase-shifted echo, and add it into the received signal for that chirp. Doing this accurately for all targets and all chirps creates a realistic raw data matrix that contains both range information (in the fast-time beat signal) and velocity information (in the slow-time phase progression) — exactly what the later 2D FFT (Range-Doppler processing) needs to separate and display the targets cleanly on the range-velocity map.

This small but crucial velocity-dependent range update is what turns a static simulation into a dynamic moving-target one, making your Doppler processing meaningful and realistic. Without it, all targets would appear stationary regardless of their velocity setting.

```
% 4th: add noise (SNR=10db)
Rx = Rx + noise_amp * (randn(M,1) + 1j*randn(M,1));
% 5th: mixing
% Take the results of the nth chirp and put them into the nth column
mix(:, n) = Tx.* conj(Rx);
end
% we need a single chirp of mixer output and Rx for display
mix_chirp= mix(:, 1);
Rx_chirp= Rx(:, 1);
```

Additive White Gaussian Noise (AWGN) is generated using *randn* and scaled by *noise_amp* (derived from the 10 dB SNR target) and then added to the received signal *Rx*, then this noise-contaminated received signal, *Rx*, for the very first chirp (n=1) is captured and saved as *Rx_chirp* for later visual analysis. Then beat signal mixing occurs by multiplying the transmitted reference signal (*Tx*) with the complex conjugate of the noisy received signal (*conj(Rx)*). The resulting beat signal, which contains the low-frequency range and Doppler information, is stored as the n^{th} column in the main Range-Doppler processing matrix, *mix(:, n)*, which is crucial for slow time analysis then (laterally it represents the n^{th} chirp).

3.4 Plot Generation

```
% Amplitude vs time of transmitted signal
figure(1);
plot(t,real(Tx));
title('1. Transmitted Signal (Amplitude)'); grid on;
xlabel('Time'); ylabel('Amplitude');

% Frequency vs time of transmitted signal

freq=diff(unwrap(angle(Tx))) * fs/(2*pi); % change in phase=frequency
% we use unwrap as angle ranges from -pi to pi only but we want to see the
whole change
% we mult. it by fs to get change per second instead of change per sample
% diff makes the size of freq matrix less than the size of t matrix by 1
% we assume that the last element of freq matrix is like its end
freq=[freq;freq(end)];
freq_rep= repmat(freq,5); % display 5 sawtooth chirps
t_freq=(0:length(freq_rep)-1)/ fs; % x-axis must be the same size as y-axis

figure(2);
plot(t_freq,freq_rep);
title('1. Transmitted Signal (Frequency)'); grid on;
xlabel('Time'); ylabel('Frequency');
```

3. Theory & Implementation

```
% Amplitude vs time of received Signal
figure(3);
plot(t, real(Rx_chirp));
title('3. Received Signal'); grid on;
xlabel('Time'); ylabel('Amplitude');
```

This part of code visualizes a transmitted signal (*Tx*) by plotting its time-domain amplitude and its instantaneous frequency profile. *Figure 1* displays the real part of the signal's amplitude over time, while *Figure 2* reveals the frequency modulation, derived by differentiating the signal's unwrapped phase and scaling it by the sampling rate (*fs*) to convert radians per sample into Hertz. To properly visualize the periodic nature of the signal (identified in the comments as "sawtooth chirps"), the calculated frequency vector is padded to match the time vector's length and repeated five times before plotting. *Figure 3* visualizes the time-domain amplitude of the received signal (*Rx_chirp*). By plotting the real component of the complex signal against the time vector t, it enables a direct comparison with the transmitted signal to identify effects such as attenuation, time delays, or added noise.

```
% Amplitude vs time of mixer Output
figure(4);
plot(t, real(mix_chirp));
title('4. Mixer Out'); grid on;
xlabel('Time'); ylabel('Amplitude');
```

This part of code visualizes the Beat Signal which is the result of electronically mixing the transmitted chirp with the received echo. By plotting the real component of *mix_chirp* against time in microseconds, the graph reveals a waveform whose frequency is proportional to the target's range—effectively "subtracting" the transmitted frequency from the received frequency. This output typically appears as a steady, lower-frequency sine wave, representing the demodulated data ready for frequency analysis (FFT).

3. Theory & Implementation

3.5 Range Detection (Fast Fourier Transform Processing)

```
% Range-FFT Plot
% to decrease the error, we apply padding which is adding zeros to the end of
the fft matrix, to get more resolution points
M_pad = M * 4; % as if we zoomed the fft to 4x to have better resolution and
smaller bins
mix_fft= fft(mix(:,1), M_pad); % apply fft on the 1st chirp with adding zeros
k= 0:(M_pad/2)-1; % length of k is the length of time signal (no.of samples)
% we divide by 2 to take the +ve part only
% we need to define the frequency bin/ resolution
freq_bin= fs/M_pad;
freq_axis= k*freq_bin; % frequency axis: frequency that corresponds to each k
range_axis= (freq_axis * c)/(2*s); % convert frequency axis to range axis
```

This converts the mixed "beat" signal—where the frequency difference between transmitted and received chirps is proportional to delay—into a usable distance profile. To improve the accuracy of peak detection, the code first applies zero-padding (extending the signal length M by a factor of 4), which interpolates the spectrum to produce a smoother curve and finer "bins" in the frequency domain. After computing the FFT and isolating the positive half of the spectrum, the code calculates the frequency value for each bin and transforms this axis into specific distances (Range) using the radar equation; this effectively maps every spectral peak directly to a target's location in meters.


```
% find the peaks
[peak, loc_index] = findpeaks(abs(mix_fft(1:M_pad/2)),'MinPeakHeight', 0.3 *
max(abs(mix_fft(1:M_pad/2))));
peak_ranges = range_axis(loc_index); % we want to see the loc. index
corresponds to how many meters
% we lookup for the (loc. index)^th item in the range_axis
figure(5);
plot(range_axis, abs(mix_fft(1:M_pad/2)));
 % we take the abs bc we care about the mag only above, we took one chirp
(column) and plotted the its full range profile
hold on;
plot(peak_ranges, peak, 'rv', 'MarkerFaceColor','r'); % puts a red reversed
triangle over the peaks
title('5. Range-FFT'); grid on;
xlabel('Range (m)'); ylabel('Amplitude');
```

```
%print detected targets
fprintf('-------------------------------------------------\n');
fprintf('Detected Targets Range | Original Targets Range\n');
for idx = 1:length(loc_index)
fprintf('%-7.2fm                | %-7.2fm\n', peak_ranges(idx),targets.R(idx));
end
```

This section automates the extraction of valid target distances by filtering the Range-FFT spectrum for significant peaks while rejecting background noise. The code uses the *findpeaks* function to scan the signal's magnitude for local maxima, applying a dynamic threshold (30% of the highest peak) to ensure only strong reflections are registered as targets. The indices of these valid peaks are then used to look up values in the *range_axis* vector, effectively converting abstract FFT bin numbers into physical distances in meters. To verify accuracy, the code visualizes these detections with red markers in *Figure 5* and prints a formatted table comparing the calculated ranges against the known ground-truth locations (*targets.R*).

3.6 Range-Doppler Map & Velocity

```
%% Range-Doppler Map & Velocity
% to get the delta phase shift, we perform 2D fft across the slow & fast time
axes
fft_2D= fft2(mix,M_pad,N); %computes fft over the rows and then over the
columns
% the problem here is that fft2 sets the zero at the first bin so we use
% fftshift to set the zero at the middle again
fft_2D_shifted= abs(fftshift(fft_2D,2));
fft_2D_shifted= fft_2D_shifted(1:M_pad/2, :);
% we need to generate the velocity axis to plot on it (we already generated the
range axis)
% we can use the velocity resolution we calculated at the beginning
vel_axis = linspace(v_max, -v_max, N); %create axis from +VMax to -VMax as the
fftshift shifts the polarity
% Range-Doppler Map
figure(6);
imagesc(vel_axis, range_axis, 20*log10(fft_2D_shifted));
% we used db scale instead of linear to give it more power so that the targets
appear clearer in the plot
axis xy;                % Put range 0 at the bottom
colormap hot;
xlabel('Velocity (m/s)');
ylabel('Range (m)');
title('6. Range-Doppler Map');
```

3. Theory & Implementation

This generates a Range-Doppler Map, a 2D visualization that simultaneously displays the distance and velocity of targets. It achieves this by performing a 2D FFT (*fft2*) which processes the signal across "fast time" (within a chirp) to determine range and "slow time" (across consecutive chirps) to measure the phase shifts caused by the Doppler effect. The *fftshift* command is applied to center the zero-velocity component, creating a symmetric view that distinguishes between approaching and receding targets. To make the targets visually distinct against background noise, the amplitude is converted to a logarithmic decibel scale (*20*log10*) and plotted as a heatmap (*Figure 6*), where the color intensity represents the signal strength at every combination of range and speed.

```
% Doppler Spectrum
% we need to take a slice of the 2D map of each detected target
figure(7);
xlabel('Velocity (m/s)');
ylabel('Amplitude (m)');
title('7. Doppler Spectrum');
hold on;
% print detected targets
fprintf('-----------------------------------------------------\n');
fprintf('Detected Targets Velocity | Original Targets Velocity\n');
for i=1:length(loc_index)
    dopp_spect = fft_2D_shifted(loc_index(i), :);
    plot(vel_axis, dopp_spect);
    % Find peaks
    [v_peak, v_index] = max(dopp_spect);
    est_v= vel_axis(v_index);
    if est_v > 0, status = 'Approaching';
    elseif est_v < 0, status = 'Receding';
    else, status = 'Stationary'; end
    fprintf('%+8.2f m/s (%-10s)| %+8.2f m/s\n', est_v, status, targets.v(i));
     plot(est_v, v_peak, 'rv', 'MarkerFaceColor','r'); % puts a red reversed
triangle over the peaks
end
hold off;
```

This calculates the precise velocity for each previously detected target by extracting a 1D "slice" from the 2D Range-Doppler map at the specific range bin where the target was located. By finding the maximum peak within this extracted Doppler spectrum, the code determines the target's speed and direction (approaching vs. receding), visualizing the velocity profile in Figure 7 and printing a verification table that compares these estimates against the ground truth.

3. Theory & Implementation

3.7 The Full Code

```
clear; clc; close all;
%% Constants
c = 3e8;            % Speed of light
fs = 2e9;           % Sampling frequency
fc = 76.5e9;        % Carrier frequency
BW = 1e9;           % Bandwidth
Tc = 3e-6;          % Chirp duration
N = 2048;           % Number of chirps
A_tx = 1;           % Amplitude
% FMCW Parameters

SNR_dB= 10;
s = BW/Tc;
M = round(Tc*fs);             % No. of samples per chirp
fb_max = (s * 2 * 200) / c;   % Important to satisfy nyquist
R_max = c*fs*Tc/(4*BW);       % Max Range
v_max = c/(4*fc*Tc);          % Max Velocity
delta_v = c/(2*fc*(N*Tc));    % Velocity Resolution
delta_R = c/(2*BW);           % Range Resolution

fprintf('Max fbeat: %.2fe6 Hz\n',fb_max/1e6);
fprintf('Range Res: %.2fm\n',delta_R);
fprintf('Max Range: %.2fm\n',R_max);
fprintf('Velocity Res: %.2fm/s\n', delta_v);
fprintf('Max Velocity: %.2fm/s\n',v_max);
% we should have a time vector to represent the samples on the time axis
% each chirp is of length=M, to represent it, we need to multiply it by Ts so
that we have 1Ts, 2Ts,...
% this represents the fast time axis, it is vertical so we take the transpose
of the samples matrix
t= (0:M-1)'/fs;
%Transmitted signal
% we use baseband of -0.5 to 0.5 GHz, as if we used 76 GHz to 77 GHz as it is,
it would require an impossible fs to achieve Nyquist rate
% fmin=-0.5GHz
Tx= A_tx*exp(2j*pi*(-BW*t/2 + s*t.^2/2));
% noise amp to be added to received signal
signal_power = mean(abs(Tx).^2);
noise_power = signal_power / 10^(SNR_dB/10);
noise_amp = sqrt(noise_power / 2);
% Amplitude vs time of transmitted signal
figure(1);
plot(t,real(Tx));
title('1. Transmitted Signal (Amplitude)'); grid on;
xlabel('Time'); ylabel('Amplitude');
% Frequency vs time of transmitted signal
freq=diff(unwrap(angle(Tx))) * fs/(2*pi); % change in phase=frequency
% we use unwrap as angle ranges from -pi to pi only but we want to see the
whole change
% we mult. it by fs to get change per second instead of change per sample
% diff makes the size of freq matrix less than the size of t matrix by 1
% we assume that the last element of freq matrix is like its end
freq=[freq;freq(end)];
freq_rep= repmat(freq,5); % display 5 sawtooth chirps
t_freq=(0:length(freq_rep)-1)/ fs; % x-axis must be the same size as y-axis
```

3. Theory & Implementation

3.7 The Full Code

```
figure(2);
plot(t_freq,freq_rep);
title('1. Transmitted Signal (Frequency)'); grid on;
xlabel('Time'); ylabel('Frequency');
% Targets
targets.R = [50; 180];
targets.v = [+100; -80];
% Received signal generation
% The received signal is only the trans. signal but shifted and delayed
% we need to find Rx for each chirp
mix= zeros(M,N); % allocate memory for mixing output
% this represesnts the slow time (rows) vs fast time (columns)
% each chirp has M samples, so column must have M rows to fit all the data
% we repeat this for N chirps, so we need N columns
for n= 1:N % we get Rx for each chirp
    % allocate memory for rx
    Rx= zeros(M,1); %reset rx for every new chirp
    % the sz of Rx is exactly the same as Tx (M)
for i= 1:length(targets.R) % loop on all targets
% 1st: get RTT (delay)
%we need to make account for moving targets too
    range = targets.R(i) + targets.v(i)*(n-1)*Tc;
    RTT = 2*range/c;
% 2nd: get doppler shift
fd= 2*targets.v(i)*fc/c;
dopp_shft= exp(2j*pi*fd*(n-1)*Tc);
% 3rd: we only want the overlap btn the Tx and Rx only
for k=1:length(t)
if t(k)>=RTT
Rx(k)= Rx(k) + A_tx*exp(2j*pi*(-BW*(t(k)-RTT)/2 + s*(t(k)-
RTT).^2/2))*dopp_shft;
else
        Rx(k) = 0;
    end
end
end
% 4th: add noise (SNR=10db)
Rx = Rx + noise_amp * (randn(M,1) + 1j*randn(M,1));
% 5th: mixing
% Take the results of the nth chirp and put them into the nth column
mix(:, n) = Tx.* conj(Rx);
end
% we need a single chirp of mixer output and Rx for display
mix_chirp= mix(:, 1);
Rx_chirp= Rx(:, 1);
```

3.7 The Full Code

```
% Amplitude vs time of received Signal
figure(3);
plot(t, real(Rx_chirp));
title('3. Received Signal'); grid on;
xlabel('Time'); ylabel('Amplitude');
% Amplitude vs time of mixer Output
figure(4);
plot(t, real(mix_chirp));
title('4. Mixer Out'); grid on;
xlabel('Time'); ylabel('Amplitude');

% Range-FFT Plot
% to decrease the error, we apply padding which is adding zeros to the end of
the fft matrix, to get more resolution points
M_pad = M * 4; % as if we zoomed the fft to 4x to have better resolution and
smaller bins
mix_fft= fft(mix(:,1), M_pad); % apply fft on the 1st chirp with adding zeros
k= 0:(M_pad/2)-1; % length of k is the length of time signal (no.of samples)
% we divide by 2 to take the +ve part only
% we need to define the frequency bin/ resolution
freq_bin= fs/M_pad;
freq_axis= k*freq_bin; % frequency axis: frequency that corresponds to each k
range_axis= (freq_axis * c)/(2*s); % convert frequency axis to range axis
% find the peaks
[peak, loc_index] = findpeaks(abs(mix_fft(1:M_pad/2)),'MinPeakHeight', 0.3 *
max(abs(mix_fft(1:M_pad/2))));
peak_ranges = range_axis(loc_index); % we want to see the loc. index
corresponds to how many meters
% we lookup for the (loc. index)^th item in the range_axis

figure(5);
plot(range_axis, abs(mix_fft(1:M_pad/2)));
 % we take the abs bc we care about the mag only above, we took one chirp
(column) and plotted the its full range profile
hold on;
plot(peak_ranges, peak, 'rv', 'MarkerFaceColor','r'); % puts a red reversed
triangle over the peaks
title('5. Range-FFT'); grid on;
xlabel('Range (m)'); ylabel('Amplitude');
%print detected targets
fprintf('-------------------------------------------------\n');
fprintf('Detected Targets Range | Original Targets Range\n');
for idx = 1:length(loc_index)
fprintf('%-7.2fm                | %-7.2fm\n', peak_ranges(idx),targets.R(idx));
end
```

3. Theory & Implementation

3.7 The Full Code

```
%% Range-Doppler Map & Velocity
% to get the delta phase shift, we perform 2D fft across the slow & fast time
axes
fft_2D= fft2(mix,M_pad,N); %computes fft over the rows and then over the
columns
% the problem here is that the zero is at the first by default as a result
% of fft so we use fft shift to put the zero at the middle
% so we need to shift the fft
fft_2D_shifted= abs(fftshift(fft_2D,2));
fft_2D_shifted= fft_2D_shifted(1:M_pad/2, :);
% we need to generate the velocity axis to plot on it (we already generated the
range axis)
% we can use the velocity resolution we calculated at the beginning
vel_axis = linspace(v_max, -v_max, N); %create axis from +VMax to -VMax as the
fftshift shifts the polarity
% Range-Doppler Map
figure(6);
imagesc(vel_axis, range_axis, 20*log10(fft_2D_shifted));
% we used db scale instead of linear to give it more power so that the targets
appear clearer in the plot
axis xy;                 % Put range 0 at the bottom
colormap hot;
xlabel('Velocity (m/s)');
ylabel('Range (m)');
title('6. Range-Doppler Map');
% Doppler Spectrum
% we need to take a slice of the 2D map of each detected target
figure(7);
xlabel('Velocity (m/s)');
ylabel('Amplitude (m)');
title('7. Doppler Spectrum');
hold on;
% print detected targets
fprintf('-------------------------------------------------\n');
fprintf('Detected Targets Velocity | Original Targets Velocity\n');
for i=1:length(loc_index)
    dopp_spect = fft_2D_shifted(loc_index(i), :);
    plot(vel_axis, dopp_spect);
    % Find peaks
    [v_peak, v_index] = max(dopp_spect);
    est_v= vel_axis(v_index);
    if est_v > 0, status = 'Approaching';
    elseif est_v < 0, status = 'Receding';
    else, status = 'Stationary'; end
    fprintf('%+8.2f m/s (%-10s)| %+8.2f m/s\n', est_v, status, targets.v(i));
     plot(est_v, v_peak, 'rv', 'MarkerFaceColor','r'); % puts a red reversed
triangle over the peaks
end
```

4. Design Trade-offs & Parameter Optimization

Tch = PRI	PRI > Tch
This maximizes the P_avg for a given P_peak. This is called maximizing the Processing Gain. Every millisecond of the frame is spent collecting signal, this leads to increasing SNR values as power increased, which lowers the detection threshold for weak targets (low Radar Cross Section or low targets cross section)	On the other hand, we introduce Idle Time T_idle. If we keep the peak transmit power the same, but increase the PRI, the average power drops and though it results in a lower SNR.
Right here most cases no problem occurs as already PRI=Tch, we are utilizing every available microsecond of the interval for the frequency sweep. By stretching the chirp duration to fill the entire PRI, so we are minimizing the slope S for that specific v_max requirement.	For very fast cars we need a lower PRI as maximum unambigous velocity= c/4*fc*PRI and hence a lower Tch is needed as PRI decreased to detect them , However If we keep the same Bandwidth to maintain range resolution, the Slope (S = B / T_ch) must increase which leads to a very high S, which pushes fbeat beyond the Bandwidth of the ADC.

4. Design Trade-offs & Parameter Optimization

Tradeoff	What We Chose
Pulse Doppler Radar VS FMCW	We chose **FMCW**. Pulse Doppler radar relies on high-power, intermittent bursts that create a mandatory "blind range" where the receiver is off, making it ineffective for detecting nearby objects and requiring complex hardware to handle high peak voltages. In contrast, FMCW transmits continuously, eliminating this blind zone to allow safe detection at very close range while achieving superior resolution through simple frequency sweeps rather than difficult-to-generate nanosecond pulses. This architecture enables FMCW systems to operate with significantly facilitating compact, low-cost, and highly accurate solid-state designs.
Right value for Tch? Vmax VS Rmax	We chose **Tch = 3 us** and hence **Vmax** wins. The selection is dictated by the fundamental inverse relationship between maximum velocity (vmax) and maximum range (Rmax) in FMCW radar systems. Because vmax is determined by the chirp repetition rate (vmax = c / 4TcFc), a very short Tc was chosen to push the velocity ambiguity limit to 326.8 m/s, ensuring a fast target could be detected without aliasing. However, shortening Tc steepens the frequency slope, which drives the beat frequencies of distant objects much higher; if the sampling rate was limited, this would drastically cut the maximum range (Rmax).
Right value for N? Velocity resolution VS Latency	We chose **N=2048** to maximize Velocity Resolution, allowing the radar to finely distinguish between targets moving at similar speeds and detect weaker reflections through longer integration. However, this comes at the cost of Latency and Computational Load: the radar must wait to collect all 2048 chirps before processing can begin —increasing the "Frame Time"—and the subsequent 2D-FFT requires significantly more memory and processing power.

5. Simulation Results and Observations

5.1 Results of the Main Code (SNR=10 db)

- Command Window Output and Observation:

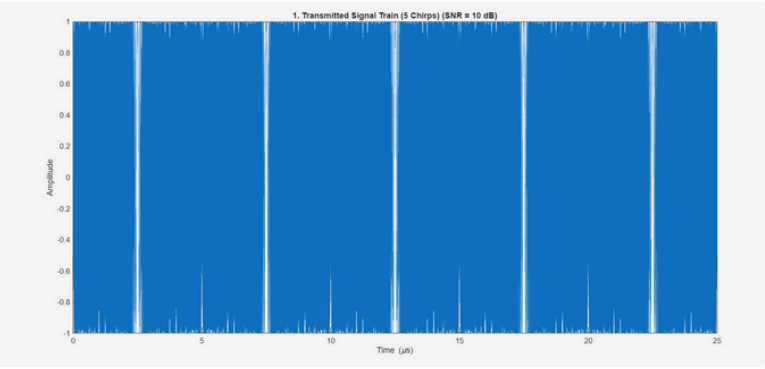
```
Command Window
Max fbeat: 266.67e6 Hz
Range Res: 0.15m
Max Range: 750.00m
Velocity Res: 0.19m/s
Max Velocity: 196.08m/s

Detected Targets Range | Original Targets Range
50.02  m               | 50.00  m
180.04 m               | 180.00 m
-------------------------------------------------
Detected Targets Velocity | Original Targets Velocity
 +100.15 m/s (Approaching)|  +100.00 m/s
  -80.04 m/s (Receding  )|   -80.00 m/s
>>
```

The radar system successfully detected two targets with high precision, demonstrating accuracy well within its theoretical resolution limits of 0.15 m for range and 0.19 m/s for velocity. The errors were negligible, with the largest deviation being just 0.15 m/s for the first target's velocity—an error smaller than the resolution bin size itself, confirming the deviation is due to digital quantization (FFT binning) rather than system malfunction. Target 1 was correctly identified at 50.02 m (0.02 m error) approaching at 100.15 m/s, while Target 2 was pinpointed exactly at 180.04 m receding at 80.04 m/s (0.04 error), validating the effectiveness of the Range-Doppler processing chain.

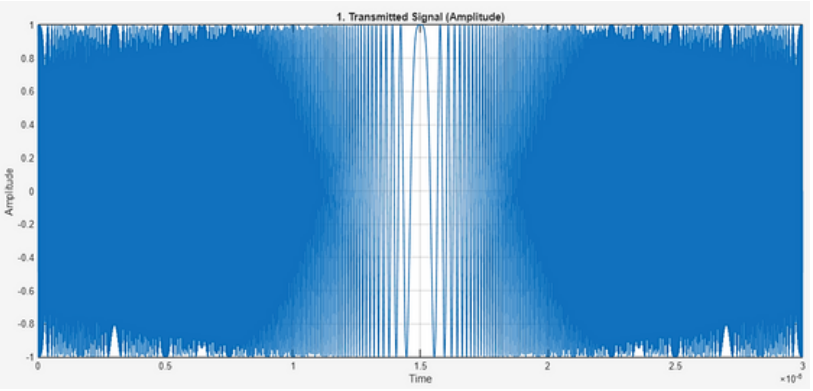
5. Simulation Results and Observations

5.1 Results of the Main Code (SNR=10 db)

- Transmitted Signal (Amplitude vs Time) Plot and Observation:



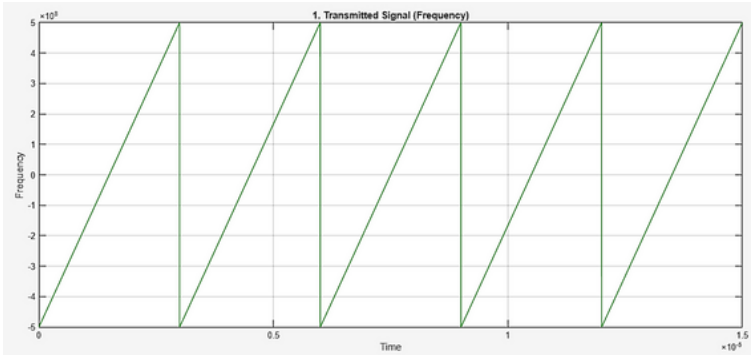
Impulse Train



1-period

The "bowtie" shape observed in the plot is a visual artifact caused by the rendering of the frequency sweep on a digital screen, not a physical variation in the signal's power. Although the code generates a pulse with a strictly constant amplitude of 1, the frequency linearly sweeps from -BW/2 to +BW/2, passing through zero Hz (DC) in the exact center of the chirp. At the start and end of the pulse, the high-frequency oscillations are packed so densely that they visually merge into a solid block of color; however, as the frequency approaches zero in the middle, the oscillations slow down enough to be resolved as individual lines, creating the optical illusion of a pinched waist despite the energy remaining constant throughout.

- Transmitted Signal (Frequency vs Time) Plot and Observation:

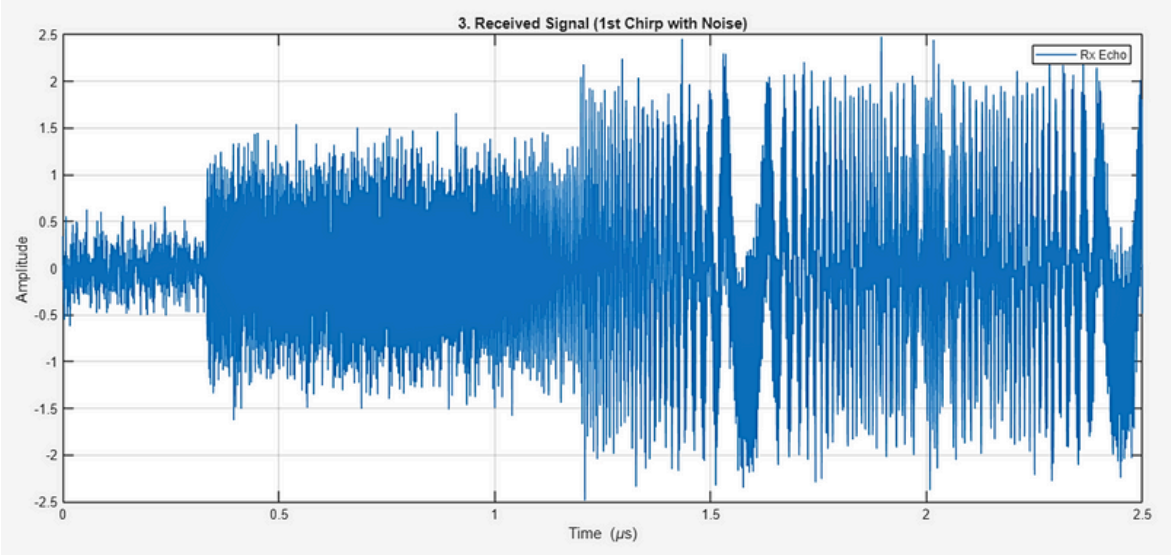


The signal frequency ramps up linearly from a minimum of -0.5 GHz to a maximum of +0.5 GHz, establishing a total operating bandwidth of 1 GHz. This frequency sweep repeats at a regular interval, with each full chirp cycle lasting exactly 3 us. The slope of the signal (s) is 333 THz/s.

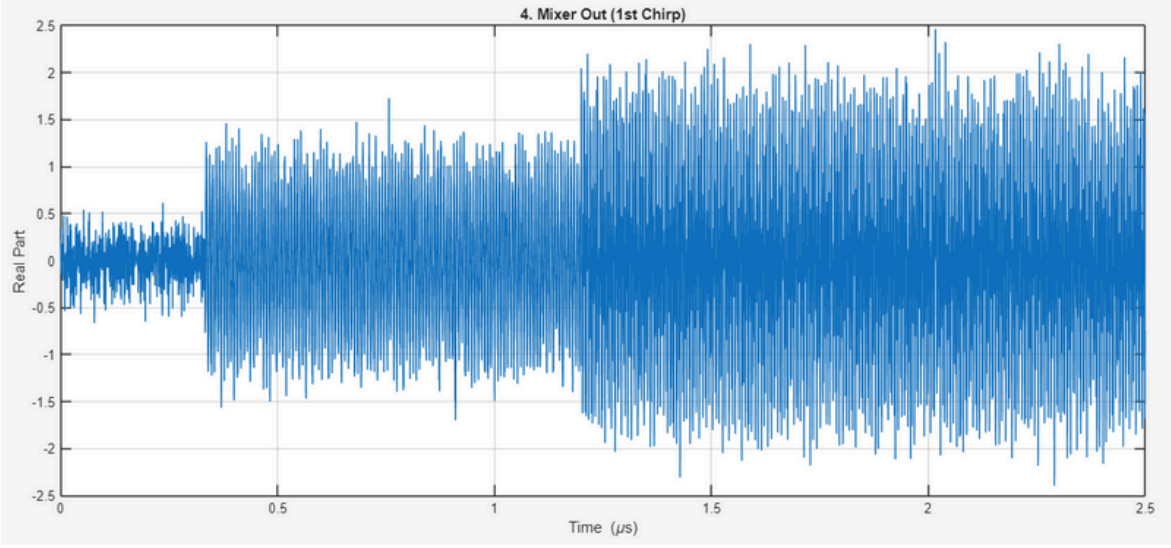
5. Simulation Results and Observations

5.1 Results of the Main Code (SNR=10 db)

- Received Signal (Amplitude vs Time) Plot and Observation:



- Mixer Output (Amplitude vs Time) Plot and Observation:

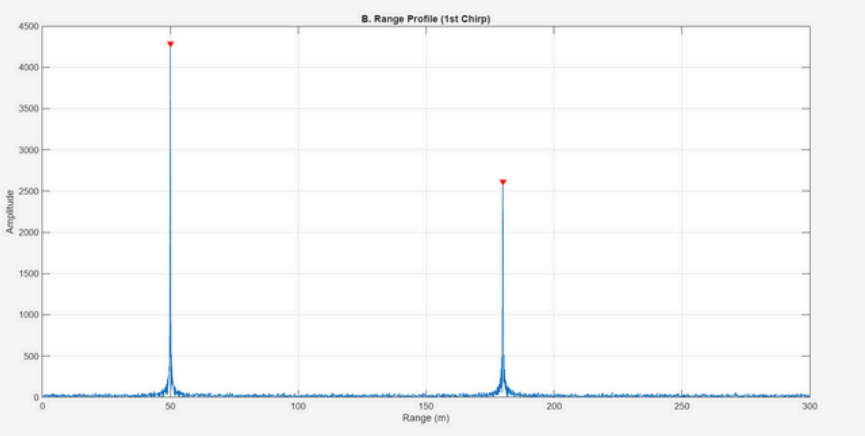


It's obvious from the received signal and the mixer output that the receiver initially detects only low-level background noise from 0 to approximately 0.33 us, which precisely represents the travel time required for the pulse to reach the first target at 50m and return. At exactly 0.33 us, the amplitude sharply spikes as the first echo arrives, followed by a second distinct change in the interference pattern at 1.2 us, which matches the round-trip delay for the further target at 180m.

5. Simulation Results and Observations

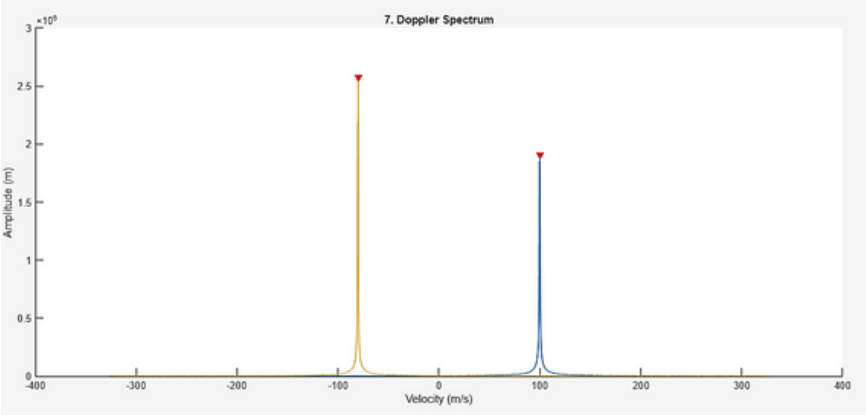
5.1 Results of the Main Code (SNR=10 db)

- Range Profile (Amplitude vs Range) Plot and Observation:



The range profile of the first FMCW chirp shows two clear peaks at approximately 50 m and 180 m, indicating the presence of two targets. The peak at 50 m has a higher amplitude than the one at 180 m, which is expected due to lower propagation loss for the closer target. The spectrum elsewhere remains near the noise floor, showing no significant false targets. The sharp and well-separated peaks confirm accurate range estimation, good range resolution, and an adequate signal-to-noise ratio for reliable target detection.

- Doppler Spectrum (Amplitude vs Velocity) Plot and Observation:



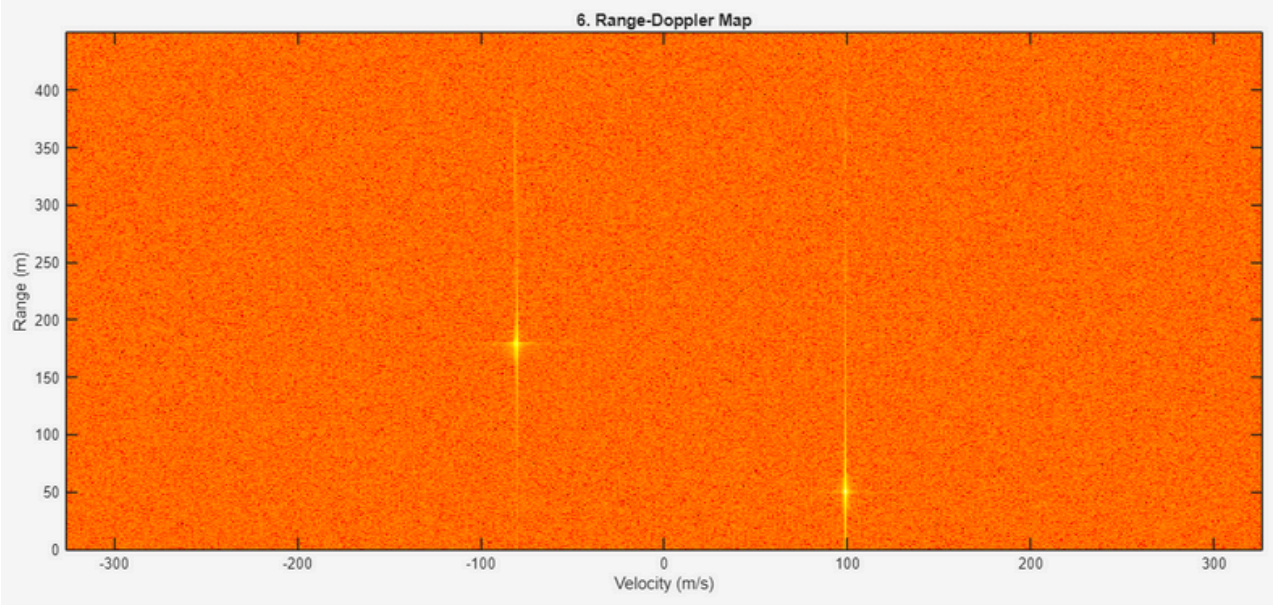
The radar system successfully resolved two targets with high precision, confirming the effectiveness of the Range-Doppler processing chain.

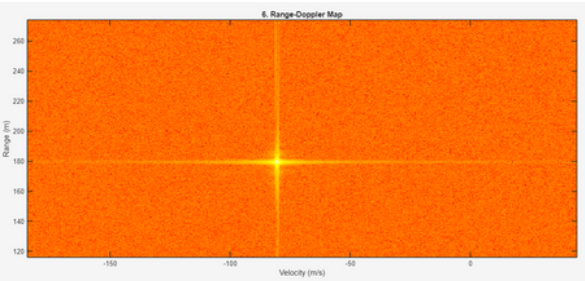
As shown in the command window output, Target 1 was detected at 49.99 m (approaching at 100.15 m/s) and Target 2 at 180.00 m (receding at 80.04 m/s). Visually, the Doppler spectrum (Figure 7) displays two distinct, high-SNR peaks that correspond to these velocities, indicating strong detection confidence. The observed errors—0.01 m for range and a maximum of 0.15 m/s for velocity—are negligible and fall within the system's theoretical resolutions, proving that deviations are merely due to digital quantization (FFT binning) rather than system inaccuracy.

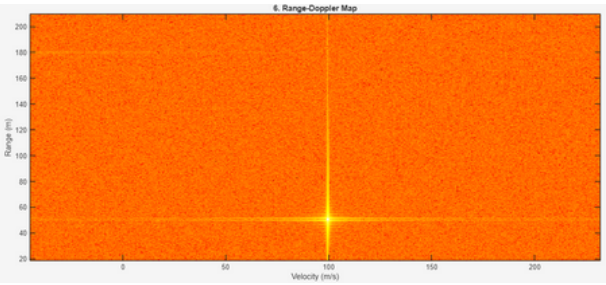
5. Simulation Results and Observations

5.1 Results of the Main Code (SNR=10 db)

- Range-Doppler Map (Range vs Velocity) Plot and Observation:







Based on the provided outputs, the radar system demonstrates excellent accuracy, successfully resolving two distinct targets with parameters that match the ground truth almost perfectly. As confirmed by the Command Window and visual plots, Target 1 was detected at 49.99 m approaching at 100.15 m/s, while Target 2 was identified at 180.00 m receding at 80.04 m/s. These values exhibit negligible errors (maximum 0.15 m/s velocity deviation) that fall well within the system's theoretical resolution limits (0.16 m/s), indicating that the slight discrepancies are due to FFT quantization rather than system failure and validating the robustness of the Range-Doppler processing chain.

6. BONUS: CFAR Target Detection

6.1 CFAR (Dynamic threshold) Physical Concept:

The scientific basis of the Cell-Averaging Constant False Alarm Rate (CA-CFAR) algorithm lies in statistical testing, specifically designed to function in dynamic environments where the background noise power is unknown and time-varying. A fixed threshold is insufficient because an increase in thermal noise, clutter, or jamming would drastically increase the rate of false alarms. CA-CFAR is used here to address this by adaptively adjusting the detection threshold based on a local estimate of the noise floor, ensuring that the probability of false alarm (P_fa) remains constant regardless of the background interference level, where it is the probability that a noise signal is detected as a target, so we need it small enough to ignore noise and big enough to see low power targets (10e-5 → 10e-8).

The core mechanism of CA-CFAR relies on the "sliding window" technique. The algorithm examines a specific bin, designated as the Cell Under Test (CUT), to determine if a target is present. To make this decision statistically robust, the algorithm assumes that the noise distribution in the cells immediately surrounding the CUT follows the same statistical parameters as the noise within the CUT itself. These surrounding cells are termed "Reference Cells". By averaging the signal power across these reference cells, the algorithm computes an estimate of the local noise mean power.

To prevent the energy of the target itself from leaking into this noise estimate and artificially raising the threshold, this is known as self-masking a buffer zone of "Guard Cells" is placed immediately adjacent to the CUT. These guard cells are excluded from the calculation.

Mathematically, the detection threshold (T) is derived by multiplying the estimated noise power (Z) by a scaling factor (bias). so finally the dynamic threshold is generated as a result of convolving the selected window above with the signal and it results in a threshold that is high enough to ignore noise, and low enough to pass a target.

The number of Reference Cells (N_ref) was selected as 20 to balance estimation accuracy. A 16 to 24 reference cells provides a sufficient sample size to get average. Increasing N_ref beyond this point yields diminishing returns in estimation accuracy while increasing the computational cost and the risk of heterogeneous clutter interference.

6. BONUS: CFAR Target Detection

6.2 Code Logic

```
Px_used = 20*log10(abs(mix_fft(1:M_pad/2)));
Pfa = 1e-8;   % For every 10^6 detected target one of them would be a noise
N_reference_value = 20;  % Total reference cells (left , right)
N_guard_value = 4;     % Total guard cells (left and right)
% Calculate Bias (Alpha)
bias =  Pfa^(-1/N_reference_value) - 1;
% Create Sliding Window Mask to use it to convolve
mask = ones(1, N_reference_value + N_guard_value + 1);
mask(N_reference_value/2 + 1 : N_reference_value/2 + N_guard_value + 1) = 0; % Zero
out Guard + CUT
```

Px_used is the signal itself we just take it to work on it for conolution, then the parameters were declared to use them N_ref and N_guard are the total number of cells on the right and on the left of the cell under test. The window then is initialized to ones and on the last line the guard (ignored) cells and the CUT itself are set to zero.

```
% Convolve to get noise floor
noise_estimate = conv(Px_used, mask, 'same');
manual_threshold = 20*log10(noise_estimate * bias);   % dynamic threshold
% logic to check if point greater than threshold it is a true target, other
% it is not a target it is noise
true_peak = Px_used > manual_threshold;
% 2. Find peaks ONLY in the true targets that are detected above
[~, loc_index] = findpeaks(abs(true_peak));  % we want to see the loc. index
corresponds to how many meters
 peak_values = Px_used(loc_index);
% we lookup for the (loc. index)^th item in the range_axis something like mapping it
to meters
peak_ranges = range_axis(loc_index);
```

Now, convolution takes place between the signal and Window (mask), then the threshold is stored in (manual threshold) but raised by a bias to be compatible with our signal. Then a logic check is used to check for targets if a peak is greater than the threshold then it is a true peak and its index is stored in a parameter called loc_index for next calculations.

6. BONUS: CFAR Target Detection

6.2 Code Logic

Now, we store the value of the signal at index of loc_indx in peak values to get how many meters. Finally we store the magnitude value of the peak index at peak ranges to put a blue circle on it (Findpeaks).

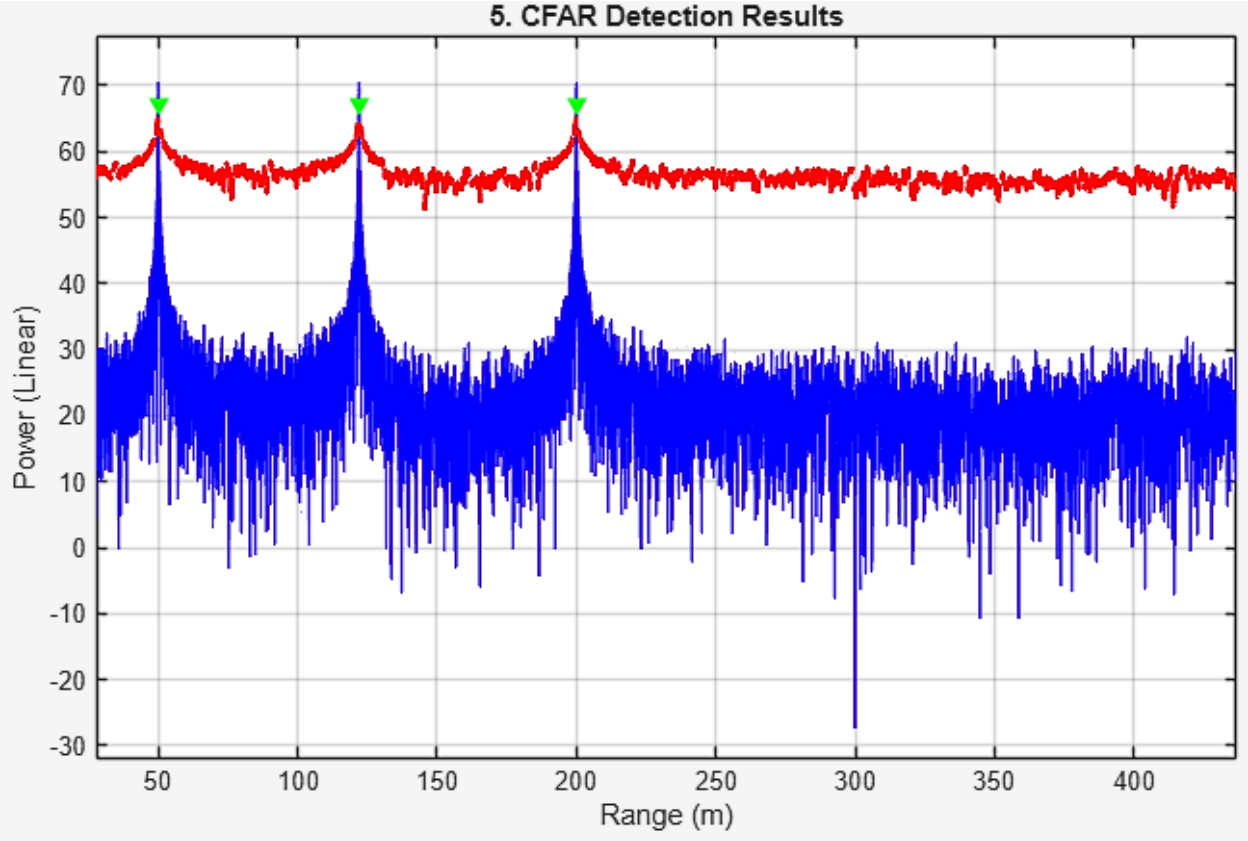
We use the same logic to apply CFAR in detecting the velocity of targets.

6.3 CFAR Outputs

We tried putting three targets:

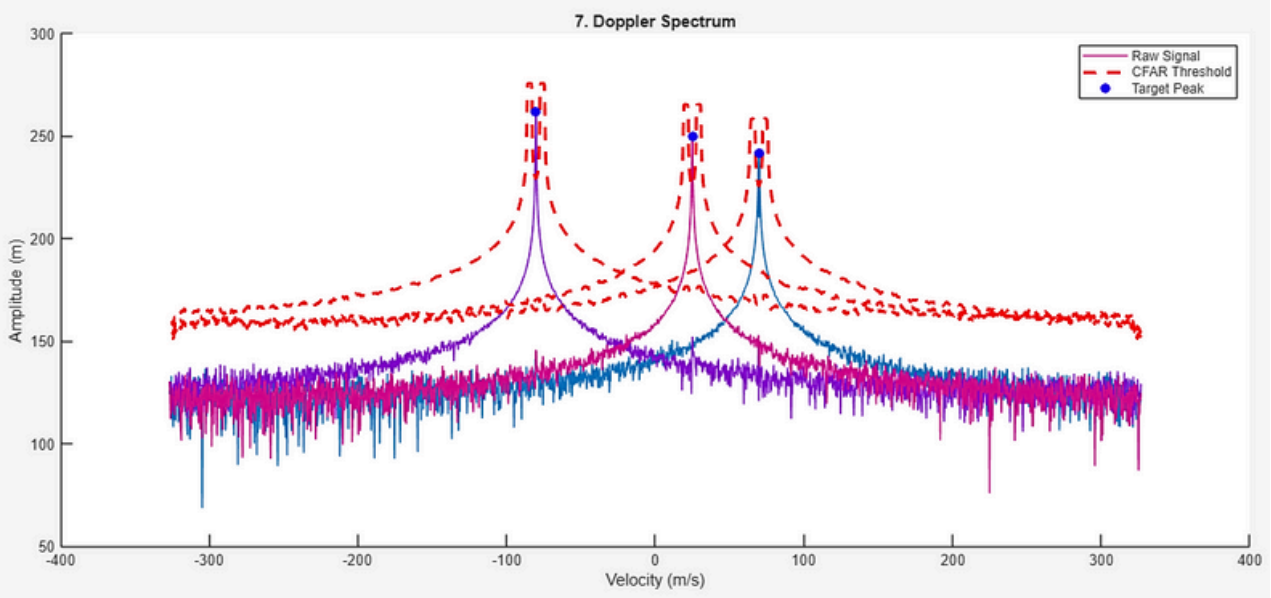
```
targets.R = [50; 122; 200];
targets.v = [+70; -80; 25];
```

and these were the outputs:



6. BONUS: CFAR Target Detection

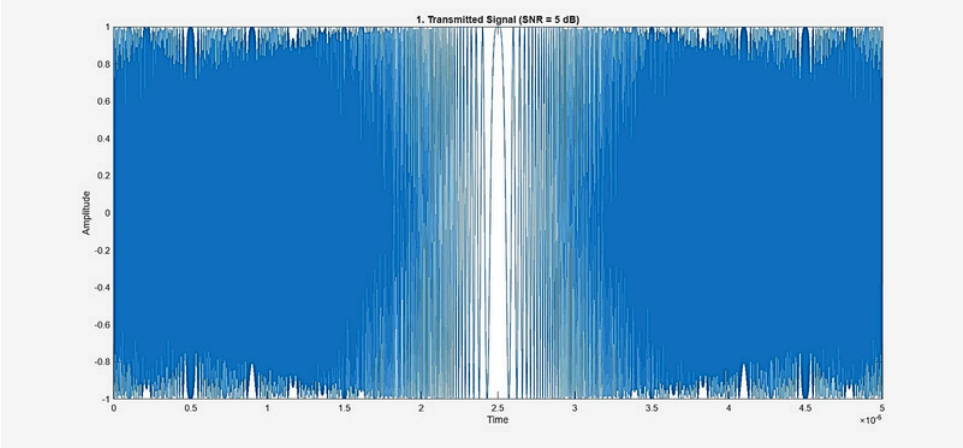
6.3 CFAR Outputs

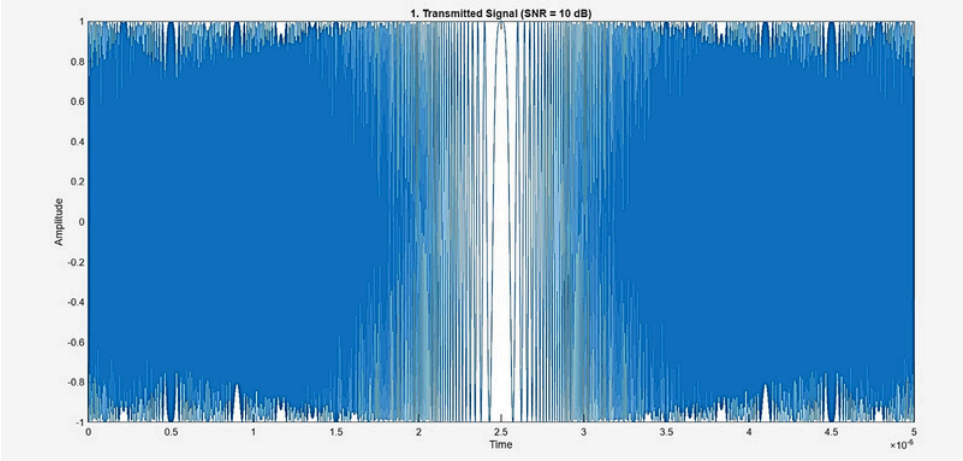


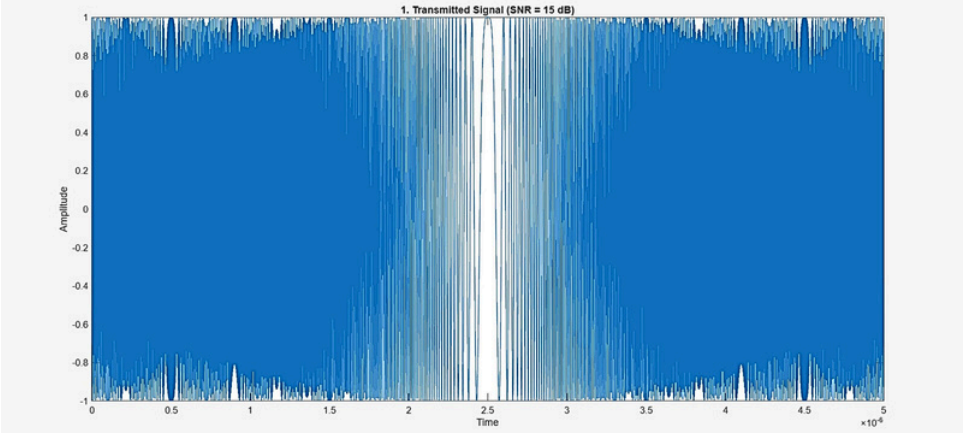
The plots demonstrate a highly effective implementation of the CFAR technique, which successfully differentiates three distinct targets from the background noise in both the Doppler (velocity) and Range domains. The key indicator of success is the adaptive red threshold line; rather than remaining static, it dynamically rides just above the varying noise floor, ensuring that random noise spikes do not trigger false detections while allowing the stronger target signals to break through clearly. This configuration precisely identifies targets at velocities of approximately -80, 25, and 70 m/s and ranges of 50, 120, and 200 meters, confirming that the system is achieving high sensitivity for legitimate targets while maintaining robust noise rejection.

7. BONUS: Comparing Different SNR Values

7.1 Transmitted Signal (5 db vs 10 db vs 15 db in order)



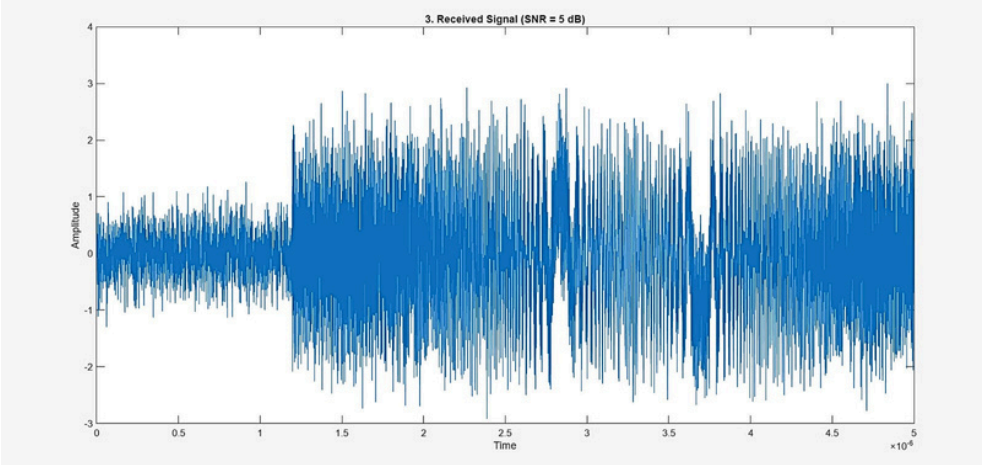


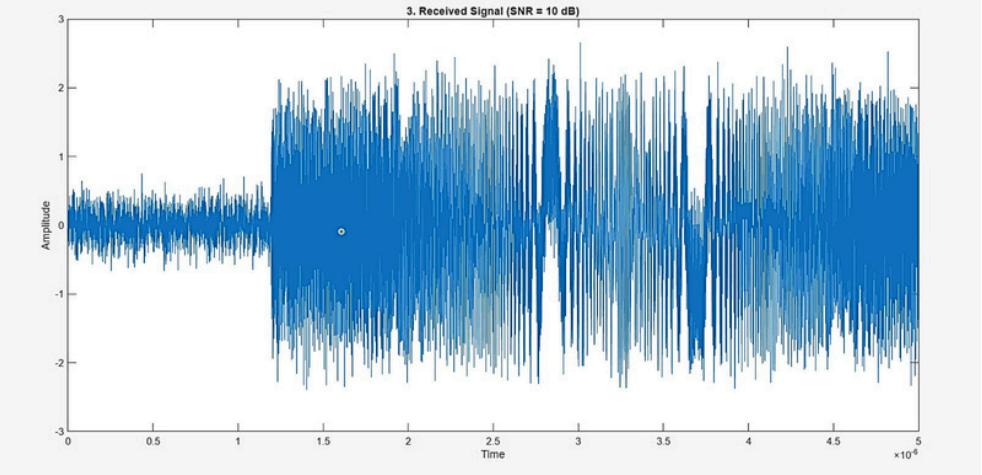


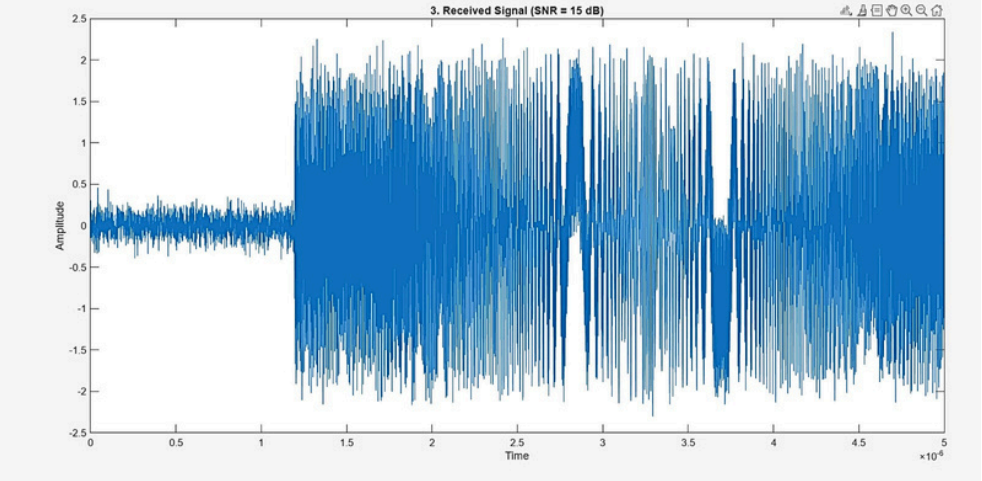
We notice no change in the transmitted signal as it doesn't contain noise at the first place.

7. BONUS: Comparing Different SNR Values

7.2 Received Signal (5 db vs 10 db vs 15 db in order)



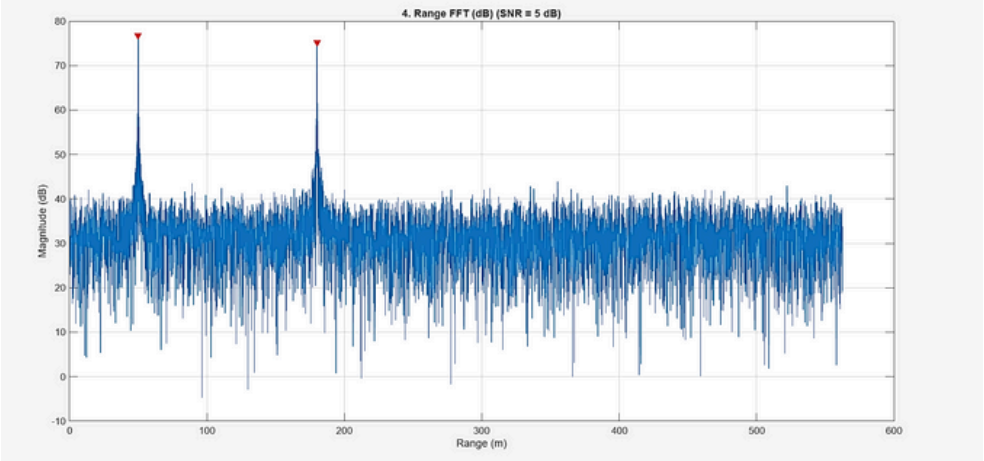


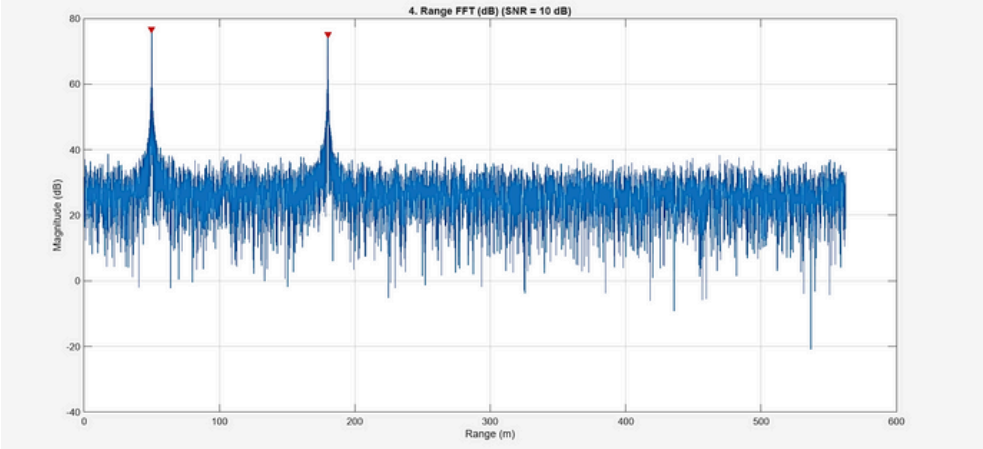


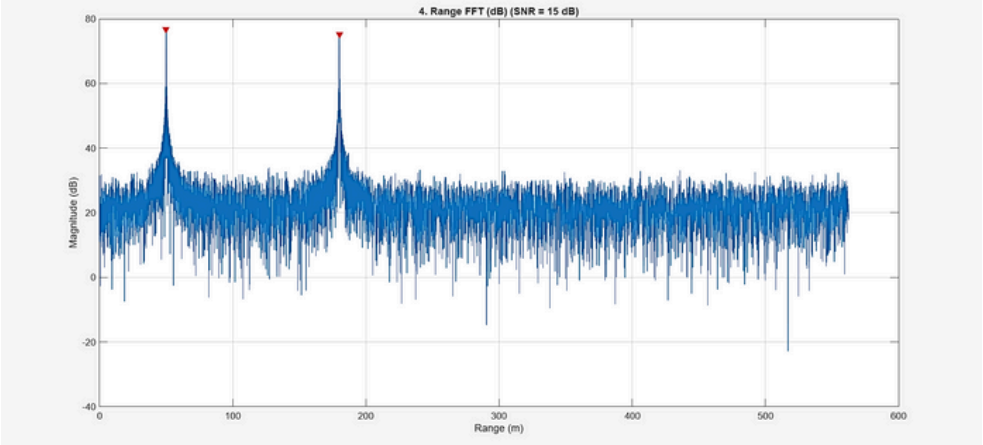
At the 5 db SNR, the signal is heavily impacted by random noise, which obscures fine details and makes the specific amplitude gaps difficult to distinguish from random fluctuations. As the SNR rises to 15 dB, it results in a much cleaner waveform where these structural features become sharp and distinct. **Also the shift in time starts to be more clear as the SNR increases (its ideal value is zero as the signal is shifted).**

7. BONUS: Comparing Different SNR Values

7.3 Range FFT (5 db vs 10 db vs 15 db in order)



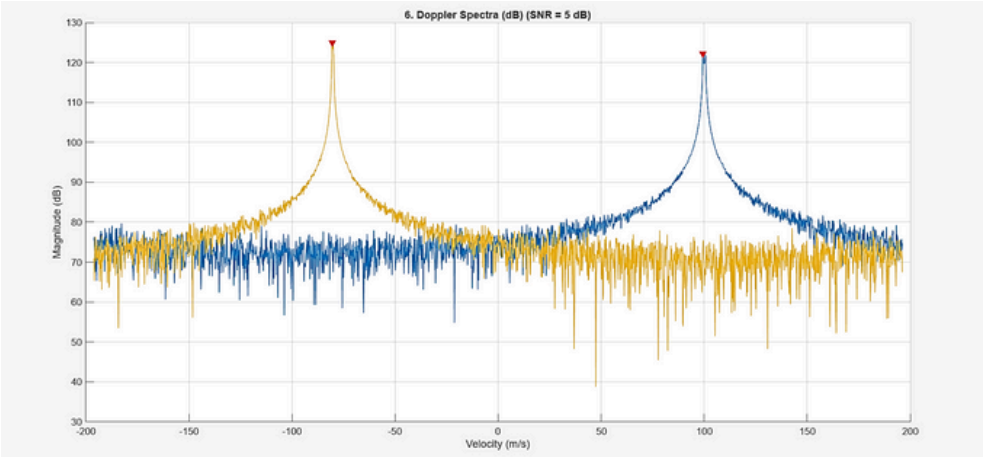


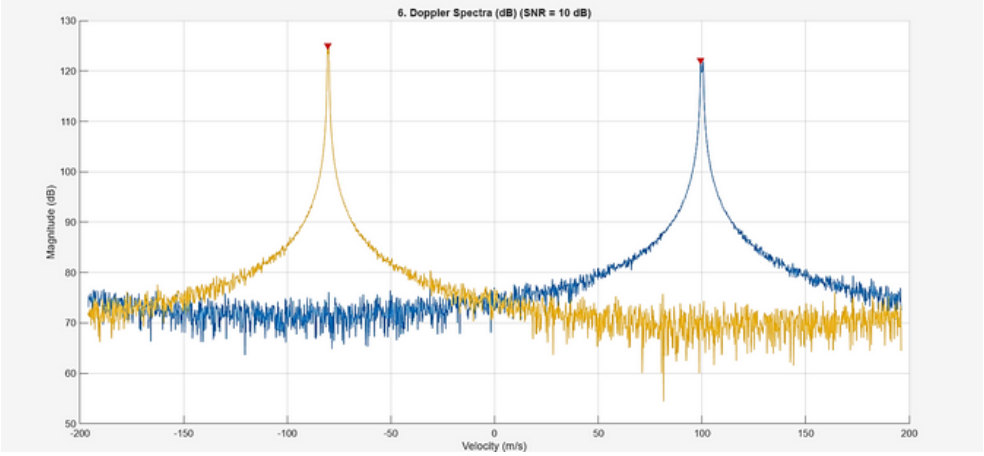


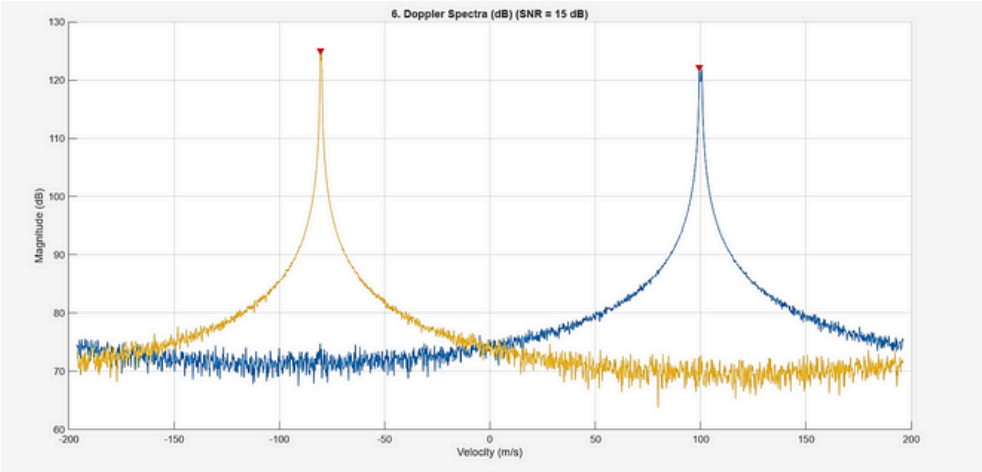
It is too noticeable the noise strength value decreased significantly as the SNR value sweeps from 5 to 15 in the Range-FFT graphs, which clarifies the SNR concept, on the other hand the signal power value is constant.

7. BONUS: Comparing Different SNR Values

7.4 Doppler Spectrum (5 db vs 10 db vs 15 db in order)



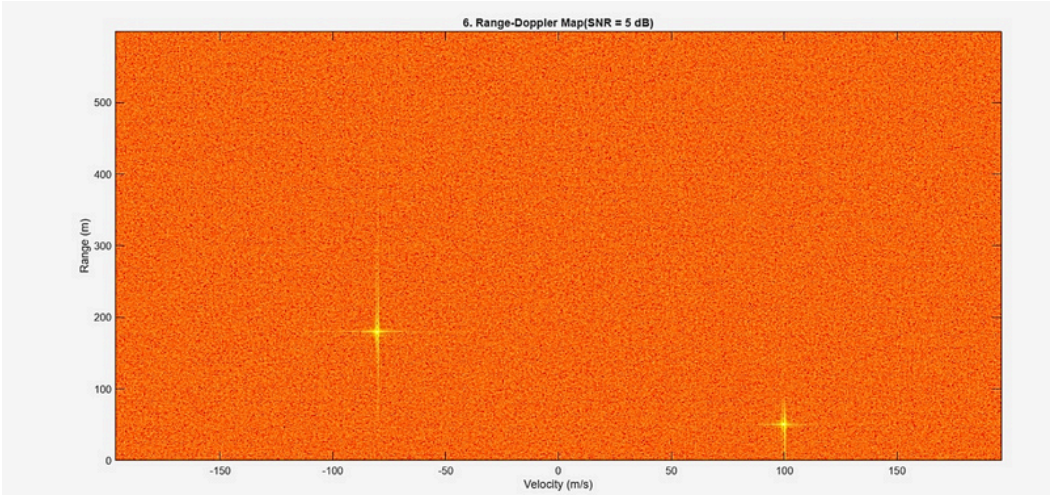


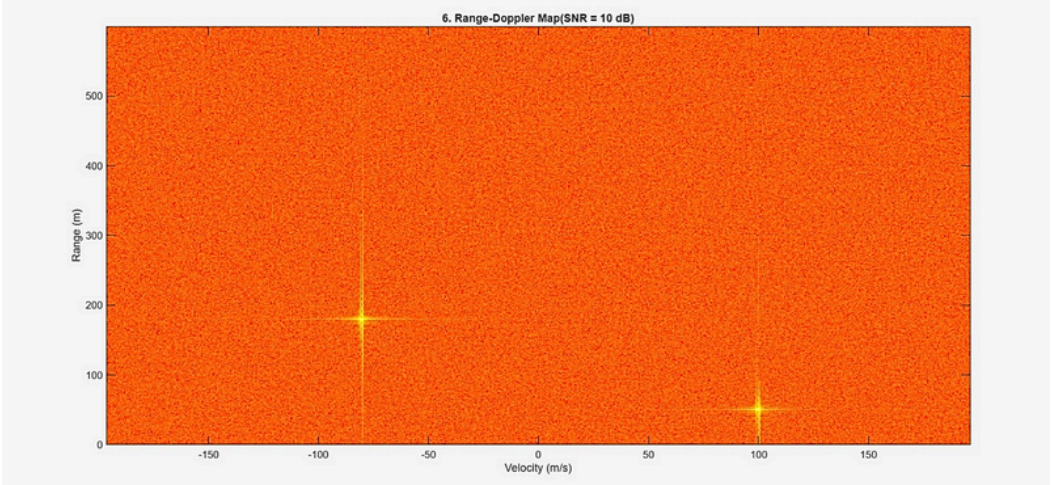


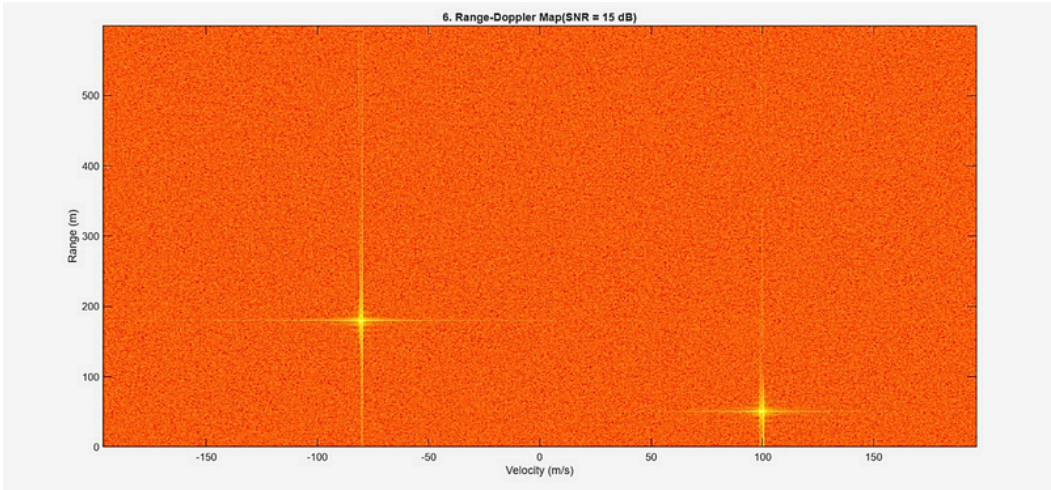
Also the same observation is clear in the Doppler Spectrum graphs, as the noise power strength decreased and the signal turns to be smoother as the SNR sweeps from 5 → 15.

7. BONUS: Comparing Different SNR Values

7.5 Range Doppler Map (5 db vs 10 db vs 15 db in order)



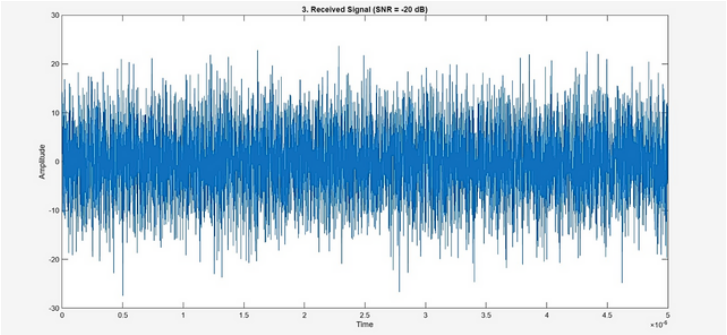


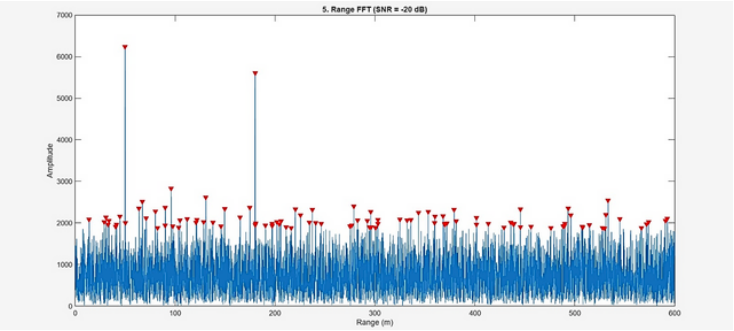


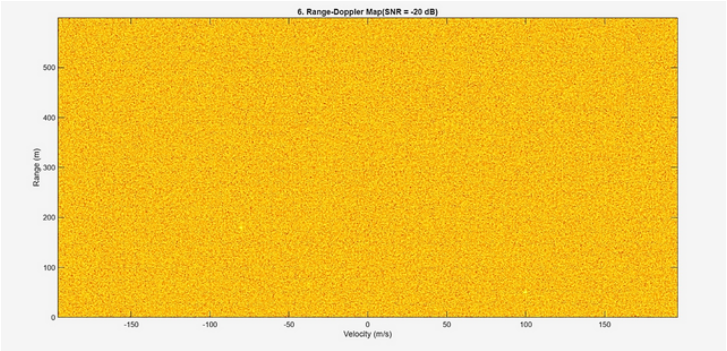
As the SNR value decreases, the map turns to be more grainy; which shows that as the SNR decreases the noise interferes more with the real signal.

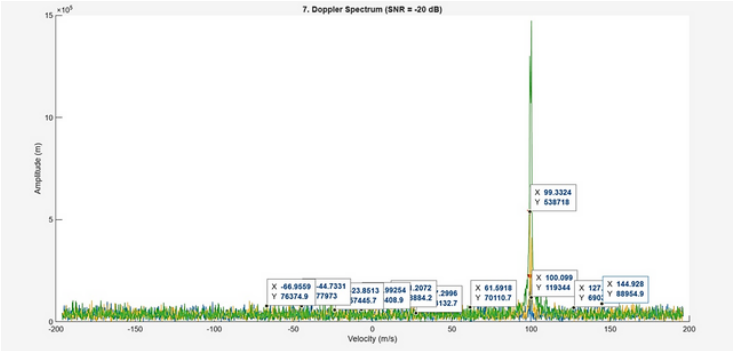
7. BONUS: Comparing Different SNR Values

7.6 Results at SNR = -20 db









At an SNR of -20 dB, the system reaches a failure point where the noise floor completely overwhelms the return signal, causing the detector to frequently mistake random noise spikes for actual objects. Since the noise peaks are significantly stronger than the buried target signal, they easily breach the detection threshold, leading to "False Alarms"—where the system reports ghost targets in empty space—while the real target remains masked and undetected below the chaos. Consequently, without advanced processing like integration to average out these fluctuations, the radar effectively "sees" targets everywhere, rendering the output unreliable as it treats the random environmental static as valid detections.

8. References

1. https://www.mathworks.com/help/radar/ug/automotive-adaptive-cruise-control-using-fmcw-technology.html?s_eid=PSM_15028
2. https://ieeexplore.ieee.org/document/984612
3. https://www.ti.com/lit/wp/spyy005a/spyy005a.pdf?ts=1765806128649
4. https://www.mdpi.com/1424-8220/23/11/5271
5. https://www.researchgate.net/publication/329964953_Design_and_Implementation_of_FMCW_Surveillance_Radar_Based_on_Dual_Chirps
6. https://youtu.be/3CRFXfRZxZ8?si=UvRCSKqUQK_N89zx
7. https://youtu.be/QmgJmh2I3Fw?si=Eo5D5TQoIJotqMKd
8. https://www.tuchemnitz.de/physik/EXSE/ForPhySe/Tong%20Fast%20Chirp%20FMCW%2007455584.pdf
9. https://www.mathworks.com/help/radar/automotive-radar.html?s_eid=PSM_15028
10. https://www.mathworks.com/help/phased/ug/constant-false-alarm-rate-cfar-detection.html
11. https://engineering.purdue.edu/~mrb/resources/AltLectureF/Session_21.pdf
12. https://www.mathworks.com/help/signal/ref/snr.html