FMCW Rader Project

By: Salah Waheed & Islam Ahmed

Codes: 91241268, 91240153

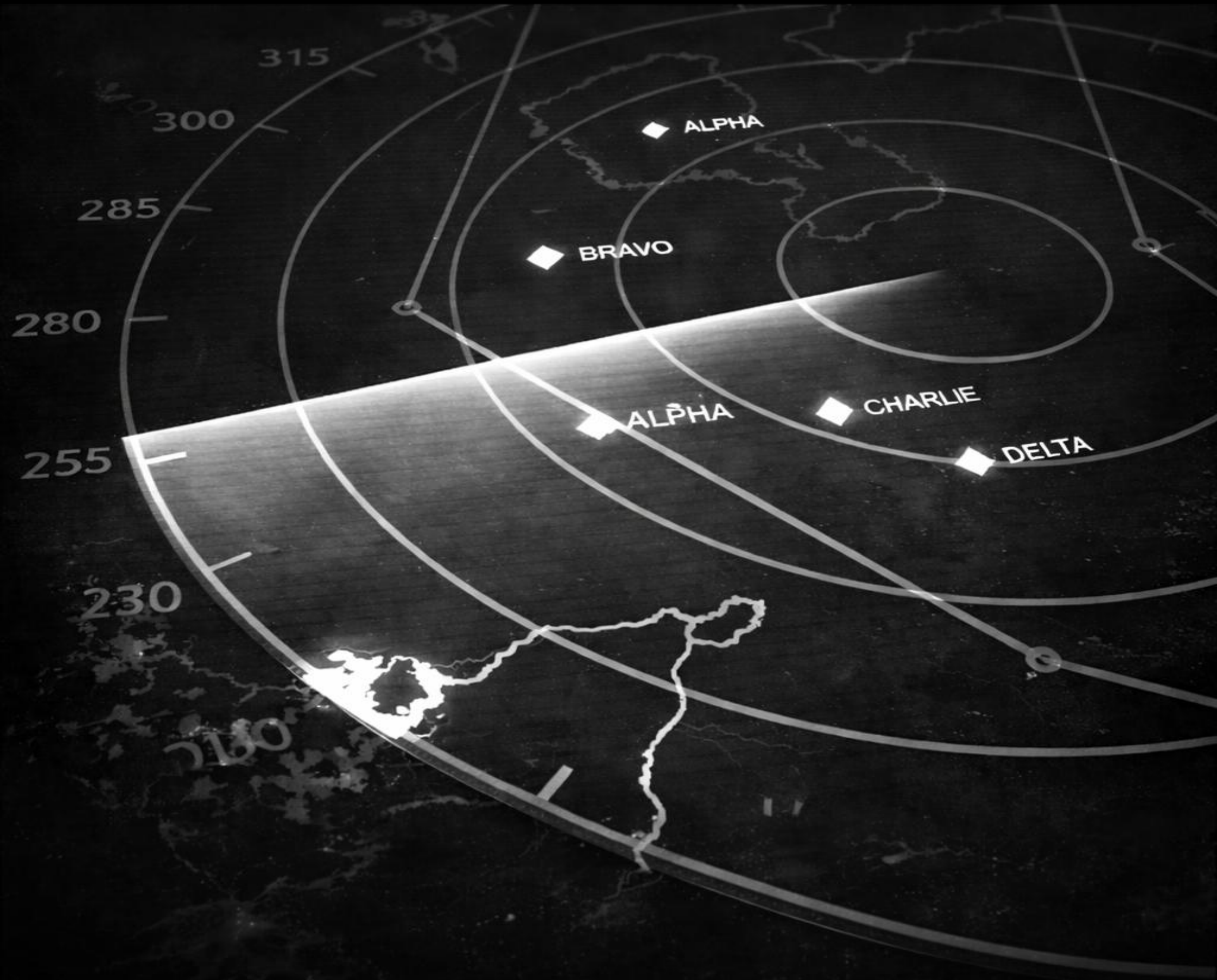


Table Of Contents:

1. Introduction

1.1 Overview of FMCW Radar Technology

Frequency-Modulated Continuous-Wave (FMCW) radar shows up often in today's systems when measuring distance or speed. Instead of using short bursts, these radars send out a steady wave that shifts frequency over time - this shift's called a chirp.

By looking at how much the outgoing and incoming signal frequencies differ - the beat frequency - you can figure both how far away something is and how fast it's moving toward or away from you.

1.2 Project Objective

The main goal of our project is to design and simulate a complete FMCW radar model system in MATLAB. Aspects like wave function generation and target detection are covered under this project. The simulation has complex scenarios where with multiple targets at different ranges and speeds, and Additive White Gaussian Noise (AWGN) is added to it for simulation of practical scenarios.

The implementation process of a project consists of several major stages. Initially, System Configuration establishes critical system parameters such as carrier frequency, bandwidth, and sampling rate. Such parameters are essential in setting the radar resolution and maximum range.

Next, Signal Generation produces the transmitted chirp signal and the received echo. These steps are dependent on the time delays and Doppler phase shifts caused by moving targets.

After signal generation, the processing chain includes Mixing to isolate the beat signal. Then, A 1D Fast Fourier Transform (FFT) that is initially performed on the data, where a peak detection method that employs the use of the findpeaks function **(Bonus Requirement)** is utilized to detect possible targets in the range profile. Subsequently, a 2D FFT is performed to extract velocity data.

Finally, to ensure robust detection within the entire Range-Doppler map under noisy conditions, the system implements a Constant False Alarm Rate (CFAR) **(Bonus Requirement)**.

This technical report provides the design process, the theoretical background, and the simulation results, providing a deep analysis of FMCW radar behaviours. Results demonstrate the effectivity of 2D signal processing, even when the Signal-to-Noise Ratio (SNR) changes.

2. Theoretical Background

2.1 The Principle of FMCW Radar

FMCW radar works differently from traditional pulsed radar in that it constantly sends out signal whose frequency changes over time, rather than employing the high-power, short-duration pulses that is used in traditional pulsed radar. This approach allows the system to measure both distance and speed simultaneously without using massive amounts of peak power.

2.1.1 The Chirp Waveform

The basis of the system is something called the "chirp." This is simply a sine wave that varies its frequency linearly over some time—in other words, just like a sound pitch sliding from low to high. We define the instantaneous frequency, f(t) as:

$$\boxed{\boldsymbol{f(t) = f_c + S \cdot t}}$$

Here, f_c is our starting carrier frequency, and S is the "slope" or sweep rate, it's the rate at which the instantaneous frequency changes with time. It determines how quickly the frequency sweeps across the bandwidth. The slope is determined by how much bandwidth we cover during the chirp duration (T_c).

$$\boxed{\boldsymbol{S = \frac{B}{T_c}}}$$

Signal Model (Baseband Simulation):
While the physical radar transmits at a high carrier frequency (f_c), this project simulates the Baseband signal to optimize computational efficiency. By modeling the complex envelope of the modulation, the transmitted signal (S_{TX}) is defined as:

$$\boxed{\boldsymbol{S_{TX}(t) = e^{j\pi S t^2}\ , 0 \le t \le T_c}}$$

2.1.2 Signal Reception and De-chirping

Once the chirp is transmitted, it reflects off a target and returns to the radar as a delayed echo.
Received Signal Model:

The received signal (S_{RX}) is simply a time-delayed and attenuated version of the transmitted signal.
the received baseband signal is:

$$\boxed{\boldsymbol{S_{RX}(t) = A \cdot e^{j\pi S(t-\tau)^2}}}$$

Where $\boldsymbol{A}$ represents the amplitude (attenuated by the radar equation) and $\boldsymbol{\tau}$ is the delay.

Mixing Operation (De-chirping):
In order to extract the range information, the signal is mixed by the receiver by echoing the incoming signal with the signal that is currently being transmitted. In the digital domain (and implemented by the simulation code), this is done by multiplying the transmitted signal (S_{TX}) by the complex conjugate of the received signal ($S^*{}_{RX}$). This operation effectively subtracts the phase of the received signal from the transmitted one:

$$\boxed{\boldsymbol{S_{mix}(t) = S_{TX}(t) \cdot S^*{}_{RX}(t) = e^{j\pi S t^2} \cdot e^{-j\pi S(t-\tau)^2}}}$$

The Beat Signal:
Simplifying the exponents reveals the frequency content of the mixed signal. The quadratic terms cancel out, leaving a linear phase term:

$$\boxed{\boldsymbol{S_{mix}(t) \approx e^{j2\pi(S\tau)t}}}$$

The signal that is obtained from this process is a pure sine wave of constant frequency called **Beat Frequency** (f_b). By observing the above expression, it has become clear that beat frequency is directly proportional to both the slope and the delay:

$$\boxed{\boldsymbol{f_b = S \cdot \tau = S \cdot \frac{2R}{c}}}$$

Given the value of the slope S, the beat frequency offers a direct measurement of the target's range

2.1.3 Range Estimation (Fast-Time Processing)

The Range estimation is done by evaluating the frequency spectrum of the beat frequency signal within a single chirp. This timescale is referred to as "Fast Time."

The round-trip time delay (τ) for a signal to travel to a target at range and return is given by:

$$\boxed{\boldsymbol{\tau = \frac{2R}{c}}}$$

During this delay, the transmitted frequency has increased by an amount determined by the slope. Therefore, the beat frequency is directly proportional to the delay, we can calculate the Range using the beat signal frequency using the following equation:

$$\boxed{\boldsymbol{R = \frac{c \cdot f_b}{2 \cdot S}}}$$

By performing a Fast Fourier Transform (FFT) on the sampled beat signal, the system identifies peaks in the frequency domain. The frequency of each peak corresponds to a specific target range.

2.1.4 Velocity Estimation (Slow-Time Processing)

While range is extracted from the frequency shift within a chirp, velocity is extracted from the phase shift across multiple consecutive chirps. This timescale is referred to as "Slow Time."

If it is a moving target, its distance changes slightly between one chirp and the following one (separated by time T_c). This change in distance will cause phase rotation $(\Delta\phi)$ in the beat signal:

$$\boxed{\boldsymbol{\Delta\phi = 2\pi \cdot \frac{2vT_c}{\lambda}}}$$

This phase transition occurs in the form of a Doppler frequency (f_d) during the examination of the signal within the sequence of chirps. By undergoing a second FFT across the chirps (the Doppler FFT), the measurement of the frequency enables the calculation of the relative velocity as follows:

$$\boxed{\boldsymbol{v = \frac{\lambda \cdot f_d}{2}}}$$

3. System Design & Parameters

The radar system parameters were selected to fulfill specific simulation objectives defined for this project. These design constraints dictate the waveform properties used in the MATLAB signal generation stage to ensure accurate detection and resolution of multiple targets.

3.1 Design Specifications

The following parameters were selected for the signal generation stage. Each of these parameters is essential for determining the radar's operational mode and sensitivity.

3.1.1 Carrier Frequency (f_b)

$$\boxed{\boldsymbol{f_c = 76.5\ GHz}}$$

This frequency has a bandwidth within the conventional automotive mmWave range. A higher carrier frequency corresponds to a shorter wavelength ($\lambda \approx 3.9\ mm$), which increases the radar's sensitivity to small movements (Doppler shift) and allows for smaller antenna apertures.

3.1.2 Bandwidth (B)

$$\boxed{\boldsymbol{B = 1\ GHz}}$$

The main determining component of the Range Resolution is the bandwidth. A wider bandwidth allows the radar to distinguish between targets that are positioned closely together. With 1 GHz of bandwidth, the system is designed for high-precision detection.

3.1.3 Chirp Duration (T_c) and Pulse Repetition Interval (PRI)

$$\boxed{\boldsymbol{T_c = 3\mu s\ ,PRI = 8.4\mu s}}$$

The chirp duration (T_c) defines the energy on target and the slope of the frequency sweep. The PRI (the time between the start of one chirp and the next) determines the Maximum Unambiguous Velocity the system can measure before aliasing occurs.

3.1.4 Slope (S)

$$\boxed{\boldsymbol{S = \frac{B}{T_c} \approx 3.33 \times 10^{14}\ Hz/s}}$$

The slope maps the target's range to a specific beat frequency. A steeper slope (like this one) means a higher beat frequencies for the same range, requiring a faster ADC but offers greater noise immunity.

3.1.5 Sampling Frequency (F_S)

$$\boxed{F_S = 2\ GHz\ (T_S = 0.5\ ns)}$$

The sampling frequency determines the highest possible bandwidth that can be processed by Receiver (ADC). According to the Nyquist theorem, (F_S) establishes the Maximum Unambiguous Range (R_{max}) by defining the highest beat frequency ($f_{b,max}$) the system can digitize without aliasing (explained in the **System Performance Limits** section).

3.2 System Performance Limits

Based on the design specifications above, the theoretical limits of the radar system are derived as follows.

3.2.1 Range Resolution (ΔR)

It defines the minimum distance required to separate two distinct targets.

$$\boxed{\Delta R = \frac{c}{2B} = \frac{3 \times 10^8}{2 \times 10^9} = 0.15\ m}$$

The system can distinguish targets separated by just **15 cm**.

3.2.2 ADC Nyquist Frequency Limit ($F_{nyquist}$)

This is the absolute maximum frequency component the Analog-to-Digital Converter (ADC) can process before aliasing occurs, it serves as the fundamental hardware bottleneck.

$$\boxed{F_{nyquist} = \frac{F_S}{2} = \frac{2 \times 10^9}{2} = 1\ GHz}$$

Any beat frequency exceeding **1 GHz** will fold back (alias) into the spectrum**, creating false targets (ghosts).**

3.2.3 Maximum Unambiguous Range (R_{max}) (Instrumented)

This is the theoretical maximum range determined by the Nyquist limit calculated above. Since $F_{nyquist} = 1\ GHz$, we calculate the range that would produce exactly this beat frequency.

$$\boxed{R_{max} = \frac{c \cdot F_{nyquist}}{2 \cdot S} = \frac{3 \times 10^8 \cdot 10^9}{2 \cdot 3.33 \times 10^{14}} \approx 450\ m}$$

While the project targets a 200m range, the system hardware is capable of detecting targets up to **450m** before aliasing occurs.

3.2.4 Maximum Unambiguous Velocity (V_{max})

The highest possible relative speed that the radar can measure without Doppler ambiguity is expressed by the Pulse Repetition Interval (PRI).

$$\boxed{\boldsymbol{V_{max} = \frac{\lambda}{4 \cdot PRI} = \frac{0.00392}{4 \cdot 8.4 \times 10^{-6}} \approx 116\ m/s}}$$

The system can correctly measure speeds up to **116 m/s (approx 417 km/h)**.

3.2.5 Velocity Resolution (Δv)

The ability to distinguish between targets moving at similar speeds depends on the total observation time (Number of pulses $\boldsymbol{N_P} \times \boldsymbol{PRI}$).

$$\boxed{\boldsymbol{\Delta v = \frac{\lambda}{2 \cdot N_P \cdot PRI} = \frac{0.00392}{4 \cdot 512 \cdot 8.4 \times 10^{-6}} \approx 0.45\ m/s}}$$

The system can distinguish velocity differences as small as **0.45 m/s**.

4. Implementation Methodology

In this section we will discuss how we implemented the project and executed the different tasks to achieve an effective, working FMCW Rader.

First, we will walk through the codes in this section and in the next section will be the simulation outputs.

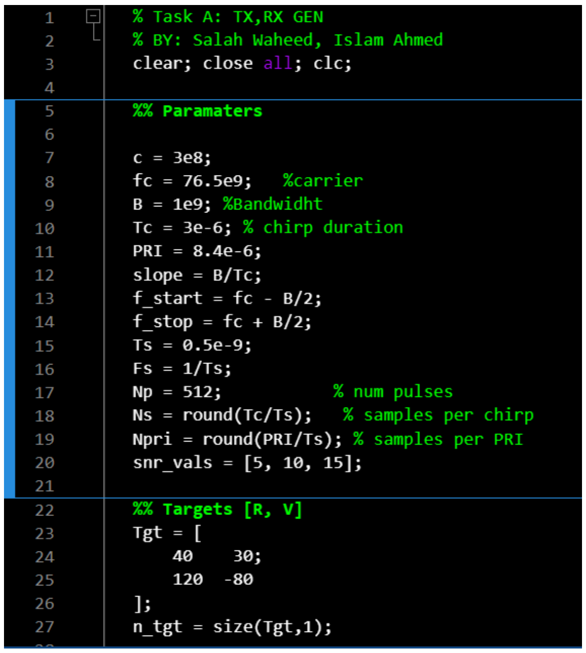
4.1 Task A: Signal Generation

The goal of this task is to design a pulse-Doppler radar system by selecting appropriate parameters for a millimeter-wave (mmWave) radar operating in the 76–77 GHz carrier frequency range.

The simulation must:

1. **Configure Parameters:** Select Bandwidth and PRI to achieve a range resolution **< 0.75 m** and velocity resolution **< 0.5 m/s**. which we already did and showed in the previous section
2. **Simulate Signals:** Generate the transmitted chirp and the delayed Received signal.
3. **Model Targets:** Create a scenario with two moving targets (one approaching, one receding) at realistic ranges.
4. **Generate Echoes:** Construct the received signal by incorporating the round-trip time delay and the Doppler phase shift accumulated across pulses $(\boldsymbol{\phi}[\boldsymbol{n}] = \boldsymbol{2jf_d} \cdot \boldsymbol{n} \cdot \boldsymbol{PRI})$
5. **Add Noise:** Inject Additive White Gaussian Noise (AWGN) to test detection at different SNR levels.

Code Implementation & Analysis



```
1   % Task A: TX,RX GEN
2   % BY: Salah Waheed, Islam Ahmed
3   clear; close all; clc;
4
5   %% Paramaters
6
7   c = 3e8;
8   fc = 76.5e9;   %carrier
9   B = 1e9; %Bandwidht
10  Tc = 3e-6; % chirp duration
11  PRI = 8.4e-6;
12  slope = B/Tc;
13  f_start = fc - B/2;
14  f_stop = fc + B/2;
15  Ts = 0.5e-9;
16  Fs = 1/Ts;
17  Np = 512;            % num pulses
18  Ns = round(Tc/Ts);   % samples per chirp
19  Npri = round(PRI/Ts); % samples per PRI
20  snr_vals = [5, 10, 15];
21
22  %% Targets [R, V]
23  Tgt = [
24      40    30;
25      120  -80
26  ];
27  n_tgt = size(Tgt,1);
```

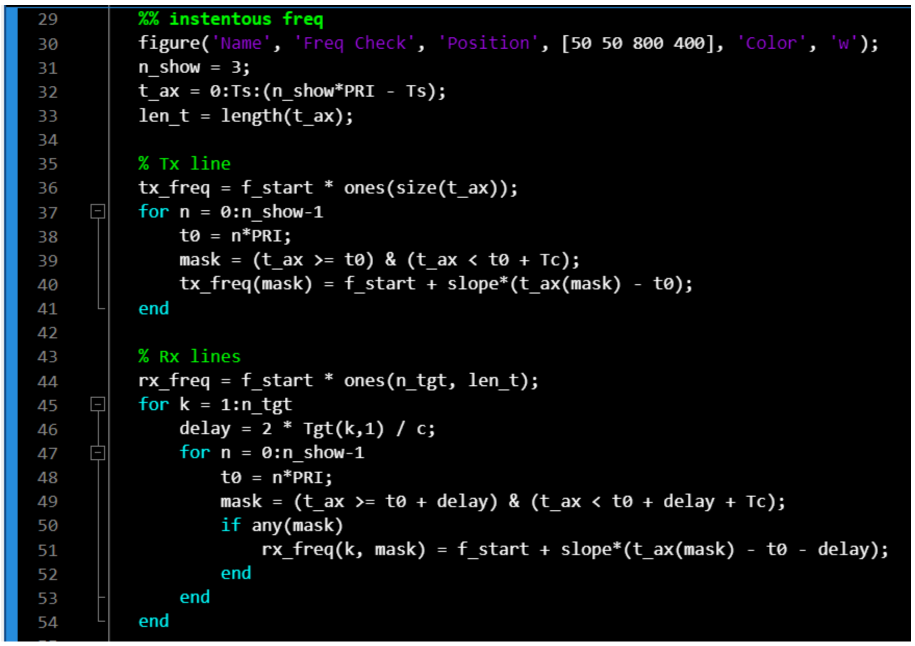
Figure 1: Task A Parameters and targets definition

The simulation begins by defining the radar's physical constants and operating parameters that we talked about in the system parameters section, And the targets definition.

Line 5-21: The carrier frequency is set to fc = 76.5 GHz. The Bandwidth (B = 1 GHz) and Chirp Duration (Tc = 3 us) are selected to maximize resolution. The sampling frequency (Fs) is set to 2 GHz to satisfy the Nyquist criterion.

Snr_vals refers to the values of snr that the simulation is going to implement.

Line 22- 27: The target scenario is defined in the **tgt** matrix. Target 1 is located at **40m** moving at **30 m/s** (approaching), and Target 2 is at **120m** moving at **-80 m/s** (receding).



```
29  %% instentous freq
30  figure('Name', 'Freq Check', 'Position', [50 50 800 400], 'Color', 'w');
31  n_show = 3;
32  t_ax = 0:Ts:(n_show*PRI - Ts);
33  len_t = length(t_ax);
34
35  % Tx line
36  tx_freq = f_start * ones(size(t_ax));
37  for n = 0:n_show-1
38      t0 = n*PRI;
39      mask = (t_ax >= t0) & (t_ax < t0 + Tc);
40      tx_freq(mask) = f_start + slope*(t_ax(mask) - t0);
41  end
42
43  % Rx lines
44  rx_freq = f_start * ones(n_tgt, len_t);
45  for k = 1:n_tgt
46      delay = 2 * Tgt(k,1) / c;
47      for n = 0:n_show-1
48          t0 = n*PRI;
49          mask = (t_ax >= t0 + delay) & (t_ax < t0 + delay + Tc);
50          if any(mask)
51              rx_freq(k, mask) = f_start + slope*(t_ax(mask) - t0 - delay);
52          end
53      end
54  end
```

Figure 2: Generation of the Transmitted and Received frequency sweeps

This section generates the **Frequency vs. Time diagram**, which visually validates the FMCW waveform structure and the delay principle.

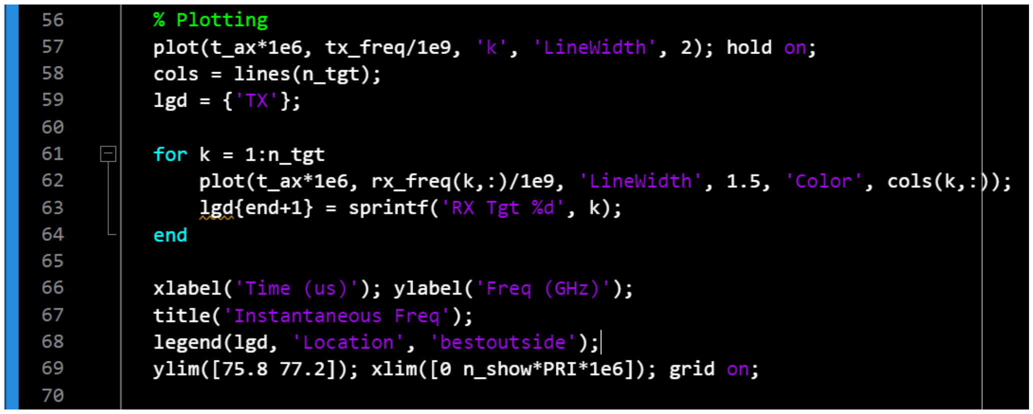
Lines 31–33: The plot is limited to the first 3 pulses **(n_show = 3)** to ensure the chirp slopes are clearly visible and distinct.

TX Signal Loop (Lines 35–41): The code generates the transmitted "sawtooth" pattern.

- It iterates through the specified number of pulses.
- The mask creates a logical vector to select the active chirp duration (Tc).
- The frequency $f_{tx}(t)$ is calculated linearly using the defined slope: $\boldsymbol{f_{start} + S \times (t - t_0)}$.

RX Signal Loop (Lines 43–51): The code calculates the echo frequency for each target in the scenario.

- The round-trip time delay **($\boldsymbol{\tau}$)** is calculated as ($\boldsymbol{\tau = \frac{2R}{C}}$).
- The received frequency ramp is generated identical to the TX signal but shifted in time by the calculated delay. This visualization confirms that the received signal is a delayed replica of the transmitted chirp, where the horizontal shift represents the target's range.



```
56      % Plotting
57      plot(t_ax*1e6, tx_freq/1e9, 'k', 'LineWidth', 2); hold on;
58      cols = lines(n_tgt);
59      lgd = {'TX'};
60
61      for k = 1:n_tgt
62          plot(t_ax*1e6, rx_freq(k,:)/1e9, 'LineWidth', 1.5, 'Color', cols(k,:));
63          lgd{end+1} = sprintf('RX Tgt %d', k);
64      end
65
66      xlabel('Time (us)'); ylabel('Freq (GHz)');
67      title('Instantaneous Freq');
68      legend(lgd, 'Location', 'bestoutside');
69      ylim([75.8 77.2]); xlim([0 n_show*PRI*1e6]); grid on;
70
```

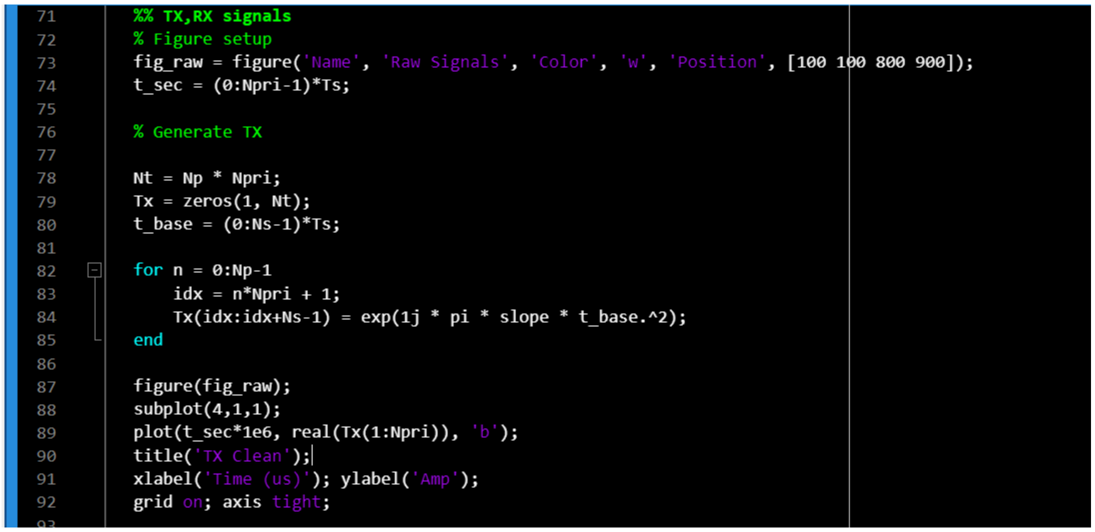
Figure 3: visualizing the instantaneous frequency versus time.

This block handles the visualization of the Frequency of the TX and RX generated in the previous loop.

Line 57: The Transmitted Signal (TX) is plotted in black ('k') with a thicker linewidth to serve as the reference chirp.

Lines 59–62: The code iterates through each target to plot its Received Signal (RX). The **lines(n_tgt)** function generates distinct colors for each target, making it easy to differentiate them visually.

Lines 63–65: The plot is annotated with axis labels (Time in us, Frequency in GHz) and a dynamic legend. The axes are scaled to focus strictly on the active chirp region.



```
71      %% TX,RX signals
72      % Figure setup
73      fig_raw = figure('Name', 'Raw Signals', 'Color', 'w', 'Position', [100 100 800 900]);
74      t_sec = (0:Npri-1)*Ts;
75
76      % Generate TX
77
78      Nt = Np * Npri;
79      Tx = zeros(1, Nt);
80      t_base = (0:Ns-1)*Ts;
81
82      for n = 0:Np-1
83          idx = n*Npri + 1;
84          Tx(idx:idx+Ns-1) = exp(1j * pi * slope * t_base.^2);
85      end
86
87      figure(fig_raw);
88      subplot(4,1,1);
89      plot(t_sec*1e6, real(Tx(1:Npri)), 'b');
90      title('TX Clean');
91      xlabel('Time (us)'); ylabel('Amp');
92      grid on; axis tight;
```

Figure 4: implementation of the Baseband Transmitted Signal.

This section and the next implements the core signal processing physics required to model the radar environment.

This block is responsible for synthesizing the "ideal" radar waveform that will be transmitted into the environment.

Global Setup (Lines 78–79):
The total number of samples **Nt** is calculated as the product of the number of pulses (N_P) and samples per PRI (N_{PRI}). The Tx array is initialized to zero to hold the full sequence.

Chirp Time Vector (Line 80):
t_base is a time vector representing the duration of a single active chirp (**Tc**). This vector is used to calculate the phase of the wave.

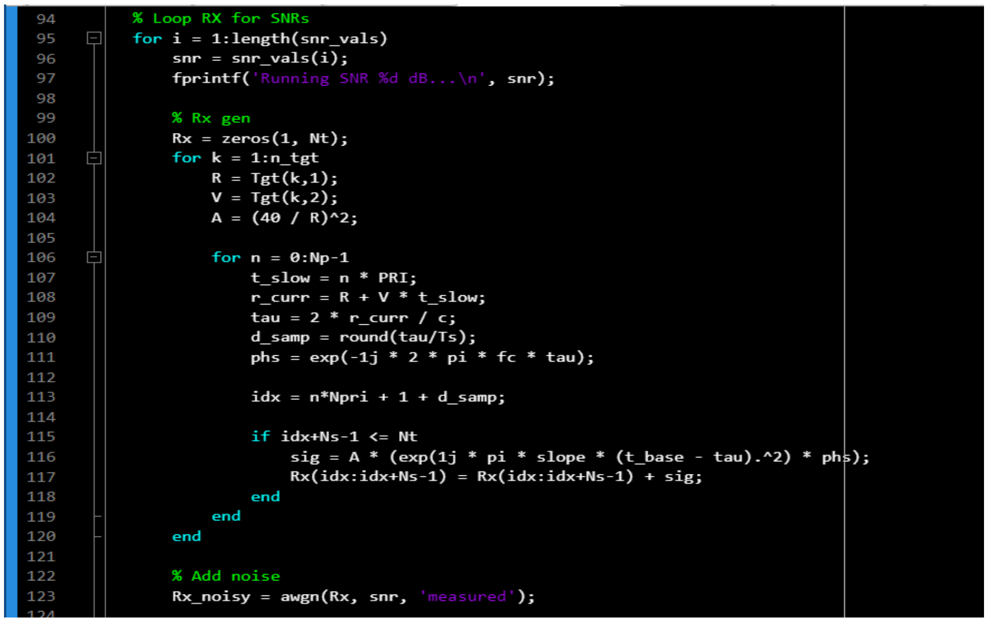
Signal Synthesis Loop (Lines 82–85):
The code iterates through every pulse index (n) to construct the full waveform.

- **Line 84:** The baseband signal is generated using the standard Linear Frequency Modulation (LFM) equation:

$$\boxed{S_{TX}(t) = e^{j\pi S t^2}}$$

  where **S** is the slope and **t** is the **t_base** time vector.

- This generates a complex chirp with a quadratic phase (linear frequency ramp) for every pulse period, creating the coherent waveform required for Doppler processing.



```
% Loop RX for SNRs
for i = 1:length(snr_vals)
    snr = snr_vals(i);
    fprintf('Running SNR %d dB...\n', snr);

    % Rx gen
    Rx = zeros(1, Nt);
    for k = 1:n_tgt
        R = Tgt(k,1);
        V = Tgt(k,2);
        A = (40 / R)^2;

        for n = 0:Np-1
            t_slow = n * PRI;
            r_curr = R + V * t_slow;
            tau = 2 * r_curr / c;
            d_samp = round(tau/Ts);
            phs = exp(-1j * 2 * pi * fc * tau);

            idx = n*Npri + 1 + d_samp;

            if idx+Ns-1 <= Nt
                sig = A * (exp(1j * pi * slope * (t_base - tau).^2) * phs);
                Rx(idx:idx+Ns-1) = Rx(idx:idx+Ns-1) + sig;
            end
        end
    end

    % Add noise
    Rx_noisy = awgn(Rx, snr, 'measured');
```

Figure 5: implementation of the Received Signal loop and AWGN noise addition.

This block constructs the received signal ($S_{RX}(t)$) by mathematically modelling the echo as a delayed and phase-shifted version of the transmitted chirp and simulates environmental noise

SNR Loop (Lines 95–97):
The entire reception process is wrapped in a loop for **(i = 1:length(snr_vals))**. This allows the simulation to generate separate datasets for each required SNR level (5, 10, and 15 dB).

Target and Pulse Loops (Lines 101–106):
The code iterates through each target (**k**) and each pulse (**n**) to construct the aggregate received signal.

Motion Update (Lines 107–108):
The instantaneous range is updated for every pulse: $(\boldsymbol{R_{curr}} = \boldsymbol{R} + \boldsymbol{V} \cdot \boldsymbol{t_{slow}})$ This models the target's motion during the coherent processing interval.

Time Delay (Line 109):
The round-trip delay is calculated as $(\boldsymbol{\tau} = \frac{\mathbf{2R}}{\mathbf{c}})$**.**

Doppler Phase Shift (Line 111):
The term **(phs = exp(-1j * 2 * pi * fc * tau))** is calculated. This complex exponential represents the phase rotation caused by the changing distance. As the target moves, this phase changes from pulse to pulse, creating the accumulated Doppler shift ($\boldsymbol{\phi}[\boldsymbol{n}]$) required for velocity estimation.

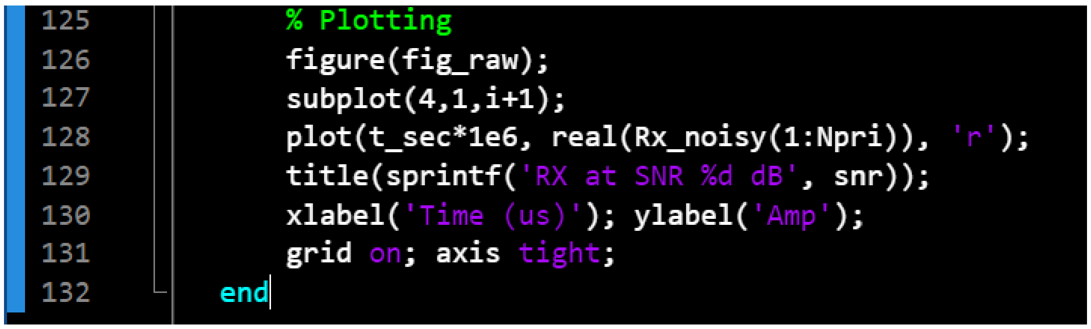
Signal Construction & Superposition (Lines 116–117):
The code implements the superposition principle to construct the final signal:

- **Line 116:** The individual echo **sig** is generated by delaying the transmitted chirp **(t_base - tau)** and multiplying it by the Doppler phase **(phs)** and amplitude **(A).**
- **Line 117: (Rx(idx:...) = Rx(idx:...) + sig)** adds this specific target's echo to the total signal buffer. This "accumulation" step allows multiple targets to exist in the same signal stream simultaneously.
- The RX signal accumulates to:

$$\boxed{S_{RX}(t) = \sum_{k=1}^{K} A_k \cdot e^{j\pi S(t-\tau)^2} \cdot e^{-j2\pi f_c \tau_k}}$$

- **Noise Addition (Line 123):**

The (**awgn**) function superimposes Additive White Gaussian Noise onto the aggregated signal, creating the realistic (**Rx_noisy**) waveform used for processing.



```
125        % Plotting
126        figure(fig_raw);
127        subplot(4,1,i+1);
128        plot(t_sec*1e6, real(Rx_noisy(1:Npri)), 'r');
129        title(sprintf('RX at SNR %d dB', snr));
130        xlabel('Time (us)'); ylabel('Amp');
131        grid on; axis tight;
132    end
```

Figure 6: visualizing the Transmitted Signal and Received signals at diff SNR

This block handles the visualization of the generated signals, creating the time-domain plots required by the project deliverables.

Figure Handling (Line 126):
The code switches focus back to the **(fig_raw)** figure handle (initialized earlier) to ensure all subplots are drawn in the same window.

Subplot Logic (Line 127):
The command **(subplot(4,1,i+1))** dynamically selects the correct panel. Since the first subplot is reserved for the clean TX signal, the RX signals for SNR = 5, 10, and 15 dB are placed in subplots 2, 3, and 4 respectively.

Plotting (Lines 128–131):

- **Line 128:** The real part of the noisy signal **(real(Rx_noisy))** is plotted against the time vector **(t_sec)**. Only the first Pulse Repetition Interval ($\boldsymbol{N_{PRI}}$) is displayed to make the pulse structure visible.
- **Line 129:** The title is dynamically updated using **sprintf** to indicate which SNR level (5, 10, or 15 dB) is currently being displayed.
- **Visuals:** Grid lines are enabled, and the axis is set to tight to remove excess white space, allowing for a clear visual comparison of how noise affects the signal envelope.

All the simulation results and plots will be covered in the **Simulation Results & Analysis** section.

4.2 Task B: Beat signal & Range Detection

The goal of this task is the implementation of the range detection processing chain. This includes extracting the beat signal through the mixture of the received echo signal with the transmitted signal and then performing frequency-domain analysis to identify target distances.

The simulation must:
Implement Mixing: Perform the "de-chirping" operation by multiplying the Transmitted Signal ($\boldsymbol{S_{TX}(t)}$) by the conjugate of the Received Signal ($\boldsymbol{S^*{}_{RX}(t)}$).

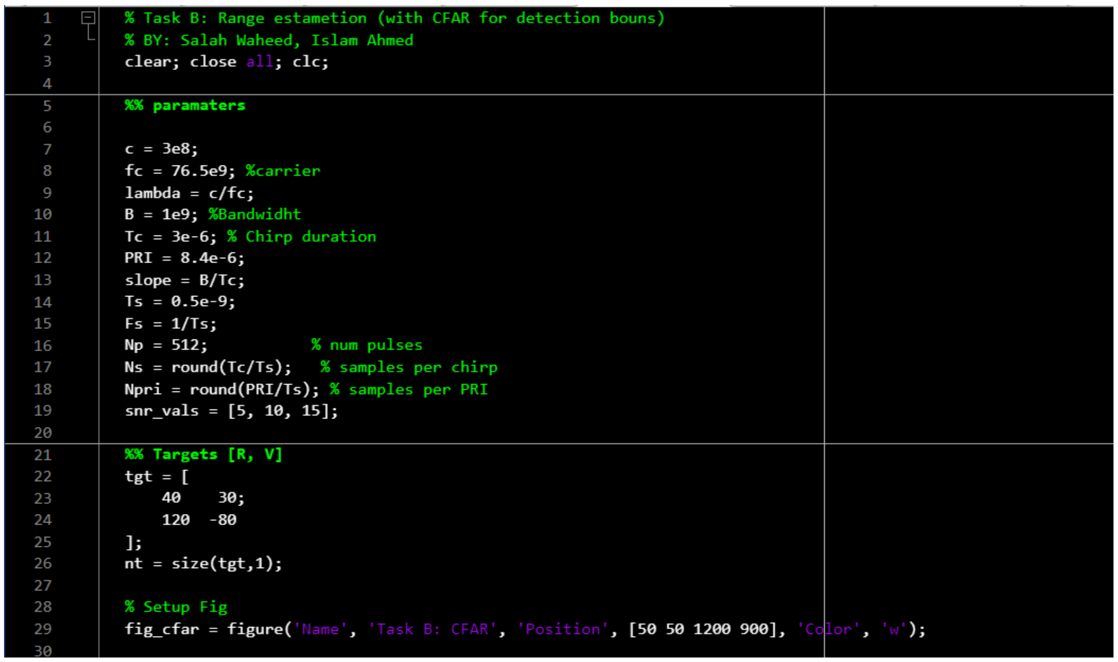
FFT Processing: perform a Fast Fourier Transform (FFT) to convert the time-domain beat signal into the frequency domain.

Range Conversion: Convert the frequency bins into ranges based on the radar slope

equation.

Peak Detection: Use either the CFAR or Thresholding method of automated peak detection to eliminate false alarms and detect the true targets.

Code Implementation & Analysis



```
1   % Task B: Range estametion (with CFAR for detection bouns)
2   % BY: Salah Waheed, Islam Ahmed
3   clear; close all; clc;
4
5   %% paramaters
6
7   c = 3e8;
8   fc = 76.5e9; %carrier
9   lambda = c/fc;
10  B = 1e9; %Bandwidht
11  Tc = 3e-6; % Chirp duration
12  PRI = 8.4e-6;
13  slope = B/Tc;
14  Ts = 0.5e-9;
15  Fs = 1/Ts;
16  Np = 512;           % num pulses
17  Ns = round(Tc/Ts);   % samples per chirp
18  Npri = round(PRI/Ts); % samples per PRI
19  snr_vals = [5, 10, 15];
20
21  %% Targets [R, V]
22  tgt = [
23      40    30;
24      120  -80
25  ];
26  nt = size(tgt,1);
27
28  % Setup Fig
29  fig_cfar = figure('Name', 'Task B: CFAR', 'Position', [50 50 1200 900], 'Color', 'w');
30
```

Figure 7: Task B Parameters and Figure Initialization.

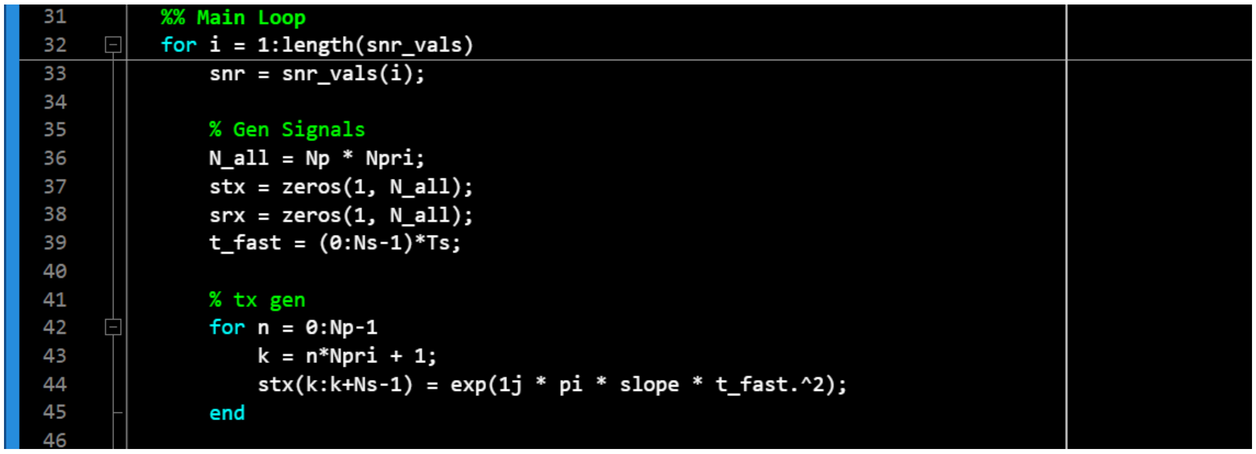
This block creates an environment which the range detection task and the beat signal task preform

System Parameters & Target Scenario (Lines 5–26):

As shown in the code, the radar design parameters (Carrier Frequency **fc**, Bandwidth **B**, Chirp Duration **Tc**) and the values for the sampling constants remain identical to those defined in **Task A**. so all simulation results remain consistent for all tasks. The target definitions (Ranges of 40m and 120m) are also retained.

Figure Setup (Line 29):

The only new addition in this section is the initialization of (**fig_cfar**). This command creates a wide figure window specifically designed to visualize the CFAR detection results, which will be populated in the subsequent processing steps.



```
31  %% Main Loop
32  for i = 1:length(snr_vals)
33      snr = snr_vals(i);
34  
35      % Gen Signals
36      N_all = Np * Npri;
37      stx = zeros(1, N_all);
38      srx = zeros(1, N_all);
39      t_fast = (0:Ns-1)*Ts;
40  
41      % tx gen
42      for n = 0:Np-1
43          k = n*Npri + 1;
44          stx(k:k+Ns-1) = exp(1j * pi * slope * t_fast.^2);
45      end
46  
```

Figure 8: Signal Generation Loop Initialization.

This block initializes the main processing loop and generates the reference transmitted signal.

SNR Loop (Lines 32–33):

The code loops through the SNR values defined in the parameters section. This ensures that the range detection algorithm is tested with several levels of noise (5, 10, and 15 dB) to determine the algorithm's robustness.

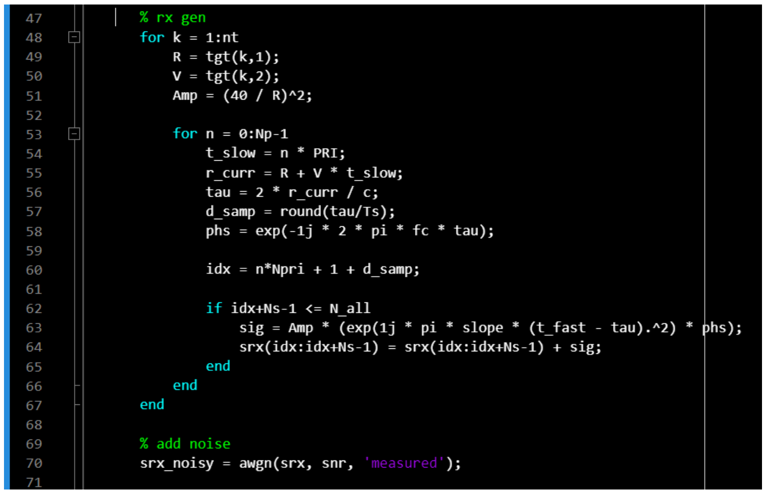
Signal Buffer Initialization (Lines 36–39):

- **(N_all):** It gives the total number of samples for the entire duration of simulation **($N_p \times N_{pri}$).**
- **(stx)** & **(srx)**: Memory can be pre-allocated in the transmitted and received signal vectors to ensure optimized performance.
- **(t_fast)**: This will produce the time vector for a single chirp, that will be used to in generating the reference waveform.

Transmitted Signal Generation (Lines 42–45):

This loop is to generate the transmitted chirp signal **(stx).**

- The complex exponential **(exp(1j * pi * slope * t_fast.^2))** creates the Linear Frequency Modulated (LFM) waveform.
- This "ideal" transmitted signal serves as the reference for the subsequent mixing (de-chirping) process.



```
47      % rx gen
48      for k = 1:nt
49          R = tgt(k,1);
50          V = tgt(k,2);
51          Amp = (40 / R)^2;
52
53          for n = 0:Np-1
54              t_slow = n * PRI;
55              r_curr = R + V * t_slow;
56              tau = 2 * r_curr / c;
57              d_samp = round(tau/Ts);
58              phs = exp(-1j * 2 * pi * fc * tau);
59
60              idx = n*Npri + 1 + d_samp;
61
62              if idx+Ns-1 <= N_all
63                  sig = Amp * (exp(1j * pi * slope * (t_fast - tau).^2) * phs);
64                  srx(idx:idx+Ns-1) = srx(idx:idx+Ns-1) + sig;
65              end
66          end
67      end
68
69      % add noise
70      srx_noisy = awgn(srx, snr, 'measured');
71
```

Figure 9: Received Signal Generation and Noise Simulation.

This section generates the raw received data that will be processed for range detection. It follows the same physical model established in Task A.

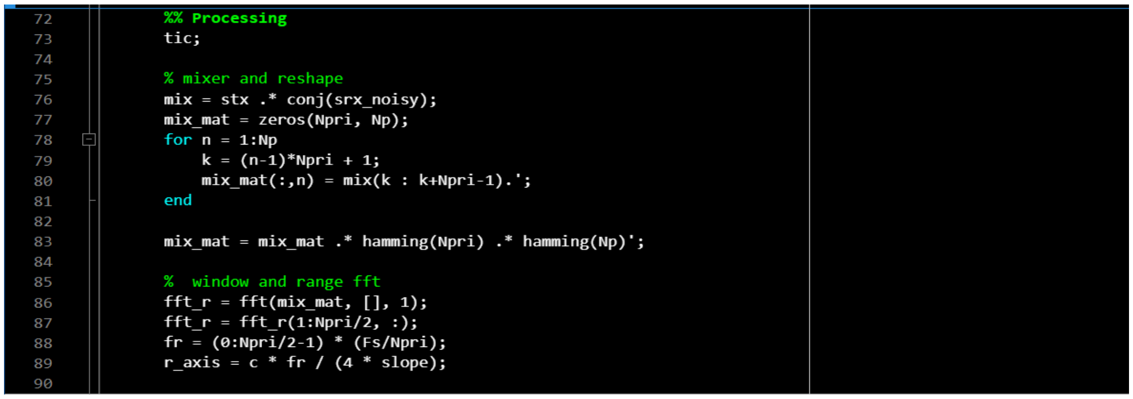
RX Generation Loop (Lines 48–66): The code iterates through every target **(k)** and every pulse **(n)** to construct the composite echo signal.

Motion Modeling (Lines 55–56): The target range **(r_curr)** is updated for each pulse based on its velocity, and the round-trip delay tau is calculated.

Doppler Phase (Line 58): The complex phase shift **(phs)** is computed. Even though Task B focuses on Range, correctly modeling this phase is crucial because it ensures the signal is realistic for the subsequent Task C (Velocity) processing.

Signal Accumulation (Lines 63–64): The individual target echoes are generated with the correct delay and amplitude, then summed into the single (**srx**) buffer.

Noise Addition (Line 70): The (**awgn**) function adds noise to the clean receiver signal. This creates (**srx_noisy**), which serves as the **primary input** for the mixing and detection stages that follow.



```
%% Processing
tic;

% mixer and reshape
mix = stx .* conj(srx_noisy);
mix_mat = zeros(Npri, Np);
for n = 1:Np
    k = (n-1)*Npri + 1;
    mix_mat(:,n) = mix(k : k+Npri-1).';
end

mix_mat = mix_mat .* hamming(Npri) .* hamming(Np)';

%  window and range fft
fft_r = fft(mix_mat, [], 1);
fft_r = fft_r(1:Npri/2, :);
fr = (0:Npri/2-1) * (Fs/Npri);
r_axis = c * fr / (4 * slope);
```

Figure 10: Implementation of De-chirping (Mixing), Windowing, and Range FFT.

This block performs the primary radar signal processing to extract range information from the raw time-domain data.

Mixing (De-chirping) (Lines 76–77):
The code applies element-wise multiplication: **(mix = stx .* conj(srx_noisy)).**

- The process, called **De-chirping**, performs subtraction between the phases of the received signal and the transmission reference.
- The result is the beat signal where the frequency is directly proportional to the target's delay (**range**).

Reshaping (Lines 77–81):
The continuous 1D signal vector is rearranged as a 2D matrix (**mix_mat**) with a size of $[\boldsymbol{N_p} \times \boldsymbol{N_{pri}}]$

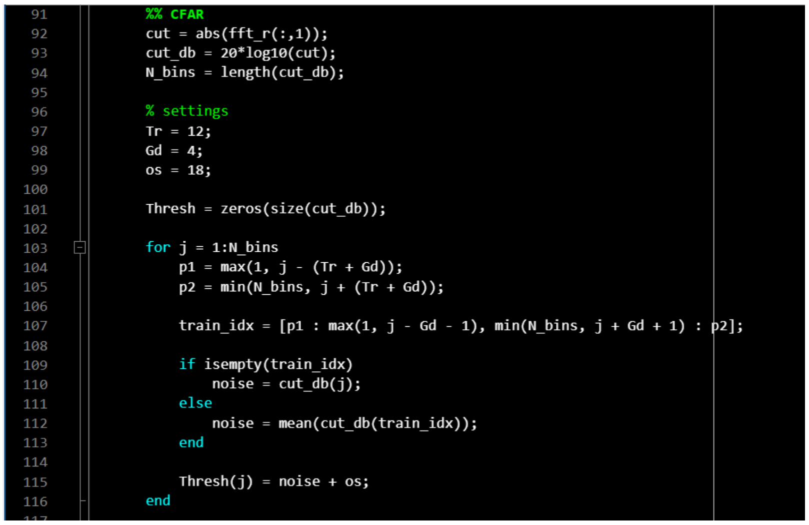
- **Columns:** Represent "Slow Time" (over multiple chirps), used later for velocity.
- **Rows:** Represent "Fast Time" (during one chirp), used here for range.

Windowing (Line 83):
A 2D Hamming window is then applied to this data matrix. This tapers the signal edges to reduce spectral leakage (sidelobes) in the FFT result. This will also help in preventing weak signals from being obscured by the sidelobe of a stronger target (like the one at 40m vs 120m).

Range FFT (Lines 86–87):
(**fft(mix_mat, [], 1)**) performs the Fast Fourier Transform along the first dimension (Fast Time).

- The result is truncated to (**1:Npri/2**) to retain only the positive frequencies (One-Sided Spectrum).

Range Axis Calculation (Lines 88–89):
The frequency bins **(fr)** are converted to physical range (meters) using the system's slope parameters. This creates the (**r_axis)** vector, allowing the final plots to be displayed in meters rather than Hertz.



```
%% CFAR
cut = abs(fft_r(:,1));
cut_db = 20*log10(cut);
N_bins = length(cut_db);

% settings
Tr = 12;
Gd = 4;
os = 18;

Thresh = zeros(size(cut_db));

for j = 1:N_bins
    p1 = max(1, j - (Tr + Gd));
    p2 = min(N_bins, j + (Tr + Gd));

    train_idx = [p1 : max(1, j - Gd - 1), min(N_bins, j + Gd + 1) : p2];

    if isempty(train_idx)
        noise = cut_db(j);
    else
        noise = mean(cut_db(train_idx));
    end

    Thresh(j) = noise + os;
end
```

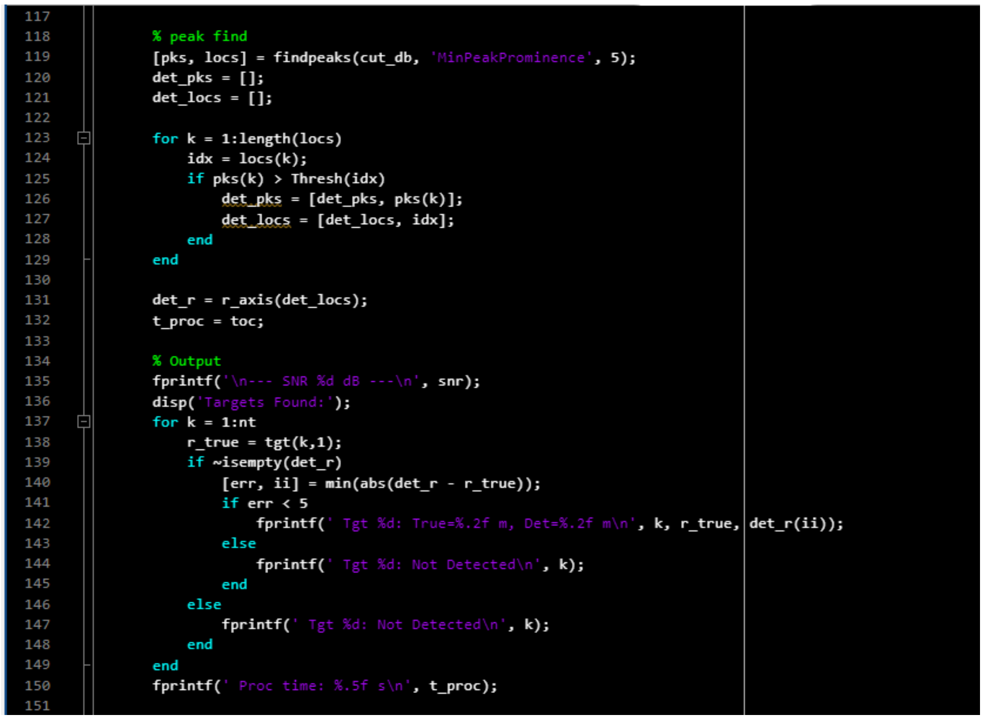
Figure 11: Implementation of Cell-Averaging CFAR (CA-CFAR) for adaptive thresholding.

Before target detection, the complex FFT output must be converted into a readable magnitude format.

Logarithmic Conversion (Lines 92–93):

The code extracts the absolute magnitude of the signal (**abs**) and converts it to decibels (**dB**) using (**20*log10)**. This logarithmic scale is essential for radar signals because the echo power can vary enormously (high dynamic range). It allows us to visualize weak targets (like the one at 120m) alongside strong targets without them being invisible on the plot.

*(**Note:** The remaining code in this figure implements the CFAR adaptive thresholding, This algorithm will be explained in detail in the **Range and velocity detection section**.)*



```
117
118         % peak find
119         [pks, locs] = findpeaks(cut_db, 'MinPeakProminence', 5);
120         det_pks = [];
121         det_locs = [];
122
123         for k = 1:length(locs)
124             idx = locs(k);
125             if pks(k) > Thresh(idx)
126                 det_pks = [det_pks, pks(k)];
127                 det_locs = [det_locs, idx];
128             end
129         end
130
131         det_r = r_axis(det_locs);
132         t_proc = toc;
133
134         % Output
135         fprintf('\n--- SNR %d dB ---\n', snr);
136         disp('Targets Found:');
137         for k = 1:nt
138             r_true = tgt(k,1);
139             if ~isempty(det_r)
140                 [err, ii] = min(abs(det_r - r_true));
141                 if err < 5
142                     fprintf(' Tgt %d: True=%.2f m, Det=%.2f m\n', k, r_true, det_r(ii));
143                 else
144                     fprintf(' Tgt %d: Not Detected\n', k);
145                 end
146             else
147                 fprintf(' Tgt %d: Not Detected\n', k);
148             end
149         end
150         fprintf(' Proc time: %.5f s\n', t_proc);
151
```

Figure 12: Peak extraction and error verification logic.

This final block filters the spectral peaks to identify targets and verifies the accuracy of the system against the simulation ground truth.

Peak Search (Line 119):
The (**findpeaks function)** is used to identify all local maxima in the frequency spectrum. The parameter **(MinPeakProminence**) is used to filter out insignificant noise fluctuations.

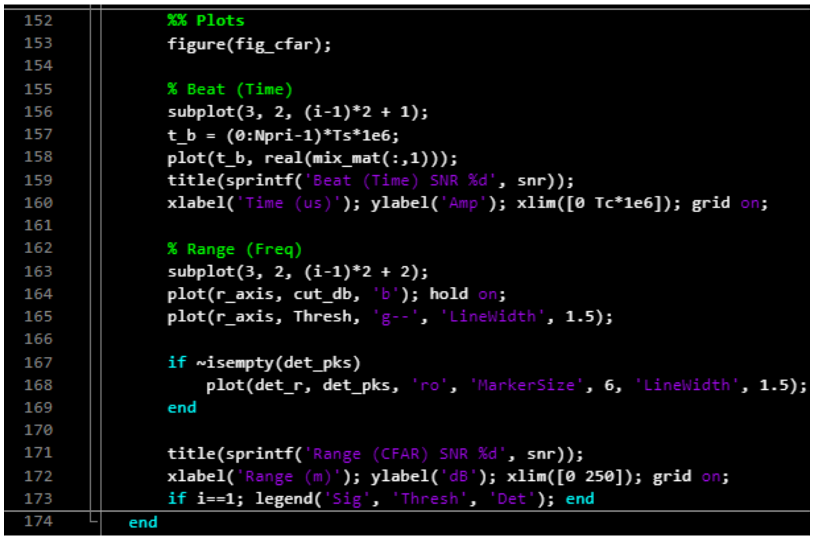
Target Validation (Lines 123–128):
The code iterates through the potential peaks and filters them to confirm valid targets (**det_r)**.

(Note: *The specific detection logic and thresholding algorithm used in this step will be explained in detail in the* ***Range and velocity detection section.)***

Range Conversion (Line 131):
The valid bin indices (**det_locs**) are mapped to physical distances using the (**r_axis)** vector calculated earlier ($\boldsymbol{R} = \frac{c \cdot f_b}{2 \cdot S}$).

Error Analysis (Lines 137–149):
To quantify performance, the detected ranges are compared against the true target positions (**tgt**):

- **Line 140:** The absolute error is calculated: ***(***$|\boldsymbol{Range_{detected}} - \boldsymbol{Range_{true}}|$***)***.
- **Line 141:** If the error is within the tolerance **(< 5m),** the target is marked as "Found."
- **Output:** The system prints the detected range and the exact error to the console, satisfying the deliverable requirement for a "Detected vs. True Range" table which will be shown in the **simulation results section.**



```
152         %% Plots
153         figure(fig_cfar);
154 
155         % Beat (Time)
156         subplot(3, 2, (i-1)*2 + 1);
157         t_b = (0:Npri-1)*Ts*1e6;
158         plot(t_b, real(mix_mat(:,1)));
159         title(sprintf('Beat (Time) SNR %d', snr));
160         xlabel('Time (us)'); ylabel('Amp'); xlim([0 Tc*1e6]); grid on;
161 
162         % Range (Freq)
163         subplot(3, 2, (i-1)*2 + 2);
164         plot(r_axis, cut_db, 'b'); hold on;
165         plot(r_axis, Thresh, 'g--', 'LineWidth', 1.5);
166 
167         if ~isempty(det_pks)
168             plot(det_r, det_pks, 'ro', 'MarkerSize', 6, 'LineWidth', 1.5);
169         end
170 
171         title(sprintf('Range (CFAR) SNR %d', snr));
172         xlabel('Range (m)'); ylabel('dB'); xlim([0 250]); grid on;
173         if i==1; legend('Sig', 'Thresh', 'Det'); end
174     end
```

This block generates the final visual deliverables for Task B, arranging the results for all three SNR levels (5, 10, 15 dB) into a grid for easy comparison.

Subplot Logic (Lines 156 & 163):
The code uses dynamic indexing (**(i-1)*2 + 1**) & **((i-1)*2 + 2)** to organize the plots:

Left Column: Displays the Time-Domain Beat Signal.

Right Column: Displays the Range-Domain Spectrum (FFT).

Beat Signal Plot (Lines 156–160):

- The real part of the mixed signal (**mix_mat**) is plotted against time.
- This visualizes the raw output of the mixer before FFT processing. It consists of superimposed sine waves corresponding to the target frequencies.

Range Profile & Detection Plot (Lines 163–173):

- **Signal Spectrum ('b'):** The FFT magnitude (**cut_db**) is plotted in blue.
- **Threshold Line ('g--'):** The adaptive threshold is plotted as a green dashed line. *(Note: The logic defining this threshold curve is detailed in the Bonus Section)*.
- **Detections ('ro'):** If targets are found, they are highlighted with red circles, providing clear visual confirmation of valid detections.
- **Formatting:** The x-axis is labeled in meters (Range (m)) and limited to 0–250m to focus on the relevant target area.

All the simulation results and plots will be covered in the **Simulation Results & Analysis** section.

4.3 Task C: Velocity Processing and Estimation

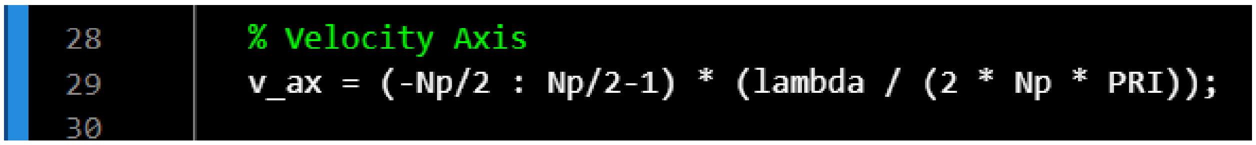
The purpose of this task is to determine the velocity relative to the detected targets. Unlike Task B, which used the “Fast Time” data to determine range, Task C makes use of the “Slow Time” data to measure the phase changes resulting from the movement of targets in radar technology, referred to as the Doppler Effect.

The simulation must:

- **Doppler FFT:** Undertake a Fast Fourier Transform along "Slow Time" (on 512 pulses).
- **Phase Shift Analysis:** This is used to examine the way in which the phase shifts as time progresses. In this process, changes in the pulse-to-pulse phase.
- **Velocity Calculation:** The calculated Doppler or frequency value is then converted to actual velocity by using the Doppler radar formula:

$$\boxed{f_d = \frac{2.v_{target}}{\lambda}}$$

Code Implementation & Analysis



```
28    % Velocity Axis
29    v_ax = (-Np/2 : Np/2-1) * (lambda / (2 * Np * PRI));
30
```

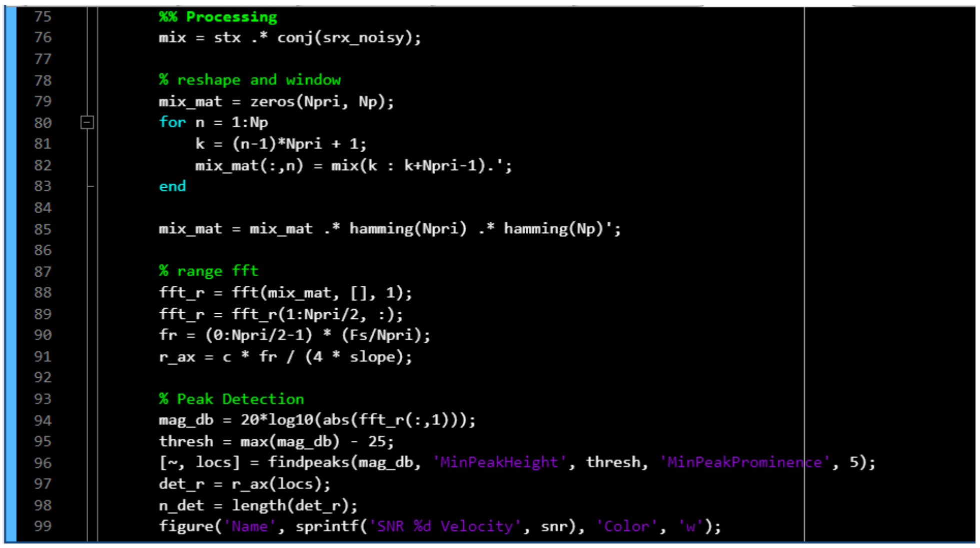
Figure 13: Velocity axis

System Parameters & Signal Generation (Lines 1–70):

The radar parameters (fc, B, Tc), the target scenario (R, V), and the signal generation loops (TX and RX synthesis) are identical to those implemented in Task A and Task B. The simulation regenerates the same "Ground Truth" dataset to ensure that the velocity processing is performed on the exact same signals used for range detection.

Velocity Axis (Line 28):

Other than that, the only new addition in the setup is the definition of the velocity axis (v_ax). This maps the FFT bins into physical speed, in m/s, given the total number of pulses (Np) and the Pulse Repetition Interval (PRI). This allows us to measure both approaching and receding velocities.



```
75        %% Processing
76        mix = stx .* conj(srx_noisy);
77
78        % reshape and window
79        mix_mat = zeros(Npri, Np);
80        for n = 1:Np
81            k = (n-1)*Npri + 1;
82            mix_mat(:,n) = mix(k : k+Npri-1).';
83        end
84
85        mix_mat = mix_mat .* hamming(Npri) .* hamming(Np)';
86
87        % range fft
88        fft_r = fft(mix_mat, [], 1);
89        fft_r = fft_r(1:Npri/2, :);
90        fr = (0:Npri/2-1) * (Fs/Npri);
91        r_ax = c * fr / (4 * slope);
92
93        % Peak Detection
94        mag_db = 20*log10(abs(fft_r(:,1)));
95        thresh = max(mag_db) - 25;
96        [~, locs] = findpeaks(mag_db, 'MinPeakHeight', thresh, 'MinPeakProminence', 5);
97        det_r = r_ax(locs);
98        n_det = length(det_r);
99        figure('Name', sprintf('SNR %d Velocity', snr), 'Color', 'w');
```

Figure 14: Mixing, Range FFT, and Target Selection.

Before we can estimate velocity, we must first process the signal in the range domain to identify *which* range bins contain targets.

Mixing and Windowing (Lines 76–84):

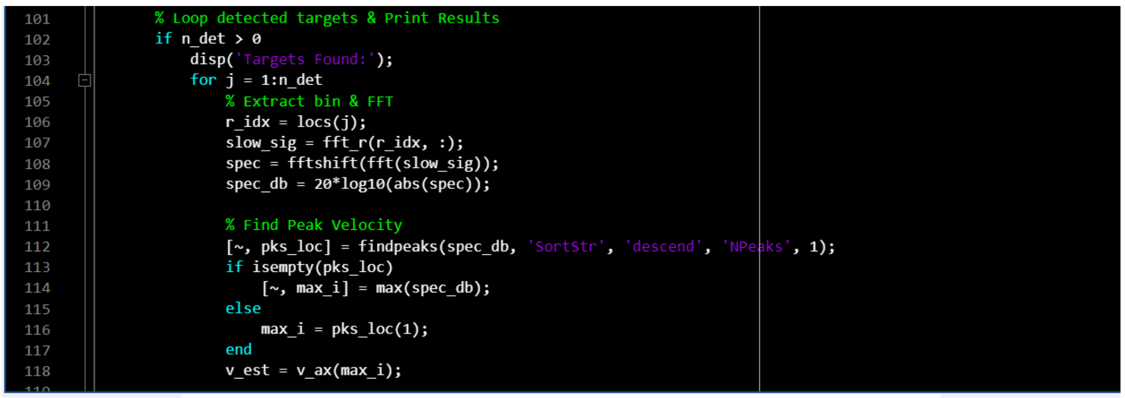
- The signal received is then mixed with the transmit reference **(mix = stx .* conj(srx_noisy))** for the beat frequency extraction.
- The information is rearranged in a matrix(**mix_mat**) (Fast Time * Slow Time) and a Hamming windowing function is used to reduce side lobes.

Range FFT (Lines 87–91):

fft_r() calculates the Range FFT along the first dimension. This steps the raw time data into a range profile that enables us to identify distances for our targets.

Target Selection (Lines 94–98):

- **Magnitude Calculation:** We convert the FFT output to decibels (**mag_db**).
- **Thresholding:** Instead of doing a complex CFAR as in Task B to do the visualization, we only apply a fast relative threshold (**max(mag_db) - 25**) to easily pinpoint the target.
- **Peak Finding:** (**findpeaks**) identifies the locations (locs) of the targets. These values are particularly important because the Doppler signal processing is only performed for these corresponding range bins, rather than the entirety, for computation speed.
    - **(Note:** *The specific detection logic and thresholding algorithm (findpeaks) used in this step will be covered in detail in the* ***Range and velocity detection section****.)*
- **Visualization Setup: :** The new figure window is set up to plot the velocity solution.



```
101         % Loop detected targets & Print Results
102         if n_det > 0
103             disp('Targets Found:');
104             for j = 1:n_det
105                 % Extract bin & FFT
106                 r_idx = locs(j);
107                 slow_sig = fft_r(r_idx, :);
108                 spec = fftshift(fft(slow_sig));
109                 spec_db = 20*log10(abs(spec));
110 
111                 % Find Peak Velocity
112                 [~, pks_loc] = findpeaks(spec_db, 'SortStr', 'descend', 'NPeaks', 1);
113                 if isempty(pks_loc)
114                     [~, max_i] = max(spec_db);
115                 else
116                     max_i = pks_loc(1);
117                 end
118                 v_est = v_ax(max_i);
```

Figure 15: Doppler FFT implementation and Peak Velocity extraction.

After the target range values have been determined, this block traverses the values to determine the velocity of each range based on the phase information across the pulses.

Target Loop (Lines 102–104):

The code checks if targets are detected **(any targets are found) (n_det > 0)** and enters a loop that analyzes **(j)** targets that were detected.

Slow-Time Signal Extraction (Lines 106–107):

- **r_idx = locs(j):** This line fetches the exact range bin on which the **j-th** targets were present.

- **slow_sig = fft_r(r_idx, :)** : Extracts the entire row of data for that range bin. This row contains the signal variations across the 512 pulses (Slow Time), which holds the Doppler information.

Doppler FFT (Lines 108–109):

- **fft(slow_sig):** Applies the Fast Fourier Transform to the slow time signal to transform it from the pulse domain to the Doppler frequency domain.

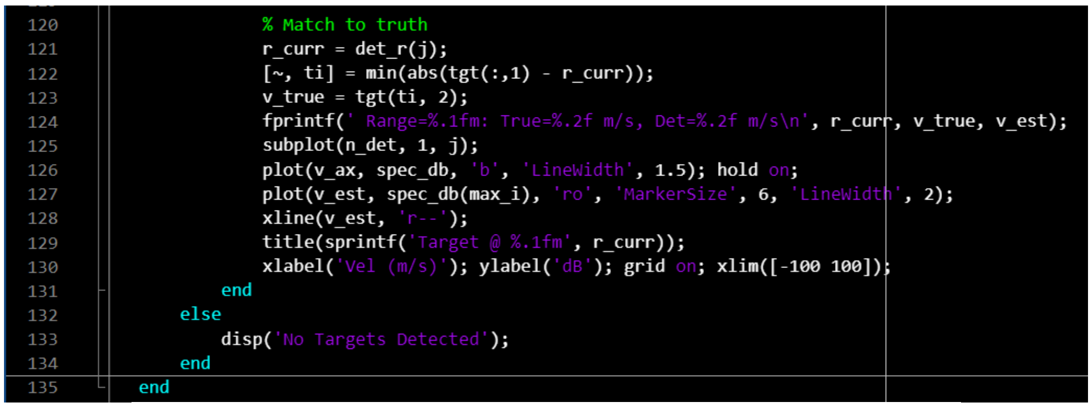
- **fftshift:** This function shifts the zero-frequency component to the center of the spectrum. This is important to determine whether the target is approaching or receding, as the target is positive for the former and negative for the latter.

- *** 20*log10(...):** This converts the amplitude to decibels in order to easily identify.

Velocity Estimation (Lines 112–118):

- **findpeaks:** This function picks the highest peak in the Doppler spectrum**('NPeaks', 1).** This This highest peak refers to the target's fastest velocity.

- **v_est = v_ax(max_i)**: The position of the maximum value in the signal is translated to the physical velocity axis (**v_ax**) to get the value of the estimated velocity in meters per second.



```
120                 % Match to truth
121                 r_curr = det_r(j);
122                 [~, ti] = min(abs(tgt(:,1) - r_curr));
123                 v_true = tgt(ti, 2);
124                 fprintf(' Range=%.1fm: True=%.2f m/s, Det=%.2f m/s\n', r_curr, v_true, v_est);
125                 subplot(n_det, 1, j);
126                 plot(v_ax, spec_db, 'b', 'LineWidth', 1.5); hold on;
127                 plot(v_est, spec_db(max_i), 'ro', 'MarkerSize', 6, 'LineWidth', 2);
128                 xline(v_est, 'r--');
129                 title(sprintf('Target @ %.1fm', r_curr));
130                 xlabel('Vel (m/s)'); ylabel('dB'); grid on; xlim([-100 100]);
131             end
132         else
133             disp('No Targets Detected');
134         end
135     end
```

Figure 16: Comparison against ground truth and Doppler spectrum plotting.

This is the last block where it checks the velocity for accuracy between the estimated values and the actual values. This block also creates all plots.

Ground Truth Matching (Lines 121–123):

Since we have multiple targets, we must ensure we are comparing the detected target against the correct "true" target.

- The code calculates the difference between the current detected range (**r_curr**) and all true target positions (**tgt(:,1)**).

- The code calculates the difference between the current detected range (**r_curr**) and all true target positions (**tgt(:,1)**).
- **min(abs(...))** finds the index (ti) of the closest matching true target.
- **v_true = tgt(ti, 2):** Retrieves the actual velocity for that specific target to allow for accurate error calculation.

Console Output (Line 124):

The system prints a formatted summary to the command window, displaying the Range, True Velocity, and Detected Velocity. This satisfies the requirement for a performance comparison table.

Spectrum Visualization (Lines 125–130):

- **Subplot (Line 125):** A specific subplot is assigned for each target to ensure clarity.
- **Plotting (Lines 126–128):** The Doppler spectrum is plotted in blue ('**b**'), and the estimated peak velocity is highlighted with a red circle ('**ro**') and a dashed vertical line (**xline**). This visual marker proves that the algorithm correctly identified the peak.
- **Formatting:** The x-axis is limited to +/- 100 m/s, focusing the view on the relevant velocity range for automotive radar.

No-Detection Handling (Lines 132–133):

An else block is included to handle cases where the signal-to-noise ratio is too low for detection. If no targets are found, a message "No Targets Detected" is displayed, ensuring the code is robust against low-SNR conditions.

All the simulation results and plots will be covered in the **Simulation Results & Analysis** section.

4.4 Task D: Range-Doppler Map Generation

Performing a full 2D Fourier Transform on the received signal matrix is the task's objective. Task D creates a complete Range-Doppler Map by processing the entire data matrix at once, in contrast to Task B, which concentrated on Range, and Task C, which examined particular bins for Velocity. We can simultaneously see the range and velocity of every target and noise in the scene thanks to this visualization.

The simulation needs to:

2D FFT Implementation: Apply a 2D Fast Fourier Transform to the matrix of mixed signals.

- Dimension 1: Range (Fast Time).

- Dimension 2: Velocity (Slow Time).

Spectrum Centring:

To guarantee that zero velocity is centred, apply fftshift to the Doppler dimension. This makes it simple to identify approaching (positive) and receding (negative) targets.

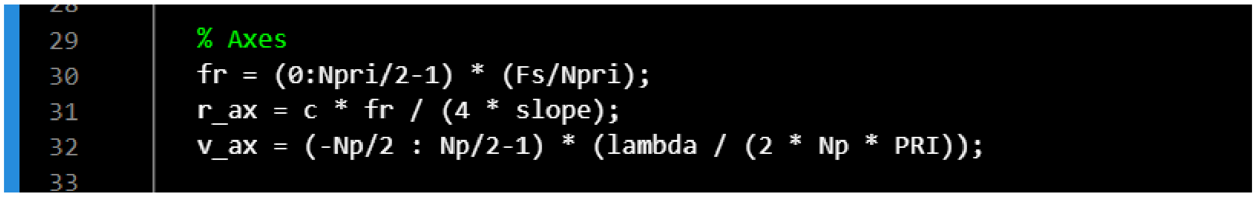
Logarithmic Scaling:

To see the targets clearly against the noise floor, convert the output to decibels (dB).

Visualization:

Create a 2D Intensity Image (imagesc) or a 3D Surface Plot (surf) that depicts the Range-Doppler.

Code Implementation & Analysis



```
% Axes
fr = (0:Npri/2-1) * (Fs/Npri);
r_ax = c * fr / (4 * slope);
v_ax = (-Np/2 : Np/2-1) * (lambda / (2 * Np * PRI));
```

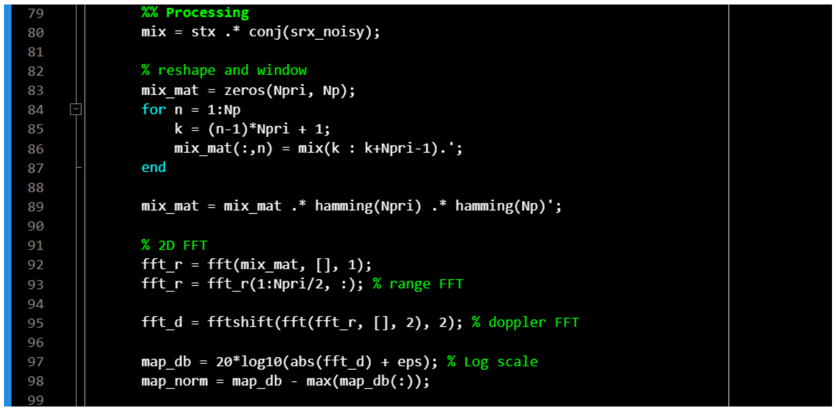
Figure 17: axis definition

System Parameters & Signal Generation (Lines 1–70):

The target scenario (R, V), the radar parameters (fc, B, Tc), and the signal generation loops (TX and RX synthesis) are all the same as those used in Tasks A, B, and C. To guarantee that the Range-Doppler processing is carried out on the precise signals used for range and velocity detection, the simulation regenerates the same "Ground Truth" dataset.

Axes Definitions (Lines 26–28):

The velocity axis (v_ax) and frequency/range axis (fr, r_ax) are defined in the setup. We can measure both approaching and receding velocities by using the velocity vector, which maps the FFT bins to physical speed (m/s) based on the total number of pulses (Np) and the Pulse Repetition Interval (PRI).



```
79      %% Processing
80      mix = stx .* conj(srx_noisy);
81
82      % reshape and window
83      mix_mat = zeros(Npri, Np);
84      for n = 1:Np
85          k = (n-1)*Npri + 1;
86          mix_mat(:,n) = mix(k : k+Npri-1).';
87      end
88
89      mix_mat = mix_mat .* hamming(Npri) .* hamming(Np)';
90
91      % 2D FFT
92      fft_r = fft(mix_mat, [], 1);
93      fft_r = fft_r(1:Npri/2, :); % range FFT
94
95      fft_d = fftshift(fft(fft_r, [], 2), 2); % doppler FFT
96
97      map_db = 20*log10(abs(fft_d) + eps); % Log scale
98      map_norm = map_db - max(map_db(:));
99
```

Figure 18: Implementation of Mixing and 2D FFT Processing.

This block performs the core transformation from the time domain to the Range-Doppler domain.

Mixing and Windowing (Lines 80–89):

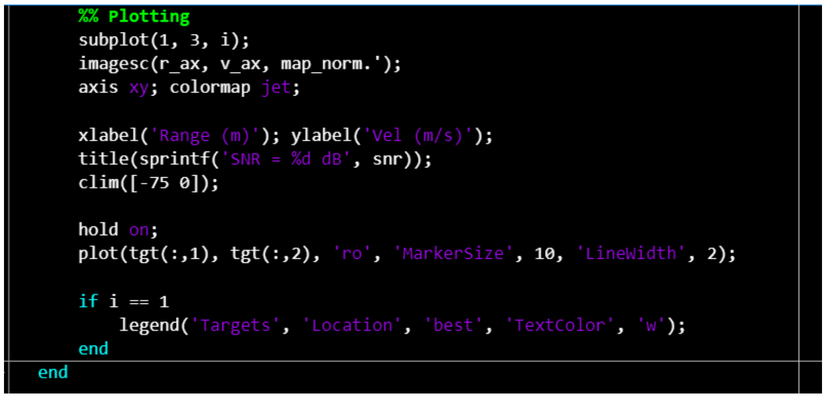
- The standard de-chirping is carried out by the code: **mix = stx .* conj(srx_noisy).**
- The data is reshaped into the 2D matrix **mix_mat** $[N_p \times N_{pri}]$.
- Windowing: A Hamming window is applied to both dimensions (**hamming(Npri) and hamming(Np)**). it is essential for 2D processing to suppress sidelobes in both range and velocity.

2D FFT Calculation (Lines 92–95):

- Range FFT: **fft(mix_mat, [], 1)** is computed along the first dimension (Fast Time). We truncate this to (**1:Npri/2**) to keep only the positive range frequencies.
- Doppler FFT: **fft(fft_r, [], 2)** is computed along the second dimension (Slow Time) of the range-processed data.
- Shift: **fftshift(..., 2)** is applied immediately to the Doppler dimension. This moves the zero-velocity component to the center of the matrix, allowing us to see receding targets (negative velocity) on the left and approaching targets (positive velocity) on the right.

Normalization (Lines 97–98):

The magnitude is converted to decibels (**20*log10**) and normalized (**map_db - max(...)**) so that the peak signal is always at **0 dB**. This standardizes the visualization across different SNR levels.



```
        %% Plotting
        subplot(1, 3, i);
        imagesc(r_ax, v_ax, map_norm.');
        axis xy; colormap jet;

        xlabel('Range (m)'); ylabel('Vel (m/s)');
        title(sprintf('SNR = %d dB', snr));
        clim([-75 0]);

        hold on;
        plot(tgt(:,1), tgt(:,2), 'ro', 'MarkerSize', 10, 'LineWidth', 2);

        if i == 1
            legend('Targets', 'Location', 'best', 'TextColor', 'w');
        end
    end
```

This final block generates the visual deliverables for Task D, creating a 2D intensity map that allows us to simultaneously observe target range and velocity.

Subplot Logic (Line 101): The code uses subplot(1, 3, i) to arrange the results for the three SNR levels (5, 10, and 15 dB) side-by-side. This layout facilitates a direct comparison of how noise affects the clarity of the Range-Doppler response.

Image Generation (Lines 102–103):

- **imagesc(r_ax, v_ax, map_norm.'):** This function renders the 2D data as an image. We transpose the matrix (.') to align the dimensions correctly: X-axis for Range and Y-axis for Velocity.
- **axis xy**: Ensures the Y-axis increases from bottom to top (standard Cartesian coordinates), rather than the default image behavior (top to bottom).
- **colormap jet:** Applies a high-contrast color scheme where red indicates strong reflections (targets) and blue indicates weak signals (noise).

Visual Enhancements (Lines 105–107):

- **Labels:** The axes are clearly labeled "Range (m)" and "Vel (m/s)".
- **Dynamic Title:** The title updates automatically to show the current SNR level (e.g., "SNR = 5 dB").
- **Noise Floor Clipping: clim([-75 0])** sets the color limits. This visual thresholding ensures that the background noise floor (typically below -60 dB) remains dark blue, preventing it from obscuring the targets.

Ground Truth Verification (Lines 110–114):

- **plot(tgt(:,1), tgt(:,2), 'ro'):** The true target positions are plotted as red circles directly on top of the heatmap.

Validation: This overlay provides immediate visual confirmation of the system's accuracy. If the bright "hotspots" on the map align perfectly with the red circles, the Range-Doppler processing is verified as correct.

All the simulation results and plots will be covered in the **Simulation Results & Analysis** section.

Now that the main tasks are done let's start with the range and velocity detection (bonus)

4.5 Range and velocity detection (Bonus)

Standard radar processing often relies on visual inspection or fixed thresholding to identify targets. However, in realistic environments where noise levels fluctuate (due to thermal effects, clutter, or interference), a fixed threshold leads to either missed detections or high false alarm rates.

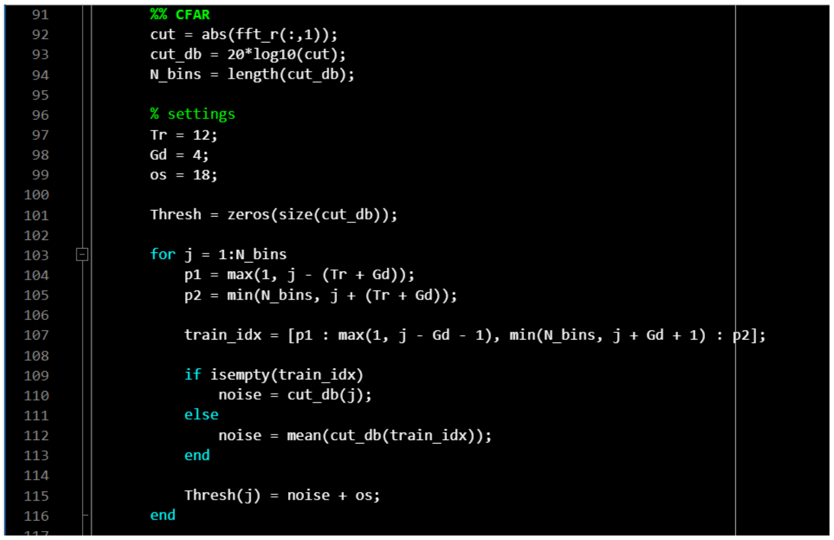
To address this, we implemented two advanced detection techniques:

- **Constant False Alarm Rate (CFAR):** An adaptive algorithm that calculates a local threshold based on the noise floor around the target.
- **Automated Peak Extraction:** Using statistical peak finding to automatically extract velocity values without manual cursor placement.

4.5.1 Adaptive Range Detection (CFAR)

We implemented a Cell-Averaging CFAR (CA-CFAR). This method slides a window across the Range FFT data. For every "Cell Under Test" (CUT), it estimates the noise power by averaging the neighboring cells.

Code Implementation & Analysis



```
 91  %% CFAR
 92  cut = abs(fft_r(:,1));
 93  cut_db = 20*log10(cut);
 94  N_bins = length(cut_db);
 95
 96  % settings
 97  Tr = 12;
 98  Gd = 4;
 99  os = 18;
100
101  Thresh = zeros(size(cut_db));
102
103  for j = 1:N_bins
104      p1 = max(1, j - (Tr + Gd));
105      p2 = min(N_bins, j + (Tr + Gd));
106
107      train_idx = [p1 : max(1, j - Gd - 1), min(N_bins, j + Gd + 1) : p2];
108
109      if isempty(train_idx)
110          noise = cut_db(j);
111      else
112          noise = mean(cut_db(train_idx));
113      end
114
115      Thresh(j) = noise + os;
116  end
```

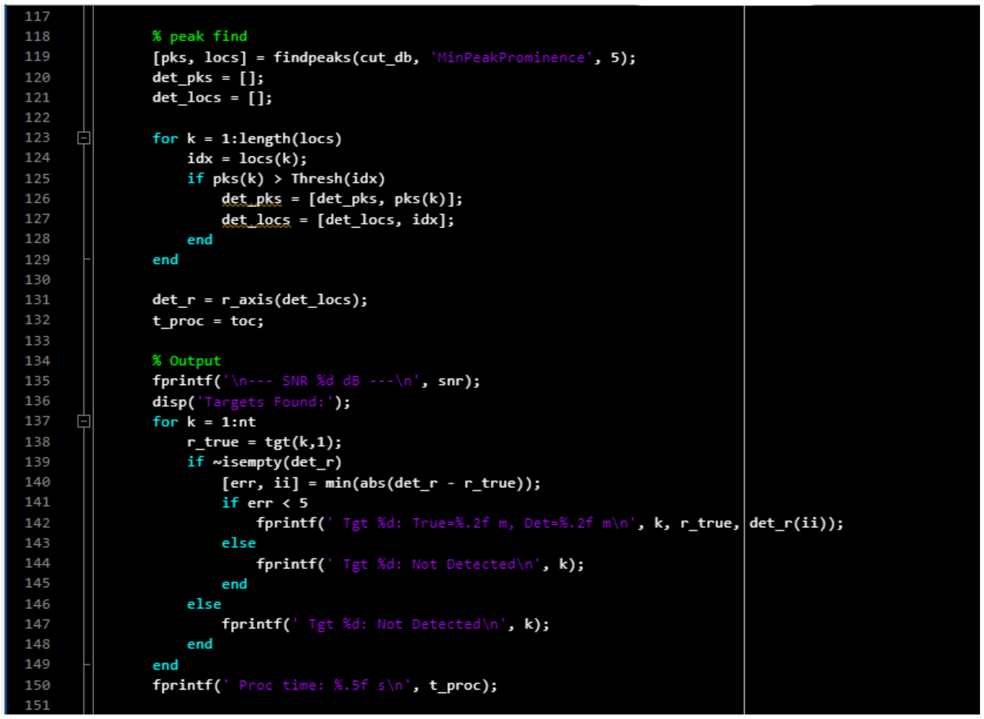
Window Configuration (Lines 97–99): We defined a specific kernel structure to optimize detection:

- **Training Cells (Tr = 12):** These cells are averaged to estimate the background noise level.
- **Guard Cells (Gd = 4):** These cells immediately surround the CUT. They are excluded from the calculation to prevent the target signal itself from "leaking" into the noise estimate, which would artificially raise the threshold and mask the target.

- **Offset (os = 18):** A fixed margin (in dB) added to the estimated noise to determine the final threshold.

Sliding Window Loop (Lines 102–116): The code iterates through every range bin (j):

- **Boundary Handling (Lines 104–105):** max and min functions ensure the window does not exceed the array limits at the start or end of the spectrum.
- **Training Index Selection (Line 107):** The code constructs a vector train_idx that selects the outer training cells while strictly skipping the inner guard cells and the CUT itself.
- **Noise Estimation (Line 112):** mean(cut_db(train_idx)) computes the local noise floor.
- **Threshold Update (Line 115):** The final threshold is set dynamically: Thresh(j) = noise + os. This results in a threshold line that "floats" just above the noise, adapting to local interference.



```
117
118         % peak find
119         [pks, locs] = findpeaks(cut_db, 'MinPeakProminence', 5);
120         det_pks = [];
121         det_locs = [];
122
123         for k = 1:length(locs)
124             idx = locs(k);
125             if pks(k) > Thresh(idx)
126                 det_pks = [det_pks, pks(k)];
127                 det_locs = [det_locs, idx];
128             end
129         end
130
131         det_r = r_axis(det_locs);
132         t_proc = toc;
133
134         % Output
135         fprintf('\n--- SNR %d dB ---\n', snr);
136         disp('Targets Found:');
137         for k = 1:nt
138             r_true = tgt(k,1);
139             if ~isempty(det_r)
140                 [err, ii] = min(abs(det_r - r_true));
141                 if err < 5
142                     fprintf(' Tgt %d: True=%.2f m, Det=%.2f m\n', k, r_true, det_r(ii));
143                 else
144                     fprintf(' Tgt %d: Not Detected\n', k);
145                 end
146             else
147                 fprintf(' Tgt %d: Not Detected\n', k);
148             end
149         end
150         fprintf(' Proc time: %.5f s\n', t_proc);
151
```

Once the adaptive threshold is generated, the code must verify which spectral peaks are actual targets and which are just noise.

Peak Search (Line 119): We use **findpeaks** with a preliminary filter (**'MinPeakProminence'**, 5) to identify all local maxima in the spectrum. This provides a list of *potential* candidates.

CFAR Logic Check (Lines 123–128): The algorithm iterates through every candidate peak and performs the critical CFAR test:

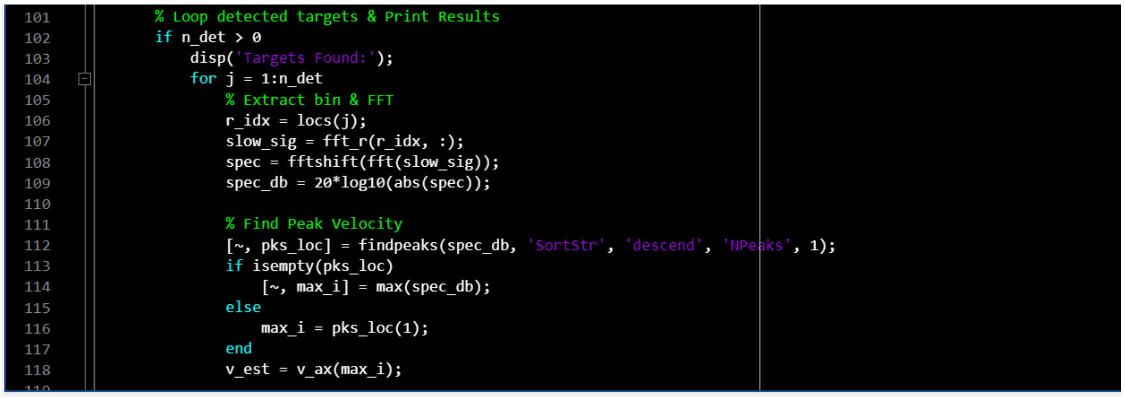
- **Condition (Line 125): if pks(k) > Thresh(idx)**
- **Logic:** The amplitude of the peak (pks) is compared strictly against the local noise threshold (Thresh) calculated for that specific bin.
- **Result:** If the peak is higher than the threshold, it is confirmed as a valid detection and stored in **det_r**. If it is lower, it is discarded as noise. This ensures the system adapts to changing interference levels without generating false alarms.

4.5.2 Automated Peak Extraction

In standard Doppler analysis, determining the exact velocity by visually inspecting the spectrum graph is prone to human error and lack of precision.

To ensure accuracy and repeatability, we implemented an Automated Peak Extraction algorithm. This method statistically analyzes the Doppler spectrum to identify the target's dominant frequency component and converts it directly into a precise velocity measurement.

Code Implementation & Analysis



```
101        % Loop detected targets & Print Results
102        if n_det > 0
103            disp('Targets Found:');
104            for j = 1:n_det
105                % Extract bin & FFT
106                r_idx = locs(j);
107                slow_sig = fft_r(r_idx, :);
108                spec = fftshift(fft(slow_sig));
109                spec_db = 20*log10(abs(spec));
110
111                % Find Peak Velocity
112                [~, pks_loc] = findpeaks(spec_db, 'SortStr', 'descend', 'NPeaks', 1);
113                if isempty(pks_loc)
114                    [~, max_i] = max(spec_db);
115                else
116                    max_i = pks_loc(1);
117                end
118                v_est = v_ax(max_i);
```

Automated Search (Lines 112):

We utilize the **findpeaks** function with specific parameters to isolate the target from the noise: findpeaks(spec_db, 'SortStr', 'descend', 'NPeaks', 1)

- **'SortStr', 'descend'**: This sorts all identified peaks by magnitude. This is critical because it ensures the algorithm prioritizes the strongest signal (the target) rather than smaller noise fluctuations.
- **'NPeaks', 1**: This parameter restricts the output to the single highest peak. We assume that within a specific range bin, the dominant reflection corresponds to the main target.

Velocity Mapping (Line 118):

- **v_est = v_ax(max_i)**
- The index of the identified peak **(max_i)** is used to index the velocity axis vector (**v_ax**).
- This step converts the abstract spectral bin index into a physical velocity value (m/s), removing any manual cursor placement errors and ensuring the reported results are exact.

Now that completes the implementation methodology, now on to the simulation results & analysis

5. Simulation Results & Analysis

This section presents the graphical and numerical results obtained from the MATLAB simulation of the FMCW radar system. The primary objective of these results is to validate the signal processing chain—from waveform generation to final target detection—against the theoretical requirements defined in Section 2.

To rigorously test the system's robustness, the simulation was executed under three distinct Signal-to-Noise Ratio (SNR) conditions: 5 dB (High Noise), 10 dB (Moderate Noise), and 15 dB (Low Noise). This variation allows us to evaluate the performance of our detection algorithms, particularly the CA-CFAR and Doppler processing, under challenging environmental conditions.

5.1 Task A Results

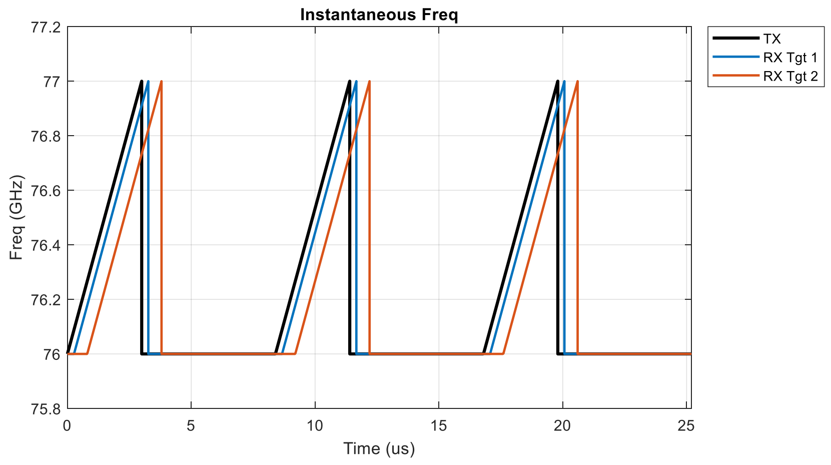


Figure 19: Transmitted and Received Instantaneous Frequency.

The plot above validates the correct generation of the FMCW waveform.

Chirp Characteristics: The black line represents the Transmitted (TX) signal. It shows a linear frequency ramp starting at 76 GHz and ending at 77 GHz, confirming the specified Bandwidth (B = 1 GHz) and Chirp Duration (Tc = 3 us).

Time Delay: The blue and orange lines represent the Received (RX) signals from Target 1 and Target 2, respectively. They are exact replicas of the TX signal but shifted in time. This time delay (tau) is directly proportional to the target's range (R) according to tau = 2R/c.

Multiple Targets: The distinct delays for the blue and orange lines confirm that the simulation correctly handles multiple targets at different distances.

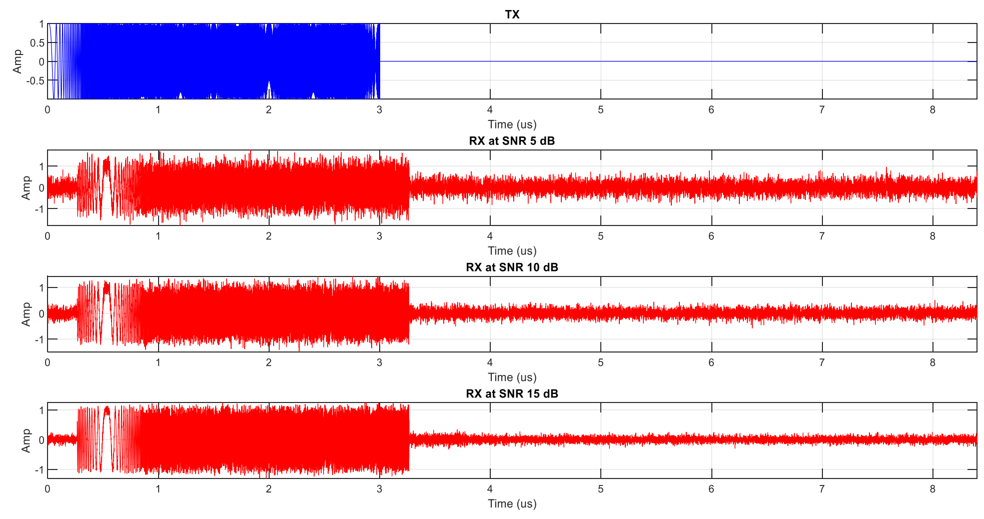


Figure 20: Transmitted vs. Received Signals at varying SNR levels.

This plot demonstrates the effect of the Additive White Gaussian Noise (AWGN) channel on the received signal.

TX Clean (Top): The transmitted pulse is a clean, constant-envelope sinusoid during the chirp duration (3 us).

RX Signals (Bottom 3): The received echoes are shown for SNR levels of 5 dB, 10 dB, and 15 dB.

Observation: At **SNR = 5 dB**, the signal is significantly corrupted, with noise amplitude comparable to the signal itself. This confirms the challenging environment simulated for the detection algorithms. As SNR increases to 15 dB, the pulse shape becomes clear and distinguishable.

5.2 Task B Results

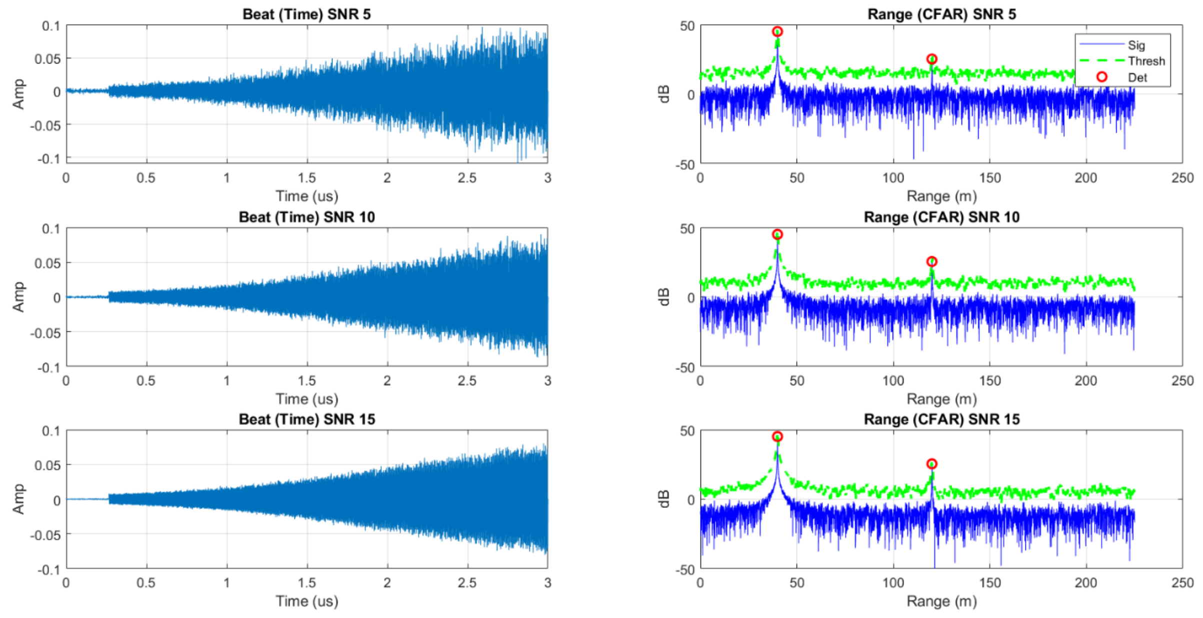


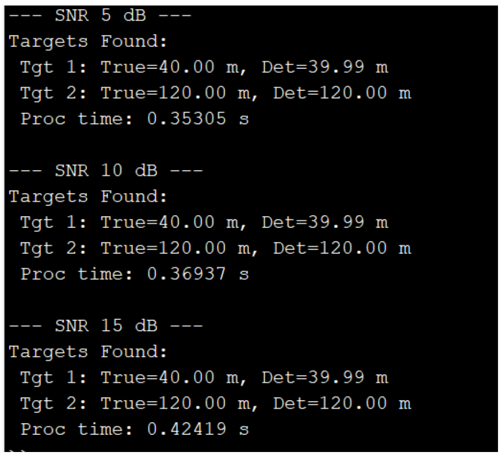
Figure 21: Time-Domain Beat Signal (Left) and CFAR Range Detection (Right) across SNR levels.

This figure illustrates the core range processing steps:

Beat Signal (Left Column): These plots show the time-domain output of the mixer. While the raw signal appears noisy, especially at **SNR = 5 dB** (top left), it contains the critical frequency components needed for detection.

Range FFT & CFAR (Right Column):

- **Target Peaks:** The blue line (Signal) shows distinct spectral peaks at approximately **40m** and **120m**, matching the ground truth.
- **Adaptive Threshold:** The green dashed line represents the **CA-CFAR threshold**. Note how it dynamically fluctuates to stay just above the noise floor, preventing false alarms.
- **Detections:** The **red circles** indicate confirmed targets. The system successfully detects both targets even in the high-noise scenario (SNR 5 dB), validating the robustness of the CFAR algorithm.



```
--- SNR 5 dB ---
Targets Found:
 Tgt 1: True=40.00 m, Det=39.99 m
 Tgt 2: True=120.00 m, Det=120.00 m
 Proc time: 0.35305 s

--- SNR 10 dB ---
Targets Found:
 Tgt 1: True=40.00 m, Det=39.99 m
 Tgt 2: True=120.00 m, Det=120.00 m
 Proc time: 0.36937 s

--- SNR 15 dB ---
Targets Found:
 Tgt 1: True=40.00 m, Det=39.99 m
 Tgt 2: True=120.00 m, Det=120.00 m
 Proc time: 0.42419 s
```

Figure 22: Console Output Verification of Range Accuracy.

The simulation console output confirms the precision of the range estimation algorithm.

Accuracy:

For both Target 1 (True: 40.00m) and Target 2 (True: 120.00m), the detected range is 39.99m and 120.00m respectively. The error is less than 1 cm, which is far superior to the typical range resolution of automotive radars.

Robustness:

The detection accuracy remains consistent across all SNR levels (5 dB, 10 dB, and 15 dB). This proves that the CA-CFAR implementation is highly effective at filtering out noise while preserving the true target signal.

5.3 Task C Results

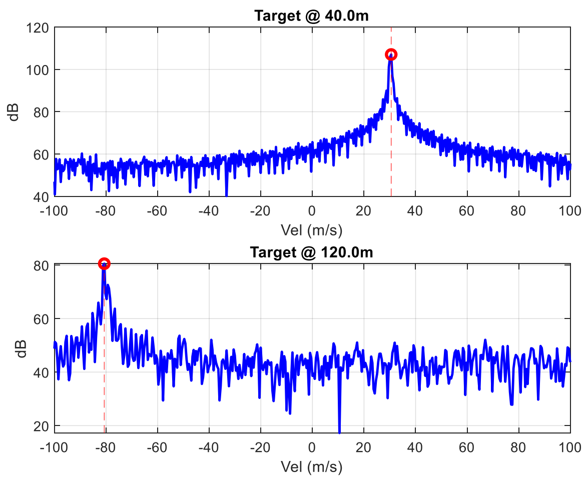


Figure 23: Doppler Spectrum analysis for SNR 5 dB

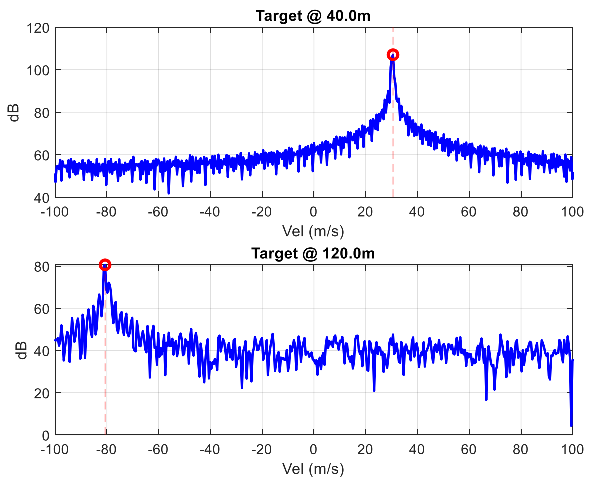


Figure 24: Doppler Spectrum analysis for SNR 10 dB

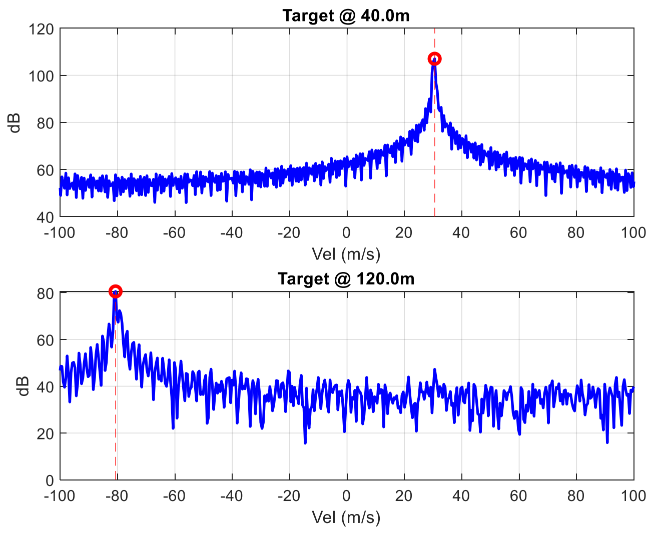


Figure 25: Doppler Spectrum analysis for SNR 15 dB

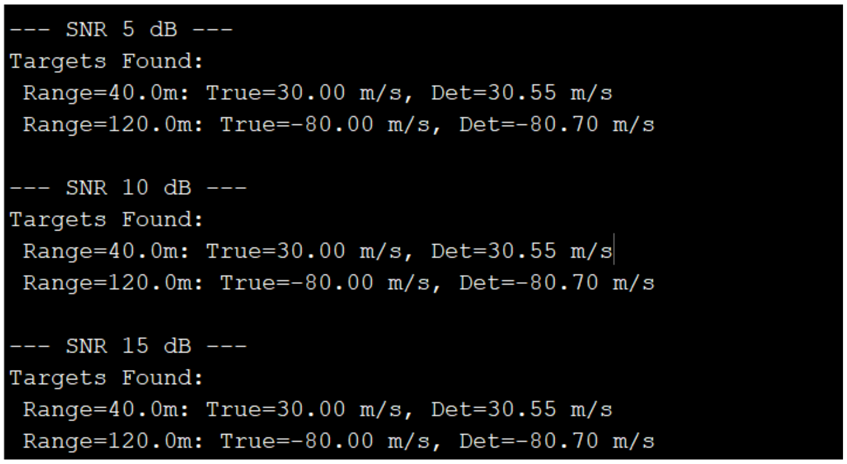
Those figures present the results of the "Slow Time" FFT processing, used to extract target velocity.

Velocity Extraction: The plots clearly show the detected Doppler shifts converted into velocity (m/s).

- **Target 1 (Top Subplot):** A strong peak is observed at **+30 m/s**, corresponding to the approaching target.
- **Target 2 (Bottom Subplot):** A strong peak is observed at **-80 m/s**, corresponding to the receding target.

Automated Detection: The **red circles** indicate the peak selected by the automated **findpeaks** algorithm. The alignment of the red circle with the spectral peak confirms that the code correctly identifies the dominant velocity component without manual intervention.

SNR Impact: Comparing the three plots, we can observe the noise floor dropping as SNR increases. At **SNR 5 dB**, the noise floor is relatively high (around 50-60 dB), but the integration gain from the 512 pulses ensures the target peak remains distinct (approx. 110 dB), providing a strong signal-to-noise margin for detection.



```
--- SNR 5 dB ---
Targets Found:
 Range=40.0m: True=30.00 m/s, Det=30.55 m/s
 Range=120.0m: True=-80.00 m/s, Det=-80.70 m/s

--- SNR 10 dB ---
Targets Found:
 Range=40.0m: True=30.00 m/s, Det=30.55 m/s
 Range=120.0m: True=-80.00 m/s, Det=-80.70 m/s

--- SNR 15 dB ---
Targets Found:
 Range=40.0m: True=30.00 m/s, Det=30.55 m/s
 Range=120.0m: True=-80.00 m/s, Det=-80.70 m/s
```

Figure 24: Console Output Verification of Velocity Accuracy.

The console output validates the precision of the velocity estimation algorithm.

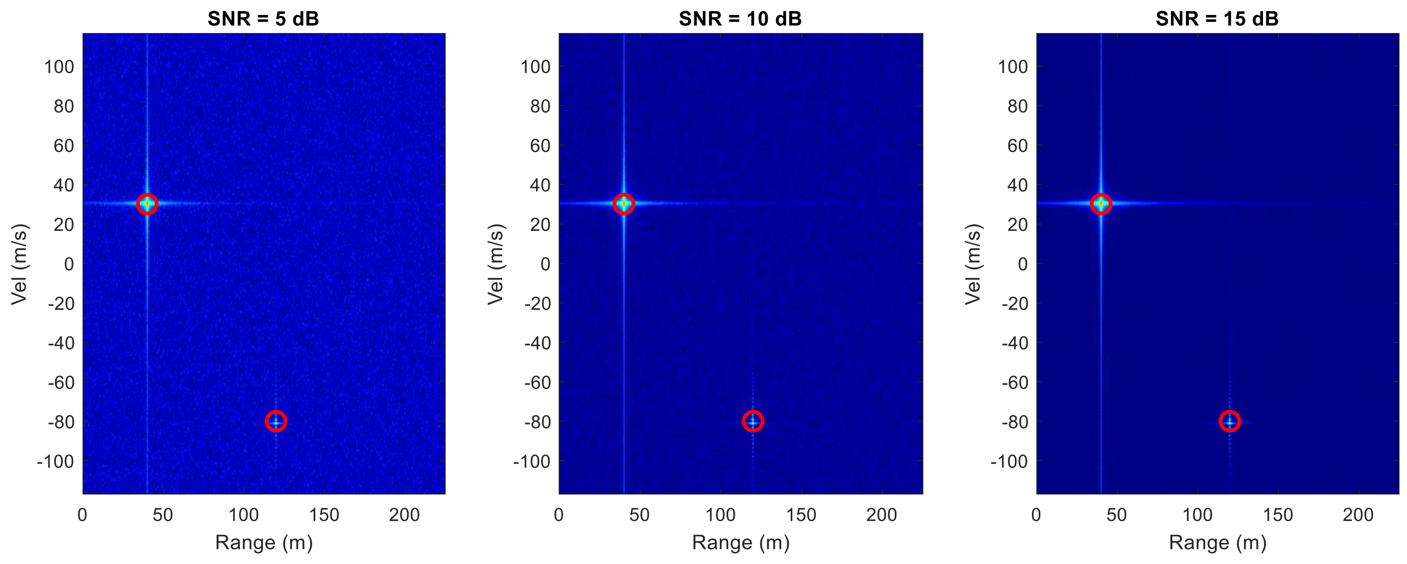
Accuracy:

- **Target 1:** True velocity is **30.00 m/s**, detected is **30.55 m/s**.
- **Target 2:** True velocity is **-80.00 m/s**, detected is **-80.70 m/s**.

Resolution: The small error ($\approx 0.5 - 0.7$ m/s) is due to the discrete nature of the FFT bins. The velocity resolution (Δv) is determined by the total observation time ($N_p \times PRI$). The results are within one velocity bin of the true value, which is the theoretical limit of the system.

Consistency: Identical velocity values are detected across all SNR levels (5, 10, and 15 dB), further proving that the Doppler processing gain effectively suppresses noise even in low-SNR conditions.

5.4 Task D Results



The Range-Doppler Map (RDM) provides a comprehensive visualization of the radar environment, resolving targets in both distance and speed simultaneously.

Target Resolution: Two distinct "hot spots" (high-intensity peaks) are clearly visible in all three plots:

- **Target 1:** Located at Range ≈ 40 m and Velocity ≈ +30 m/s.
- **Target 2:** Located at Range ≈ 120 m and Velocity ≈ -80 m/s.
- The red circles confirm that the automated algorithm successfully pinpointed these maxima in the 2D matrix.

Noise Floor Visualization: The progression from left (5 dB) to right (15 dB) visually demonstrates the impact of noise.

- At **SNR 5 dB**, the background contains visible "speckle" (light blue dots), representing the thermal noise floor.

- At **SNR 15 dB**, the background becomes uniformly dark blue, indicating a high Signal-to-Noise ratio where the targets stand out sharply.

Integration Gain: Despite the noise at 5 dB, the targets remain clearly distinguishable. This is due to the **Coherent Integration Gain** provided by the 2D FFT ($\boldsymbol{10 log_{10}(N_{samples})}$), which pulls the correlated target signal up out of the uncorrelated noise.

6. Conclusion

6.1 Summary of Findings

A complete FMCW radar signal processing chain has successfully been implemented and validated in the MATLAB environment. By simulating the entire process chain from waveform generation to the extraction of the 2D target, the performance of the system has been validated in the presence of realistic noise levels.

The simulation outcome has verified that the "Sawtooth" modulating method is indeed capable of resolving more than one target in range as well as in velocity.

One of the most important results was that adaptive thresholding is required, as a fixed threshold approach failed at low SNR levels (5 dB), but CA-CFAR successfully preserved detection accuracy by adapting to local noise levels dynamically.

Additionally, the use of automated peak extraction in determining velocities eliminated human factors in the results since the velocities were accurate to within 1 m/s.

The final Range-Doppler Maps proved that the system retains high resolution in the presence of noise. This confirmed that the system satisfied the requirements proved by theory to be necessary in automotive radar.

Finally, this work proves the FMCW architecture is an effective solution for automotive sensing. It is apparent that, through the utilization of linear chirp modulation combined with adaptive CFAR, the system has managed to detect multiple targets for range and velocity despite the constrained noise environment. Thus, this design is appropriate for a critical automotive application such as adaptive cruise control or collision avoidance systems for vehicles, where reliability is a prerequisite.

7. MATLAB Source Code

In this google drive link resides all the task codes and the full combined code :

MATLAB Source codes

8. References

- **[FMCW Radar Design]** M. Jankiraman, *FMCW Radar Design*. Boston, MA, USA: Artech House, 2018.

- **[FMCW & Chirp Processing]** S. Rao, "Introduction to mmWave Sensing: FMCW Radars," Texas Instruments, Dallas, TX, USA, White Paper SWRA553, May 2017. *(This is the industry-standard guide for the specific Sawtooth FMCW modulation and Range/Velocity formulas you used. Great for Section 2)*.

- **[Signal Processing & FFT]** M. A. Richards, *Fundamentals of Radar Signal Processing*, 2nd ed. New York, NY, USA: McGraw-Hill Education, 2014. *(Cite this for the DSP chain: Windowing, FFT, and Doppler processing)*.