Contents


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Radar Quiz 4 Simulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;
clc;
close all;
```

--- Parameters Initialization ---

```
% Student Specific Parameter
roll_no_str = '2022006';
KEY = str2double(roll_no_str(end-1:end));
if KEY == 0
    KEY = 100; % As per instruction for 00
end
fprintf('Using KEY = %d\n', KEY);

% Radar Parameters
c = 3e8;                      % Speed of light (m/s)
fc = 77e9;                    % Carrier frequency (Hz) - 77 GHz Automotive Radar
lambda = c / fc;              % Wavelength (m)

PRI = KEY * 1e-6;             % Pulse Repetition Interval (s) - KEY microseconds
PRF = 1 / PRI;                % Pulse Repetition Frequency (Hz)
dutyCycle = 0.10;             % 10% duty cycle
tau = PRI * dutyCycle;        % Pulse width (s)

% Polyphase Code Parameters
code = [+1, +1, +1, +1, +1, +1j, -1, -1j, +1, -1, +1, -1, +1, -1j, -1, +1j];
N_code = length(code);        % Number of bits in the code (16)
T_bit = tau / N_code;         % Duration of each code bit (s)
Bw = 1 / T_bit;               % Approximate bandwidth (Hz)

fprintf('--- Radar Specifications ---\n');
fprintf('Carrier Frequency (fc): %.2f GHz\n', fc/1e9);
fprintf('Wavelength (lambda): %.4f mm\n', lambda*1e3);
fprintf('Pulse Repetition Interval (PRI): %.2f us\n', PRI*1e6);
fprintf('Pulse Repetition Frequency (PRF): %.2f kHz\n', PRF/1e3);
fprintf('Pulse Width (tau): %.2f us\n', tau*1e6);
fprintf('Code Length (N_code): %d bits\n', N_code);
fprintf('Bit Duration (T_bit): %.2f ns\n', T_bit*1e9);
fprintf('Approximate Bandwidth (Bw): %.2f MHz\n', Bw/1e6);

% Target Parameters
R1 = 100;      % Range of target 1 (m)
v1 = -5;       % Velocity of target 1 (m/s) - Negative for towards radar
R2 = 50;       % Range of target 2 (m)
v2 = 2.5;      % Velocity of target 2 (m/s) - Positive for away from radar

fprintf('--- Target Parameters ---\n');
fprintf('Target 1: Range = %.1f m, Velocity = %.1f m/s\n', R1, v1);
fprintf('Target 2: Range = %.1f m, Velocity = %.1f m/s\n', R2, v2);

% Simulation Parameters
N_pulses = 128;               % Number of pulses for Doppler processing (Coherent Processing Interval - CPI)
Fs = 2 * Bw;                  % Sampling frequency (Hz) - At least Nyquist rate
                              % Let's ensure integer samples per bit for better simulation
N_samples_per_bit_ideal = Fs * T_bit;
N_samples_per_bit = ceil(N_samples_per_bit_ideal * 4) / 4; % Choose a slightly higher Fs for integer/half samples, e.g., factor of 4
if N_samples_per_bit < 2
    N_samples_per_bit = 4; % Ensure at least a few samples per bit
end
Fs = N_samples_per_bit / T_bit; % Recalculate Fs based on chosen samples per bit
```

```
dt = 1 / Fs;                % Time resolution (s)

N_samples_pulse = N_code * N_samples_per_bit; % Samples per pulse width
N_samples_PRI = round(PRI * Fs);           % Samples per PRI
t_pulse = (0:N_samples_pulse-1) * dt;      % Time vector for one pulse duration
t_pri = (0:N_samples_PRI-1) * dt;          % Time vector for one PRI duration

fprintf('--- Simulation Setup ---\n');
fprintf('Sampling Frequency (Fs): %.2f MHz\n', Fs/1e6);
fprintf('Samples per Bit: %d\n', N_samples_per_bit);
fprintf('Samples per Pulse: %d\n', N_samples_pulse);
fprintf('Samples per PRI: %d\n', N_samples_PRI);
fprintf('Number of Pulses in CPI (N_pulses): %d\n', N_pulses);
```

```
Using KEY = 6
--- Radar Specifications ---
Carrier Frequency (fc): 77.00 GHz
Wavelength (lambda): 3.8961 mm
Pulse Repetition Interval (PRI): 6.00 us
Pulse Repetition Frequency (PRF): 166.67 kHz
Pulse Width (tau): 0.60 us
Code Length (N_code): 16 bits
Bit Duration (T_bit): 37.50 ns
Approximate Bandwidth (Bw): 26.67 MHz
--- Target Parameters ---
Target 1: Range = 100.0 m, Velocity = -5.0 m/s
Target 2: Range = 50.0 m, Velocity = 2.5 m/s
--- Simulation Setup ---
Sampling Frequency (Fs): 53.33 MHz
Samples per Bit: 2
Samples per Pulse: 32
Samples per PRI: 320
Number of Pulses in CPI (N_pulses): 128
```

--- Generate Transmitted Baseband Signal (One Pulse) ---

```
s_baseband_pulse = zeros(1, N_samples_pulse);
for k = 1:N_code
    start_idx = (k-1) * N_samples_per_bit + 1;
    end_idx = k * N_samples_per_bit;
    s_baseband_pulse(start_idx:end_idx) = code(k);
end

% Create the full signal for one PRI (pulse + zeros)
s_pri = zeros(1, N_samples_PRI);
s_pri(1:N_samples_pulse) = s_baseband_pulse;
```

--- Simulate Received Signal (Down-converted Baseband) ---

```
% Calculate expected delays and Doppler shifts
delay1 = 2 * R1 / c;
fd1 = 2 * v1 / lambda; % Doppler shift for target 1
delay_samples1 = round(delay1 * Fs);

delay2 = 2 * R2 / c;
fd2 = 2 * v2 / lambda; % Doppler shift for target 2
delay_samples2 = round(delay2 * Fs);

fprintf('--- Expected Target Properties ---\n');
fprintf('Target 1: Delay = %.2f us (Samples = %d), Doppler = %.2f Hz\n', delay1*1e6, delay_samples1, fd1);
fprintf('Target 2: Delay = %.2f us (Samples = %d), Doppler = %.2f Hz\n', delay2*1e6, delay_samples2, fd2);


% Initialize received signal matrix (Range x Pulses)
rx_signal = zeros(N_samples_PRI, N_pulses);

% Simulate received signal for each pulse
SNR_dB = 20; % Assume Signal-to-Noise Ratio in dB for simulation clarity
signal_power = mean(abs(s_baseband_pulse).^2);
noise_power = signal_power / (10^(SNR_dB/10));
noise_std_dev = sqrt(noise_power / 2); % Standard deviation for real/imag parts

for m = 1:N_pulses % Loop through pulses
    % Current time for Doppler phase calculation
```

```
    t_slow = (m-1) * PRI;

    % Generate noise for this pulse
    noise = noise_std_dev * (randn(N_samples_PRI, 1) + 1j * randn(N_samples_PRI, 1));
    rx_pulse_m = noise;

    % --- Target 1 Signal ---
    % Calculate Doppler phase shift for this pulse
    doppler_phase1 = exp(1j * 2 * pi * fd1 * t_slow);
    % Generate received pulse (delayed and phase shifted)
    signal1 = zeros(N_samples_PRI, 1);
    if (delay_samples1 + N_samples_pulse) <= N_samples_PRI
       signal1(delay_samples1 + (1:N_samples_pulse)) = s_baseband_pulse * doppler_phase1;
    end
    rx_pulse_m = rx_pulse_m + signal1;

    % --- Target 2 Signal ---
    % Calculate Doppler phase shift for this pulse
    doppler_phase2 = exp(1j * 2 * pi * fd2 * t_slow);
     % Generate received pulse (delayed and phase shifted)
    signal2 = zeros(N_samples_PRI, 1);
    if (delay_samples2 + N_samples_pulse) <= N_samples_PRI
       signal2(delay_samples2 + (1:N_samples_pulse)) = s_baseband_pulse * doppler_phase2;
    end
    rx_pulse_m = rx_pulse_m + signal2;

    % Store the received signal for this pulse
    rx_signal(:, m) = rx_pulse_m;
end

disp('Received signal simulation complete.');
```

```
--- Expected Target Properties ---
Target 1: Delay = 0.67 us (Samples = 36), Doppler = -2566.67 Hz
Target 2: Delay = 0.33 us (Samples = 18), Doppler = 1283.33 Hz
Received signal simulation complete.
```

--- Signal Processing ---

```
% 1. Pulse Compression (Matched Filtering)
matched_filter = conj(flipud(s_baseband_pulse(:))); % Matched filter (complex conjugate, time reversed)
pc_signal = zeros(N_samples_PRI, N_pulses);

fprintf('Applying Matched Filter (Pulse Compression)...\n');
for m = 1:N_pulses
    % Apply matched filter using convolution
    % 'same' ensures output length matches input (N_samples_PRI)
    % Convolution with the time-reversed conjugate is equivalent to correlation
    pc_signal(:, m) = conv(rx_signal(:, m), matched_filter, 'same');
end
disp('Pulse Compression complete.');

% 2. Doppler Processing (FFT across pulses)
fprintf('Applying Doppler FFT...\n');
% Apply FFT across the 'slow time' (pulse) dimension for each range bin
range_doppler_map = fftshift(fft(pc_signal, N_pulses, 2), 2);
disp('Range-Doppler Map generated.');
```

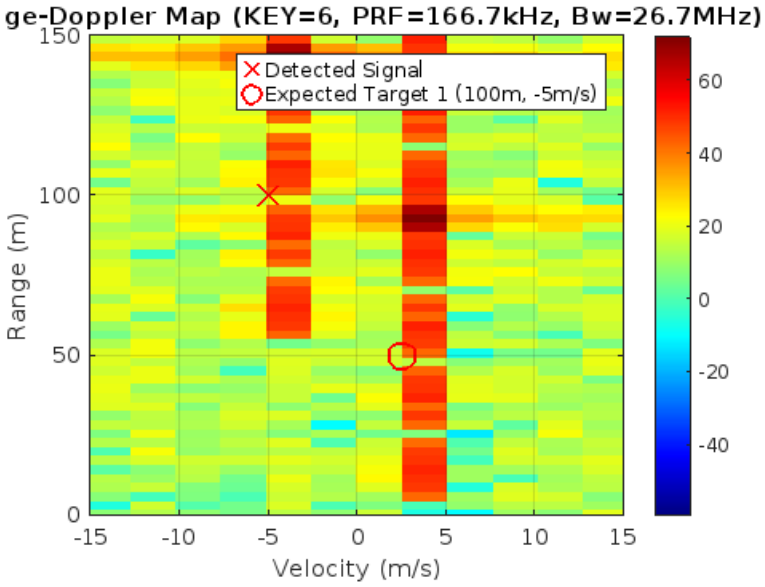
```
Applying Matched Filter (Pulse Compression)...
Pulse Compression complete.
Applying Doppler FFT...
Range-Doppler Map generated.
```

--- Plotting the Range-Doppler Map ---

```
% Create axes
range_axis = (0:N_samples_PRI-1) * dt * c / 2; % Convert sample index to range (m)
doppler_freq_axis = linspace(-PRF/2, PRF/2, N_pulses); % Doppler frequency axis (Hz)
velocity_axis = doppler_freq_axis * lambda / 2;       % Convert Doppler frequency to velocity (m/s)

figure;
imagesc(velocity_axis, range_axis, 20*log10(abs(range_doppler_map)));
```

```
colormap('jet'); % Use a standard colormap
colorbar; % Show the color scale (in dB)
xlabel('Velocity (m/s)');
ylabel('Range (m)');
title(sprintf('Range-Doppler Map (KEY=%d, PRF=%.1fkHz, Bw=%.1fMHz)', KEY, PRF/1e3, Bw/1e6));
axis xy; % Ensure range starts from 0 at the bottom
grid on;

% Optional: Add markers for expected target locations
hold on;
plot(v1, R1, 'rx', 'MarkerSize', 12, 'LineWidth', 2); % Expected Target 1
plot(v2, R2, 'ro', 'MarkerSize', 12, 'LineWidth', 2); % Expected Target 2
legend('Detected Signal', 'Expected Target 1 (100m, -5m/s)', 'Expected Target 2 (50m, +2.5m/s)');
hold off;

% Adjust axis limits for better visualization if needed
ylim([0 max(R1, R2) + 50]); % Focus range axis around targets
v_lim = max(abs(v1), abs(v2));
xlim([-v_lim-10 v_lim+10]); % Focus velocity axis around targets
```

```
Warning: Ignoring extra legend entries.
```



--- Coded Calculations for Written Component ---

```
% Range Resolution
delta_R = c / (2 * Bw);
fprintf('\n--- Resolution & Ambiguity Calculations ---\n');
fprintf('Range Resolution (delta_R = c / (2 * Bw)): %.3f m\n', delta_R);

% Doppler Velocity Resolution
T_cpi = N_pulses * PRI; % Coherent Processing Interval duration
delta_fd = 1 / T_cpi;   % Doppler Frequency Resolution
delta_V = delta_fd * lambda / 2; % Doppler Velocity Resolution
fprintf('Doppler Velocity Resolution (delta_V = lambda / (2 * N_pulses * PRI)): %.3f m/s\n', delta_V);

% Maximum Unambiguous Range
R_unamb = c * PRI / 2;
fprintf('Maximum Unambiguous Range (R_unamb = c * PRI / 2): %.1f m\n', R_unamb);

% Maximum Unambiguous Velocity
V_unamb = lambda * PRF / 4; % Magnitude of max unambiguous velocity (+/- V_unamb)
fprintf('Maximum Unambiguous Velocity (+/- V_unamb = +/- lambda * PRF / 4): +/- %.2f m/s\n', V_unamb);
```

```
--- Resolution & Ambiguity Calculations ---
Range Resolution (delta_R = c / (2 * Bw)): 5.625 m
Doppler Velocity Resolution (delta_V = lambda / (2 * N_pulses * PRI)): 2.537 m/s
Maximum Unambiguous Range (R_unamb = c * PRI / 2): 900.0 m
Maximum Unambiguous Velocity (+/- V_unamb = +/- lambda * PRF / 4): +/- 162.34 m/s
```

Published with MATLAB® R2024b