Table of Contents

Radar Parameters Definition

```
clear; clc; close all;

% --- Basic Parameters ---
KEY = 6; % From Roll No. 2022006
T_pri = KEY * 1e-6; % Pulse Repetition Interval (s)
duty_cycle = 0.1; % 10% Duty Cycle
fc = 77e9; % Carrier Frequency (Hz)
c = 3e8; % Speed of Light (m/s)
lambda = c / fc; % Wavelength (m)

% --- Derived Pulse Parameters ---
tau = duty_cycle * T_pri; % Pulse Width (s)
N_code = 16; % Number of bits in the polyphase code
T_bit = tau / N_code; % Bit duration (s)
B = 1 / T_bit; % Approximate Bandwidth (Hz) -> More accurately B = N_code /
tau = 1/T_bit

% --- Polyphase Code Sequence ---
code = [+1, +1, +1, +1, +1, +1j, -1, -1j, +1, -1, +1, -1, +1, -1j, -1, +1j];

% --- Target Parameters ---
R1 = 100; % Range of Target 1 (m)
v1 = -5;  % Velocity of Target 1 (m/s, negative for towards)
R2 = 50;  % Range of Target 2 (m)
v2 = 2.5; % Velocity of Target 2 (m/s, positive for away)

% --- Simulation Parameters ---
fs_oversample_factor = 4; % Oversampling factor relative to bandwidth (chip
rate)
fs = fs_oversample_factor * B; % Sampling Frequency (Hz)
Ts = 1/fs; % Sampling Period (s)
N_pulses = 128; % Number of pulses for Doppler processing (Coherent
Processing Interval)

% --- Calculate Radar Performance Metrics ---
prf = 1 / T_pri; % Pulse Repetition Frequency (Hz)
delta_R = c * T_bit / 2; % Range Resolution (m) - using T_bit for pulse
compressed resolution
% delta_R_uncompressed = c * tau / 2; % Uncompressed range resolution
R_max_unamb = c * T_pri / 2; % Maximum Unambiguous Range (m)
```

```
v_max_unamb_total = lambda * prf / 2; % Total unambiguous velocity range (+/-)
v_max_unamb_speed = lambda * prf / 4; % Maximum unambiguous speed (m/s)
delta_v = lambda / (2 * N_pulses * T_pri); % Doppler Velocity Resolution (m/s)

fprintf('--- Radar Parameters ---\n');
fprintf('KEY: %d\n', KEY);
fprintf('PRI (T_pri): %.2f us\n', T_pri * 1e6);
fprintf('PRF: %.2f kHz\n', prf / 1e3);
fprintf('Pulse Width (tau): %.2f us\n', tau * 1e6);
fprintf('Bit Duration (T_bit): %.2f ns\n', T_bit * 1e9);
fprintf('Bandwidth (B): %.2f MHz\n', B / 1e6);
fprintf('Carrier Frequency (fc): %.1f GHz\n', fc / 1e9);
fprintf('Wavelength (lambda): %.2f mm\n', lambda * 1e3);
fprintf('Sampling Frequency (fs): %.2f MHz\n', fs / 1e6);
fprintf('Number of Pulses (N_pulses): %d\n', N_pulses);
fprintf('\n--- Performance Metrics ---\n');
fprintf('Range Resolution (delta_R): %.3f m\n', delta_R);
fprintf('Max Unambiguous Range (R_max_unamb): %.2f m\n', R_max_unamb);
fprintf('Max Unambiguous Speed (v_max_unamb_speed): +/- %.2f m/s\n',
v_max_unamb_speed);
fprintf('Velocity Resolution (delta_v): %.3f m/s\n', delta_v);
fprintf('\n');
```

```
--- Radar Parameters ---
KEY: 6
PRI (T_pri): 6.00 us
PRF: 166.67 kHz
Pulse Width (tau): 0.60 us
Bit Duration (T_bit): 37.50 ns
Bandwidth (B): 26.67 MHz
Carrier Frequency (fc): 77.0 GHz
Wavelength (lambda): 3.90 mm
Sampling Frequency (fs): 106.67 MHz
Number of Pulses (N_pulses): 128

--- Performance Metrics ---
Range Resolution (delta_R): 5.625 m
Max Unambiguous Range (R_max_unamb): 900.00 m
Max Unambiguous Speed (v_max_unamb_speed): +/- 162.34 m/s
Velocity Resolution (delta_v): 2.537 m/s
```

Generate Transmitted Baseband Pulse

```
samples_per_bit = round(T_bit / Ts);
pulse_length_samples = N_code * samples_per_bit;

% Create the upsampled code sequence
s_baseband = repelem(code, samples_per_bit);
s_baseband = s_baseband(:); % Ensure it's a column vector

% Time vector for one pulse
t_pulse = (0:length(s_baseband)-1) * Ts;
```

Generate Received Signal (Down-converted Baseband)

```
% Time vector for one PRI
max_time_pri = T_pri;
N_samples_pri = round(max_time_pri / Ts);
t_pri = (0:N_samples_pri-1) * Ts;

% Initialize received signal matrix (Range x Pulses)
rx_signal_matrix = zeros(N_samples_pri, N_pulses);

% Calculate delays and Doppler shifts
td1 = 2 * R1 / c;
td2 = 2 * R2 / c;
fd1 = 2 * v1 / lambda;
fd2 = 2 * v2 / lambda;

% Target amplitudes (relative)
A1 = 1.0;
A2 = 0.8; % Assume slightly lower amplitude for closer target (can vary)

% Noise parameters
NoiseVariance = 0.05; % Adjust noise level as needed

% Simulate pulse by pulse
for i = 1:N_pulses
    % Current time within the coherent processing interval
    current_T = (i-1) * T_pri;

    % --- Target 1 ---
    % Phase shift due to Doppler for this pulse
    phi1 = 2 * pi * fd1 * current_T;
    % Time index for the start of the received pulse
    idx1 = round(td1 / Ts);
    % Generate the received pulse contribution (including intra-pulse Doppler)
    rx_pulse1 = A1 * s_baseband .* exp(1j * 2 * pi * fd1 * t_pulse.') *
exp(1j * phi1);

    % --- Target 2 ---
    % Phase shift due to Doppler for this pulse
    phi2 = 2 * pi * fd2 * current_T;
    % Time index for the start of the received pulse
    idx2 = round(td2 / Ts);
    % Generate the received pulse contribution (including intra-pulse Doppler)
    rx_pulse2 = A2 * s_baseband .* exp(1j * 2 * pi * fd2 * t_pulse.') *
exp(1j * phi2);

    % --- Combine signals for this PRI ---
    rx_pulse_i = zeros(N_samples_pri, 1);
    % Add target 1 signal (check bounds)
    if idx1 + pulse_length_samples <= N_samples_pri
        rx_pulse_i(idx1 + 1 : idx1 + pulse_length_samples) = ...
```

```
            rx_pulse_i(idx1 + 1 : idx1 + pulse_length_samples) + rx_pulse1;
    end
     % Add target 2 signal (check bounds)
    if idx2 + pulse_length_samples <= N_samples_pri
        rx_pulse_i(idx2 + 1 : idx2 + pulse_length_samples) = ...
            rx_pulse_i(idx2 + 1 : idx2 + pulse_length_samples) + rx_pulse2;
    end

    % --- Add Complex White Gaussian Noise ---
    noise = sqrt(NoiseVariance/2) * (randn(N_samples_pri, 1) + 1j *
randn(N_samples_pri, 1));
    rx_signal_matrix(:, i) = rx_pulse_i + noise;
end
```

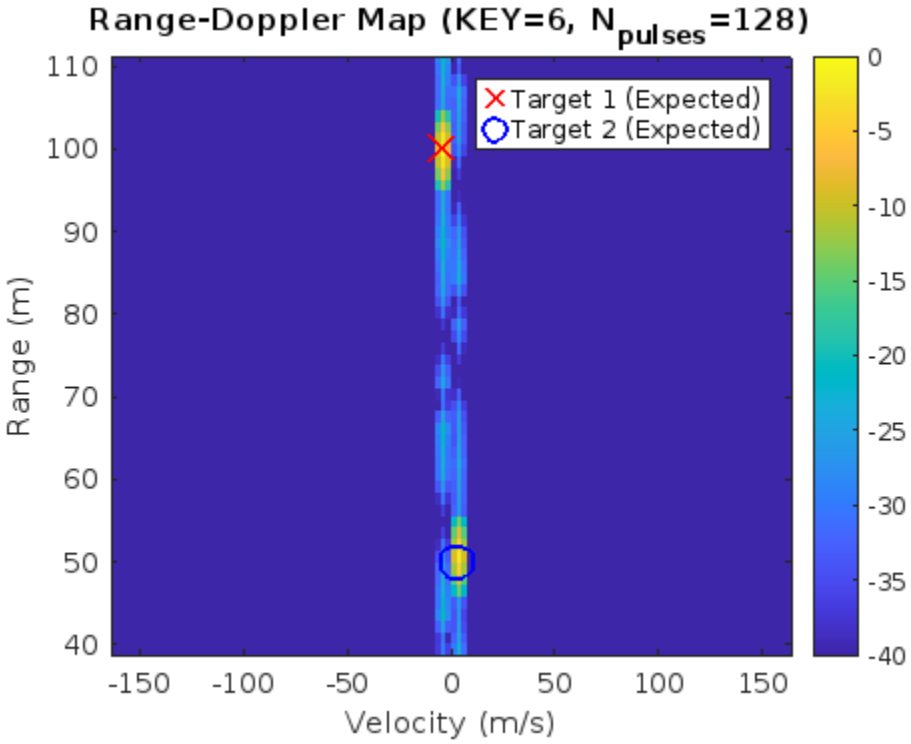
Process the Received Signal

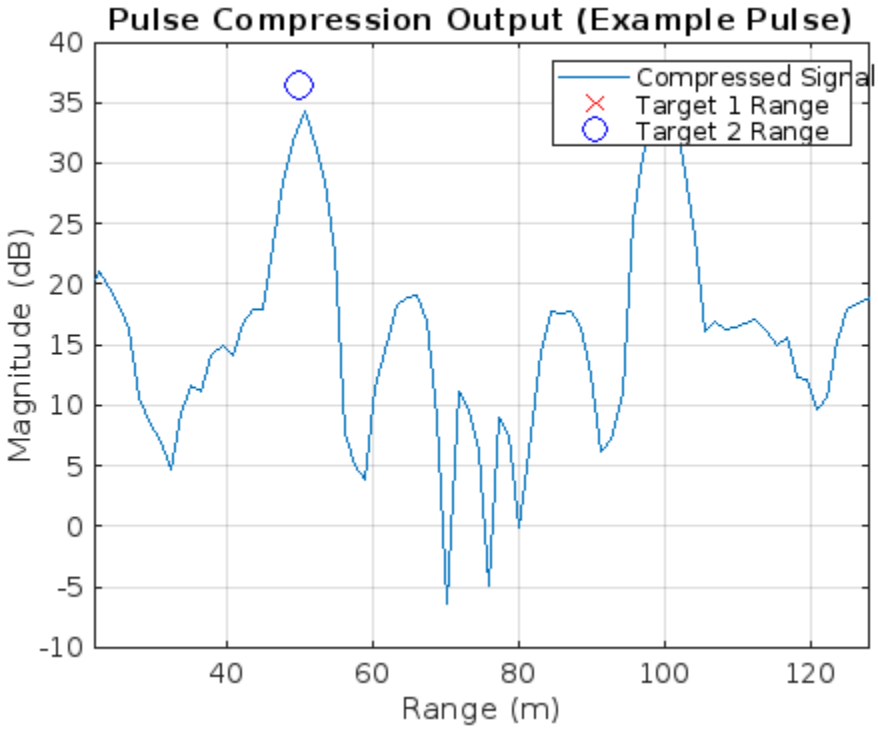
```
% --- Matched Filtering (Pulse Compression) ---
matched_filter_coeff = conj(flipud(s_baseband)); % Time-reversed conjugate
compressed_matrix = zeros(size(rx_signal_matrix));

for i = 1:N_pulses
    % Apply matched filter using convolution or filter
    % Using filter is efficient; output length is same as input
    compressed_matrix(:, i) = filter(matched_filter_coeff, 1,
rx_signal_matrix(:, i));
    % Note: The peak output of filter occurs after a delay equal to filter
length - 1
end

% --- Doppler Processing (FFT across pulses) ---
% Apply a window function (e.g., Hamming) across pulses to reduce Doppler
sidelobes
window = hamming(N_pulses);
windowed_compressed = compressed_matrix .* window.'; % Apply window

% Perform FFT across pulses (dimension 2) and shift zero-frequency component
range_doppler_map = fftshift(fft(windowed_compressed, N_pulses, 2), 2);
```

Visualization

```
% --- Create Axes ---
% Range Axis: Correct for matched filter delay
range_axis = ( (0:N_samples_pri-1) - (pulse_length_samples - 1) ) * Ts * c /
2;

% Velocity Axis: From -v_max_unamb_speed to +v_max_unamb_speed
velocity_axis = linspace(-v_max_unamb_speed, v_max_unamb_speed, N_pulses);

% --- Plotting ---
figure;
% Convert to dB scale for better visualization
range_doppler_db = 10 * log10(abs(range_doppler_map).^2);
% Normalize for better display
```

```
range_doppler_db = range_doppler_db - max(range_doppler_db(:));

imagesc(velocity_axis, range_axis, range_doppler_db);
colorbar;
xlabel('Velocity (m/s)');
ylabel('Range (m)');
title(sprintf('Range-Doppler Map (KEY=%d, N_{pulses}=%d)', KEY, N_pulses));
axis xy; % Set origin to lower-left corner

% Set reasonable plot limits (optional, adjust as needed)
ylim([min(R1, R2)-2*delta_R, max(R1, R2)+2*delta_R]); % Focus range around
targets
% ylim([0 R_max_unamb]); % Show full unambiguous range
clim([-40, 0]); % Adjust color limits (dB) for visibility

% Add markers for expected target locations
hold on;
plot(v1, R1, 'rx', 'MarkerSize', 12, 'LineWidth', 2); % Target 1
plot(v2, R2, 'bo', 'MarkerSize', 12, 'LineWidth', 2); % Target 2
legend('Target 1 (Expected)', 'Target 2 (Expected)');
hold off;

% --- Optional: Plot Pulse Compression Output for one pulse ---
figure;
plot(range_axis, 10*log10(abs(compressed_matrix(:, N_pulses/2)).^2));
title('Pulse Compression Output (Example Pulse)');
xlabel('Range (m)');
ylabel('Magnitude (dB)');
grid on;
xlim([min(R1, R2)-5*delta_R, max(R1, R2)+5*delta_R]);
% % hold on;
plot(R1, max(10*log10(abs(compressed_matrix(:, N_pulses/2)).^2)), 'rx',
'MarkerSize', 10);
plot(R2, max(10*log10(abs(compressed_matrix(:, N_pulses/2)).^2)), 'bo',
'MarkerSize', 10);
legend('Compressed Signal', 'Target 1 Range', 'Target 2 Range');
hold off;
```





End of Code

Published with MATLAB® R2024b