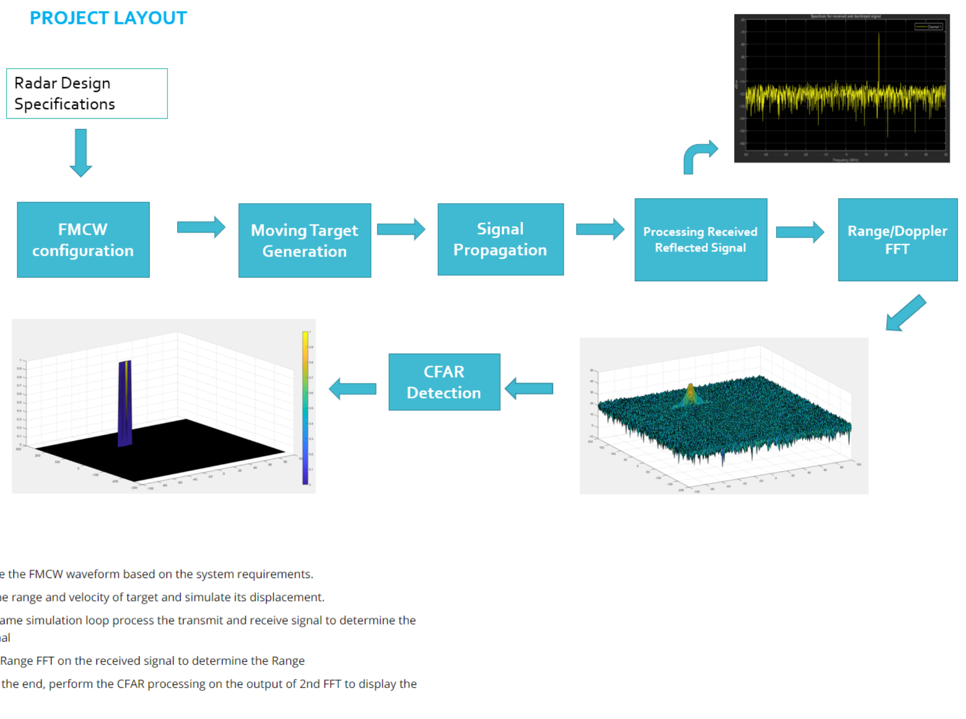
Udacity Sensor Fusion - Final Project - Radar Target Generation and Detection

Deniz Bardakci

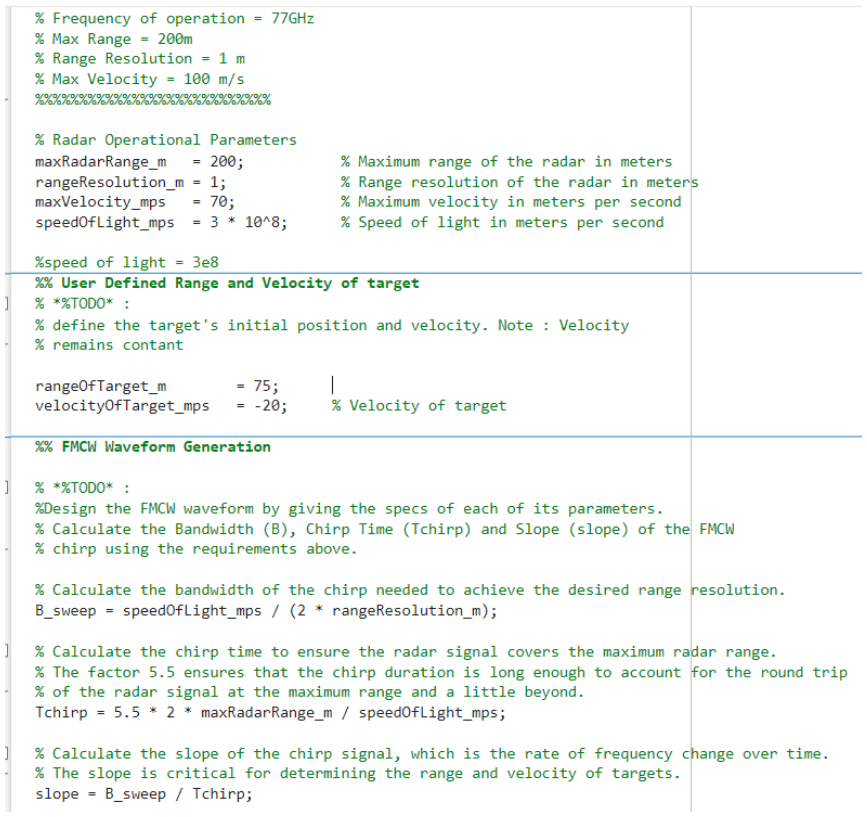
March 2024

1 Project Layout



- Configure the FMCW waveform based on the system requirements.
- Define the range and velocity of target and simulate its displacement.
- For the same simulation loop process the transmit and receive signal to determine the beat signal
- Perform Range FFT on the received signal to determine the Range
- Towards the end, perform the CFAR processing on the output of 2nd FFT to display the target.

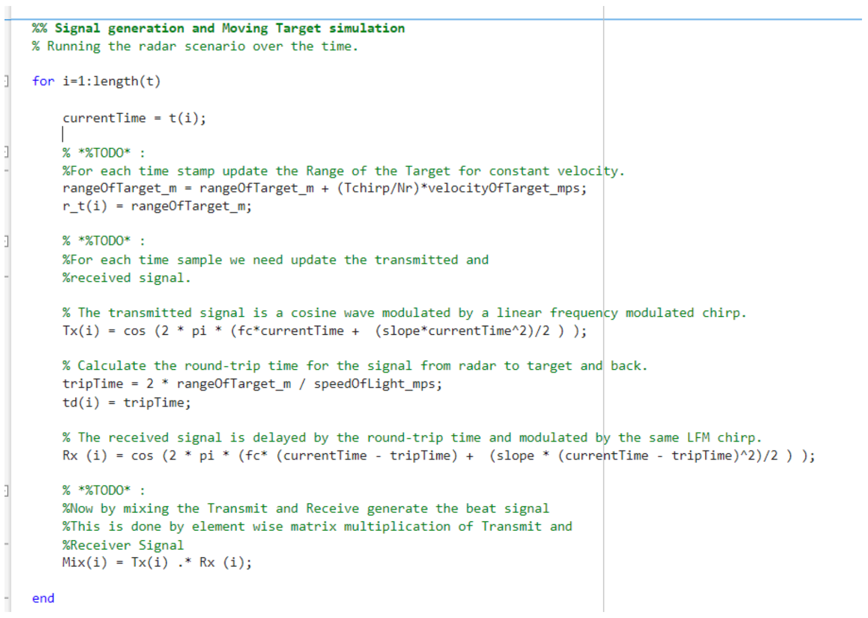
Figure 1: Project Layouy



```
% Frequency of operation = 77GHz
% Max Range = 200m
% Range Resolution = 1 m
% Max Velocity = 100 m/s
%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Radar Operational Parameters
maxRadarRange_m   = 200;          % Maximum range of the radar in meters
rangeResolution_m = 1;            % Range resolution of the radar in meters
maxVelocity_mps   = 70;           % Maximum velocity in meters per second
speedOfLight_mps  = 3 * 10^8;     % Speed of light in meters per second

%speed of light = 3e8
%% User Defined Range and Velocity of target
% *%TODO* :
% define the target's initial position and velocity. Note : Velocity
% remains contant

rangeOfTarget_m       = 75;
velocityOfTarget_mps  = -20;     % Velocity of target

%% FMCW Waveform Generation

% *%TODO* :
%Design the FMCW waveform by giving the specs of each of its parameters.
% Calculate the Bandwidth (B), Chirp Time (Tchirp) and Slope (slope) of the FMCW
% chirp using the requirements above.

% Calculate the bandwidth of the chirp needed to achieve the desired range resolution.
B_sweep = speedOfLight_mps / (2 * rangeResolution_m);

% Calculate the chirp time to ensure the radar signal covers the maximum radar range.
% The factor 5.5 ensures that the chirp duration is long enough to account for the round trip
% of the radar signal at the maximum range and a little beyond.
Tchirp = 5.5 * 2 * maxRadarRange_m / speedOfLight_mps;

% Calculate the slope of the chirp signal, which is the rate of frequency change over time.
% The slope is critical for determining the range and velocity of targets.
slope = B_sweep / Tchirp;
```

Figure 2: System Requirements, User Defined Inputs, FMCW Generation Steps

2 Signal Generation and Moving Target Simulation



```
%% Signal generation and Moving Target simulation
% Running the radar scenario over the time.

for i=1:length(t)

    currentTime = t(i);

    % *%TODO* :
    %For each time stamp update the Range of the Target for constant velocity.
    rangeOfTarget_m = rangeOfTarget_m + (Tchirp/Nr)*velocityOfTarget_mps;
    r_t(i) = rangeOfTarget_m;

    % *%TODO* :
    %For each time sample we need update the transmitted and
    %received signal.

    % The transmitted signal is a cosine wave modulated by a linear frequency modulated chirp.
    Tx(i) = cos (2 * pi * (fc*currentTime +  (slope*currentTime^2)/2 ) );

    % Calculate the round-trip time for the signal from radar to target and back.
    tripTime = 2 * rangeOfTarget_m / speedOfLight_mps;
    td(i) = tripTime;

    % The received signal is delayed by the round-trip time and modulated by the same LFM chirp.
    Rx (i) = cos (2 * pi * (fc* (currentTime - tripTime) +  (slope * (currentTime - tripTime)^2)/2 ) );

    % *%TODO* :
    %Now by mixing the Transmit and Receive generate the beat signal
    %This is done by element wise matrix multiplication of Transmit and
    %Receiver Signal
    Mix(i) = Tx(i) .* Rx (i);

end
```

Figure 3: Signal Generation and Moving Target Simulation

The code in the Figure 3, iterates through each time step, updating the target's range based on its constant velocity and calculating both the transmitted and received signals. The transmitted signal is modeled as a cosine wave modulated by a linear frequency-modulated (LFM) chirp, reflecting the change in frequency over time to encode the range information. For each time step, the code computes the round-trip time of the radar signal to the moving target and back, adjusting the received signal to account for this delay and the target's range. The received signal, delayed to simulate the round-trip time, is also modulated by the same LFM chirp. Finally, the code generates a beat signal by element-wise multiplication of the transmitted and received signals, capturing the difference in frequency between these signals, which is crucial for determining the target's range and velocity from the radar returns. This process effectively simulates the interaction between a radar system and a moving target, providing the foundation for subsequent range and velocity estimation.

3 Range/Doppler FFT

A correct implementation should generate a peak at the correct range, i.e the initial position of target assigned with an error margin of +/- 10 meters. The initial position was 75 as can be seen in Fig. 2.

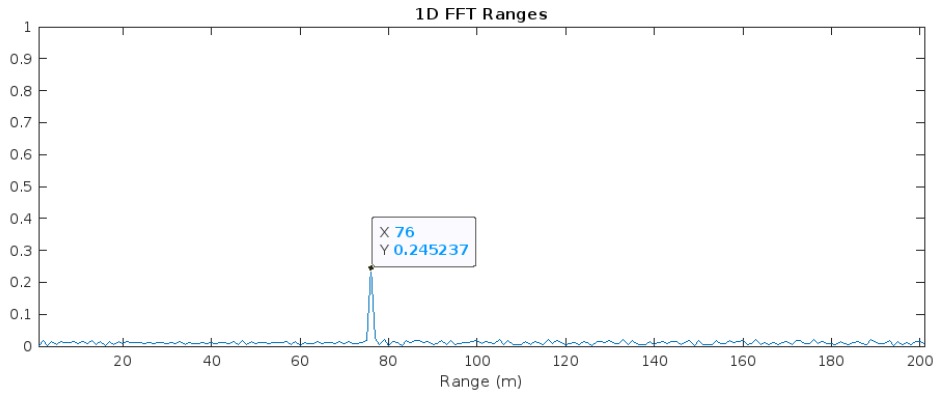


Figure 4: Range FFT

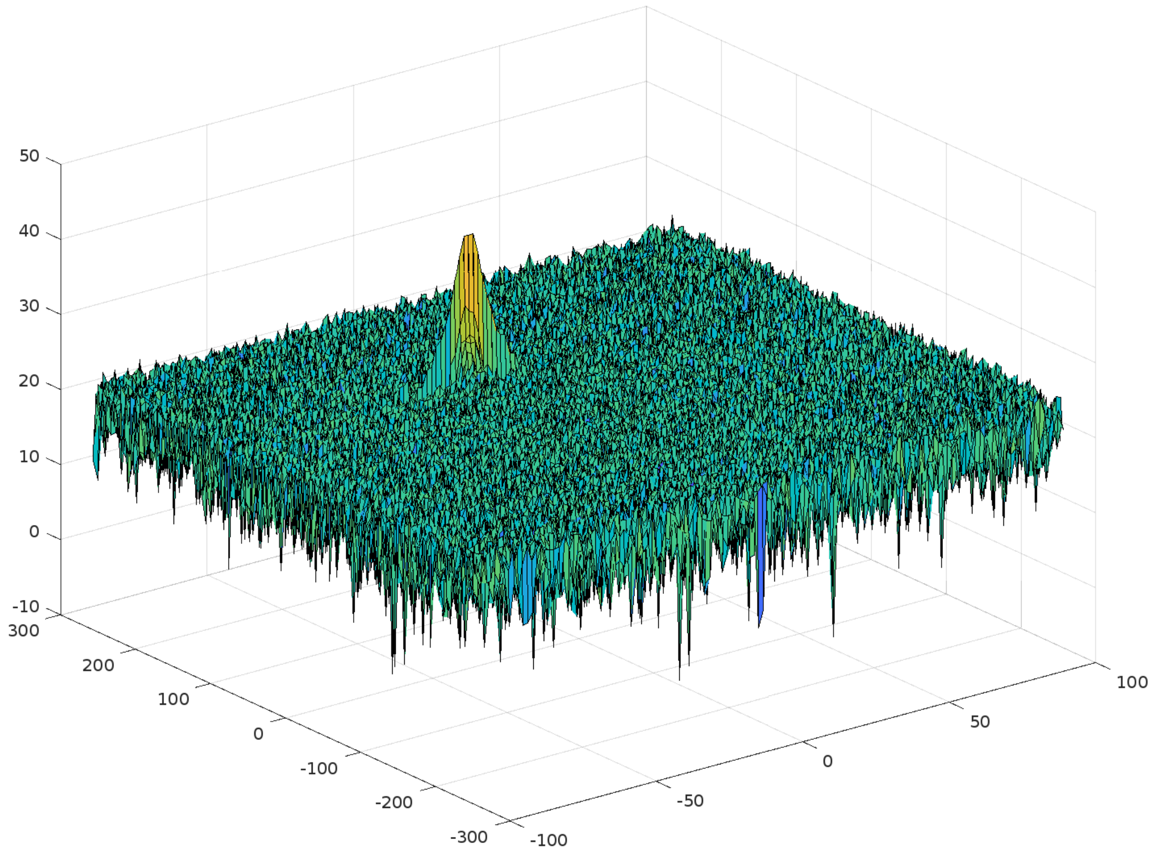
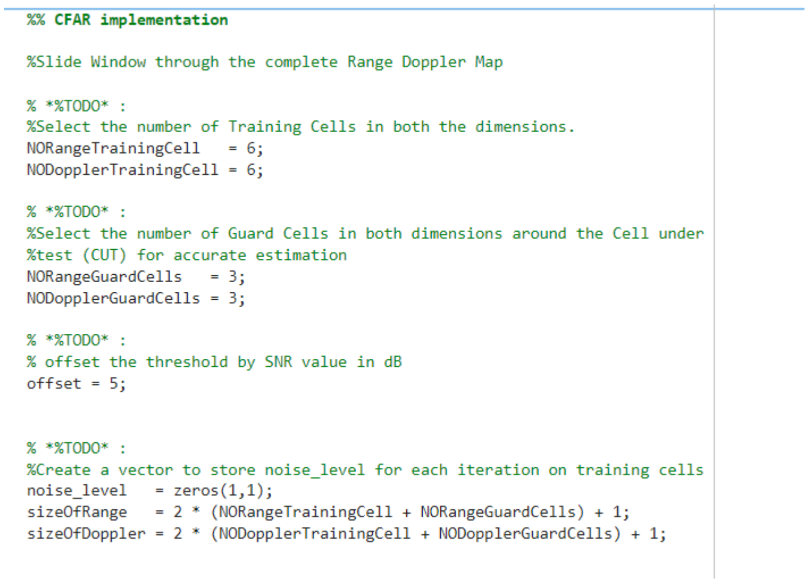


Figure 5: Range Doppler Map

4 CFAR Detection

The code for CFAR calculation efficiently implements the Constant False Alarm Rate (CFAR) detection algorithm for identifying targets within a Range Doppler Map (RDM), crucial for radar signal processing. By sliding a window across each Cell Under Test (CUT) and calculating the surrounding noise level from Training cells—excluding the Guard cells and CUT itself—the algorithm dynamically adjusts detection thresholds to maintain a constant false alarm rate. It optimizes performance by converting the RDM from decibels to power once, leveraging MATLAB's vectorized operations for summing and thresholding, and marking detections in an output matrix. This approach ensures accurate target detection amidst background noise, balancing sensitivity and specificity by adjusting the threshold based on local noise, thereby enhancing the radar system's reliability and efficiency.

- The CFAR Results, Fig.8 shows that the velocity of the target vehicle is within the +- 10 m/s criteria.



```
%% CFAR implementation

%Slide Window through the complete Range Doppler Map

% *%TODO* :
%Select the number of Training Cells in both the dimensions.
NORangeTrainingCell   = 6;
NODopplerTrainingCell = 6;

% *%TODO* :
%Select the number of Guard Cells in both dimensions around the Cell under
%test (CUT) for accurate estimation
NORangeGuardCells   = 3;
NODopplerGuardCells = 3;

% *%TODO* :
% offset the threshold by SNR value in dB
offset = 5;


% *%TODO* :
%Create a vector to store noise_level for each iteration on training cells
noise_level   = zeros(1,1);
sizeOfRange   = 2 * (NORangeTrainingCell + NORangeGuardCells) + 1;
sizeOfDoppler = 2 * (NODopplerTrainingCell + NODopplerGuardCells) + 1;
```

Figure 6: CFAR Parameters

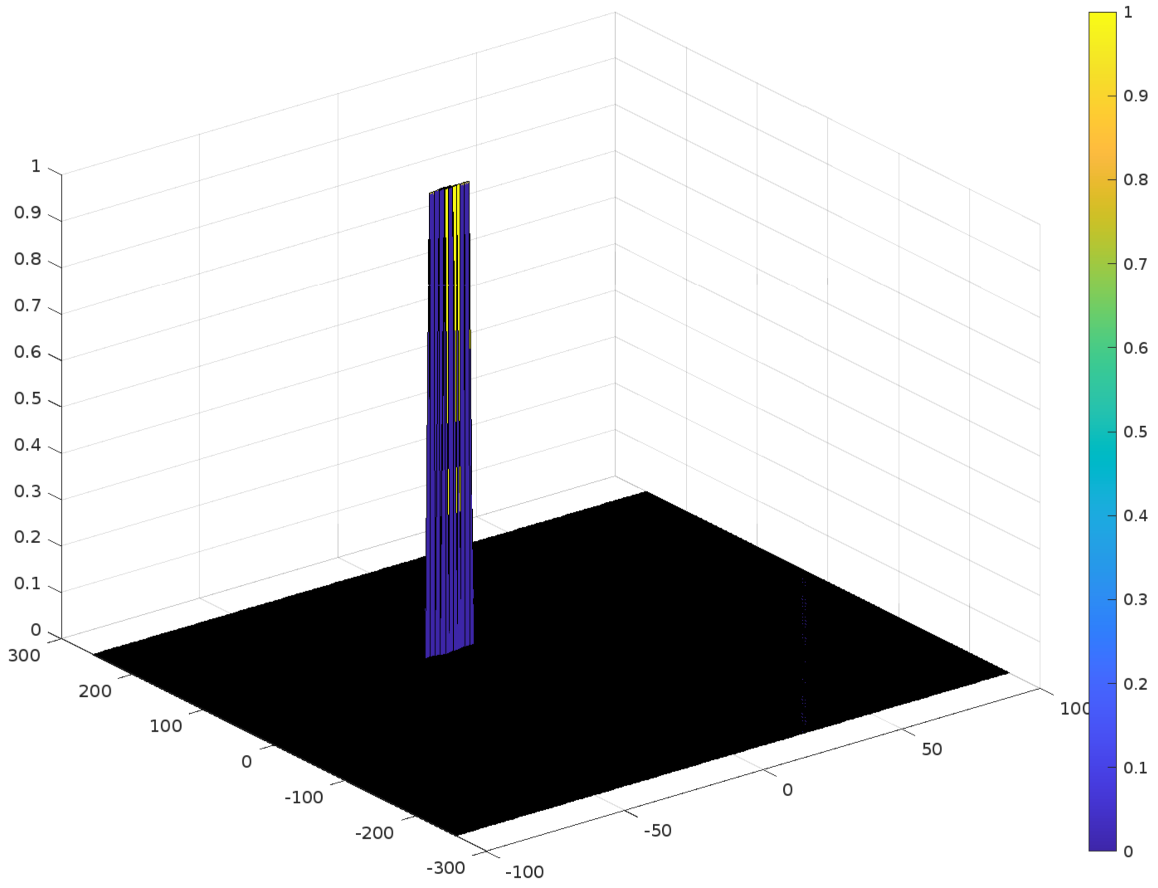


Figure 7: Constant False Alarm Rate detection results

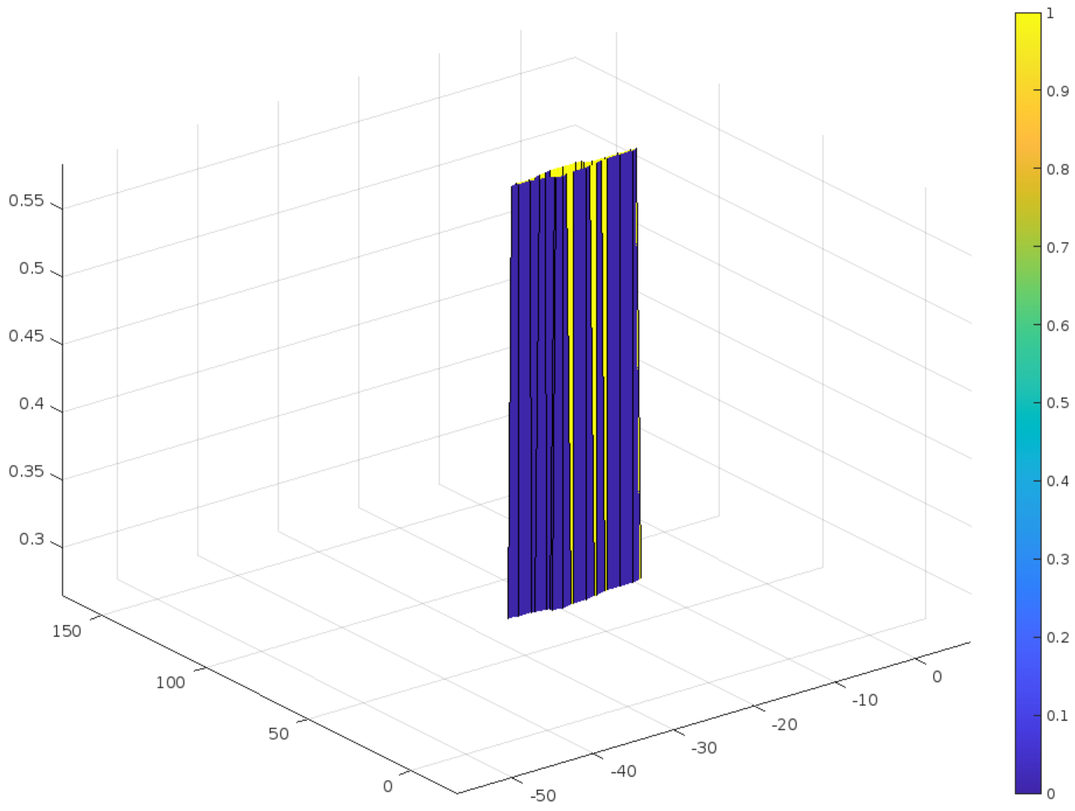


Figure 8: Constant False Alarm Rate detection results - Zoomed Version