Table of Contents


```
clear all
clc;
```

Radar Specifications

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Frequency of operation = 77GHz
% Max Range = 200m
% Range Resolution = 1 m
% Max Velocity = 100 m/s
%%%%%%%%%%%%%%%%%%%%%%%%%%%
radar_max_range = 200;
radar_range_resolution = 1;
radar_max_velocity = 100;
speed_of_light = 3e8;
```

User Defined Range and Velocity of target

```
% Here we define the target's initial position and velocity. Note :
 Velocity
% remains contant
target_range = 50;
target_velocity = -30;
```

FMCW Waveform Generation

```
% 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.
% Find the Bsweep of chirp for 1 m resolution

B = speed_of_light/(2*radar_range_resolution);

% Calculate the chirp time based on the Radar's Max Range

Tchirp = 5.5*(radar_max_range*2/speed_of_light);
% Slope is in line with the expectation: 2.0455e+13
```

```
slope = B/Tchirp;


% Operating carrier frequency of Radar
fc= 77e9;             %carrier freq


% The number of chirps in one sequence. Its ideal to have 2^ value for
 the ease of running the FFT
% for Doppler Estimation.
Nd=128;                   % #of doppler cells OR #of sent periods %
 number of chirps

% The number of samples on each chirp.
Nr=1024;                  %for length of time OR # of range cells

% Timestamp for running the displacement scenario for every sample on
 each
% chirp
t=linspace(0,Nd*Tchirp,Nr*Nd); %total time for samples


% Creating the vectors for Tx, Rx and Mix based on the total samples
 input.
Tx=zeros(1,length(t)); %transmitted signal
Rx=zeros(1,length(t)); %received signal
Mix = zeros(1,length(t)); %beat signal

% Similar vectors for range_covered and time delay.
r_t=zeros(1,length(t));
td=zeros(1,length(t));
```

Signal generation and Moving Target simulation

Running the radar scenario over the time by having the constant velocity model

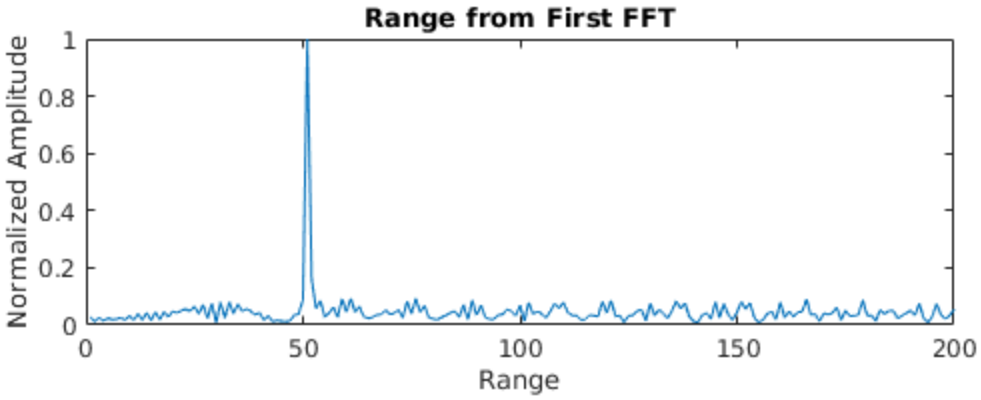
```
for i=1:length(t)

    %For each time stamp update the Range of the Target for constant
 velocity.
    r_t(i) = target_range + target_velocity*t(i);

    %For each time sample we need update the transmitted and
    %received signal.
    Tx(i) = cos(2*pi*(fc*t(i)+(slope*t(i)^2)/2));
    td(i) = (2*r_t(i))/speed_of_light;
    Rx (i) = cos(2*pi*(fc*(t(i)-td(i))+(slope*(t(i)-td(i))^2)/2));

    %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
```

RANGE MEASUREMENT

The task is to 1D FFT of the received signal after converting it 2D

```
%reshape the vector into Nr*Nd array. Nr and Nd here would also define
 the size of
%Range and Doppler FFT respectively.
Mix = reshape(Mix, [Nr,Nd]);

%run the FFT on the beat signal along the range bins dimension (Nr)
 and
%normalize.
FFT_beat = fft(Mix, Nr);
FFT_beat = FFT_beat./max(FFT_beat);

% Take the absolute value of FFT output
FFT_beat = abs(FFT_beat);


% Output of FFT is double sided signal, but we are interested in only
 one side of the spectrum.
% Hence we throw out half of the samples.
FFT_beat = FFT_beat(1:Nr/2 -1);

%plotting the range
figure ('Name','Range from First FFT')
subplot(2,1,1)

 % plot FFT output

plot(FFT_beat);
axis ([0 200 0 1]);
title('Range from First FFT');
ylabel('Normalized Amplitude');
xlabel('Range');
```

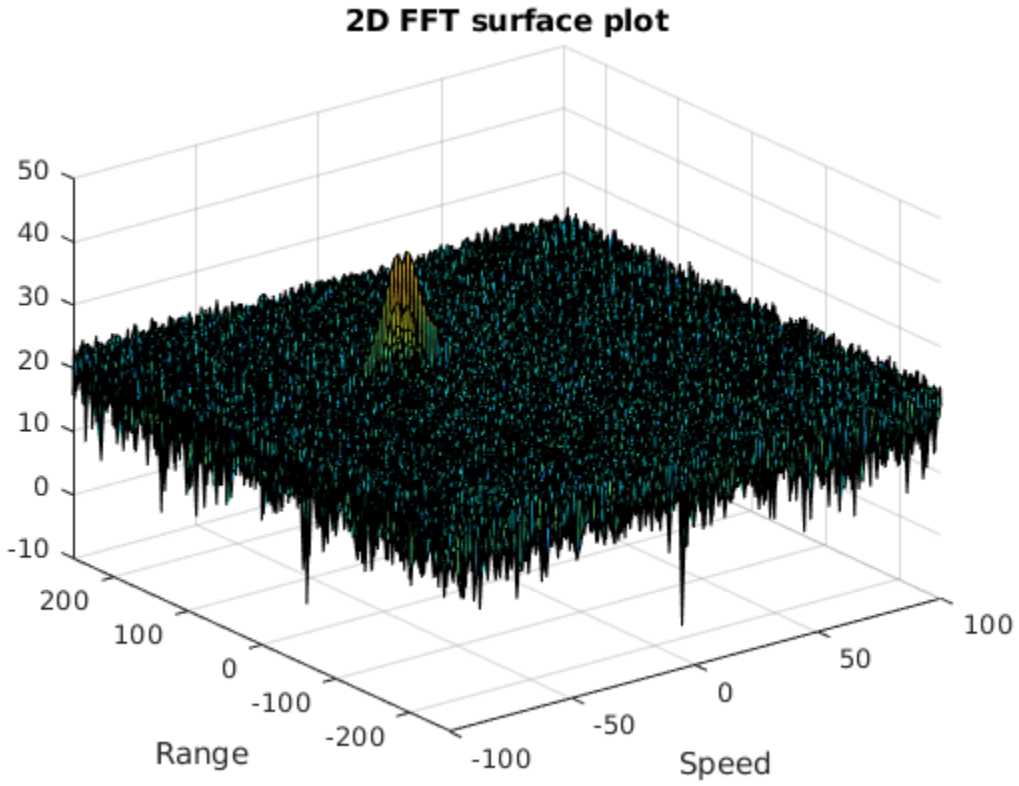


RANGE DOPPLER RESPONSE (already given)

The 2D FFT implementation is already provided here. This will run a 2DFFT on the mixed signal (beat signal) output and generate a range doppler map.You will implement CFAR on the generated RDM

```
% Range Doppler Map Generation.

% The output of the 2D FFT is an image that has reponse in the range
 and
% doppler FFT bins. So, it is important to convert the axis from bin
 sizes
% to range and doppler based on their Max values.

Mix=reshape(Mix,[Nr,Nd]);

% 2D FFT using the FFT size for both dimensions.
sig_fft2 = fft2(Mix,Nr,Nd);

% Taking just one side of signal from Range dimension.
sig_fft2 = sig_fft2(1:Nr/2,1:Nd);
sig_fft2 = fftshift (sig_fft2);
RDM = abs(sig_fft2);
RDM = 10*log10(RDM) ;

%use the surf function to plot the output of 2DFFT and to show axis in
 both
```

```
%dimensions
doppler_axis = linspace(-100,100,Nd);
range_axis = linspace(-200,200,Nr/2)*((Nr/2)/400);
figure,surf(doppler_axis,range_axis,RDM);
title( '2D FFT surface plot');
xlabel('Speed');
ylabel('Range');
```

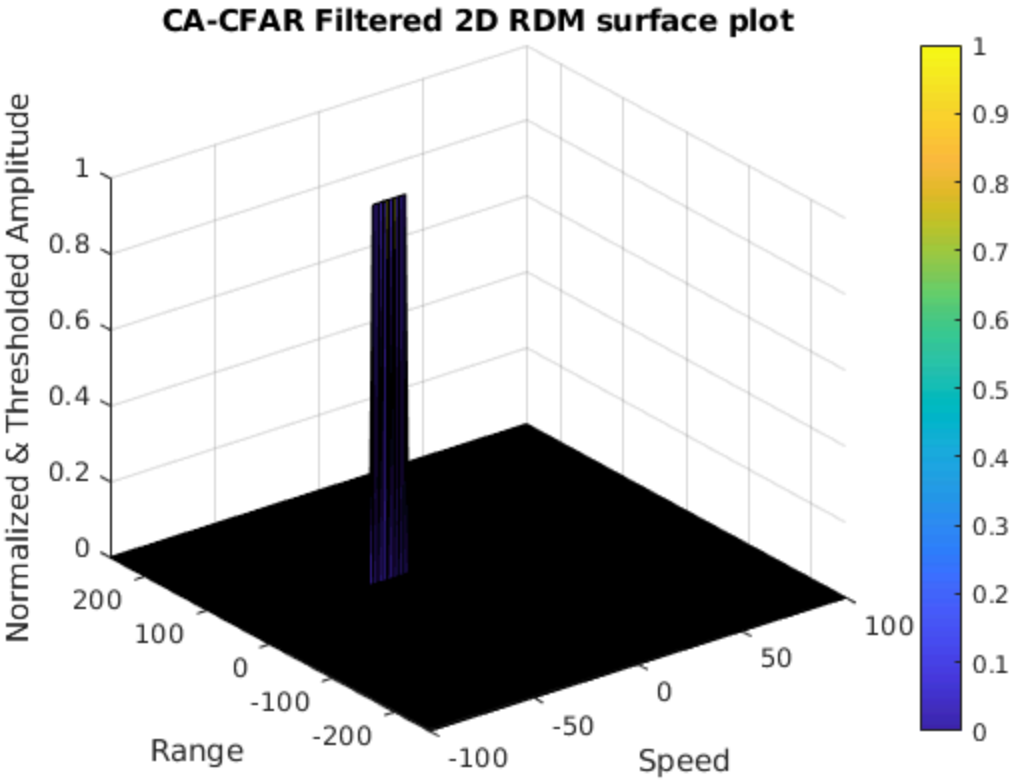


CFAR implementation

```
% Slide Window through the complete Range Doppler Map

%Select the number of Training Cells in both the dimensions.
Tr = 12;
Td = 10;

%Select the number of Guard Cells in both dimensions around the Cell
 under
%test (CUT) for accurate estimation
Gr = 4;
Gd = 4;

% offset the threshold by SNR value in dB
% This value will be calculated based on the quantile function
offset = 0;
```

```
%design a loop such that it slides the CUT across range doppler map by
%giving margins at the edges for Training and Guard Cells.
%For every iteration sum the signal level within all the training
%cells. To sum convert the value from logarithmic to linear using
 db2pow
%function. Average the summed values for all of the training
%cells used. After averaging convert it back to logarithimic using
 pow2db.
%Further add the offset to it to determine the threshold. Next,
 compare the
%signal under CUT with this threshold. If the CUT level > threshold
 assign
%it a value of 1, else equate it to 0.


% Use RDM[x,y] as the matrix from the output of 2D FFT for
 implementing
% CFAR

% normalize the range doppler matrix
RDM = RDM /max(RDM(:));
% define a 1D vector for the surrounding training cell values
std_of_noise = zeros((2*Tr + 2*Gr + 1)*(2*Td + 2*Gd + 1) - (2*Gr
+1)*(2*Gd+1) ,1);

for col = 1+Gr+Tr:(Nr/2)-(Gr+Tr)
    for row = 1+Gd+Td:(Nd)-(Gd+Td)
        %Create a vector to store noise_level for each iteration on
 training cells
        noise_level = zeros(1,1);
        i = 0;
        for col2 = col-(Gr+Tr): col+(Gr+Tr)
            for row2 = row-(Gd+Td): row+(Gd+Td)
                % Skip the CUT and guard cells
                if (abs(row-row2)> Gd || abs(col-col2)>Gr)
                    i = i + 1;
                    std_of_noise(i) = RDM(col2,row2);
                    noise_level = noise_level +
 db2pow(RDM(col2,row2));
                end
            end
        end
        % Calculate threshold from noise average then add the offset
        threshold = pow2db(noise_level / ((2*Tr + 2*Gr + 1)*(2*Td +
 2*Gd + 1) - (2*Gr+1)*(2*Gd+1)));
        % the offset value can also be identified as a function of
 standard
        % deviation. In our experiment 2*std(std_of_noise) has also
 worked
        % as a effective offset value
        offset = quantile(std_of_noise,0.8);

        % reset the vector contetn for the next calculation
        std_of_noise = zeros(length(std_of_noise),1);
```

```
            % redefine the threashold with the offset
            threshold = threshold + offset;

            cell_under_test = RDM(col,row);

            if (cell_under_test < threshold)
                RDM(col, row) = 0;
            else
                RDM(col, row) = 1;
            end
        end
    end

    % The process above will generate a thresholded block, which is
     smaller
    %than the Range Doppler Map as the CUT cannot be located at the edges
     of
    %matrix. Hence,few cells will not be thresholded. To keep the map size
     same
    % set those values to 0. Set any value apart from 0 and 1 to 0
    RDM(RDM~=0 & RDM~=1) = 0;


    %display the CFAR output using the Surf function like we did for Range
    %Doppler Response output.
    figure,surf(doppler_axis,range_axis,RDM);
    colorbar;
    title( 'CA-CFAR Filtered 2D RDM surface plot');
    xlabel('Speed');
    ylabel('Range');
    zlabel('Normalized & Thresholded Amplitude');
```



Published with MATLAB® R2021a