Homework 2 : Radar Imaging

Prof. Andrea Virgilio Monti Guarnieri & Dr. Marco Manzoni

Thibault Fievez 10729319
Group members : Hiva Amiri,Kimia Kiyan ,Vahid Rajabi

October 2020

1 Characteristic of the mission and Introduction

The characteristic and parameters of the mission are encoded at the beginning of our code as seen on figure 1.

```
%% initialization
clc
C = 3e8; % spead of light
B = 170e6; % bandwidth
kr = 1.597256e12; % chirp rate
fc = 5.42876e9;
lambda = C / fc;
fs = 24485000; % sampling freq
ts = 1/fs ;
beam_width = 0.1920;
prf = 307.292; % pulse repetition frequency
```

Figure 1: Parameters

It is also asked to provide the resolution in range and azimuth, as dictated by the following equations $\rho_r = \frac{C}{2B}$ and $\rho_y = \frac{\lambda}{2L_s}R$. [1] This range and azimuth resolution will dictate what is the minimal size of cell we should consider in our projection as seen on figure 2

2 Focusing by Time Domain Back Projection (TDBP)

2.1 Creating the back projection grid

As stated previously , our pixel size and therefore back projection grid must be generated by keeping in mind the range and azimuth resolution that limits our system.

[1] L_s is the length of the array , in our case , it is a synthetic array , therefore it is dictated by the speed of the plane and the number of sample generated.

```
%% Range/azimuth resolution
Range_resolution =  C / (2*B);

R0 = plane_position(3,1);
p1 = plane_position(:,1); % position of the first sensor

p2 = plane_position(:,2); % position of the second sensor
dis = norm( p2 - p1 ); % distance of 2 consequtive sensor/position of the plane of SAR
% we want the total distance of our Synthethic aperture array ,

tpr = 1/prf;
v = dis / tpr; % spead of the plane
Ls = v*tpr*3883;
Azimuth_resolution = (lambda/(2*Ls))*R0;

%% creating pixels :

dx_r = 0.75 * Range_resolution;
dx_a = 0.75 * Azimuth_resolution;


x_mesh = plane_position(2,1) : -dx_a : plane_position(2,1000); % plane_position(2,sensor_num);
y_mesh = plane_position(1,1) +5*dx_r : -dx_r : plane_position(1,1) -5*dx_r;

[pixel_x,pixel_y] = meshgrid(x_mesh,y_mesh);% creat the pixel , pixel_x containes the x postions of each pixel , same goes for y
pixel_size = size(pixel_x);
pixel_size_x = pixel_size(1,1)
pixel_size_y = pixel_size(1,2)
```

Figure 2: We extract the final and initial positions of the plane and compute both resolutions. with that we create pixels

It is good as a rule of thumb to use 0.75 times the resolution in azimuth and range to determine the size of our pixels.

2.2 Change of spatial reference system

From then, as explain on appendix B in the provided document, we convert our initial reference system, the position data of the plane from latitude , longitude and altitude (LLA) to an intermediate referential that yields Northing , Easting and altitude (ENU). To do this transformation we needed to use a reference , we settled on using the mean value of longitude and latitude. Finally we convert this into our final referential by applying a rotation matrix to the ENU coordinates so that the East and North axis should is in a way that the East axis aligns in the direction of the trajectory. From there we determine an azimuth, range , altitude reference system. This procedure can be visualised on figure 3.

```
%% changing the coordinates to Range/Azimuth


lat0 = mean(geom(2,:));
lon0 = mean(geom(3,:));
h0 = 0;

[row_num col_num] = size(geom);
ENU = zeros(3,col_num); % new matrix of positions
spheroid = wgs84Ellipsoid;
for  n = 1:1:col_num % creating new matrix of position by parsing the data
    lat  = geom(2,n); %take the second row of the geom which is lat
    lon = geom(3,n); %take the second row of the geom which is lon
    h = geom(4,n); %take the second row of the geom which is alt
    [easting,northing,z] =geodetic2enu(lat,lon,h,lat0,lon0,h0,spheroid); % LLA to ENU
    ENU(1,n) = easting;
    ENU(2,n) = northing;
    ENU(3,n) = z;
end

theta_temp = zeros(1,col_num); % initilaise the azimuth
for n = 1:1:col_num
    theta_temp(n) = atan(ENU(2,n)/ENU(1,n)); % computing angle for the rotation matrix
end
theta = mean(theta_temp); % mean value of the angle
% we take the mean value of theta betweent he postionion to rotate our axis
% to have x along the east and y along the north . x=range , y azimuth
RangeAzimuth = zeros(2,col_num);
rotation_matrix = [cos(theta) -sin(theta) ; sin(theta) cos(theta)]; % rotation matrix
for n=1:1:col_num

    A = [ENU(2,n) ;ENU(1,n)];

    R = rotation_matrix*A; % rotating the axis
   RangeAzimuth(1,n) = -R(1,1);
   RangeAzimuth(2,n) = R(2,1);
end

plane_position = [RangeAzimuth ; ENU(3,:)]; % Range azimuth matri contain range and azimuth , enu values is just the z ,  maka a matrix
sensor_num = size(plane_position);
```

Figure 3: Transformation of our spacial reference system from LLA to ENU to Range/Azimuth/height.

2.3 Distance Computations

In this section, now that the coordinates system has been redefined , we can compute the distance from each pixel to each element of our virtual array (each slow time position of the plane) as showcased on figure 4

```matlab
%% Distance Computation
temp_distance = zeros(1,sensor_num); % initialise matrix
distance = zeros(pixel_size_x*pixel_size_y,sensor_num); % matrix of projecting ( size of the grid on the ground)
% each row is pixel , each column is a distance to an atnenna
counter = 1;
for n = 1 : 1 : pixel_size_x
    for m =1 : 1 : pixel_size_y
        x_position = pixel_x(n,m);
        y_position = pixel_y(n,m);
        pixel_position = [x_position;y_position;0];

       for k = 1 : 1 : sensor_num
           temp_distance(1,k) = norm(pixel_position - plane_position(:,k)); % calculate distance between pixel and antenna
       end
       distance(counter,:) = temp_distance; % each row is pixel , each column is a distance to an atnenna
       counter = counter + 1 ; % stops at teh end of the for loop
    end
end
```

Figure 4: Calculating the distance from each pixel , to each of the 3884 "virtual antennas" created by the trajectory of our plane .

2.4 Focus pixel on the ground

This section is to artificially create the range compressed data that would have been receiving on the plane. Now that all the slow time distance to all pixel have been computed , we can find their corresponding double travel time (we are talking in fast time ,to go from the plane to the pixel and back) and map it to have the corresponding slow time.Also we can create the focus pixel as described on part 4.See figure 5

```matlab
%% focused pixel in ground
step = size(distance); % step(1) number of rows , all the pixel we have
A = zeros(1,step(1)); % as many column as pixels
for n = 1 : 1 : step(1) % number of pixel
    sum_temp = 0;
    for m = 1 : 1 : step(2) %number of the sattelite
        D_tow = distance(n,m);
        time = D_tow * 2 / C; % 2 bc back and forth
        mapped_time = floor(time/ts)+1; %maps to lower integer , this gives you the time index on the data that you should evaluate
        R = dat(mapped_time,m); %find the corresponding value in the data file
        sum_temp = sum_temp + R * exp(j * 4 * pi * distance(1,m) / lambda) * exp(-j * 4 * pi * kr * (distance(1,m)^2) / C^2 ); % summ all the contribution usig the formula
    end
    A(1,n) = sum_temp; %everything saved here
end
```

Figure 5: Creating Focused pixel

2.5 Frequency to distance convertion

The idea of this section , is to compute a FFT in fast time of the signal received by the plane , by doing so and coupled by that fact that we are using a chirp there is a clear correspondence between the frequency domain and the spacial (distance) domain[2]. So to say,the values of a pixel at distance r_1 is mapped in our signal at a frequency $f_1 = K_r \frac{2r_1}{C}$, with K_r being the chip rate , the speed at which the chip change in frequency with respect to time. Therefore for each virtual antenna, we can compute the FFT in fast time and transform the frequency axis into a distance axis, by applying the equation stated previously , and project to each pixel the corresponding signal value. By doing so and by adding the contribution of each virtual antenna we get our final desired "picture". In order to do so we will also use some zero padding (adding zeros at the end of our signal) while doing the FFT to "artificially" increase the number of possible indices in the frequency domain. It is important to keep in mind that no matter what the highest index corresponds to the sampling frequency f_s.

[2]See Appendix A of homework.

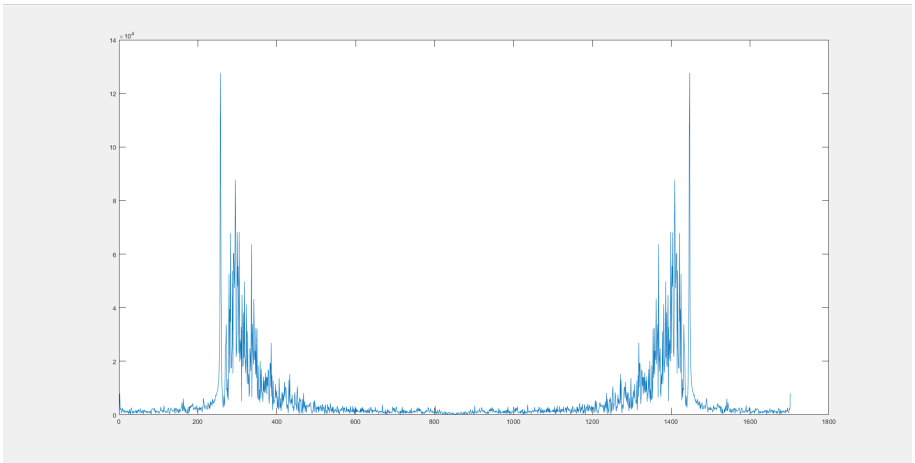

Figure 6: FFT in fast time of our received signal, without zero padding, the signal being real is symmetrical, so only the first half matters.

```
%% frequency _axis to range_axis

% computing range axis
Data=dat;
%  we have 3884 datas sets , but eah as 1702 samples in fast time
% let's 0 pad
FttData= zeros (16384,3884);
for n=1:1:3884
    FttData(:,n)= fft(Data(:,n),16384); % as proposed in the document
end
dn =0:fs/16384:fs-1/16384; % freq axis


dt = dn./kr; % time axis

dr = dt.* C/2; % range axis


plot(dr,FttData(:,1));


```

Figure 7: FFT in fast time with zero padding and range transformation.

2.6 Selecting a random pixel

To start off the TDBP we need to pick a target pixel in a random position on our grid, see figure 8.

```
%% plce the target in a random position

clc
%rand_x_pixel = floor(pixel_size_x*rand())+1; % pick a reandom pixel index
%rand_y_pixel = floor(pixel_size_y*rand())+1; % pick a reandom pixel index
rand_x_pixel = 4;
rand_y_pixel = 120;
x_position = pixel_x(rand_x_pixel,rand_y_pixel); % random point coordinates
y_position = pixel_y(rand_x_pixel,rand_y_pixel); % random point coordinates

```

Figure 8: Picking a random pixel from the grid

2.7 Range compressed data simulation

Now, the range compressed data is "artificially" generated according to the formula on point 5.2. In this formula W is the antenna pattern, we decided to go for a sinc antenna pattern for our implementation. That sinc has as argument all the angle divided by the beam width.See figure 9.

```
%% range compressed data computation
% calculating R0 = di[]sttance from the random target to the trajectory of
theta = -pi:2*pi/16384:pi - 1/16384;

row_num = (rand_x_pixel - 1) * pixel_size_y + rand_y_pixel;
target_dis = distance(row_num , :);
R_tow_r = zeros(3384,16384);

for n = 1 : 1 : 3384
    R_tow_r_temp = sinc(theta./beam_width).*sinc((dr - target_dis(1,n))/Range_resolution)*exp(-j*4*pi*target_dis(1,n)/lambda);
    R_tow_r(n,:) = R_tow_r_temp;

end
plot(dr,abs(R_tow_r(1,:)))
```

Figure 9: Generating the range compressed data

2.8 Time Domain back projection

For this section, we created two different implementation that also using different interpolation functions.

2.8.1 First implementation

As explain in section 2.5 , the idea is that for each slow time we find the corresponding ranged compressed data with the frequency/range mapping. Knowing the distance , we know the range , we therefore also know at what frequency to look at. Then having found the corresponding value , we find all the other pixels at the same distance from the SAR as the current one. We repeat this for each 3884 slow time and add each contribution. Then we should have a 2D SINC function around the main pixel position.

```
%% TDBP

distance;

grid = zeros(pixel_size_x,pixel_size_y); % creating the grids

target_dis;
f = FttData(:,1)';
plot(dr,abs(f));
% find the corresponding index of the range_comoressed_signal
for l = 1:1:3884

    for n = 1 : 1 : 16384
        if dr(1,n) > target_dis(1,l)
            n;
            break
        end
    end

    f = FttData(:,l)'; % choosing the fft of the corresponding slow time
    amp =abs(f(1,n))*exp(j*4*pi*distance(row_num ,l)/lambda); % finding the value of the ranged cpmressed data

    neighbor = [rand_x_pixel+1 rand_y_pixel;rand_x_pixel+1 rand_y_pixel+1;rand_x_pixel rand_y_pixel+1;
        rand_x_pixel-1 rand_y_pixel+1;rand_x_pixel-1 rand_y_pixel;rand_x_pixel-1 rand_y_pixel-1;
        rand_x_pixel rand_y_pixel-1;rand_x_pixel+1 rand_y_pixel-1]; %neighbors of the selected pixel
    neighbor_pos = zeros(8,3);

    for n = 1:1:8
        neighbor_pos(n,1) = pixel_x(neighbor(n,1),neighbor(n,2));
        neighbor_pos(n,2) = pixel_y(neighbor(n,1),neighbor(n,2));

    end
    neighbor_pos; % position of the neighbors
    dif = plane_position(:,l)'-neighbor_pos; % vector representing the distsnce of each neighbor to the plane position
    neighbor_dis = zeros(8,1);
    for m = 1:1:8
        neighbor_dis(m,1) = norm(dif(m,:));
    end
    neighbor_dis; % distance of each neighbor to the plane position
    comp_nei_dist =abs(neighbor_dis - target_dis(1,l)); %subtracting the distance of neighbor from the target distance
```

Figure 10: TDBP1.1

```
    comp_nei_dist =abs(neighbor_dis - target_dis(1,1)); %subtracting the distance of neighbor from the target distance
    % finding 2 minimum values
    [x1_min ind1_min] = min(comp_nei_dist);
    comp_nei_dist(ind1_min,1) = 10;

    [x2_min, ind2_min] = min(comp_nei_dist);
    ind = ind2_min;
    % now that we have find 2 pixels with the most similar distnce from the
    % plane position we should continue for each pixel in 2 for loop
    for k = 1 : 1 : max(pixel_size_x,pixel_size_y)


        if k ==1
            new_neighbor = neighbor(ind1_min,:);
        end

        rand_x_pixel_n1 = new_neighbor(1,1); % new x_pixel
        rand_y_pixel_n1 = new_neighbor(1,2); % new y_pixel
        row_num = (rand_x_pixel_n1 - 1) * rand_y_pixel_n1 + rand_y_pixel_n1; % corresponding distance from all the place position
        target_dis_1 = distance(row_num , :); % selecting the corresponding distance
        % % find the corresponding index of the range_comoressed_signal
        for n = 1 : 1 : 16384
            if dr(1,n) > target_dis_1(1,1)
                n;
                break
            end
        end
        f = FttData(:,1)';
        amp =abs(f(1,n))*exp(j*4*pi*distance(row_num ,1)/lambda);% finding the value of the ranged cpmressed data


        neighbor_1 = [rand_x_pixel_n1+1 rand_y_pixel_n1;rand_x_pixel_n1+1 rand_y_pixel_n1+1;rand_x_pixel_n1 rand_y_pixel_n1+1;
        rand_x_pixel_n1-1 rand_y_pixel_n1+1;rand_x_pixel_n1-1 rand_y_pixel_n1;rand_x_pixel_n1-1 rand_y_pixel_n1-1;
        rand_x_pixel_n1 rand_y_pixel_n1-1;rand_x_pixel_n1+1 rand_y_pixel_n1-1];
        neighbor_1; % new neighbors
        neighbor_pos_1 = zeros(8,3);
        for n = 1:1:8
            neighbor_pos_1(n,1) = pixel_x(neighbor_1(n,1),neighbor_1(n,2));
            neighbor_pos_1(n,2) = pixel_y(neighbor_1(n,1),neighbor_1(n,2));
        end
```

Figure 11: TDBP1.2

```
        end
        neighbor_pos_1; % new neighbors position
        dif_1 = plane_position(:,1)'-neighbor_pos_1;% vector representing the distsnce of each neighbor to the plane position
        neighbor_dis_1 = zeros(8,1);
        for m = 1:1:8
            neighbor_dis_1(m,1) = norm(dif_1(m,:)); % calculating the distance
        end
        if k == 1
            comp_nei_dist_1 =abs( neighbor_dis_1 - target_dis(1,1));  %subtracting the distance of neighbor from the target distance

        else
            comp_nei_dist_1 =abs( neighbor_dis_1 - temp_dis); %subtracting the distance of neighbor from the prevouse target distance
            % check wheather the distance is too different from the main
            % distance
            o = abs( neighbor_dis_1 - target_dis(1,1));
            if o>1
               break
            end

        end
        % find the the most similar distance
        [x1_min ind1_min] = min(comp_nei_dist_1);

        temp_dis = neighbor_dis_1(ind1_min,1);
        comp_nei_dist_1(ind1_min,1) = 10000;
        [x2_min ind2_min] = min(comp_nei_dist_1);

        new_neighbor = neighbor_1(ind2_min,:);
        grid(rand_x_pixel_n1,rand_y_pixel_n1) = grid(rand_x_pixel_n1,rand_y_pixel_n1) + amp;

        % check whether we have reached to the boundry of the pixel
        if new_neighbor(1,1) == 1
            break
        elseif new_neighbor(1,1) == pixel_size_x
            break
        elseif new_neighbor(1,2) == 1
            break
        elseif new_neighbor(1,2) == pixel_size_y
            break
        end
```

Figure 12: TDBP1.3

```
        end


    end
    for k = 1 : 1 : max(pixel_size_x,pixel_size_y)


        if k ==1
            new_neighbor = neighbor(ind,:);
        end

        rand_x_pixel_n2 = new_neighbor(1,1);% new x_pixel
        rand_y_pixel_n2 = new_neighbor(1,2);% new y_pixel
        row_num = (rand_x_pixel_n2 - 1) * rand_y_pixel_n2 + rand_y_pixel_n2; % corresponding distance from all the place position
        target_dis_2 = distance(row_num , :);% selecting the corresponding distance
        for n = 1 : 1 : 16384
            if dr(1,n) > target_dis_2(1,1)
                n;
                break
            end
        end
        f = FttData(:,1)';
        amp =abs(f(1,n)) * exp(j*4*pi*distance(row_num ,1)/lambda);% obtaining the corresponding value

        target_dis_2 = [rand_x_pixel_n2+1 rand_y_pixel_n2;rand_x_pixel_n2+1 rand_y_pixel_n2+1;rand_x_pixel_n2 rand_y_pixel_n2+1;
        rand_x_pixel_n2-1 rand_y_pixel_n2+1;rand_x_pixel_n2-1 rand_y_pixel_n2;rand_x_pixel_n2-1 rand_y_pixel_n2-1;
        rand_x_pixel_n2 rand_y_pixel_n2-1;rand_x_pixel_n2+1 rand_y_pixel_n2-1];
        target_dis_2; % new neihbor
        neighbor_pos_2 = zeros(8,3);
        for n = 1:1:8
            neighbor_pos_2(n,1) = pixel_x(target_dis_2(n,1),target_dis_2(n,2));
            neighbor_pos_2(n,2) = pixel_y(target_dis_2(n,1),target_dis_2(n,2));
        end
        neighbor_pos_2; %position of the neighbor
        dif_2 = plane_position(:,1)'-neighbor_pos_2; %vector representing the distsnce of each neighbor to the plane position

        neighbor_dis_2 = zeros(8,1);
        for m = 1:1:8
            neighbor_dis_2(m,1) = norm(dif_2(m,:));
        end
```

Figure 13: TDBP1.4

```
        end
        neighbor_dis_2; % distance of the neighbors
        if k == 1
            comp_nei_dist_1 =abs( neighbor_dis_2 - target_dis(1,1));%subtracting the distance of neighbor from the target distance

        else
            comp_nei_dist_1 =abs( neighbor_dis_2 - temp_dis);
            % check wheather the distance is too different from the main
            % distance
            o = abs( neighbor_dis_2 - target_dis(1,1));
            if o>1
               break
            end
        end
         % find the the most similar distance
        [x1_min ind1_min] = min(comp_nei_dist_1);

        temp_dis = neighbor_dis_2(ind1_min,1);
        comp_nei_dist_1(ind1_min,1) = 10000;
        [x2_min ind2_min] = min(comp_nei_dist_1);

        new_neighbor = target_dis_2(ind2_min,:);
        grid(rand_x_pixel_n2,rand_y_pixel_n2) = grid(rand_x_pixel_n2,rand_y_pixel_n2) + amp;

        % check whether we have reached to the boundry of the pixel
        if new_neighbor(1,1) == 1
            break
        elseif new_neighbor(1,1) == pixel_size_x
            break
        elseif new_neighbor(1,2) == 1
            break
        elseif new_neighbor(1,2) == pixel_size_y
            break
        end

    end



end
```

Figure 14: TDBP1.5

2.8.2 Second implementation

In this section we will use the interp1 function.

We start off by finding the slow time at which teh SAR see the selected pixel, therefore the angle of the target with respect to the positions of the SAR is computed. If the object is at a bigger angle than the beamwidth , we can consider that it is not seen by the plane.

```
%% find the times that the SAR see target
clc

i = 1;

for m = 1 : 1 : 3884

    z = plane_position(3,m);
    l = [plane_position(1,m),plane_position(2,m)];
    k = [y_position,x_position];

        if (atan(norm(l-k)/z)) < (beam_width/2)

            target_seen_slowtime(i)=m;
             i=i+1;
        end

end

```

Figure 15: Is the target seen by the SAR position.

The next step is to find all the pixels seen by the SAR in each individual slow time. By basing our calculation on the geometry we can easily extract the distance and azimuth that the plane observe at each slow time. Because we know the size of the pixel in the azimuth axis (Y) we can compute how many pixels the radar can pickup at each specific slow time but also the pixel that change in each consecutive slow time. We end up storing the totality of pixels seen by the radar along its trajectory.

```
%% find the seen pixels in the slow time according to the angle by plane



for m = 1:1:3884

     z = plane_position(3,m);
     az_seen = tan(beam_width/2)*z*2; % distance seen in each slow time
     az_seen_pixel = floor(az_seen/dx_a);% number of pixel seen in each slow time
end
start_seen_pixel = round(az_seen_pixel/2);
step_size_pixel = round((plane_position(2,1) - plane_position(2,2))/dx_a); % number of pixel added in each slow time

seen_pixel = zeros(3884,pixel_size_y);
cnt = 0;
% which pixels are seen in each slow time
for n = 1:1:3884
    if n < start_seen_pixel / step_size_pixel
        cnt = cnt + 1 ;
        for m =1:1:start_seen_pixel + step_size_pixel * (n-1)
            seen_pixel(n,m) = m;

        end
    else
        if az_seen_pixel + n * 5 < pixel_size_y
            for m = (n - cnt) * step_size_pixel : 1 : az_seen_pixel + (n - cnt) * step_size_pixel
                seen_pixel(n,m) = m;
            end
        else
            for m = (n - cnt) * step_size_pixel : 1 : pixel_size_y
                seen_pixel(n,m) = m;
            end
        end

    end

end

```

Figure 16: Determining which pixels from the grid are seen by the SAR

From that we can reevaluate the matrix of distance by only containing the pixels that are actually seen by the SAR. This matrix is a sub matrix of our original distance matrix. Now that that new matrix contains only the pixels of interest, we can proceed with the TDBP, once again in each slow time and only for the pixels seen by the SAR at that slow time moment we interpolate the ranged compressed signal [3] onto a matrix containing all the distance between the radar and the pixels. We then proceed to do that for each slow time positions and sum the contribution on each pixel respectively.

[3]because after doing the FFT and making the transformation to a distance axis , this range compressed data has a X axis corresponding to a distance, if a specific pixel is in between two represented "x"axis distances , we need to interpolate between those two indices

```
%% new distance matrix accordig to the rows
temp_distance = zeros(1,sensor_num); % initialise matrix
distance_1 = zeros(pixel_size_x*pixel_size_y,sensor_num); % matrix of projecting ( size of the grid on the ground)
% each row is pixel , each column is a distance to an atnenna
counter = 1;
for n = 1 : 1 : pixel_size_y
    for m =1 : 1 : pixel_size_x
        x_position = pixel_x(m,n);
        y_position = pixel_y(m,n);
        pixel_position = [x_position;y_position;0];

       for k = 1 : 1 : sensor_num
           temp_distance(1,k) = norm(pixel_position - plane_position(:,k)); % calculate distance between pixel and antenna
       end
       distance_1(counter,:) = temp_distance; % each row is pixel , each column is a distance to an atnenna
       counter = counter + 1 ; % stops at teh end of the for loop
    end
end

```

Figure 17: Reevaluating the distance matrix.

```
%% TDBP
clc
seen_step = size(target_seen_slowtime);
seen_step = seen_step(1,2);


grid = zeros(size(distance_1));
t = size(grid);
t = t(1,1);
for i = 1 : 1 : seen_step
    i
    slow_time = target_seen_slowtime(1,i);
    temp = seen_pixel(i,:);
    temp(temp==0) = [];
    siz = size(temp);
    siz = siz(1,2);
    xp = distance_1(temp(1,1):temp(1,siz),:); % distance matrix of seen pixels in selected slow times
    v = FttData(:,slow_time); % values of ranged compressed signal in seen slow time
    x = dr; % range complex axis
    vq = interp1(x,v,xp); % do interpolation
    vq_size = size(vq);
    vq_size = vq_size(1,1);
    % add zeros to the begining and the end of the distance matrix for
    % coherent summation
    if temp(1,1) == 1

        v_q = [vq ; zeros(t-vq_size,3884)];
        grid = grid + v_q ;
    else
        if siz < pixel_size_y
            v_q = [zeros(temp(1,1) , 3884); vq ; zeros(t -temp(1,1) - vq_size  )]
            grid = grid + v_q ;
        else
            v_q = [zeros(t -vq_size ,3884);vq];
            grid = grid + v_q ;
        end
    end
end

```

Figure 18: Appliying the TDBP but only considering the "seen" pixels.

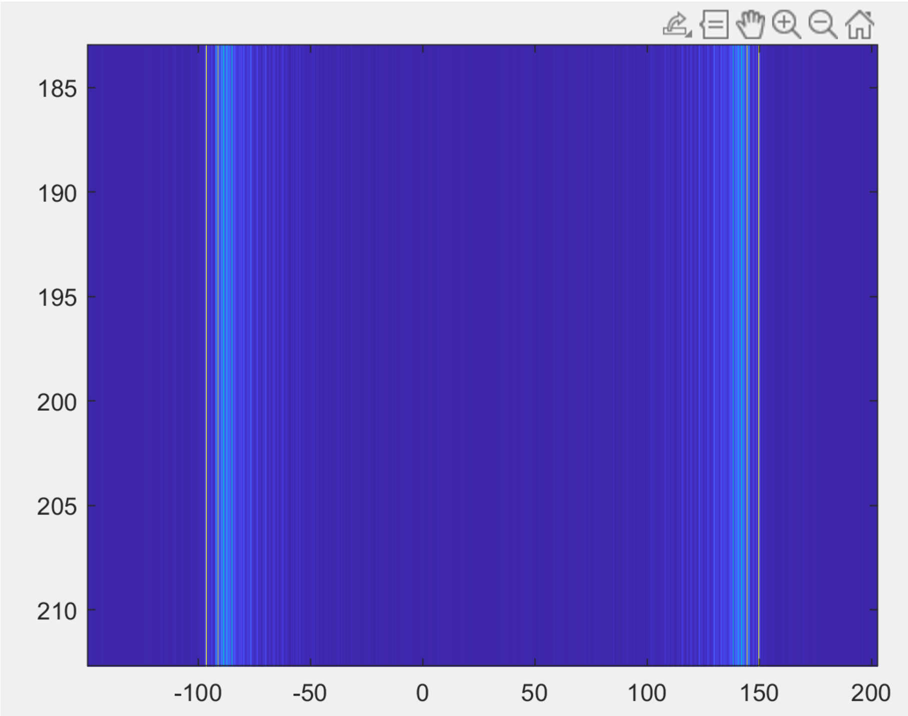

Figure 19: Range compressed pulse using imagesc.