Contents


```
% program: main_read_raw_adc_data_DCA_board_format.m
% updated: 02-August-2018

% This routine reads a frame of ADC data collected by the DCA board.
```

Part 1. Initial Setup

=======================

Define the filename

```
directory_filename   = '..\DCA_module\raw_data\';
suffix               = '_2018_1015v2';
filename_root        = ['xwr16xx_crw_dca_module',suffix,'.bin'];
filename             = [directory_filename,filename_root];
```

Verify that the filename is valid

```
good_filename  = exist(filename,'file');
if(good_filename == 2)
    disp(['reading file: ',filename,', ...']);
else
```

```
    disp(['not a valid filename: ',filename,', ...program paused...']);
    pause
end % end if(good_filename == 2)
```

```
reading file: ..\DCA_module\raw_data\xwr16xx_crw_dca_module_2018_1015v2.bin, ...
```

Define the radar model (used to define Q factor)

```
radar_model    = 16;    % for AWR1642
%radar_model    = 14;    % for AWR1443
```

Define format attributes

```
% change these radar parameters based on radar configuration

if(radar_model == 16)

   % These values are dependent on awr16xx chip
   num_ADCBits     = 16; % number of ADC bits per sample
   num_Tx          = 2; % number of transmitters
   num_Rx          = 4; % number of receivers
   num_Lanes       = 2; % do not change. number of lanes is always 2
   isReal          = 0; % set to 1 if real only data, 0 if complex data0

   % Get the radar operating parameters
   % This function call sets radar operating parameters for different user
   % defined input strings. Valid strings are:
   % 'best_range_res'       = best range resolution
   % 'DCA_configuration'    = configuration used in testing DCA output

   [chirp_parameters]    = func_defineChirpParameters( 'DCA_configuration');
   % The outputs are:
   % chirp_parameters = [f_start, alpha_chirp, N_sample, f_ADC_sample, ...
   %    N_chirp, N_frame, Frame_Duration, chirp_idle_time, chirp_ramp_time];

   num_samples     = chirp_parameters(3); % number of ADC samples per chirp
   num_chirps      = chirp_parameters(5);
   num_frames      = chirp_parameters(6);

end % end if(radar_model == 16)

% if(radar_model == 14)
%    num_ADCSamples   = NaN; % number of ADC samples per chirp
%    num_ADCBits      = 16; % number of ADC bits per sample
%    numTx           = 2; % number of transmitters
%    numRX           = 1; % number of receivers
%    numLanes        = 4; % do not change. number of lanes is always 4
%    isReal          = 0; % set to 1 if real only data, 0 if complex data0
% end % end if(radar_model == 16)
```

Part 2. Read the ADC voltages into Matlab workspace (*.bin file)

==

Open the *.bin file and read all of the data

```
fid      = fopen(filename,'r');
adcData  = fread(fid, 'int16');

% close the file
fclose(fid);

% get total file size
fileSize    = size(adcData, 1);
```

Decompose the data into (Tx, Rx) channels (8 virtual radars)

```
% The data format is defined in the mmwave_studio_user_guide on page 57.

% make up a number of chirps
%num_chirps           = 128;
block_size           = num_Tx*(num_Rx * num_samples * 2);
end_block_index      = 0;

Tx1_Rx_real              = zeros(num_Rx, num_frames, num_chirps, num_samples);
Tx1_Rx_imag              = zeros(num_Rx, num_frames, num_chirps, num_samples);
Tx2_Rx_real              = zeros(num_Rx, num_frames, num_chirps, num_samples);
Tx2_Rx_imag              = zeros(num_Rx, num_frames, num_chirps, num_samples);

Tx_Rx_real               = zeros(num_Tx, num_Rx, num_frames, num_chirps, num_samples);
Tx_Rx_imag               = zeros(num_Tx, num_Rx, num_frames, num_chirps, num_samples);

% process each frame
for k = 1:num_frames

   % process each chirp
   for i = 1:num_chirps

      % Get a block of data
      %start_block_index    = 1;
      start_block_index    = end_block_index + 1;
      end_block_index      = start_block_index + block_size - 1;

      % Get the data for this block
      data_block           = adcData(start_block_index:end_block_index);

      % Disect this block
      Tx1_real_part    = zeros(num_Rx,num_samples);
      Tx1_imag_part    = zeros(num_Rx,num_samples);
      Tx2_real_part    = zeros(num_Rx,num_samples);
      Tx2_imag_part    = zeros(num_Rx,num_samples);

      % Get the Tx1 data
      for r = 1:4
         index    = 0;
         for c = 1:num_samples/2
```

```
        % Get first real and imaginary elements
        index                = index + 1;
        j                    = ((r-1)*num_samples*2) + (c-1)*4 + 1;
        Tx1_real_part(r,index)    = data_block(j);
        j                    = ((r-1)*num_samples*2) + (c-1)*4 + 3;
        Tx1_imag_part(r,index)    = data_block(j);

        % get second real and imaginary elements
        index                = index + 1;
        j                    = ((r-1)*num_samples*2) + (c-1)*4 + 2;
        Tx1_real_part(r,index)    = data_block(j);
        j                    = ((r-1)*num_samples*2) + (c-1)*4 + 4;
        Tx1_imag_part(r,index)    = data_block(j);

    end % end for c loop

end % end for r loop

% where is the pointer after reading Tx1?
j_Tx1    = j;
% Get the Tx2 data
for r = 1:4
    index    = 0;
    for c = 1:num_samples/2

        % Get first real and imaginary elements
        index                = index + 1;
        j                    = j_Tx1 + ((r-1)*num_samples*2) + (c-1)*4 + 1;
        Tx2_real_part(r,index)    = data_block(j);
        j                    = j_Tx1 + ((r-1)*num_samples*2) + (c-1)*4 + 3;
        Tx2_imag_part(r,index)    = data_block(j);

        % get second real and imaginary elements
        index                = index + 1;
        j                    = j_Tx1 + ((r-1)*num_samples*2) + (c-1)*4 + 2;
        Tx2_real_part(r,index)    = data_block(j);
        j                    = j_Tx1 + ((r-1)*num_samples*2) + (c-1)*4 + 4;
        Tx2_imag_part(r,index)    = data_block(j);

    end % end for c loop

end % end for r loop


% Put this chirp of data into the master matrix
% oder is (num_RX, num_frames, num_chirps, num_ADCSamples)
Tx1_Rx_real(:,k,i,:)    = Tx1_real_part;
Tx1_Rx_imag(:,k,i,:)    = Tx1_imag_part;

Tx2_Rx_real(:,k,i,:)    = Tx2_real_part;
Tx2_Rx_imag(:,k,i,:)    = Tx2_imag_part;

Tx_Rx_real(1,:,k,i,:)    = Tx1_real_part;
Tx_Rx_imag(1,:,k,i,:)    = Tx1_imag_part;
Tx_Rx_real(2,:,k,i,:)    = Tx2_real_part;
Tx_Rx_imag(2,:,k,i,:)    = Tx2_imag_part;
```

```
    end % end for i loop

end % end for k loop
```

Part 3. Process each Frame of Data (Mimic Real-Time Processing)

===

```
% Define some terms
% This section processes only a single frame of data at a time.
% Thus, the code loops a total of num_frames times.
```

Define the Frame loop (the outer loop)

```
for r = 1:num_frames
```

(Process a single Frame)

A. Perform range_FFT on each chirp

```
    % define the range_FFT matrix
    range_FFT   = zeros(num_Tx, num_Rx, num_chirps, num_samples);

    % Define the Hanning window for N_sample range_FFT
    hann_range_window = hann(num_samples);

    % Process each Tx
    for i = 1:num_Tx

        % Process each Rx
        for j = 1:num_Rx

            % Process each chirp
            for k = 1:num_chirps

                % get the ADC I & Q voltages
                I_input  = squeeze(Tx_Rx_real(i,j,r,k,:));
                Q_input  = squeeze(Tx_Rx_imag(i,j,r,k,:));

                [I_input, Q_input] = func_rotate_IQ_time_series(I_input, Q_input);

                % apply a Hanning window to the input voltages
                I_input  = hann_range_window .* I_input;
                Q_input  = hann_range_window .* Q_input;

                [IQ_fft_complex, ~] = func_calc_complex_FFT(I_input, Q_input);

                range_FFT(i,j,k,:)  = IQ_fft_complex;

            end % end for c loop
        end % end for j loop
    end % end for i loop
```

B. Calculate mean profile for each (Tx,Rx) pair

```
range_FFT_mean_pow   = zeros(num_Tx, num_Rx, num_samples);
range_FFT_mean_real  = zeros(num_Tx, num_Rx, num_samples);
range_FFT_mean_imag  = zeros(num_Tx, num_Rx, num_samples);

for i = 1:num_Tx
   for j = 1:num_Rx

      % process each range gate
      for k = 1:num_samples

         % process each chirp
         accum_real    = 0;
         accum_imag    = 0;
         accum_pow     = 0;

         for c = 1:num_chirps

            real_part    = real(range_FFT(i,j,c,k));
            imag_part    = imag(range_FFT(i,j,c,k));

            accum_real   = accum_real + real_part;
            accum_imag   = accum_imag + imag_part;
            accum_pow    = accum_pow + real_part*real_part + imag_part*imag_part;

         end % end for k loop

         % save the accumulated values
         range_FFT_mean_real(i,j,k)     = accum_real;
         range_FFT_mean_imag(i,j,k)     = accum_imag;
         range_FFT_mean_pow(i,j,k)      = accum_pow;

      end % end for c loop

   end % end for j loop
end % end for i loop
```

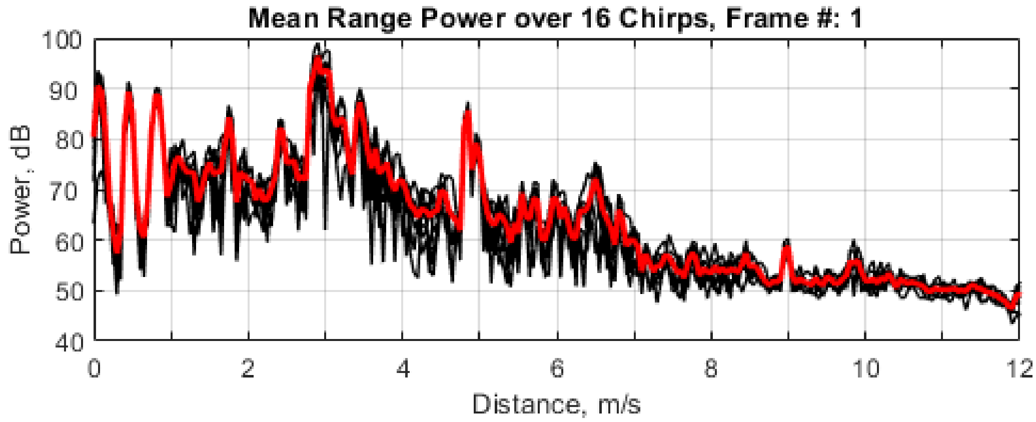
C. Calculate mean profile for all (Tx,Rx) profiles

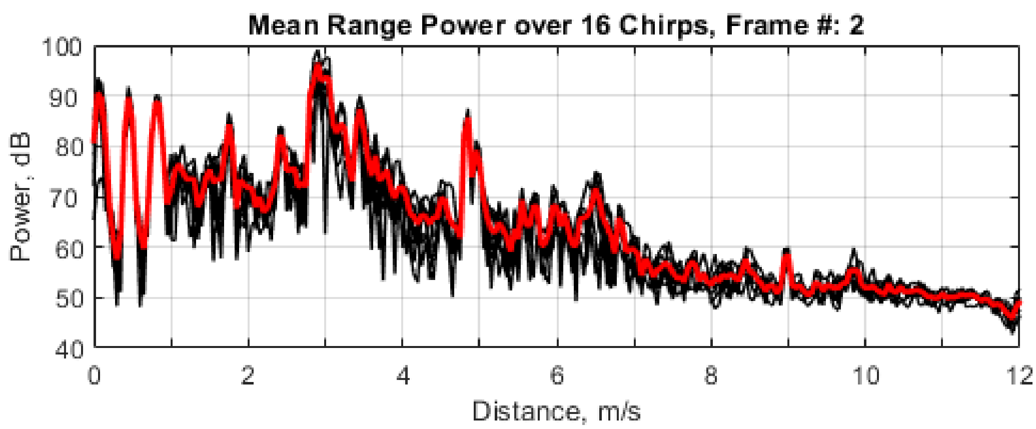
```
% This calculation is just for plotting purposes.
% Do not implement in xwr16xx processing.

range_FFT_gmean_lin      = zeros(1,num_samples);

for k = 1:num_samples

   accum     = 0;
   for i = 1:num_Tx
      for j = 1:num_Rx
         accum     = accum + range_FFT_mean_pow(i,j,k);
      end % end for j loop
   end % end for i loop

   % calculate the mean profile
```

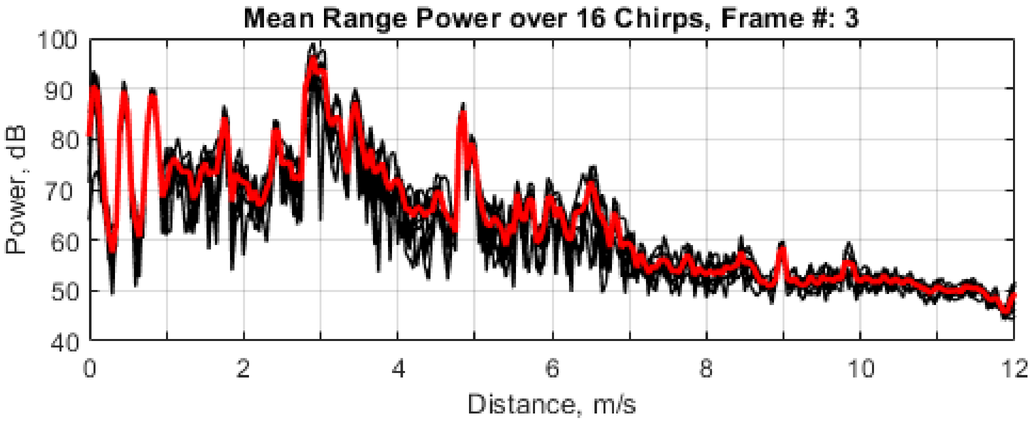
```
      range_FFT_gmean_lin(1,k)    = (accum)/(num_Tx*num_Rx);

   end % end for k loop
```

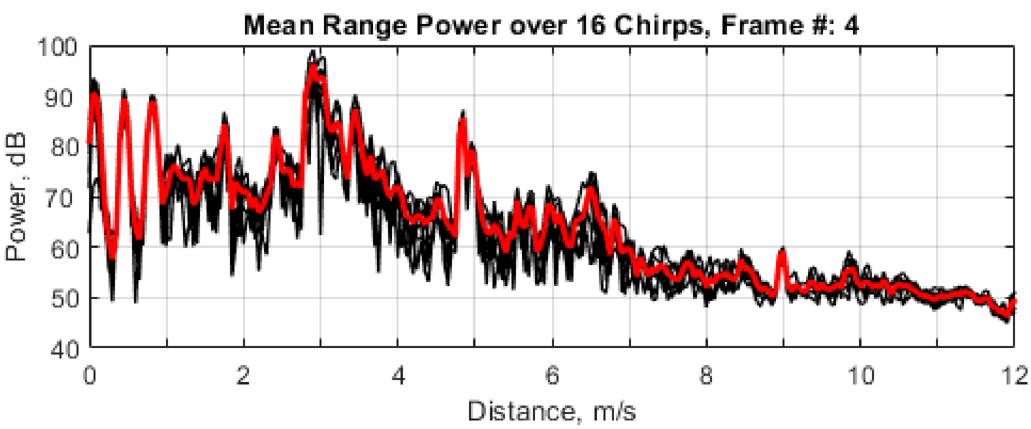
D. Plot the range_FFT magnitudes

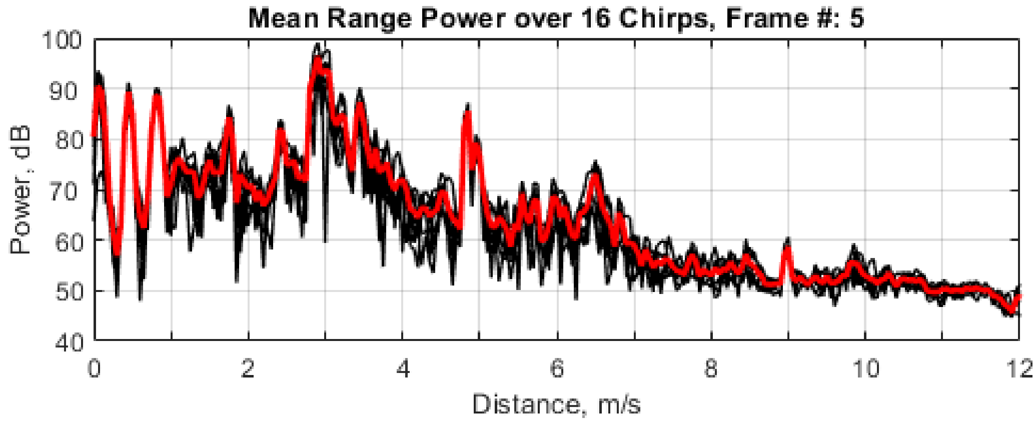
```
   range_distance = (0:1:num_samples-1) .* (0.05);

   close all
   figure
   subplot(2,1,1)
   plot(range_distance,10.*log10(range_FFT_gmean_lin),'r','linewidth',2)
   hold on
   for i = 1:num_Tx
      for j = 1:num_Rx
         plot(range_distance,10.*log10(squeeze(range_FFT_mean_pow(i,j,:))),'k','linewidth',1)
      end % end for j loop
   end % end for i loop
   plot(range_distance,10.*log10(range_FFT_gmean_lin),'r','linewidth',2)
   axis([0 12 40 100])
   grid on
   set(gca,'ytick',40:10:100,'yticklabel',{'40','50','60','70 ','80','90 ','100'});
   set(gca,'xtick',0:1:12,'xticklabel',{'0',' ','2',' ','4',' ','6',' ','8',' ','10',' ','12'
});
   xlabel('Distance, m/s')
   ylabel('Power, dB')
   title(['Mean Range Power over ',num2str(num_chirps),' Chirps, Frame #: ',num2str(r)]);

   filename = ['mean_range_power_frame_',num2str_2digits(r),'.tif'];
   print('-dtiff',filename);
```

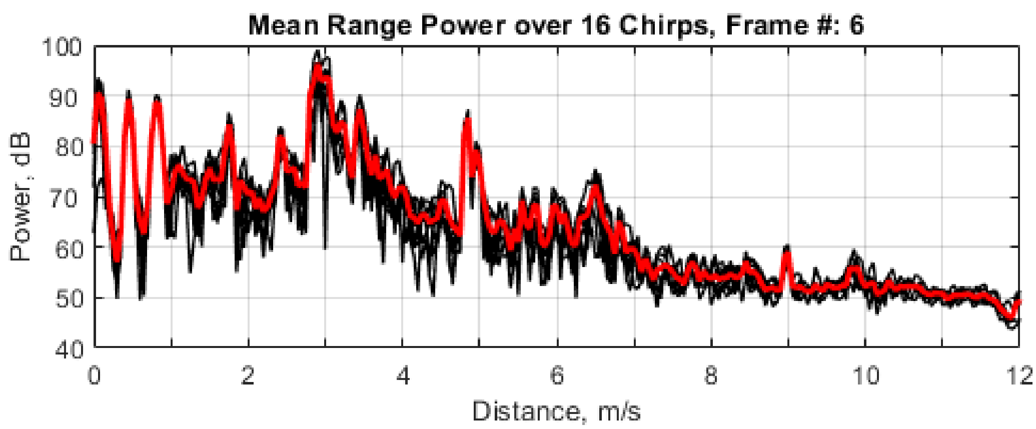

Mean Range Power over 16 Chirps, Frame #: 1
Power, dB
Distance, m/s
100
90
80
70
60
50
40
0
2
4
6
8
10
12

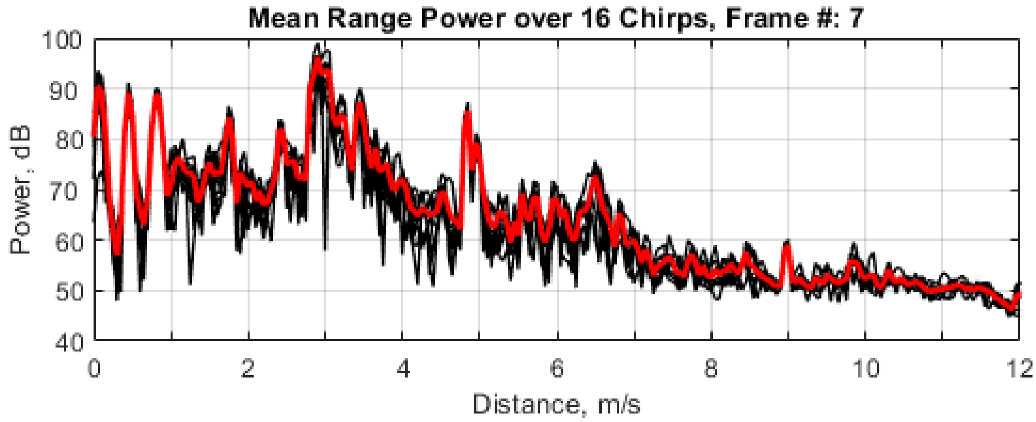

Mean Range Power over 16 Chirps, Frame #: 2
Power, dB
Distance, m/s
100
90
80
70
60
50
40
0
2
4
6
8
10
12

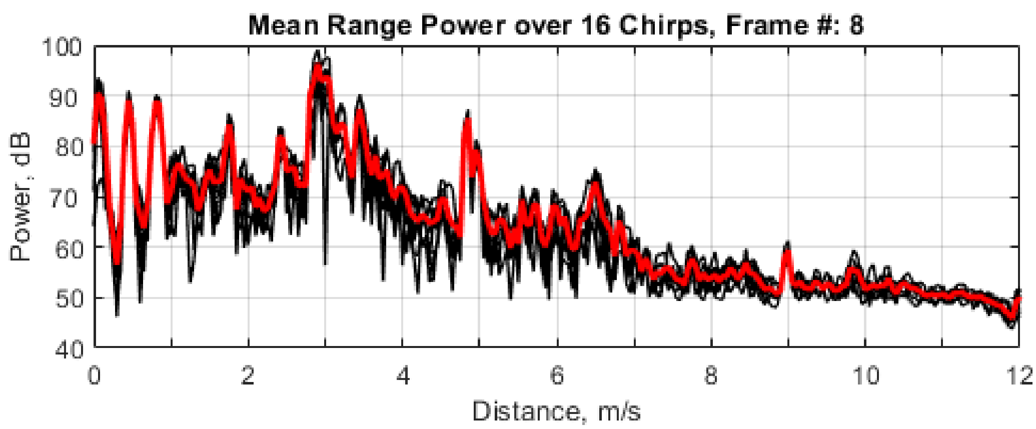

Mean Range Power over 16 Chirps, Frame #: 3
Power, dB
100
90
80
70
60
50
40
0
2
4
6
8
10
12
Distance, m/s


Mean Range Power over 16 Chirps, Frame #: 4
Power, dB
100
90
80
70
60
50
40
0
2
4
6
8
10
12
Distance, m/s


Mean Range Power over 16 Chirps, Frame #: 5
Power, dB
100
90
80
70
60
50
40
0
2
4
6
8
10
12
Distance, m/s


Mean Range Power over 16 Chirps, Frame #: 6
Power, dB
100
90
80
70
60
50
40
0
2
4
6
8
10
12
Distance, m/s





E. Calculate angle-FFT for each range gate

```
% Define the number of points in the angle-FFT
num_angles              = 64;

% The angle-of-arrival is not uniform. Estimate angle_of_arrival.

% Define the linear spacing for each bin of the angle_FFT
delta_radian   = (2*pi)/(num_angles);
angle_bin_rad = -1*pi + ((1:num_angles)-1) * delta_radian;
%angle_bin_deg = angle_bin_rad * (180/pi);

% Define the angle of arrival for each bin of angle_FFT
% This accounts for delta_phi = k*(lambda/2) sin(theta)
% delta_phi = (2*pi/lambda)*(lambda/2) sin(theta) = pi*sin(theta)
% where theta is the angle-of-arrival

%angle_of_arrival_deg    = zeros(1,num_angles);
angle_of_arrival_rad    = zeros(1,num_angles);
for c = 1:num_angles
   %angle_of_arrival_deg(c) = asin(angle_bin_rad(c)/pi) * (180/pi);
   angle_of_arrival_rad(c) = asin(angle_bin_rad(c)/pi);
end % end for c loop

% Estimate the angle-of-arrival for each range gate
% =================================================
range_angle_FFT_pow   = zeros(num_angles, num_samples);

% Loop through all ranges
for k = 1:num_samples

   % pre-define the real and imag input vectors
   I_input  = zeros(num_angles,1);
   Q_input  = zeros(num_angles,1);

   % put the observations into the first num_Tx * num_Rx locations
   index                            = 0;

   for i = 1:num_Tx
      for j = 1:num_Rx
         index                                  = index + 1;
         I_input(index)        = range_FFT_mean_real(i,j,k);
         Q_input(index)        = range_FFT_mean_imag(i,j,k);
      end % end for c loop
   end % end for r loop

   % Keep only the power of the computed FFT
   [~, IQ_fft_power]           = func_calc_complex_FFT(I_input, Q_input);

   range_angle_FFT_pow(:,k)    = IQ_fft_power;

end % end for k loop
```

F. Find single target in each range gate

```
% Use the angle-FFT method (it is much faster and less noisy)
% Input variables:
```

```
%      range_FFT_angle_mag(:,r,c,k)    = angle_FFT_power;
% dimensions: (range_angle_deg, num_frames, num_chirps, num_ADCSamples)

single_target_mag    = zeros(1, num_samples);
single_target_ang    = zeros(1, num_samples);
single_target_x      = zeros(1, num_samples);
single_target_y      = zeros(1, num_samples);

% Look only in a limited angle range +/- of boresight
% Define the width of angles to search
% With num_angles = 64:
% angle bins  9, 10, 11, 12 = -48.59, -45.95, -43.43, -41.01
% angle bins 54, 55, 56, 57 =  41.01,  43.43,  45.95,  48.59

%max_look_angle          = 46;
%angle_index             = abs(range_angle_deg) <= max_look_angle;
%ang_array               = range_angle_deg(angle_index);
start_index             = 10;
end_index               = 56;
short_ang_array         = angle_of_arrival_rad(start_index:end_index);

% process each range gate
for k = 1:num_samples

   % Get the values between -46 and +46 degrees
   short_mag_array         = range_angle_FFT_pow(start_index:end_index,k);

   % Find the largest magnitude in this range gate
   [max_value, max_index]  = max(short_mag_array);

   % save these values
   single_target_mag(1,k)  = max_value;
   single_target_ang(1,k)  = short_ang_array(max_index); % in radians

   single_target_x(1,k)    = range_distance(k) * sin(single_target_ang(1,k));
   single_target_y(1,k)    = range_distance(k) * cos(single_target_ang(1,k));

end % end for k loop
```

G. Find Three targets per range-gate

```
% Use the angle-FFT method (it is much faster and less noisy)
% Input variables:
%      range_FFT_angle_mag(:,r,c,k)    = angle_FFT_power;
% dimensions: (range_angle_deg, num_frames, num_chirps, num_ADCSamples)

% Find the peak (primary target).
% Look to left for secondary target based on valley threshold
% Look to right for secondary target based on valley threshold

primary_target_mag   = zeros(1, num_samples);
primary_target_ang   = zeros(1, num_samples);
primary_target_x     = zeros(1, num_samples);
primary_target_y     = zeros(1, num_samples);

left_target_mag   = zeros(1, num_samples);
```

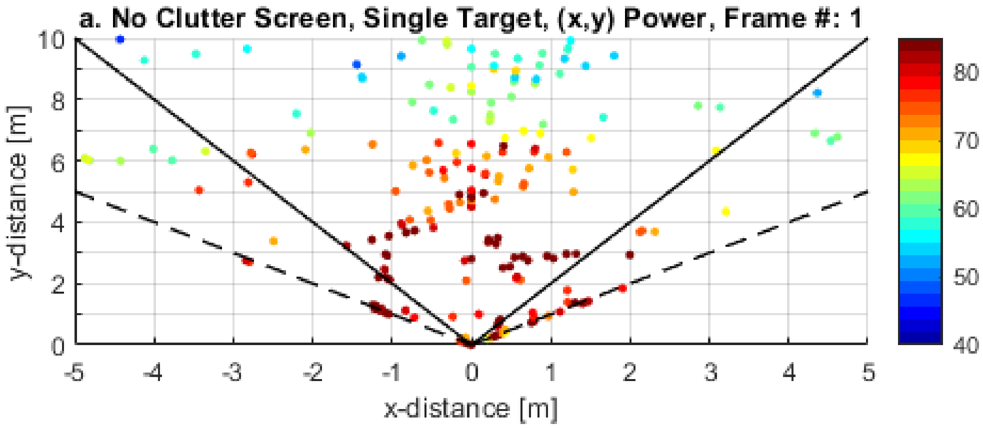
```
left_target_ang   = zeros(1, num_samples);
left_target_x     = zeros(1, num_samples);
left_target_y     = zeros(1, num_samples);

left_null_mag   = zeros(1, num_samples);
left_null_ang   = zeros(1, num_samples);
left_null_x     = zeros(1, num_samples);
left_null_y     = zeros(1, num_samples);

right_target_mag   = zeros(1, num_samples);
right_target_ang   = zeros(1, num_samples);
right_target_x     = zeros(1, num_samples);
right_target_y     = zeros(1, num_samples);

right_null_mag   = zeros(1, num_samples);
right_null_ang   = zeros(1, num_samples);
right_null_x     = zeros(1, num_samples);
right_null_y     = zeros(1, num_samples);

% Look only in a limited angle range +/- of boresight
% Define the width of angles to search
% With num_angles = 64:
% angle bins  9, 10, 11, 12 = -48.59, -45.95, -43.43, -41.01
% angle bins 54, 55, 56, 57 =  41.01,  43.43,  45.95,  48.59

%max_look_angle          = 46;
%angle_index             = abs(range_angle_deg) <= max_look_angle;
%ang_array               = range_angle_deg(angle_index);
start_index             = 10;
end_index               = 56;
short_ang_array         = angle_of_arrival_rad(start_index:end_index);
max_num_angles          = length(short_ang_array);

% process each range gate
for k = 1:num_samples

   % Find the primary peak attributes
   % ================================

   % Get the values between -46 and +46 degrees
   short_mag_array   = range_angle_FFT_pow(start_index:end_index,k);

   % Find the largest magnitude in this range gate
   [max_value, max_index] = max(short_mag_array);

   % save these values
   primary_target_mag(1,k)  = max_value;
   primary_target_ang(1,k)  = short_ang_array(max_index); % in radians

   % find the left target
   % ============================
   % The magnitude will first drop to the null.
   % Then, magnitude will increase.
   % Find either the next peak to the left (or the edge angle).

   [left_target_mag(1,k), left_target_ang(1,k), ...
      left_null_mag(1,k), left_null_ang(1,k)] = ...
```

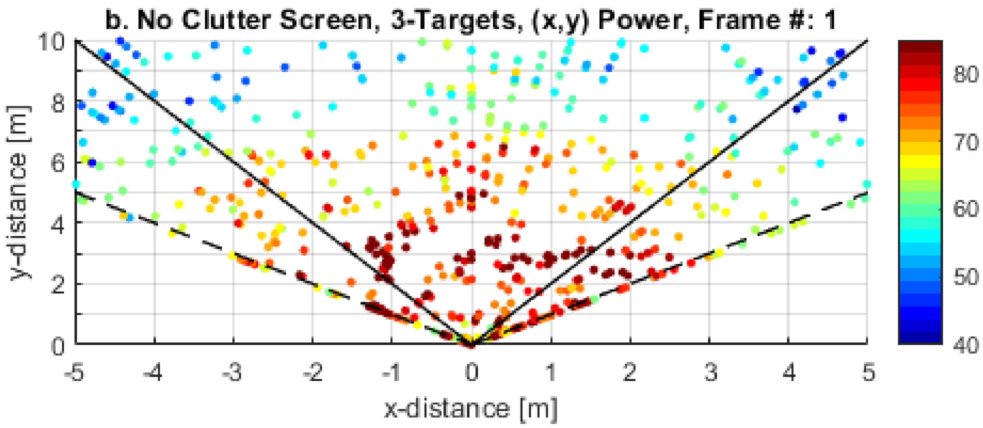
```
        func_find_left_target(max_value, max_index, short_mag_array, short_ang_array);

    % Find the right target
    % ============================
    % The magnitude will first drop to the null.
    % Then, magnitude will increase.
    % Find either the next peak to the right (or the edge angle).

    [right_target_mag(1,k), right_target_ang(1,k), ...
       right_null_mag(1,k), right_null_ang(1,k)] = ...
       func_find_right_target(max_value, max_index, short_mag_array, short_ang_array,...
       max_num_angles);

    % Calculate the (x,y) corridinates
    % ================================
    primary_target_x(1,k)    = range_distance(k) * sin(primary_target_ang(1,k));
    primary_target_y(1,k)    = range_distance(k) * cos(primary_target_ang(1,k));

    left_null_x(1,k)        = range_distance(k) * sin(left_null_ang(1,k));
    left_null_y(1,k)        = range_distance(k) * cos(left_null_ang(1,k));

    left_target_x(1,k)      = range_distance(k) * sin(left_target_ang(1,k));
    left_target_y(1,k)      = range_distance(k) * cos(left_target_ang(1,k));

    right_null_x(1,k)       = range_distance(k) * sin(right_null_ang(1,k));
    right_null_y(1,k)       = range_distance(k) * cos(right_null_ang(1,k));

    right_target_x(1,k)     = range_distance(k) * sin(right_target_ang(1,k));
    right_target_y(1,k)     = range_distance(k) * cos(right_target_ang(1,k));

end % end for k loop
```

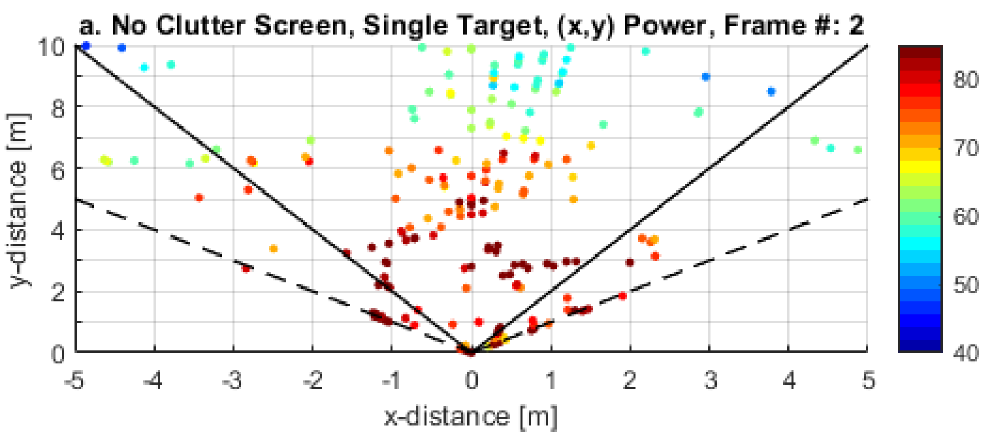
H. Plot the single target & three targets per range gate

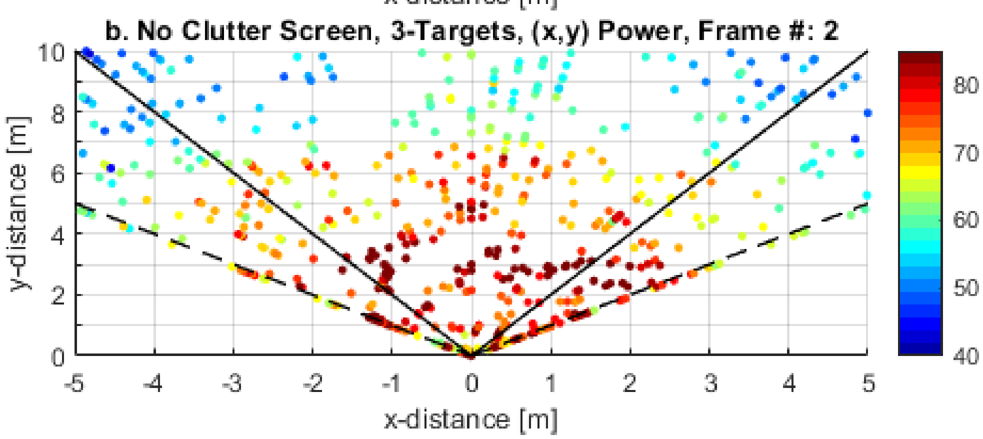
```
% Do not include in xwr16xx processsing

close all
figure
colormap('jet(24)')

subplot(2,1,1)
scatter(single_target_x,single_target_y,10,10.*log10(single_target_mag),'filled');
colorbar
caxis([40 85])
axis([-5 5 0 10])
grid on
hold on
% (x,y) = (3,5) ==> theta ~= 31 degrees
% (x,y) = (2.89,5) ==> theta ~= 30 degrees
% (x,y) = (3.5,5) ==> theta ~= 45 degrees
plot([-5  0],[10 0],'k','linewidth',1)
plot([ 0  5],[0 10],'k','linewidth',1)
plot([-5  0],[5 0],'k--','linewidth',1)
plot([ 0  5],[0 5],'k--','linewidth',1)

set(gca,'xtick',-5:1:5,'xticklabel',{'-5','-4','-3','-2','-1','0','1','2','3','4','5'});
```

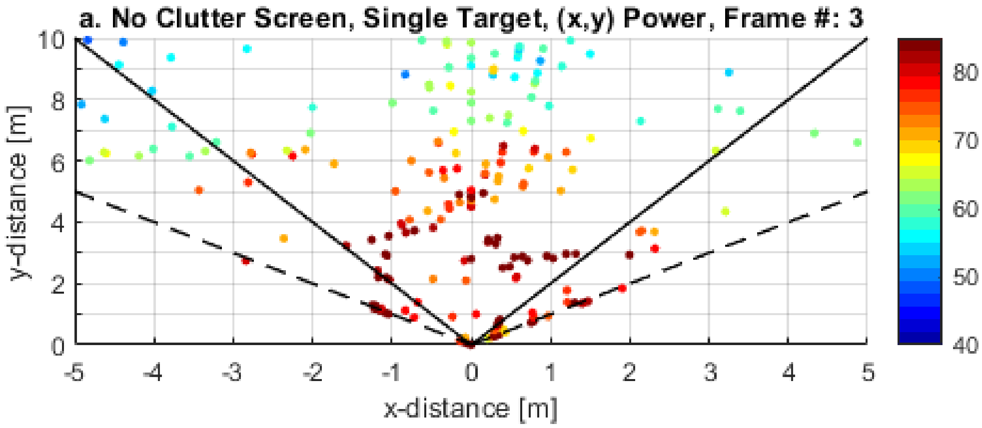
```
set(gca,'ytick',0:1:10,'yticklabel',{'0',' ','2',' ','4',' ','6',' ','8',' ','10'});
xlabel('x-distance [m]');
ylabel('y-distance [m]')
title(['a. No Clutter Screen, Single Target, (x,y) Power, Frame #: ',num2str(r)])

subplot(2,1,2)
scatter(primary_target_x,primary_target_y,10,10.*log10(primary_target_mag),'fill')
colorbar
caxis([40 85])
axis([-5 5 0 10])
grid on
hold on
scatter(left_target_x,left_target_y,10,10.*log10(left_target_mag),'fill')
scatter(right_target_x,right_target_y,10,10.*log10(right_target_mag),'fill')

% (x,y) = (3,5) ==> theta ~= 31 degrees
% (x,y) = (2.89,5) ==> theta ~= 30 degrees
% (x,y) = (3.5,5) ==> theta ~= 45 degrees
plot([-5  0],[10 0],'k','linewidth',1)
plot([ 0  5],[0 10],'k','linewidth',1)
plot([-5  0],[5 0],'k--','linewidth',1)
plot([ 0  5],[0 5],'k--','linewidth',1)

set(gca,'xtick',-5:1:5,'xticklabel',{'-5','-4','-3','-2','-1','0','1','2','3','4','5'});
set(gca,'ytick',0:1:10,'yticklabel',{'0',' ','2',' ','4',' ','6',' ','8',' ','10'});

xlabel('x-distance [m]');
ylabel('y-distance [m]')
title(['b. No Clutter Screen, 3-Targets, (x,y) Power, Frame #: ',num2str(r)])

filename = ['target_xy_plot_frame_',num2str_2digits(r),'.tif'];
print('-dtiff',filename);
```

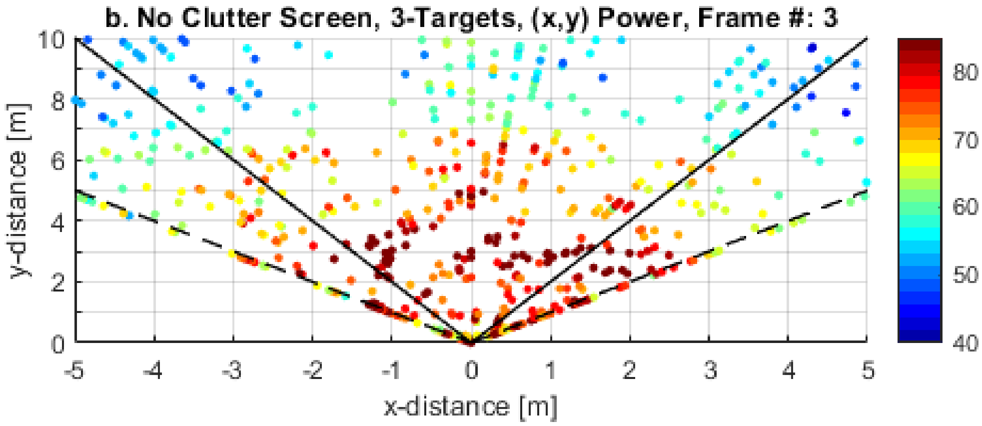

a. No Clutter Screen, Single Target, (x,y) Power, Frame #: 1
y-distance [m]
x-distance [m]
40
50
60
70
80

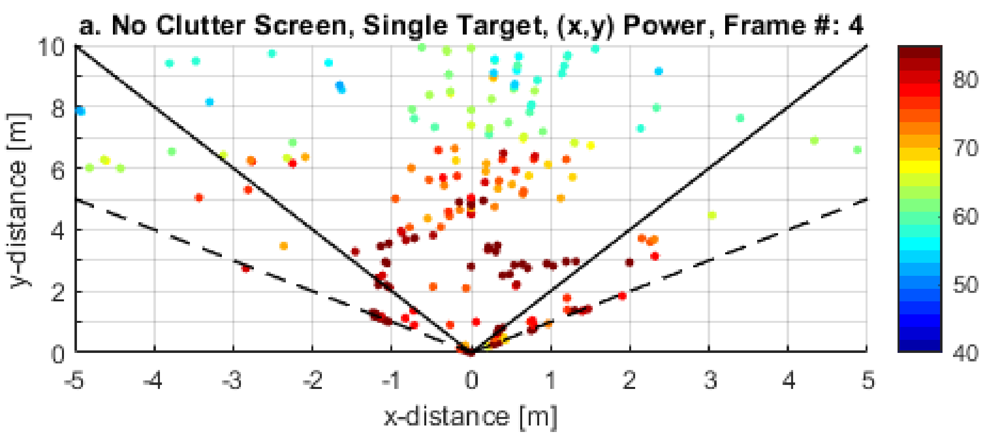

b. No Clutter Screen, 3-Targets, (x,y) Power, Frame #: 1
y-distance [m]
x-distance [m]
40
50
60
70
80

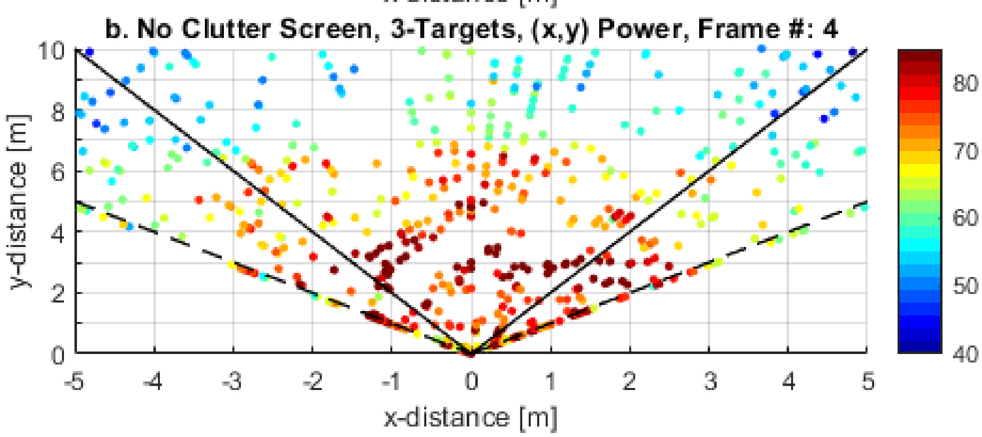

a. No Clutter Screen, Single Target, (x,y) Power, Frame #: 2
y-distance [m]
x-distance [m]
40
50
60
70
80

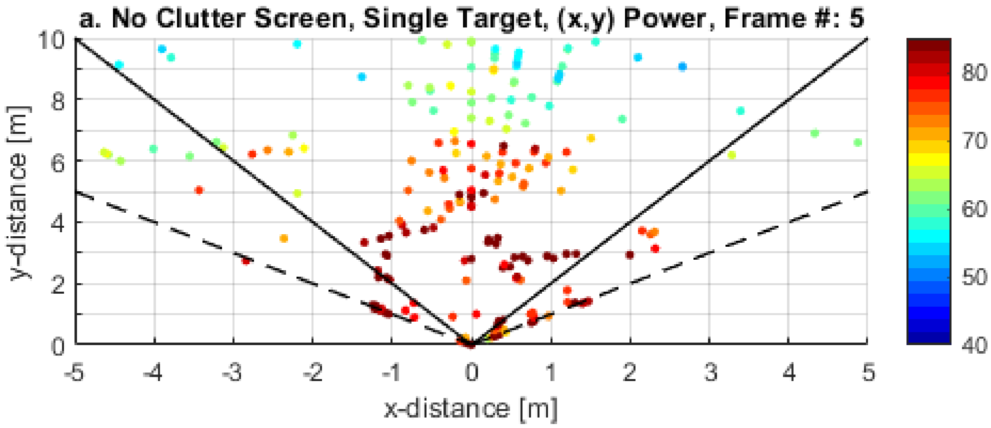

b. No Clutter Screen, 3-Targets, (x,y) Power, Frame #: 2
y-distance [m]
x-distance [m]
40
50
60
70
80

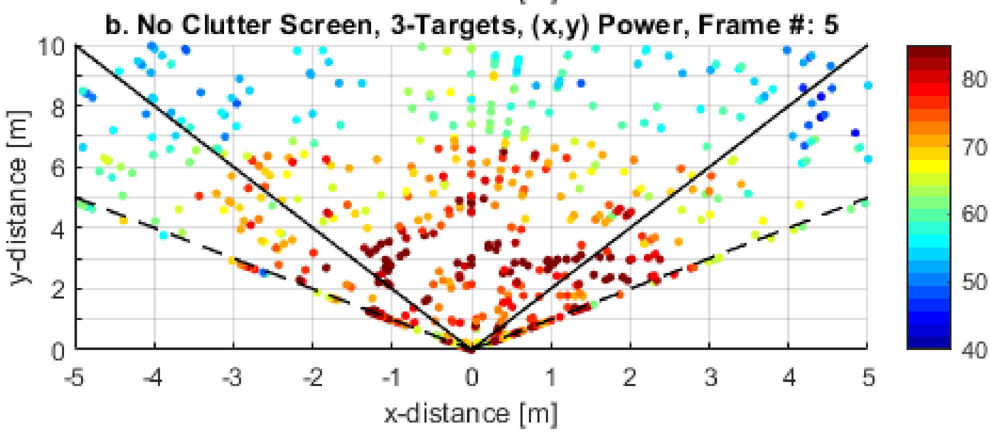

a. No Clutter Screen, Single Target, (x,y) Power, Frame #: 3
y-distance [m]
x-distance [m]
10
8
6
4
2
0
-5
-4
-3
-2
-1
0
1
2
3
4
5
80
70
60
50
40

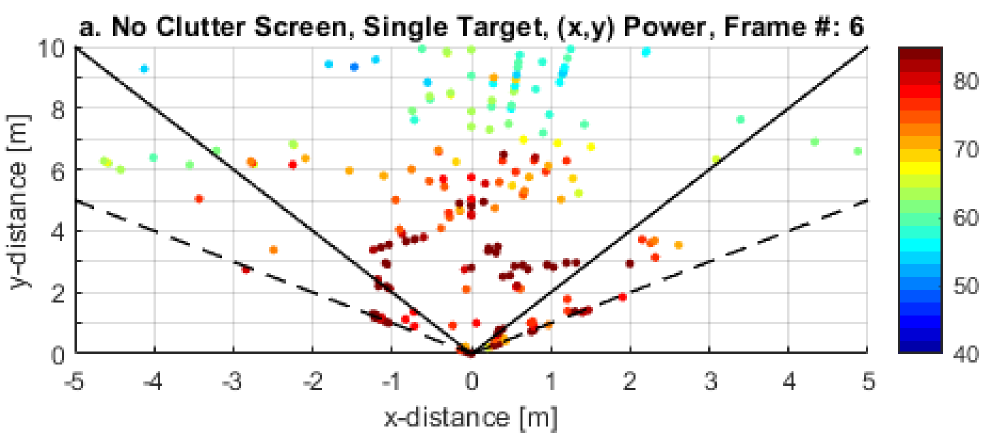

b. No Clutter Screen, 3-Targets, (x,y) Power, Frame #: 3
y-distance [m]
x-distance [m]
10
8
6
4
2
0
-5
-4
-3
-2
-1
0
1
2
3
4
5
80
70
60
50
40

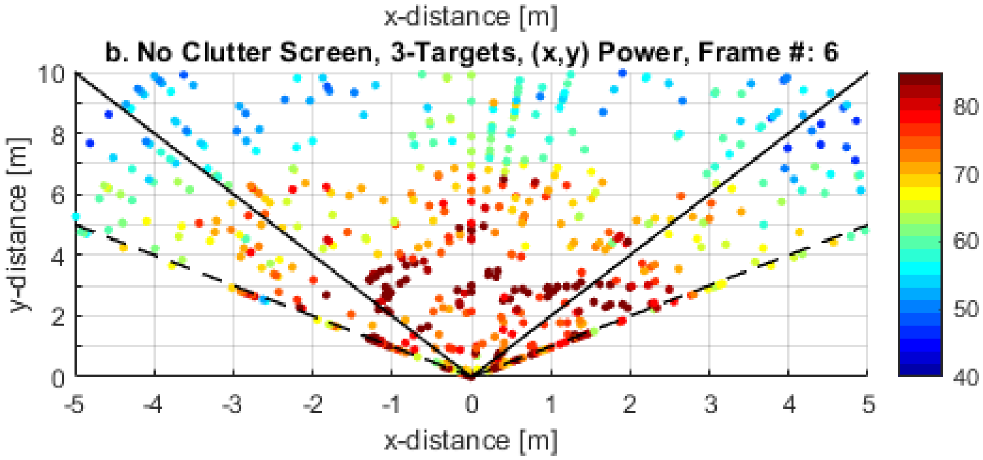

a. No Clutter Screen, Single Target, (x,y) Power, Frame #: 4
y-distance [m]
x-distance [m]
10
8
6
4
2
0
-5
-4
-3
-2
-1
0
1
2
3
4
5
80
70
60
50
40

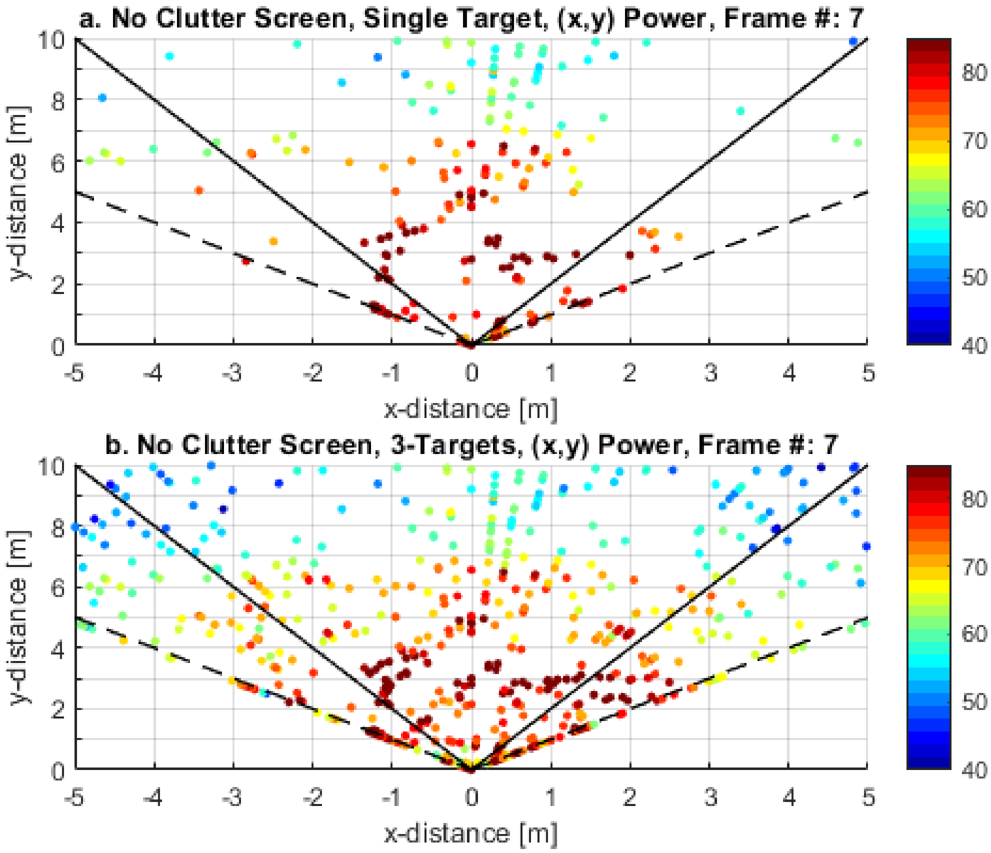

b. No Clutter Screen, 3-Targets, (x,y) Power, Frame #: 4
y-distance [m]
x-distance [m]
10
8
6
4
2
0
-5
-4
-3
-2
-1
0
1
2
3
4
5
80
70
60
50
40

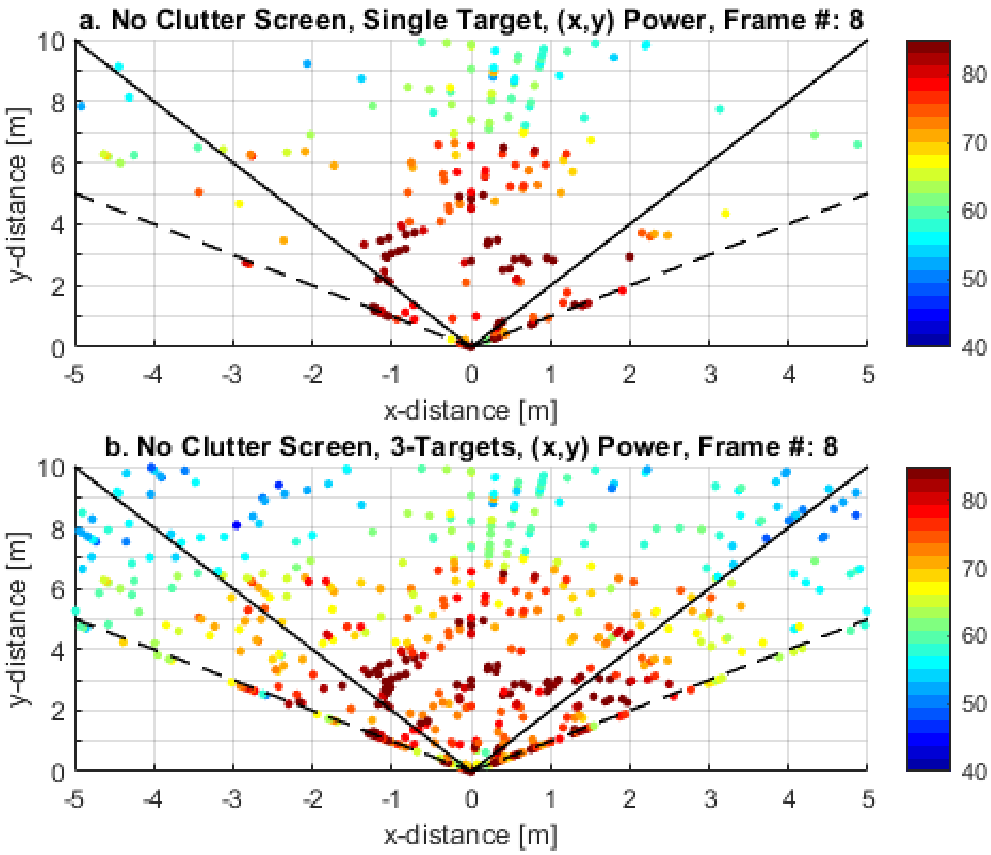

a. No Clutter Screen, Single Target, (x,y) Power, Frame #: 5
y-distance [m]
x-distance [m]
0 2 4 6 8 10
-5 -4 -3 -2 -1 0 1 2 3 4 5
40 50 60 70 80


b. No Clutter Screen, 3-Targets, (x,y) Power, Frame #: 5
y-distance [m]
x-distance [m]
0 2 4 6 8 10
-5 -4 -3 -2 -1 0 1 2 3 4 5
40 50 60 70 80


a. No Clutter Screen, Single Target, (x,y) Power, Frame #: 6
y-distance [m]
x-distance [m]
0 2 4 6 8 10
-5 -4 -3 -2 -1 0 1 2 3 4 5
40 50 60 70 80


b. No Clutter Screen, 3-Targets, (x,y) Power, Frame #: 6
y-distance [m]
x-distance [m]
0 2 4 6 8 10
-5 -4 -3 -2 -1 0 1 2 3 4 5
40 50 60 70 80






```
end % end for r loop
```