
USING MATLAB CODE FOR RADAR SIGNAL PROCESSING

EEC 134B Winter 2016
Amanda Williams
997387195
Team Hertz

CONTENTS:

- I. Introduction
- II. Note Concerning Sources
- III. Requirements for Correct Functionality
- IV. GUI Format
 - a. "Get base recording" button
 - b. "Select File" button
 - c. "Select File for 20m Mode" button
 - d. "Distance" output box
 - e. "Time to process" output box
 - f. Upper Axis
 - g. Lower Axis
- V. Function Explanation
 - a. file_to_analyze()
 - b. get_data_array()
 - i. dbv()
 - c. find_max_power()
- VI. Sample Runs and Respective Percent Error
 - a. 5.422m indoor test
 - b. 10.202m indoor test
 - c. 26.254 outdoor test (slightly overcast and windy)
 - d. 35-490m outdoor test (slightly overcast and windy)

I. INTRODUCTION

Using an audio jack and computer sound card, the filtered and amplified signal from the radar system can be sampled and saved into a .wav file using Audacity. The computer sound card can safely support up to a 1 V_{p-p} wave, so the radar system's adjustable gain stages are adjusted to ensure that the output remains below this value. This file holding the signal data can be processed using MATLAB to produce a distance value for a detected object. The following discusses the MATLAB GUI code and functions used to determine the range of the object and the accuracy of the measurement.

II. NOTE CONCERNING SOURCES

The GUI is based on a program called "read_data_RTI" written by Gregory L. Charvat for the MIT IAP Radar Course 20112.5 which can be found at:

<http://ocw.mit.edu/resources/res-ll-003-build-a-small-radar-system-capable-of-sensing-range-doppler-and-synthetic-aperture-radar-imaging-january-iap-2011/projects/>

The file function "dbv" was included within the downloaded program and remained in the final version.

All adaptations differing from this program were developed and implemented by our team.

III. REQUIREMENTS FOR CORRECT FUNCTIONALITY

For the GUI and program to function correctly and produce accurate data, the following requirements must be adhered to:

- 1) The code here is ran through MATLAB R2011a. Using other versions of the program may require some changes in syntax, particularly the manner in which the .wav file is read.
- 2) The .wav file to analyze **must** already exist. In this explanation and the example cases included, they were recorded and exported using Audacity and saved to the same file as the rest of the program parts. Recording and processing using solely MATLAB is possible, however it was not thoroughly tested by the team due to lack of hardware to support it and therefore is not explained here.
- 3) All program files (functions, GUI format, and recorded .wav files) **must** be located in the same folder. The program is **not** equipped to handle the case if this is not followed and will produce an error.
- 4) In order for any distance to be produced, a base recording of only the environment must be present in the same folder as the rest of the file and should be of approximately the same length as the samples including the target. This file must also be loaded into the program and analyzed before any other analysis may occur and the completion of such is signaled within the GUI. This is to serve as environmental effects cancellation and is one of the few key parts of the analysis. The program will not function correctly without it.

IV. GUI FORMAT

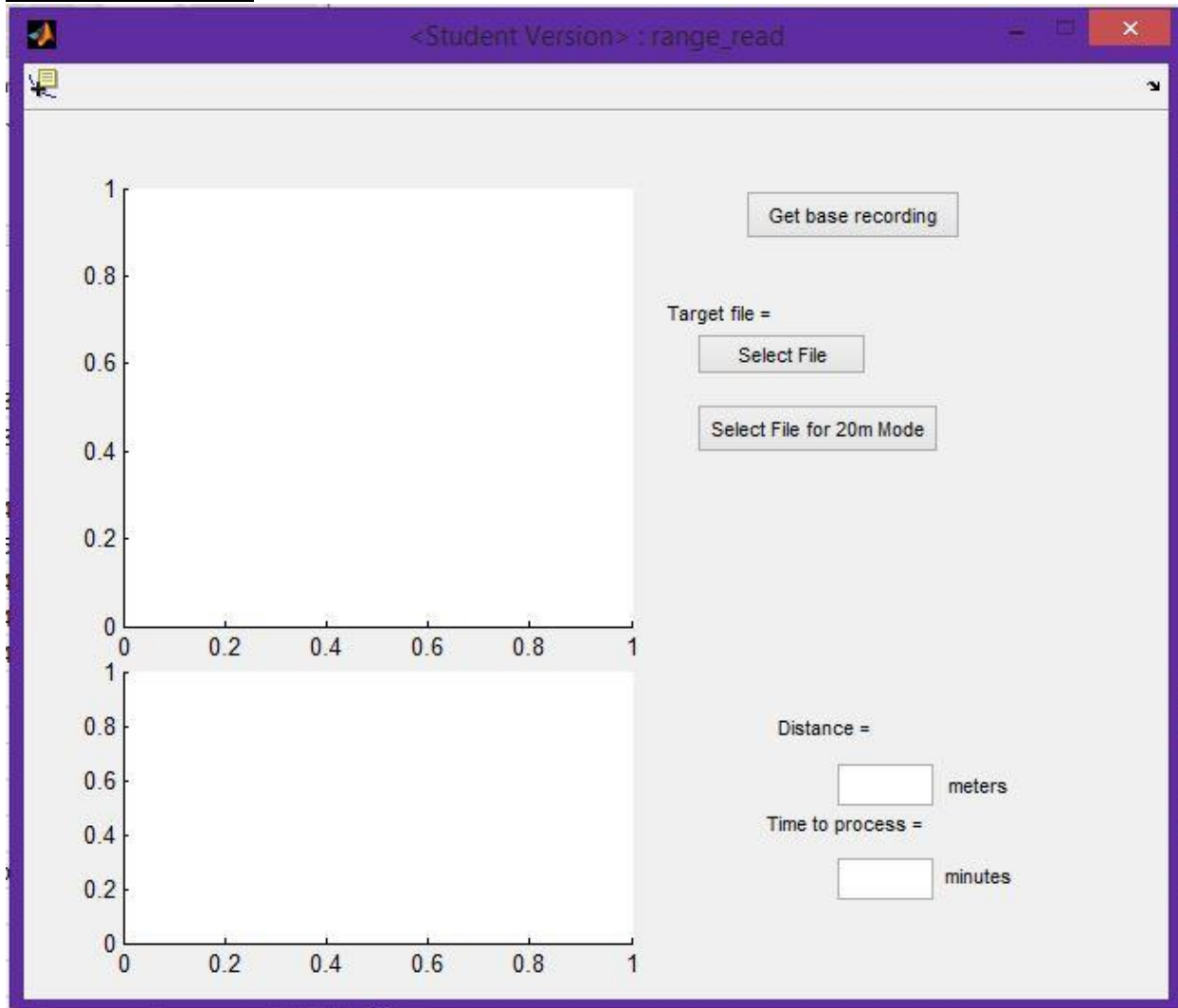


Figure 1: GUI format for range detection upon initialization

The GUI is designed to make the signal processing easier and more efficient. It requires only button presses and the selection of the desired file. The rest is automated and fixed within the program itself.

All aspects of the GUI are recorded and explained below. A picture of each function for the button is included, followed by a brief explanation. Any functions within these will be explained in the next section (Section V) of the report.

a. “Get base recording” button

```
% --- Executes on button press in get_base.
function get_base_Callback(hObject, eventdata, handles)
% hObject    handle to get_base (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%basically, basis will be an array of data holding the data
%concerning the area to help clean up other data
%start by loading in base and analyzing it first

global basis;
warning('off', 'MATLAB:colon:nonIntegerIndex');

[Y, FS, NBITS] = file_to_analyze();

%constants
c = 3E8; % (m/s) speed of light
%radar parameters
Tp = 22.5E-3; % (s) pulse time
fstart = 2257.1E6; % (Hz) LFM start frequency for example
fstop = 2679.3E6; % (Hz) LFM stop frequency for example

[time R basis] = get_data_array(Y, FS, Tp, fstart, fstop, c);

p = max(max(basis));
basis = basis - p;
basis = mean(basis);

set(handles.done_box, 'String', 'done');
end
```

Figure 2 Function run when "Get Base Recording" button is pressed

This function produces the “basis” file used for canceling out any noise in the received signal by recording and storing the data from the environment first. It retrieves the environment data (Y), sampling frequency (FS), and number of bits (NBITS) from the wav recording first, using `file_to_analyze()` to convert them into arrays that MATLAB can use. Then, it declares the radar constants that will be used in the program, using them as inputs to `get_data_array()` that will produce the actual arrays of time (time), overall distance (R), and corresponding radar output voltage data for each distance (basis) for the given .wav file. The start and stop frequencies are based on the VCO frequencies corresponding to 0.5V and 4.5V respectively. The array of voltage ratios (in dBv) are normalized to the maximum overall value before being averaged over time so that only the range vs. voltage measurements remain. This array is made to be a global value so that it may be used for other data analysis (aka to find the target). When it is complete, the function will produce a “done” signal next to the button to signify that analysis of recordings of targets may be completed.

b. "Select File" button

```
% --- Executes on button press in select_file_button.
function select_file_button_Callback(hObject, eventdata, handles)
% hObject    handle to select_file_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
warning('off', 'MATLAB:colon:nonIntegerIndex');
global basis;

%start timer and signify start
tic;

[Y, FS, NBITS] = file_to_analyze();

%constants
c = 3E8; % (m/s) speed of light
%radar parameters
Tp = 22.5E-3; % (s) pulse time => might be where the issue is?
fstart = 2257.1E6; % (Hz) LFM start frequency for example
fstop = 2679.3E6; % (Hz) LFM stop frequency for example

[time R S] = get_data_array(Y, FS, Tp, fstart, fstop, c);

m = max(max(S));

%plotting stuff

%imagesc(x_corner = C(1,1), y_corner = C(m,n), bar_low, bar_high)
axes(handles.axes1);
imagesc(R,time,S-m, [-50, 0]);

colorbar;
ylabel('time (s)');
xlabel('range (m)');
title('Image of Output');

[R2 A2 distance] = find_max_power(S, R, 0, basis);
axes(handles.axes2);
plot(R2,A2);

set(handles.dist_out, 'String', distance);
%record time in minutes
timer = toc/60;
set(handles.time_out, 'String', timer);
end
```

Figure 3: Function run when "Select File" button is pressed

This function produces the data to graph for both axes and the final output distance value. Similar to what is completed for the basis file, it retrieves the environment data (Y), sampling frequency (FS), and number of bits (NBITS) from the wav recording for the target first, using `file_to_analyze()` to convert them into arrays that MATLAB can use. Then, it declares the radar constants that will be used in the program, using them as inputs to `get_data_array()` that will produce the actual arrays of time (time), overall distance (R), and corresponding voltage ratio values for each distance (S) for the given .wav file. The frequencies are based on the VCO frequencies corresponding to 0.5V and 4.5V respectively. This data is then graphed using `imagesc` (using the voltage data normalized by the maximum) to get a visual representation of overall range vs. recording time vs. voltage data ratio (dBv). The array of voltage ratios are then sent to `find_max_power()` alongside the distance array, the basis array, and a signifier for the analysis mode (0) to produce a graph to show only range versus voltage ratios and to output the calculated distance value for the data. This value and the time it took (in minutes) to produce this value (not including the formation of the basis array) are output in their respective output boxes.

c. **“Select File for 20m Mode” button**

The only change from the above mode is the following line:

```
[R2 A2 distance] = find_max_power(S, R, 1, basis);
```

Figure 4: Changes from "Select File" to complete "Select File for 20m Mode"

The “20m Mode” runs almost identically to the default mode, however in order to produce the actual distance more accurately, it requires a “1” to be entered into the analysis mode to signify a shorter analysis range. This cuts down on analysis time and, generally, produces a more accurate result in the presence of a noisier signal.

d. **“Distance” output box**

```
set(handles.dist_out, 'String', distance);
```

Figure 5: Code to Display Distance Output in GUI

This box shows the distance that is output by either the 20m mode or the standard mode depending on the button hit to signal the analysis.

e. “Time to process” output box

```
%start timer and signify start
tic;

%record time in minutes
timer = toc/60;
set(handles.time_out, 'String', timer);
```

Figure 6: Code to Display the "Time to Process" output

This box shows the time required to output the distance mentioned above. It only signifies the time used to analyze the actual data, not including the bias analysis time. It is measured in respect to minutes.

f. Upper Axis

```
%imagesc(x_corner = C(1,1),y_corner = C(m,n), bar_low, bar_high)
axes(handles.axes1);
imagesc(R,time,S-m, [-50, 0]);

colorbar;
ylabel('time (s)');
xlabel('range (m)');
title('Image of Output');
```

Figure 7: Code to Produce the Graph in the Upper Axis

This part of the code plots the scaled voltage level for each time and range versus the time and ranges to produce a visual version of what the radar is detecting. This allows an initial view of what the data that is being analyzed looks like, without the basis or averaging being applied to the data. The final entry in the imagesc command signifies the scale that the data will be graphed against, selected to be from -50dBv to 0dBv to produce more distinct location lines that were producible in a wider range, thereby giving the user a clearer image to analyze. This does not affect the program outputs.

g. Lower Axis

```
[R2 A2 distance] = find_max_power(S, R, 0, basis);  
axes(handles.axes2);  
plot(R2,A2);
```

or

```
[R2 A2 distance] = find_max_power(S, R, 1, basis);  
  
axes(handles.axes2);  
plot(R2,A2);
```

Figure 8: Two Versions of Code to Produce the Graph in the Lower Axis

This part of the code plots the average voltage level (in inverse magnitude) over time against distance. This allows for a simplified version of the data above, giving a line in only two dimensions that can be examined to see relative levels instead of a graph in three dimensions, making it easier for the user to physically see and pick a maximum value to confirm the program output. In this way, it also allows the observation of the system as only one distinctive peak should show and an increased number of peaks signify noise or errors in the system or data collection.

V. Function Explanation

a. file_to_analyze()

```
function [sampled_data, sample_rate, num_bits] = file_to_analyze()  
%To allow the selection of the wav file from the file explorer menu  
%and break it down into analyzable data  
% Allows the selection of a wav file from the file explorer and breaks  
% the file down into sampled_data, sample_rate, num_bits  
% sampled_data = the sampled data from the given wav file  
% sample_rate = the sample rate in Hertz  
% num_bits = number of bits per sample used to encode the data  
  
filename = uigetfile('../*.*wav');  
[sampled_data, sample_rate, num_bits] = wavread(filename);  
  
end
```

Figure 9: Code for file_to_analyze

This function is called whenever the user needs to open a file for base or the normal or 20m modes of analyzation. On the button click, it opens up the window explorer so that the user can select the desired .wav file and reads the sample data, sample rate, and number of bits for that .wav file, returning those arrays of data to the calling function.

b. get_data_array()

```
function [ time range_array data_array ] = get_data_array(sampled_data, ...
    sampling_frequency, pulse_time, start_frequency, stop_frequency, c)
%compiles the data from the wav file to allow analysis of data
% Outputs a data array of power magnitude levels corresponding to the
% input system and wav data.

N = pulse_time*sampling_frequency; %number of samples per pulse
BW = stop_frequency-start_frequency; %(Hz) transmit bandwidth
f = linspace(start_frequency, stop_frequency, N/2); %instantaneous transmit frequency

%range resolution
rr = c/(2*BW);
max_range = rr*N/2;

%the input appears to be inverted
trig = -1*sampled_data(:,1);
s = -1*sampled_data(:,2);
clear sampled_data;

%decodes data

%parse the data here by triggering off rising edge of sync pulse
count = 0;
thresh = 0;
start = (trig > thresh);
for ii = 100:(size(start,1)-N)
    if start(ii) == 1 & mean(start(ii-11:ii-1)) == 0
        %start2(ii) = 1;
        count = count + 1;
        sif(count,:) = s(ii:ii+N-1);
        time(count) = ii*1/sampling_frequency;
    end
end

%subtract the average
ave = mean(sif,1);
for ii = 1:size(sif,1);
    sif(ii,:) = sif(ii,:) - ave;
end

zpad = 8*N/2;

sif2 = sif(2:size(sif,1),:)-sif(1:size(sif,1)-1,:);

v = ifft(sif2,zpad,2);

S=v;
range_array = linspace(0,max_range,zpad);

data_array = dbv(S(:,1:size(v,2)/2));
end
```

Figure 10: Code for get_data_array

This part of the code begins by defining the range resolution (rr) and the presumed max range of the radar (max_range) according to the calculated resolution and bandwidth (BW). After these are defined, it splits the signal into two, one for the SYNC data (trig) and one for the measured data (s). Then, the program parses the measured data according to the rising edge of the sync pulse, thereby creating the measured data array (sif) and time array (time). It does this by checking for changing SYNC pulses, looking only for the rising edge (trig>thresh) and confirming the edge (if start(ii) == 1 & mean(start(ii-11:ii-1)) == 0) before parsing the data and including it into its appropriate position in the sif matrix or time matrix.

Once this is collected, the average of the sif matrix is subtracted from all entries in order to get rid of the average DC term. Then, the data is converted to time domain using `ifft()`, or an inverse fast Fourier transform algorithm. It then converts the values to dBv using `dbv()`, creating the final, voltage ratio output for `data_array`. It also creates a range array using the number of samples per pulse as a guideline for distance spacing over the calculated max distance, outputting `range_array`.

i. `dbv()`

```
function out = dbv(in)
    out = 20 * log10(abs(in));
```

Figure 11: Code for dbv

This function takes the input argument (in this case the array of voltage values that are the parsed data from the signal) and converts it to decibel scale referenced to a volt.

c. find_max_power()

```
function [ scaled_distance scaled_magnitude distance ] = ...
    find_max_power( data_array, distance_values, analysis_mode, basis)
%finds the distance associated with the maximum received power level
%within the given data array
% data_array = the array of power data values
% distance values = the range of values calculated for the data set
% analysis_mode = 0 for no restrictions
% = 1 for limited to 20m distance

m = max(max(data_array));
data_array = data_array - m; %to normalize values to maximum, makes positive

i = 1;

%resize/rescale distance array to match size of A
while (i < 1984)
    scaled_distance(1,i) = (distance_values(1,2*i-1) + distance_values(1,2*i))/2;
    i = i+1;
end
scaled_distance(1,i) = (distance_values(1,1984)); %include last one

%average each row of A to average the power across time
A2 = mean(data_array);
%removing average of basis over time to make more appropriate values
%would make all A2 values positive
scaled_magnitude = A2 - basis;

%take averages of groups of u indices to find range with maximum
%averaging the magnitude over select ranges to find highest density
%of highest power signal

n = 1;
u = 19; %max range, options are 10 and 22 for performance
t = u - 1; %range - 1 to not exceed array limits

while (n+t < 993)
    sum = 0;
    q = 0;
    while (q < u)
        %sums up the magnitudes over the range
        sum = sum + scaled_magnitude(1,n+q);
        q = q + 1;
    end
    %takes average of the range and records in matrix
    avg_mag(1,n) = sum/u;
    %records middle of range to be used to find location
    mid_loc(1,n) = n + idivide(t, int32(2), 'round');
    n = n + 1;
end

%find index of the maximum average
%index 250 is approximately 20m out, setting if analysis_mode = 1

if analysis_mode == 1
    [num idx] = max(avg_mag(1,1:250));
else
    [num idx] = max(avg_mag);
end
%find the index of the middle location
R2_loc = mid_loc(idx);
%output the actual distance for the middle location
distance = scaled_distance(R2_loc);
end
```

Figure 12: Code for find_max_power

The purpose of this function is to take the data array, basis file, overall distance values and the analysis mode (either normal or 20m) and produce an array of scaled distances that will match the size of the data array, as well as a final, single distance value corresponding to the highest data value. This allows the two data sets to be graphed linearly against one another on the second axis to allow an easier point-and-click distance that may be used if the calculated distance appears incorrect.

To do so, first the array of measured data is normalized by the maximum available value. Then, because the distance array is twice the size of the data array, the distance values are rescaled and fit into an array of the same size of the distance array, the error of such is negligible. In case the array of distance value is of unusual size and to prevent issues with graphing, the last element is doubled.

The array of measured data is averaged across time to leave only one column of values of the averaged input data that can be plotted versus distance. Then, the basis results that were initially produced can be removed from the data, thereby leaving the actual received signal of the system.

This array of data is then examined in overlapping increments of approximately 3m which is averaged and stored into a secondary array, the middle locations of each section recorded as the location of the average value (this leaves an up to a 1.5m error). This is repeated throughout the data.

Once this is complete, the maximum of the data and its location is searched for in the averaged data, the overall range of such dependent on the analysis mode (either only about 20m out is examined, or the full 50m is examined). The location of this maximum value is then used to find the index of the middle location which can finally be used to find the actual scaled distance of that maximum magnitude since these two arrays are equivalent and output this distance along with the scaled distance and data arrays. This retains the maximum 1.5m error from the maximum search.

VI. SAMPLE RUNS AND RESPECTIVE PERCENT ERROR

The following GUI snapshots are recorded from two test runs the team performed to examine the quality of the code and the PCB set-up we possessed. The PCBs were two separate boards, one for the RF side of the system and the other for the baseband filtering and amplification of the system, connected together with soldered on wires. The output was hooked up to an audio cable plugged into the computer.

The first two tests were taken indoors at Kemper Hall, near the large front window. The rain had just stopped and the sky was clearing up, so some distortion may be attributed to that. The last two tests were taken outdoors in a small field. The dorm buildings, students, and bike racks were nearby, and the sky was partly cloudy and it was very windy, possibly causing some distortion in results. The can lids were on in all tests due to better observed performance.

All of the following outdoor tests were conducted using an approximately 4 second recording time. Tests were conducted using an 8 second recording time, however test results were not significantly different.

All of the following indoor tests were conducted using an approximately 8 second recording time. Tests were conducted using a 16 second recording time, however test results were not significantly different. The following summarizes the results of the program and its corresponding “point and click” method.

a. 5.422m indoor test

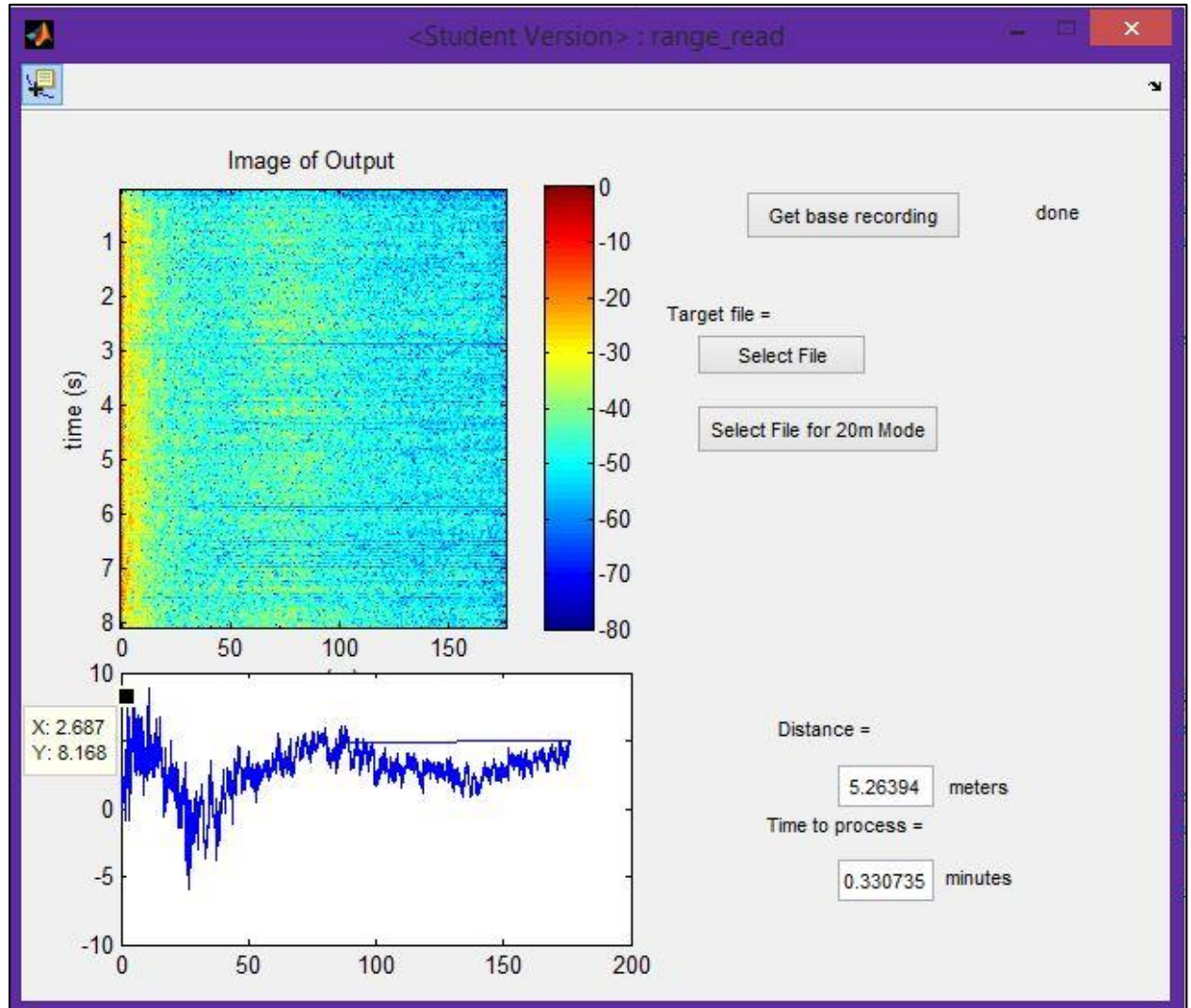


Figure 13: Capture of Output for 5.422m test

This test was completed indoors on the upper floor walkway along the front window of Kemper Hall. It was raining when the initial bias recording was completed and the rain stopped sometime during testing, so the change in environmental light may be the cause of some error or unusual peaks in the secondary graph. It was completed using the 20m mode.

Actual distance = 5.422m

Program output distance = 5.264m => 2.91% error

Point and click distance = 2.687m => 50.44% error

b. 10.202m indoor test

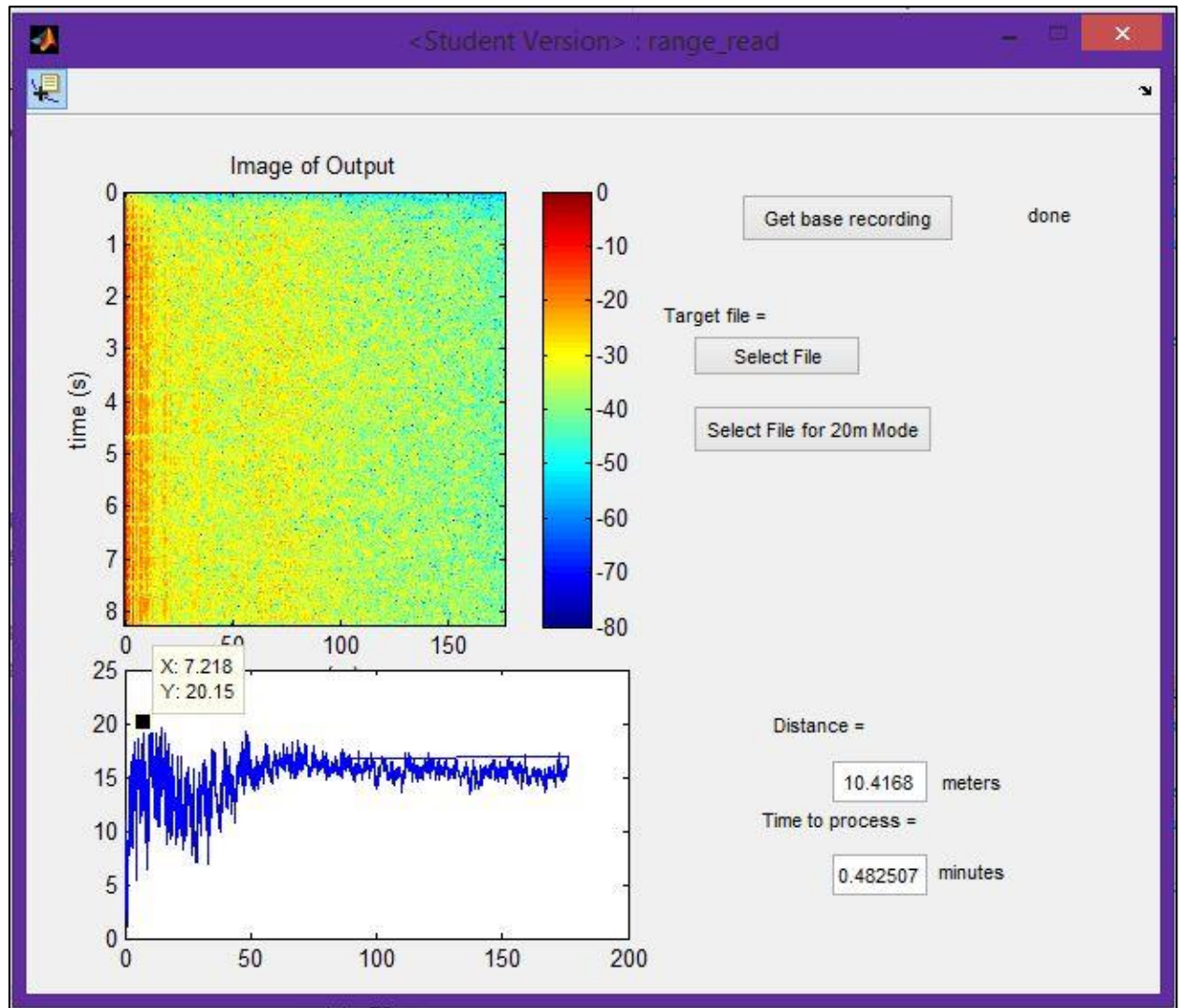


Figure 14: Capture of Output for 10.202m test

This test was completed indoors on the upper floor walkway along the front window of Kemper Hall. It was raining when the initial bias recording was completed and the rain stopped sometime during testing, so the change in environmental light may be the cause of some error or unusual peaks in the secondary graph. It was completed using the 20m mode.

Actual distance = 10.202m

Program output distance = 10.4168m => 2.11% error

Point and click distance = 7.218m => 29.25% error

c. 26.254m outdoor test

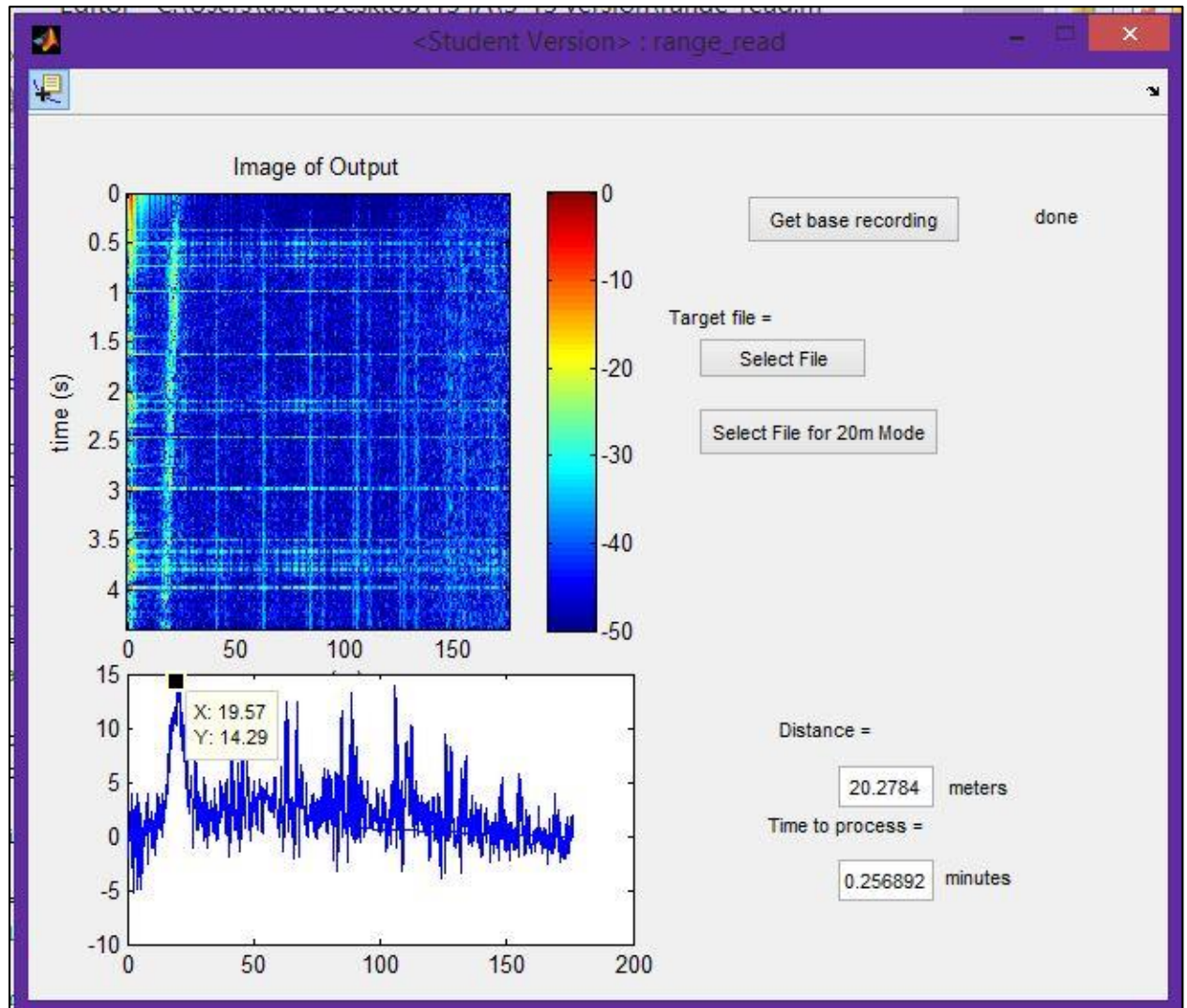


Figure 15: Figure of Output for 26.254m test

This test was completed outdoors on a small grassy area near the dorms. Some bike racks and other obscurities were present which do not appear in the initial image. It was partially cloudy and quite windy when the initial bias recording was completed, so the change in environment may be the cause of some error or unusual peaks in the secondary graph. It was completed using the standard mode.

Actual distance = 26.254m

Program output distance = 20.2784m => 22.76% error

Point and click distance = 19.57m => 25.46% error

d. 35.490m outdoor test

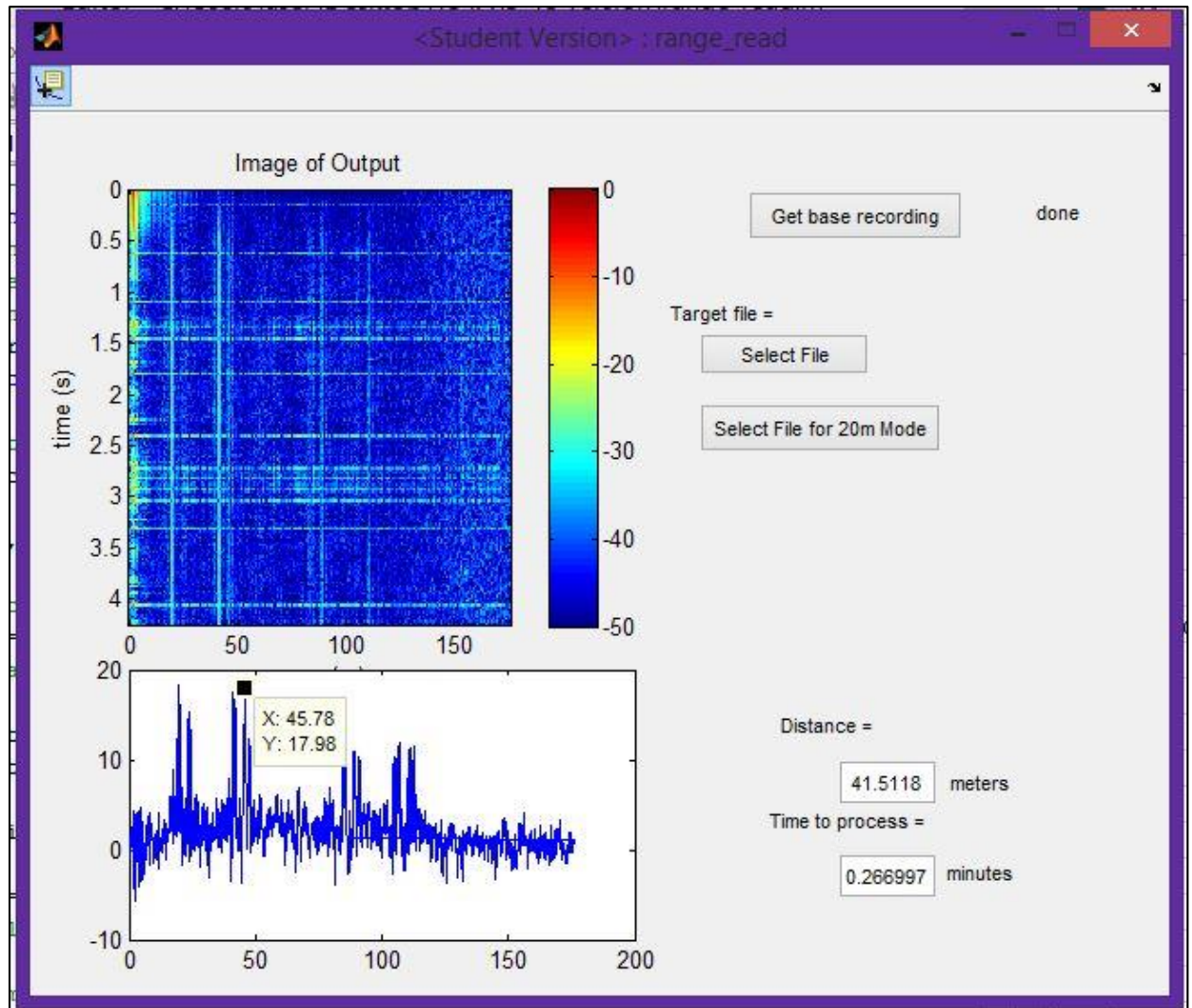


Figure 16: Figure of Output for 35.490m test

This test was completed outdoors on a small grassy area near the dorms. Some bike racks and other obscurities were present which do not appear in the initial image. It was partially cloudy and quite windy when the initial bias recording was completed, so the change in environment may be the cause of some error or unusual peaks in the secondary graph. It was completed using the standard mode.

Actual distance = 35.490m

Program output distance = 41.5118m => 16.97% error

Point and click distance = 45.78m => 28.99% error