
ACCELEROMETER DATA PROCESSING TOOL

PROJET P08A08: ACCELEROMETRES & FERROVIAIRE

07 / 01 / 2009
JIHANE NASSEH

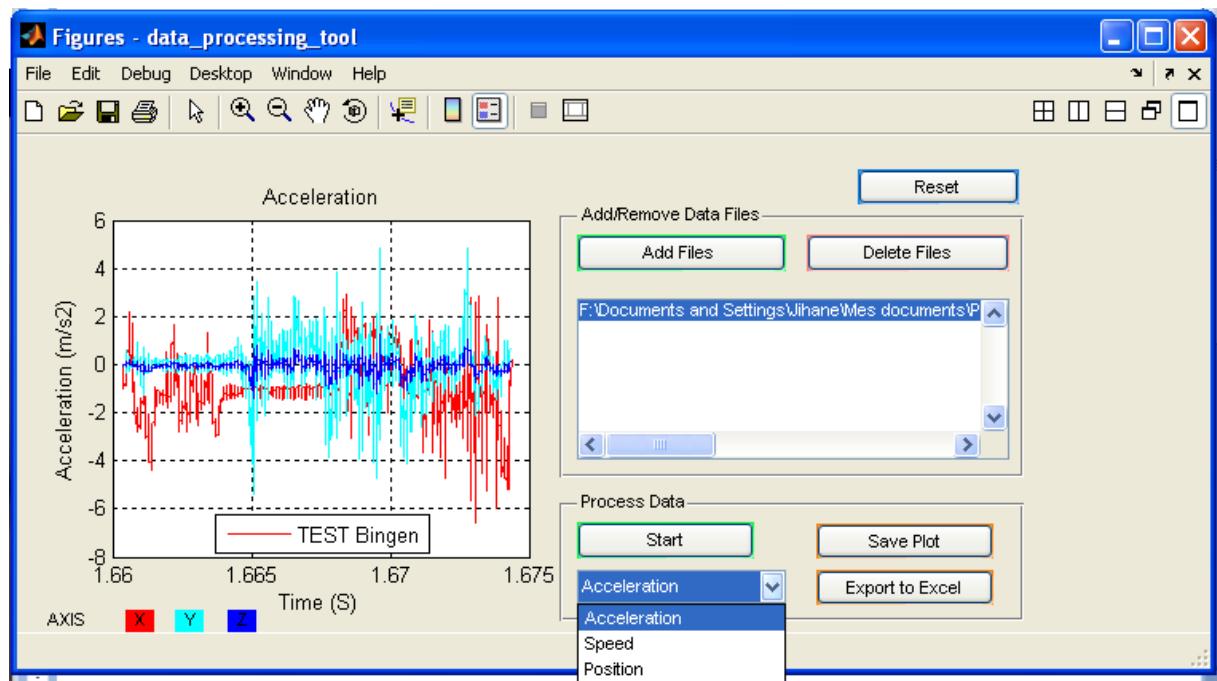
Table of Content

<i>Introduction</i>	4
<i>1. Visual Portion of the GUI</i>	5
<i>2. Add Files and Delete Files Callback Functions</i>	6
<i>3. Processing the Data</i>	8
3.1 Start pushbutton Callback	8
3.2 Disable Buttons and enable Buttons functions	9
3.3 Convert acceleration to g function	10
3.4 Calibration function.....	12
3.5 Mean function.....	13
3.6 Plot Data function.....	13
<i>4. Pop-up Menu and Save Plot Callbacks</i>	15
<i>5. Exporting the Data to Excel</i>	17
<i>6. Opening Function and Close GUI Confirmation</i>	19
<i>7. Reset Button</i>	20

Introduction



This tutorial will provide you with information about our data processing GUI and all the functions that were implemented in it. After reading it, you will understand how we implemented all the functions and be able to make an eventual modification or development. Following, you will find the visual aspect of the GUI, and the code to each component callback. This tutorial draws upon many of the basic GUI elements: adding files to a listbox, parsing data, plotting data onto the GUI, saving GUI plots, disabling/enabling buttons, exporting data to Excel format, and many other things.



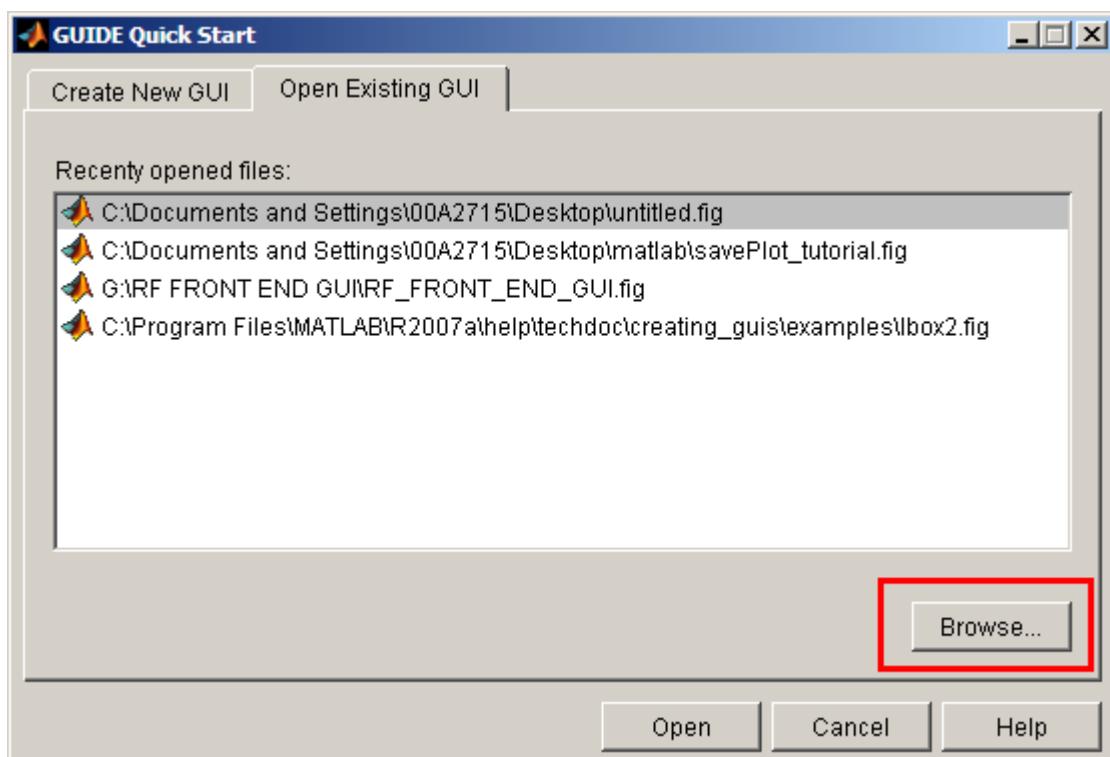
This tutorial is written for those with a good amount of experience creating a Matlab GUI. If you are new to creating GUIs in Matlab, you should visit this website: www.blinkdagger.com, which provides a clear introduction to Matlab and GUIs and contains further explanations of the basic GUI elements. Basic/Advanced knowledge of Matlab is highly recommended. Matlab version R2006a is used in writing this tutorial. Both earlier versions and new versions should be compatible as well (as long as it isn't too outdated).

1. Visual Portion of the GUI

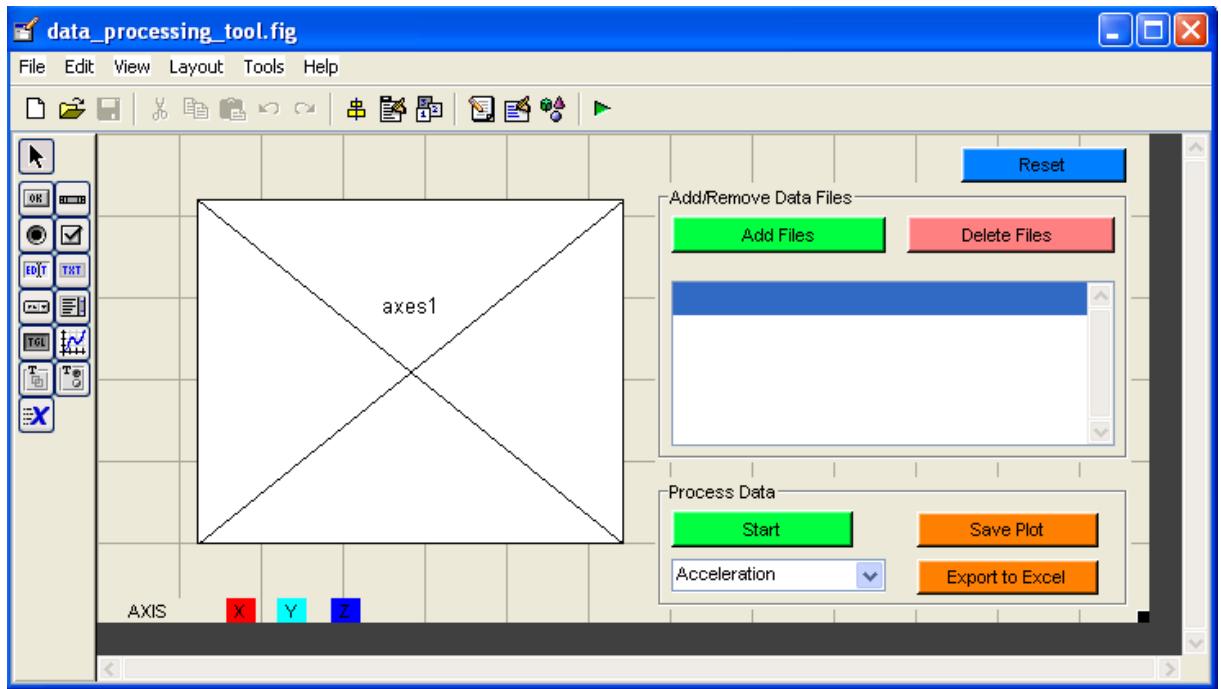
1. First, download the GUI from the USB allocated to our project. Unzip the files and place them wherever you please.
2. Now, type `guide` at the command prompt.



3. Choose to open the GUI by clicking on "Open Existing GUI". Click on "Browse" to locate where you saved the GUI files.



4. Here is what the GUI should look like when you open it:



5. Click on the icon on the GUI figure to bring up the accompanying .m file.

2. Add Files and Delete Files Callback Functions

The first two callbacks that you are going to discover are *addFiles_pushbutton_Callback* and *deleteFiles_pushbutton_Callback*.

1. Under the *addFiles_pushbutton_Callback*, you will find the following code:

```

1. function addFiles_pushbutton_Callback(hObject, eventdata, handles)
2.
3. %gets input file(s) from user. the sample data files have
4. %extension .txt
5. [input_file,pathname] = uigetfile( ...
6.     {'*.txt', 'Data Files (*.txt)'; ...
7.     '.*', 'All Files (*.*)'}, ...
8.     'Select files', ...
9.     'MultiSelect', 'on');
10.
11. %if file selection is cancelled, pathname should be zero
12. %and nothing should happen
13. if pathname == 0
14.     return
15. end
16.
17. %gets the current data file names inside the listbox
18. inputFileName = get(handles.inputFiles_listbox, 'String');
19.
20. %if they only select one file, then the data will not be a cell
21. if iscell(input_file) == 0
22.     %add the most recent data file selected to the cell
        containing

```

```

23.      %all the data file names
24.      inputFileNames{length(inputFileNames)+1} =
25.          fullfile(pathname,input_file);
26.      %else, data will be in cell format
27.      else
28.          %stores full file path into inputFileNames
29.          for n = 1:length(input_file)
30.              inputFileNames{length(inputFileNames)+1} =
31.                  fullfile(pathname,input_file{n});
32.          end
33.
34.      %updates the gui to display all filenames in the listbox
35.      set(handles.inputFiles_listbox,'String',inputFileNames);
36.
37.      %make sure first file is always selected so it doesn't go out
38.      %of range
39.      %the GUI will break if this value is out of range
40.      set(handles.inputFiles_listbox,'Value',1);
41.
42.      % Update handles structure
43.      guidata(hObject, handles);

```

2. But what happens if you accidentally added too many files? This is what the delete button is for! Under the *deleteFiles_Callback*, you will find the following code:

```

43.      %get the current list of file names from the listbox
44.      inputFileNames = get(handles.inputFiles_listbox,'String');
45.
46.      %get the values for the selected file names
47.      option = get(handles.inputFiles_listbox,'Value');
48.
49.      %is there is nothing to delete, nothing happens
50.      if isempty(option) == 1 || option(1) == 0
51.          return
52.      end
53.
54.      %erases the contents of highlighted item in data array
55.      inputFileNames(option) = [];
56.
57.      %updates the gui, erasing the selected item from the listbox
58.      set(handles.inputFiles_listbox,'String',inputFileNames);
59.
60.      %moves the highlighted item to an appropriate value or else will get error
61.      if option(end) > length(inputFileNames)
62.          set(handles.inputFiles_listbox,'Value',length(inputFileNames));
63.      end
64.
65.      % Update handles structure
66.      guidata(hObject, handles);

```

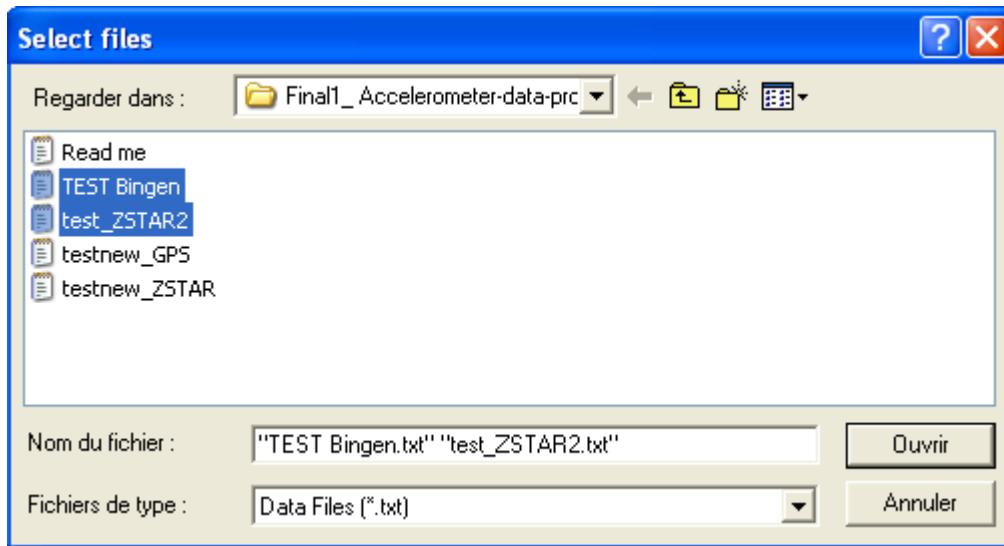
To allow multiple files to be selected within the listbox, the following code was added to the opening function. Alternatively, you can change these properties through the Property Inspector using GUIDE.

```

set(handles.inputFiles_listbox,'Max',2);
set(handles.inputFiles_listbox,'Min',0);

```

To make sure that it work in the GUI. You can try adding/removing files to the GUI. The menu below pops up when you click on the "Add Files" button.



3. Processing the Data

The next callback we will work with is the *start_pushbutton_Callback*. This is probably the most complicated callback of this program because of all the subfunctions within it. Sub functions can be placed within the GUI m-file itself, or they can be placed in their own separate m-file as long as the m-files containing the sub functions are in the same directory as the GUI m-files. Once we have added the files that we want to process, we want to process the data. First, the callback will have to parse the input data files, convert it to g ($g=9.81 \text{ m/s}^2$), make the calibration and finally plot the results converted in m/s² onto the axes. How is this accomplished?

3.1 Start pushbutton Callback:

In *start_pushbutton_Callback*, you will find:

```

1. %get the list of input file names from the listbox
2. inputFileNames = get(handles.inputFiles_listbox, 'String');

3. %checks to see if the user selected any input files
4. %if not, nothing happens
5. if isempty(inputFileNames)
6. return
7. end

8. %disables the button while data is processing
9. disableButtons(handles);
10. %refresh the figure to reflect changes
11. refresh(data_processing_tool);

12.%initialize the cell arrays
13.%if you don't know what cell arrays are, it might be a good
   idea to
14.%review this by using the Matlab Help Files
15. handles.data = {};
16. handles.legendData = {};

```

```

17. for x = 1 : length(inputFileNames)
18. %gets the filename without the extension
19. [ignore,fileNmae,ext,ignore]=fileparts(inputFileNames{x});
20. %store filenames so that it will display on the legend
21. handles.legendData(x) = {fileName};

22. %store acceleration data converted in g in the variable anew
23. a=convert_accele_to_g(inputFileNames{x});
24. end

25. % Distinction of motion activity from the gravity in the
   Terrestrial axis
26. [d,p,h]=calibration(a);

27. t=a(:,1);      % extraction of the time

28. acceleration= h* 9.81; % conversion of acceleration from 'g' to
   'm/s2'

29. handles.data{x}=[t, acceleration]; % storing all the data in
   one variable (time ax ay az)

30. handles.legendObject
   plotData(handles.data,handles.legendData,handles.axes1,get(handles.plot_popupmenu,'Value'));

31. %the data must be done processing before other Callbacks will
   be
32. %able to function properly. this variable will be used as a
   "check"
33. %to see whether the data has been processed or not
34. handles.processDataCompleted = 1;

35. %data is done processing, so re-enable the buttons
36. enableButtons(handles);
37. guidata(hObject, handles);

```

3.2 Disable Buttons and enable Buttons functions:

The first thing you might notice is the *disableButtons* and *enableButtons* functions. Basically, these functions are included so that while Matlab is busy processing the data, the user cannot click on any of the other buttons. The code for the two functions is:

```

1.  function disableButtons(handles)
2.  set(handles.figure1,'Pointer','watch');
3.  set(handles.start_pushbutton,'Enable','off');
4.  set(handles.reset_pushbutton,'Enable','off');
5.  set(handles.addFiles_pushButton,'Enable','off');
6.  set(handles.savePlot_pushButton,'Enable','off');
7.  set(handles.deleteFiles_pushButton,'Enable','off');
8.  set(handles.inputFiles_listbox,'Enable','off');
9.  set(handles.plot_popupmenu,'Enable','off');
10. set(handles.export_pushButton,'Enable','off');
11. function enableButtons(handles)
12. set(handles.figure1,'Pointer','arrow');
13. set(handles.start_pushbutton,'Enable','on');
14. set(handles.reset_pushbutton,'Enable','on');
15. set(handles.addFiles_pushButton,'Enable','on');
16. set(handles.savePlot_pushButton,'Enable','on');
17. set(handles.deleteFiles_pushButton,'Enable','on');

```

```

18. set(handles.inputFiles_listbox,'Enable','on');
19. set(handles.plot_popupmenu,'Enable','on');
20. set(handles.export_pushbutton,'Enable','on');

```

3.3 Convert acceleration to g function:

The next function you might have questions about is the `convert_accele_to_g` function. This function converts the acceleration from binary to ' $g=9.8 \text{ m/s}^2$ ', please note that the conversion process and values are specific to the Freescale accelerometer used in the project, for other type of accelerometers some modifications might be necessary, for that, you should refer to their data sheets and eventually to the technical services of the supplier.

The following is the code for the function.

```

function anew=convert_accele_to_g(data)

m=0;
j=0;
sw=0;

%the input to this function is the file name of the data file.
%If the data file is not in the current Matlab directory
%you must include the entire directory path.
%opens the file
fid2 =fopen(data);
while feof(fid2) == 0

    data2=fgetl(fid2);

    remain2 = data2;

    if (size(remain2)==[1 31]) %All the valide frames have the size
[1 31]
        j=j+1;
        [Heure, remain2] = strtok(remain2,',');
        H(j)=str2double(Heure);
        [Minute, remain2] = strtok(remain2,',');
        Mn(j)=str2double(Minute);
        [Seconde, remain2] = strtok(remain2,',');
        S(j)=str2double(Seconde);
        t1(j)=(H(j)*3600)+(Mn(j)*60)+S(j);%time of traveling per
second
        t(j)=t1(j)-t1(1);%time calibration

        %Extraction of acceleration on x axis
        [trame, remain2] = strtok(remain2,'');
        m=m+1;

        if size(trame)==[1 19]
            datax1_8(m)=bin2dec(trame(4));
            datax2_8(m)=bin2dec(trame(5));
            x_dec(m)=((256*datax1_8(m))+datax2_8(m)); %Decimal
value relatively to axis x on 16 bits

            %Extraction of acceleration on y axis
            datay1_8(m)=bin2dec(trame(7));

```

```

        datay2_8(m)=bin2dec(trame(8));
        y_dec(m)=((256*datay1_8(m))+datay2_8(m)); %Decimal
value relatively to axis y on 16 bits

        %Extraction of acceleration on z axis
        dataz1_8(m)=bin2dec(trame(10));
        dataz2_8(m)=bin2dec(trame(11));
        z_dec(m)=((256*dataz1_8(m))+dataz2_8(m)); %Decimal
value relatively to axis y on 16 bits

        %Extraction of the raw bandgap
        bp1_8(m)=bin2dec(trame(18));
        bp2_8(m)=bin2dec(trame(19));
        brp_dec(m)=(256*(bp1_8(m)))+bp2_8(m); %Decimal value
of the raw band gap on 16 bits

    else
        n=n+1;%number of wrong values (just for information)

        x_dec(m)=x_dec(m-1); %%Decimal value relatively to axis y
on 16 bits

        y_dec(m)=y_dec(m-1);

        z_dec(m)=z_dec(m-1);

        brp_dec(m)=brp_dec(m-1);%Decimal value on 16 bits

    end
end
end
fclose(fid2);

for k=1:m
%Compute of battery's voltage
Vbat(k)=((65536;brp_dec(k))*1.20);
ADCres(k)=Vbat(k)/65536;
%Compute of the output voltage on each axis
Vxout(k)= x_dec(k)*ADCres(k);
Vyout(k)= y_dec(k)*ADCres(k);
Vzout(k)= z_dec(k)*ADCres(k);
%conversion to g (9.81 m/s2)
Vdd=3.2;
S=0.8;
ax1(k)=(1/S)*(Vxout(k) - (Vdd/2));
ay1(k)=(1/S)*(Vyout(k) - (Vdd/2));
az1(k)=(1/S)*(Vzout(k) - (Vdd/2));

end
tempst';
a=[tempst' ax1' ay1' az1']; %Time in secondes, ax, ay, az in g

```

3.4 Calibration function:

The following function you will discover is *calibration* function, that distinguishes motion activity from the gravity in the Terrestrial axis and returns "d"; dynamic component of the acceleration, "p": the vertical component of the dynamic acceleration vector d, and "h": the horizontal component of the dynamic acceleration, for more information about the calibration process, please read the application note related.

```
1. function [d,p,h]=calibration(data)

2. % Estimation of the gravity component on each axis by averaging
   all the
3. % readings in the chosen sampling interval (typically a few
   seconds)
4. [data_x,data_y,data_z]=mean(data,2); % average of the first 21
   values
5. g=[data_x,data_y,data_z];

6. % Vector made up of the three acceleration measurements
7. a=[data(:,2),data(:,3),data(:,4)];

8. [nl nc]=size(a);

9. % Dynamic component of a, that is caused by the motion rather
   than gravity
10. for i=2:nl
11. for j=1:nc
12. d(i,j)=a(i,j)- g(j);

13. end
14. end

15. % We compute the projection p of d upon the vertical axis z as:
16. p=((d*g')/(dot(g,g)))*g; % p is the vertical component of the
   dynamic acceleration vector d

17. h=d-p; % We compute the horizontal component of the dynamic
   acceleration by vector subtraction

18. return
```

3.5 Mean function:

In the last function, you must have noticed the *mean* function, which computes the average of the n first values of each axis; its code is written as bellow:

```
1. function [data_x,data_y,data_z]=mean(data,n)

2. data_x=0;
3. data_y=0;
4. data_z=0;

5. for i=2:n

6. data_x=data_x+ data(i,2);
```

```

7. data_y=data_y+ data(i,3);
8. data_z=data_z+ data(i,4);
9. end
10. data_x= data_x/(n-1);
11. data_y= data_y/(n-1);
12. data_z= data_z/(n-1);

13. return

```

3.6 Plot Data function:

Finally, the last custom function within this callback is the plotData function. This function returns the legend object of the plot so that it can be used later in other parts of this GUI. The function takes in 4 arguments. The data that will be plotted, the legend data, the axes that the data will be plotted on (this argument is helpful if you have more than 1 axes on your GUI), and the type of plot.

```

function [legendObject]=plotData(data,legendData,axesName,option)
cla(axesName); %clear the axes
axes(axesName); %set the axes to plot
hold on
grid on

%plot the acceleration plot
if (option==1)
    for x=1:(length(data))
        plot(data{x}(:,1),data{x}(:,2), 'r');
        hold on;
        plot(data{x}(:,1),data{x}(:,3), 'c');
        hold on;
        plot(data{x}(:,1),data{x}(:,4)), 'g';
        hold off;
    end
    %add a legend to the plot
    legendObject = legend(legendData,'Location','Best');
    title('Acceleration')
    xlabel('Time (S)')
    ylabel('Acceleration (m/s2)');
end

%plot speed plot
elseif (option==2)

    for x=1:(length(data))
        [speed]=Integration(data{x},0,0,0,0,0,0); %integration of
acceleration to get the speed

        plot(data{x}(:,1),speed(:,2), 'r');
        hold on;
        plot(data{x}(:,1),speed(:,3), 'c');
        hold on;
        plot(data{x}(:,1),speed(:,4)), 'g';
        hold off;
    end
    %add a legend to the plot
    legendObject= legend(legendData,'Location','Best');
    title('Speed')
    xlabel('Time(s)')
    ylabel('Speed (m/s)');
end

```

```

%plot position plot
else
    for x=1:(length(data))
        [speed,           position]=Integration(data{x},0,0,0,0,0,0);
        %integration of acceleration to get the position

            plot(data{x}(:,1),position(:,2),'r');
            hold on;
            plot(data{x}(:,1),position(:,3),'c');
            hold on;
            plot(data{x}(:,1),position(:,4)),'g';
            hold off;
    end
    %add a legend to the plot
    legendObject= legend(legendData,'Location','Best');
    title('Position')
    xlabel('Time(s)')
    ylabel('Position (m)');
end
hold off

%allow legend titles to be displayed properly
set(legendObject,'Interpreter','none');

```

As you could notice, to be able to display the speed and position, it was necessary to call the *Integration* function, that has the following code:

```

function[speed,position]=Integration(acceleration,x0,y0,z0,v0_X,v0_Y,
v0_Z)

[nl nc]=size(acceleration);

%Initial parameters
position(1,2)=x0;
position(1,3)=y0;
position(1,4)=z0;
speed(1,2)=v0_X;
speed(1,3)=v0_Y;
speed(1,4)=v0_Z;

%Speed vector
for i=2:nl
    for j=2:4
        t(i,1)= acceleration(i,1)- acceleration(i-1,1);
        speed(i,j)=acceleration(i,j)*t(i,1)+speed(i-1,j);
    end
end

%Displacement vector
for i=2:nl
    for j=2:4
        t(i,1)= (acceleration(i,1)- acceleration(i-1,1));
        position(i,j)=0.5*acceleration(i,j)*t(i,1)^2+speed(i-
1,j)*t(i,1)+position(i-1,j);
    end
end

```

```

end

return

```

4. Pop-up Menu and Save Plot Callbacks

1. The pop-up menu is relatively easy to use and allows the user to select between displaying the acceleration plot, speed plot or the position plot on the axes. You will find the following code on *plot_popupmenu_Callback*

```

2. function plot_popupmenu_Callback(hObject, eventdata, handles)
3.
4. %if the data hasn't been processed yet,
5. %nothing happens when this button is pressed
6. if (handles.processDataCompleted == 0)
7.     return
8. end
9.
10. %get the value of the current Pop-up menu selection
11. %plotType = 1, if acceleration option is chosen
12. %plotType = 2, if speed option is chosen
13. %plotType = 3, if position option is chosen
14. plotType = get(hObject, 'Value');
15.
16. %plots the data
17. handles.legendObject =
    plotData(handles.data, handles.legendData, handles.axes1, plotType
    );
18. guidata(hObject, handles);

```

19. The next callback is the *savePlot_pushbutton_Callback*. In this callback, we put the *handles.legendObject* to use. The *handles.legendObject*, if you remember, is an output from the *plotData* function.

```

20.     %if the data hasn't been processed yet,
21.     %nothing happens when this button is pressed
22.     if (handles.processDataCompleted == 0)
23.         return
24.     end
25.
26.     disableButtons(handles);
27.     refresh(data_processing_tool);
28.     savePlotWithinGUI(handles.axes1, handles.legendObject);
29.     enableButtons(handles);
guidata(hObject, handles);
function savePlotWithinGUI (axesObject, legendObject)
%this function takes in two arguments
%axesObject is the axes object that will be saved (required input)
%legendObject is the legend object that will be saved (optional input)

%stores savepath for the phase plot
[filename, pathname] = uiputfile({ '*.emf', 'Enhanced Meta File (*.emf)' };...

```

```

'*.bmp','Bitmap (*.bmp)'; '*.fig','Figure (*.fig)'}}, ...
'Save picture as','default');

%if user cancels save command, nothing happens
if isequal(filename,0) || isequal(pathname,0)
    return
end
%create a new figure
newFig = figure;

%get the units and position of the axes object
axes_units = get(axesObject,'Units');
axes_pos = get(axesObject,'Position');

%copies axesObject onto new figure
axesObject2 = copyobj(axesObject,newFig);

%realign the axes object on the new figure
set(axesObject2,'Units',axes_units);
set(axesObject2,'Position',[15 5 axes_pos(3) axes_pos(4)]);

%if a legendObject was passed to this function . .
if (exist('legendObject'))

    %get the units and position of the legend object
    legend_units = get(legendObject,'Units');
    legend_pos = get(legendObject,'Position');

    %copies the legend onto the the new figure
    legendObject2 = copyobj(legendObject,newFig);

    %realign the legend object on the new figure
    set(legendObject2,'Units',legend_units);
    set(legendObject2,'Position',[15-axes_pos(1)+legend_pos(1)
axes_pos(2)+legend_pos(2) legend_pos(3) legend_pos(4)] );
    5-
end

%adjusts the new figure accordingly
set(newFig,'Units',axes_units);
set(newFig,'Position',[15 5 axes_pos(3)+30 axes_pos(4)+10]);

%saves the plot
saveas(newFig,fullfile(pathname, filename))

%closes the figure
close(newFig)

```

5. Exporting the Data to Excel

Exporting data to a medium that users can manipulate is very desirable. Since many people work with Excel, adding a feature to export data to Excel can come in quite handy. Lets see how we did to export data to Excel. On `export_pushbutton_Callback` you will find the following code:

```

%if the data hasn't been processed yet,
%nothing happens when this button is pressed
if (handles.processDataCompleted == 0)
    return
end

saveDataToExcel(handles.data,handles.legendData);

saveDataToExcel code is saved as its own separate m-file

function saveDataToExcel(data, fileNames)

```

```

%stores savepath for the phase plot
[filename, pathname] = uiputfile({'*.xls','Excel (*.xls)'}, 'Save Data
to Excel File','default');

Time = [data{1}(:,1)];

for x = 1:length(fileNames)
    accelerationData = [data{x}];
    speedxyz=Integration(data{x},0,0,0,0,0,0); %integration      of
acceleration to get the position
    [speedo,           positionxyz]=Integration(data{x},0,0,0,0,0,0);
%integration of acceleration to get the position
    accel{x} = 'Acceleration_XYZ(m/s2)';
    speed{x} = 'Speed_XYZ(m/s)';
    position{x} = 'Position_XYZ(m)';
end

saveFileName = fullfile(pathname, filename);
xlswrite(saveFileName,['Data        File'      fileNames], 'Acceleration
Data','A1');
xlswrite(saveFileName,['Time(s)'   accel], 'Acceleration Data', 'A2');
xlswrite(saveFileName,accelerationData, 'Acceleration Data', 'A3');

speedtxyz=[Time speedxyz(:,2) speedxyz(:,3) speedxyz(:,4)];
xlswrite(saveFileName,['Data File' fileNames], 'Speed Data', 'A1');
xlswrite(saveFileName,['Time(s)' speed], 'Speed Data', 'A2');
xlswrite(saveFileName,speedtxyz, 'Speed Data', 'A3');

positiontxyz=[Time           positionxyz(:,2)           positionxyz(:,3)
positionxyz(:,4)];
xlswrite(saveFileName,['Data File' fileNames], 'Position Data', 'A1');
xlswrite(saveFileName,['Time(s)' position], 'Position Data', 'A2');
xlswrite(saveFileName,positiontxyz, 'Position Data', 'A3');

deleteEmptyExcelSheets(saveFileName);

```

And to erase those empty sheets when the Excel file is created, we can use the following function:

```

function deleteEmptyExcelSheets(fileName)
%this function erases any empty sheets in an excel document
%the input fileName is the entire path of the file
%for example, fileName = 'C:\Documents and Settings\matlab\myExcelFile.xls'

excelObj = actxserver('Excel.Application');
%opens up an excel object
excelWorkbook = excelObj.workbooks.Open(fileName);
worksheets = excelObj.sheets;
%total number of sheets in workbook
numSheets = worksheets.Count;

count=1;
for x=1:numSheets
    %stores the current number of sheets in the workbook
    %this number will change if sheets are deleted
    temp = worksheets.count;

    %if there's only one sheet left, we must leave it or else
    %there will be an error.
    if (temp == 1)

```

```

        break;
    end

%this command will only delete the sheet if it is empty
worksheets.Item(count).Delete;

%if a sheet was not deleted, we move on to the next one
%by incrementing the count variable
if (temp == worksheets.count)
    count = count + 1;
end
end
excelWorkbook.Save;
excelWorkbook.Close(false);
excelObj.Quit;
delete(excelObj);

```

The following figure shows how the excel document with the exported data should look like:

	A	B	C	D	E
1	Data File	TEST Bingen			
2	Time(s)	Acceleration_XYZ(m/s2)			
3	1,6604	0	0	0	
4	1,6604	-0,96686272	-0,09471985	-0,07987558	
5	1,6605	-0,72193858	0,09171509	-0,01941926	
6	1,6605	-0,87917912	-0,11364887	-0,07944583	
7	1,6605	-0,69262349	-0,11746302	-0,06950372	
8	1,6605	-0,55457905	-0,27518112	-0,10050106	
9	1,6605	-0,50853127	-0,30065948	-0,10412269	
10	1,6605	-0,3539042	0,00109654	-0,02038074	
11	1,6605	0,01672872	0,10926284	0,02803108	
12	1,6605	0,16587114	0,35598774	0,09782657	
13	1,6605	0,07236577	0,500526	0,12815952	
14	1,6605	-0,02893642	0,5649386	0,13819736	
15	1,6605	0,26440409	0,4840588	0,13528852	
16	1,6605	0,48842937	0,19988128	0,07799567	
17	1,6605	0,28730727	-0,02537174	0,01048359	
18	1,6605	0,36947839	-0,3620341	-0,06808314	
19	1,6605	0,13271046	-0,36844878	-0,0834882	
20	1,6605	0,37165211	-0,21272649	-0,03098583	
21	1,6606	0,69571072	-0,04791578	0,02873399	
22	1,6606	0,75158816	-0,14594635	0,00772113	
23	1,6606	1,09030866	-0,24335131	0,00336863	
24	1,6606	1,24196687	0,05030854	0,08493262	
25	1,6606	1,35601028	-0,0263646	0,07260243	
26	1,6606	1,50040742	-0,16594467	0,046467	
27	1,6606	1,21789044	0,04969594	0,08337593	

6. Opening Function, Close GUI Confirmation:

- First, lets deal with the opening function. Here, you will find the standard tool bar and the "close GUI confirmation dialog".

```
2.      %%
3.      function data_processing_tool_OpeningFcn(hObject, eventdata, handles,
4.          varargin)
5.      handles.output = hObject;
6.      set(hObject,'toolbar','figure'); %enables toolbar
7.
8.      %this variable used to prevent users from breaking the GUI
9.      %the variable is set to 1 once the data has been processed
10.     handles.processDataCompleted = 0; %
11.
12.     %this command asks the user to confirm closing of GUI
13.     set(handles.figure1,'CloseRequestFcn','closeGUI');
14.
15.     % Update handles structure
guidata(hObject, handles);

%%%
function closeGUI

selection = questdlg('Do you want to close the GUI?',...
    'Close Request Function',...
    'Yes','No','Yes');

switch selection,
    case 'Yes',
        delete(gcf)
    case 'No'
        return
end
```



7. Reset Button

Now, for the reset button. Resetting your GUI to the default state can save the user a lot of time and frustration. Instead of closing and opening the GUI to get to the starting state, the user can simply click on the "reset" button.

```
16.      function reset_pushbutton_Callback(hObject, eventdata, handles)
17.          %resets the GUI by clearing all relevant fields
18.
19.          handles.processDataCompleted = 0;
20.
21.          %clears the axes
22.          cla(handles.axes1,'reset');
23.
24.          %set the popupmenu to default value
25.          set(handles.plot_popupmenu,'Value',1);
26.
27.          %clears the contents of the listbox
```

```
28.     set(handles.inputFiles_listbox,'String','');
29.     set(handles.inputFiles_listbox,'Value',0);
30.
31.     %updates the handles structure
guidata(hObject, handles);
```

This is the end of the tutorial.