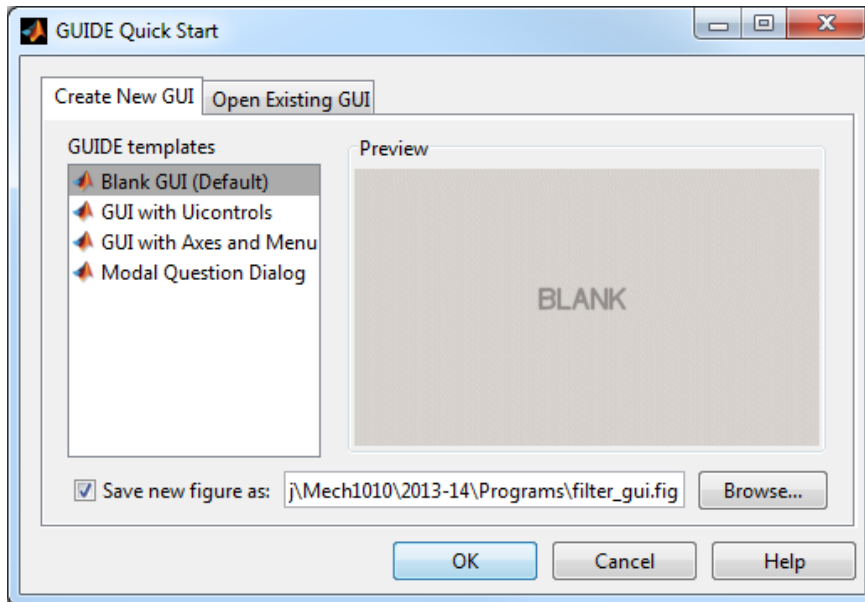






## Exercise 7.1 – Creating a Graphical User Interface

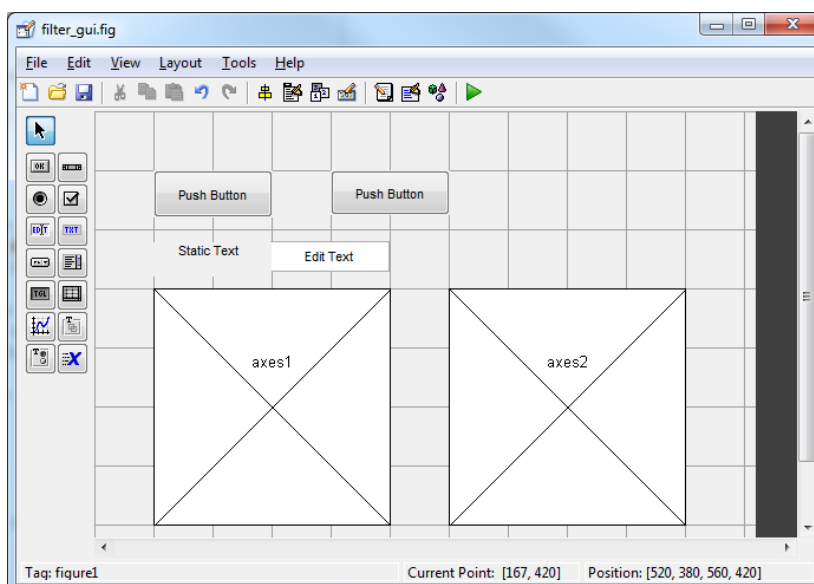
In this exercise we will look creating a Graphical user interface for the data filter program we created in a previous's lab. From it we should see that while more complex with MATLAB than with LabVIEW we can still create User Interfaces with relative ease using the GUIDE toolbox.

1. Open MATLAB and make sure you are set to the correct working directory. Type `guide` in the Command Window to open GUIDE. The GUIDE Quick Start window will open. On the **Create New GUI** select **Blank GUI (Default)** from the GUIDE templates then check the **Save new figure as:** box and enter the name `filter_gui.fig` saving to your working directory and click OK. This will open the GUIDE editor window and also the associated `.m` file containing the code used by the GUI.



2. We are now going to build a simple GUI which will feature two push buttons, a text box and two axes. All of these UI objects can be placed on the figure workspace by clicking on the relevant button on the toolbar.

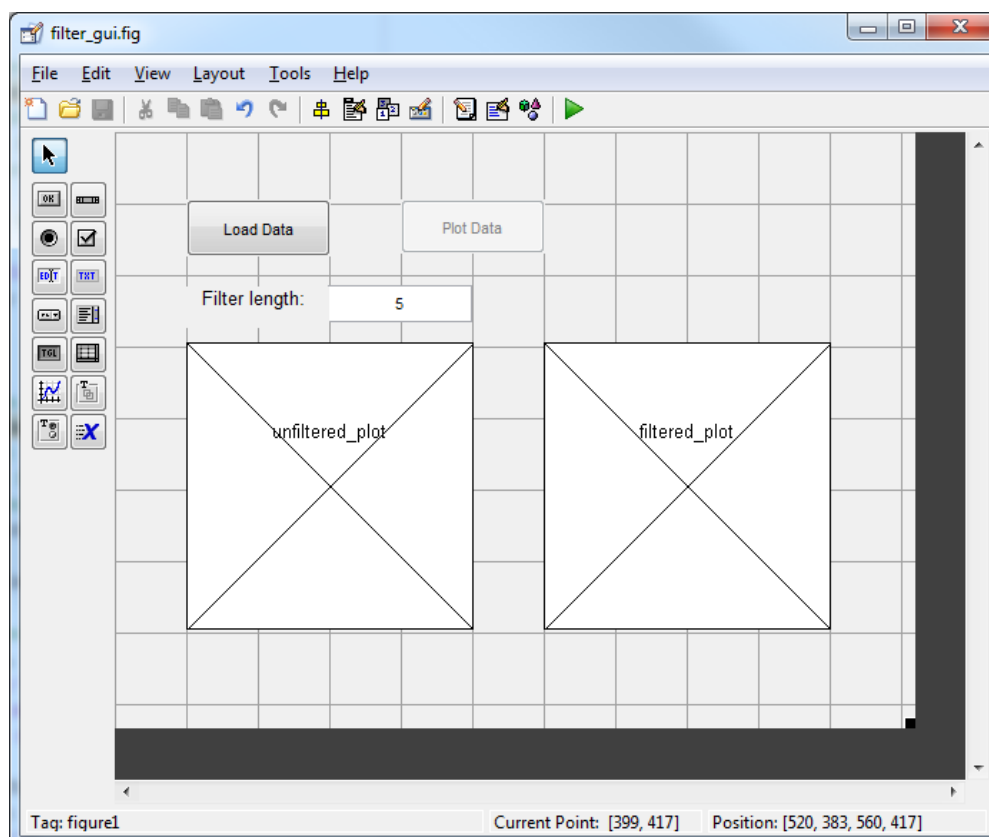
Click the icon for a push button:  then left-click in the blank figure and drag the mouse to place the push button, you'll notice the default text is Push Button, we will change this shortly. Add another push button and then an edit text box: , a static text box:  and two axes: . Arrange the objects on the figure to look like this:



- We are now going to alter some of the properties of these UI objects. Firstly we will look at the first push button. If we click on it we should see in the bottom left of the window the text Tag: pushbutton1. Now double click on the push button. This will open the properties inspector for pushbutton1. What we see here is a list of all the properties in the left column and the values these properties are set to in the right column. Have a look at all the properties you can alter for a push button. We are now going to change two properties, firstly look for the String property. This is the text that appears on the push button. In the value column, change this to: Load Data. Now we are going to change the Tag, this is how you will access the properties of the pushbutton using `handles.tag`. Change this to `load_data`. Note that tag must follow the same rules as variable names, so no spaces or punctuation other than an underscore. Once you've typed the new tag, remember to press enter otherwise the tag may not be updated, you can check by selecting an object and looking at the tag shown in the bottom left corner of the GUIDE window.
- Now repeat this for the other 5 objects, The table below shows the properties for each object that should be changed:

UI Object	Property Name	Property Value
pushbutton1	String	Load Data
	Tag	load_data
pushbutton2	String	Plot Data
	Tag	plot_data
text1	String	Filter length:
	FontSize	10
edit1	String	5
	Tag	filter_length
axes1	Tag	unfiltered_plot
axes2	Tag	filtered_plot

- Your GUIDE window should now look something like this:



We are now ready to start adding functionality to our code. Save the `filter_gui.fig` in the GUIDE window. You should see the editor window update to take into account the new UI objects we've added to the GUI. Close the GUIDE window.

6. Have a look at `filter_gui.m` in the Editor. You'll see a number of functions in the file. The top one `filter_gui` sets up the figure, we should not make any changes to this function. Scroll down the page and we should see the second function `filter_gui_OpeningFcn`. This function executes just before the GUI is made visible. You can put code in here for initialising values in text boxes or disabling push buttons. For this exercise we are going to set the *Enable* property of the Plot Data button to *off*. To do this we will use the `set` function. This is of the form:

```
set(handles.tag, 'PropertyName', 'PropertyValue')
```

In this case our *tag* is `plot_data`, our property name is 'Enable' and our 'PropertyValue' is 'off'. Note these are case sensitive and the apostrophes should be included. Place this at the end of the `filter_gui_OpeningFcn`.

The next function `filter_gui_OutputFcn` is for returning outputs to the command line, we'll not be editing this function. Below these three functions are the functions we are interested in. We should have 3 callback functions `load_data_Callback`, `plot_data_Callback` and `filter_length_Callback`. We also have the `filter_length_CreateFcn` which sets some properties of the edit text box. We are interested in the first two Callbacks.

7. The first one we are going to look at is the `load_data_Callback`. The code contained within this function will execute when the user presses the Load Data button. At present this is empty aside from some comments. Now open your `enforcer.m` file from week 6 in the Editor. If you didn't complete the exercise, download the solution from the Lab 6 section on the VLE.

In `enforcer.m` select the lines of code that load in the data and change the filename to `noisy_data.csv` and the two lines of code that create the vectors `unfiltered_data` and `time`. Cut them from `data_filter.m` and paste them at the bottom of the `load_data_Callback` function in `filter_gui.m`.

8. We now have two further tasks for this Callback, firstly we need to be able to pass the variables `unfiltered_data` and `time` to other Callbacks and secondly we need to enable the Plot Data button:

- a. To pass the data between Callbacks, we are going to use the `guidata` function. We could use globals but this way is perhaps neater. `guidata` can be used to pass the data using the handles data structure, as such we need to place our variables into the handles data structure. A handle is simply a number that points to a graphical object and is used to access the properties of the object. The handles structure already contains all the handles to UI objects but we wish to add our data to it too. To do this we simply add handles and then a full-stop before any variable we wish to add so we should end up with:

```
handles.unfiltered_data = data(:,2);
handles.time = data(:,1);
```

We now need to store the handle data in the user interface so it can be retrieved by other Callbacks. To do this we use the function `guidata` passing it the first input argument as a handle to the GUI figure, in this case `hObject` and then the data we wish to store, in this case it is the `handles` structure. So we should have:

```
guidata(hObject, handles)
```

at the bottom of the `load_data_Callback`.

- b. Now we have loaded the data, we wish to enable the Plot Data push button. To do this we will use the `set` function as in step 6, but this time the property value will be set to 'on'. Place this at the end of the `load_data_Callback` function.

9. So we have now completed the code that will run when the user presses the Load Data button. We are now going to look at what needs to happen when the Plot Data button is pressed, now it is enabled. When the Plot Data button is pressed we are going to call our UDF `moving_average_filter`. This will take two input arguments, our unfiltered data and the length of our filter, `n`. We need to get the value of `n` from the text box in the GUI, allowing the user to update the filter length. To access the value we need to use the `get` function, this takes the form

```
result = get(handles.tag, 'PropertyName')
```

in this case our *tag* is `filter_length` and our 'PropertyName' is 'String'. This will return the numeric value in the text box as a character string. We need to use the function `str2num` to change it to a numeric. So the code will look like:

```
length_of_filter = get(handles.filter_length, 'String');
```

```
n = str2num(length_of_filter);
```

in the `plot_data_Callback`. We can now call our UDF in the normal way, however we need to maintain the structure of the handles data so when we pass the UDF the `unfiltered_data`, we need to use `handles.unfiltered_data`:

```
filtered_data = moving_average_filter(handles.unfiltered_data,n);
```

10. Finally we are going to plot the unfiltered data against time and place it in our first axes, specified by the tag `unfiltered_plot` and plot our `filtered_data` against time and place in the second axes, specified by the tag `filtered_plot`. We can use our plot functions as normal except we need to pass the handle to the object as the first input argument. Here is the code that you will use for the unfiltered plot:

```
plot(handles.unfiltered_plot, handles.time, handles.unfiltered_data, 'r');  
xlabel(handles.unfiltered_plot, 'time (sec)')  
ylabel(handles.unfiltered_plot, 'position (m)')  
title(handles.unfiltered_plot, 'unfiltered data')
```

Now do the same for the second plot with the filtered data, note that filtered data has not been added to the handles data structure so just `filtered_data` will suffice.

11. Save your `filter_gui.m` and run it. You should see the GUI window open, test the functionality of your code by loading data and then plotting data. Also update the filter length and plot the data again. If the spacing of your GUI isn't quite right, launch GUIDE and open your saved `.fig` file and rearrange the UI objects to make more space, then save your figure again before running your m-file.

If you finish, have a look at other properties that you can change in UI objects and also the figure window itself.

## End of exercise 7.1