

Intro to GUI Design with Matlab App Designer

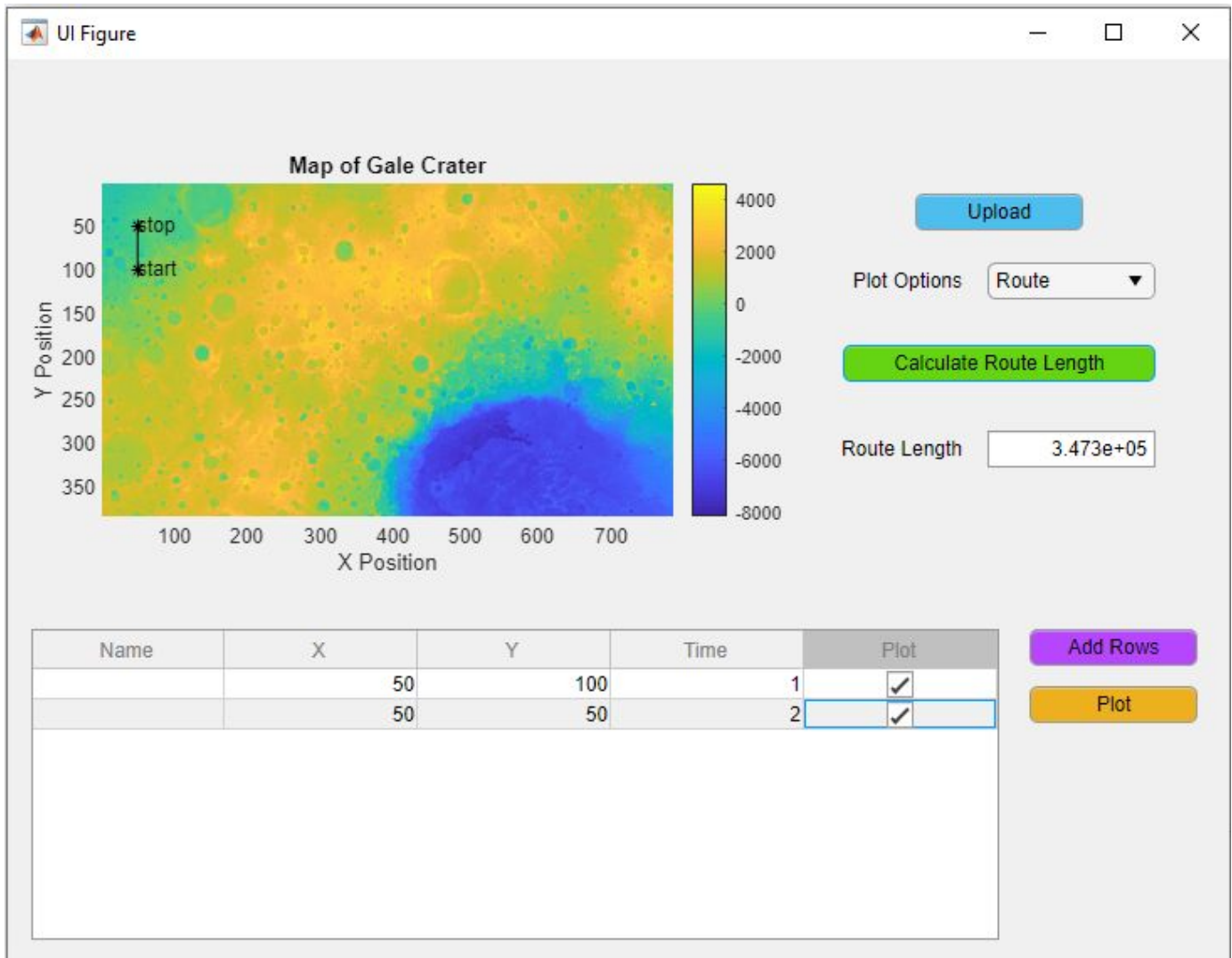
Tutorial



Prepared by Katelyn Brinker (brinker@iastate.edu)
Fall 2020

Workshop Objectives

The objective of this workshop is to get familiar with the Matlab app designer platform through building an app. The app we'll be building in this tutorial is a Mars rover route planning app that will allow the user to plot rover routes on a map of Mars. In building this app, you'll get familiar with the various components (axes, buttons, drop down menus, tables, and edit fields) that are commonly used in apps.



Getting Matlab Setup

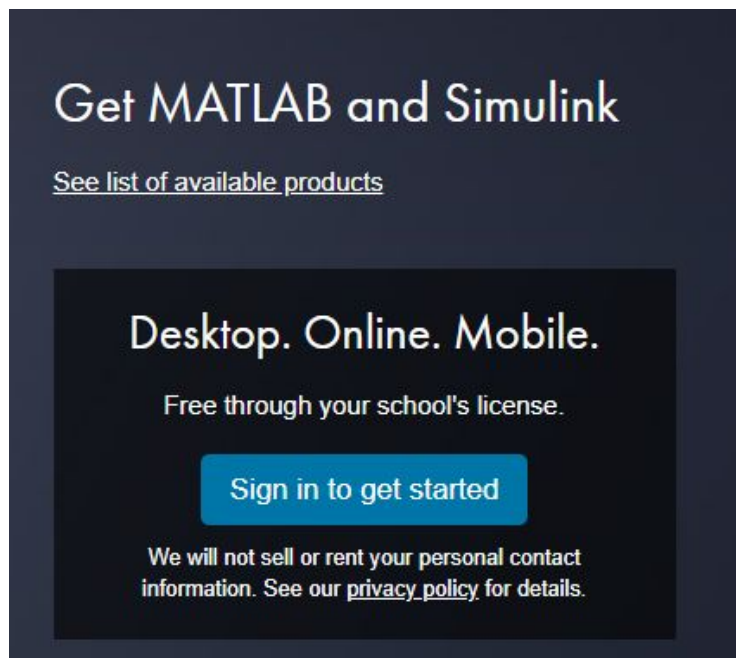
Iowa State has a full university license which means that any student can download Matlab and any/all toolboxes to their personal computers. This also allows you to use the Matlab online coding environment. For this workshop, it is recommended that you set up an account and use the online coding environment unless you know your computer can handle running the Matlab application.

To get started use these links:

<https://it.engineering.iastate.edu/how-to/installing-matlab/>

<https://www.mathworks.com/academia/tah-portal/iowa-state-university-1082052.html>

Scroll down and click on the “Sign In to Get Started” button. If you don’t already have a Matlab account you’ll need to set one up with your ISU email address.

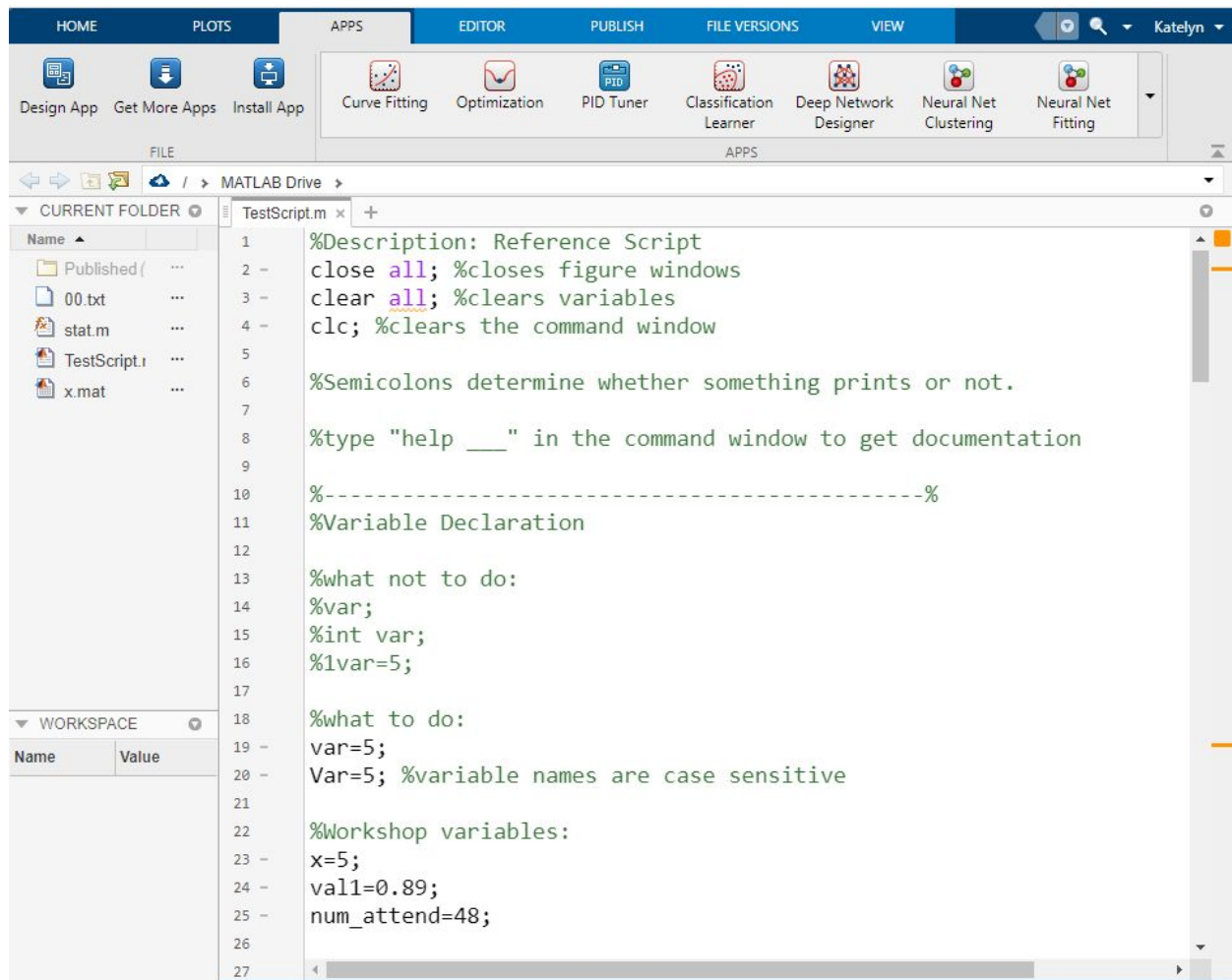


Once you’ve gone through the steps and have a student licence affiliated with your account, you’ll be able to open the Matlab online coding environment through this link:

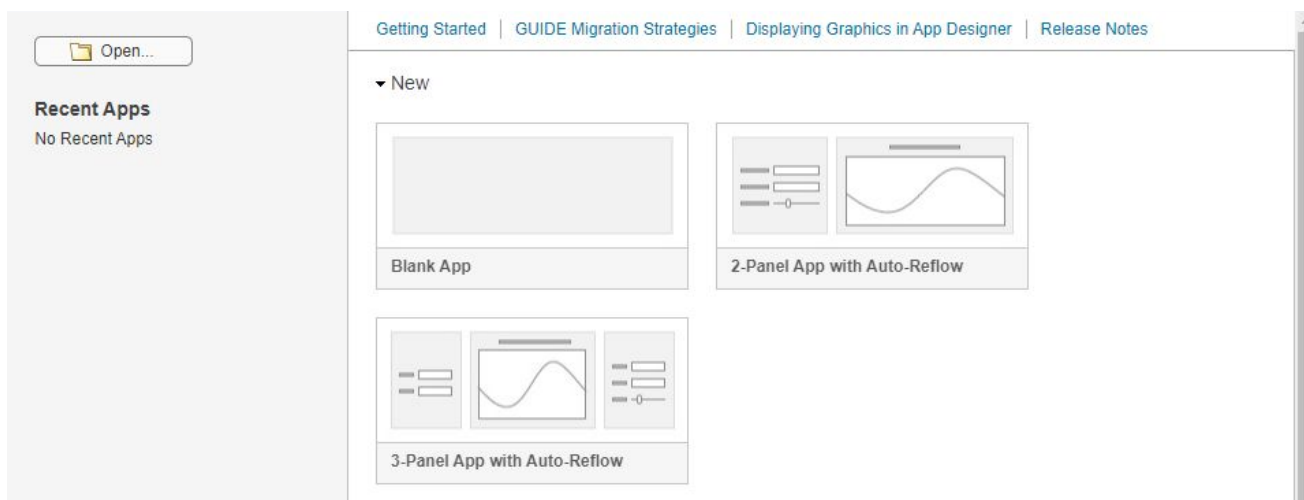
<https://www.mathworks.com/products/matlab-online.html>

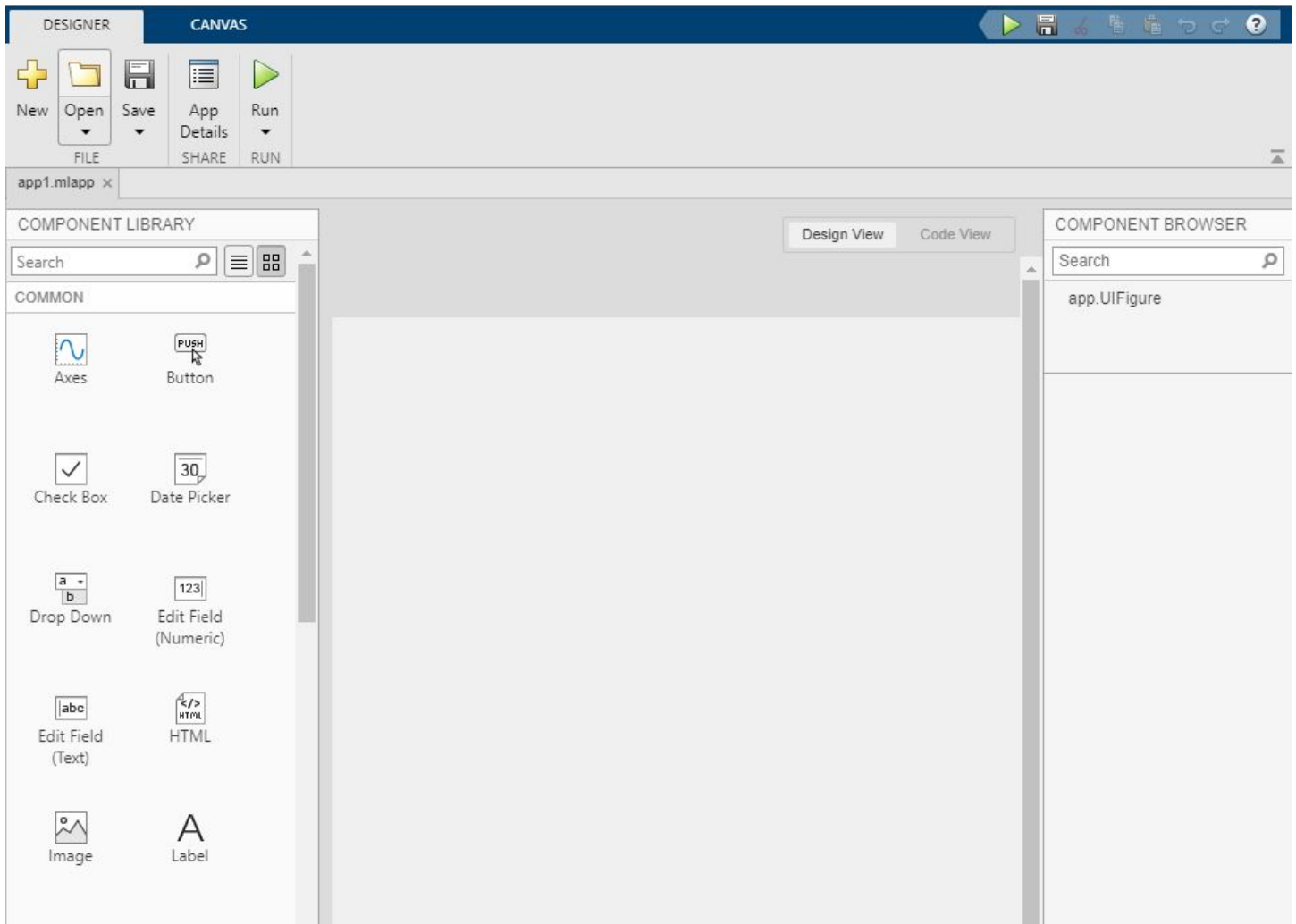
Alternatively, you can use the VDI server and run Matlab through that without creating a Mathworks account. Instructions for this can be found here: <https://it.engineering.iastate.edu/how-to/install-connect-to-vdi-pc/>

Once you have Matlab installed or setup with the online coding environment, open it and go to the APPS tab in the top menu.



Click “Design App” in the upper left corner and then select “Blank App” in the App Designer Start Page.



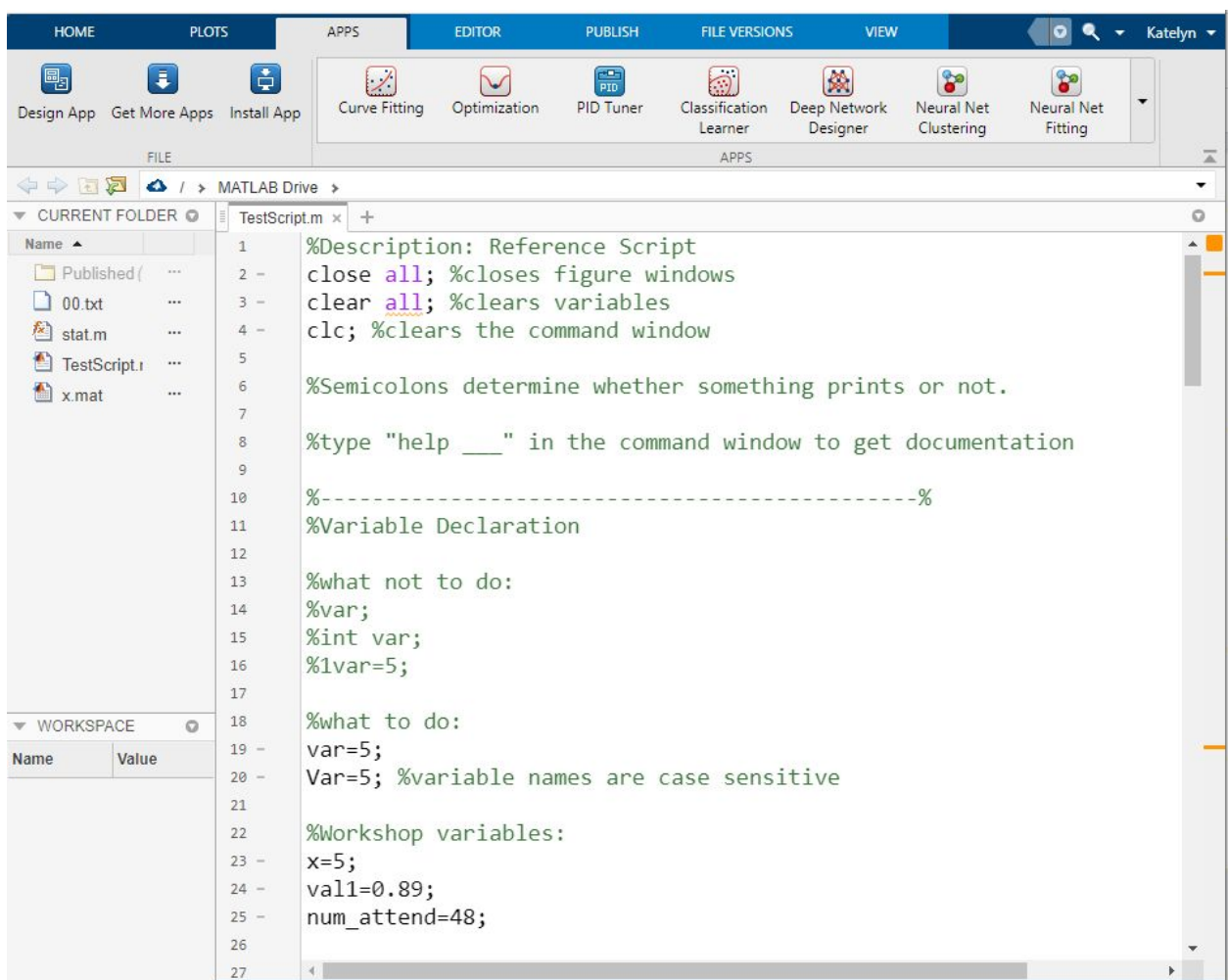


You are now ready to start designing your app.

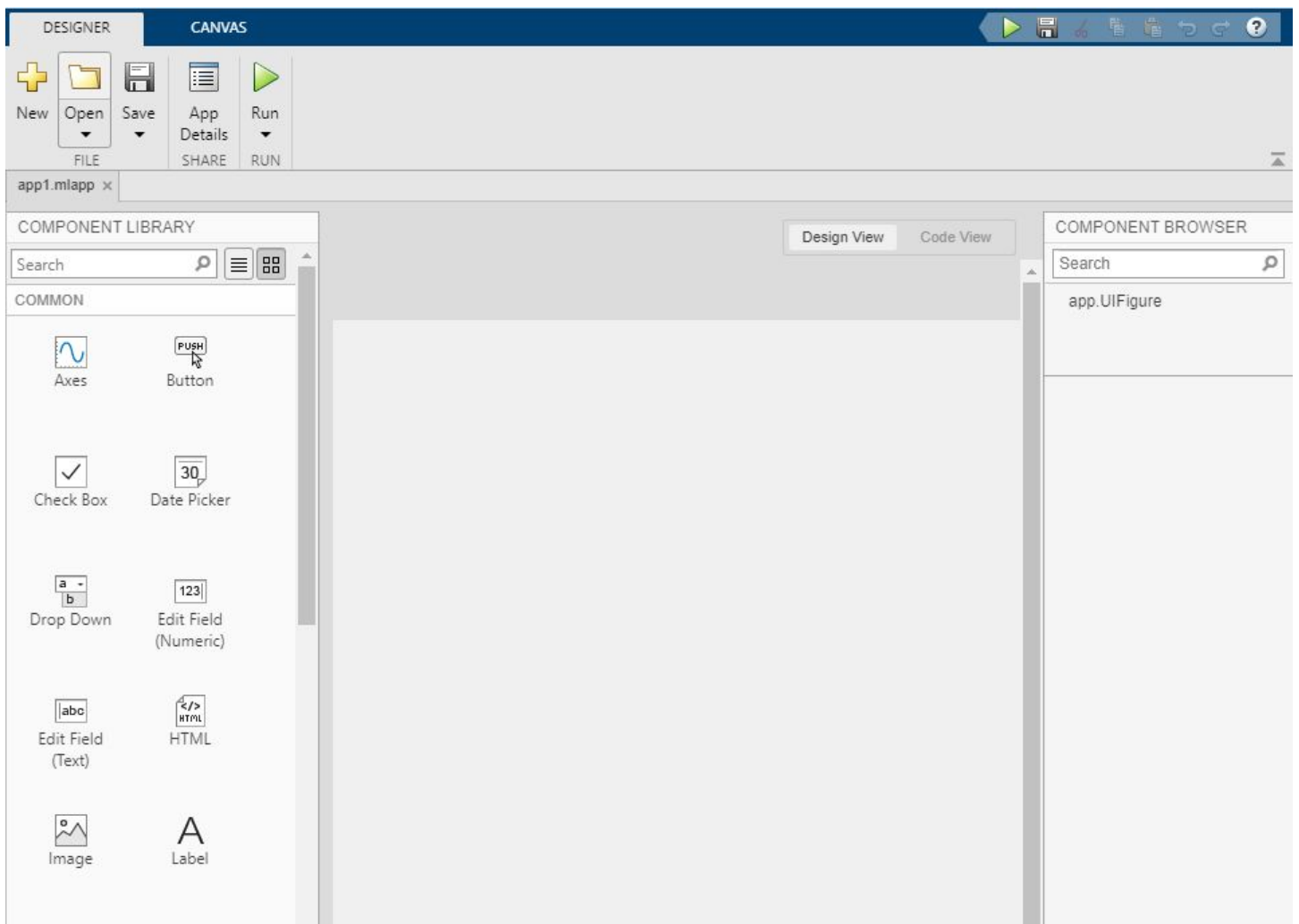
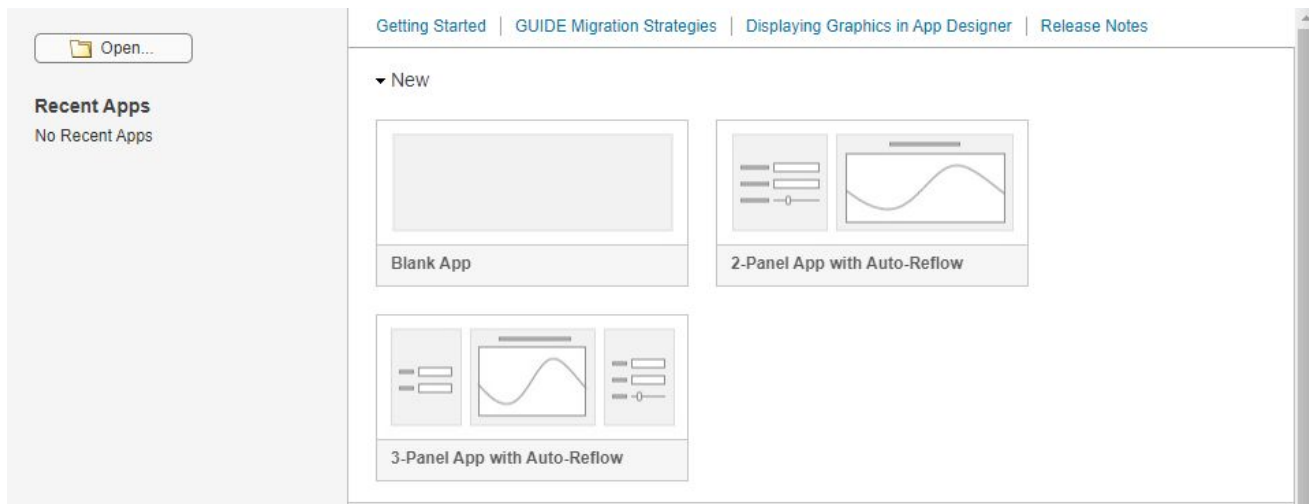
Getting Setup

In Matlab we typically operate out of a folder. We want all of our data files, function files, and scripts to reside in this folder. Included in the workshop zip folder there is a data file “DEM.mat” and an app file “RoverRoute.mlapp”

1. Place the “DEM.mat” file and the “RoverRoute.mlapp” file in the folder you plan to operate out of.
 - a. If you are using the online coding platform and have already started a new app, switch to the Matlab Online tab in your web browser. Navigate to the “Home” tab at the top of the interface and then click the “Upload” button that resides towards the upper left of the ribbon. Select the files and check that they appear in the “Current Folder” list on the left after they’ve uploaded.
2. If you haven’t started a new app yet, do so now. Go to the APPS tab in the top menu.



3. Click “Design App” in the upper left corner and then select “Blank App” in the App Designer Start Page.

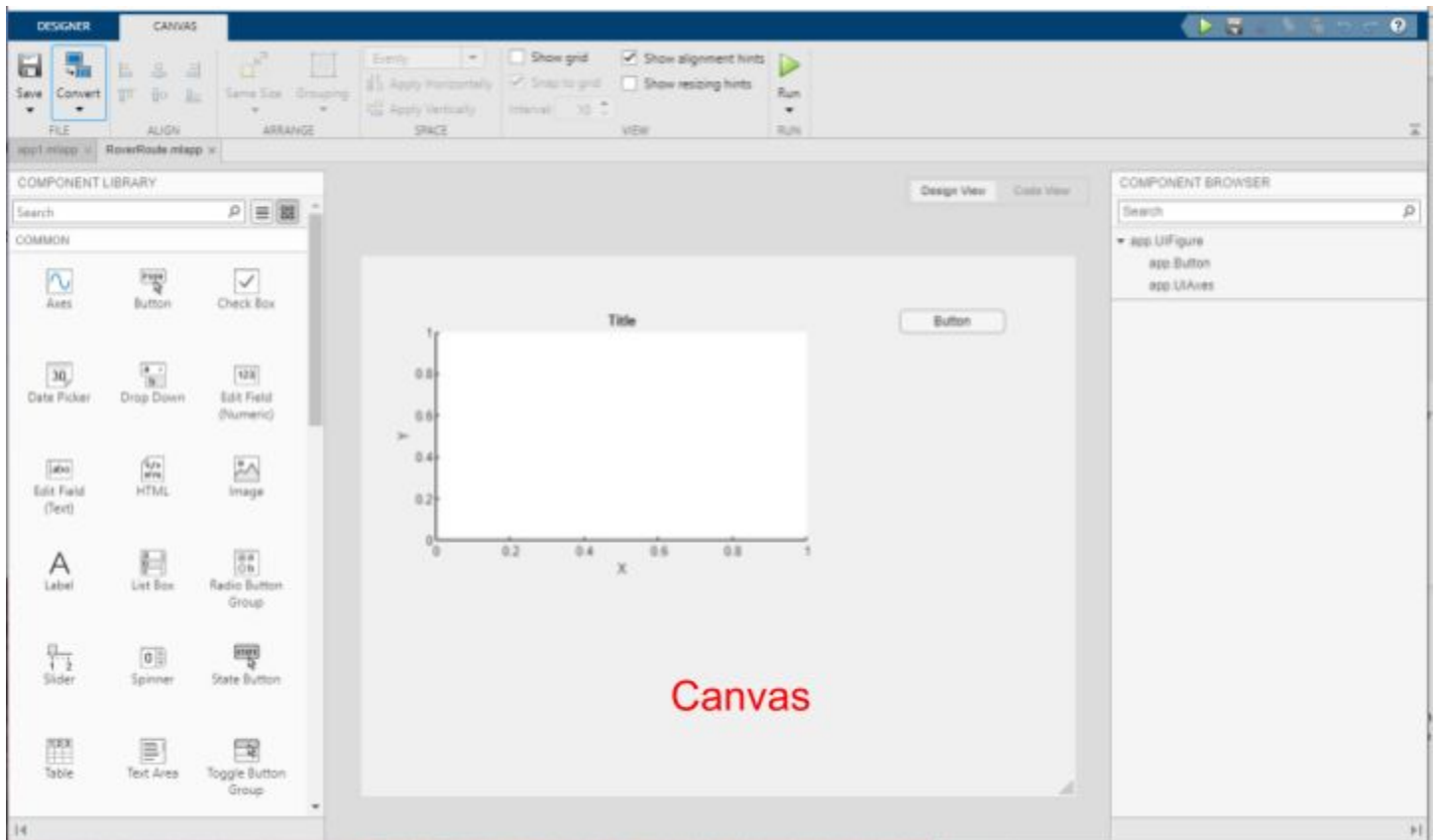


4. Save your app by clicking the save button in the upper left. Be sure to save your app in the same folder as your other workshop files.

A Tour of App Designer

In App Designer there are two main views: the design view and the code view. Generally, we use the design view to do the front end development of our app and the code view to do the back end development of our app.

The **design view** can be described by its three main components: the Component Library, the Component Browser, and the Canvas.



The Component Library provides you with component options that you can add to your app, while the Component Browser tells you what components are currently in your app. The Canvas is where you layout your app components. In the Canvas tab at the top of the App Designer window you'll find tools for aligning your components.

In the **code view**, you'll find the Code Browser, your code, and the Component Browser again. The code browser allows you to easily navigate to your callbacks and functions. If you click on something in the code browser your code will jump to that spot. You'll also notice in the top ribbon that you have the option to add functions, properties, and callbacks. What these are and how they're used will be covered later in this tutorial.

DESIGNER

EDITOR

Save

Callback

Function

Property

App Input Arguments

Go To

Find

Comment

Indent

Enable app coding alerts

Show Tips

Run

FILE

INSERT

NAVIGATE

EDIT

VIEW

RESOURCES

RUN

RoverRoute.mlapp x

CODE BROWSER

Callbacks

Functions

Properties

Search

+

Add a callback function to make your app respond to user interactions such as button clicks.

APP LAYOUT

Design View

Code View

COMPONENT BROWSER

Search

app.UIFigure

app.Button

app.UIAxes

```

1 classdef RoverRoute < matlab.apps.AppBase
2
3     % Properties that correspond to app components
4     properties (Access = public)
5         UIFigure matlab.ui.Figure
6         Button    matlab.ui.control.Button
7         UIAxes    matlab.ui.control.UIAxes
8     end
9
10    % Component initialization
11    methods (Access = private)
12
13        % Create UIFigure and components
14        function createComponents(app)
15
16            % Create UIFigure and hide until all components are created
17            app.UIFigure = uifigure('Visible', 'off');
18            app.UIFigure.Position = [100 100 679 512];
19            app.UIFigure.Name = 'MATLAB App';
20
21            % Create Button
22            app.Button = uibutton(app.UIFigure, 'push');
23            app.Button.Position = [513 440 100 22];
24
25            % Create UIAxes
26            app.UIAxes = uiaxes(app.UIFigure);
27            title(app.UIAxes, 'Title')
28            xlabel(app.UIAxes, 'X')
29            ylabel(app.UIAxes, 'Y')
30            app.UIAxes.Position = [33 209 400 253];
31
32            % Show the figure after all components are created
33            app.UIFigure.Visible = 'on';

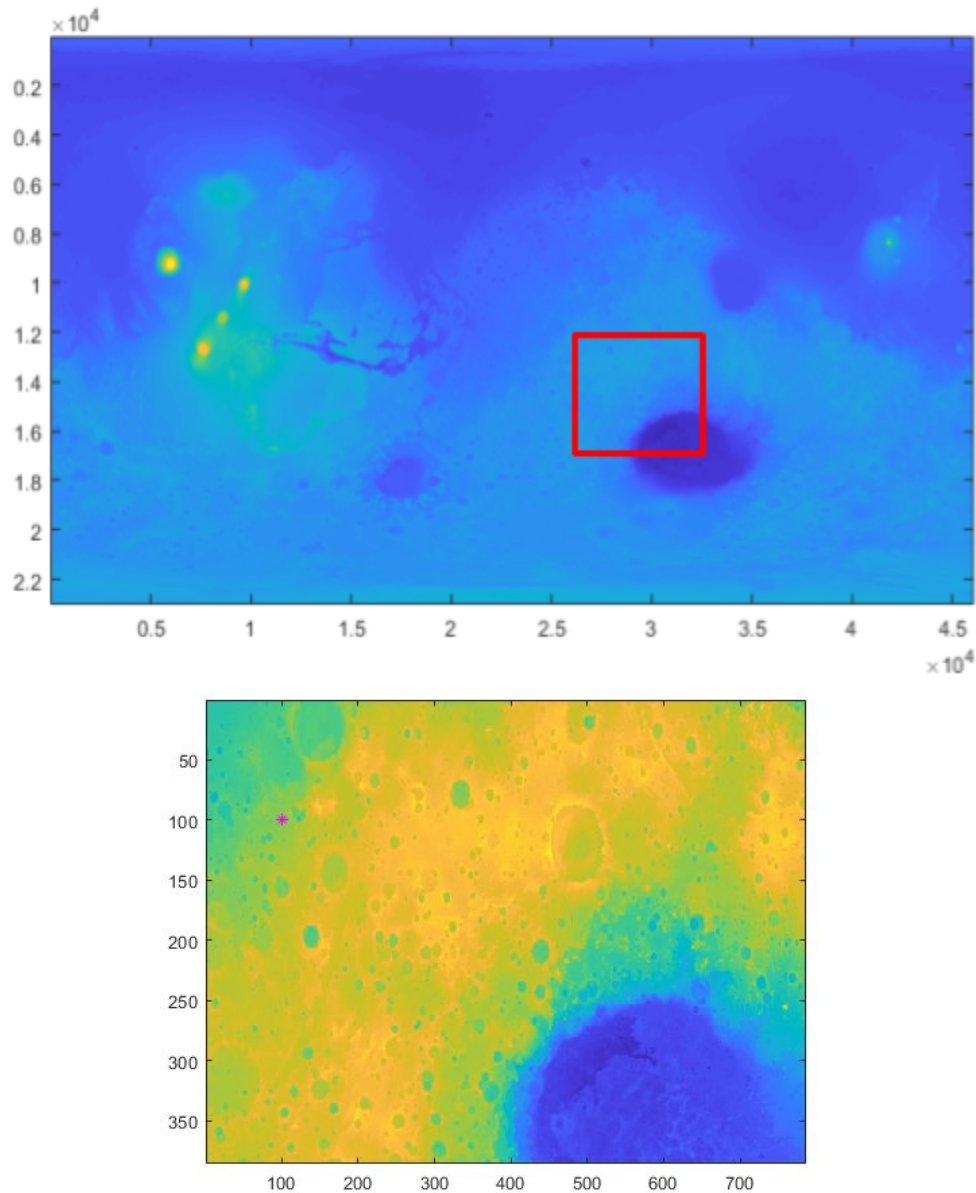
```

Data Preparation

The terrain data used for this app is stored in DEM.mat. The terrain data was originally pulled from the Mars Orbiter Laster Altimeter (MOLA) data sets available from USGS.

- https://astrogeology.usgs.gov/search/details/Mars/GlobalSurveyor/MOLA/Mars_MGS_MOLA_DEM_mosaic_global_463m/cub

MOLA, an instrument on the Mars Global Surveyor, produced the first global topographic map of Mars. A section of this data from around Gale Crater was pulled, imported into Matlab, and then cropped and downsampled. Lastly new axes were created to make it easier to index points in the map.



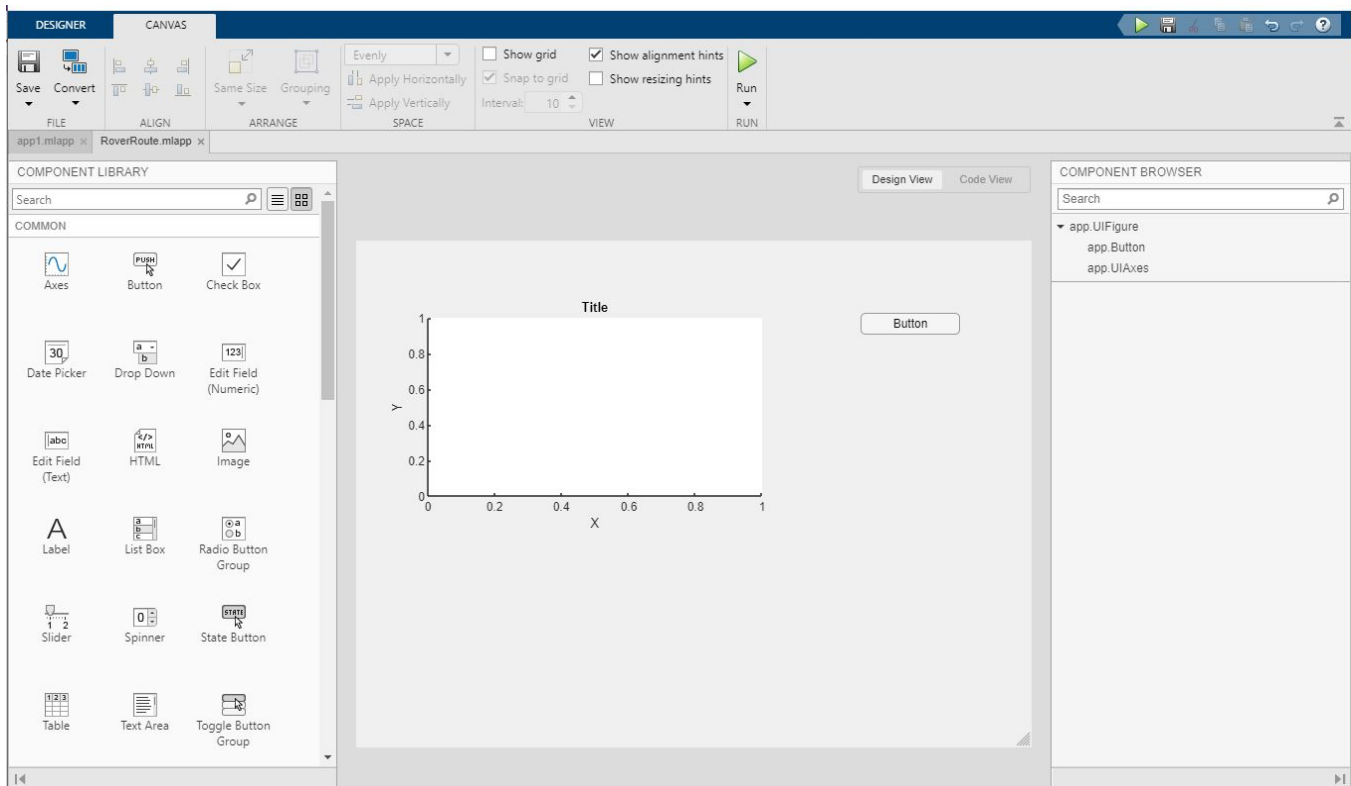
The file DEM.mat has the digital elevation map (DEM) data as a matrix, the axes as vectors, and the resolution of the DEM (meters per grid spot/matrix entry).

Section 1: App Building- Uploading Data and Plotting

App Building: Uploading Data and Plotting

The first functionality we are going to implement is uploading the DEM file and plotting it as a contour map when a button is clicked.

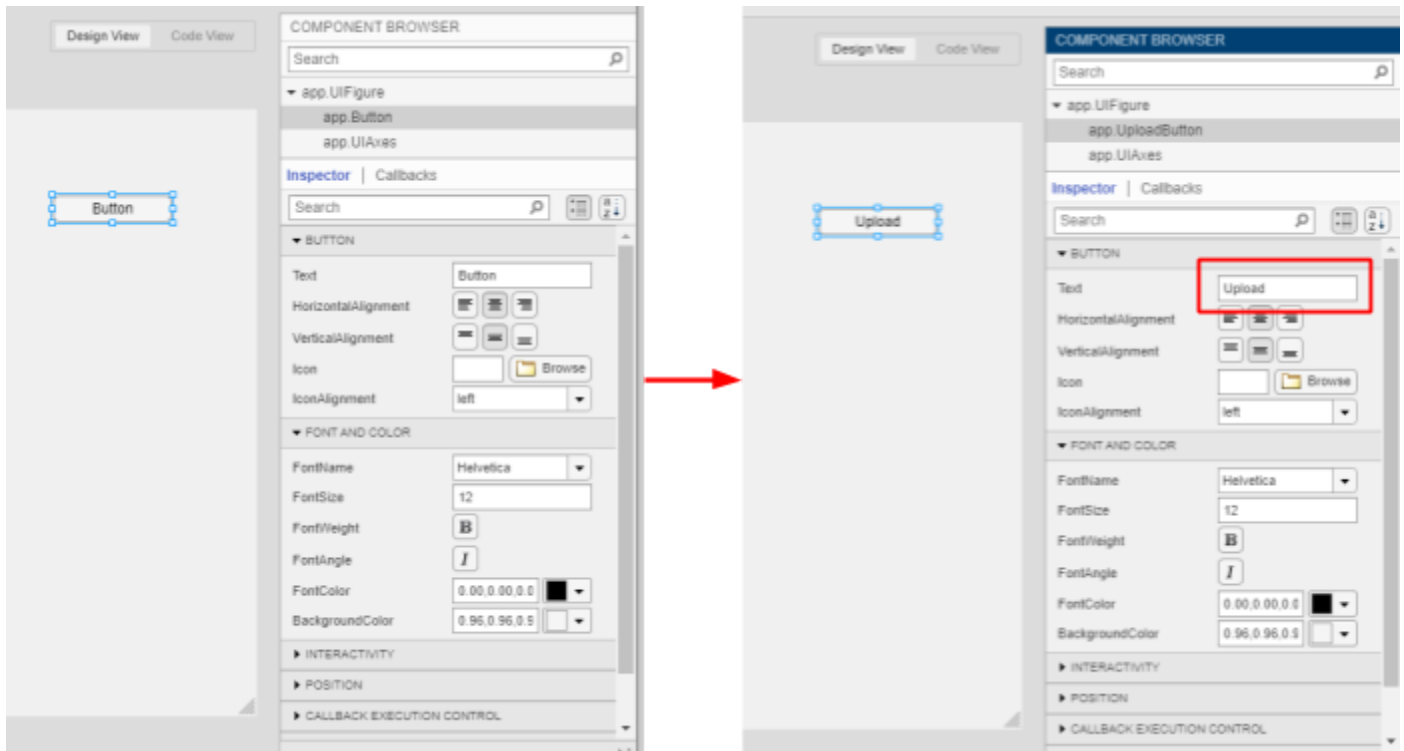
1. In the Design view drag and drop a button and an axes object into the canvas.



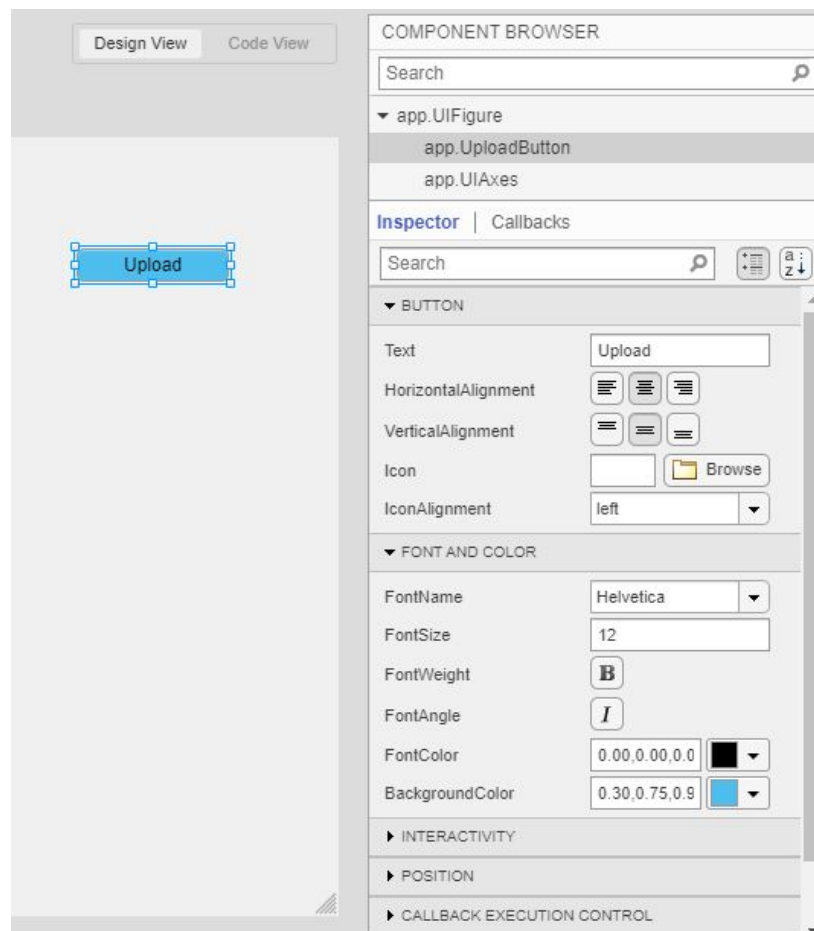
Note: You can change the size of components in the canvas by selecting them and then pulling on the corners and sides of the object, just like you would resize a picture or shape in PowerPoint.

2. Click on the button either in the canvas or in the Component Browser to pull up the Property Inspector. This will appear below the Component Browser on the right. Change the text displayed on the button by editing the Text field in the Property Inspector.

Notice that the name of the button in the Component Browser changed when you changed the text displayed on the button. You can right click on a component in the Component Browser and change its name, too. When this is done, your code will automatically update. The name in the Component Browser is independent from the text you display on your button in the canvas. However, it's good to name your components intuitively so that it's easier to code with them.

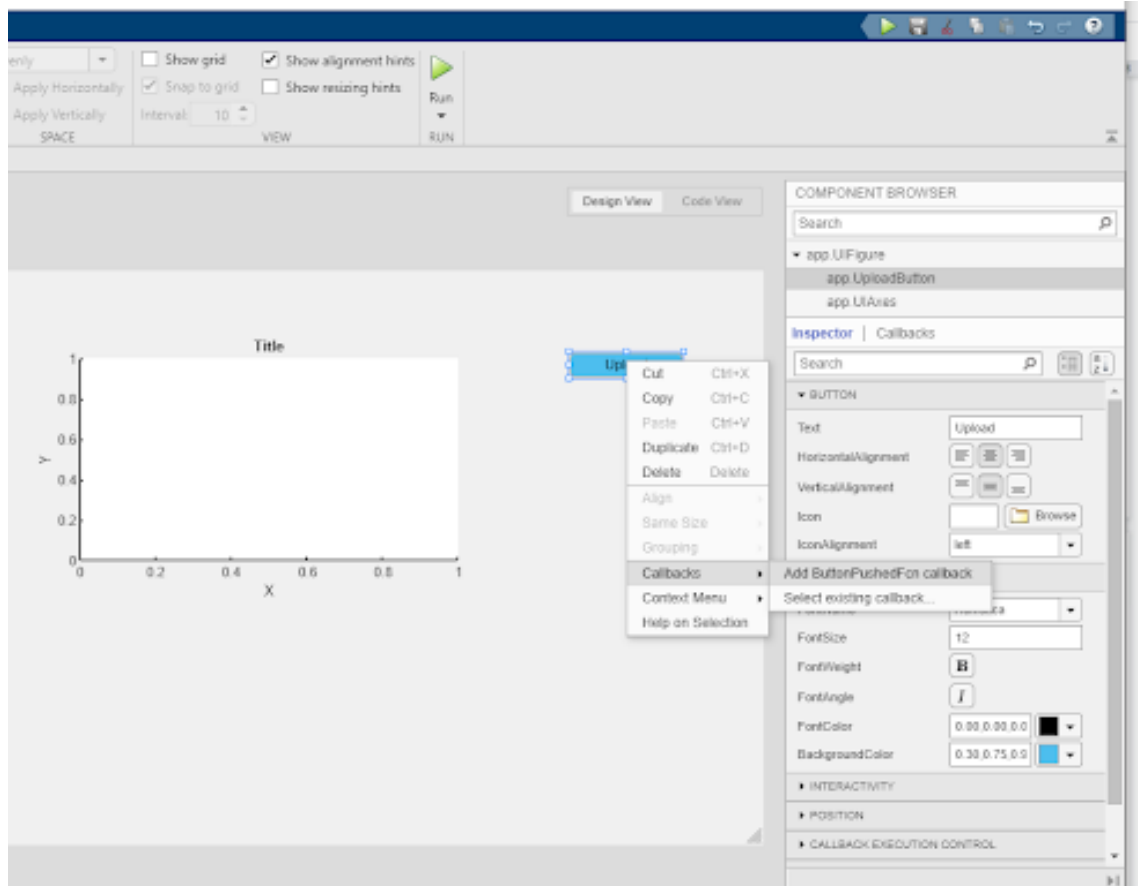


3. Change the background color of the button by selecting a color from the dropdown menu for the BackgroundColor field in the Property Inspector.



Callbacks are the code that gets executed when a user interacts with the app.

4. Create a callback for the button: right click on the button in the canvas or in the Component Browser. Select Callbacks and then “Add ButtonPushedFcn callback.” This will bring you to the code view and start a new function in the callbacks section of your code.



In the callback for when the Upload button gets pushed, we want to upload the DEM.mat file, assign the variables stored in the file to properties, and then plot the data on the Axes component we placed in the canvas earlier.

Properties are like global variables for an app. They can be used in any function or callback throughout the app and they retain their values from one operation to the next. **Private Properties** can only be shared within the app while **Public Properties** can be shared within and outside the app.

For this app, we want to be able to use Matlab to debug so we'll use Public Properties.

5. Inside the UploadButtonPushed function (between the word `function` and the word `end`) implement the following code to load the DEM.mat file and to assign the variables within the DEM.mat file to Public Properties (app.grid, app.gridsize, etc.). Add a semicolon at the end of each line to avoid printing variable values to the command window back in Matlab every time the app is run.

```

% Callbacks that handle component events
methods (Access = private)

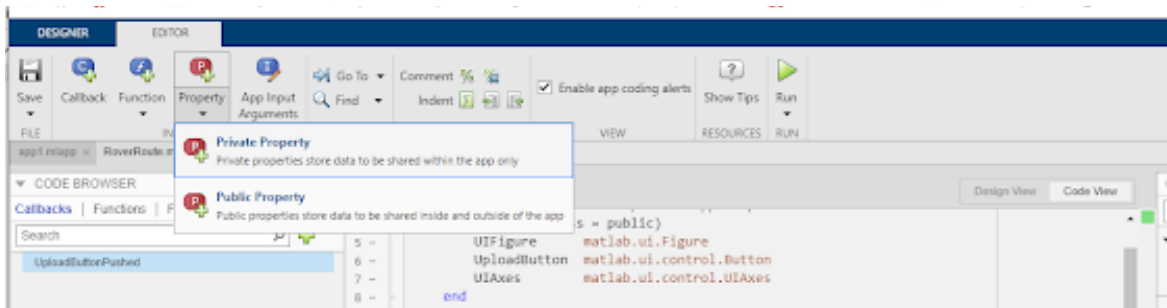
    % Callback function
    function UploadButtonPushed(app, event)

        %load data file
        Data=load('DEM.mat');
        %assign file variables to app private properties
        app.grid=Data.grid;|
        app.gridsize=Data.gridsize;
        app.dX=Data.dX;
        app.dY=Data.dY;

    end
end

```

6. Add Public Properties by selecting the Public Property option from the drop down in the ribbon at the top of App Designer. This will transport you to the Public Properties portion of your code.



7. Add the following code to the Public Properties section of your code. This code will define Public Property variables.

```

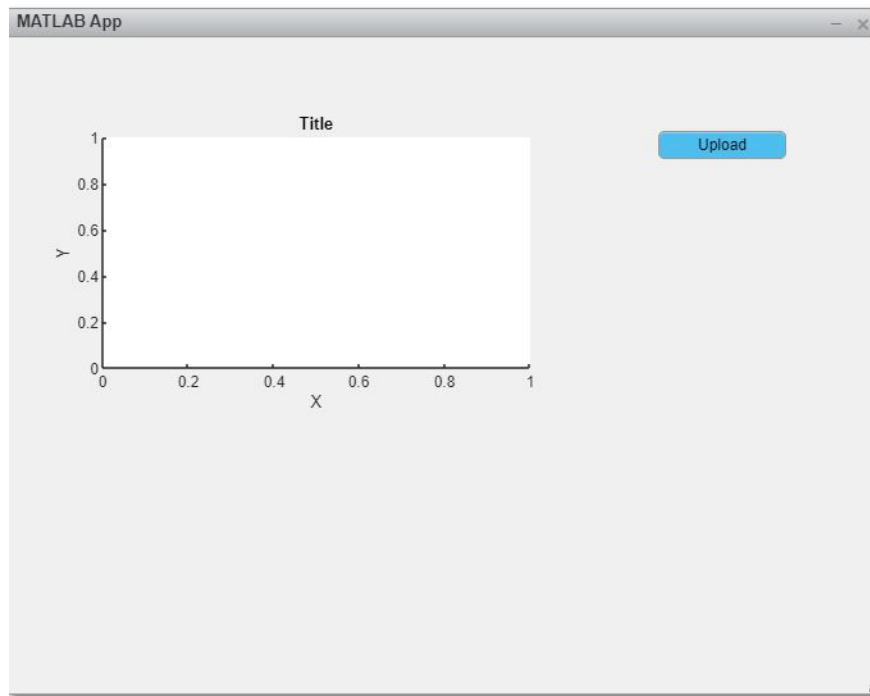
properties (Access = public)
    %DEM File Data
    grid
    dX
    dY
    gridsize
end

```

Note that in our callback functions we used `app.grid` or `app.dX` to assign variables from the `DEM.mat` file to Public Properties, while here we do not have the “app.” prefix. This is because the Public Properties are variables that belong to the app class. To access parts of a class we use dot indexing. You can read more about Matlab classes here:

- https://www.mathworks.com/help/matlab/matlab_oop/create-a-simple-class.html

8. Test your app by clicking run in the ribbon at the top of App Designer. A new window will come up with your app. Click the upload button. If no errors occur then your app is working correctly so far. If errors do occur see page 19 for debugging tips.



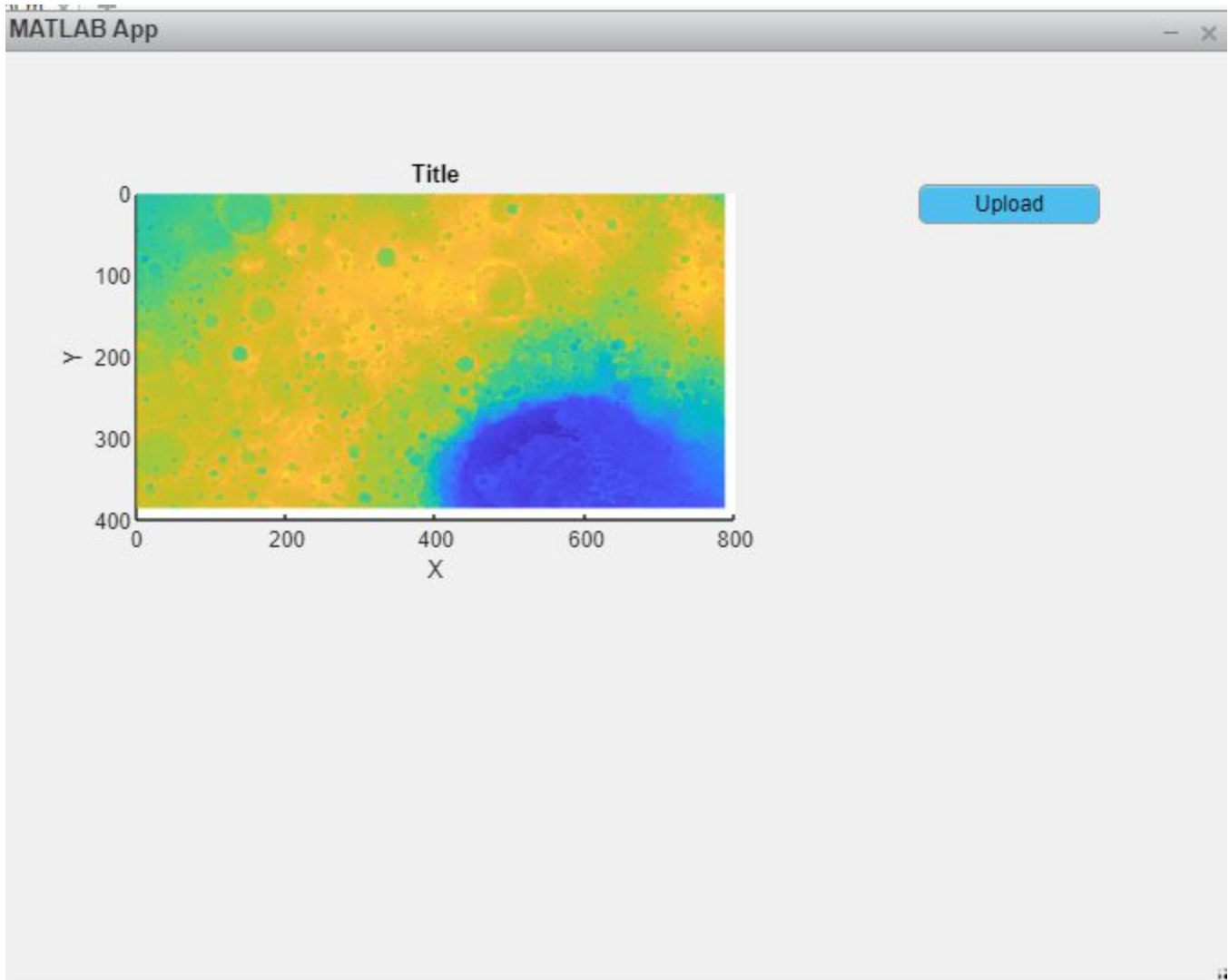
9. Plot the DEM by adding the following code to your UploadButtonPushed callback function. You can navigate to this callback function quickly by clicking on it in the Code Browser on the left in the Code View.

```
% Callbacks that handle component events
methods (Access = private)

% Callback function
function UploadButtonPushed(app, event)
    %load data file
    Data=load('DEM.mat');
    %assign file variables to app private properties
    app.grid=Data.grid;
    app.gridsize=Data.gridsize;
    app.dX=Data.dX;
    app.dY=Data.dY;
    %plot the grid on the axes component
    imagesc(app.UIAxes, app.dX, app.dY, app.grid);
end
end
```

The `imagesc` codes plots a scaled color image of the grid data. The first argument in the `imagesc()` function is the component of our app that we want to plot on: `app.UIAxes`. `app.UIAxes` is the Axes component we dropped into the canvas earlier/the only axes component we have in our Component Browser. The next two arguments in the `imagesc` function (`app.dX` and `app.dY`) are the Public Properties we assigned earlier in the `UploadButtonPushed` callback. Here they are telling the `imagesc` function what to use for the X- and Y-axes of the plot. `app.grid` is the DEM data in matrix form. This is what will be plotted on the axes component with higher elevations corresponding to yellow and lower elevations corresponding to blue.

10. Test your app by running it. When you click the Upload button you should see the map plot on your axes object.



App Building: Manipulating Plots

The map above is functional but it's not as pretty as it could be: the axes aren't labeled very descriptively, there's white space around the bottom and right hand side of the plot, and there isn't a title.

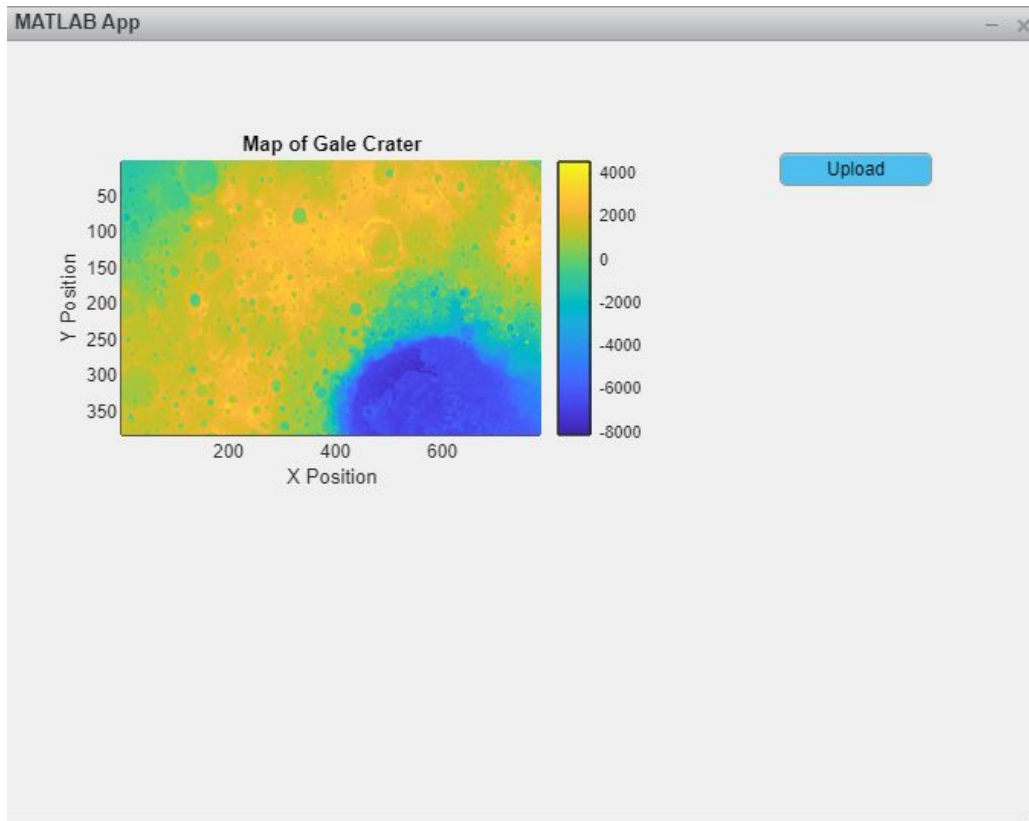
1. Add the following code below the `imagesc` function in your `UploadButtonPushed` callback function.

```
% Callbacks that handle component events
methods (Access = private)

% Button pushed function: UploadButton
function UploadButtonPushed(app, event)
    %load data file
    Data=load('DEM.mat');
    %assign file variables to app private properties
    app.grid=Data.grid;
    app.gridsize=Data.gridsize;
    app.dX=Data.dX;
    app.dY=Data.dY;
    %plot the grid on the axes component
    imagesc(app.UIAxes, app.dX, app.dY, app.grid);
    %Annotate Plot
    app.UIAxes.XLim=[1 max(app.dX)];
    app.UIAxes.YLim=[1 max(app.dY)];
    app.UIAxes.Title.String='Map of Gale Crater';
    app.UIAxes.XLabel.String='X Position';
    app.UIAxes.YLabel.String='Y Position';
    colorbar(app.UIAxes);
end
end
```

Similarly to how the components are objects of the app class, each component also has objects and properties associated with it that can be indexed through dot indexing. For example, the `UIAxes` component has a number of properties (`XLim`, `YLim`, `Title`, etc.) that we're accessing and changing above.

2. Test your app again and verify that your map plot looks how you want it to.

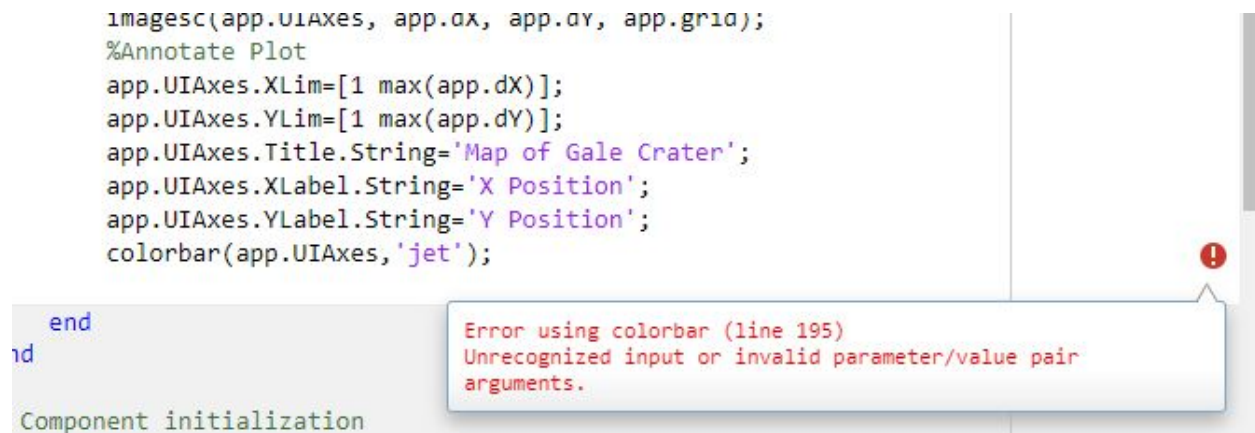


Documentation/Resources on buttons and plotting:

- Button properties: <https://www.mathworks.com/help/matlab/ref/matlab.ui.control.button-properties.html>
- App building components: https://www.mathworks.com/help/matlab/creating_guis/choose-components-for-your-app-designer-app.html
- UIAxes properties: <https://www.mathworks.com/help/matlab/ref/matlab.ui.control.uiaxes-properties.html>
- Colormaps: <https://www.mathworks.com/help/matlab/ref/colormap.html>

Debugging

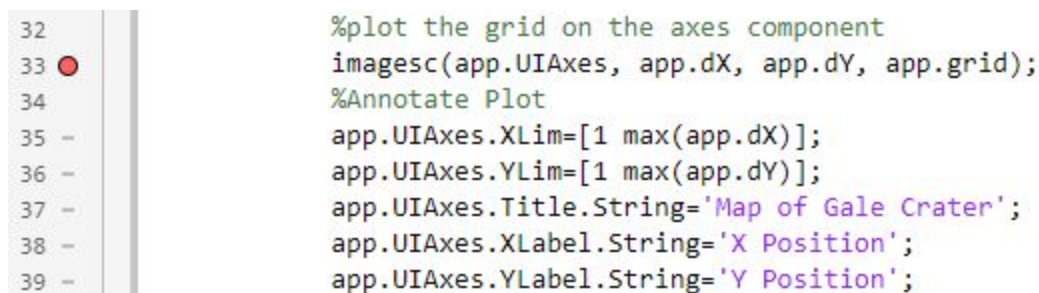
When you have an error in your app, you'll be brought to the code view of your app and an error message will be displayed on the line of the error, like in the image below.



In this example, 'jet' is an input that doesn't belong in the colorbar function. Instead, if I wanted to change the colormap to of my Mars map to jet or autumn I would need to add another line of code with the colormap function like below:

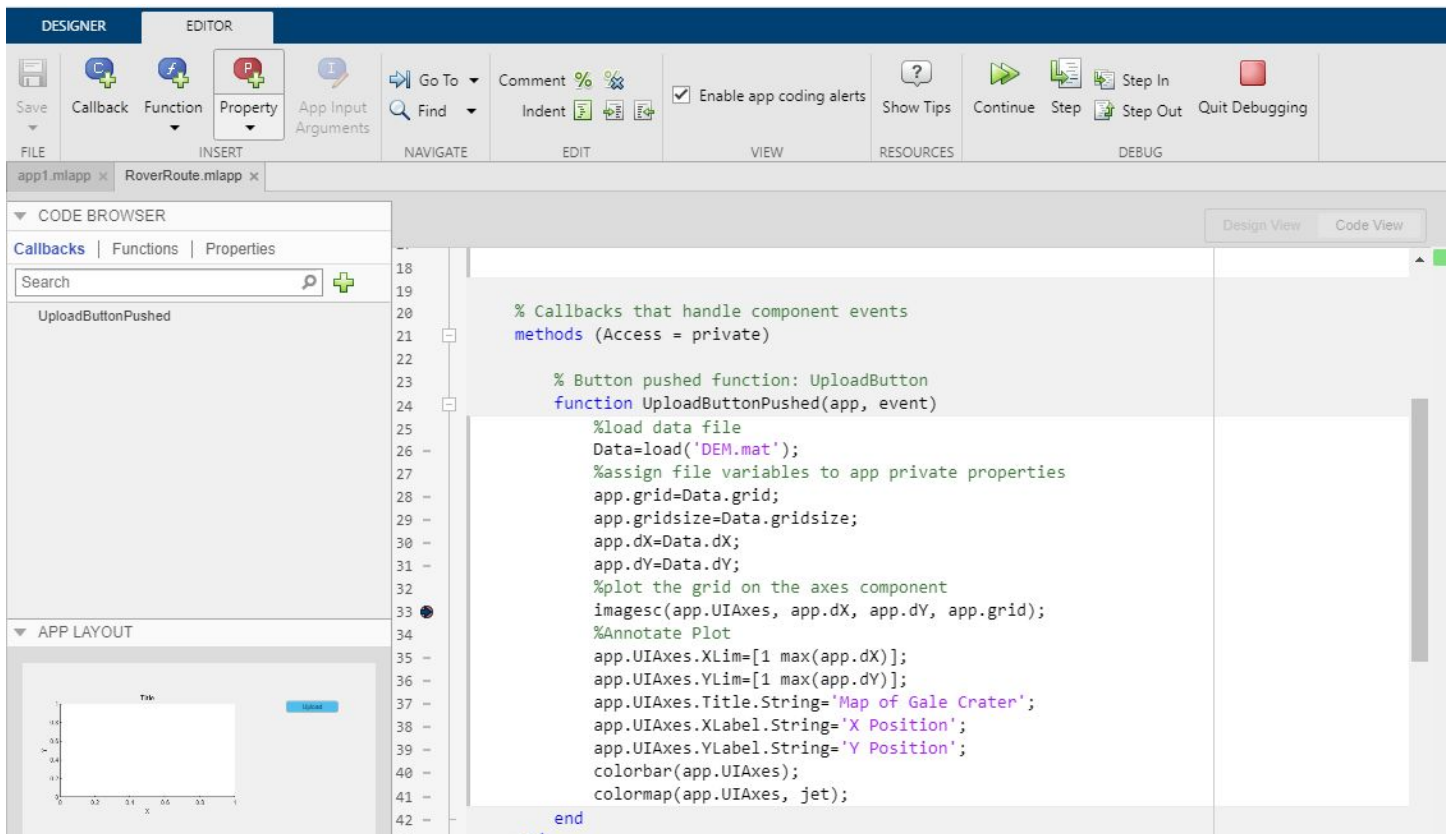
```
colorbar(app.UIAxes);
colormap(app.UIAxes, jet);
```

Another tool used in debugging is **breakpoints**. Breakpoints will pause your code before executing the line the breakpoint is on. Breakpoints are assigned by clicking on the dash near the line number on the left hand side of the code view. When a breakpoint is assigned, you'll see a red bubble next to that line number.



When the app is run with this breakpoint it'll stop and transport you back to the code view when the line is hit before executing the breakpoint line. In this example, when you click the upload button you'll invoke the breakpoint.

You'll notice in the image below that the breakpoint bubble is black and you have options in the top ribbon to continue, step, and quit debugging. If you hit continue, the rest of your code invoked by the user action (pressing the button) will execute. If you switch over to the Matlab window, you'll be able to see the variables/properties currently in your app and their current values.



Before hitting continue, you'll notice that in your workspace back in Matlab you have the variables "app", "event", and "Data". If you double click on "app" in your workspace you'll be able to look at all the properties and their values that are currently in your app.

To get rid of a breakpoint, click on it again in the code view.

HOME **PLOTS** **APPS** **EDITOR** **PUBLISH** **FILE VERSIONS** **VIEW**

New Save Find Files Go To Find Breakpoints Continue Step Step Out Function Call Stack: Quit Debugging

FILE NAVIGATE EDIT BREAKPOINTS DEBUG

MATLAB Drive

CURRENT FOLDER

Name

- Published (my site)
- 00.txt
- app1.mlapp
- DEM.mat
- RoverRoute.mlapp
- stat.m
- TestScript.m
- x.mat

WORKSPACE - ROVERROUTE.U...

| Name | Value |
|-------|------------|
| app | 1×1 Rover |
| Data | 1×1 struct |
| event | 1×1 Button |

TestScript.m

```

1 %Description: Reference Script
2 close all; %closes figure windows
3 clear all; %clears variables
4 clc; %clears the command window
5
6 %Semicolons determine whether something is displayed
7
8 %type "help ___" in the command window for more information
9
10 %-----
11 %Variable Declaration
12
13 %what not to do:
14 %var;
15 %int var;
16 %1var=5;
17
18 %what to do:
19 var=5;
20 Var=5; %variable names are case sensitive
21
22 %Workshop variables:
23
24 %-----
25
26 %-----
27
28 %-----
29
30 %-----
31
32
33 imagesc(app.UIAxes, app.dX, app.dY, app.grid);
K>>

```

MATLAB App

Y

1

0.8

0.6

0.4

0.2

0

| app x | | | |
|----------------|---------------|---------|-----------------|
| 1×1 RoverRoute | | | |
| Property | Value | Size | Class |
| UIFigure | 1×1 Figure | 1×1 | matlab.ui.Fi... |
| UploadButton | 1×1 Button | 1×1 | matlab.ui.c... |
| UIAxes | 1×1 UIAxes | 1×1 | matlab.ui.c... |
| grid | 385×768 in... | 385×768 | int16 |
| dX | 1×786 double | 1×786 | double |
| dY | 1×385 double | 1×385 | double |
| gridsize | 6945 | 1×1 | double |

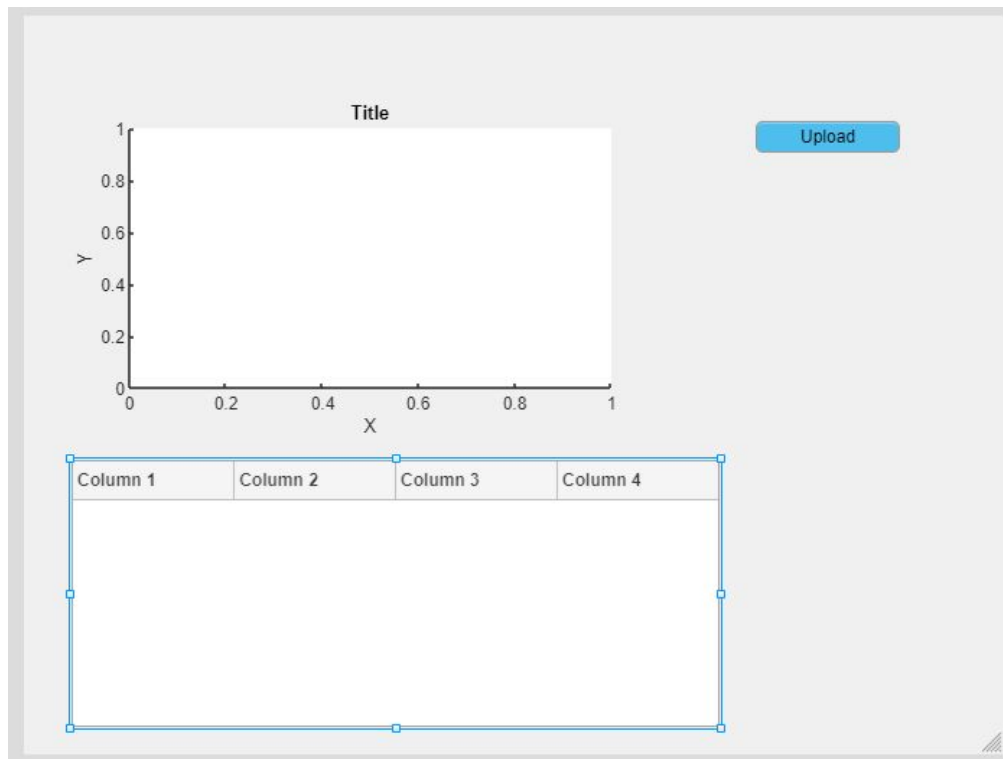
Section 2: App Building- Waypoint Table

Adding an Editable Table

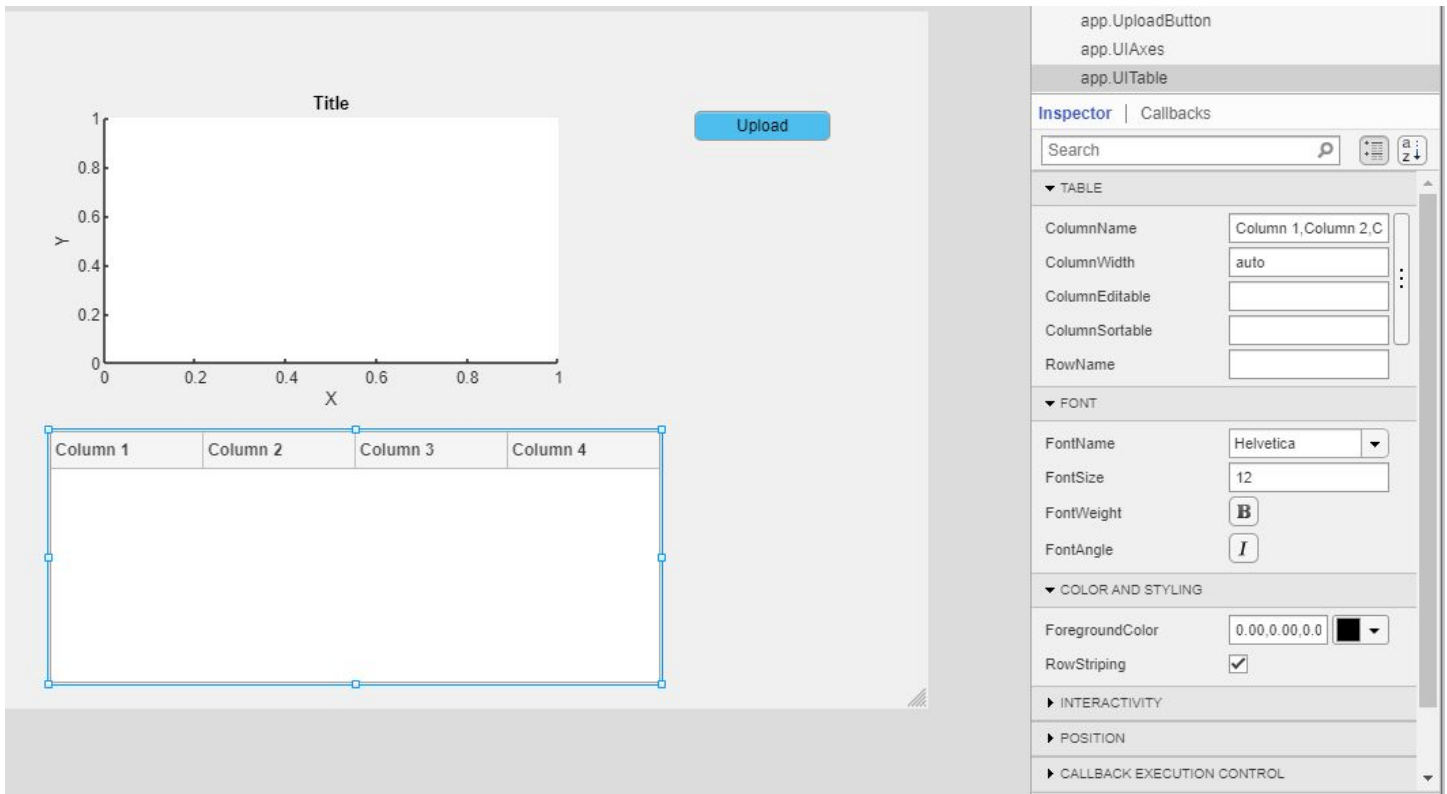
A table is a specific data type in Matlab where each variable in the table can have a different size and type. Indexing tables is very similar to indexing cell arrays.

In our app, the table is going to be where the user enters waypoints for their rover route. The user will be able to select the name, location, time, and if they want their rover to travel to that location. This information will then be used to plot the route on the map.

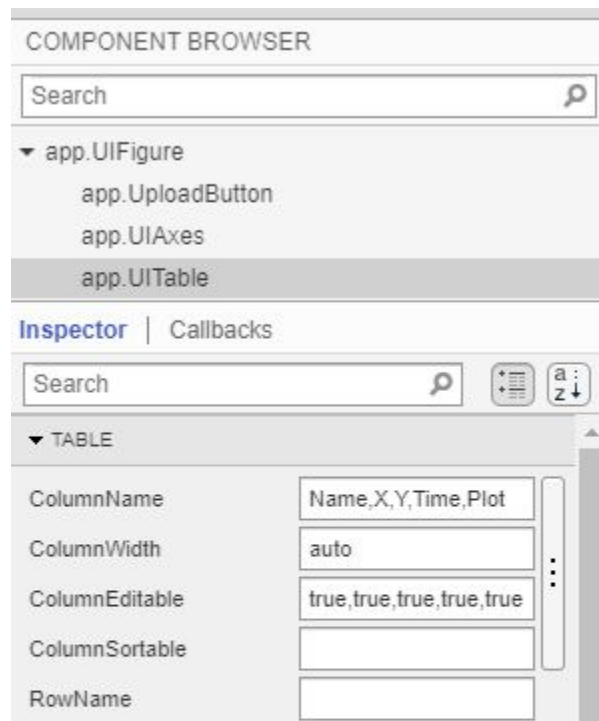
1. Add a table to the canvas from the Component Library.



2. Change the column names in the Property Inspector to be "Name, X, Y, Time, Plot" by editing the ColumnName field. Column names are separated by commas.

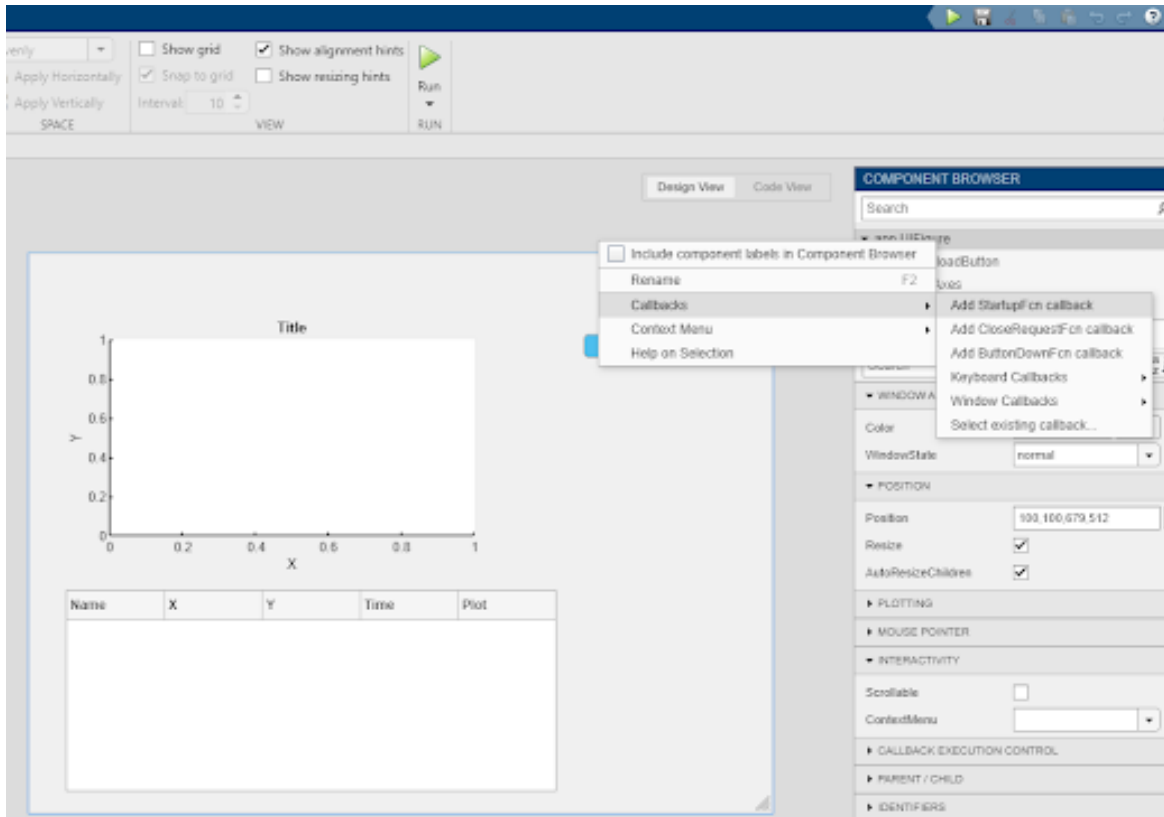


- Set the editability of each column in the table. This is done by typing “true” or “false” in the ColumnEditable field for each column. Since there are 5 columns, we will enter “true” 5 times separated by commas.



The **startup function** is what initializes your app. We’re going to initialize our table in the startup function.

4. Create a startup function callback by right clicking of app.UIFigure at the top of the Component Browser and then navigating to Callbacks and then to “Add StartupFcn callback”.



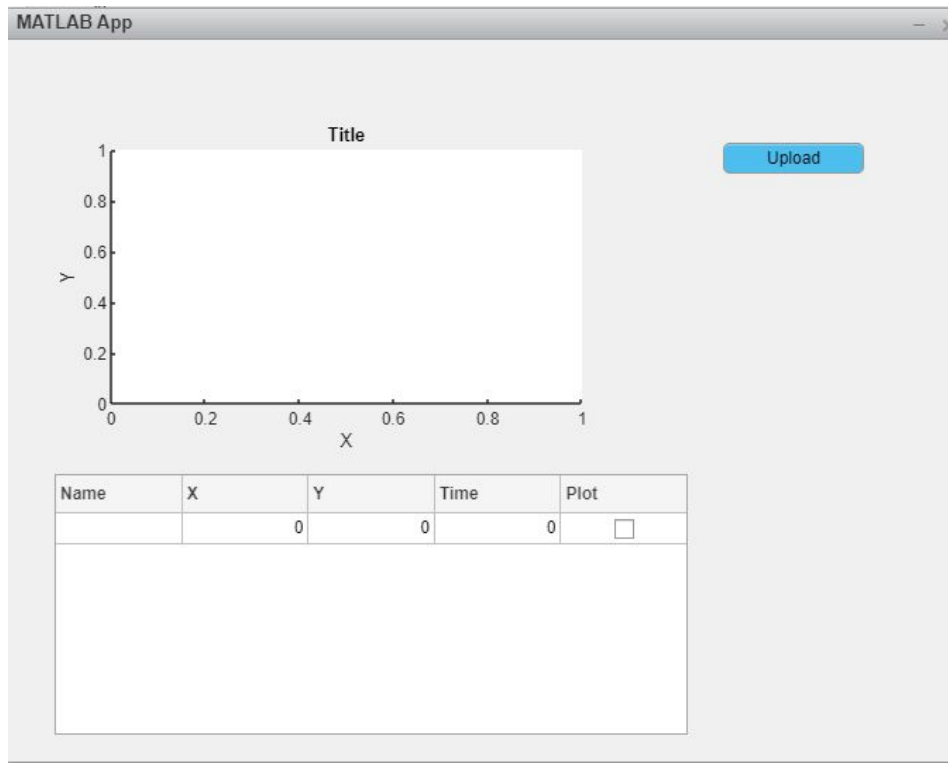
5. Add the following code to the startup function. This will initialize the first row of the Waypoints table and set the data types of the columns to be string, double, double, double, boolean.

```
% Code that executes after component creation
function startupFcn(app)
    app.UITable.Data={' ',0,0,0,false};
end
```

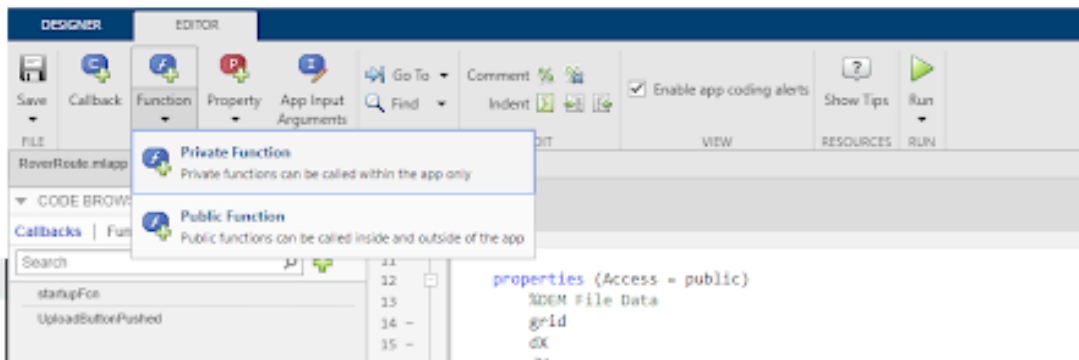
Make sure that there is a space between the single quotes for the first data element above.

6. Test the app by running it. The first row of the table should now have values in it (blank, 0, 0, 0, and checkbox).

We want the table to hold values when we edit it and we want to be able to use those table values to perform other operations in our app. Thus, we need to create some functions that will store the table values into properties whenever the table is updated.



7. Create a function by selecting the private function option from the top ribbon in the code view.



8. Name the function "updateTable." The "results=" portion of the default function name can be removed.

```

methods (Access = private)

    function updateTable(app)

    end

end

```

The update table function has a general form of pull the current table data, assign the table data to properties, and then push the table data back out.

9. Implement the following code in the updateTable function.

```
methods (Access = private)

function updateTable(app)
    %pull table data
    tableData=get(app.UITable,'Data')

    %assign data
    for ii=1:size(tableData,1)
        app.name(ii)=tableData{ii,1};
        app.X(ii)=tableData{ii,2};
        app.Y(ii)=tableData{ii,3};
        app.time(ii)=tableData{ii,4};
        app.plotCheck(ii)=tableData{ii,5};
    end

    %Push data
    app.UITable.Data=tableData;
end
end
```

10. Add the variables used for storing table data (app.name, app.X, app.Y, app.time, and app.plotCheck) to the public properties section you started previously.

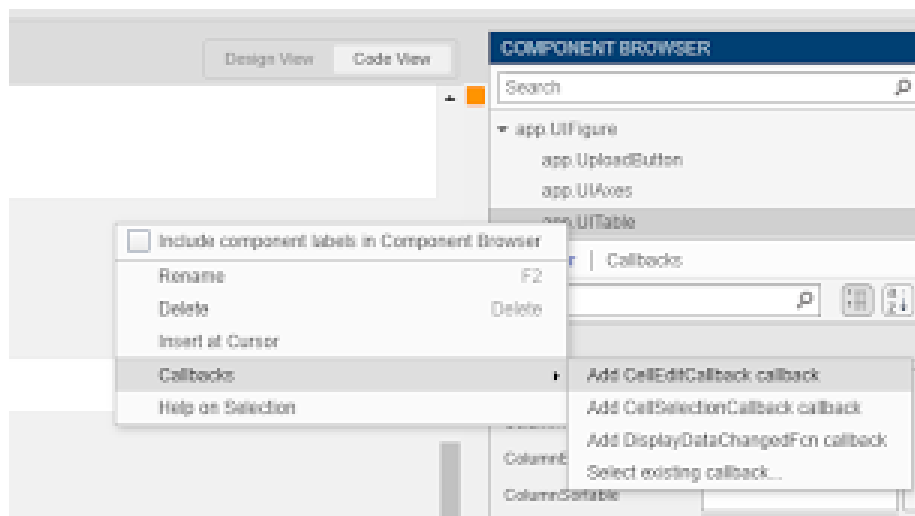
```
properties (Access = public)
    %DEM File Data
    grid
    dX
    dY
    gridsize

    %table variables
    name
    X
    Y
    time
    plotCheck
end
```

11. Add the updateTable function to the startup function after the table initialization. This will store the initial values in the properties.

```
function startupFcn(app)
    app.UITable.Data={' ',0,0,0,false};
    updateTable(app);
end
```

12. Add a CellEditCallback to the UITable component by right clicking on it either in the design view or in the Component Browser and then navigating to Callbacks and then to “Add CellEditCallback callback”.



Adding this callback will again transport you back to the code view and will start the callback for you. We will not need the two code lines provided.

```
% Cell edit callback: UITable
function UITableCellEdit(app, event)
    indices = event.Indices;
    newData = event.NewData;

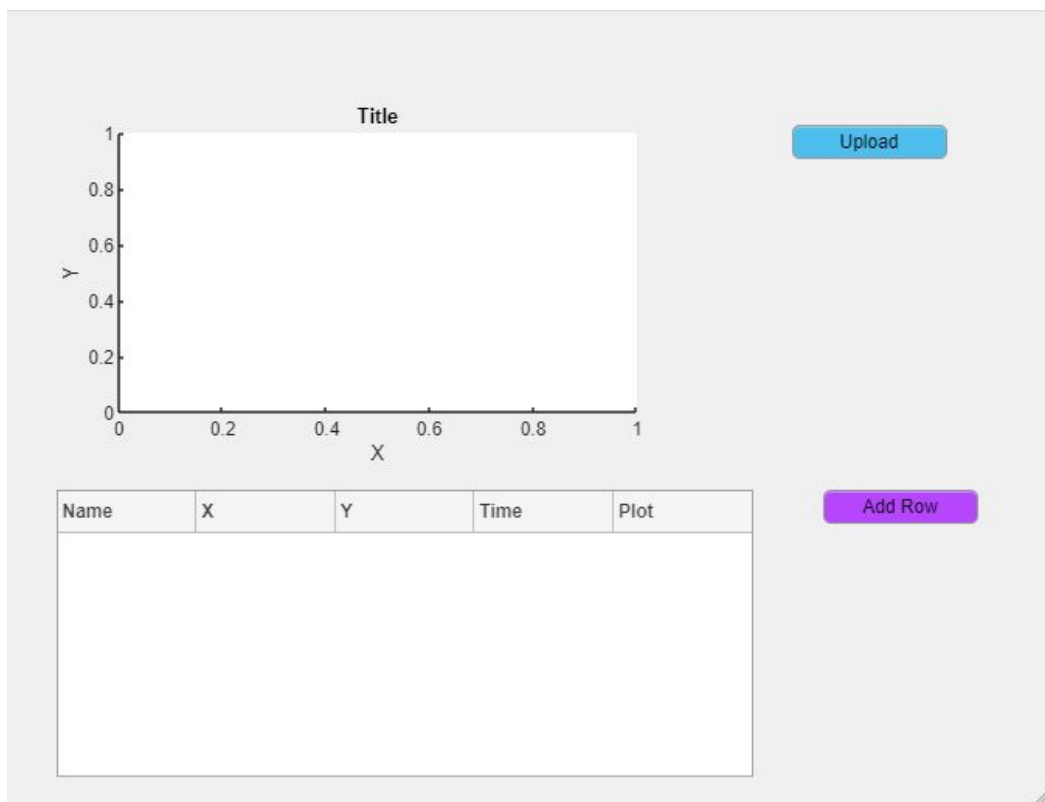
end
```

13. Remove the two provided code lines and add the updateTable function to the cell edit callback.

```
% Cell edit callback: UITable
function UITableCellEdit(app, event)
    updateTable(app);

end
```

14. Make an “Add Row” Button to the right of the table. Add a button pushed callback to the add row button and implement the following code.



```
% Button pushed function: AddRowButton
function AddRowButtonPushed(app, event)
    T=app.UITable.Data;
    numRows=size(T,1)+1;

    newData={' ',0,0,0,false};%initialize new row

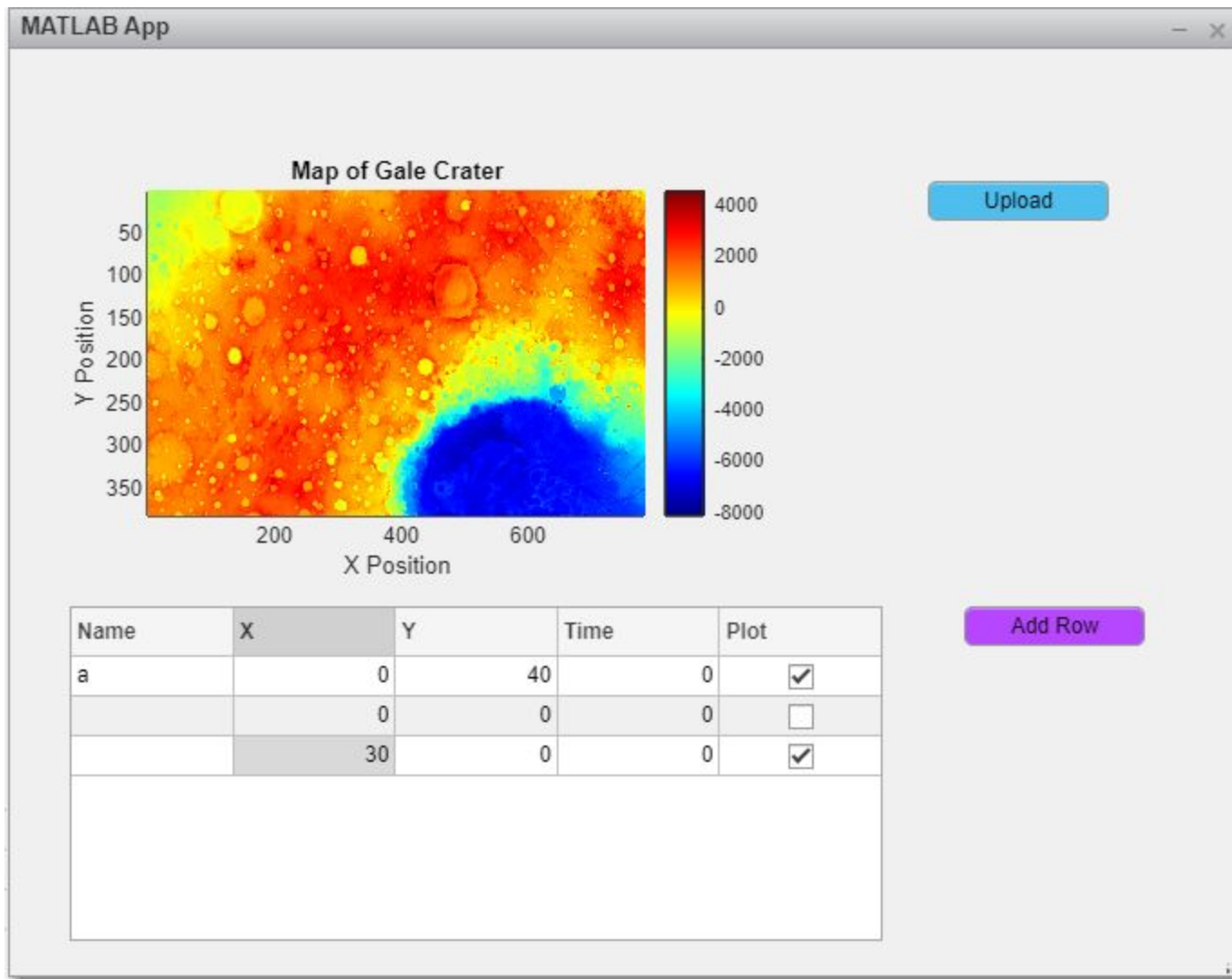
    app.name(numRow)=newData{1,1};
    app.X(numRow)=newData{1,2};
    app.Y(numRow)=newData{1,3};
    app.time(numRow)=newData{1,4};
    app.plotCheck(numRow)=newData{1,5};

    app.UITable.Data=[T;newData];%put the new data with the old data into the table

    updateTable(app);|
end
```

Make sure that the newData variable matches what was used to initialize the table in the startup function exactly.

15. Test your app by running it. Press the upload button and make sure your map is still plotting properly. Try to edit the first row of the table (change one of the 0's to a different number or check on of the checkboxes). Click the add row button and see if another row is added to the table.



Resources and Info on Tables:

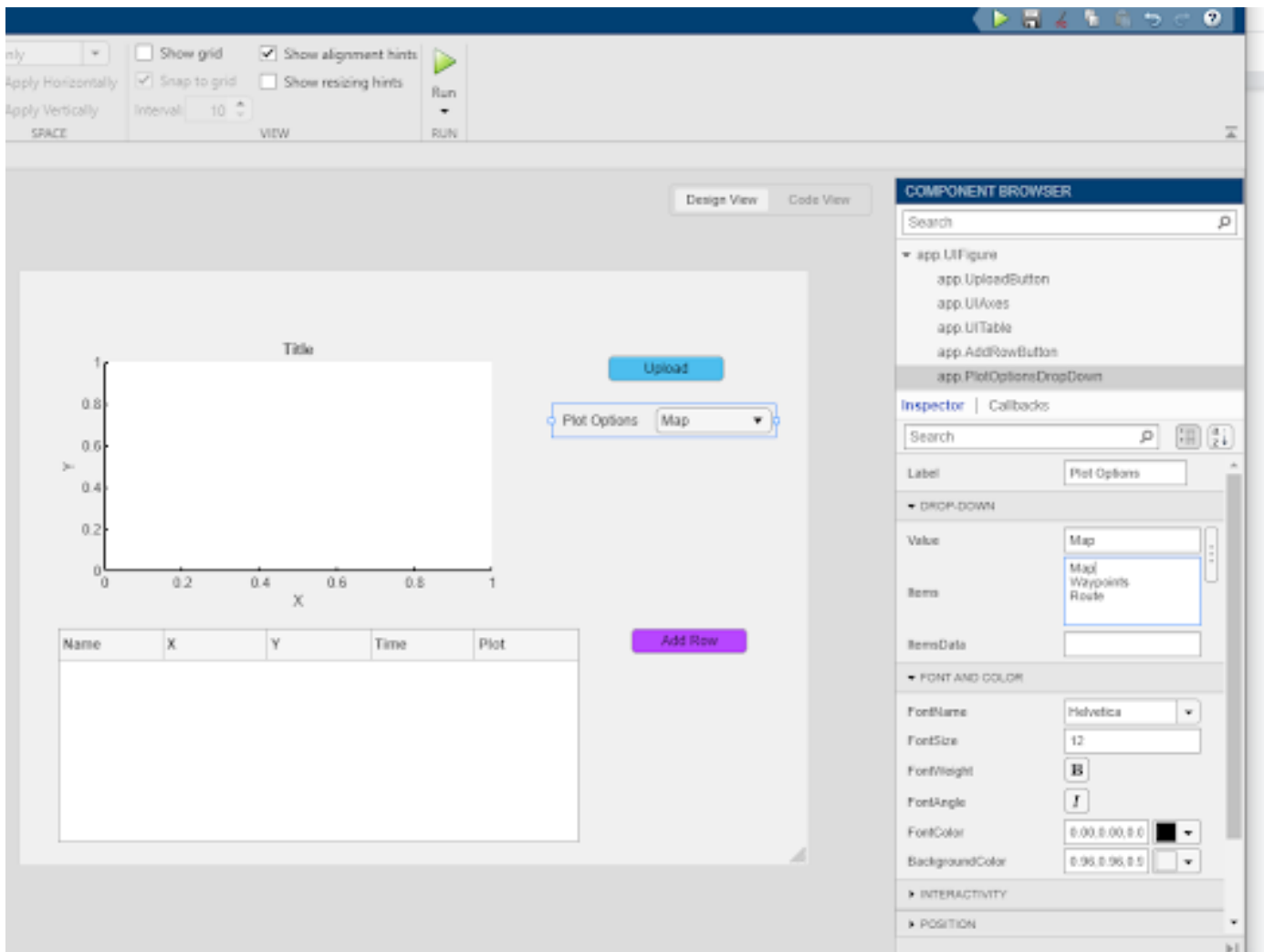
- Tables: <https://www.mathworks.com/help/matlab/tables.html>
- Tables in App Designer: https://www.mathworks.com/help/matlab/creating_guis/display-an-interactive-table-in-app-designer.html
- Cell arrays: <https://www.mathworks.com/help/matlab/cell-arrays.html>

Section 3: App Building- Plotting Waypoints on the Map

Creating a Dropdown Menu and Waypoint Plotting Functionality

We want the user to be able to select whether they just plot the map, plot the map with waypoints, or plot the map with the waypoints and route. We're going to provide these options in a dropdown menu.

1. Add a dropdown menu from the Component Library. In the Property Inspector change the Label to be "Plot Options" and change the Items to be "Map", "Waypoints", and "Route". Note that you need to press the return key and enter each option on a new line.



2. Add a Plot Button below the Add Row Button and create a button pushed callback for it.



In this call back we want to plot all the waypoints that have their checkbox checked in the table. We'll first need to replot the map and then use the "hold on" functionality in Matlab to plot the waypoints on top of the map. After the waypoints are plotted, we want to set the value of the dropdown menu to be "Waypoints" so the user knows what is being displayed.

3. Add the following code to the PlotButton Pushed callback:

```
% Button pushed function: PlotButton
function PlotButtonPushed(app, event)
    T=app.UITable.Data;

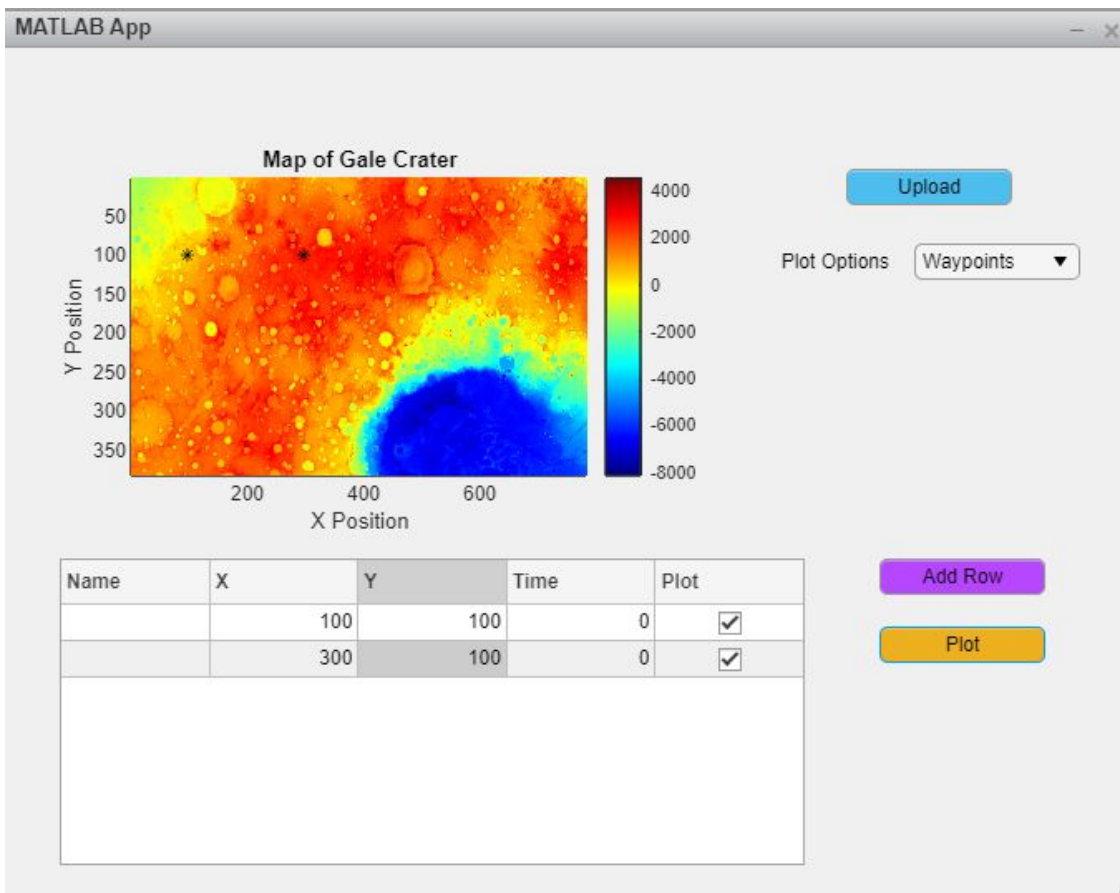
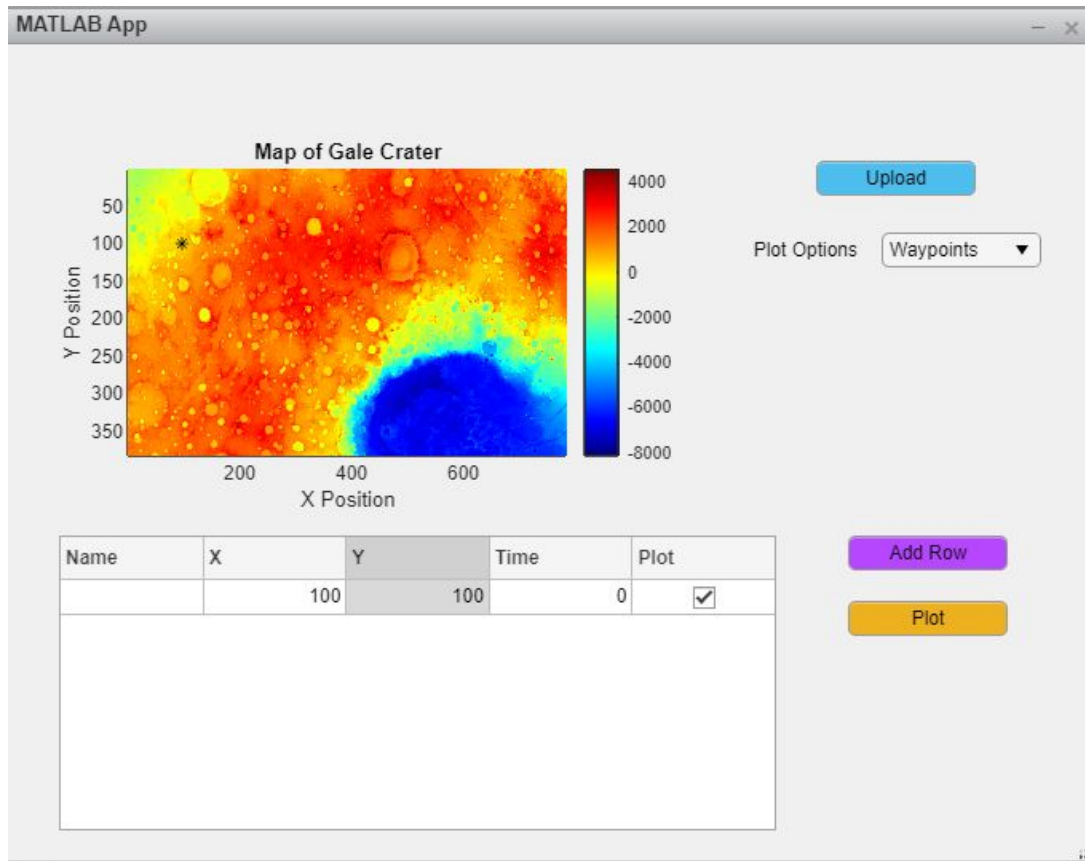
    %replot the map
    imagesc(app.UIAxes,app.dX,app.dY,app.grid);
    hold(app.UIAxes,'on');

    %plot waypoints for which the plot checkbox is checked
    for ii=1:size(T,1)
        if app.plotCheck(ii)==true
            plot(app.UIAxes, app.X(ii), app.Y(ii),'*k');
        end
    end
    |
    %Annotate Plot
    app.UIAxes.XLim=[1 max(app.dX)];
    app.UIAxes.YLim=[1 max(app.dY)];
    app.UIAxes.Title.String='Map of Gale Crater';
    app.UIAxes.XLabel.String='X Position';
    app.UIAxes.YLabel.String='Y Position';
    colorbar(app.UIAxes);
    %Set Drop Down Menu
    app.PlotOptionsDropDown.Value='Waypoints';
end
```

4. Test your app by running it.

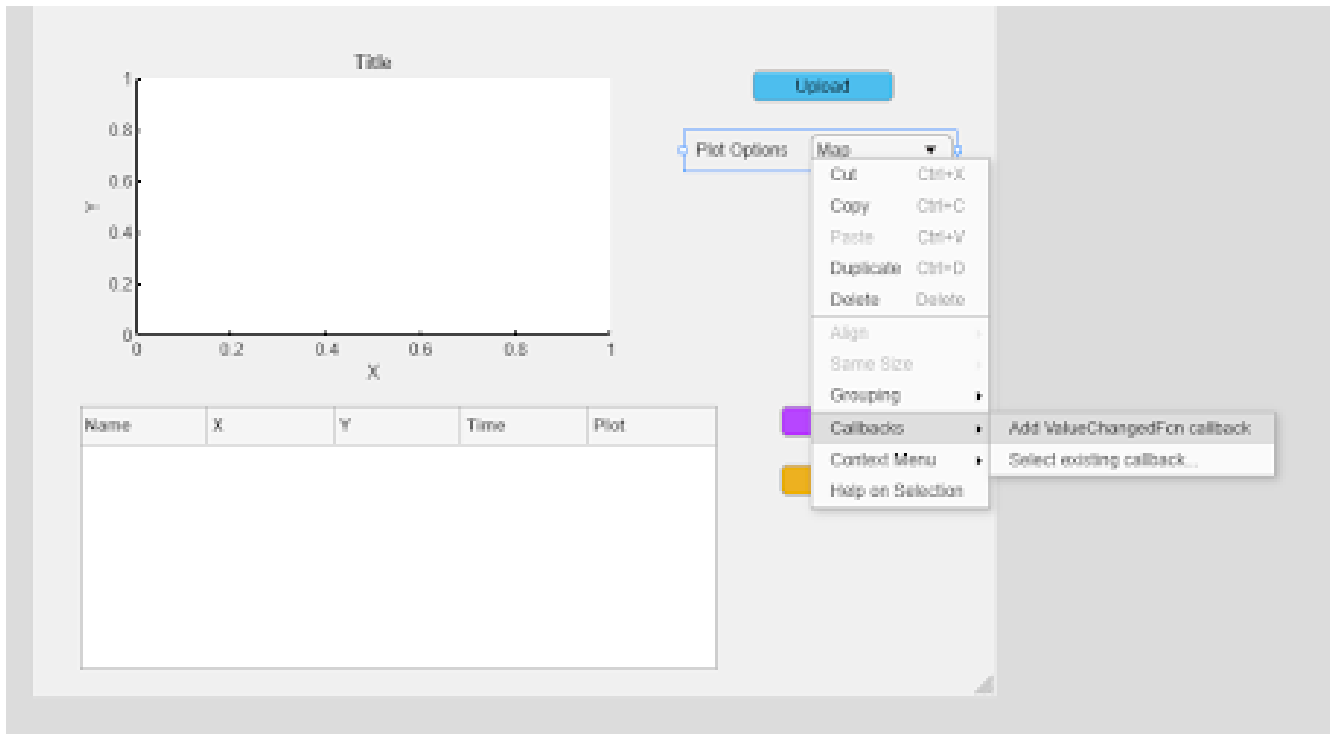
- a. Click the upload button and make sure the map plots currently.
 - i. Enter data into the first row of the Waypoints table. The location should be within the axes limits shown in the plot and the checkbox should be checked.
 - ii. Click the plot button and see if a black star appears on your map at the location you entered and if the dropdown menu is displaying the “Waypoints” option.
 - iii. Add a second row to the Waypoints table and enter data into it. Click the plot button again and see if your map looks as you would expect.

Note that at this point no functionality is coded for the dropdown menu (i.e., no callbacks have been implemented for it so nothing happens when you select from the menu.



Create a Callback for the Dropdown Menu

1. Create a callback for the dropdown menu. This will be a “ValueChangedFcn” callback meaning that when the value of the dropdown menu is changed, this callback code will be executed.



In dropdown menu callbacks we use switch cases to implement different functionality for different menu options.

2. Add the following code to the dropdown menu callback. It is very important that your case values (Map, Waypoints, Route) match the names you entered for the dropdown menus exactly. Matlab is case sensitive and therefore so are these values.

```
% Value changed function: PlotOptionsDropDown
function PlotOptionsDropDownValueChanged(app, event)
    value = app.PlotOptionsDropDown.Value;
    switch value
        case 'Map'

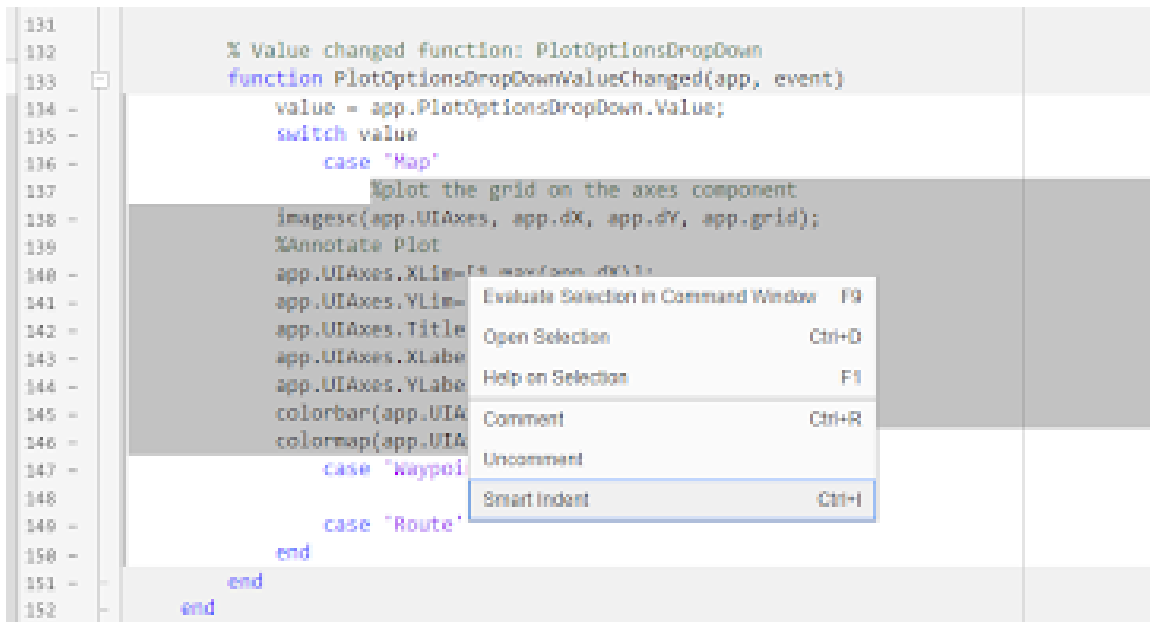
        case 'Waypoints'

        case 'Route'
    end
end
```

3. Add the code for the 'Map' case. This will be the same plotting code used in the UploadButtonPushed callback function and it can be copy and pasted here. If the copy and pasting doesn't indent everything properly you can highlight the code you wish to properly indent, then right click, and then select smart indent.

```
% Value changed function: PlotOptionsDropDown
function PlotOptionsDropDownValueChanged(app, event)
    value = app.PlotOptionsDropDown.Value;
    switch value
        case 'Map'
            %plot the grid on the axes component
            imagesc(app.UIAxes, app.dX, app.dY, app.grid);
            %Annotate Plot
            app.UIAxes.XLim=[1 max(app.dX)];
            app.UIAxes.YLim=[1 max(app.dY)];
            app.UIAxes.Title.String='Map of Gale Crater';
            app.UIAxes.XLabel.String='X Position';
            app.UIAxes.YLabel.String='Y Position';
            colorbar(app.UIAxes);
            colormap(app.UIAxes, jet);|
        case 'Waypoints'

        case 'Route'
    end
end
```



4. Add the code for the 'Waypoints' option. This is the same plotting code used in the PlotButtonPushed callback function and can be copy and pasted here.

```

% Value changed function: PlotOptionsDropDown
function PlotOptionsDropDownValueChanged(app, event)
    value = app.PlotOptionsDropDown.Value;
    switch value
        case 'Map'
            %plot the grid on the axes component
            imagesc(app.UIAxes, app.dX, app.dY, app.grid);
            %Annotate Plot
            app.UIAxes.XLim=[1 max(app.dX)];
            app.UIAxes.YLim=[1 max(app.dY)];
            app.UIAxes.Title.String='Map of Gale Crater';
            app.UIAxes.XLabel.String='X Position';
            app.UIAxes.YLabel.String='Y Position';
            colorbar(app.UIAxes);
            colormap(app.UIAxes, jet);
        case 'Waypoints'
            T=app.UITable.Data;
            imagesc(app.UIAxes, app.dX, app.dY, app.grid);
            hold(app.UIAxes, 'on');
            for ii=1:size(T,1)
                if app.plotCheck(ii)==true
                    plot(app.UIAxes, app.X(ii), app.Y(ii), '*k');
                end
            end
            %Annotate Plot
            app.UIAxes.XLim=[1 max(app.dX)];
            app.UIAxes.YLim=[1 max(app.dY)];
            app.UIAxes.Title.String='Map of Gale Crater';
            app.UIAxes.XLabel.String='X Position';
            app.UIAxes.YLabel.String='Y Position';
            colorbar(app.UIAxes);
        case 'Route'
            |
    end
end
end

```

For the route, we need to sort the waypoints based on their times and then tell our rover to travel to the points in chronological order. We also want to indicate what the start point is and what the end point is.

5. Add in the code for the 'Route' option.

```

case 'Route'
    %Pull out waypoints to be plotted
    T=app.UITable.Data;
    T1=sortrows(T,4);
    cnt=1;
    for ii=1:size(T,1);
        if T1{ii,5}==1
            app.RouteX(cnt)=T1{ii,2};
            app.RouteY(cnt)=T1{ii,3};
            cnt=cnt+1;
        end
    end
    %Plot the map
    imagesc(app.UIAxes,app.dX,app.dY,app.grid);
    hold(app.UIAxes,'on');
    plot(app.UIAxes,app.RouteX,app.RouteY,'k');
    plot(app.UIAxes,app.RouteX(1),app.RouteY(1),'*k');
    text(app.UIAxes,app.RouteX(1),app.RouteY(1)-2,'start');
    plot(app.UIAxes,app.RouteX(end),app.RouteY(end),'*k');
    text(app.UIAxes,app.RouteX(end),app.RouteY(end)-2,'stop');
    if size(app.RouteX,2)>1
        app.RouteExists=1;
    end
end

```

6. Add the route properties (RouteX, RouteY, and RouteExists) to the properties list.

```

properties (Access = public)
    %DEM File Data
    grid
    dX
    dY
    gridsize

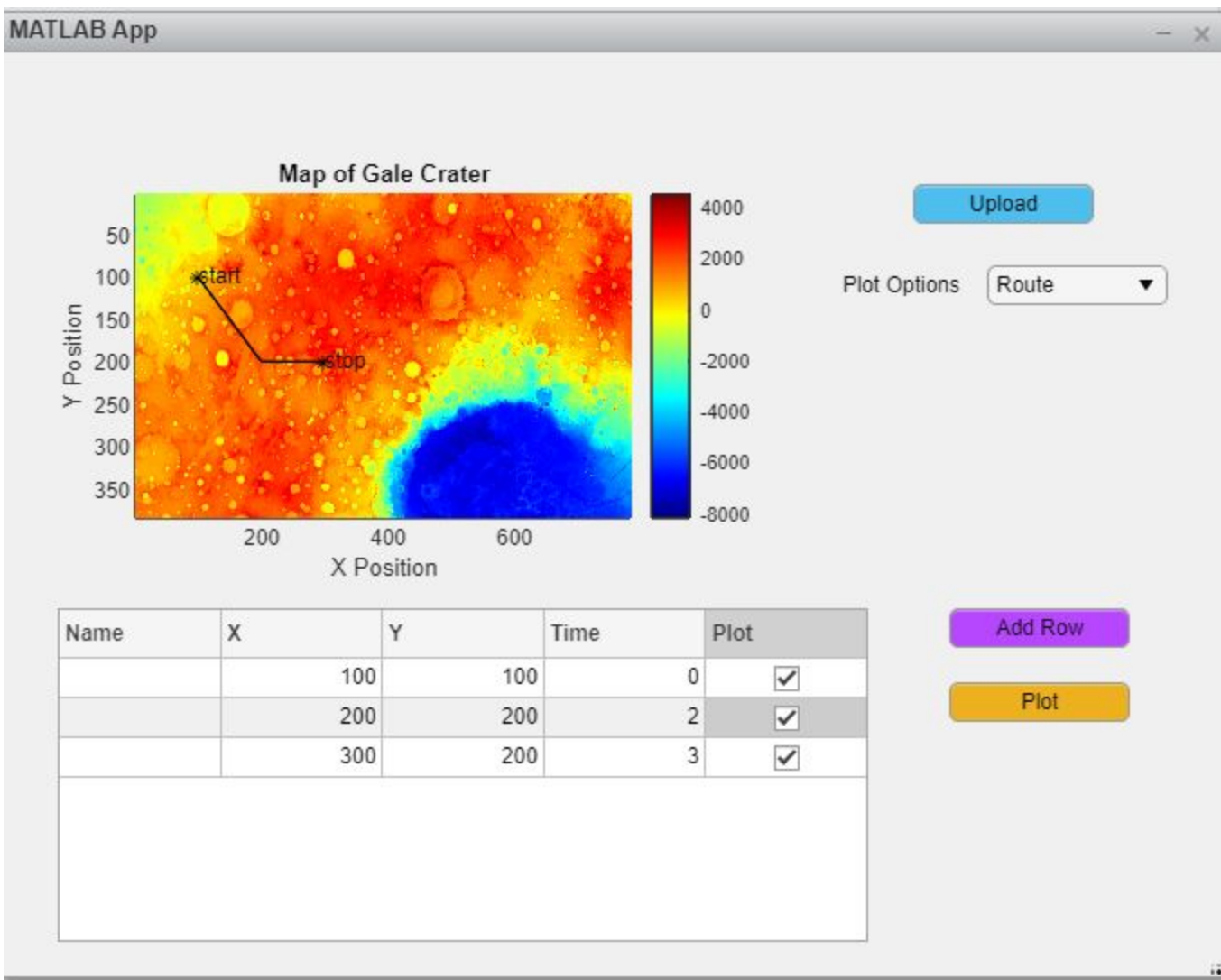
    %table variables
    name
    X
    Y
    time
    plotCheck

    %Routes
    RouteX
    RouteY
    RouteExists
end

```

7. Test your app:

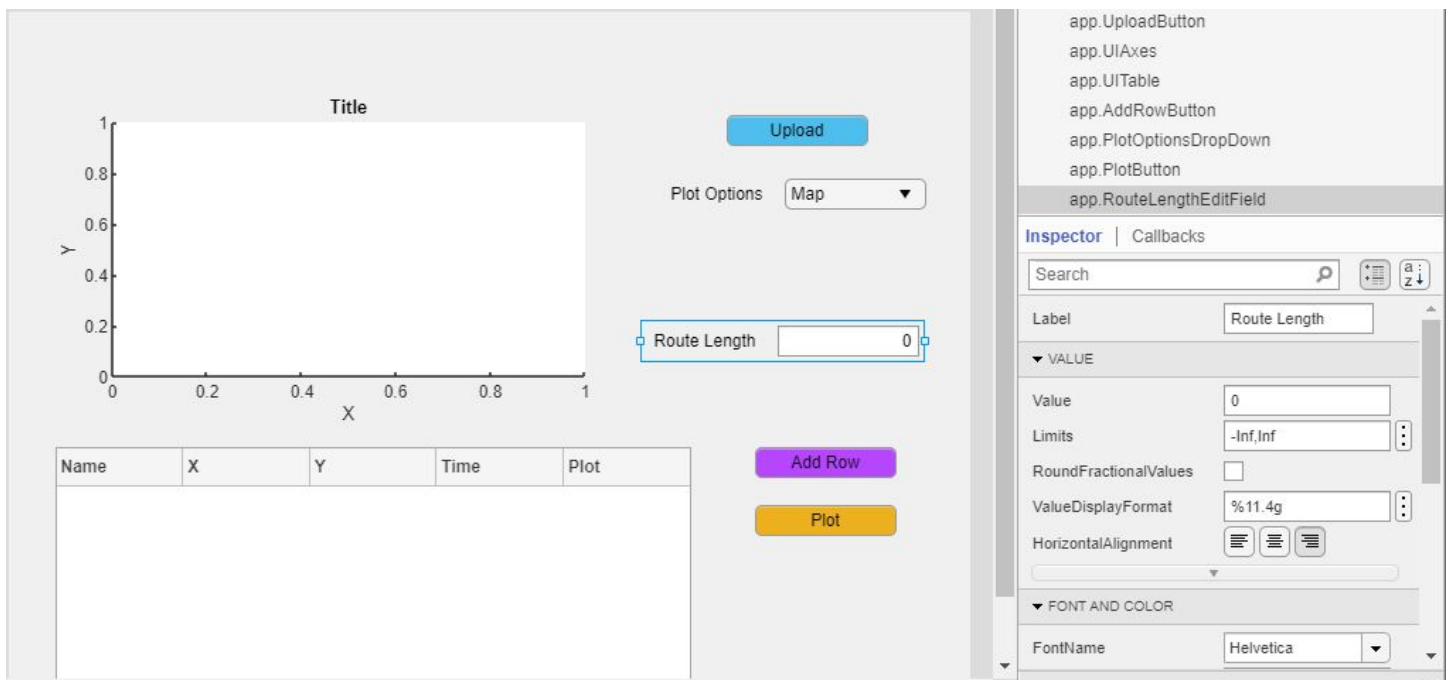
- Press the upload button and see if your map plots.
- Add 2 additional rows and fill in the waypoints table for all 3 rows.
- Click the Plot button and make sure your defined waypoints plot.
- Select the Route option from the dropdown menu. Check that the route plots with start and stop labels.
- Select Map from the drop down menu and check that the waypoints and route are removed from the map.
- Select the Waypoints or Route option from the drop down menu and make sure they return to the map.



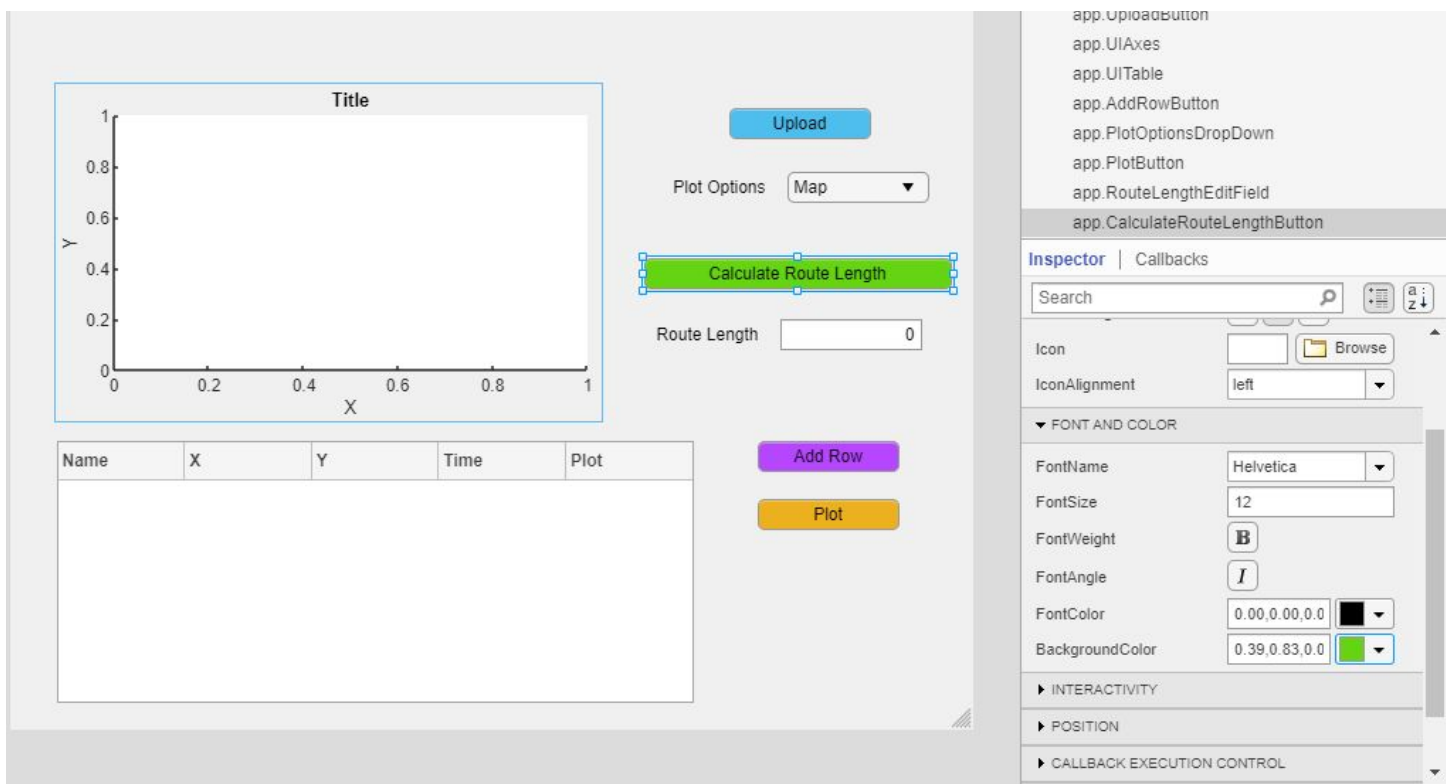
Section 4: App Building- Calculating Route Length and Finishing Touches

Calculating and Displaying the Route Length

1. Add an Edit Field (numeric) to the app and change its label to Route Length in the Property Inspector.



2. Add a button for calculating route length and add a callback for this button.

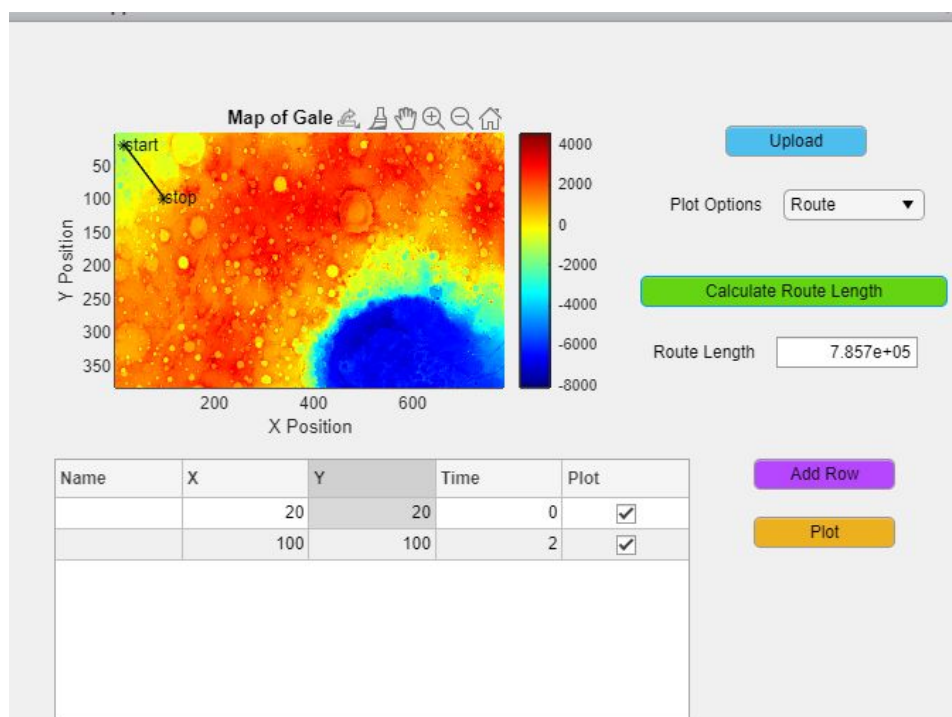


If the route exists (i.e., there are more than 2 waypoints and the app.RouteExists flag has been set to 1) then we want to calculate it's length. We're going to calculate the route length as the crow flies one leg at a time using the 2d distance equation and then set this result of this calculation as the value of the edit field we just created.

3. Add the following code to the CalculateRouteLengthButtonPushed callback.

```
% Button pushed function: CalculateRouteLengthButton
function CalculateRouteLengthButtonPushed(app, event)
    if app.RouteExists==1
        dist=0;
        for ii=1:length(app.RouteX)-1
            d1=sqrt((app.RouteX(ii+1)-app.RouteX(ii)).^2 + (app.RouteY(ii+1)-app.RouteY(ii)).^2);
            d1=d1*app.gridsize;
            dist=dist+d1;
        end
        app.RouteLengthEditField.Value=dist;
    else
        app.RouteLengthEditField.Value=0;
    end
end
```

4. Test your app.
 - a. Press the upload button and see if your map plots.
 - b. Add an additional rows and fill in the waypoints table for both rows.
 - c. Click the Plot button and make sure your defined waypoints plot.
 - d. Select the Route option from the dropdown menu. Check that the route plots with start and stop labels.
 - e. Click the Calculate Route Length Button. The displayed route length will be in meters and should be large due to each grid space being 6945 m large.



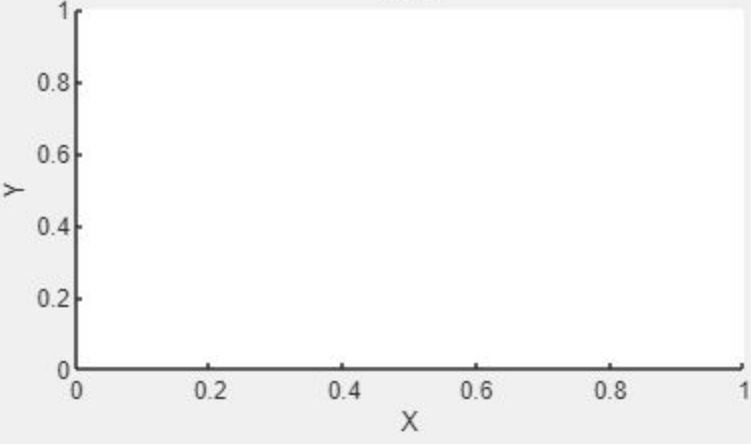
Finishing Touches

You can also add labels to your app.

1. Drag and drop some labels from the Component Library. Edit them with the Property Inspector.

Rover Route Planner

Title



Y

X

Waypoints

| Name | X | Y | Time | Plot |
|------|---|---|------|------|
|------|---|---|------|------|

Upload

Plot Options Map ▼

Calculate Route Length

Route Length

Add Row

Plot

Additional Remarks/Resources

- There are other components that we didn't use in this tutorial
- Can use Matlab toolboxes in data manipulation in the app
 - Not all the toolbox plotting options work with the UIAxes component
- Can use the App Testing Framework to do unit testing and automate scenarios
 - <https://www.mathworks.com/help/matlab/app-testing-framework.html>

Additional Tutorials:

- https://www.mathworks.com/help/matlab/creating_guis/create-a-simple-app-or-gui-using-app-designer.html
- <https://www.mathworks.com/help/matlab/app-designer.html>
- <https://lifeinplaintextblog.wordpress.com/matlab-app-designer-tutorial-1-english/>