Assignment Cover Sheet

Neuroimaging for Research / Imaging Sc/Dip/Cert/PPD

Please include your matriculation number at the beginning of the **file name** for individual submissions, e.g. s098765_YOURFILENAME.doc OR your group name for group submissions, e.g. Group1_YOURFILENAME.

To ensure anonymous marking, do not include your name anywhere in the file name or within the file itself.

Your document MUST be in Rich Text (.rtf), Text (.txt) or Word (.doc) format.

Please enter your details and assessment details below and include this at the beginning of the document you are submitting (do not submit this as a separate file). Assignments with incomplete data will not be accepted.

Date: 23rd November 2023

Student Matriculation Number (for individual submissions) [redacted]

Group Name (for group submissions):

Course: Practical Image Analysis 1

Assignment Name as per instructions: PART 4 – GUI creation

OWN WORK DECLARATION

Making a submission using this form signifies that you have read, understood and agree with the statements below.

1. I confirm that all this work is my own except where indicated, and that:
2. I have read and understood the Plagiarism Rules & Regulations in the course sections and Programme Handbooks;
3. I have clearly referenced / listed all sources as appropriate;
4. I have referenced and appropriately indicated all quoted text of more than three words (from books, web, etc);
5. I have given the sources of all pictures, data etc that are not my own;
6. I have not made any use of the essay(s) of any other student(s) either past or present;
7. I have not submitted for assessment work previously submitted for any other course,
8. degree or qualification;
9. I have not incorporated any work from or used the help of any external professional
10. agencies other than extracts from attributed sources;
11. I have acknowledged in appropriate places any help that I have received from others (e.g. fellow students, teachers in schools, external sources);
12. I have complied with any other plagiarism criteria specified in the course and Programme handbooks;
13. I understand that any false claim for any of the above will mean that the relevant piece of work will be penalised in accordance with the University regulations;
14. I hereby grant the University of Edinburgh, SFC, HEFCE and TurnitinUK a non-exclusive licence to make an electronic copy of the work and make it available for a assessment and archiving purposes.

PART 4 – GUI creation

Part 4.1

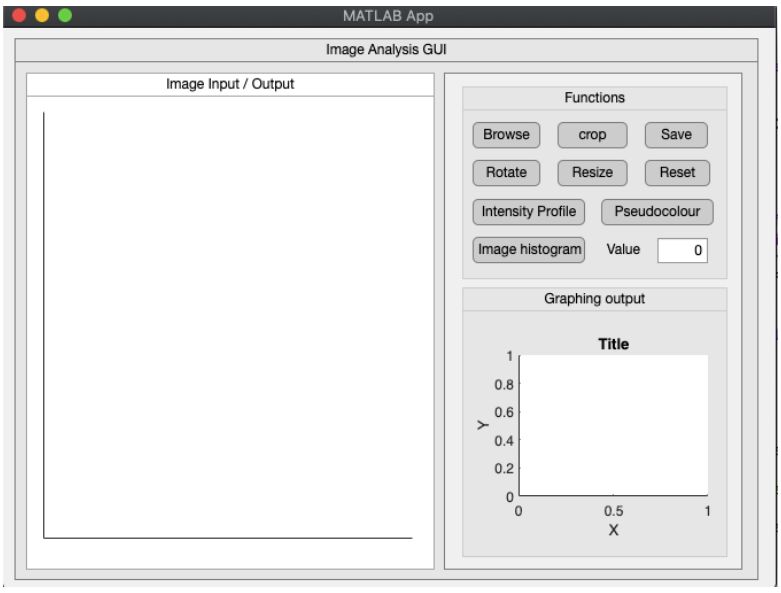
Introduction

The Image Analysis GUI is a MATLAB application designed for basic image processing and analysis, supporting both grayscale images of dimension M*N and RGB images of dimension M*N*3. This graphical user interface (GUI) offers a range of tools to load images, perform various image manipulations, and visualize analysis results in real-time. This report provides an overview of the GUI's enhanced functionality, emphasizing its adaptability to both grayscale and RGB images, along with the explanation of the operations in GUI.

Components and Layout

The GUI's layout comprises components organized for user-friendly interaction:

1. **Image Input/Output Panel:** Positioned on the left side, this panel displays the original image and serves as the primary area for image input and output. The panel contains a UI axes – app.UIAxes where images are displayed.
2. **Functions Panel:** Located on the right side, this panel contains buttons triggering specific image processing functions. Functions include browsing for an image, cropping, rotating, resizing, resetting, generating intensity profiles, pseudocolouring, displaying histograms, and saving images or graphs.
3. **Graphing Output Panel:** Below the Functions Panel, this panel dynamically displays graphical outputs, such as intensity profiles or histograms, based on the chosen functions. It adapts seamlessly to the nature of the processed image, whether grayscale or RGB. The panel contains a UI axes – app.UIAxes2 where graphs are displayed.



Operations/Functions

Browse: This opens a file selection dialog using *uigetfile*. The selected image file is read using *imread*, and if successful, it is displayed on a specified UI axes (*app.UIAxes*). The function includes error handling to manage cases where the user cancels the file selection or an error occurs during image reading or display. The selected image is stored in a global variable named *img*, making it accessible across different parts of the application.

Code:

```
        % Button pushed function: BrowseButton
        function BrowseButtonPushed(app, event)
            % Declare a global variable to store the selected image
            global img;

            % Open a file selection dialog
            [filename, pathname] = uigetfile('*.*', 'Pick an Image');

            % Check if the user cancelled the operation
            if isequal(filename, 0) || isequal(pathname, 0)
                % User cancelled the operation
                title(app.UIAxes, 'No image selected, selection cancelled');
                disp('Image selection cancelled.');
                return;
            end

            % Try to read the selected image
            try
                % Read the image using imread and create the full file path
                img = imread(fullfile(pathname, filename));

                % Display the image on the UIAxes
                imshow(img, 'Parent', app.UIAxes);

                % Set the title of the UIAxes to indicate it's the original
image
                title(app.UIAxes, 'Original Image');
            catch ME
                % Handle errors during image reading or display
                title(app.UIAxes, 'No image displayed', ME.message);
                disp(['Error reading image: ', ME.message]);
                return;
            end
        end
```

Reset: This serves to restore the GUI to its initial state. Upon activation, it first checks if a global variable named *img* is not empty, indicating the presence of a previously loaded image. In such a case, it displays the original image on the primary UI axes (*app.UIAxes*) and clears the contents of a secondary UI axes (*app.UIAxes2*) or the graph section. The function further resets the title of ***UIAxes2*** to an empty string and sets the value of a designated edit field (*app.ValueEditField*) to 0. If no image is loaded (i.e., when the global variable *img* is empty), the function outputs an error message to the console and sets the title of *UIAxes* to 'No Image loaded'.

```
  % Button pushed function: ResetButton
        function ResetButtonPushed(app, event)
```

```
        global img;

         % Check if the global variable img is not empty
         if ~isempty(img)
             % Display the original image in UIAxes
             imshow(img, 'Parent', app.UIAxes);
             title(app.UIAxes, 'Original Image');

             % Clear the second axes and reset the title
             cla(app.UIAxes2);
             title(app.UIAxes2, '');

             % Reset the scale factor to 0
             app.ValueEditField.Value = 0;
         else
             % If img is empty, display an error message
             disp('No image loaded.');
             title(app.UIAxes, 'No Image loaded');
         end
    end
```

Crop: This function allows the user to interactively crop an image displayed on a primary UI axes (*app.UIAxes).* It first checks if an image is loaded by searching for an image object within the UI axes. If no image is found, it prints an error message to the console and sets the UI axes title to 'No Image loaded.' If an image is present, the function extracts the image data, allowing the user to draw a rectangle to define the cropping region interactively. The cropped image is then displayed on the UI axes with the title 'Cropped Image.' Additionally, the function closes the figure opened by the interactive cropping operation. Error handling is implemented to catch and display any errors that may occur during the image cropping process.

Code:

```
  % Button pushed function: cropButton
    function cropButtonPushed(app, event)
         imgObj = findobj(app.UIAxes, 'Type', 'Image');
         if isempty(imgObj)
             disp('No image loaded.');
             title(app.UIAxes, 'No Image loaded');
             return; % Exit the function if no image is loaded
         end

         try
             % Get the image data from the Image object
             imageData = get(imgObj, 'CData');

             % Interactive crop: Let the user draw a rectangle
             croppedImage = imcrop(imageData);

             % Display the cropped image
             imshow(croppedImage, 'Parent', app.UIAxes);
             title(app.UIAxes, 'Cropped Image');

             % Close the figure opened by imcrop
             close gcf; % gcf refers to the current figure

         catch ME
             disp(['Error  image: ', ME.message]);
             return;
```

```
        end
    end
```

Pseudocolor: This pseudocolourizes an image displayed on the primary UI axes (*app.UIAxes*). The function first checks if an image is loaded by searching for an image object within the UI axes. If no image is found, it prints an error message to the console and sets the UI axes title to 'No Image loaded.' If an image is present, the function proceeds to convert the image to grayscale using **im2gray**. Subsequently, it pseudocolourizes the grayscale image using the current colormap, creating an RGB image. The pseudocoloured image is then displayed on the UI axes with the title 'Pseudocoloured Image.' Additionally, the function closes the figure opened during the pseudocolourization process. Error handling is implemented to catch and display any errors that may occur during the pseudocolourization.

Code:

```
% Button pushed function: PseudocolourButton
    function PseudocolourButtonPushed(app, event)
        imgObj = findobj(app.UIAxes, 'Type', 'Image');
        if isempty(imgObj)
            disp('No image loaded.');
            title(app.UIAxes, 'No Image loaded');
            return; % Exit the function if no image is loaded
        end

        try
            % Get the image data from the Image object
            imageData = get(imgObj, 'CData');

            % Convert the image to grayscale
            gray_image = im2gray(imageData);

            % Pseudocolour the image using the current colormap
            indexed_image = gray2ind(gray_image, 256);

            % Use the scaled indexed image directly with ind2rgb
            rgb_image = ind2rgb(indexed_image, colormap);

            % Display the cropped image
            imshow(rgb_image,'Parent', app.UIAxes);
            title(app.UIAxes, 'Pseudocoloured Image');

            % Close the figure opened by pseudocolour
            close gcf; % gcf refers to the current figure
        catch ME
            disp(['Error  image: ', ME.message]);
            return;
        end
    end
```

Rotate: This rotates a loaded image displayed on the primary UI axes (app.UIAxes). It first checks if an image is loaded by searching for an image object within the UI axes. If no image is found, it prints an error message to the console and sets the UI axes title to 'No Image loaded.' If an image is present, the function proceeds to retrieve the rotation angle from a specified edit field (app.ValueEditField). It validates the rotation angle, ensuring it is a numeric, scalar value. In case of an invalid angle, an error dialog is displayed. It then retrieves the image data, rotates the image by the specified angle using imrotate, and displays the rotated image on the UI axes. The UI axes title is updated to indicate the degree of rotation. Error handling is

implemented to address potential issues during the rotation process, with error messages displayed in the console and the UI axes title set to 'Error occurred' in case of an error.

Code:

```
% Button pushed function: RotateButton
        function RotateButtonPushed(app, event)
            imgObj = findobj(app.UIAxes, 'Type', 'Image');
            if isempty(imgObj)
                disp('No image loaded.');
                title(app.UIAxes, 'No Image loaded');
                return; % Exit the function if no image is loaded
            end

            try
                % Get the rotation angle from the ValueEditField
                rotationAngle = app.ValueEditField.Value;

                % Validate the rotation angle
                if ~isnumeric(rotationAngle) || ~isscalar(rotationAngle) ||
isnan(rotationAngle)
                    errordlg('Invalid rotation angle. Please enter a numeric
value (positive or negative) in the value field');
                end

                % Get the image data from the Image object
                imageData = get(imgObj, 'CData');

                % Rotate the image
                rotatedImage = imrotate(imageData, rotationAngle);

                % Display the rotated image in UIAxes
                imshow(rotatedImage, 'Parent', app.UIAxes);
                title(app.UIAxes, ['Rotated Image by ',
num2str(rotationAngle), ' degrees']);

            catch ME
                % Handle errors
                disp(['Error: ', ME.message]);
                title(app.UIAxes, 'Error occurred');
            end
        end
```

Image Histogram: This generates and displays the histogram of a loaded image on a secondary UI axes (app.UIAxes2). It first locates the image object in the primary UI axes (app.UIAxes). If no image is found, it prints an error message to the console and sets the UI axes title to 'No Image loaded.' If an image is present, the function retrieves the image data and uses the histogram function to plot the pixel value distribution on secondary UI axes (app.UIAxes2). The secondary UI axes title is set to 'Image Histogram,' and axis labels are added. Error handling is implemented to manage potential issues during histogram generation, with error messages displayed in the console and the secondary UI axes title set to 'Error occurred' in case of an error.

Code:

```
% Button pushed function: ImagehistogramButton
        function ImagehistogramButtonPushed(app, event)
```

```
            % Find the image object in UIAxes
            imgObj = findobj(app.UIAxes, 'Type', 'Image');

             % Check if there is no image loaded
            if isempty(imgObj)
                disp('No image loaded.');
                title(app.UIAxes, 'No Image loaded');
                return; % Exit the function if no image is loaded
            end

            try
                % Get the image data from the Image object
                imageData = get(imgObj, 'CData');

                % Display the histogram of the current image UIAxes2 using
histogram
                histogram(app.UIAxes2, imageData(:), 'BinMethod',
'auto','EdgeColor','none','LineWidth', 0.5);

                % Set the title and labels
                title(app.UIAxes2, 'Image Histogram');
                xlabel(app.UIAxes2, 'Pixel Value');
                ylabel(app.UIAxes2, 'Frequency');
            catch ME
                % Handle errors
                disp(['Error: ', ME.message]);
                title(app.UIAxes2, 'Error occurred');
            end
        end
```

Resize: This facilitates the resizing of a loaded image displayed on app.UIAxes. It begins by finding the image object in the primary UI axes (app.UIAxes). If no image is found, it prints an error message to the console and sets the UI axes title to 'No Image loaded.' If an image is present, the function retrieves the resizing scale factor from app.ValueEditField, validating it to ensure it is a positive numeric value. If the scale factor is invalid, an error dialog prompts the user to enter a valid value.
Next, the function creates a dialog box to allow the user to choose from various resizing methods. The default method is set to 'bicubic.' After the user selects, it retrieves the image data, resizes the image using the selected method, and displays the resized image on the UI axes. The UI axes title is updated to reflect the applied resizing factor and method. Error handling is implemented to manage potential issues during the resizing process, with error messages displayed in the console and the UI axes title set to 'Error occurred' in case of an error.

Code:

```
    % Button pushed function: ResizeButton
        function ResizeButtonPushed(app, event)
           % Find the image object in UIAxes
            imgObj = findobj(app.UIAxes, 'Type', 'Image');

            % Check if there is no image loaded
            if isempty(imgObj)
                disp('No image loaded.');
                title(app.UIAxes, 'No Image loaded');
                return; % Exit the function if no image is loaded
            end
```

```
            try
                % Get the scale factor from the ValueEditField
                scaleFactor = app.ValueEditField.Value;

                % Validate the scale factor
                if ~isnumeric(scaleFactor) || scaleFactor <= 0 ||
isnan(scaleFactor)
                    % Prompt the user to enter a valid scale factor
                    errordlg('Please enter a valid scale factor in the value
field before resizing.', 'Invalid Scale Factor');
                    return;
                end

                % Create a dialog box to select the resizing method
                methods = {'bilinear', 'nearest', 'bicubic',
'lanczos3','lanczos2', 'cubic', 'triangle', 'box'};
                defaultMethod = 'bicubic'; % Set the default method to
bicubic

                % Use inputdlg instead of listdlg to handle the case where
the user closes the dialog
                dlgTitle = 'Select Resizing Method';
                prompt = 'Choose resizing method:';
                [methodIndex, ok] = listdlg('ListString', methods,
'SelectionMode', 'single', ...
                    'ListSize', [200, 300], 'InitialValue',
find(strcmpi(defaultMethod, methods)), ...
                    'Name', dlgTitle, 'PromptString', prompt);

                % Check if the user canceled the operation
                if isempty(methodIndex) || ~ok
                    disp('Resizing canceled.');
                    return;
                end

                % Map the selected index to the method
                method = methods{methodIndex};

                % Get the image data from the Image object
                imageData = get(imgObj, 'CData');

                % Resize the image using the selected method
                resizedImage = imresize(imageData, scaleFactor, method);

                % Display the resized image in UIAxes
                imshow(resizedImage, 'Parent', app.UIAxes);
                title(app.UIAxes, ['Resized Image by a factor of ',
num2str(scaleFactor), 'by ', method, ' method']);

            catch ME
                % Handle errors
                disp(['Error: ', ME.message]);
                title(app.UIAxes, 'Error occurred');
            end
        end
```

Intensity Profile: This generates and displays the intensity profile of a loaded image on app.UIAxes2. It begins by checking if an image is loaded and prints an error message to the

console if not. The function then prompts the user to choose an exploration option, including exploring all channels, individual channels, or intensity in grayscale if the image is an RGB image. Users can draw a line on the image, then double clicking the end of the line drawn, the function generates intensity profiles based on their choice, plotting the profiles on app.UIAxes2. The function generates a single intensity profile for grayscale images. Error handling is implemented to address potential issues during the exploration process, with error messages displayed in the console and the UI axes title set to 'Intensity profile cancelled' or 'Error occurred' as appropriate.

Code:

```
% Button pushed function: IntensityProfileButton
        function IntensityProfileButtonPushed(app, event)
            % Check if an image is loaded
            imgObj = findobj(app.UIAxes, 'Type', 'Image');
            if isempty(imgObj)
                disp('No image loaded.');
                title(app.UIAxes2, 'No Image loaded');
                return; % Exit the function if no image is loaded
            end

            % Get the image data from the Image object
            imageData = get(imgObj, 'CData');

            % Check image dimensions
            [~, ~, channels] = size(imageData);

            if channels == 3
                % RGB image
                % Let the user choose the exploration option using a custom
dialog box
                dlgTitle = 'Intensity Profile Exploration';
                prompt = 'Choose exploration option:';
                options = {'Explore all channels', 'Explore individual
channel', 'Explore intensity in grayscale'};
                choice = uiconfirm(app.UIFigure, prompt, dlgTitle,
'Options', options, 'DefaultOption', 'Explore all channels');

                if isempty(choice)
                    disp('Intensity profile canceled.');
                    return;
                end

                disp(choice);

                % Let the user draw a line on the image with a different
color
                h = drawline(app.UIAxes);
                wait(h);    % Wait for the user to finish drawing the line
                position = h.Position;
                % delete(h);  % Remove the drawn line from the UIAxes

                % Extract the coordinates of the line
                x = position(:, 1);
                y = position(:, 2);

                colors = {'red', 'green', 'blue'};

                % Generate the intensity profiles based on user choice
```

```
                if strcmpi(choice, 'Explore all channels')
                    % Generate the intensity profiles along the line for
each channel
                    numChannels = size(imageData, 3);
                    profileValues = []; % Initialize as empty, size will be
adjusted dynamically
                    for channel = 1:numChannels
                        channelData = imageData(:,:,channel);
                        tempProfile = improfile(channelData, x, y);

                        % Check the size and adjust profileValues
accordingly
                        if isempty(profileValues)
                            profileValues = zeros(length(tempProfile),
numChannels);
                        else
                            % Check if the size matches
                            if size(tempProfile, 1) ~= size(profileValues,
1)
                                % Adjust the size of profileValues
                                profileValues = zeros(length(tempProfile),
numChannels);
                            end
                        end

                    profileValues(:, channel) = tempProfile;

                    % Plot each channel with its corresponding color
                    plot(app.UIAxes2, tempProfile, 'LineWidth', 1.5,
'DisplayName', sprintf('Channel %d', channel), 'Color', colors{channel});
                    hold(app.UIAxes2, 'on');
                        end

                % Set plot title and labels
                title(app.UIAxes2, 'Intensity Profiles (All Channels)');
                xlabel(app.UIAxes2, 'Position');
                ylabel(app.UIAxes2, 'Intensity');
                hold(app.UIAxes2, 'off');

                elseif strcmpi(choice, 'Explore individual channel')
                    % Ask the user to select a channel
                    channelLabels = cell(1, channels);
                    for channel = 1:channels
                        channelLabels{channel} = sprintf('%d - %s', channel,
colors{channel});
                    end

                    channelChoice = listdlg('PromptString', 'Choose a
channel:', 'ListString', channelLabels, 'SelectionMode', 'single');

                    if isempty(channelChoice)
                        disp('Intensity profile cancelled.');
                        return;
                    end

                    % Extract the selected channel data
                    selectedChannel = imageData(:,:,channelChoice);

                    % Generate the intensity profile along the line for the
selected channel
                    profileValues = improfile(selectedChannel, x, y);
```

```
                    % Plot the intensity profile on UIAxes2 using the
selected color
                    selectedColor = colors{channelChoice};
                    plot(app.UIAxes2, profileValues, 'LineWidth', 1.5,
'DisplayName', sprintf('Channel %d - %s', channelChoice, selectedColor),
'Color', selectedColor);
                    title(app.UIAxes2, ['Intensity Profile (Channel ',
num2str(channelChoice), ' - ', selectedColor, ')']);
                    xlabel(app.UIAxes2, 'Distance along profile');
                    ylabel(app.UIAxes2, 'Intensity');
                else
                    % Convert RGB to grayscale
                    grayscaleImage = rgb2gray(imageData);

                    % Generate the intensity profile along the line for the
grayscale image
                    profileValues = improfile(grayscaleImage, x, y);

                    % Plot the intensity profile on UIAxes2
                    plot(app.UIAxes2, profileValues);
                    title(app.UIAxes2, 'Intensity Profile (RGB Image in
Grayscale)');
                    xlabel(app.UIAxes2, 'Distance along profile');
                    ylabel(app.UIAxes2, 'Intensity');
                end
            else
                % Grayscale image (M*N)
                % Let the user draw a line on the image
                h = drawline(app.UIAxes);
                wait(h);    % Wait for the user to finish drawing the line
                position = h.Position;

                % Extract the coordinates of the line
                x = position(:, 1);
                y = position(:, 2);

                % Generate the intensity profile along the line for the
grayscale image
                profileValues = improfile(imageData, x, y);

                % Plot the intensity profile on UIAxes2
                plot(app.UIAxes2, profileValues);
                title(app.UIAxes2, 'Intensity Profile for M*N image');
                xlabel(app.UIAxes2, 'Distance along profile');
                ylabel(app.UIAxes2, 'Intensity');
            end
        end
```

Save: This enables users to save either the displayed image or the graph. It starts by checking if an image is loaded and prints an error message if not. A confirmation dialog then prompts the user to choose between saving the image, saving the graph, or cancelling the operation. Based on the user's choice, the function opens a file dialog for the respective file type (png, jpg, bmp) and destination. It then captures the content of the relevant UI axes (app.UIAxes or app.UIAxes2) and saves it as an image file. The console outputs messages confirming the successful save or indicating cancellation by the user. The function provides a straightforward way for users to save visual content from the GUI.

Code:

```
% Button pushed function: SaveButton
        function SaveButtonPushed(app, event)
            imgObj = findobj(app.UIAxes, 'Type', 'Image');
            if isempty(imgObj)
                disp('No image loaded.');
                title(app.UIAxes, 'No Image loaded');
                return; % Exit the function if no image is loaded
            end

            % Create a confirmation dialog box
            choice = uiconfirm(app.UIFigure, 'What do you want to save?',
'Save', ...
                'Options', {'Image', 'Graph', 'Cancel'}, 'DefaultOption',
'Image');

            switch choice
                case 'Image'
                    % Save the image
                    [file, path] = uiputfile({'*.png'; '*.jpg'; '*.bmp'},
'Save Image As');
                    if isequal(file, 0) || isequal(path, 0)
                        disp('Image saving cancelled.');
                        return;
                    end
                     % Capture the content of UIAxes and save it as an image
                    exportgraphics(app.UIAxes, fullfile(path, file));
                    disp(['Image saved successfully: ', fullfile(path,
file)]);
                case 'Graph'
                    % Save the intensity profile graph
                    [file, path] = uiputfile({'*.png'; '*.jpg'; '*.bmp'},
'Save Graph As');
                    if isequal(file, 0) || isequal(path, 0)
                        disp('Graph saving cancelled.');
                        return;
                    end
                     % Capture the content of UIAxes2 and save it as an
image
                    exportgraphics(app.UIAxes2, fullfile(path, file));
                    disp(['Graph saved successfully: ', fullfile(path,
file)]);
                case 'Cancel'
                    % User cancelled the operation
                    disp('Saving cancelled.');
            end
        end
    end
```

Information about MATLAB version and operating system used

MATLAB Version: 9.14.0.2337262 (R2023a) Update 5

MATLAB License Number: 904098

Operating System: Mac OS X Version: 10.15.7 Build: 19H2

Java Version: Java 1.8.0_202-b08 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode

---

MATLAB Version 9.14 (R2023a)

Image Processing Toolbox	Version 11.7	(R2023a)
Statistics and Machine Learning Toolbox	Version 12.5	(R2023a)

Acknowledgements

With no prior knowledge of creating MATLAB GUIs, I want to acknowledge that I sought help from the MATLAB documentation, ChatGPT, and watched a couple of YouTube videos. Additionally, I explored and worked with MATLAB App Designer on my own to complete this assignment.

Mathlab documentation links:
https://uk.mathworks.com/help/matlab/ref/appdesigner.html

https://uk.mathworks.com/support/search.html/videos/matlab-and-simulink-robotics-arena-building-apps-with-matlab-and-app-designer-1513378634144.html?fq%5B%5D=asset_type_name:video&fq%5B%5D=category:matlab/gui-development&page=1

Youtube videos:
https://www.youtube.com/watch?v=MHT2W4NjMJA
https://www.youtube.com/watch?v=h2_6FbMsQqU
www.youtube.com/watch?v=kehbc89XRVM