Image Processing Project: Matlab Photoshop GUI

Jonathan Lafleur

Rohit Peesapati

Ruben Martins

Project Submission Date: December 18th, 2017

1st Group to Present Project

I pledge my honor that I have abided by the Stevens Honor System.

Signed:

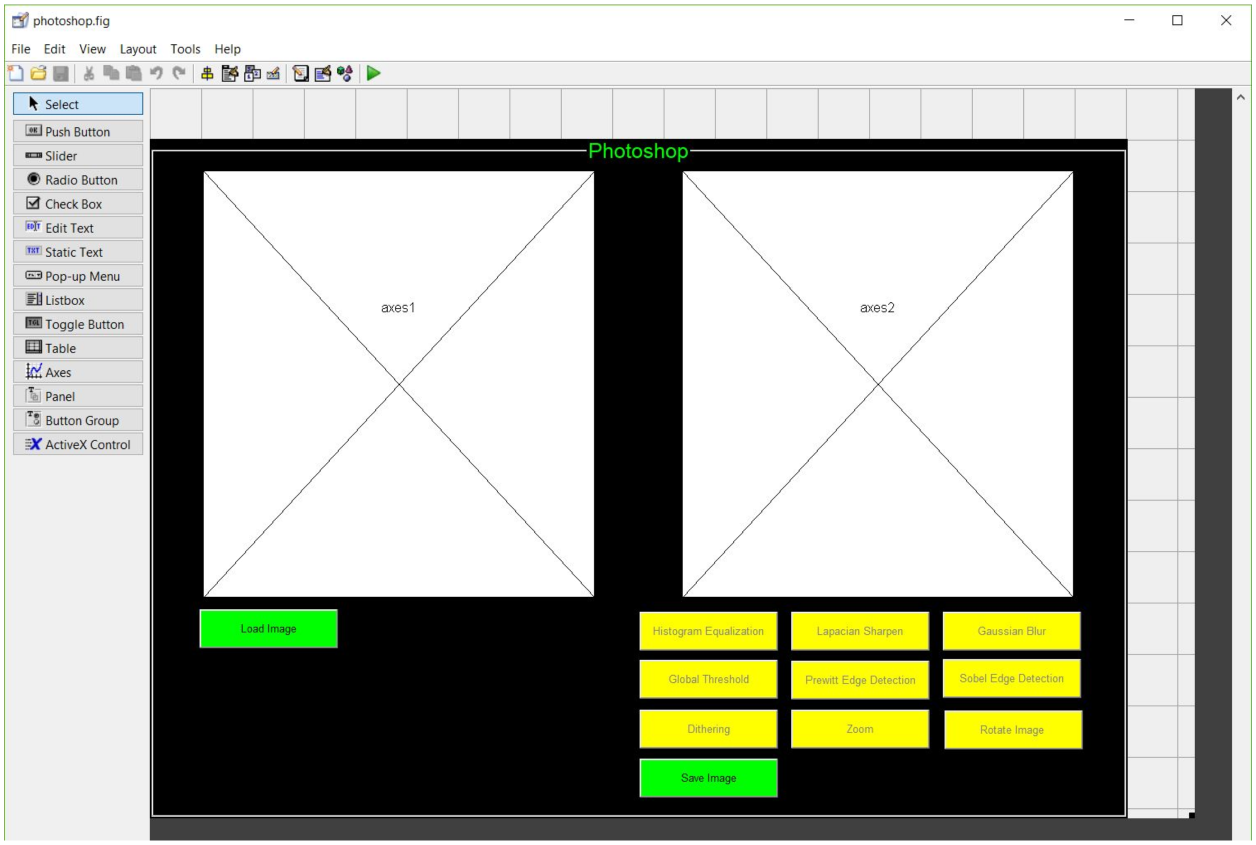
[illegible]	Jonathan Lafleur	12/18/2017
[illegible]	Rohit Peesapati	12/18/2017
[illegible]	Ruben Martins	12/18/2017

Introduction :

In this project, the group decided to use Matlab to create a program that resembles Photoshop by using image processing techniques learned throughout the semester. In creating this program, the group wanted to make a graphical user interface (GUI) that would allow the user to select any picture on their computer, make some changes to the picture using the functions provided in the program, and then save the modified picture. This was one of the main reasons that the group decided to use Matlab, as it provided an easy method of creating a GUI and was powerful enough to implement multiple image processing techniques. The functions that the team included in the final version of the program were histogram equalization, image sharpening, image blurring, image edge detection, global thresholding, dithering, image cropping, and image rotation. By completing assignments in class, the group realized that many of these functions were already implemented within Matlab allowing the group to specify the image and perform image processing. However, the group felt that using these functions would not help the members understand how these functions are actually able to transform the image, thus the group decided to not use any of the available image processing tools in matlab and write our own code to implement the image processing functions. By creating the code, the group hoped to better understand and reinforce all of the image processing techniques taught in class. Another objective in this project was to allow all the group members to learn more about how Matlab can be used to perform complex matrix calculations and transformations that would allow the group to perform image processing. The first step when starting the project was to learn how a GUI could be created through Matlab.

After a quick online search, the group realized that Matlab had a tool called GUIDE(GUI development environment) that allowed the group to design a custom user interface by adding axis to display the input and output images and buttons that could be used to load the image, save the image, and perform each of the image processing functions. The tool also included a layout editor that actually allowed the group to place the buttons and axis exactly where the group wanted them to be. Another benefit with GUIDE was that it linked the code and the layout editor together, so any changes in the layout editor or the code implement the change in the other. This meant that the GUIDE program was used to place the buttons on the layout editor, which then transformed into code that the group could modify to implement one of the image processing functions. An explanation of how the group implemented each of the functions and a picture of the final layout are provided. The final GUI, shown below, created by the group contained two axes side by side, with the left axes used to show the input image and the right axes used to show the output image after modification. Below the input image axis, there is a green button that allows the user to load any .jpg or .png file into the program which would then cause the image to appear above. Under the output image axis, there are multiple buttons that each are used to implement one of the image processing techniques described above and finally a save button.

Functions:



The figure above shows the final version of the photoshop.fig file opened in GUIDE. When push buttons were added to the GUI, matlab would create callback functions for what each button would do when pressed. The group implemented each of those functions.

In the photoshop.m file, the first three functions that are on line 1 to line 73 of the code were matlab generated functions that the group left in because of their usefulness. The first function titled ‘photoshop’ creates a singleton for photoshop and allows their to be only one instance of the program running at any given time. The functions titled ‘photoshop_OpeningFcn’ and ‘photoshop_OutputFcn’ allow the photoshop.fig file to update the handles structure and to output to the matlab command line.

The first function that the group wrote was the ‘loadImage_Callback’ function. This is the callback function for the Load Image button on the left side of the GUI. The function allows the user to select any .jpg or .png file from their computer. Then an aspect ratio of the image is taken to figure out the ratio of height to width of the image. Then the image is resized so that the maximum amount of pixels on either the height or the width is 512 pixels and the original aspect ratio of height to width is maintained so as to not distort the image vertically or horizontally. A maximum size of 512 pixels was chosen as a way to reduce computation resources and time spent on each computation allowing the subsequent functions to run faster.

Next, the input image was converted into a grayscale image using the function ‘RGBtoGray’ that the group made instead of using the ‘rgb2gray’ function that was already built into matlab. To complete this function the group repurposed code from homework 5 from this course. The function works by multiplying each of the r, g, and b values by different constants from the ITU color conversion matrix and then adding the values together.

After the input image is converted to grayscale, it is shown on the left axes in the GUI. The group chose to show the input image on the left axis and the output image on the right axes as a way of comparing the differences between the two images. The group chose to use global variables for the input image and the output image as a way of easily using the images in the different functions. Once the input image is shown on the left axis, all of the buttons for the different functions that can be applied to the image are enabled to be pressed because they now have an image on which to perform such functions.

The function titled ‘getGlobalThreshold’ is called at the end of the ‘loadImage_Callback’ function as a way of defining a global variable called threshold in the background when the input image is just being loaded. This allows the global threshold to be used in later functions without a need to spend time computing it when the button for that function is pressed. The way the function works is by choosing a random global threshold and then in a while loop setting an initial threshold value to that global threshold value and then figuring out how many values and the sum of those values are above and below the global threshold and using the average value of the average value above and the average value below the threshold to define a new global threshold. This process is repeated until the global threshold and the initial threshold are close to each other and a global threshold of that value is a good threshold value.

A major function to the groups program was the ‘filter3by3’ function. The function works by taking in a three by three filter to be applied to an image and an the image it will be applied to. The function works by going through each of the pixels on the inside, not the edge, of the image and doing a matrix element wise multiply of a three by three matrix from the image with the selected pixel at the center of that matrix multiplied by the three by three filter. The resultant image was the output of the function. Although this method of looping through each pixel of the image is slow in comparison to the matlab functions ‘imfilter’ or ‘conv2’, the group thought it was important to build their own functions to get a sense of how the people at mathworks implemented them. One way that the groups ‘filter3by3’ function could be improved is by doing the element wise matrix multiplications in parallel to each other instead of one after the other as is done in the loop. Parallel computing would be useful because the output pixels are independent of one another and therefore can be computed at the same time.

The function titled ‘LaplacianSharpen_Callback’ uses the the laplacian sharpening matrix applied to the ‘filter3by3’ function to sharpen each pixel of the image. The image is sharpened by amplifying the pixel at the center of the filter matrix and then subtracting the four pixels next to it, thus amplifying areas of sharp contrast and doing nothing to areas with about the same pixel values. The function ‘gaussianBlur_Callback’ also uses a matrix and the ‘filter3by3’ function to output an image. Instead of sharpening the image, the gaussian blur matrix blurs the image by adding up the average of the nine nearest pixel values and adding them to the output.

The prewitt and sobel edge detection functions each apply a vertical and a horizontal edge detection matrix to the image and then adds the absolute value of the two resulting images. The result of this is then passed through the global threshold and outputted.

The ‘globalThreshold_Callback’ takes in an image and the global threshold value that was calculated when the image was loaded. It then makes a logic matrix that compares each of the pixel values to the global threshold and makes pixels greater than the global threshold white and those less than or equal to the global threshold black.

The next function the group wrote is the ‘histogramEqualization_Callback’ function. This was the callback function that was created along with the “Histogram Equalization” button under axes2. The code for this function was created using the information from Lecture 6, Slides 19-23. The first step in this code was to use a loop to cycle through the input image using two “for” loops and figure out the number of pixels at each amplitude value. Since the input image was a grayscale, this meant that all the pixels had to have a value between 0 and 255, thus an array of type double with a size of 256 called pixelFrequency was created. The next step was to use the size of the input image to calculate the probability of each amplitude value using the formula: $p(r_k) \ = \ n(r_k)/N$ where $n(r_k)$ is the number of occurrences of a certain amplitude and N is the total number of pixels. Using the size() function on matlab, the size of the input image was known, thus the number of occurrences of each amplitude was divided by the total number of pixels in the image and the resulting values were stored back in pixelFrequency. Using this probability of amplitude values for the input image, the following equation was used to find the mapping function: $T(r_l) = Round\{255 * \sum_{k=0}^{l} p_i(r_k)\}$. In order to implement this equation in code, another array of type double called “s” was created to represent the mapping function and a variable called “sum” was created to keep a sum of the probability values, acting as the summation function. The function round() that is available in matlab was used to round the values to the nearest integers. Finally using the mapping function, all pixels at a certain amplitude on the input image were “moved” to the corresponding amplitude using the mapping function “s”, thus performing histogram equalization. Finally, the modified image would be output on axes2 to the user.

Another major image processing function was image cropping(zoom), which would have the user specify points on the input image and then output a cropped image. Since the group members were not sure of how to get the user to specify points on the image, the group decided to google the problem. The search provided with the group a mathworks page about ginput, which could be used to get graphical input from the cursor. The first step was to tell the user to pick the top left corner point and bottom right corner point of a rectangle, which is the part of the image that will be zoomed. Next, using ginput(2), the x and y coordinates for both points specified by the user are stored in a matrix called “p”. Next, these coordinate values are rounded to integer values and then using the max() and min() functions available to matlab the “xmin”, “ymin”, “xmax”, and “ymax” points are found. The reason that this was included in the code is due to the fact that the user is never specifically told to select the top left point first or the bottom right point first, thus the order that the user selects the points is unknown. Since the order is not known, the min() and max() functions are used to find the required coordinates points to create a rectangle. The next step is to calculate the width and height of the intended rectangle by using the “xmin”, “ymin”, “xmax”, and “ymax” points. The width is calculated by taking “xmax” and subtracting “xmin” and then adding one to account for the fact that “xmin”point is also included in the rectangle. Similarly, the height is calculated by taking “ymax” and subtracting “ymin” and then adding one. Using the width and height values calculated, the rectangle function in matlab was used to draw a rectangle on the input image to show the user where the picture is zoomed. Using the min and max coordinates points, the corresponding pixels of the input image were copied to the output image and the output image was shown to the user. Once the output image is shown, the program waits till the user clicks any button on their keyboard or mouse before deleting the rectangle drawn on the input image.

The next image processing function was “dithering_Callback” which corresponded to the Dithering button on the graphic layout. The group decided to include dithering in the project before the topic was introduced in class, thus the group had to rely on outside sources to understand what dithering is and how it could be implemented in this program. Online, the group was able to find a wikipedia article about “Floyd-Steinberg Dithering”, which used error diffusion to achieve dithering. This filter and topic was introduced on the last few slides of lecture 11 in class. To implement dithering, the group used two “for” loops to run through the input image. At each pixel, the pixel intensity was copied to a variable called oldpixel and using the equation: $newpixel = 255 * round(oldpixel/256)$, a new pixel value was calculated. This newpixel intensity value was then assigned to the current location of the output image. Then using the newpixel value and oldpixel value, the quantization error was calculated. Finally, using this quantization error and the weight factors from the Floyd-Steinberg matrix, the error is diffused among the neighboring pixels. At the end of the function, the dithered image is output to the user through axes2.

The ‘saveImage_Callback’ function was the last function that the group implemented. It takes the image displayed on the right side of the GUI and uses the matlab imwrite function to save the image as a file called ‘outFile.png’. It then notifies the user that the image has been saved.

The final function in the program is the "rotateImageDegrees_Callback" which is used to rotate the image a certain number of degrees that the user inputs. This means that the first step in this function is to ask the user to enter a value that they want to rotate the image and then store the value the user enters into a variable called answers. Since the user input is stored as a string, the "str2double()" function is used to convert the value in answers into a double and store the value in a variable called degree. If the user did not enter a number into the input, the program will just output whatever the input image is. Otherwise the program checks the value and scales it down to a value between 0 and 360 using the mod() function, with the new value stored back into the variable degree. If the value in degree is 90, then the image is simply transposed, otherwise the output image needs to be calculated using the input image. Slides 19- 21 from lecture 11 were used to help in writing the code to implement the image rotation function. The first step was to convert the degree value to radian as matlab used radians in its sin() and cos() functions. The input image was copied to another matrix called "copy_image" to make any changes to the original image would not affect the input image shown to the user. Then using the size of the input image and the angle provided, the size of the output matrix was calculated. To get the number of rows in the output image, the following equation was used: $Number\ of\ rows\ in\ input\ image * |cos(\theta)| + Number\ of\ columns\ in\ input\ image * |sin(\theta)|$. To get the number of rows the following equation was used: $Number\ of\ rows\ in\ input\ image * |sin(\theta)| + Number\ of\ columns\ in\ input\ image * |cos(\theta)|$. Then using the number of rows and columns in the output image, two "for" loops were created that would use interpolation methods on the original input image for the output image. In the for loops, the equations on slide 19 of lecture 11 were used: $m_1 = (n_1 - N_1/2) * cos(\theta) + (n_2 - N_2/2) * sin(\theta)$ and $m_2 = -(n_1 - N_1/2) * sin(\theta) + (n_2 - N_2/2) * cos(\theta)$ to implement a coordinate transform. Once the m_1 and m_2 were calculated, they were added to the coordinate values for the center of the input image and stored back into m_1 and m_2. Next, there was a check to see if the new m_1 and m_2 values were in the range of the input image, and if they were then the pixel at (m_1, m_2) on the input image was copied to location (i,j) of the output image. I and J were the two variables that were used in the "for" loop. Once this process completed, the rotated image stored as "image_out" is displayed to the user through axes2.

Roles:

After deciding what project the group wanted to do, the group had to decide what image processing functions had to be included in the final version. As mentioned in the introduction above, the group decided to have eight functions that were split between all three team members to make sure all members contributed to the final product. At the beginning of the project, when the group was creating the layout and trying to decide what functions to include, all members played an active role in contributing their ideas and concerns. Once this was decided, the group did break up the functions as mentioned. Jon was responsible for the image sharpening, image blurring, and image edge detection. Rohit was responsible for the histogram equalization, global thresholding, and image cropping. Finally, Ruben was responsible for the dithering and image rotation. Although, all the members were responsible for their individual functions, the group would still always try to meet together to work on the project to make sure that if someone was having problems implementing or understanding one of their image processing functions, the other members would be available to help. For the report, the members once again all took an active role helping each other write sections of the report to make sure everything about the project and how the group completed the project is explained in detail. One of the major reasons that the group tried to always split the work is to make sure that all three members felt that they had a crucial contribution to the final program. This goes along with the idea that no single member of the group should be burdened with doing the majority of the project, so by splitting the work equally between all the members before starting the project, every member will have the same amount of responsibility to complete the project. Another reason that the team decided to split the work is to provide each member of the group with a clear goal to work towards, instead of worrying about multiple different parts of the project at once. This means that the large task of creating this GUI in Matlab that can perform image processing is broken down into small, manageable parts that each member can work on. Once all the members are completed with their part, the parts were combined together to create the final project.

Accomplishments:

The group successfully implemented a GUI capable of accomplishing similar functions that are in the popular program Adobe photoshop. In the beginning of the project the group attempted using GIT in order for the group to have update changes to the code at all times. GIT would have allowed the group to push new code every time changes were made. Pushing new code using GIT would also keep the project organized and in one place for the group to easily access it; however, the group ended up not using GIT although the group invested much amount of its time in attempting to get it to work. Although the group did not use GIT, from this project, the group was able to retain foundational knowledge about GIT and how GIT works. The group had problems pushing code to GIT and retrieving it from GIT. One of the group members also had a Mac that was using Bootcamp to run Windows; however, as he used his version to run GIT, he ran into problems as some features were not compatible. Therefore, the group opted not to use GIT.

Another accomplishment was that the group was able to implement concepts of image processing learned in lectures. The group was able to apply concepts from lecture slides into the Matlab photoshop GUI. Using basic image processing techniques from lecture slides, the group was able to successfully apply them on any loaded image. Another accomplishment by the group was the ability to gain knowledge in Matlab. Before this project, the group did not know that

GUIs were able to be made in Matlab. Additionally, the group did not have much programming knowledge in the beginning of the project; therefore, the group successfully learned coding techniques in order to create the functions that controlled the image processing on the loaded image. The group was able to learn the foundations of how Adobe Photoshop works on the backend by creating the callbacks that governed how the group's Matlab Photoshop GUI worked. In conclusion, this project was a success for the group because the group's objectives were surpassed. In the project the group was able to incorporate more image processing techniques than what was expected. Image rotation and dithering were learned in the last lecture of the course, and the group decided to incorporate them as we now had the Matlab and backend knowledge to do so.