UNIVERSITY OF CALIFORNIA, SAN DIEGO

Image Processing and Analysis Application Developed on a Mobile Platform

A thesis submitted in partial satisfaction of the requirements for the degree Master of Science

in

Bioengineering

by

Michael Skinner

Committee in Charge:

Professor Sameer Shah, Chair
Professor Adam Engler
Professor Geert Schmid-Schonbein

2015

The Thesis of Michael Skinner is approved and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Sameer Shah for his support and guidance throughout the duration of my project. Without his input, my project would have taken much longer to complete.

I would also like to acknowledge the members of the Shah lab, who have provided important insight for my work.

ABSTRACT OF THE THESIS

Image Processing and Analysis Application Developed on a Mobile Platform

by

Michael Skinner

Master of Science in Bioengineering

University of California, San Diego, 2015

Professor Sameer Shah, Chair

In medicine and biology, image analysis is often used to quantify outcomes for a wide range of applications, including diagnostics, surgery, and general research. Image analysis tasks can include counting the number of cells in a given area of a fluorescently stained image or measuring the geometry of tissues during surgery. Currently, in these situations, one of the more common and powerful programs used for image analysis is ImageJ. However, ImageJ is a computer application, so before an image can be analyzed, it has to be loaded and saved to the computer. On the other hand, while there are mobile image processing applications, most of them have limited

functionality, performing manipulations such as adding filters and masks to pictures to make them look more appealing, qualitatively. My project focuses on creating a mobile application with various useful analytic functions, so that image analysis can be done quickly and without need for an external computer. Some of the functions that will be included are: cropping, zooming, measuring, counting, brightness, drawing/labeling, transforming, and masking.

INTRODUCTION

Background/Motivation - Image processing and analysis is used for a variety of reasons. The context that I will be focusing on is for scientific purposes. It is usually insufficient to qualitatively describe images in a scientific setting, so quantitative measurements such as lengths, counts, and pixel data must be used. Figure 1 shows two fluorescently stain nerve cells where image processing and analysis are important to be able to describe them. For the left image, information about the area of the rings which correlate to the laminin surrounding neurons is important and for the right image, information about the count of red dots which correlate to axons is important. For my project, I am creating a mobile application that has functionality that is useful for image processing and analysis in a scientific setting.

Two of the main areas where a mobile image processing and analysis application can be useful are fluorescent microscopy and surgery. During microscopy, it can be important to quickly determine measurements such as the number, size, and color of the cells. For these quick measurements, it would be useful to have a mobile application that could find these values without having to go through the process of loading the image into Image J. Although it seems just as easy to use Image J, there are cases when the computers are at a separate site, meaning that there would be no way to quickly find various measurements without a mobile application.

Surgery is another case that having a mobile image processing and analysis application would be useful. There is a time constraint during surgery and it can be useful for the surgeon to find specific measurements. A mobile application would

give the surgeon a new tool to immediately process and analyze images if it helps with the decision-making.

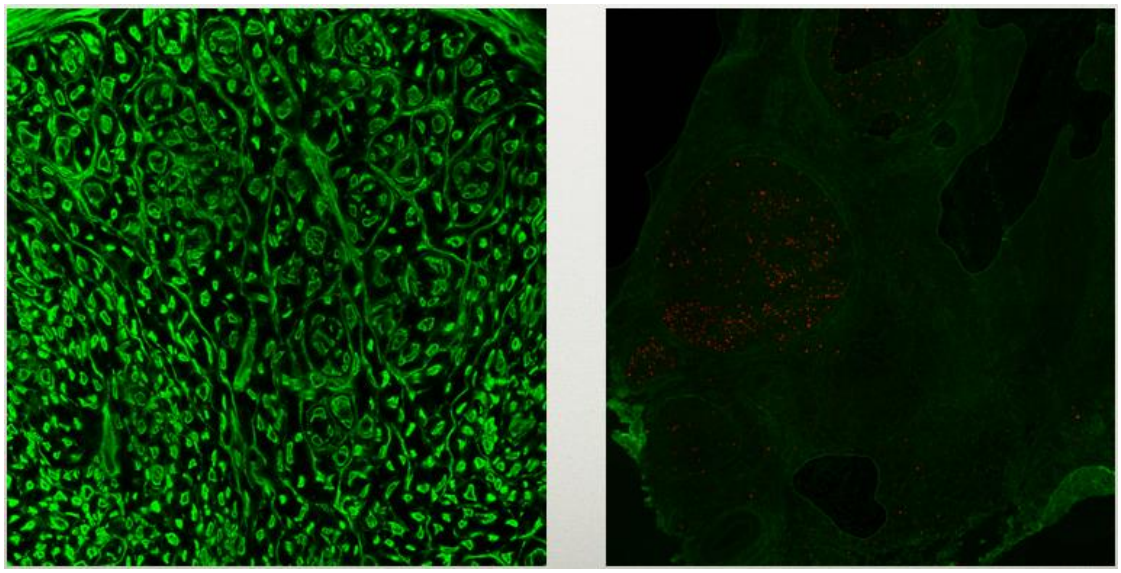

Figure 1 - Images of fluorescently labeled cells

Current Market - There are numerous image processing applications available on multiple platforms depending on the user's needs. Five of the popular computer applications are Image Tricks Pro, Flare, Image Editor, Adobe Photoshop, and Image J [1,2,3,4]. Three of the popular mobile applications are Photo Editor, Adobe Photoshop Express, and Instagram [5]. Table 1 categorizes these application based on cost, mobility, basic image processing ability, counting or measuring ability, and histogram ability. Basic image processing includes features like cropping, drawing, masking, filtering, and other processes that visually alter the image. Both Adobe Photoshop and Image J have these analysis tools including measuring, counting, and histogram creation. Image J specifically targets the scientific community and is funded by the NIH for this reason, which is why my application is modeled after it [4].

Essentially, the niche that my applications targets is being a mobile application that has similar features to Image J so that it can be used in a scientific setting.

Table 1 - Images processing applications and their functionality

	Cost	Mobile	Basic Image Processing	Measuring/Counting	Histogram
Image Tricks Pro	9.99	No	Yes	No	No
Flare	9.99	No	Yes	No	No
Image Editor	29.99	No	Yes	No	No
Adobe Photoshop	Starts at 9.99/mon	No	Yes	Yes	Yes
Image J	Free	No	Yes	Yes	Yes
Photo Editor	Free	Yes	Yes	No	No
Adobe Photoshop Express	Free (In App Purchases)	Yes	Yes	No	No
Instagram	Free	Yes	Yes	No	No

Possible Uses - One of the ways that this application can be used is in tandem with cell phone devices such as cell phone microscopes and cell phone infrared cameras. Having an image processing and analysis application can allow users of these devices to immediately analyze images taken, leading to the advancement of areas including portable medicine, field science, and emergency medicine.

Portable Medicine: Portable medicine is an area that a mobile image processing and analysis application can greatly benefit. There is a lack of resources in developing nations or remote areas and it can be very difficult for people to travel to the nearest

hospital. This means that it is important for doctors to have portable equipment that can be used to diagnose and treat people on site.

Two such applications are examining blood or skin samples. Using a cell phone microscope, blood or skin samples can be imaged and sent to a facility where other doctors can examine the images. However, in many of these remote areas, cell phone service is not reliable, and it would be useful if the doctor on site can process and analyze the samples. For a disease such as malaria, the on site doctor can do a blood smear, image it using a cell phone microscope, and analyze the image for parasites all on site using my application.

Field Science: Scientists often work in areas with no access to technical equipment. This can include oceanographers at sea, environmental scientists in the forest, and archaeologists in the desert. Being able to process and analyze images on site can provide these scientists with useful information. For example, one of the most important things to study about ocean ecology is plankton since the amount and type of plankton is crucial to the entire ecosystem. Scientists use devices such as the Schindler-Patalas trap to collect plankton samples at sea, and examine the samples taken once they get back to the lab. Having a cell phone microscope can allow the scientist to look at the samples immediately. Being able to process and analyze the image could mean that the scientists can immediately see if the sample is adequate or if a new one should be taken. Additionally, the ability to count and measure the plankton using an image processing and analysis application could allow the scientists to see trends about where plankton samples are the best, and adjust the collection of samples based on this information.

Emergency Medicine: Another potential use for my application is in ambulances. Ambulances are often transporting patients that are critically injured to hospitals that have the proper equipment to treat them. Because of the time sensitive nature of this, it is important for the EMT's in the ambulance to relay information about the patient to the hospital so that the doctors at the hospital can prepare for the patient. Using my application, things such as measuring wound size and location can be more accurately relayed to the hospital. Used in tandem with a cell phone microscope, the EMT could immediately find out if wounds are infected. Used in tandem with a cell phone infrared camera, the EMT would even be able to quantify the degree of hypothermia or frostbite a patient is suffering. These are just a few of the ways that an EMT could make use of this application.

Initial Steps - Because I am using Image J as a model for my application, it was important to extensively use Image J and learn the functionality. After this, I gathered many of the useful functions of Image J and created a survey to determine how to prioritize the functions for my application. I created a simple survey that has only three options for each of the functions: Essential, Nice, or Unnecessary (Figure 2). Because the application is intended to be used as a scientific tool, I distributed the survey to members of the Orthopaedic Surgery lab, as these people are a good approximation of my target market. I received surveys from a total of 20 people, and scored each of the functions on a scale of zero to one. I gave "Essential" answers a score of one, "Nice" answers a score of 0.5, and "Unnecessary" answers a score of zero. Then, I took the average of all of the surveys to come up with a single

importance score for each of the features (Table 2). The features that obtained a importance score of 0.8 or higher are classified as essential functions to have, and these features included crop, zoom, draw, transform, measure distance, measure area, count, and brighten/contrast. The rest of the features obtained a score below 0.8, and will be focused on slightly less than the most essential functions.

Image Processing App Survey

Return to Michael Skinner directly or at URC Room 110

Degree and/or Position: _____

(Short Description is Fine. Ex: B.S. in Bioengineering)

	Image J	Photoshop	Other
Have you used Image Processing Software?	☐	☐	☐

	Yes	No
Would a mobile image processing app be useful to you?	☐	☐

Image Processing Features	Essential	Nice	Unnecessary
Crop	☐	☐	☐
Zoom	☐	☐	☐
Draw/Paintbrush/Point Markers (Arrow/Numbers)	☐	☐	☐
Transform (Invert, Flip, Rotate)	☐	☐	☐
Grayscale	☐	☐	☐
Binary (Threshold Image)	☐	☐	☐
Masks/Overlays	☐	☐	☐
Color Filters	☐	☐	☐
Splitting Color Channels (RGB in separate images)	☐	☐	☐
Measure Distance/Area	☐	☐	☐
Count	☐	☐	☐
Brighten/Contrast	☐	☐	☐
Smooth/Sharpen/Find Edges	☐	☐	☐
Noise Reduction	☐	☐	☐
Create Histogram	☐	☐	☐

Comments:

Figure 2 - Image processing survey

Table 2 - Importance scores for each individual feature from the image processing survey. The scores range from 0 (unnecessary) to 1 (essential).

Feature	Importance Score
Crop	0.8
Zoom	0.975
Draw	0.895
Transform	0.85
Grayscale	0.65
Binary	0.684
Masks/Overlays	0.711
Color Filters	0.6
Splitting Color Channels	0.525
Measure Distance/Area	1.0
Count	0.85
Brightness/Contrast	0.95
Smooth/Sharpen/Edges	0.625
Noise Reduction	0.625
Histogram	0.65

Implementation - Xcode is the main platform for developing iOS applications, which are used for the majority of Apple products [6]. The program uses Objective-C, and has a "storyboard" that allows the developer to work directly with the user interface and easily interact with individual objects. There are three main types of files that are utilized by the program: main files, storyboard files, and xib (storyboard) files.

The main files contain the bulk of the code, as this is where all of the main actions takes place. My application contains five separate main files: AppDelegate.m, ViewController.m, CropViewController.m, DrawViewController.m, and MeasureViewController.m. The AppDelegate file is already written by Xcode and contains the actions that ensure that the application opens and closes properly. I split the main code into the other four sections based on the four separate screens that my application contains, which will be addressed later.

The header files state the functions and actions that are contained within the main files so that these things can be accessed from other locations. There is one header file for each of the main files which allows the application to interact with other files within the program. For example, an image in the main view controller can be transferred to the crop view controller so the user does not have to reload the image.

The xib file is often referred to as the storyboard and is an easy way to directly interact with the user interface. Objects such as buttons, switches, and sliders can be placed on the user interface and then connected to the main code.

Source Code Acknowledgements - My project contains sections of code that has been previously used in different applications. One of the major influential code snippets that I used was from DrawPad by Abdul Azeem Khan (Appendix 1) [7]. I used the section of code that tracks the cursor movement in multiple sections of my application including cropping, drawing, and measuring distance and area. I maintained the three functions: touchesBegan, touchesMoved, and touchesEnded and changed minor actions based on the functionality I was aiming to achieve.

Another influential snippet of code that I used was written by Goran Vuksic to convert an image to "night mode" by manipulating the pixel data (Appendix 2) [8]. I used a similar code for multiple functions that directly utilize the pixel data including grayscale, binary, reflection, inversion, and creating a histogram. I had to make basic changes such as making the pixel array mutable rather than static so that additionally actions can be done to the pixel data sequentially. For example, reflecting an image followed by grayscaling this newly reflected image requires the pixel data to be updated. I maintained the basic flow of first breaking an image down to the pixel data, then doing some action to this pixel data, and finally rendering a new image using the updated image.

There were numerous other programs that helped me mold pieces of my application. Because Xcode is a highly used developing program with many users, there are countless code examples readily available. The novelty of my application is not the individual functionality but rather the collection of features that create the big picture image processing and analysis application.

Scope of Application - Table 3 is a general summary of the scope of my application. Basic functions such as importing and saving images are not included in this table as only the processing and analysis functions are noted. The novelty of my application is taking all of these functions and bringing them together to create a good foundation for an image processing and analysis application.

Table 3 - Scope of Application

Feature	Location (View Controller)	Category
Crop	Crop (C)	Processing
Zoom	Crop (C)	Processing
Rotate	Crop (C)	Processing
Draw	Draw (D)	Processing
Measure Distance/Area	Measure (M)	Analysis
Count	Measure (M)	Analysis
Reflect	Main	Processing
Invert	Main	Processing
Grayscale	Main	Processing
Binary	Main	Processing
Brightness/Contrast	Main	Processing
Histogram	Main	Analysis

CHAPTER 1 - Main View Controller

The Main View Controller is the home screen and the first screen that pops up when the user opens the application (Figure 3). This screen has two main features: the image view, which takes up the majority of the screen and the toolbar, which allows the user to analyze and process the image. When the application is first opened, the image view is left empty, and it is not filled until the user imports an image from the camera roll. After this step, there are numerous things that the user can do with the image, including simple manipulations like inverting and cropping, finding measurements, and creating histograms.

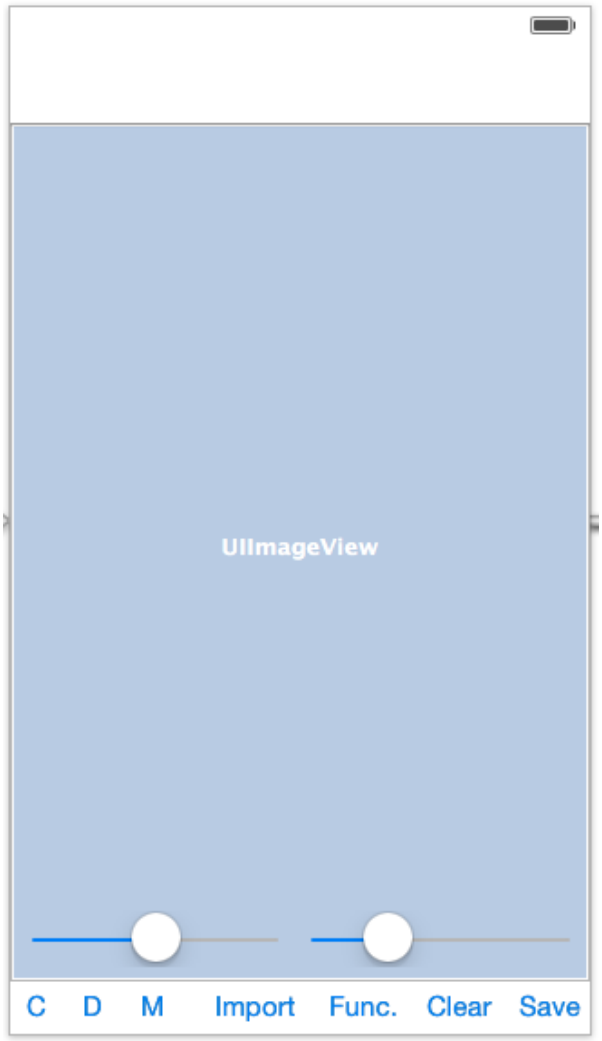

Figure 3 - Screenshot of the main view controller

Toolbar - The bottom portion of the screen (Figure 3) contains the buttons that allow the user to do actions. The first three buttons (C, D, M) are navigation controllers and segue to a different view controller. C navigates to the crop view controller (Chapter 2). D navigates to the draw view controller (Chapter 3). M navigates to the measure view controller (Chapter 4).

Import Images - The program allows the user to import images from the camera's saved photo album into the program. This means that it is a very easy process to take a picture using the phone's camera and then quickly access this picture via the application.

Functions

Grayscale - The pixel data of the image is stored in an RGBA format. This means that every pixel has four 8-bit integer values associated with it. The first three correspond to the red, green, and blue color in the individual pixel. The possible values range from 0 (no color) to 255 (max color). The fourth component refers to the alpha value of the pixel, which is the opacity. For my program, I am not altering the opacity of the images; only the RGB components. For this reason, I will not consider the alpha value of the pixel in any of the manipulations that occur.

A simple way to create a grayscale image is to match the values of the RGB components. By equalizing the values of the RGB components, the information is condensed to a single integer that represents the value of all three color channels. This also correlates to the shade of gray. If the RGB components are all 0, the pixel will be black, if the RBG components are all 255, the pixel will be white, and everything in between will be some shade of gray.

The two most non-specific ways to determine the grayscale value are the average method and the luminosity method [9]. In the average method, the RGB values are set to the average value of the three components. For my program I used the luminosity method, which is slightly more complex than the average method. Instead of taking a straight average of the three components, the average is weighted so that

the RGB values are set to (0.3 * Red + 0.59 * Green + 0.11 * Blue) [9]. The reasoning behind weighting the values like this is to account for how people perceive colors. It has been shown that people have different sensitivities to different colors [9]. The luminosity method quantifies the differing sensitivities to red, green, and blue into this weighted formula which results in a qualitatively better grayscaled image.

Figure 4 shows the difference between taking the simple average of the three components versus taking the weighted average. The image is used by Tanner Helland [9] to illustrate the differences between the grayscale methods. Despite the vastly different weighting of the RGB components, the two grayscaled images look remarkable similar with only subtle differences. Upon careful inspection, the luminosity method (bottom right of Figure 4) appears to have more definition than the average method (top right of Figure 4). Focusing on the upper left portion of the image where the text appears, the average method appears more dull than the luminosity method, which pops out slightly more.

Figure 4 - Image on the left is the original. Image in the top right is grayscaled using the average method. Image in the bottom right is grayscaled using the luminosity method.

Binary - For a binary image, each pixel must be either black or white. This means that there must be some way to determine a threshold value for the given image where pixels above the threshold are converted to white and pixels below the threshold are converted to black. Using a simple if/else statement, each individual pixel is set to either white or black, depending on whether it is above or below the threshold value. To determine this threshold value, Otsu's method is implemented. Otsu's method is a higher order iterative process that searches for the threshold value that maximizes the inter-class variance of the upper half and lower half [10]. The basic premise behind this method is that the pixels of an image can be categorized into two groups: foreground and background. Ideally the two groups will be clustered together so a threshold value can be chosen between the two groups. Figure 5a shows how the inter-class variance changes as different threshold values are iterated through. Because the graph is a smooth curve rather than having multiple local maxima, the threshold value selection will be consistent even if the image is altered slightly. Otsu's method uses the following algorithm to mathematically find the best threshold value that maximizes inter-class variance:

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \quad 10$$

Total Variance = Weight * Variance (Background) + Weight * Variance (Foreground)

Figure 5b shows binarization using Otsu's method compared with the IsoData method, which is another popular thresholding technique. The main difference between the two methods is that Otsu's method accounts for the weight of both the foreground and background pixel values while the IsoData method does not [11]. This means that Otsu's method of thresholding will tend to pick a value that is closer to the

middle of the pixel data, resulting in a binary image that has a more equalized number of black and white pixels. Determining which method is better is subjective and both methods have been used previously. My choice of Otsu's method is a personal preference.

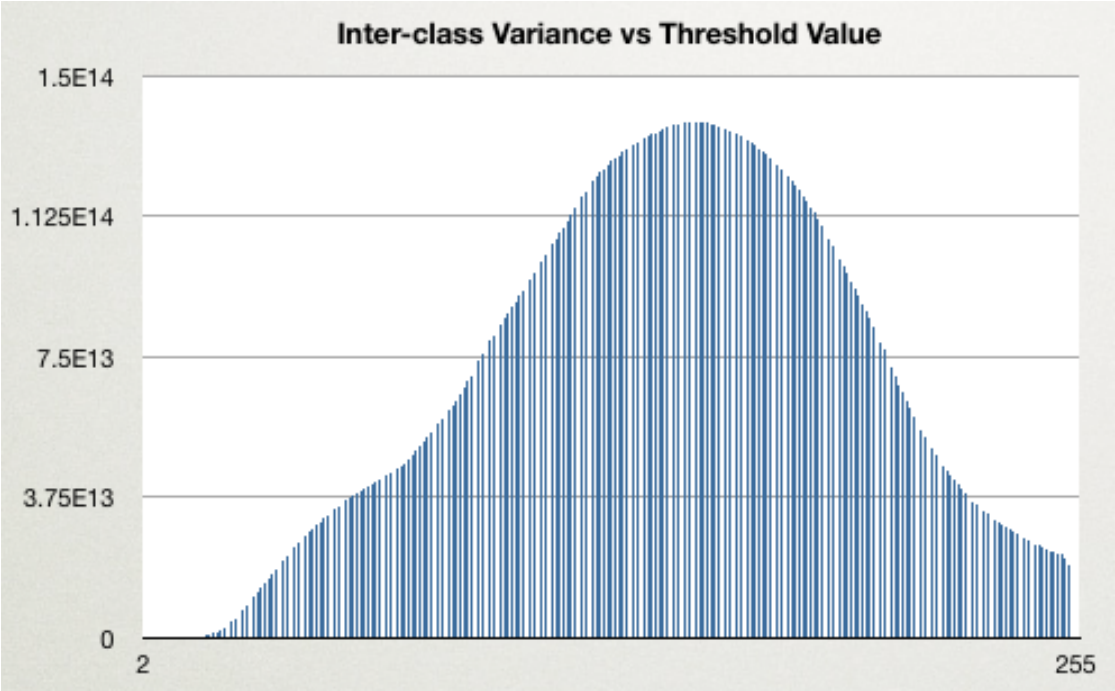

Figure 5a - The histogram of inter-class variance using Otsu's method of an image as different possible threshold values are selected.

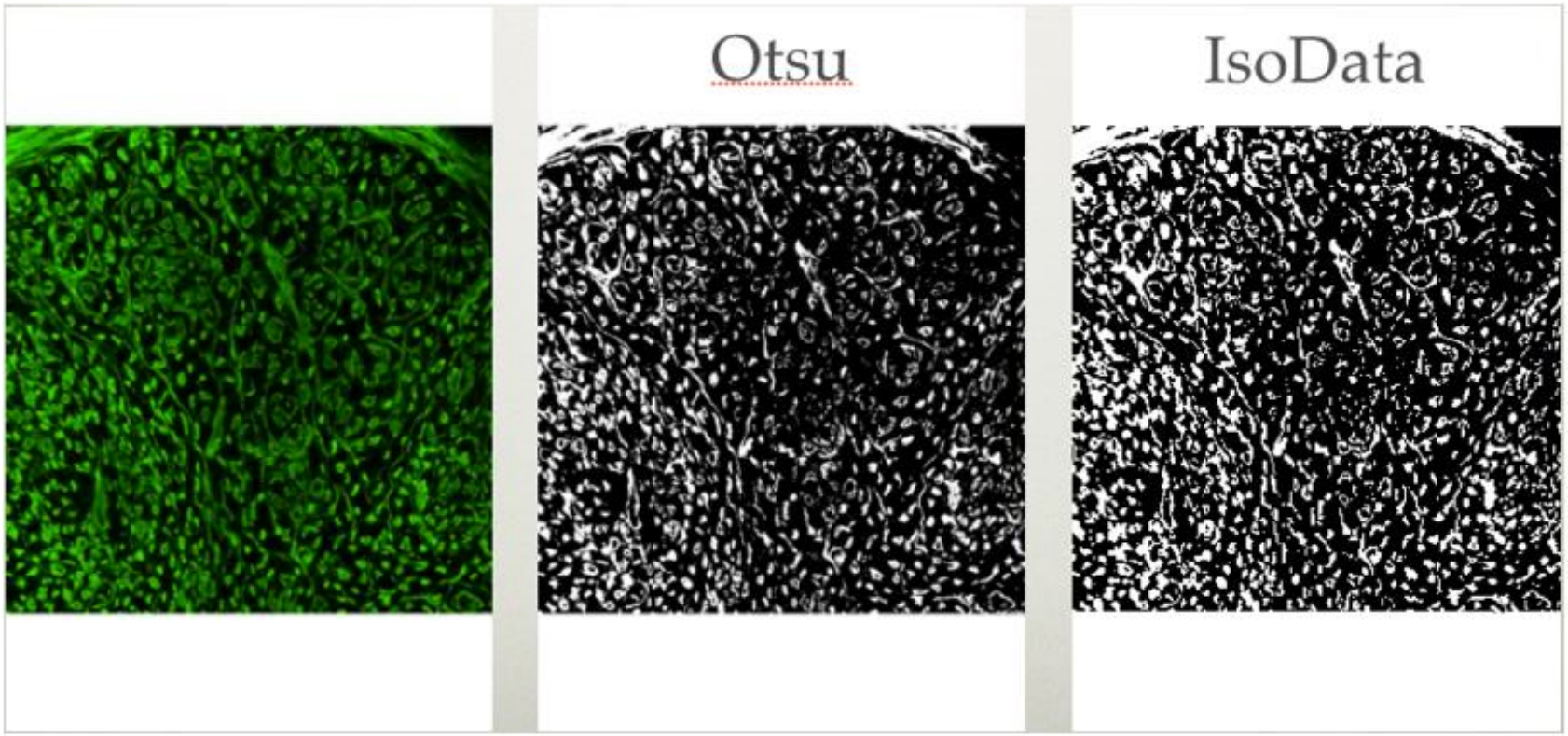

Figure 5b - The image on the left is the original. The image in the center is binarized by my application using Otsu's thresholding method. The image on the right is binarized by Image J using the IsoData thresholding method.

Invert - Inversion of an image refers to the colors of the picture. When an image is inverted, it means that each pixel is changed to the opposite color. Black pixels become white pixels and vice versa, and all throughout the color continuum, the pixels are transformed. The way the pixel data is stored, this is an easy function to implement. In each of the RGB color channels, the pixel data is converted to 255 minus the pixel data. This means that a value of 0 will be converted to 255, 1 will be converted to 254, and so on. By doing this to all of the color channels, the image is effectively inverted. Figure 6 depicts the changes that occur when an image is inverted.

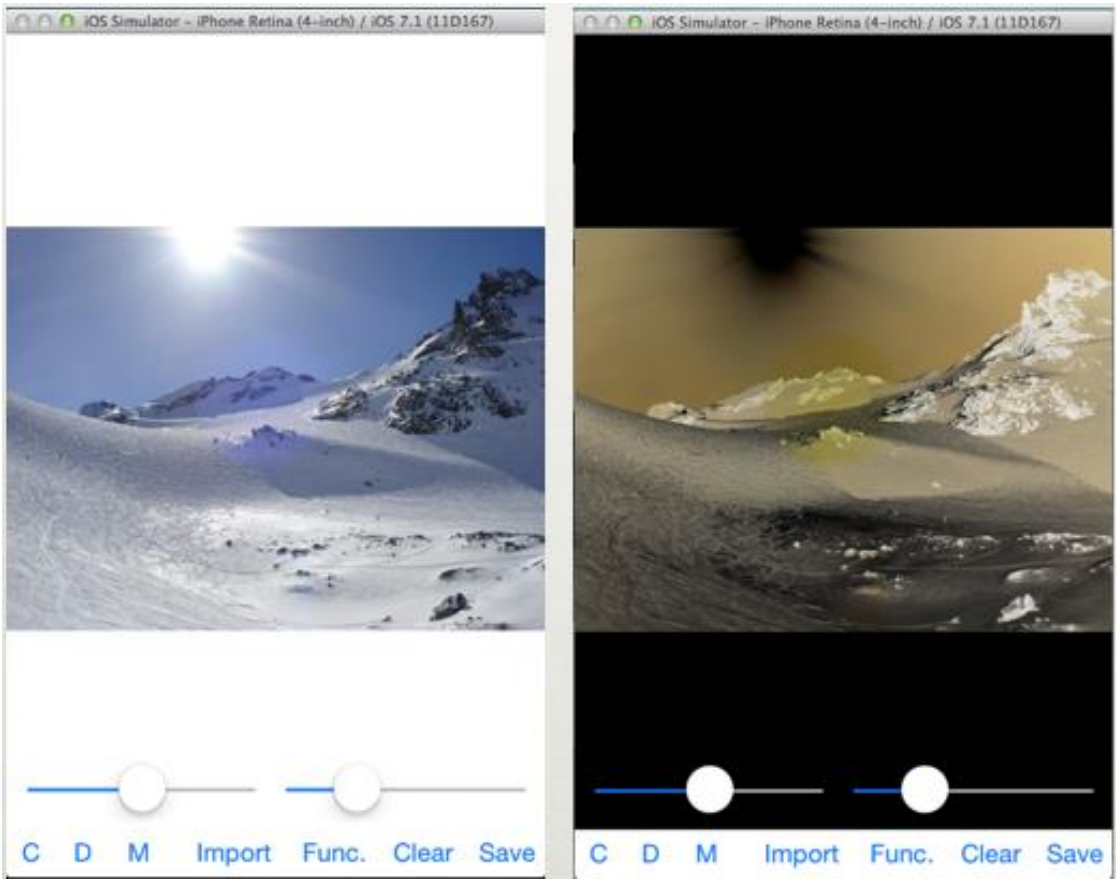

Figure 6 - Comparison between original image and inverted image

Create Histogram - There are two main aspects of creating a histogram. First, the data must be looped through and all of the pixel data must be read. Secondly, there needs to be a method to take this pixel data and display it in a logical format. To do this, I first create a .txt file that has the values 0 through 255, each on its own separate row. As the pixels are looped through, each piece of data is sorted into its corresponding value and this value on the histogram is incremented. At the end, the .txt file shows the all of the RGB data and exactly how many pixels of each value there are in the image (Figure 7). For example, line 18 of Figure 7 means that there are 120 pixels that have a red value of 18, 23 pixels that have a green value of 18, and one pixel that has a blue value of 18.

I used Image J to validate the accuracy of the histogram. Looking at the raw counts for each of the pixel values is not sufficient, as the numbers do not match up for the same image. One reason why this can occur is due to differences in the functions that read the pixel values between programs. Additionally, importing images into different applications can lead to different resolutions and slight data differences, which result in the raw pixel count data being slightly different.

Image J has a frequency graph ranging from 0 to 255 for each of the color channels that can be utilized. By taking the raw data from my application and graphing it along the same x-axis as Image J, it is clear that the histogram created by my application matches that of Image J. Figure 8 shows the comparison of my application with Image J for each of the three color channels. The similarities between the graphs on the left and right for all three color channels are convincing that my

application is developing a legitimate histogram of the image that can be used in other processing aspects of my application.

PixelValues.txt

```
Histogram (R / G / B) --- 0 (darkest) thru 255 (lightest)
0 (black): 0 / 0 / 0
1: 1 / 0 / 0
2: 0 / 0 / 0
3: 3 / 0 / 0
4: 0 / 0 / 0
5: 5 / 0 / 0
6: 8 / 0 / 0
7: 7 / 0 / 0
8: 5 / 0 / 0
9: 16 / 0 / 0
10: 18 / 1 / 0
11: 27 / 0 / 0
12: 29 / 1 / 0
13: 28 / 4 / 0
14: 58 / 4 / 0
15: 60 / 10 / 0
16: 89 / 11 / 0
17: 95 / 21 / 0
18: 120 / 23 / 1
19: 149 / 33 / 0
20: 157 / 57 / 2
21: 184 / 66 / 0
22: 209 / 99 / 3
23: 229 / 110 / 5
24: 218 / 130 / 6
25: 257 / 176 / 7
26: 257 / 161 / 25
27: 306 / 210 / 18
28: 301 / 265 / 27
```

Figure 7 - Histogram generated by the application. There are a total of 256 rows (0 through 255), each showing each of the R / G / B data.

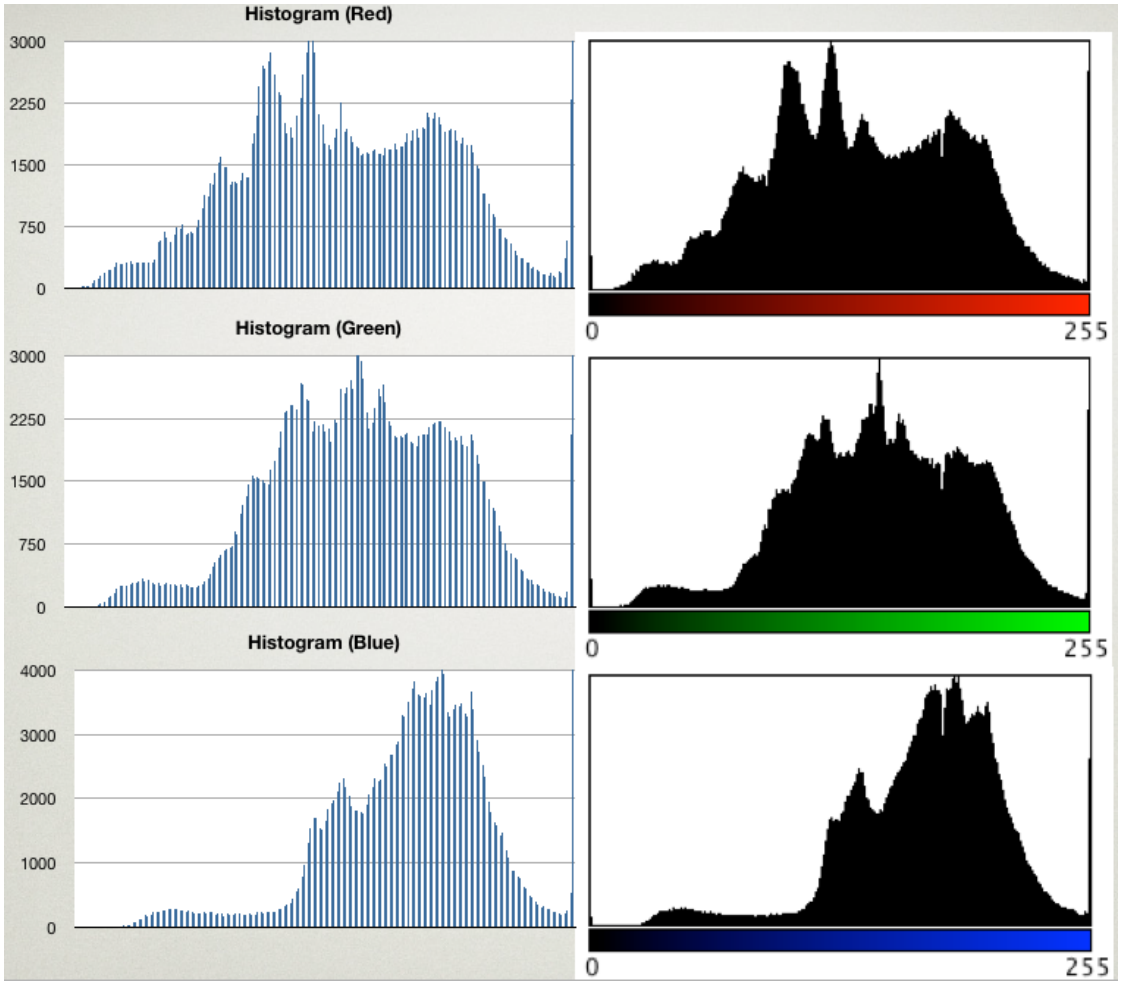

Figure 8 - Comparison of histograms from my application (left side) and Image J (right side). The top images show the red pixel data, the middle images show the green pixel data, the bottom images show the blue pixel data.

Reflect Vertically - To reflect the image vertically, I am splitting the image into two halves, top and bottom. I begin at the top half and store all of the pixel data into a new pixel array for later use. Then I rewrite this top half pixel data with the bottom half pixel data that corresponds to the vertically reflected image. Finally, I rewrite the bottom half pixel data with the corresponding original top half pixel data

which has been stored in the new array. Figure 9 depicts an image that has been reflected. The image on the right side is the result of vertically reflecting the center image.

Reflect Horizontally - To reflect the image horizontally, I am using the same method that I used to reflect the image vertically. This time, I split the image into left and right halves and go on to rewrite both halves of data with the corresponding pixels from the other half. The image on the left side of Figure 9 is the result of horizontally reflecting the center image.

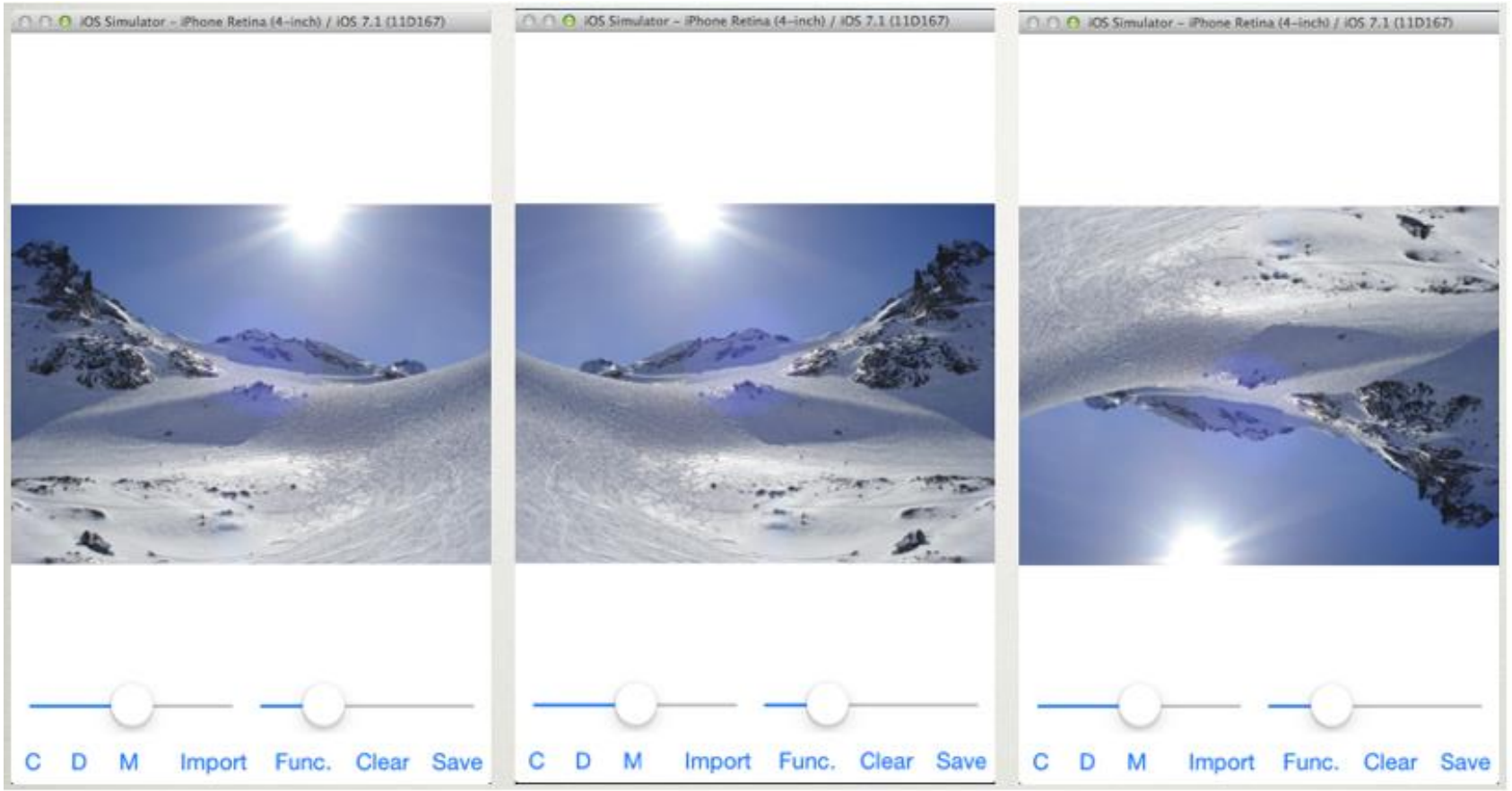

Figure 9 - Images on the left and center have been reflected horizontally. Images on the right and center have been reflected vertically.

Clear - The clear button simply resets the image view. The image is set to nil, which means that it will clear the screen and the user will just see a blank screen.

Save - The save function takes the current image view and saves it to the camera roll. The useful part of this is that the images are saved to the same location that they are imported from.

Brightness Control - The brightness control slider allows the user to dynamically change the brightness of the image. As the slider is moved, the image is continuously updated, using a built-in Xcode brightness function. The slider ranges from a value of negative one to positive, with the initial value being zero. At negative one, the image will appear to be black and at positive one, the image will appear to be white. When the image is brightened, the pixel values are being shifted upwards (towards 255) and when the image is darkened, the pixel values are shifted downwards. I compared an image brightened by my application with the same image brightened by Image J (Figure 10). The image in the center is brightened by my application and the image on the right is brightened by Image J. While there are visible differences between the images, the result is similar.

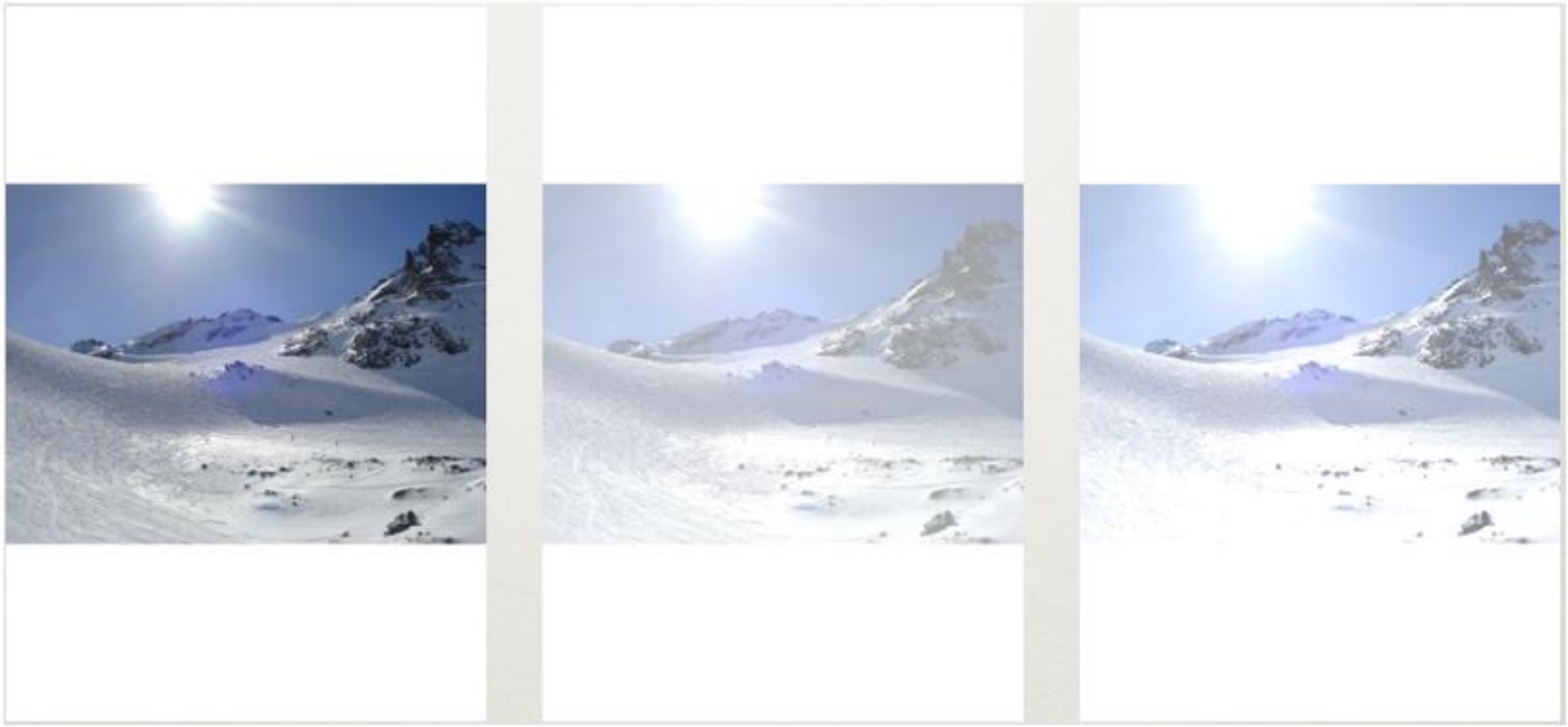

Figure 10 - Image on the left is the original. Image in the center is brightened using my application. Image on the right is brightened using Image J.

Contrast Control - The contrast control slider acts the same way as the brightness control slider, as the image is dynamically changed as the user moves the slider. For this feature, the slider ranges from a value of zero to a value of four, with the initial value being one. At zero contrast, the image will appear uniformly gray, while at four contrast, the image will have much larger differences between the colors. Although there is no upper limit on contrast, I set the maximum value to four, as the image begins to become more saturated when contrast is very high. When contrast is increased, the pixel data is being stretched out across the range of values, which creates a larger difference between pixels with different values. When contrast is decreased, the pixel data is being compressed, to the point of everything being the same value at zero contrast. Figure 11 is a comparison of an image with increased contrast using my application with Image J. The image in the center is from my application and the image on the right is from Image J. The main difference between the two is that my application tends to darken the image as contrast is increased. However, because I have separate sliders for brightness and contrast, this can be compensated for by increasing the brightness. Figure 12 shows an image where both brightness and contrast have been increased. Again, the image in the center is from my application and the image on the right is from Image J.

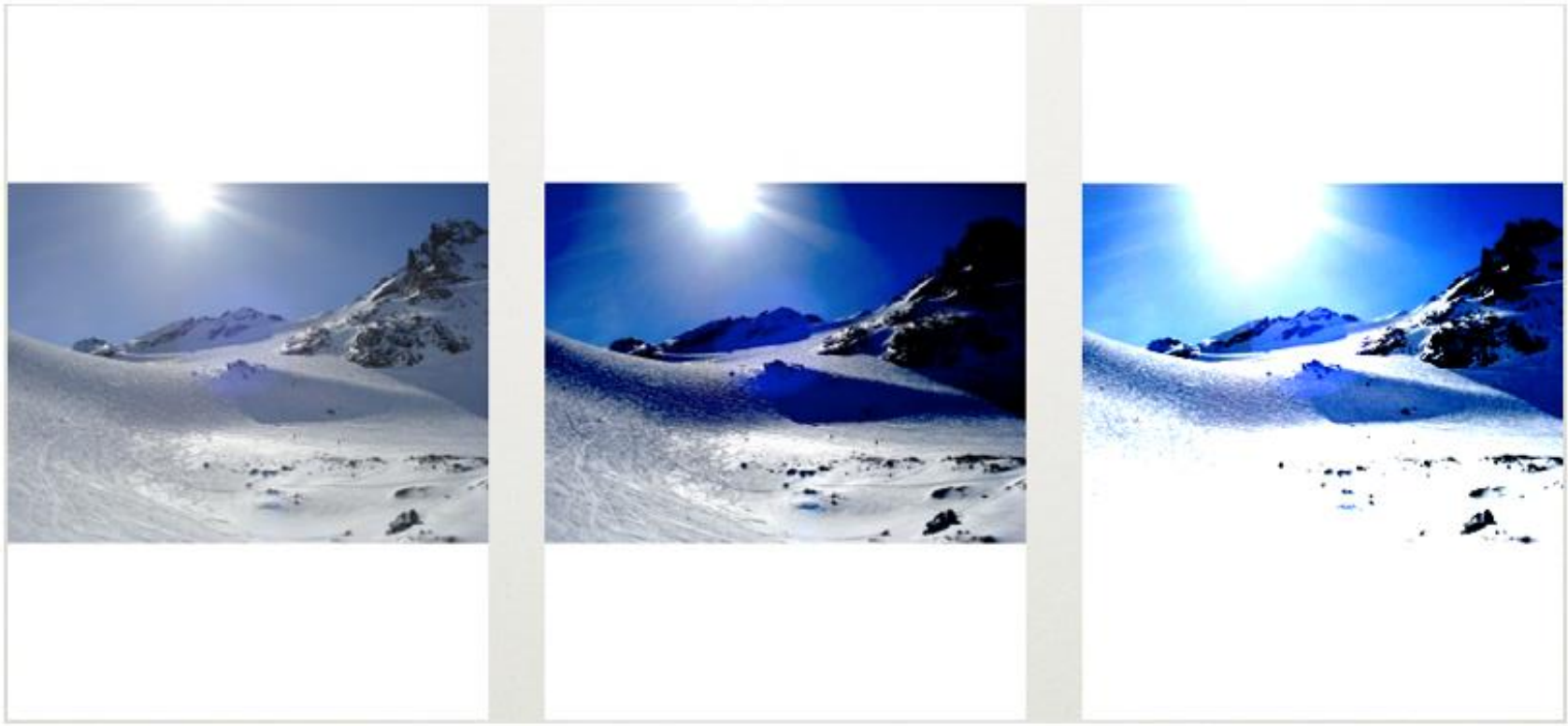

Figure 11 - Image on the left is the original. Contrast of the image in the center is increased using my application. Contrast of the image on the right is increased using Image J.

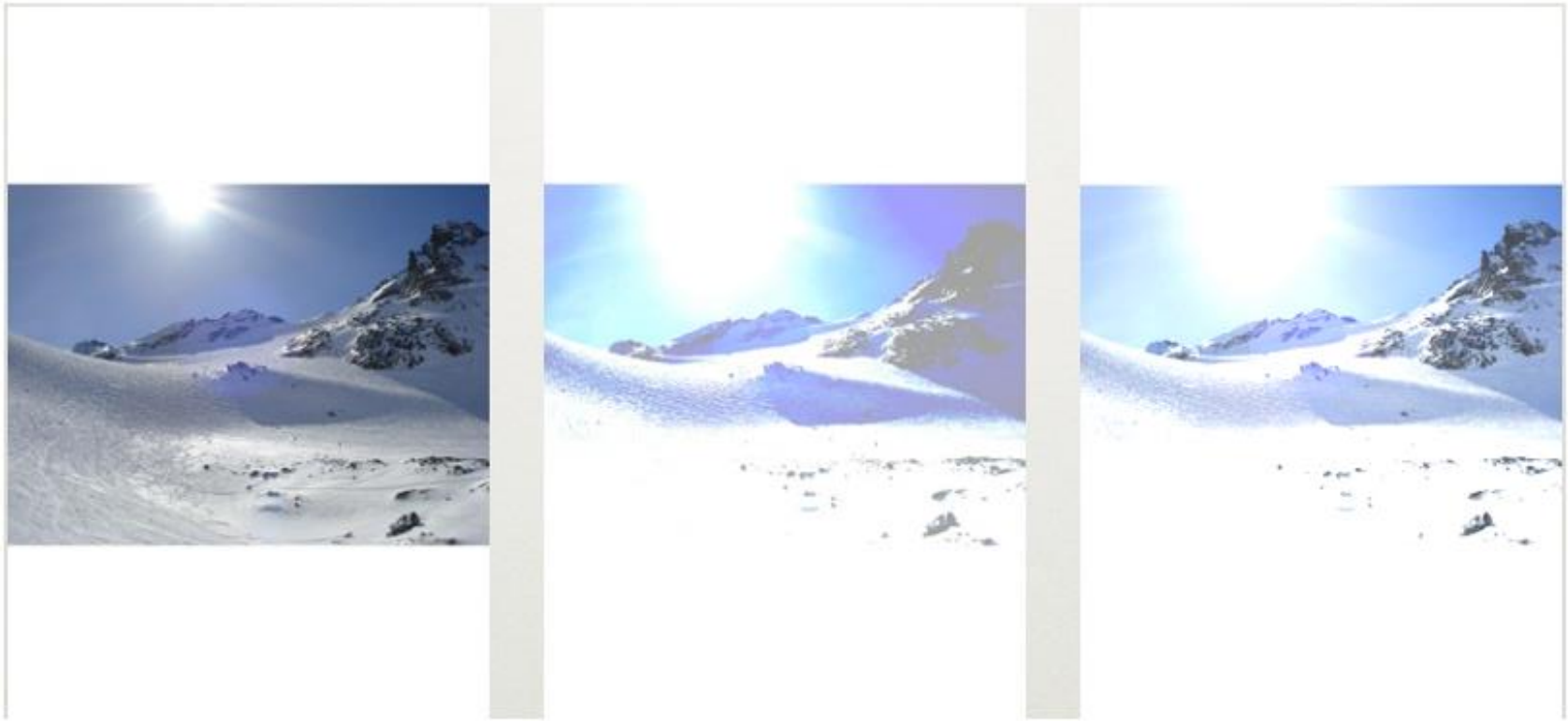

Figure 12 - Image on the left is the original. Image in the center has both brightness and contrast increased by my application. Image on the right has both brightness and contrast increased by Image J.

CHAPTER 2 - Crop View Controller

When the user navigates to the Crop View Controller, the current image is automatically loaded into the image view of the new screen (Figure 13). From this point on, the image can be further modified and saved back to the camera roll.

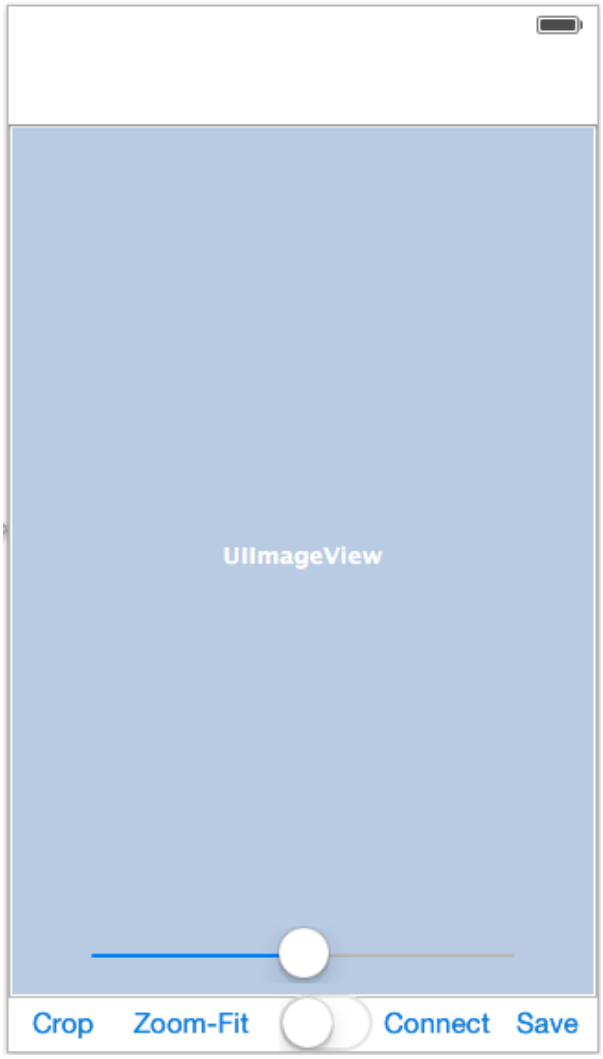

Figure 13 - Screenshot of the crop view controller

Area Selection (Default at Rectangular Selection) - As a default, when the user clicks and drags anywhere in the image view, a rectangle is drawn onto the

image. Only one rectangle can be drawn on the image at a time, so if the user is unhappy with the rectangle placed on the image, as soon as another point is clicked, the first rectangle will disappear and a new one can be drawn on the image. The purpose of this feature is to be used in tandem with the crop feature. The image on the left side of Figure 14 shows an image after the user has dragged a rectangle onto the screen.

Crop - After selecting a region of the image, the crop feature cuts out the rest of the image, leaving just the selected area. To my knowledge, the mechanism of cropping is unique. The crop works with the area selection, as the original image is first overlapped with the image with the area selection drawn on. At this point, the difference between the two images is calculated, creating a third image, which just contains the border of the area selection. At this point, a binary image that contains the area selection is created so that the inside of the area is black while the outside is white. To do this, the program begins at the left edge of the image, and turns everything white until it hits the border. The same is done beginning at each of the four edges, leaving the image white outside of the border and black inside and on the border. Finally, this binary image is used to cycle back through the original image to crop it. A simple statement is used where if the binary image pixel is white, the original image pixel is set to white, while if the binary image pixel is black, the original image pixel is left the same. Figure 14 shows the resulting image after it is cropped. Everywhere outside of the user selected region is converted to white, while the inside of the region remains the same.

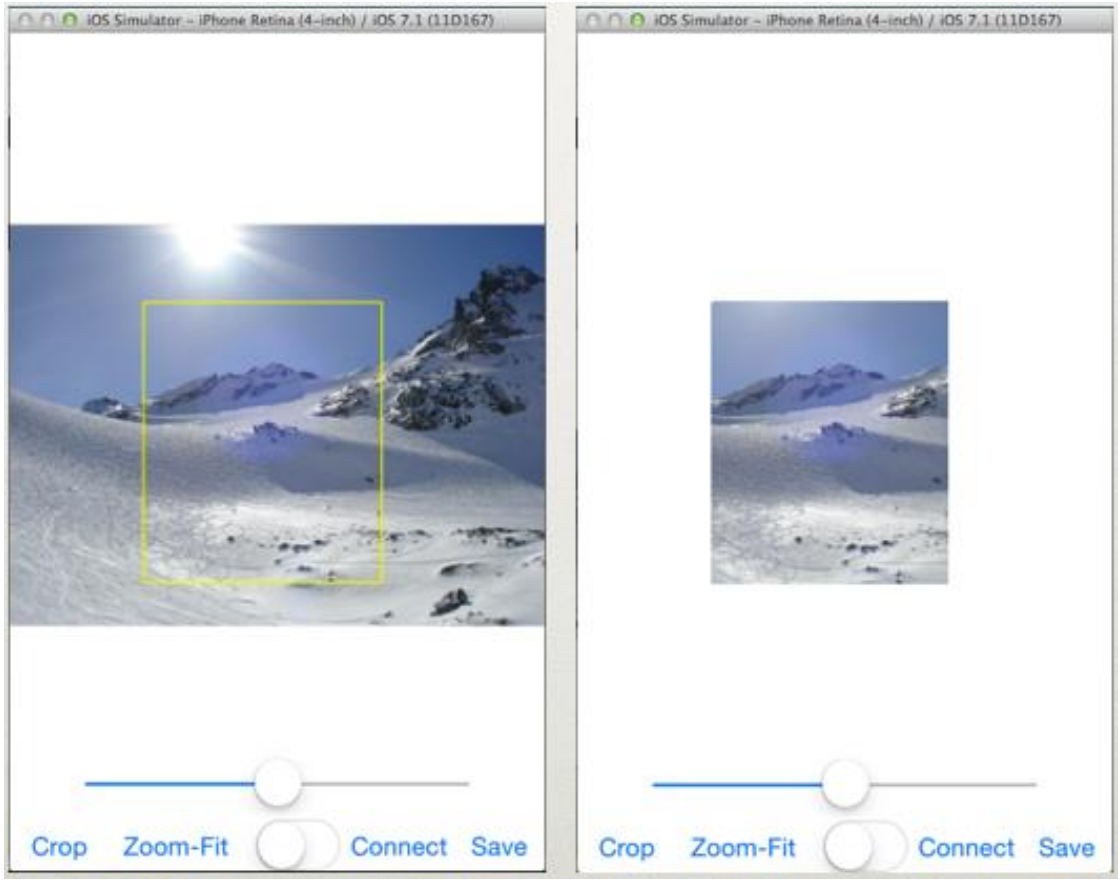

Figure 14 - Image on the left is after the user has selected the region. Image on the right is after the user has cropped.

Zoom-Fit - This function is intended to be used after cropping the image. Because the image will be smaller after cropping, the zoom-fit resizes the image to use the most of the screen. Figure 15 shows an image that has been cropped and then zoom-fit. One of the results of stretching the image out to fill the screen is a loss of resolution in the image. The amount of data is directly proportional to the size of the user selected region, meaning that if the user selects a very small region, the resulting zoomed in image will be very blurry.

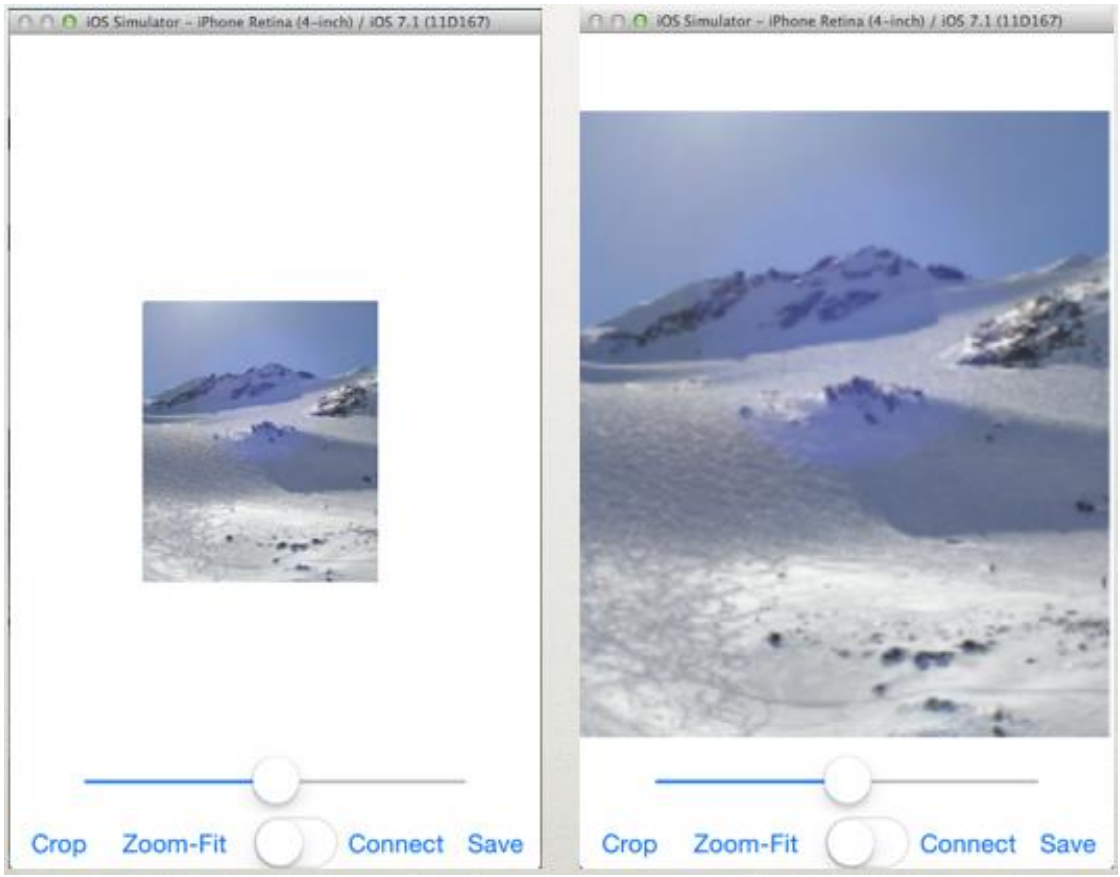

Figure 15 - Image on the left is after the user has cropped. Image on the right is after the user has zoom-fit.

Toggle Switch (Rectangular vs Polygon Area Selection) - This switch allows the user to switch from the default rectangular area selection to a more flexible polygonal area selection. When the switch is toggled on, each click on the screen leaves a vertex on the screen that is connected to the previous vertex. The number of vertices is unconstrained, giving the user flexibility for the size and shape of the area selection. The polygonal area selection is intended to be used in tandem with the connect function, as once the final vertex is selected, the user can simply click "connect" which will close off the polygon by connecting the last point with the first

point. Figure 16 shows the resulting image after it is cropped with the switch toggled on. The user can now define a complex polygonal region and only the region of the image inside this will remain after being cropped.

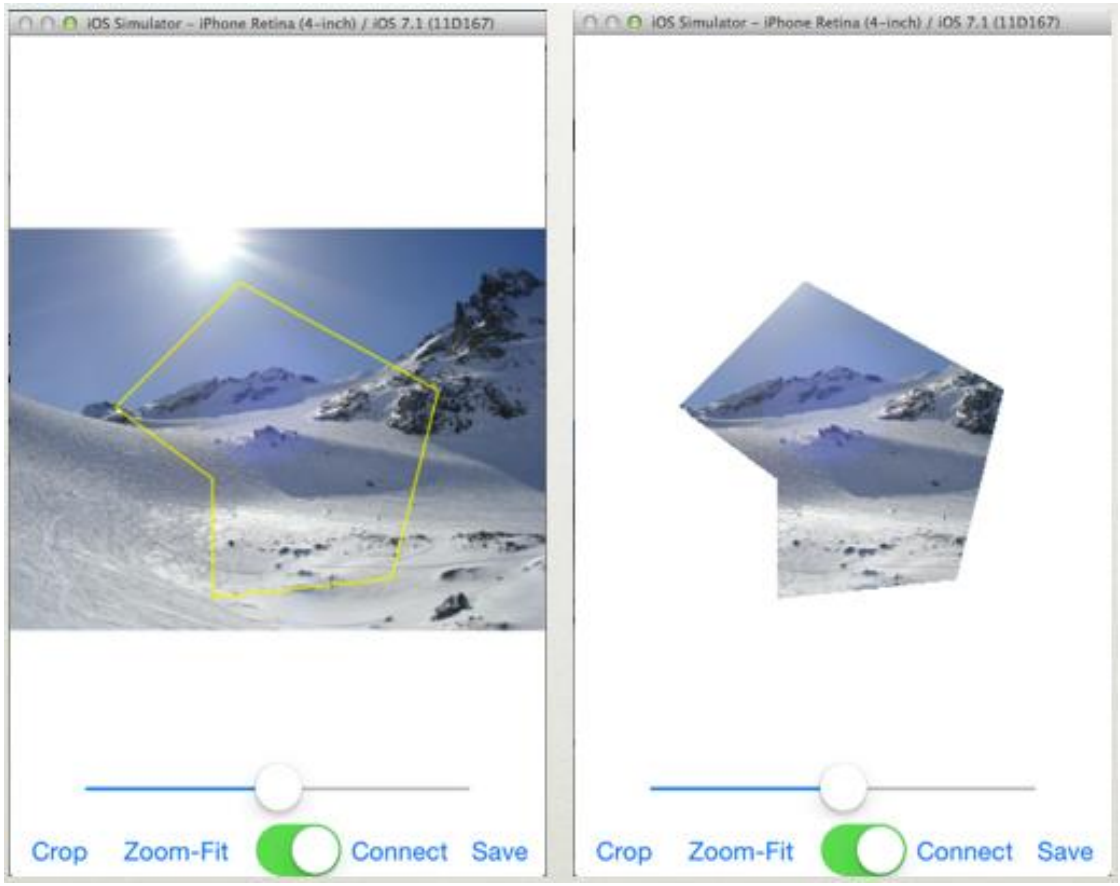

Figure 16 - Image on the left is after the user has selected the region. Image on the right is after the user has cropped.

Connect - As previously mentioned, this function is used when the user is selecting a polygonal region. The last vertex is connected to the first vertex, which closes off the polygon, making it ready to be cropped.

Save - The save function takes the current image view and saves it to the camera roll. This means that the image can then be reloaded back to the main view controller so that functions in other locations can then be accessed.

Rotation Slider - The rotation slider allows the user to rotate the image to any angle. If the slider is dragged to the left, the image is rotated counterclockwise up to 180 degrees and if the slider is dragged to the right, the image is rotated clockwise up to 180 degrees. The image is always rotated with respect to the top edge and the left edge of the frame. The corner on the left edge slides straight up or down and the corner on the top edge slides straight right or left. The other two corners adjust accordingly, which causes the image to rotate. Depending on the size of the image, this means that some of the image may be offscreen on the right hand side when the image is rotated. Figure 17 shows an image that has been rotated to different extents. The middle image has been slightly rotated clockwise and the right image has been rotated clockwise to almost the full 180 degrees.

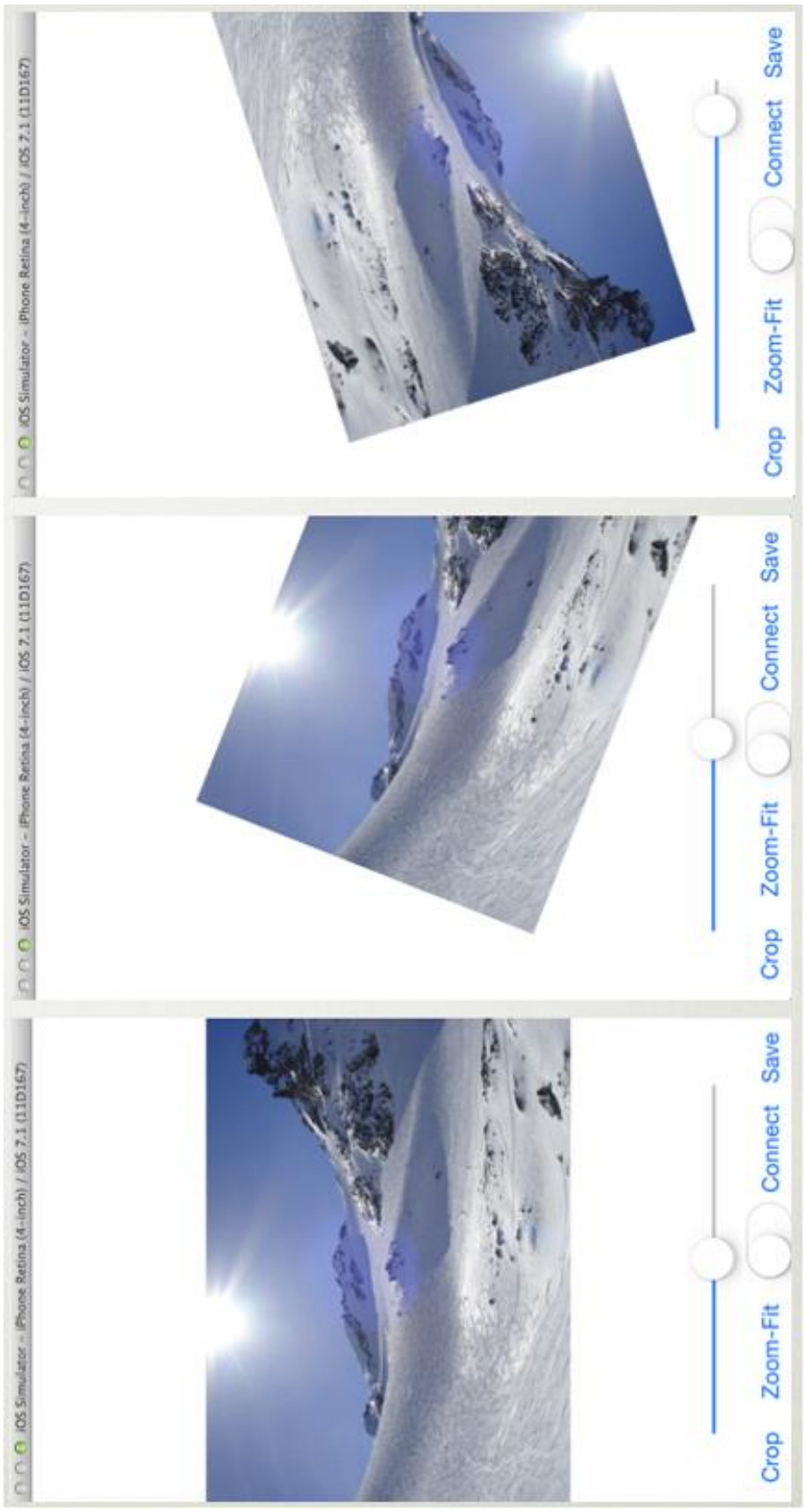

Figure 17 - Image on the left has no rotation. Image in the center has been slightly rotated clockwise. Image on the right has been drastically rotated clockwise.

CHAPTER 3 - Draw View Controller

In the draw view controller, the user can freely draw on the image by clicking anywhere on the screen and dragging (Figure 18). Currently, there is only one color, black, that can be drawn on the image and the user can click and drag as many times as needed, each one leaving the particular drawing.

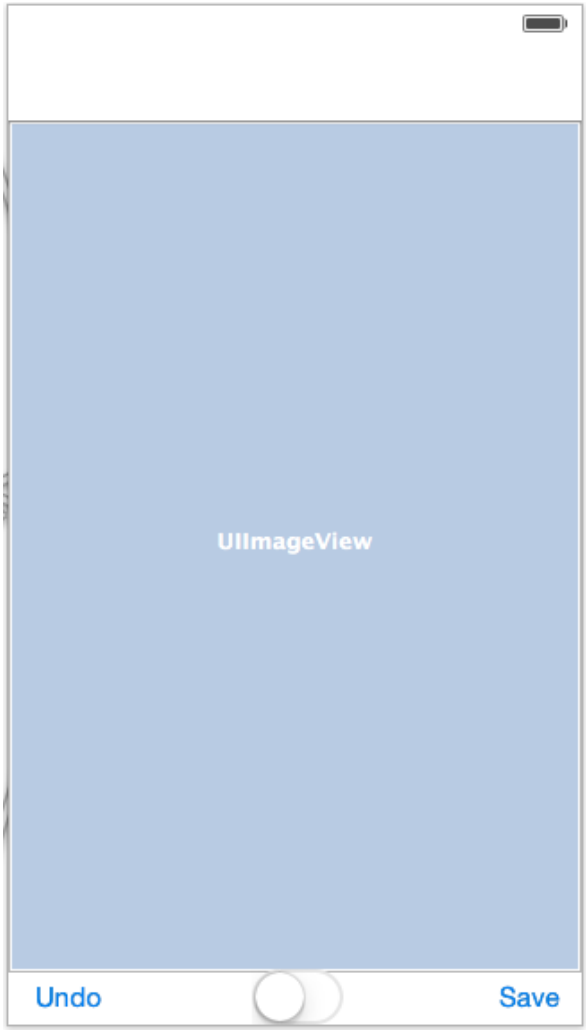

Figure 18 - Screenshot of the draw view controller

Undo - This function undoes the most recent stroke that has been drawn on the image. It only retains the information from one stroke previous, so the user cannot repeatedly undo multiple strokes that have been drawn.

Toggle Switch (Eraser) - When the switch is toggled on, the line strokes are white and thicker than the default freehand drawing. This acts like an eraser, as it can be quickly used to erase large areas of the image. The left image of Figure 19 shows an image after it has been drawn on and the right image shows that same image after a portion has been erased.

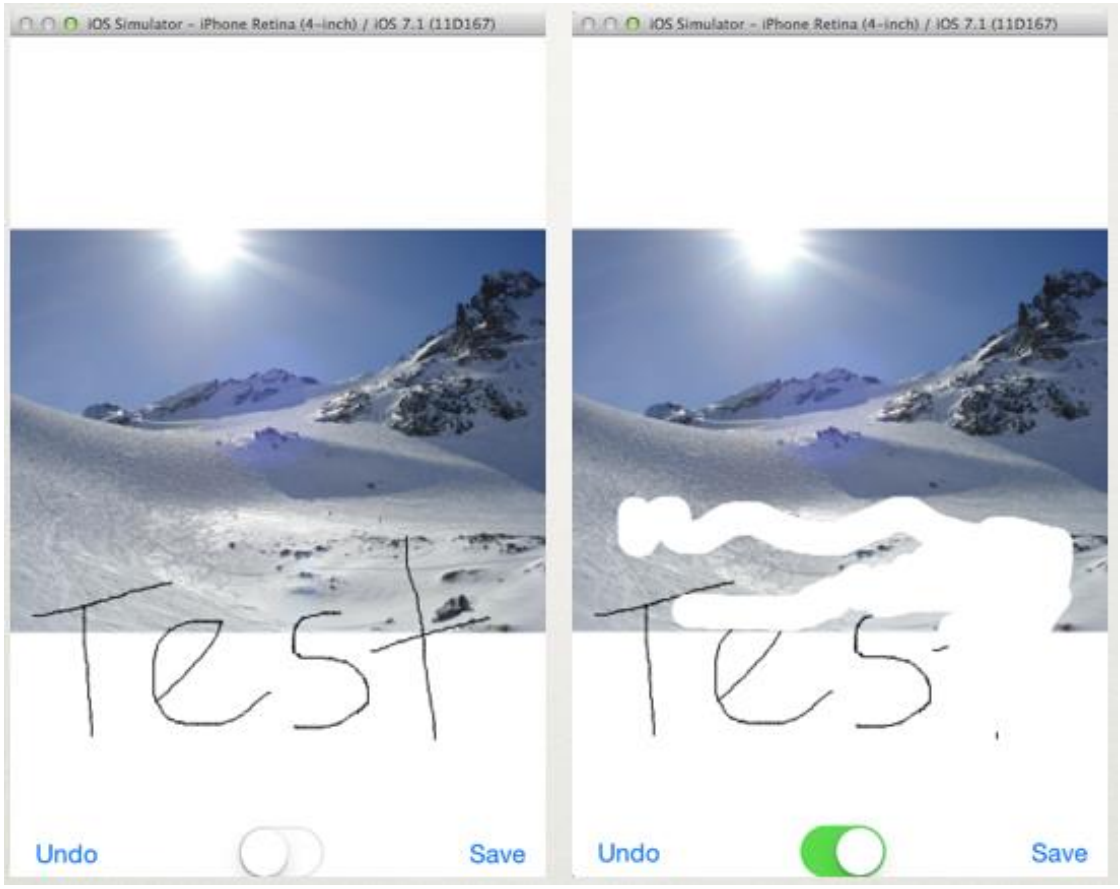

Figure 19 - Image on the left is after the draw function has been used to write “test”. Image on the right is after the eraser function has been used.

Save - Again, as in the previous sections, the save function takes the current image view and saves it to the camera roll.

CHAPTER 4 - Measure View Controller

The Measure View Controller is where length, area, and count measurements can be taken (Figure 20). One of the novel things about this application is that the length and area measurements can be calibrated by the user. By doing this, instead of length and area readings that are arbitrary numbers that correspond to number of pixels, the user can directly see the calibrated values of the measurements.

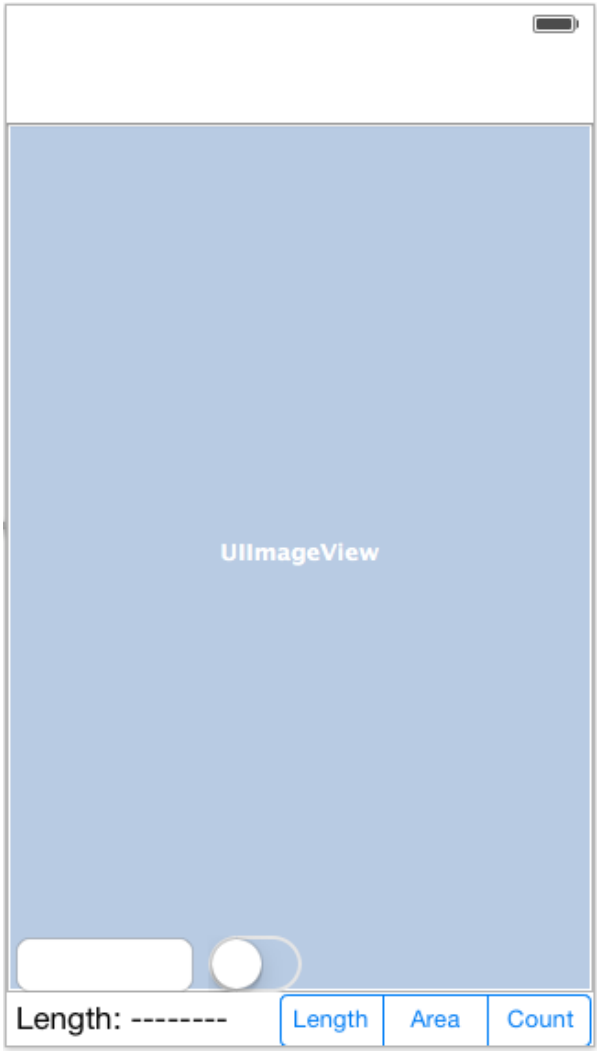

Figure 20 - Screenshot of the measure view controller

Calibration Field / Switch - This feature is used with the length measurement to calibrate the length and area measurements to a given unit. After a length is drawn on the image and the pixel measurement is displayed, the user can enter a number into the text field and toggle the switch. Once the switch is toggled on, the distance and area measurements will now be sown in relation to this calibrated value. This is intended to be used with images that have a scale bar. With a scale bar, it is easy for the user to match the length to the scale bar and calibrate the measurements to this scale.

Length - The length option measures the straight line distance between two points. The user clicks and drags a line segment onto the image, and the distance is displayed. As a default, the distance is shown in number of pixels, however, this can be changed to a unit length using the calibrate feature. Figure 21 shows the resulting measurements after the length has been previously calibrated to one inch. The left image shows a line segment that is approximately 0.5 inches and the right image shows a line segment that is approximately 1.5 inches.

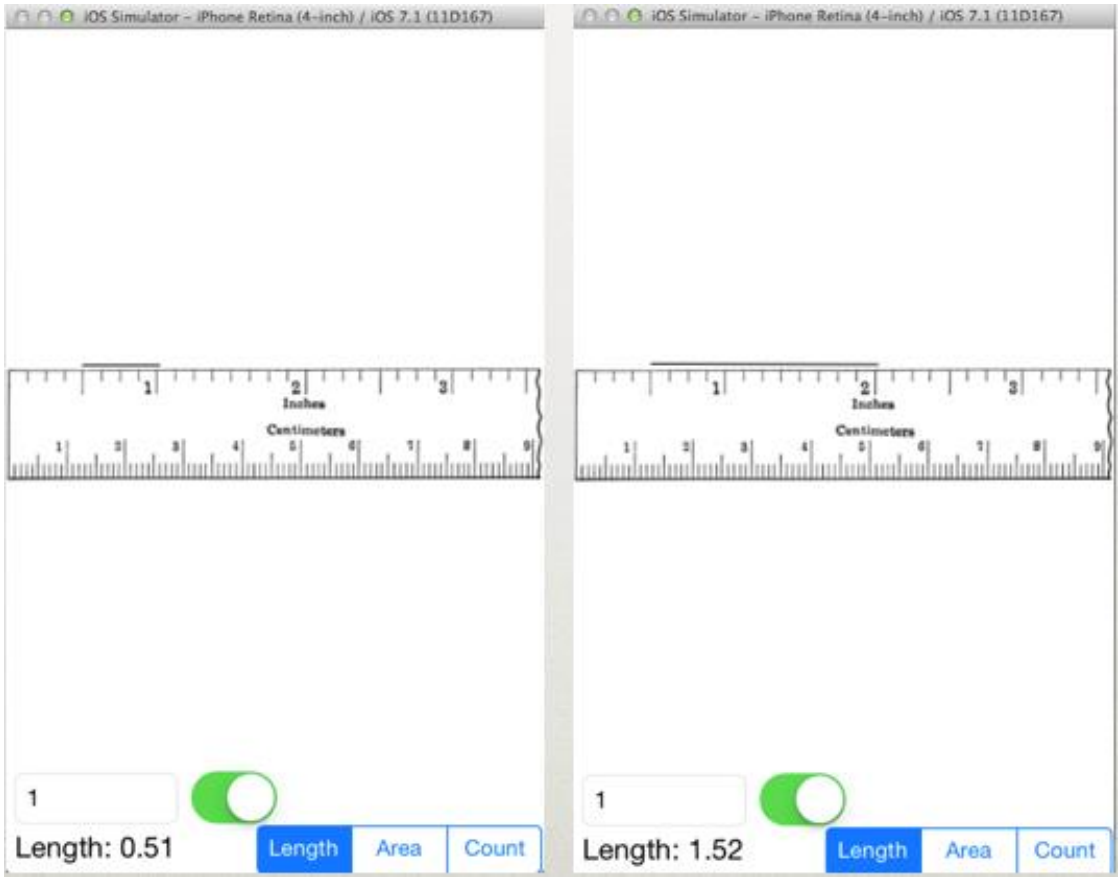

Figure 21 - Length measurements on an image that has been calibrated to one inch. The left image shows a line segment approximately 0.5 inches and the right image shows a line segment that is approximately 1.5 inches.

Area - The area option measures the area of a user selected rectangle. The user clicks and drags to form the rectangle, and the area is displayed. Similar to the length option, the default is that the area is shown in number of pixels. Figure 22 shows the resulting area measurements after the length has been calibrated so that each small box is a unit of one. The left image encompasses an area that is approximately 4

units by 4 units and the right images encompasses an area that is approximately 4 units by 15 units.

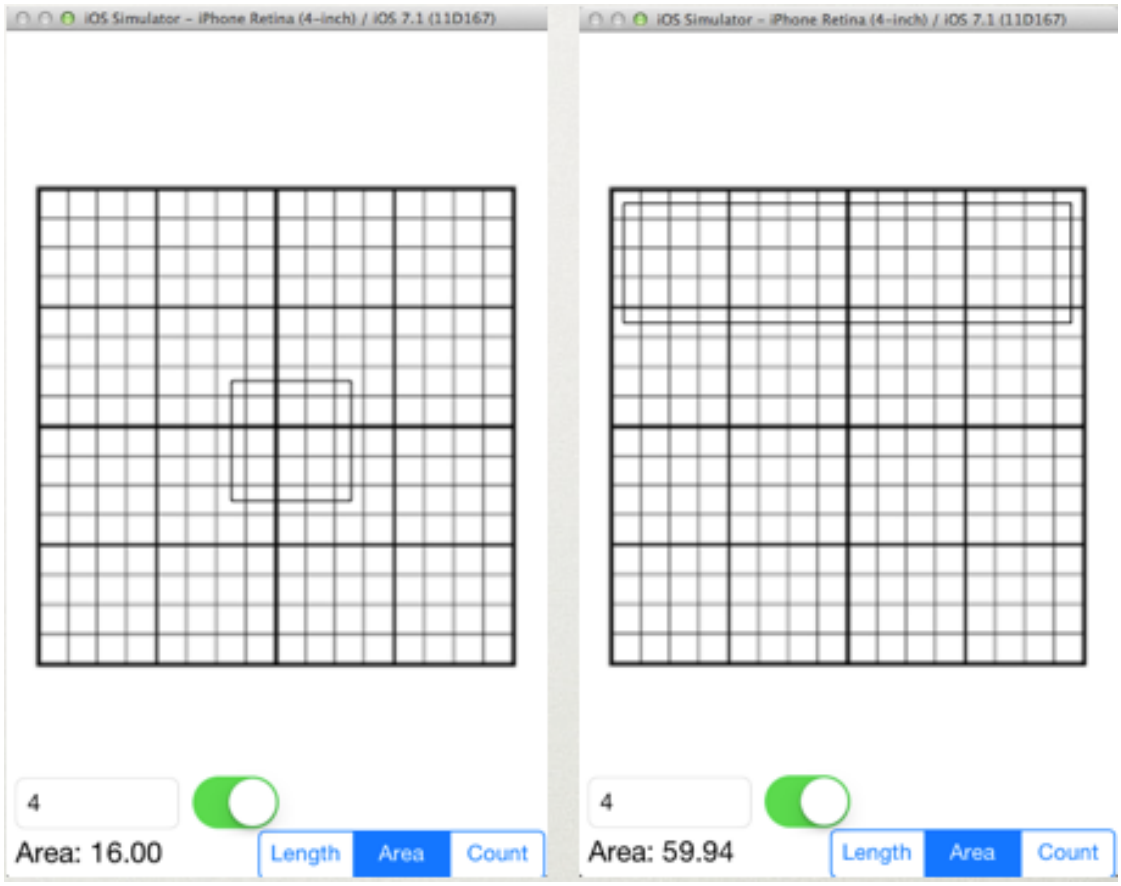

Figure 22 - Area measurements on an image that has been calibrated to one unit per box. Image on the left is approximately 16 boxes and image on the right is approximately 60 boxes.

Count - The count allows the user to manually count the number of items of interest by simply tapping on the screen. Each tap will increment the count meter by one, and this continues until the count meter is reset by clicking on the button again.

CHAPTER 5 - Conclusion/Future Directions

My application provides a solid foundation for a mobile scientific image processing application. Although the scope of my project is much smaller than current image processing applications on the market, it provides a launching point to build on. There are ways to improve most of the existing features to make the application more user friendly which is one of the future directions of the project. Specific future directions include:

- Making the polygonal crop more user friendly by allowing the user to freehand draw the region of interest
- Allowing the user to freely zoom in and out using two finger gesture (spreading or squeezing two fingers)
- Developing a more sophisticated way of displaying the histogram instead of displaying all of the raw data. This can be similar to to graphs generated by Image J (Figure 8)
- Developing a more sophisticated drawing tool that includes the ability to select line color and thickness.
- Creating a automated counting mechanism in addition to the manual counting system.

Additionally, many other custom modules can be created to add on to the application to allow for specific user needs. This process of continually building upon the application is very common to keep up with new user needs.

APPENDIX 1 - DrawPad Code Snippet

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {

    mouseSwiped = NO;
    UITouch *touch = [touches anyObject];
    lastPoint = [touch locationInView:self.view];
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {

    mouseSwiped = YES;
    UITouch *touch = [touches anyObject];
    CGPoint currentPoint = [touch locationInView:self.view];

    UIGraphicsBeginImageContext(self.view.frame.size);
    [self.tempDrawImage.image drawInRect:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height)];
    CGContextMoveToPoint(UIGraphicsGetCurrentContext(), lastPoint.x, lastPoint.y);
    CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);
    CGContextSetLineCap(UIGraphicsGetCurrentContext(), kCGLineCapRound);
    CGContextSetLineWidth(UIGraphicsGetCurrentContext(), brush );
    CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), red, green, blue,
1.0);

CGContextSetBlendMode(UIGraphicsGetCurrentContext(),kCGBlendModeNormal);

    CGContextStrokePath(UIGraphicsGetCurrentContext());
    self.tempDrawImage.image = UIGraphicsGetImageFromCurrentImageContext();
    [self.tempDrawImage setAlpha:opacity];
    UIGraphicsEndImageContext();

    lastPoint = currentPoint;
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {

    if(!mouseSwiped) {
        UIGraphicsBeginImageContext(self.view.frame.size);
        [self.tempDrawImage.image drawInRect:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height)];
        CGContextSetLineCap(UIGraphicsGetCurrentContext(), kCGLineCapRound);
        CGContextSetLineWidth(UIGraphicsGetCurrentContext(), brush);
```

```
        CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), red, green,
blue, opacity);
        CGContextMoveToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
        CGContextStrokePath(UIGraphicsGetCurrentContext());
        CGContextFlush(UIGraphicsGetCurrentContext());
        self.tempDrawImage.image = UIGraphicsGetImageFromCurrentImageContext();
        UIGraphicsEndImageContext();
    }

    UIGraphicsBeginImageContext(self.mainImage.frame.size);
    [self.mainImage.image drawInRect:CGRectMake(0, 0, self.view.frame.size.width,
self.view.frame.size.height) blendMode:kCGBlendModeNormal alpha:1.0];
    [self.tempDrawImage.image drawInRect:CGRectMake(0, 0,
self.view.frame.size.width, self.view.frame.size.height)
blendMode:kCGBlendModeNormal alpha:opacity];
    self.mainImage.image = UIGraphicsGetImageFromCurrentImageContext();
    self.tempDrawImage.image = nil;
    UIGraphicsEndImageContext();
}

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {

    SettingsViewController * settingsVC = (SettingsViewController
*)segue.destinationViewController;
    settingsVC.delegate = self;
    settingsVC.brush = brush;
    settingsVC.opacity = opacity;
    settingsVC.red = red;
    settingsVC.green = green;
    settingsVC.blue = blue;

}
```

APPENDIX 2 - Image Processing Code Snippet

```
if (theSwitch.on){

    CGImageRef sourceImage = theImage.image.CGImage;

    CFDataRef theData;
    theData = CGDataProviderCopyData(CGImageGetDataProvider(sourceImage));

    UInt8 *pixelData = (UInt8 *) CFDataGetBytePtr(theData);

    int dataLength = CFDataGetLength(theData);

    int red = 0;
    int green = 1;
    int blue = 2;

    for (int index = 0; index < dataLength; index += 4) {
      if (pixelData[index + green] - 128 > 0) {
            pixelData[index + red] = pixelData[index + green] - 128;
            pixelData[index + blue] = pixelData[index + green] - 128;
      } else {
            pixelData[index + red] = 0;
            pixelData[index + blue] = 0;
      }
    }

    CGContextRef context;
    context = CGBitmapContextCreate(pixelData,
                    CGImageGetWidth(sourceImage),
                    CGImageGetHeight(sourceImage),
                    8,
                    CGImageGetBytesPerRow(sourceImage),
                    CGImageGetColorSpace(sourceImage),
                    kCGImageAlphaPremultipliedLast);

    CGImageRef newCGImage = CGBitmapContextCreateImage(context);
    UIImage *newImage = [UIImage imageWithCGImage:newCGImage];

    CGContextRelease(context);
    CFRelease(theData);
    CGImageRelease(newCGImage);

    theImage.image = newImage;
```

```
} else {
        theImage.image = [UIImage imageNamed:@"storm-at-sea.jpg"];
}
```

APPENDIX 3 - My Application Source Code

```
//  MDSAppDelegate.h
//  Project_v1
//
//  Created by Michael Skinner on 10/20/14.
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface MDSAppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@end


//
//  MDSAppDelegate.m
//  Project_v1
//
//  Created by Michael Skinner on 10/20/14.
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import "MDSAppDelegate.h"

@implementation MDSAppDelegate

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Override point for customization after application launch.
    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application
{
    // Sent when the application is about to move from active to inactive state. This can
occur for certain types of temporary interruptions (such as an incoming phone call or
SMS message) or when the user quits the application and it begins the transition to the
background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down
OpenGL ES frame rates. Games should use this method to pause the game.
```

```
}

- (void)applicationDidEnterBackground:(UIApplication *)application
{
    // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
    // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
}

- (void)applicationWillEnterForeground:(UIApplication *)application
{
    // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
}

- (void)applicationWillTerminate:(UIApplication *)application
{
    // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
}

@end

//
//  MDSViewController.h
//  Project_v1
//
//  Created by Michael Skinner on 10/20/14.
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
#import "MDSCropViewController.h"
#import "MDSDrawViewController.h"
```

```
#import "MDSMeasureViewController.h"


@interface MDSViewController : UIViewController
<MDSCropViewControllerDelegate, MDSDrawViewControllerDelegate,
MDSMeasureViewControllerDelegate, UINavigationControllerDelegate,
UIImagePickerControllerDelegate, UIActionSheetDelegate>{
   CGPoint lastPoint;
   CGPoint currentPoint;
   BOOL mouseSwiped;
   CGSize saveSize;
}

@property (weak, nonatomic) IBOutlet UIImageView *theImage;
@property (strong, nonatomic) IBOutlet UIImageView *scrollView;
@property (strong, nonatomic) IBOutlet UIImageView *window;
@property (strong, nonatomic) IBOutlet UISlider *contrastSlider;
@property (strong, nonatomic) IBOutlet UISlider *brightnessSlider;


@end

//
//  MDSViewController.m
//  Project_v1
//
//  Created by Michael Skinner on 10/20/14.
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import "MDSViewController.h"
#import <AssetsLibrary/AssetsLibrary.h>
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>

@interface MDSViewController ()

@end

@implementation MDSViewController {
   UIImageOrientation orientation;
   UIImage* originalImage;
}

@synthesize theImage;
```

```
@synthesize scrollView;
@synthesize brightnessSlider;
@synthesize contrastSlider;

float brightness = 0;
float contrast = 1;

- (void)viewDidLoad
{
    [super viewDidLoad];
    saveSize = theImage.bounds.size;
}

- (void)viewDidUnload
{
    [self setTheImage:nil];
    [super viewDidUnload];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (void)centerScrollViewContents {
    CGSize boundsSize = self.scrollView.bounds.size;
    CGRect contentsFrame = self.theImage.frame;

    if (contentsFrame.size.width < boundsSize.width) {
        contentsFrame.origin.x = (boundsSize.width - contentsFrame.size.width) / 2.0f;
    } else {
        contentsFrame.origin.x = 0.0f;
    }

    if (contentsFrame.size.height < boundsSize.height) {
        contentsFrame.origin.y = (boundsSize.height - contentsFrame.size.height) / 2.0f;
    } else {
        contentsFrame.origin.y = 0.0f;
    }

    self.theImage.frame = contentsFrame;
}

- (IBAction)importImage:(id)sender {
```

```
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];
    picker.sourceType = UIImagePickerControllerSourceTypeSavedPhotosAlbum;
    picker.delegate = self;
    [self presentViewController:picker animated:YES completion:nil];
}

-(void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info {
    UIImage* image = [info objectForKey:UIImagePickerControllerOriginalImage];
    self.theImage.image = image;
    originalImage = image;
    [picker dismissViewControllerAnimated:YES completion:nil];
}

-(void)imagePickerControllerDidCancel:(UIImagePickerController *)picker {
    [picker dismissViewControllerAnimated:YES completion:nil];
}


- (IBAction)save:(id)sender {
    UIGraphicsBeginImageContextWithOptions(saveSize, NO, 0.0);
    [self.window.layer renderInContext:UIGraphicsGetCurrentContext()];

    UIImage *SaveImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    UIImageWriteToSavedPhotosAlbum(SaveImage,
self,@selector(image:didFinishSavingWithError:contextInfo:), nil);


/*
    UIGraphicsBeginImageContextWithOptions(self.theImage.bounds.size, NO, 0.0);
    [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];
    UIImage *SaveImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    UIImageWriteToSavedPhotosAlbum(SaveImage,
self,@selector(image:didFinishSavingWithError:contextInfo:), nil);
 */
}


- (void)image:(UIImage *)image didFinishSavingWithError:(NSError *)error
contextInfo:(void *)contextInfo
{
    if (error != NULL)
```

```
    {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error"
message:@"Image could not be saved.Please try again"  delegate:nil
cancelButtonTitle:nil otherButtonTitles:@"Close", nil];
        [alert show];
    } else {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Success"
message:@"Image was successfully saved in photoalbum"  delegate:nil
cancelButtonTitle:nil otherButtonTitles:@"Close", nil];
        [alert show];
    }
}

- (IBAction)reset:(id)sender {
    self.theImage.image = nil;
    [brightnessSlider setValue:0];
    [contrastSlider setValue:1];
    brightness = 0;
    contrast = 1;
}

- (IBAction)settings:(id)sender {
    UIActionSheet *actionSheet = [[UIActionSheet alloc] initWithTitle:@""
                                                             delegate:self
                                                    cancelButtonTitle:nil
                                               destructiveButtonTitle:nil
                                                    otherButtonTitles:@"Grayscale", @"Binary",
@"Invert", @"Create Histogram", @"Reflect Vertically", @"Reflect Horizontally",
@"Cancel", nil];
    [actionSheet showInView:self.view];

}

- (void)actionSheet:(UIActionSheet *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex
{
    if (buttonIndex == 0) //Grayscale
    {
        CGImageRef sourceImage = theImage.image.CGImage;
        CFMutableDataRef theData;
        theData = CFDataCreateMutableCopy(0, 0,
CGDataProviderCopyData(CGImageGetDataProvider(sourceImage)));
        UInt8 *pixelData = (UInt8 *) CFDataGetMutableBytePtr(theData);
//        CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
        int dataLength = CFDataGetLength(theData);
```

```
        int red = 0;
        int green = 1;
        int blue = 2;

        for (int index = 0; index < dataLength; index += 4) {
            int avg = (.3*pixelData[index+red] + .59*pixelData[index+green] +
.11*pixelData[index+blue]); //Luminosity Method of Grayscaling
            pixelData[index+red] = avg;
            pixelData[index+green] = avg;
            pixelData[index+blue] = avg;
        }

        CGContextRef context;
        context = CGBitmapContextCreate(pixelData,
                                        CGImageGetWidth(sourceImage),
                                        CGImageGetHeight(sourceImage),
                                        8,
                                        CGImageGetBytesPerRow(sourceImage),
                                        CGImageGetColorSpace(sourceImage),
                                        (CGBitmapInfo)kCGImageAlphaNoneSkipLast);

        CGImageRef newCGImage = CGBitmapContextCreateImage(context);
        UIImage *newImage = [UIImage imageWithCGImage:newCGImage];
        CGContextRelease(context);
        CFRelease(theData);
        CGImageRelease(newCGImage);
        theImage.image = newImage;
        originalImage = theImage.image;
    }

    if (buttonIndex == 1) //Binary
    {
        CGImageRef sourceImage = theImage.image.CGImage;
        CFMutableDataRef theData;
        theData = CFDataCreateMutableCopy(0, 0,
CGDataProviderCopyData(CGImageGetDataProvider(sourceImage)));
        UInt8 *pixelData = (UInt8 *) CFDataGetMutableBytePtr(theData);
        CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
        int dataLength = CFDataGetLength(theData);
        int red = 0;
        int green = 1;
        int blue = 2;

        int pixelArrayValue[256] = {};
        float lowerMean, upperMean, totalVariance;
```

```
        float maxVariance = 0;
        int totalSum = 0;
        int lowerSum = 0;
        int lowerWeight = 0;
        int upperWeight;
        int numPixels;
        int threshold = 0;

        for (int index = 0; index < dataLength; index += 4) {
            for (int i = 0; i < 255; i++)
                if (i == pixelData[index]) {
                    pixelArrayValue[i]++;
                }
        }

        //OTSU'S METHOD
        //http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

        for (int i = 0; i < 256; i++)
        {
            totalSum += pixelArrayValue[i] * i;
            numPixels += pixelArrayValue[i];
        }

        for (int j = 0; j < 256; j++)
        {
            lowerWeight += pixelArrayValue[j]; //Number of pixels below the threshold

            if (lowerWeight == 0)
                continue;

            upperWeight = numPixels - lowerWeight;  //Number of pixels above the
threshold

            if (upperWeight == 0)
                break;

            lowerSum += pixelArrayValue[j] * j;
            lowerMean = lowerSum / lowerWeight;
            upperMean = (totalSum - lowerSum) / upperWeight;

            //interclass variance
            totalVariance = (float)lowerWeight * (float)upperWeight * (lowerMean -
upperMean) * (lowerMean - upperMean);
```

```
            if (totalVariance > maxVariance)
            {
                maxVariance = totalVariance;
                threshold = j;
            }
        }



        for (int index = 0; index < dataLength; index += 4) {
            if (pixelData[index + red] > threshold) {
                pixelData[index + red] = 255;
                pixelData[index + green] = 255;
                pixelData[index + blue] = 255;
            } else {
                pixelData[index + red] = 0;
                pixelData[index + green] = 0;
                pixelData[index + blue] = 0;
            }
        }

        CGContextRef context;
        context = CGBitmapContextCreate(pixelData,
                                        CGImageGetWidth(sourceImage),
                                        CGImageGetHeight(sourceImage),
                                        8,
                                        CGImageGetBytesPerRow(sourceImage),
                                        colorSpace,
                                        (CGBitmapInfo)kCGImageAlphaNoneSkipLast);

        CGImageRef newCGImage = CGBitmapContextCreateImage(context);
        UIImage *newImage = [UIImage imageWithCGImage:newCGImage];
        CGContextRelease(context);
        CFRelease(theData);
        CGImageRelease(newCGImage);
        theImage.image = newImage;
    }

    if (buttonIndex == 2) //Invert
    {
        CGImageRef sourceImage = theImage.image.CGImage;
        CFMutableDataRef theData;
        theData = CFDataCreateMutableCopy(0, 0,
CGDataProviderCopyData(CGImageGetDataProvider(sourceImage)));
```

```
        UInt8 *pixelData = (UInt8 *) CFDataGetMutableBytePtr(theData);
        CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
        int dataLength = CFDataGetLength(theData);
        int red = 0;
        int green = 1;
        int blue = 2;

        for (int index = 0; index < dataLength; index += 4) {
            pixelData[index + red] = 255 - pixelData[index + red];
            pixelData[index + green] = 255 - pixelData[index + green];
            pixelData[index + blue] = 255 - pixelData[index + blue];
        }

        CGContextRef context;
        context = CGBitmapContextCreate(pixelData, CGImageGetWidth(sourceImage),
CGImageGetHeight(sourceImage), 8, CGImageGetBytesPerRow(sourceImage),
colorSpace,(CGBitmapInfo)kCGImageAlphaNoneSkipLast);

        CGImageRef newCGImage = CGBitmapContextCreateImage(context);
        UIImage *newImage = [UIImage imageWithCGImage:newCGImage];
        CGContextRelease(context);
        CFRelease(theData);
        CGImageRelease(newCGImage);
        theImage.image = newImage;
        originalImage = theImage.image;
    }
    if (buttonIndex == 3) //Create Histogram
    {
        CGImageRef sourceImage = theImage.image.CGImage;
        CFMutableDataRef theData;
        theData = CFDataCreateMutableCopy(0, 0,
CGDataProviderCopyData(CGImageGetDataProvider(sourceImage)));
        UInt8 *pixelData = (UInt8 *) CFDataGetMutableBytePtr(theData);
        int dataLength = CFDataGetLength(theData);

        NSMutableString* outputString = [NSMutableString
stringWithCapacity:dataLength/4];
        int pixelArrayRed[256] = {};
        int pixelArrayGreen[256] = {};
        int pixelArrayBlue[256] = {};

        for (int index = 0; index < dataLength; index += 4) {
            for (int i = 0; i < 256; i ++)
                if (i == pixelData[index]) {
                    pixelArrayRed[i]++;
```

```
            }
        }
        for (int index = 1; index < dataLength; index += 4) {
            for (int i = 0; i < 256; i ++)
                if (i == pixelData[index]) {
                    pixelArrayGreen[i]++;
                }
        }

        for (int index = 2; index < dataLength; index += 4) {
            for (int i = 0; i < 256; i ++)
                if (i == pixelData[index]) {
                    pixelArrayBlue[i]++;
                }
        }

        [outputString appendFormat:@"Histogram (R / G / B) --- 0 (darkest) thru 255
(lightest) \n"];

        for (int x = 0; x<256; x++){
            if (x==0) {
                [outputString appendFormat:@"%d (black): %d / %d / %d \n", x,
pixelArrayRed[x], pixelArrayGreen[x], pixelArrayBlue[x]];
            } else if (x==255){
                [outputString appendFormat:@"%d (white): %d / %d / %d \n", x,
pixelArrayRed[x], pixelArrayGreen[x], pixelArrayBlue[x]];
            }
            else {
            [outputString appendFormat:@"%d: %d / %d / %d \n", x, pixelArrayRed[x],
pixelArrayGreen[x], pixelArrayBlue[x]];
//          [outputString appendFormat:@"%d \n", pixelArrayRed[x]];

            }
        }

        NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
        NSString *documentsDirectory = [paths objectAtIndex:0];
        NSString *filePath =  [documentsDirectory
stringByAppendingPathComponent:@"PixelValues.txt"];
        NSError *error;
        [outputString writeToFile:filePath atomically:YES
encoding:NSUTF8StringEncoding error:&error];
    }
```

```
    if (buttonIndex == 4) //VERTICAL FLIP
    {
        CGImageRef sourceImage = theImage.image.CGImage;
        CFMutableDataRef theData;
        theData = CFDataCreateMutableCopy(0, 0,
CGDataProviderCopyData(CGImageGetDataProvider(sourceImage)));
        UInt8 *pixelData = (UInt8 *) CFDataGetMutableBytePtr(theData);
        CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();

        int dataLength = CFDataGetLength(theData);
        int red = 0;
        int green = 1;
        int blue = 2;
        int horizontal = 4*CGImageGetWidth(sourceImage);
        int vertical = CGImageGetHeight(sourceImage);
        int pixelDataCopy[dataLength/2];

        for (int i = 0; i < horizontal; i += 4)
        {
            for (int j = 0; j < vertical/2; j += 1)
            {
                pixelDataCopy[j*horizontal + i + red] = pixelData[j*horizontal + i + red];
                pixelDataCopy[j*horizontal + i + green] = pixelData[j*horizontal + i +
green];
                pixelDataCopy[j*horizontal + i + blue] = pixelData[j*horizontal + i + blue];

                pixelData[j*horizontal + i + red] = pixelData[(vertical-j)*horizontal + i +
red];
                pixelData[j*horizontal + i + green] = pixelData[(vertical-j)*horizontal + i +
green];
                pixelData[j*horizontal + i + blue] = pixelData[(vertical-j)*horizontal + i +
blue];
            }
        }
        for (int i = 0; i < horizontal; i += 4)
        {
            for (int j = vertical/2 + 1; j < vertical; j += 1)
            {
                pixelData[j*horizontal + i + red] = pixelDataCopy[(vertical-j)*horizontal + i
+ red];
                pixelData[j*horizontal + i + green] = pixelDataCopy[(vertical-j)*horizontal
+ i + green];
                pixelData[j*horizontal + i + blue] = pixelDataCopy[(vertical-j)*horizontal +
i + blue];
```

```
            }
        }

        CGContextRef context;
        context = CGBitmapContextCreate(pixelData, CGImageGetWidth(sourceImage),
CGImageGetHeight(sourceImage), 8, CGImageGetBytesPerRow(sourceImage),
colorSpace,(CGBitmapInfo)kCGImageAlphaNoneSkipLast);

        CGImageRef newCGImage = CGBitmapContextCreateImage(context);
        UIImage *newImage = [UIImage imageWithCGImage:newCGImage];
        CGContextRelease(context);
        CFRelease(theData);
        CGImageRelease(newCGImage);
        theImage.image = newImage;
        originalImage = theImage.image;
    }

    if (buttonIndex == 5) //HORIZONTAL FLIP
    {
        CGImageRef sourceImage = theImage.image.CGImage;
        CFMutableDataRef theData;
        theData = CFDataCreateMutableCopy(0, 0,
CGDataProviderCopyData(CGImageGetDataProvider(sourceImage)));
        UInt8 *pixelData = (UInt8 *) CFDataGetMutableBytePtr(theData);
        CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();

        int dataLength = CFDataGetLength(theData);
        int red = 0;
        int green = 1;
        int blue = 2;
        int horizontal = 4*CGImageGetWidth(sourceImage);
        int vertical = CGImageGetHeight(sourceImage);
        int pixelDataCopy[dataLength/2];

        for (int i = 0; i < horizontal/2; i += 4)
        {
            for (int j = 0; j < vertical; j += 1)
            {
                pixelDataCopy[4*j + (i)*vertical + red] = pixelData[j*horizontal + i + red];
                pixelDataCopy[4*j + (i)*vertical + green]  = pixelData[j*horizontal + i +
green];
                pixelDataCopy[4*j + (i)*vertical + blue]  = pixelData[j*horizontal + i +
blue];
```

```
            pixelData[j*horizontal + i + red] = pixelData[j*horizontal + horizontal - i - 4
+ red];
            pixelData[j*horizontal + i + green] = pixelData[j*horizontal + horizontal - i
- 4 + green];
            pixelData[j*horizontal + i + blue] = pixelData[j*horizontal + horizontal - i -
4 + blue];
        }
    }
    for (int i = horizontal/2 + 4; i < horizontal; i += 4)
    {
        for (int j = 0; j < vertical; j += 1)
        {
            pixelData[j*horizontal + i + red] = pixelDataCopy[4*j + ((horizontal-
i))*vertical + red];
            pixelData[j*horizontal + i + green] = pixelDataCopy[4*j + ((horizontal-
i))*vertical + green];
            pixelData[j*horizontal + i + blue] = pixelDataCopy[4*j + ((horizontal-
i))*vertical + blue];
        }
    }

    CGContextRef context;
    context = CGBitmapContextCreate(pixelData, CGImageGetWidth(sourceImage),
CGImageGetHeight(sourceImage), 8, CGImageGetBytesPerRow(sourceImage),
colorSpace,(CGBitmapInfo)kCGImageAlphaNoneSkipLast);

    CGImageRef newCGImage = CGBitmapContextCreateImage(context);
    UIImage *newImage = [UIImage imageWithCGImage:newCGImage];
    CGContextRelease(context);
    CFRelease(theData);
    CGImageRelease(newCGImage);
    theImage.image = newImage;
    originalImage = theImage.image;
  }
}

- (IBAction)brightnessChanged:(UISlider *)slider {
  CGImageRef sourceImage = originalImage.CGImage;
  CIContext *context = [CIContext contextWithOptions:nil];
  CIImage *image = [CIImage imageWithCGImage:(sourceImage)];

  float slideValue = slider.value;

  CIImage *outputImage = [self oldPhoto:image withAmount:(slideValue-
brightness)];
```

```
    CGImageRef cgimg = [context createCGImage:outputImage
                                     fromRect:[outputImage extent]];

    UIImage *newImage = [UIImage imageWithCGImage:cgimg scale:1.0
orientation:orientation];
    self.theImage.image = newImage;

    CGImageRelease(cgimg);

    [slider addTarget:self action:@selector(sliderReleased:)
     forControlEvents:UIControlEventTouchUpInside];
}

- (IBAction)contrastChanged:(UISlider *)slider {
    CGImageRef sourceImage = originalImage.CGImage;
    CIContext *context = [CIContext contextWithOptions:nil];
    CIImage *image = [CIImage imageWithCGImage:(sourceImage)];

    float slideValue = slider.value;

    CIImage *outputImage = [self oldPhoto2:image withAmount:(slideValue-
contrast+1)];
    CGImageRef cgimg = [context createCGImage:outputImage
                                     fromRect:[outputImage extent]];

    UIImage *newImage = [UIImage imageWithCGImage:cgimg scale:1.0
orientation:orientation];
    self.theImage.image = newImage;

    CGImageRelease(cgimg);

    [slider addTarget:self action:@selector(sliderReleased:)
     forControlEvents:UIControlEventTouchUpInside];
}

- (void)sliderReleased:(id)sender {
    originalImage = theImage.image;
    contrast = contrastSlider.value;
    brightness = brightnessSlider.value;
//    [brightnessSlider setValue:0];
//    [contrastSlider setValue:1];
}

//Brightness
-(CIImage *)oldPhoto:(CIImage *)img withAmount:(float)intensity {
```

```
    CIFilter *lighten = [CIFilter filterWithName:@"CIColorControls"];
    [lighten setValue:img forKey:kCIInputImageKey];
    [lighten setValue:@(intensity) forKey:@"inputBrightness"];

    return lighten.outputImage;
}

//Contrast
-(CIImage *)oldPhoto2:(CIImage *)img withAmount:(float)intensity {

    CIFilter *contrast = [CIFilter filterWithName:@"CIColorControls"];
    [contrast setValue:img forKey:kCIInputImageKey];
    [contrast setValue:@(intensity) forKey:@"inputContrast"];

    return contrast.outputImage;
}

//Sharpness
-(CIImage *)oldPhoto3:(CIImage *)img withAmount:(float)intensity {

    CIFilter *sharpness = [CIFilter filterWithName:@"CISharpenLuminance"];
    [sharpness setValue:img forKey:kCIInputImageKey];
    [sharpness setValue:@(intensity-1) forKey:kCIInputSharpnessKey];

    return sharpness.outputImage;
}

- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {

    if ([[segue identifier] isEqualToString:@"cropSegue"])
    {
        MDSCropViewController * cropVC = (MDSCropViewController
*)segue.destinationViewController;
        cropVC.delegate = self;
        cropVC.cropImage = theImage.image;
    }

    if ([[segue identifier] isEqualToString:@"drawSegue"])
    {
        MDSDrawViewController * drawVC = (MDSDrawViewController
*)segue.destinationViewController;
        drawVC.delegate = self;
        drawVC.drawImage = theImage.image;
    }
```

```
    if ([[segue identifier] isEqualToString:@"measureSegue"])
    {
        MDSMeasureViewController * measureVC = (MDSMeasureViewController
*)segue.destinationViewController;
        measureVC.delegate = self;
        measureVC.measureImage = theImage.image;
    }
}

@end

//
//  MDSCropViewController.h
//  Project_v1
//
//  Created by Michael Skinner on 12/9/14.
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import <UIKit/UIKit.h>

@protocol MDSCropViewControllerDelegate <NSObject>
//- (void)closeCrop:(id)sender;
@end

@interface MDSCropViewController : UIViewController
{
    CGPoint lastPoint;
    CGPoint currentPoint;
    CGPoint firstPoint;
    BOOL mouseSwiped;
    CGSize saveSize;
}

@property (weak, nonatomic) id<MDSCropViewControllerDelegate> delegate;

@property (weak, nonatomic) IBOutlet UIImage *cropImage;
@property (strong, nonatomic) IBOutlet UIImageView *theImage;
@property (strong, nonatomic) IBOutlet UIImageView *window;
@property (strong, nonatomic) IBOutlet UISlider *slider;
@property (strong, nonatomic) IBOutlet UISwitch *polygonSwitch;


@end
```

```
//
//  MDSCropViewController.m
//  Project_v1
//
//  Created by Michael Skinner on 12/9/14.
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import "MDSCropViewController.h"

@interface MDSCropViewController ()

@end

@implementation MDSCropViewController

@synthesize cropImage;
@synthesize theImage;
@synthesize polygonSwitch;

int flipSwitch = 0;
int smallestx;
int largestx;
int smallesty;
int largesty;

NSMutableArray *Points;


- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
        // Do any additional setup after loading the view.
}
```

```
- (void)viewDidUnload
{
    [self setTheImage:nil];
    //    [self setOriginalImage:nil];
    [super viewDidUnload];
}


- (void)viewWillAppear:(BOOL)animated {

    self.theImage.image = cropImage;
    saveSize = theImage.bounds.size;
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}


- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    mouseSwiped = NO;
    UITouch *touch = [touches anyObject];
    lastPoint = [touch locationInView:self.theImage];
    if (polygonSwitch.on){
        if (flipSwitch == 1){
            currentPoint = lastPoint;
            firstPoint = currentPoint;
            flipSwitch = 0;
            smallestx = currentPoint.x;
            largestx = currentPoint.x;
            smallesty = currentPoint.y;
            largesty = currentPoint.y;
        } else {

        }

    }
    else{
        currentPoint = lastPoint;
    }
}
```

```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {

    mouseSwiped = YES;
    UITouch *touch = [touches anyObject];
    currentPoint = [touch locationInView:self.theImage];

    if (polygonSwitch.on)
    {}
    else{
    UIGraphicsBeginImageContext(self.theImage.frame.size);
    theImage.image = cropImage;

    [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];

    CGContextMoveToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);

    CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
lastPoint.y);
    CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
    CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
currentPoint.y);
    CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);

    CGContextSetLineCap(UIGraphicsGetCurrentContext(), kCGLineCapRound);
    CGContextSetLineWidth(UIGraphicsGetCurrentContext(), 1 );
    CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), 1, 1, 0, 1);

CGContextSetBlendMode(UIGraphicsGetCurrentContext(),kCGBlendModeNormal);

    CGContextStrokePath(UIGraphicsGetCurrentContext());
    self.theImage.image = UIGraphicsGetImageFromCurrentImageContext();
    [self.theImage setAlpha:1];
    UIGraphicsEndImageContext();
    }
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
//    self.theImage.image = cropImage;

    UIGraphicsBeginImageContext(self.theImage.frame.size);
```

```
    [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];

    CGContextMoveToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);

    if (polygonSwitch.on){

        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
        currentPoint = lastPoint;
        if (currentPoint.x < smallestx){
            smallestx = currentPoint.x;
        }else if (currentPoint.x > largestx){
            largestx = currentPoint.x;
        }
        if (currentPoint.y < smallesty){
            smallesty = currentPoint.y;
        }else if (currentPoint.y > largesty){
            largesty = currentPoint.y;
        }
    }

    CGContextSetLineCap(UIGraphicsGetCurrentContext(), kCGLineCapRound);
    CGContextSetLineWidth(UIGraphicsGetCurrentContext(), 1 );
    CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), 1, 1, 0, 1);

CGContextSetBlendMode(UIGraphicsGetCurrentContext(),kCGBlendModeNormal);

    CGContextStrokePath(UIGraphicsGetCurrentContext());
    self.theImage.image = UIGraphicsGetImageFromCurrentImageContext();
//    [self.theImage setAlpha:1];
    UIGraphicsEndImageContext();
}


- (IBAction)crop:(id)sender {

    UIGraphicsBeginImageContext(self.theImage.frame.size);

    [self.cropImage drawInRect:CGRectMake(0, 0, self.theImage.frame.size.width,
self.theImage.frame.size.height) blendMode:kCGBlendModeNormal alpha:1];
    [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)
blendMode:kCGBlendModeDifference alpha:1];
```

```
    self.theImage.image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();

    {
        CGImageRef sourceImage = theImage.image.CGImage;
        CFMutableDataRef theData;
        theData = CFDataCreateMutableCopy(0, 0,
CGDataProviderCopyData(CGImageGetDataProvider(sourceImage)));
        UInt8 *pixelData = (UInt8 *) CFDataGetMutableBytePtr(theData);

        int dataLength = CFDataGetLength(theData); //608,000
        int red = 0;
        int green = 1;
        int blue = 2;

        int horizontal = 4*CGImageGetWidth(sourceImage); //1280 (in bytes so times 4)
        int vertical = CGImageGetHeight(sourceImage);  //1900 (in pixels)

        //IDEAS:  Left to right, when you hit line take step back and go up and down.
        //Hit line again


        for (int i = 0; i < vertical; i += 1) //Left to Right
        {
            for (int j = 0; j < horizontal; j += 4)
            {
                if (pixelData[i*horizontal + j + red] == 0 && pixelData[i*horizontal + j +
green] == 0 && pixelData[i*horizontal + j + blue] == 0) //Not on line
                {
                    pixelData[i*horizontal + j + red] = 0;
                    pixelData[i*horizontal + j + green] = 255;
                    pixelData[i*horizontal + j + blue] = 0;
                } else { //On line
                    pixelData[i*horizontal + j + red] = 0;
                    pixelData[i*horizontal + j + green] = 0;
                    pixelData[i*horizontal + j + blue] = 0;
                    break;

                }
            }
        }
        for (int i = 0; i < vertical; i += 1) //Right to Left
        {
            for (int j = horizontal; j > 0; j -= 4)
```

```
            {
                if (pixelData[i*horizontal + j + red] == 0 && pixelData[i*horizontal + j +
blue] == 0) //Not on line
                {
                    pixelData[i*horizontal + j + red] = 0;
                    pixelData[i*horizontal + j + green] = 255;
                    pixelData[i*horizontal + j + blue] = 0;
                } else { //On line
                    pixelData[i*horizontal + j + red] = 0;
                    pixelData[i*horizontal + j + green] = 0;
                    pixelData[i*horizontal + j + blue] = 0;
                    break;

                }
            }
        }

        for (int i = 0; i < horizontal; i += 4) //Top to Bottom
        {
            for (int j = 0; j < vertical; j += 1)
            {
                if (pixelData[j*horizontal + i + red] == 0 && pixelData[j*horizontal + i +
blue] == 0)
                {
                    pixelData[j*horizontal + i + red] = 0;
                    pixelData[j*horizontal + i + green] = 255;
                    pixelData[j*horizontal + i + blue] = 0;
                } else {
                    pixelData[j*horizontal + i + red] = 0;
                    pixelData[j*horizontal + i + green] = 0;
                    pixelData[j*horizontal + i + blue] = 0;
                    break;
                }
            }
        }

        for (int i = 0; i < horizontal; i += 4) //Bottom to Top
        {
            for (int j = vertical; j > 0; j -= 1)
            {
                if (pixelData[j*horizontal + i + red] == 0 && pixelData[j*horizontal + i +
blue] == 0)
                {
                    pixelData[j*horizontal + i + red] = 0;
                    pixelData[j*horizontal + i + green] = 255;
```

```
                pixelData[j*horizontal + i + blue] = 0;
            } else {
                pixelData[j*horizontal + i + red] = 0;
                pixelData[j*horizontal + i + green] = 0;
                pixelData[j*horizontal + i + blue] = 0;
                break;
            }
        }
    }

    for (int i = 0; i < dataLength; i += 4)
    {
        if (pixelData[i+green] == 255)
        {
            pixelData[i+red] = 255;
            pixelData[i+blue] = 255;
        }
    }

    CGContextRef context;
    context = CGBitmapContextCreate(pixelData,
    CGImageGetWidth(sourceImage),
    CGImageGetHeight(sourceImage),
    8,
    CGImageGetBytesPerRow(sourceImage),
    CGImageGetColorSpace(sourceImage),
    (CGBitmapInfo)kCGImageAlphaNoneSkipLast);


    CGImageRef newCGImage = CGBitmapContextCreateImage(context);


    UIImage *newImage = [UIImage imageWithCGImage:newCGImage];
    CGContextRelease(context);
    CFRelease(theData);
    CGImageRelease(newCGImage);
    theImage.image = newImage;
  }
  UIGraphicsBeginImageContext(self.theImage.frame.size);
  [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)
blendMode:kCGBlendModeNormal alpha:1];
  [self.cropImage drawInRect:CGRectMake(0, 0, self.theImage.frame.size.width,
self.theImage.frame.size.height) blendMode:kCGBlendModeLighten alpha:1];
```

```
    self.theImage.image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
}


- (IBAction)zoomFit:(id)sender {
    if (polygonSwitch.on){

        CGRect cropRect = CGRectMake(smallestx, smallesty, (largestx - smallestx),
(largesty - smallesty));
        CGImageRef imageRef = CGImageCreateWithImageInRect([theImage.image
CGImage], cropRect);
        theImage.image   = [UIImage imageWithCGImage:imageRef];
        CGImageRelease(imageRef);
    }
    else{
        CGRect cropRect = CGRectMake(lastPoint.x, lastPoint.y, (currentPoint.x -
lastPoint.x), (currentPoint.y - lastPoint.y));
        CGImageRef imageRef = CGImageCreateWithImageInRect([theImage.image
CGImage], cropRect);
        theImage.image   = [UIImage imageWithCGImage:imageRef];
        CGImageRelease(imageRef);
    }
}


- (IBAction)rotateSlider:(id)sender {

    theImage.transform = CGAffineTransformMakeRotation(_slider.value);
}



- (IBAction)save:(id)sender {

    UIGraphicsBeginImageContextWithOptions(saveSize, NO, 0.0);
//    [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];
//   [self.theImage.image drawInRect:CGRectMake(0, 0, 320, 475)];
    [self.window.layer renderInContext:UIGraphicsGetCurrentContext()];

    UIImage *SaveImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    UIImageWriteToSavedPhotosAlbum(SaveImage,
self,@selector(image:didFinishSavingWithError:contextInfo:), nil);
```

```
}

- (void)image:(UIImage *)image didFinishSavingWithError:(NSError *)error
contextInfo:(void *)contextInfo
{
    // Was there an error?
    if (error != NULL)
    {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error"
message:@"Image could not be saved.Please try again"  delegate:nil
cancelButtonTitle:nil otherButtonTitles:@"Close", nil];
        [alert show];
    } else {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Success"
message:@"Image was successfully saved in photoalbum"  delegate:nil
cancelButtonTitle:nil otherButtonTitles:@"Close", nil];
        [alert show];
    }
}

- (IBAction)toggleSwitch:(id)sender {
    self.theImage.image = cropImage;
    flipSwitch = 1;
}

- (IBAction)connect:(id)sender {
    UIGraphicsBeginImageContext(self.theImage.frame.size);
    [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];
    CGContextMoveToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);
    CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), firstPoint.x,
firstPoint.y);

    CGContextSetLineCap(UIGraphicsGetCurrentContext(), kCGLineCapRound);
    CGContextSetLineWidth(UIGraphicsGetCurrentContext(), 1 );
    CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), 1, 1, 0, 1.0);

CGContextSetBlendMode(UIGraphicsGetCurrentContext(),kCGBlendModeNormal);

    CGContextStrokePath(UIGraphicsGetCurrentContext());
    self.theImage.image = UIGraphicsGetImageFromCurrentImageContext();
    [self.theImage setAlpha:1];
    UIGraphicsEndImageContext();
    currentPoint = firstPoint;
```

```
}
/*
- (IBAction)testHistogram:(id)sender {
    CGImageRef sourceImage = theImage.image.CGImage;

    CFMutableDataRef theData;
    theData = CFDataCreateMutableCopy(0, 0,
CGDataProviderCopyData(CGImageGetDataProvider(sourceImage)));
    UInt8 *pixelData = (UInt8 *) CFDataGetMutableBytePtr(theData);
    int dataLength = CFDataGetLength(theData);

    NSMutableString* outputString = [NSMutableString
stringWithCapacity:dataLength/4];
    int pixelArray[300] = {};

    for (int index = 0; index < dataLength; index += 4) {
        //          [outputString appendFormat:@" %d", pixelData[index]];
        for (int i = 0; i < 300; i ++)
            if (i == pixelData[index]) {
                pixelArray[i]++;
            }
    }

    for (int x = 0; x<300; x++)
    {
        if (x==0)
        {
            [outputString appendFormat:@"%d (black): %d \n", x, pixelArray[x]];
        } else if (x==255)
        {
            [outputString appendFormat:@"%d (white): %d \n", x, pixelArray[x]];
        }
        else
        {
            [outputString appendFormat:@"%d: %d \n", x, pixelArray[x]];
        }
    }

    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *filePath =  [documentsDirectory
stringByAppendingPathComponent:@"PixelValues.txt"];
    NSError *error;
```

```
    [outputString writeToFile:filePath atomically:YES
encoding:NSUTF8StringEncoding error:&error];
}
*/

@end

//
//  MDSDrawViewController.h
//  Project_v1
//
//  Created by Michael Skinner on 12/12/14.
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import <UIKit/UIKit.h>

@protocol MDSDrawViewControllerDelegate <NSObject>
//- (void)closeDraw:(id)sender;
@end

@interface MDSDrawViewController : UIViewController
{
    CGPoint lastPoint;
    BOOL mouseSwiped;
}

@property (weak, nonatomic) id<MDSDrawViewControllerDelegate> delegate;

@property (weak, nonatomic) IBOutlet UIImage *drawImage;

@property (strong, nonatomic) IBOutlet UIImageView *theImage;
@property (strong, nonatomic) IBOutlet UIImageView *tempDrawImage;
@property (strong, nonatomic) IBOutlet UIImageView *undoImage;
@property (strong, nonatomic) IBOutlet UISwitch *toggleSwitch;
@property (strong, nonatomic) IBOutlet UIImageView *scrollView;


@end

//
//  MDSDrawViewController.m
//  Project_v1
//
//  Created by Michael Skinner on 12/12/14.
```

```
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import "MDSDrawViewController.h"

@interface MDSDrawViewController ()

@end

@implementation MDSDrawViewController {
   UIImage* originalImage;
}

@synthesize drawImage;
@synthesize tempDrawImage;
@synthesize theImage;
@synthesize toggleSwitch;
@synthesize scrollView;


- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil
{
   self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
   if (self) {
      // Custom initialization
   }
   return self;
}

- (void)viewDidLoad
{
   [super viewDidLoad];
      // Do any additional setup after loading the view.
}

- (void)viewDidUnload
{
   [self setTheImage:nil];
   //    [self setOriginalImage:nil];
   [super viewDidUnload];
}


- (void)viewWillAppear:(BOOL)animated {
```

```
    self.theImage.image = drawImage;
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {

//    self.undoImage.image = UIGraphicsGetImageFromCurrentImageContext();
    originalImage = theImage.image;
    mouseSwiped = NO;
    UITouch *touch = [touches anyObject];
    lastPoint = [touch locationInView:self.theImage];

}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {

    mouseSwiped = YES;
    UITouch *touch = [touches anyObject];
    CGPoint currentPoint = [touch locationInView:self.theImage];

//    UIGraphicsBeginImageContextWithOptions(self.theImage.frame.size, NO, 0);
    UIGraphicsBeginImageContext(self.theImage.frame.size);
    [self.tempDrawImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];
    CGContextMoveToPoint(UIGraphicsGetCurrentContext(), lastPoint.x, lastPoint.y);
    CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);
    CGContextSetLineCap(UIGraphicsGetCurrentContext(), kCGLineCapRound);

    if (toggleSwitch.on){
        CGContextSetLineWidth(UIGraphicsGetCurrentContext(), 20 );
        CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), 1, 1, 1, 1);

CGContextSetBlendMode(UIGraphicsGetCurrentContext(),kCGBlendModeNormal);
    } else{
        CGContextSetLineWidth(UIGraphicsGetCurrentContext(), 1 );
        CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), 0, 0, 0, 1);

CGContextSetBlendMode(UIGraphicsGetCurrentContext(),kCGBlendModeNormal);
    }
```

```
    CGContextStrokePath(UIGraphicsGetCurrentContext());
    self.tempDrawImage.image = UIGraphicsGetImageFromCurrentImageContext();
//    [self.tempDrawImage setAlpha:.5];
    UIGraphicsEndImageContext();

    lastPoint = currentPoint;
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    if(!mouseSwiped) {
  //      UIGraphicsBeginImageContextWithOptions(self.theImage.frame.size, NO, 0);
 /*       UIGraphicsBeginImageContext(self.theImage.frame.size);
        [self.tempDrawImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];
        CGContextSetLineCap(UIGraphicsGetCurrentContext(), kCGLineCapRound);
        CGContextSetLineWidth(UIGraphicsGetCurrentContext(), 1);
        CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), 0, 0, 0, 1);
        CGContextMoveToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
        CGContextStrokePath(UIGraphicsGetCurrentContext());
        CGContextFlush(UIGraphicsGetCurrentContext());
        self.tempDrawImage.image = UIGraphicsGetImageFromCurrentImageContext();
        UIGraphicsEndImageContext();*/
    }
//    UIGraphicsBeginImageContext(self.theImage.frame.size);
//    [self.tempDrawImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)
blendMode:kCGBlendModeNormal alpha:1];
//    [self.drawImage drawInRect:CGRectMake(0, 0, self.theImage.frame.size.width,
self.theImage.frame.size.height) blendMode:kCGBlendModeNormal alpha:1];
 //   [self.tempDrawImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)
blendMode:kCGBlendModeNormal alpha:1];

//    self.theImage.image = UIGraphicsGetImageFromCurrentImageContext();
//    UIGraphicsEndImageContext();
}

- (IBAction)save:(id)sender {
    UIGraphicsBeginImageContextWithOptions(self.theImage.bounds.size, NO, 0.0);
    [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];
```

```
    UIImage *SaveImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    UIImageWriteToSavedPhotosAlbum(SaveImage,
self,@selector(image:didFinishSavingWithError:contextInfo:), nil);
}


- (void)image:(UIImage *)image didFinishSavingWithError:(NSError *)error
contextInfo:(void *)contextInfo
{
    // Was there an error?
    if (error != NULL)
    {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error"
message:@"Image could not be saved.Please try again"  delegate:nil
cancelButtonTitle:nil otherButtonTitles:@"Close", nil];
        [alert show];
    } else {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Success"
message:@"Image was successfully saved in photoalbum"  delegate:nil
cancelButtonTitle:nil otherButtonTitles:@"Close", nil];
        [alert show];
    }
}

- (IBAction)undo:(id)sender {
    theImage.image = originalImage;
}
/*
- (IBAction)test:(id)sender {
    CGImageRef sourceImage = theImage.image.CGImage;

    CFMutableDataRef theData;
    theData = CFDataCreateMutableCopy(0, 0,
CGDataProviderCopyData(CGImageGetDataProvider(sourceImage)));
    UInt8 *pixelData = (UInt8 *) CFDataGetMutableBytePtr(theData);
    int dataLength = CFDataGetLength(theData);

    NSMutableString* outputString = [NSMutableString
stringWithCapacity:dataLength/4];
    int pixelArray[300] = {};

    for (int index = 0; index < dataLength; index += 4) {
        //          [outputString appendFormat:@" %d", pixelData[index]];
        for (int i = 0; i < 300; i ++)
```

```
            if (i == pixelData[index]) {
                pixelArray[i]++;
            }
    }

    for (int x = 0; x<300; x++){
        if (x==0) {
            [outputString appendFormat:@"%d (black): %d \n", x, pixelArray[x]];
        } else if (x==255){
            [outputString appendFormat:@"%d (white): %d \n", x, pixelArray[x]];
        }
        else {
            [outputString appendFormat:@"%d: %d \n", x, pixelArray[x]];
        }
    }

    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *filePath =  [documentsDirectory
stringByAppendingPathComponent:@"PixelValues2.txt"];
    NSError *error;
    [outputString writeToFile:filePath atomically:YES
encoding:NSUTF8StringEncoding error:&error];
}
*/

@end

//
//  MDSMeasureViewController.h
//  Project_v1
//
//  Created by Michael Skinner on 12/12/14.
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import <UIKit/UIKit.h>

@protocol MDSMeasureViewControllerDelegate <NSObject>

@end

@interface MDSMeasureViewController : UIViewController
{
```

```
    CGPoint lastPoint;
    CGPoint currentPoint;
    BOOL mouseSwiped;
}

@property (weak, nonatomic) id<MDSMeasureViewControllerDelegate> delegate;

@property (weak, nonatomic) IBOutlet UIImage *measureImage;

@property (strong, nonatomic) IBOutlet UIImageView *theImage;
@property (strong, nonatomic) IBOutlet UIImageView *tempDrawImage;
@property (strong, nonatomic) IBOutlet UILabel *lengthLabel;
@property (strong, nonatomic) IBOutlet UISegmentedControl *segmentControl;
@property (strong, nonatomic) IBOutlet UITextField *calibrateText;
@property (strong, nonatomic) IBOutlet UISwitch *calibrateSwitch;

@end

//
//  MDSMeasureViewController.m
//  Project_v1
//
//  Created by Michael Skinner on 12/12/14.
//  Copyright (c) 2014 ___Michael___. All rights reserved.
//

#import "MDSMeasureViewController.h"

@interface MDSMeasureViewController ()

@end

@implementation MDSMeasureViewController

@synthesize measureImage;
@synthesize theImage;
@synthesize calibrateText;
@synthesize calibrateSwitch;

int segmentCount = 0;
int count = 0;
double totalLength;
double calibrateValue;
```

```
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    UIToolbar* numberToolbar = [[UIToolbar alloc]initWithFrame:CGRectMake(0, 0,
320, 50)];
    numberToolbar.barStyle = UIBarStyleBlackTranslucent;
    numberToolbar.items = [NSArray arrayWithObjects:
                           [[UIBarButtonItem alloc]initWithTitle:@"Cancel"
style:UIBarButtonItemStyleBordered target:self
action:@selector(cancelNumberPad)],
                           [[UIBarButtonItem
alloc]initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace target:nil
action:nil],
                           [[UIBarButtonItem alloc]initWithTitle:@"Apply"
style:UIBarButtonItemStyleDone target:self action:@selector(doneWithNumberPad)],
                           nil];
    [numberToolbar sizeToFit];
    calibrateText.inputAccessoryView = numberToolbar;
        // Do any additional setup after loading the view.
}

- (void)viewDidUnload
{
    [self setTheImage:nil];
    [super viewDidUnload];
}

-(void)cancelNumberPad{
    [calibrateText resignFirstResponder];
    calibrateText.text = @"";
}

-(void)doneWithNumberPad{
//    NSString *numberFromTheKeyboard = calibrateText.text;
```

```
    [calibrateText resignFirstResponder];
}

- (void)viewWillAppear:(BOOL)animated {

    self.theImage.image = measureImage;
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {

    mouseSwiped = NO;
    UITouch *touch = [touches anyObject];
    lastPoint = [touch locationInView:self.theImage];
    currentPoint = lastPoint;
}

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {

    mouseSwiped = YES;
    UITouch *touch = [touches anyObject];
    currentPoint = [touch locationInView:self.theImage];

    UIGraphicsBeginImageContext(self.theImage.frame.size);
    theImage.image = measureImage;

    [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];

    CGContextMoveToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);

    if (segmentCount == 1) {
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
lastPoint.y);
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
currentPoint.y);
```

```
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);
    }
    if (segmentCount == 0){
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
    }

    CGContextSetLineCap(UIGraphicsGetCurrentContext(), kCGLineCapRound);
    CGContextSetLineWidth(UIGraphicsGetCurrentContext(), 1 );
    CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), 0, 0, 0, 1.0);

CGContextSetBlendMode(UIGraphicsGetCurrentContext(),kCGBlendModeNormal);

    CGContextStrokePath(UIGraphicsGetCurrentContext());
    self.theImage.image = UIGraphicsGetImageFromCurrentImageContext();
    [self.theImage setAlpha:1];
    UIGraphicsEndImageContext();
}

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    self.theImage.image = measureImage;

    UIGraphicsBeginImageContext(self.theImage.frame.size);
    [self.theImage.image drawInRect:CGRectMake(0, 0,
self.theImage.frame.size.width, self.theImage.frame.size.height)];

    CGContextMoveToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);

    if (segmentCount == 1) {
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
lastPoint.y);
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
currentPoint.y);
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), currentPoint.x,
currentPoint.y);
    }
    if (segmentCount == 0){
        CGContextAddLineToPoint(UIGraphicsGetCurrentContext(), lastPoint.x,
lastPoint.y);
    }
    if (segmentCount == 2){
```

```
        count++;
    }

    CGContextSetLineCap(UIGraphicsGetCurrentContext(), kCGLineCapRound);
    CGContextSetLineWidth(UIGraphicsGetCurrentContext(), 1 );
    CGContextSetRGBStrokeColor(UIGraphicsGetCurrentContext(), 0, 0, 0, 1.0);

CGContextSetBlendMode(UIGraphicsGetCurrentContext(),kCGBlendModeNormal);

    CGContextStrokePath(UIGraphicsGetCurrentContext());
    self.theImage.image = UIGraphicsGetImageFromCurrentImageContext();
    [self.theImage setAlpha:1];
    UIGraphicsEndImageContext();

    int xlength;
    int ylength;
    double totalArea;

    if (calibrateSwitch.on){
        xlength = currentPoint.x - lastPoint.x;
        ylength = currentPoint.y - lastPoint.y;
        totalLength = sqrt(xlength*xlength + ylength*ylength)*calibrateValue;
        totalArea = abs(xlength*ylength)*calibrateValue*calibrateValue;
    }
    else{
        xlength = currentPoint.x - lastPoint.x;
        ylength = currentPoint.y - lastPoint.y;
        totalLength = sqrt(xlength*xlength + ylength*ylength);
        totalArea = abs(xlength*ylength);
    }

    if (segmentCount == 1) {
        self.lengthLabel.text = [NSString stringWithFormat:@"Area: %.2f", totalArea];
    }
    if (segmentCount == 0){
        self.lengthLabel.text = [NSString stringWithFormat:@"Length: %.2f",
totalLength];
    }
    if (segmentCount == 2) {
        self.lengthLabel.text = [NSString stringWithFormat:@"Count: %d", count];
    }

}

- (IBAction)switchChange:(UISegmentedControl *)sender {
```

```
    switch (self.segmentControl.selectedSegmentIndex)
    {
        case 0:
            segmentCount = 0;
            self.lengthLabel.text = [NSString stringWithFormat:@"Length: ------"];
            break;
        case 1:
            segmentCount = 1;
            self.lengthLabel.text = [NSString stringWithFormat:@"Area: ------"];
            break;
        case 2:
            segmentCount = 2;
            count = 0;
            self.lengthLabel.text = [NSString stringWithFormat:@"Count: ------"];
            break;
        default:
            break;
    }
}

- (IBAction)toggleSwitch:(id)sender {
    if (calibrateSwitch.on){
        int x = [calibrateText.text intValue];
        calibrateValue = x/totalLength;
    }
}

@end
```

REFERENCES

1. Belight Software, ltd. (2014). Image Tricks Pro (Version 3.8.1). [Computer software]. Retrieved from <www.apple.com/mac/app-store>

2. The Iconfactory & Artis Software. (2015). Flare 2 (Version 2.1.1). [Computer software]. Retrieved from <www.apple.com/mac/app-store>

3. Li, Yong. (2011). Image Editor (Version 1.0). [Computer software]. Retrieved from <www.apple.com/mac/app-store>

4. Image J (Version 1.49s). (2015). Retrieved from <www.imagej.nih.gov/ij>

5. Axiem Systems. (2015). Photo Editor (Version 4.2.1). [Mobile application software]. Retrieved from <www.apple.com/iphone/apps-for-iphone/>

6. iOS Dev Center. (2015). <https://developer.apple.com/devcenter/ios/index.action>

7. Khan, Abdul Azeem. "How to Make a Simple Drawing App with UIKit." (2012). Ray Wenderlich Tutorials for Developers & Gamers. <http://www.raywenderlich.com/18840/how-to-make-a-simple-drawing-app-with-uikit>

8. Vuksic, Goran. "iPhone Image Processing." (2011). Five Agency. <http://five.agency/iphone-image-processing/>

9. Helland, Tanner. "Seven Grayscale Conversion Algorithms (With Pseudocode and VB6 Source Code)." (2012). <http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/>

10. Otsu, Nobuyuki. "A Threshold Selection Method from Gray-Level Histograms." IEEE Trans. Systems, Man, and Cybernetics 9(1), pp. 62-66, 1979.

11. Ridler, T.W. and Calvard, S. "Picture Thresholding Using an Iterative Selection Method." IEEE Trans. Systems, Man, and Cybernetics, 8(8), pp. 630-632, 1978.