Homework 1: Basic Image Manipulations

Christopher Kantor

Department of Computer Science and Engineering

University of South Florida

1. Introduction and description

This assignment focuses on the implementation of multiple common image manipulation techniques like rotating and scaling. We are also tasked with adding color image support in the .ppm format to our image processing system. Rotation, scaling, and color support are all essential for any image processing program and will be used extensively in future projects. In machine learning, data augmentation on images often involves rotation, scaling, or brightness change, all of which we now support.

In this project, we also added support for ROIs, which will allow us to target our manipulations to specific areas of an image, saving us time and space during each manipulation.

The report is organized as follows. In section 2, the algorithms used in this project are described. Section 3 describes the implementation details of this project. We display the results of the project in Section 4, and we conclude in section 5.

2. Description of algorithms

All functions from project 0, and any new ones included in this project have been changed to function inside an ROI. ROIs allow us to focus on specific portions of an image. We implemented a scaling function, which scales ROI between 1-2x and a rotation function which can rotate an ROI 90, 180, or 270 degrees. We also added color support for rotation, and additive and multiplicative brightness changes on color images, which work on a per-channel basis (RGB).

2.1. ROI

A ROI is a subset of the image defined by its upper left corner as an x and y coordinate, and the size in each dimension. We can use these to focus on specific parts of an image. Each ROI is a subset of the original image, and we can use offsets to access the pixels without needing to copy the values over for every operation. We can define multiple ROIs per image, and they can overlap as well. The operations will be completed in the order of their definition, so ordering the ROIs differently will result in a different final output.

During a call to our program, we create a new image, and apply all the changes from the ROIs to it, so we don't change the original image. After we are done processing each ROI, we save this new image to the system so that we can look at our changes from the original.

Inside the ROI, pixel (0,0) becomes pixel (x, y), and the size of the image becomes (x + size in x direction). Using offsets instead of creating separate arrays for each ROI saves us space and time in most functions, such as brightness changing functions, although we did need to implement a separate array for our rotation and scaling functions.

2.2. Rotation

Rotation is an essential image processing technique and can be implemented using a variety of methods. The method chosen for this project involved taking the transpose of the image, then reversing the order of the rows. The transpose of a matrix converts all rows into columns, and vice versa. We then reverse the order of the new columns, to get an image that is rotated 90 degrees clockwise. We can do this multiple times to get a 180 degree, or 270-degree rotation of the image.

The transpose can be obtained by swapping arr[i][j] and arr[j][i], for 0<i<size, 0<j<i in a double for loop.

We then reverse the order of each column by going through each row and reversing it. After reversing the order of elements in each row, we will have effectively revered the order of the columns.

Although this can be done as an in-place operation, we needed to create a temporary buffer to store the ROI pixels, as we had difficulty iterating through the ROI and setting pixels in place while having to worry about the offset that an ROI creates.

This increases the space complexity from O(1) to O(M^2), where M is the size of one axis of the ROI. This could lead to poor performance if the ROIs are extremely large, or if we have multiple channels to do rotations on, as we need to rotate each channel individually. In this project however, this has not been an issue.

Time complexity is slow for this operation, being O(M^2). Transposing the matrix takes O(M^2) time, and reversing the columns takes O(M * M) = O(M^2), as we need to traverse each row (M) and reverse it (M).

O(M^2) + O(M^2) = O(M^2)

Putting all the elements in the ROI into our temporary array also takes O(M^2), but this doesn't change the total runtime. For color images, rotation is applied to each color channel, but this is still O(M^2).

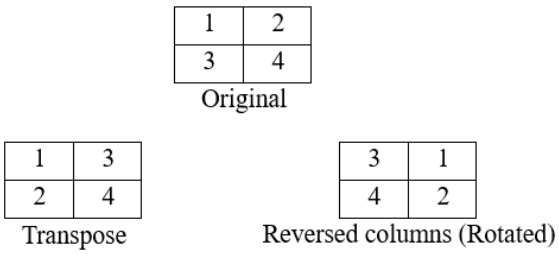


Figure 1: Shows rotation process. We first transpose the matrix, then reverse the column order to get a 90 clockwise rotation.

2.3. Scaling

Scaling is another very common image manipulation technique and can have multiple methods of implementation. We chose a very basic method for this project, where we set the scale factor to 2 if the ratio given by the user is between 1.5 and 2. The scale factor is 1 if the ratio given by the user is between 1 and 1.5. In this case, nothing

happens to the image, and we can just copy the pixels over.

If the scale factor is 2, then each pixel in the original image will show up 4 times in the new one, doubling the size of the image in each axis. Since we are using ROIs however, we only focus on the ROI window, and lose whatever pixels would normally get scaled outside of the window. This effectively means that we are focusing on the top left quadrant of the ROI whenever we call scale on it.

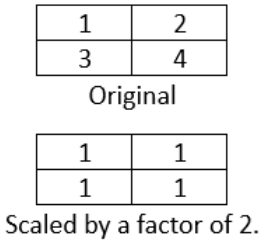


Figure 2: Shows how scaling works. We focus on the top left quadrant of the original image and lose the data that is outside of it.

2.4. Color Processing

Color is supported using a different filetype than the greyscale images we have been using thus far. A grayscale image uses the .pgm format, and only has one channel. A color image uses the .ppm format, and has 3 channels (RGB). We can process color images the same way as the greyscale ones, but we need to do all the operations for each color channel. This doesn't effectively increase our runtime as we have a constant number of channels. Adding color to a channel works the same way as adding color to a greyscale image, and rotation just needs to be performed on each channel in the color image. Color multiplication works similarly to addition, except we pass in a scaling factor, just like we do when we scale an image. The runtimes for pixel manipulations such as binarization, additive or multiplicative intensity changes, or thresholding is O(M^2), as we need to go through every pixel in the ROI and do some arithmetic on it.

3. Implementation

We built this project on top of the previous one but expanded support of the older functions to work with ROIs. To support ROIs, we created a new class called ROI in utilities.h. An ROI object holds its start coordinates, the size of each axis, and any operations and parameters they might have. We provide all the information we need including the source and target file, number of ROIs, rotation angle for all of them, and the ROIs themselves inside parameters.txt.

Each call to our function is defined on its own line, with ROIs being delimited by the string "ROI". The parameters.txt file is read in iptool.c, which holds the main function for our entire image processing system. We tokenize our input and create an array of ROI objects with all their information in this file, then call a utility function we created called ROI_Processing to process each ROI. After all ROIs are processed, we save the new image, then repeat for the next line in parameters.txt.

Each of our processing functions, such as rotate, scaling, addColor, etc. are all defined in utilities.h, and implemented in utilities.cpp, with use cases described in the readme.txt file.

4. Description and analysis of results

The results of our algorithms used are displayed here. We have 12 images for color and grey images. Our ROIs are defined for the top left, top right, and bottom right quadrant of the image.

For grey images:

- 1 original file
- 3 rotated versions of the original file, with 3 ROIs each. The rotation angles are 90, 180, and 270.
- 4 scaled rotated versions of the original file, with 3 ROIs each. The rotation angles are 0, 90, 180, and 270.
- 4 images that have had their brightness changed and have been rotated, with 3 ROIs each. Each ROI has a different brightness value. The rotation values are 0, 90, 180, and 270.

For color images:

- 1 original file
- 3 rotated versions of the original file, with 3 ROIs each. The rotation angles are 90, 180, and 270.
- 4 images that have had their color changed with a multiplicative scaler, then rotated, with 3 ROIs each. The rotation angles are 0, 90, 180, and 270.
- 4 images that have had their brightness changed by an additive scalar and have been rotated, with 3 ROIs each. Each ROI has a different brightness value. The rotation values are 0, 90, 180, and 270.

4.1. Grey images

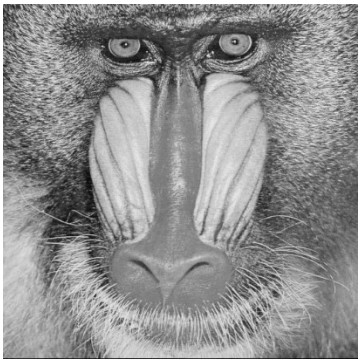
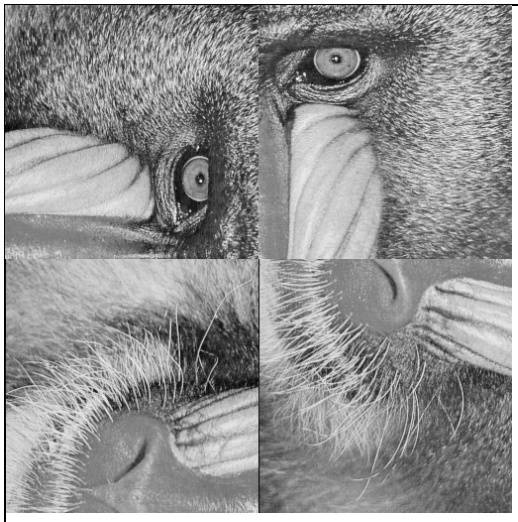
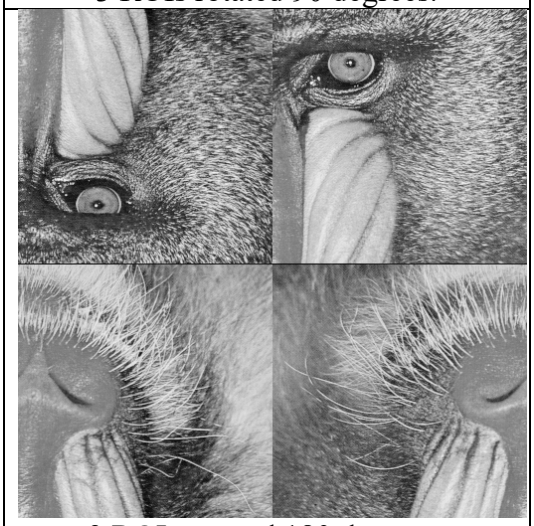


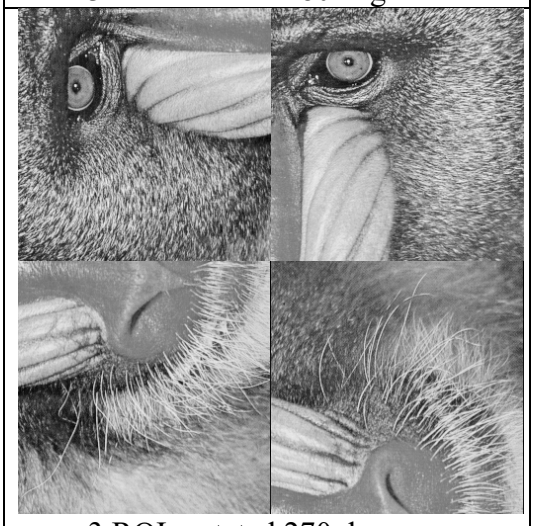
Figure 3: Original image used for the grey image manipulations.



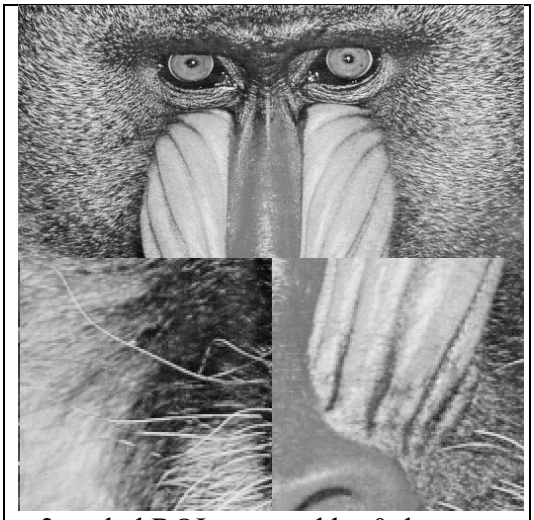
3 ROIs rotated 90 degrees.



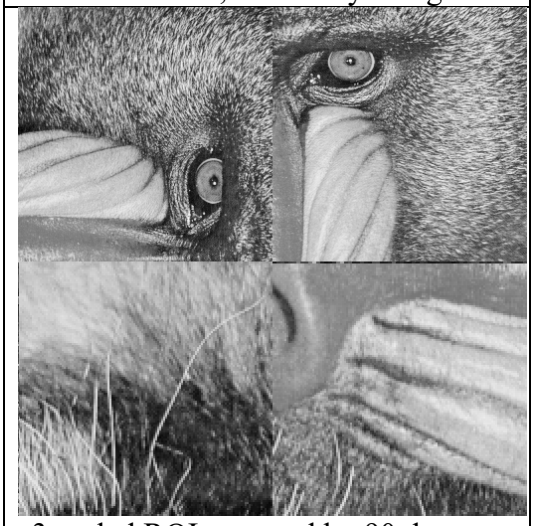
3 ROIs rotated 180 degrees.



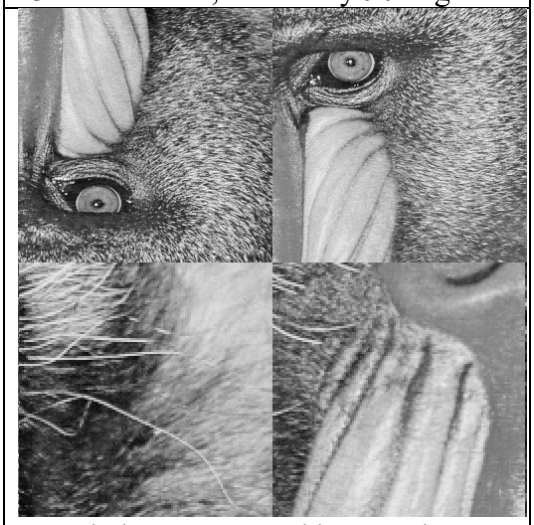
3 ROIs rotated 270 degrees.



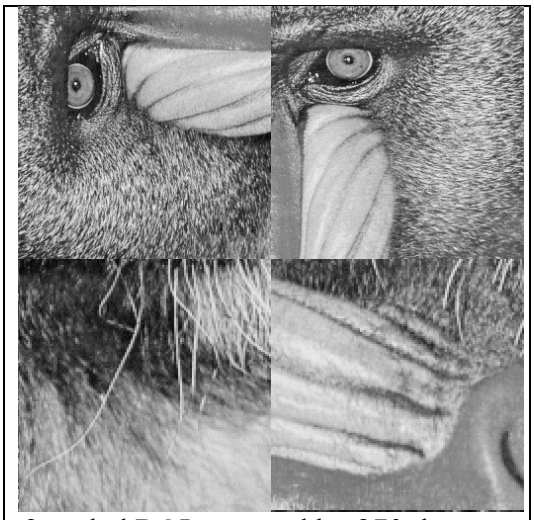
3 scaled ROIs, rotated by 0 degrees



3 scaled ROIs, rotated by 90 degrees

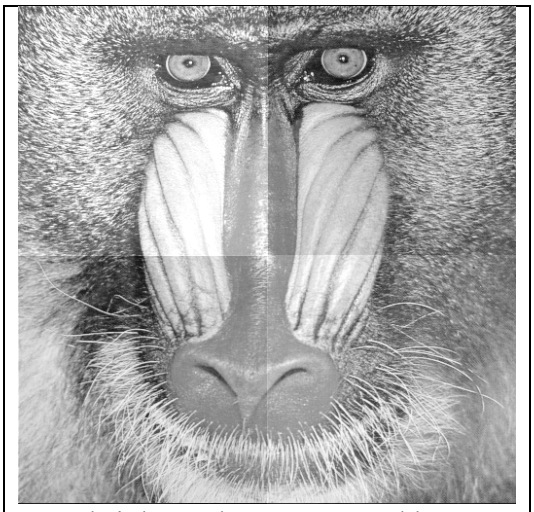


3 scaled ROIs, rotated by 180 degrees

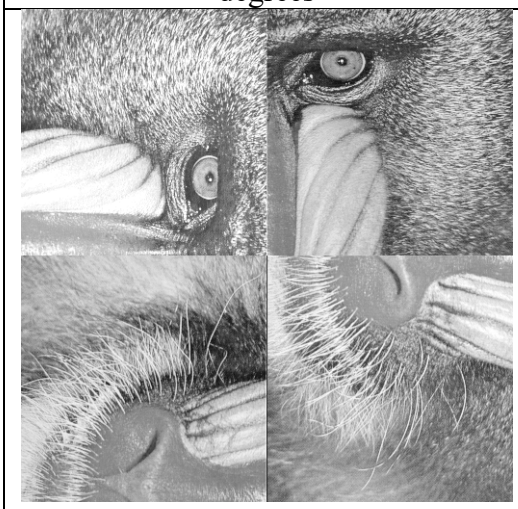


3 scaled ROIs, rotated by 270 degrees

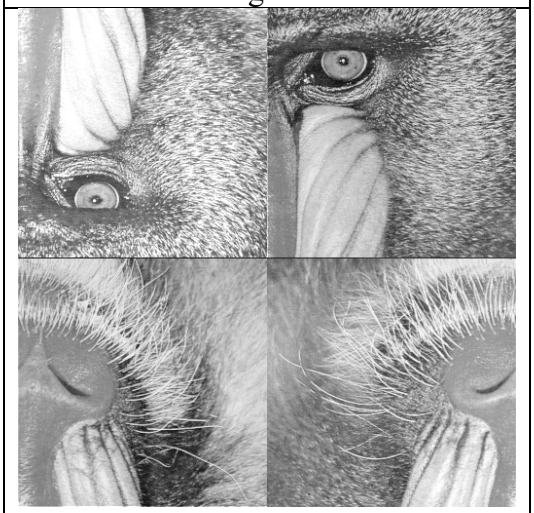
The scale factors used were 1.12 for the top left, 1.67 for bottom right, and 2 for bottom left.



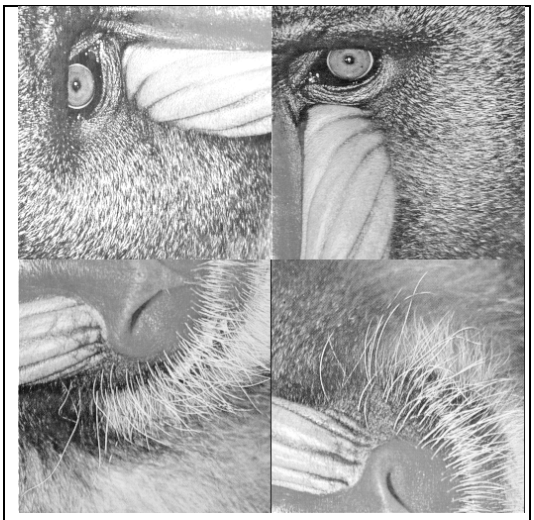
3 brightened ROIs, rotated by 0 degrees



3 brightened ROIs, rotated by 90 degrees



3 brightened ROIs, rotated by 180 degrees



3 brightened ROIs, rotated by 270 degrees

The ROIs were brightened by 50 (top left), 10 (bottom right), and 25 (bottom left).

4.2. Color Images

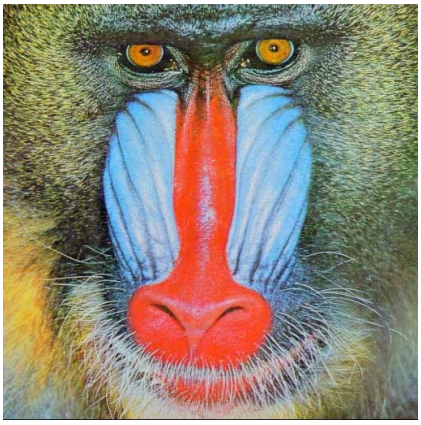
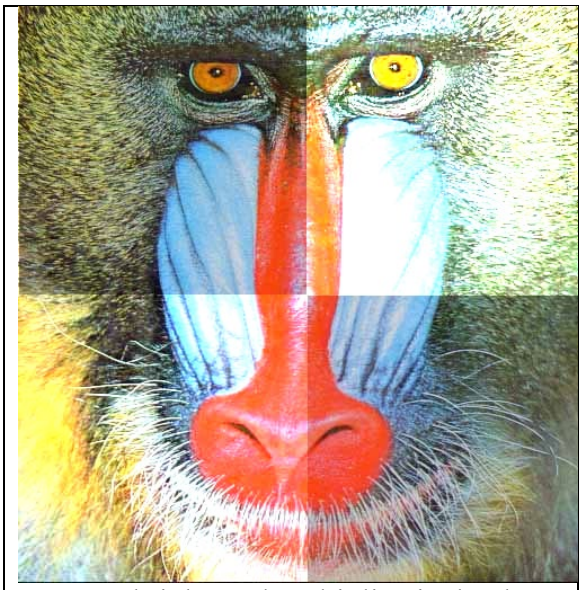


Figure 4: Original color image used for processing.

3 ROIs rotated 90 degrees.

3 ROIs rotated 180 degrees.

3 ROIs rotated 270 degrees.



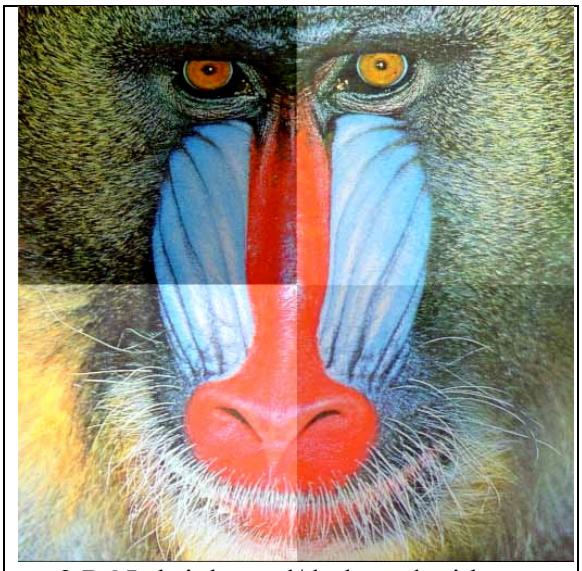
3 ROIs brightened multiplicatively, then rotated by 0 degrees

3 ROIs brightened multiplicatively, then rotated by 180 degrees

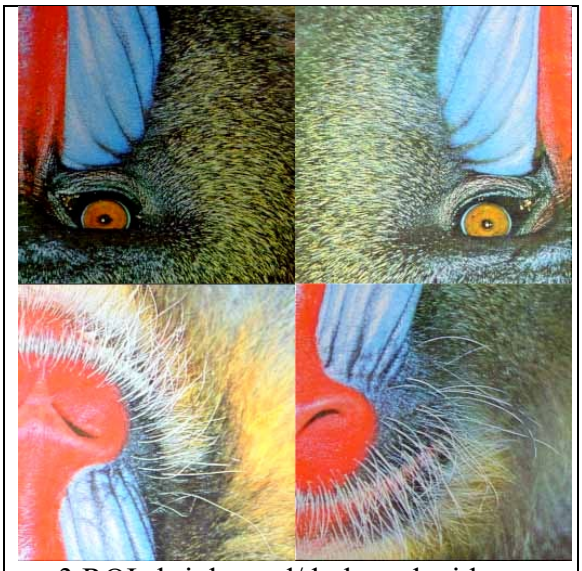
3 ROIs brightened multiplicatively, then rotated by 90 degrees

3 ROIs brightened multiplicatively, then rotated by 270 degrees

We used factors (1.25, 1.5, and 2) for this operation.

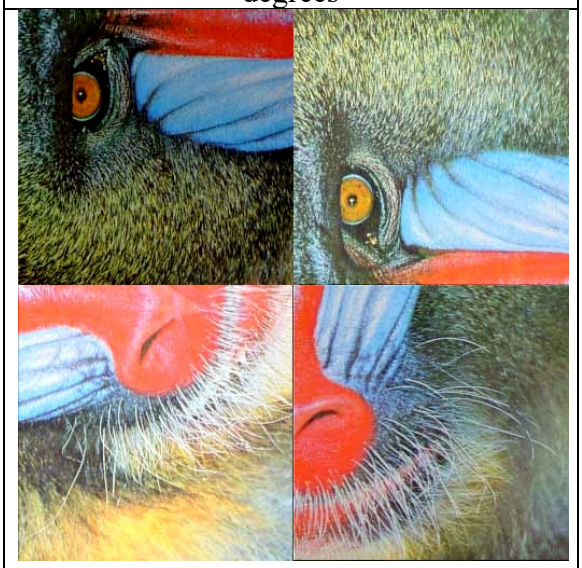


3 ROIs brightened/darkened with an additive scalar, then rotated by 0 degrees



3 ROIs brightened/darkened with an additive scalar, then rotated by 180 degrees

3 ROIs brightened/darkened with an additive scalar, then rotated by 90 degrees



3 ROIs brightened/darkened with an additive scalar, then rotated by 270 degrees

The scalars used were -50, 50, and 25.

5. Conclusion

In this assignment we were tasked with implementing basic image manipulation algorithms such as rotation and scaling. We also added support for ROIs, and color images, both of which will be used extensively in the later projects. Some optimizations can be made for rotation and scale by changing them to be in place, but we do not take a time penalty as far as big oh is concerned. Color manipulations are the same as grey scale ones, except we must do them for each color channel.

Some examples on how to call our functions are listed below as well as in readme.txt. All of these must be typed into a single line in parameters.txt, having newlines in the middle of a parameter call with mess up the reading of it.

*** EXAMPLE PARAMETERS ****

Example 1:

infile.pgm outfile.pgm 1 0 ROI 0 0 512 512 addGrey 50

// We only have 1 ROI and will not rotate it

// ROI is defined from 0,0 to 512,512, and we will add 50 intensity to every pixel in int

Example 2:

infile.pgm outfile.pgm 2 90 ROI 0 0 256 256 addGrey 50 ROI 256 256 256 256 scale 1.67

// We have 2 ROIs not, the first one is from (0,0) to (256,256) This is the top left quadrant if the image is 512 x 512

// The second is from (256, 256) to (512, 512) the bottom right quadrant

// on the first ROI, we will add 50 brightness to every pixel in it

// on the second ROI, we will scale it by a factor of 1.67

// we will also rotate each of these ROIs by 90 degrees

Example 3:

infile.ppm outfile.ppm 3 270 ROI 0 0 256 256 multColor 1.25 ROI 256 0 256 256 multColor 1.5 ROI 0 256 256 256 multColor 2

// We have 3 ROIs, and will rotate each of them by 270 degrees

// The first ROI calls multColor with a scale factor of 1.25

// The second ROI calls multColor with a scale factor of 1.5

// The third ROI calls multColor with a scale factor of 2

Example 4:

infile.pgm outfile.pgm 2 90 ROI 0 0 256 256

addGrey 50 ROI 256 256 256 256 scale 1.67

// This is incorrect as there is a newline before all of the ROIs get read.

// This will cause an error

Example 5:

infile.pgm outfile_1.pgm 1 90 ROI 0 0 256 256 addGrey 25

infile.pgm outfile_2.pgm 1 270 ROI 256 256 256 256 scale 1.56

// This has two full paramaters, and will create 2 seperate images, outfile_1 and outfile_2

// each one is processed seperately2