Part 2 Written Report

Alwin Hollebrandse

This program requires the user to provide two command line args. The first is file location that holds all images of interest. The second argument is instruction file. This code will create a new results folder in the current directory that will have a folder per image that will house all operation outputs. See the "Results" section for example output.

Image File location Arg:

This is the first argument of the program. It should contain a path to where all of the images are located. In thoery, any images that Java's Buffered Image supports would be supported by my code. An example image can found in the results section under "original".

Instruction File arg:

This is the second command line argument for the program. It should be a txt file. It is used to build the operation arraylist (discussed under Implentation/General). Accepted Strings that will cause the actions of interest are (ignoring case SingleColor, EdgeDetection, HistogramThresholdingSegmentation, KMeansSegmentation, Erosion, and Dilation. These can be found in the all_Instructions.txt file with their respective parameters. Note that SingleColor is not a required operation, but without it, all other operations default to a "gray" color operation use. Note that each of the params must be separated by spaces, and array params are denoted by having each element be separated by a space and the characters: [and].

SingleColor takes in 1 param in the file: the color to convert to. Current options are: Gray, Red, Green, and Blue

EdgeDetection takes no params.
HistogramThresholdingSegmentation takes no params.
KMeansSegmentation takes no params.
erosion takes in 3 params: filter width, filter height, colors

dilation takes in 3 params: filter width, filter height, colors

Example Instruction File Contents:

SingleColor Gray

EdgeDetection
HistogramThresholdingSegmentation
KMeansSegmentation

erosion 3 3 [255 0 255 0 255 0 255 0 255]
dilation 3 3 [255 0 255 0 255 0 255 0 255]

Implentation:

General:

This program does the following: first it validates the users command line args. It then parses the instructions.txt file into an arraylist that holds the found Strings that match those and explained in the Instrcution File arg section. This code then loops through all of the images found in the first command line argument. It then performs the user requested operations in the following order (skipping operations not requested): case SingleColor, EdgeDetection, HistogramThresholdingSegmentation, KMeansSegmentation, Erosion, and Dilation. If there is an error within one of these operations, the operation will be skipped (or if it severe enough, the whole image will be skipped). If the operation was not requested, the code will check if there are any output files related to said operation already in the results folder for the current image. If there is, it will be deleted. This was implanted such that there will be no confusion when checking results. If the operation was requested, it will be completed and added to ./results/{{current image's name}}/{{operation output}}. Each of these operations works on the outcome of the previous specified operation.

Each operation loops through all pixels in the image, which is performed in parallel. This was accomplished by creating a ParallelMatrix File. This file creates how ever many threads the system can support and divides the image pixel matrix into segments of equal size. This was done by looping through all the width pixels but jumping by MAX_THREADS and by looping through all height pixels per width iteration. Each thread starts at an incremented x value. Each thread then calls a code lambda defined in the respective operation's java file for the current width and height value of the pixel matrix.

The end of the program prints out time metrics collected through the code's run. These metrics also appear under "results/report.txt".

SingleColor:

This code is found under SingleColorScale.java. It takes in the original image and the desired color of the image. Accepted color values are (ignoring case): gray, red, green, and blue. The code calls ParallelMatrix with the following lambda: depending on what color the param was, remove unneeded channels (though the image remains in RBG format). It should be noted that research showed that gray pixels should not all be equal. The following formula was used for gray conversion: (int) (c.getRed() * 0.299) + (int) (c.getGreen() * 0.587) + (int) (c.getBlue() * 0.114);. The output is the new image using only the specified channel. The output is saved as {{user specified color}}.jpg

EdgeDetection

This section is found under EdgeDetection.java. It has no parameters. This operation begins by performing the LaPlace filter on the given image. This sharpened image then has the compass edge detection performed on it. “Edge filters” in four directions (performed through the filter operations and different weight parameters). The EdgeMap image gets a white pixel if any of the four calculated edge strengths (or their negative counter parts, which accounts for the other four directions) is greater than 1. This code returns the an EdgeMap for the provided image. This image can be found at edgeDetection.jpg.

HistogramThresholdingSegmentation

This section is found under ThresholdSegmentation.java. It accepts a histogram as an input. This operation begins by calculating the optimal pixel threshold. This is done by calculating the variance associated with a given pixel if it were the threshold. The pixel value with the minimum calculated value is selected. Then for every pixel, the pixel turns white if it less than the threshold and black otherwise. Detected objects are white greater than 0. This code returns a segmented version of the provided image. This image can be found at histogramThresholdingSegmentation.jpg.

KMeansSegmentation

This section is found under KMeansSegmentation.java. It accepts a histogram as an input. This operation performs a modified K++ algorithm on the histogram. The initial K points are selected as follows: place a point at the histogram index that has the highest occurrence value, and then place a point at the furthest histogram index from the first index while having a non-zero histogram value. The standard k means algorithm takes over from that point and results in 2 final clusters. Then for every pixel, the pixel turns white if its value is located in the “object” cluster and black otherwise. Detected objects are white greater than 0. This code returns a segmented version of the provided image. This image can be found at kMeansSegmentation.jpg.

Erosion

This section is found under MorphologicalFunctions.java. It accepts the following parameters: filterWidth, filterHeight, colors, and morphologicalType. This operation has a soft requirement of calling a segmentation operation prior to this operation. This operation occurs when morphologicalType is set to “erosion.” For each non-cropped pixel, the following happens. Check if each pixel in the current image filtered window matches the associated value in the colors array. If Each does, keep the pixel. Otherwise, remove the pixel by setting it to black. This image can be found at Erosion.jpg.

Dilation

This section is found under MorphologicalFunctions.java. It accepts the following parameters: filterWidth, filterHeight, colors, and morphologicalType. This operation has a soft requirement of calling a segmentation operation prior to this operation. This operation occurs when morphologicalType is set to "dilation." For each non-cropped pixel, the following happens. Check if the current pixel in the provided image has a non-zero value and is "object." If it is, set each associated pixel in the current filter window to "object" color if the associated colors value is non-zero. Otherwise, keep the pixel the same as it was. This image can be found at Dilation.jpg.

Example Metrics

These can be found in "results/report.txt"

Final Metrics:

Converting to a single color processing time for the entire batch (ms): 12720

Average converting to a single color processing time (ms): 25

Histogram creation processing time for the entire batch (ms): 5966

Average histogram creation processing time (ms): 11

Edge detection creation processing time for the entire batch (ms): 131924

Average edge detection processing time (ms): 263

Histogram thresholding segmentation time creation processing time for the entire batch (ms): 27517

Average histogram thresholding segmentation processing time (ms): 55

K means segmentation time creation processing time for the entire batch (ms): 20766

Average k means segmentation processing time (ms): 41

Erosion time creation processing time for the entire batch (ms): 10535

Average k means segmentation processing time (ms): 21

Dilation time creation processing time for the entire batch (ms): 7038

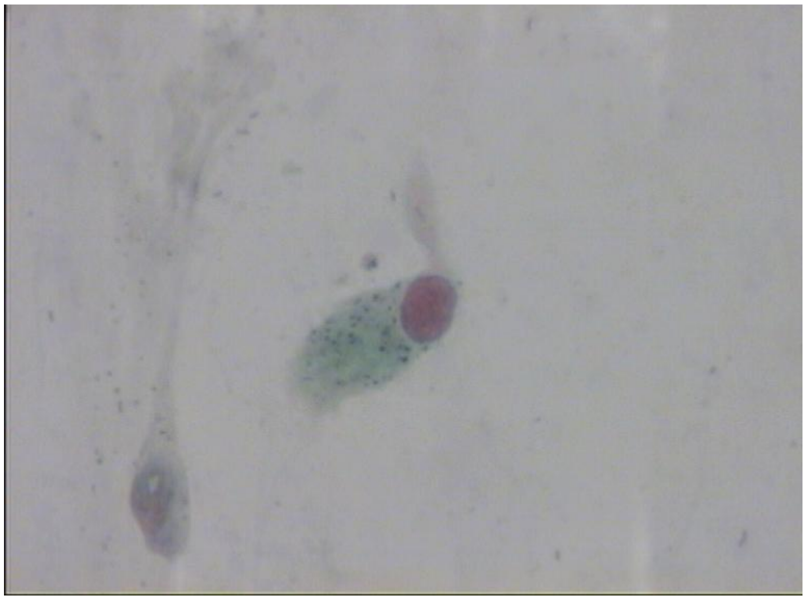
Average k means segmentation processing time (ms): 14

Total RunTime (without image exporting) (s): 216

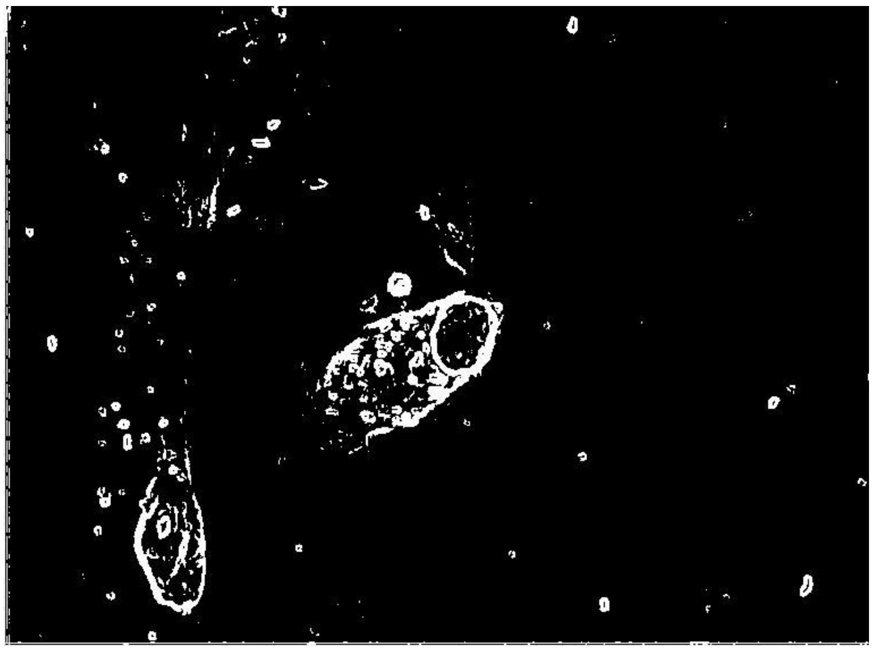
Real run time (s): 321

Results:

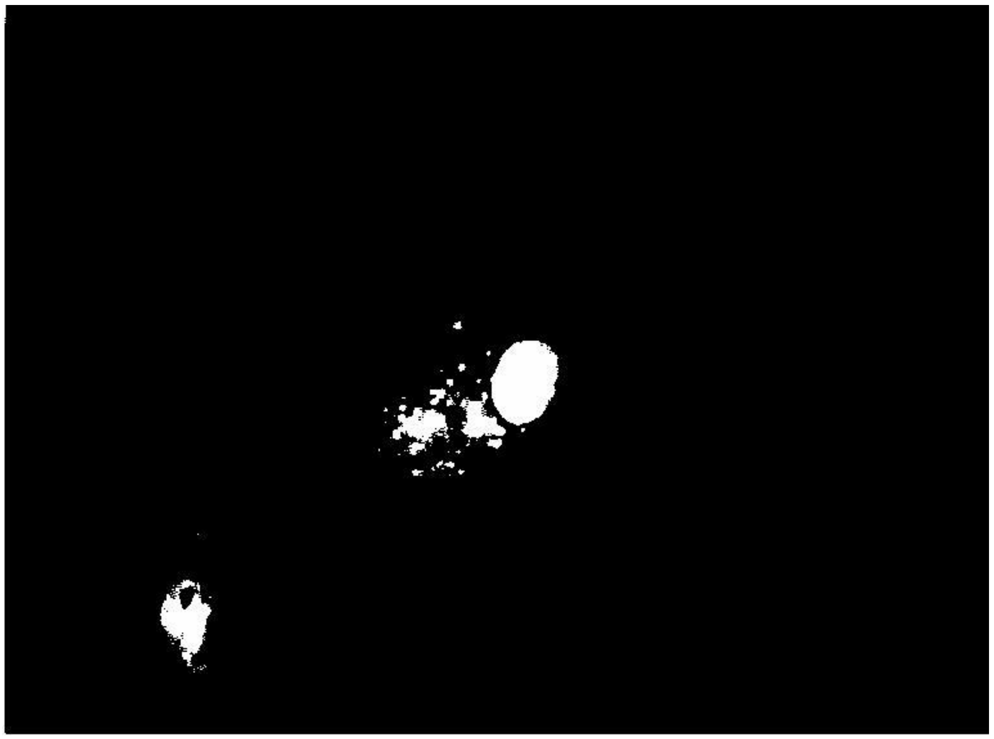
Original



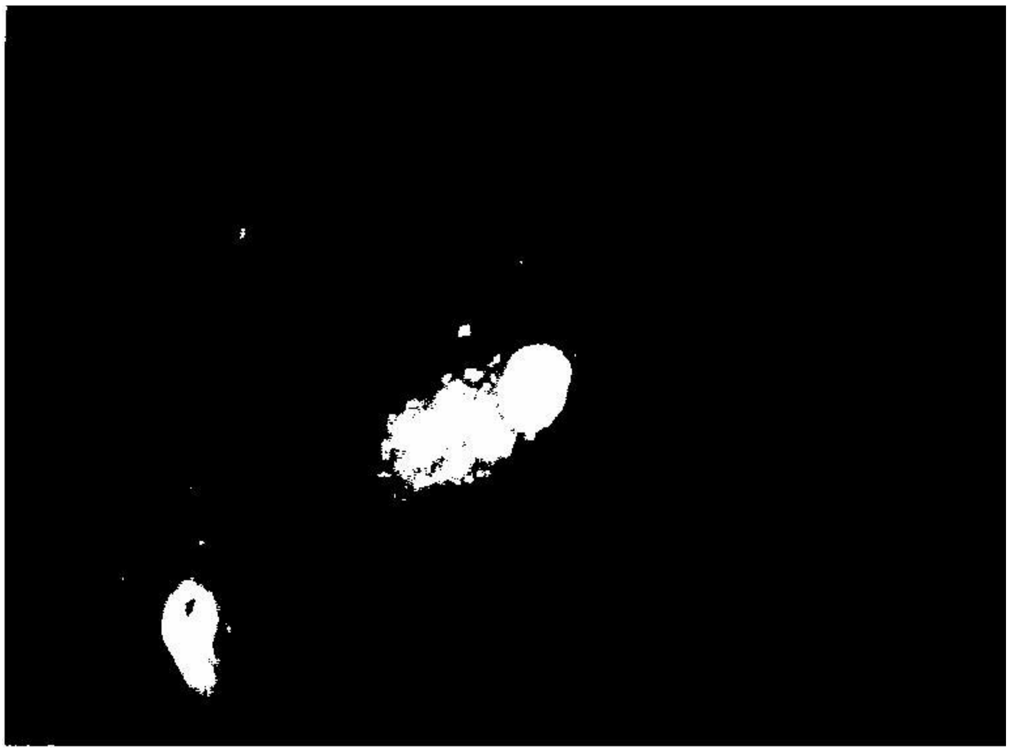
EdgeDetection



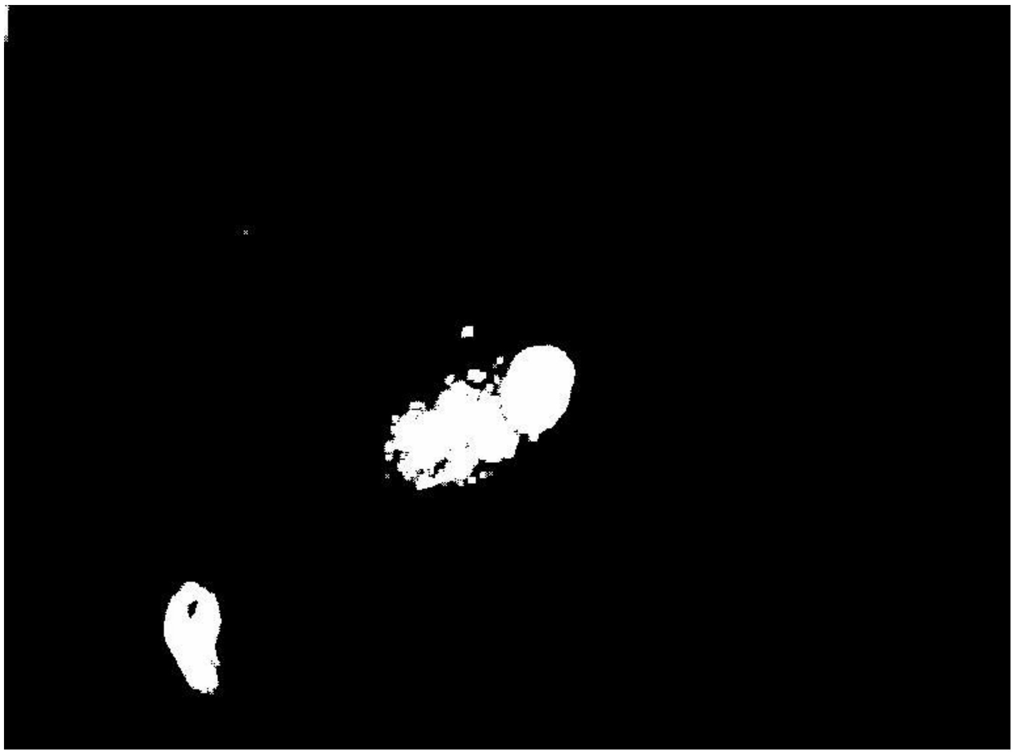
HistogramThresholdingSegmentation



KMeansSegmentation



Erosion



Dilation