Biometrics report#1

Tetiana Bakai

November 1, 2018

Contents

1 Grayscale conversion.....**2**

2 Color inversion and contrast modification.....**3**

3 Brightness modification.....**4**

4 Thresholding.....**5**

5 Horizontal/vertical projection.....**6**

6 Histogram manipulation.....**6**

6.1 Histogram manipulation.....7

6.1 Histogram stretching.....8

6.1 Histogram equalization.....10

7 Filters.....**9**

7.1 averaging filter.....9

7.2 Gauss filter.....10

7.3 sharpening filter.....10

7.4 edge detection filter.....10

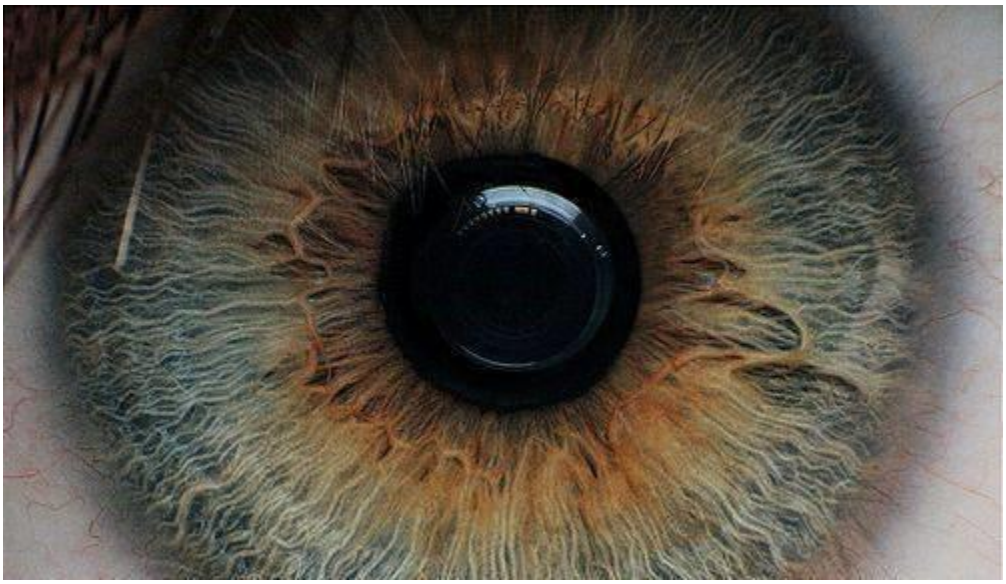
7.4.1 Roberts cross.....11

7.4.2 Sobel filter.....11

Conclusion.....**11**

All this report is about the different techniques of image processing that was applied on the image of human eye.

So here is our input image:



1 Grayscale conversion

All grayscale algorithms utilize the same basic three-step process:

1. Get the red, green, and blue values of a pixel
2. Use fancy math to turn those numbers into a single gray value
3. Replace the original red, green, and blue values with the new gray value

On step 2 I used the luminance method instead of treating red, green, and blue light equally (averaging method) since humans do not perceive all colors equally. A good grayscale conversion will weight each color based on how the human eye perceives it. A formula for this method is below:

```
int grey = (int) (r * 0.299 + g * 0.587 + b * 0.114)
```

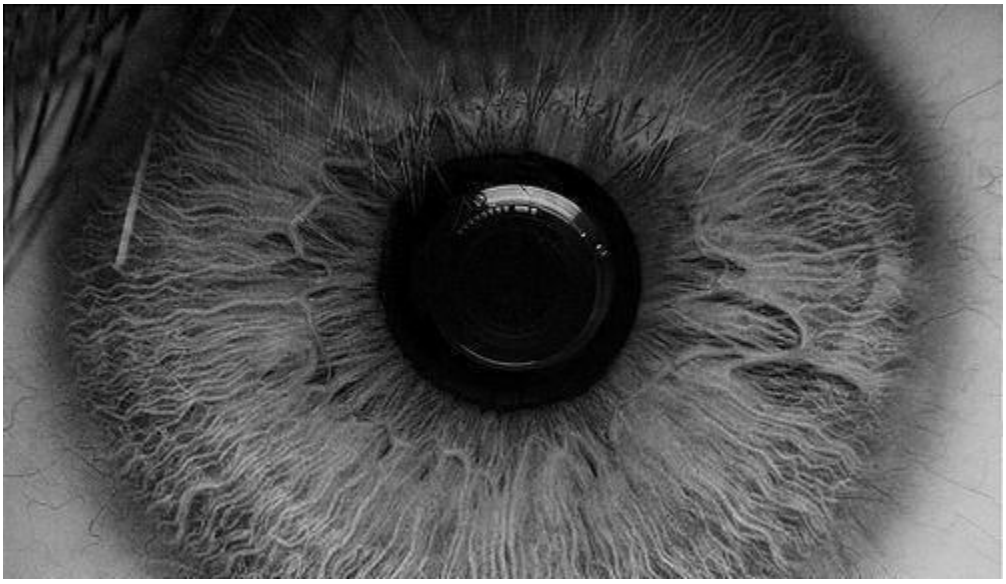
Code implementation:

```
int r = getR(p);
 int g = getG(p);
 int b = getB(p);
```

```
 //luminance method of conversion images to a greyscale
 int grey = (int) (r * 0.299 + g * 0.587 + b * 0.114);
 int a = toRGB(r, g, b);

 //replace RGB value with luminance method
 p = (a<<24) | (grey<<16) | (grey<<8) | grey;
img.setRGB(x, y, p);
```

Here is an output image of grayscale conversion:



2 Color inversion and contrast modification

Inversion means that each pixel is as close to the black as it was close to the white before the inversion. In order to modify a contrast, it is necessary to increase the differences between pixels.

Code Implementation:

```
if (r >= 256) {
 r = 255;
 } else if (r < 0) {
 r = 0;
 }

 if (g >= 256) {
 g = 255;
```

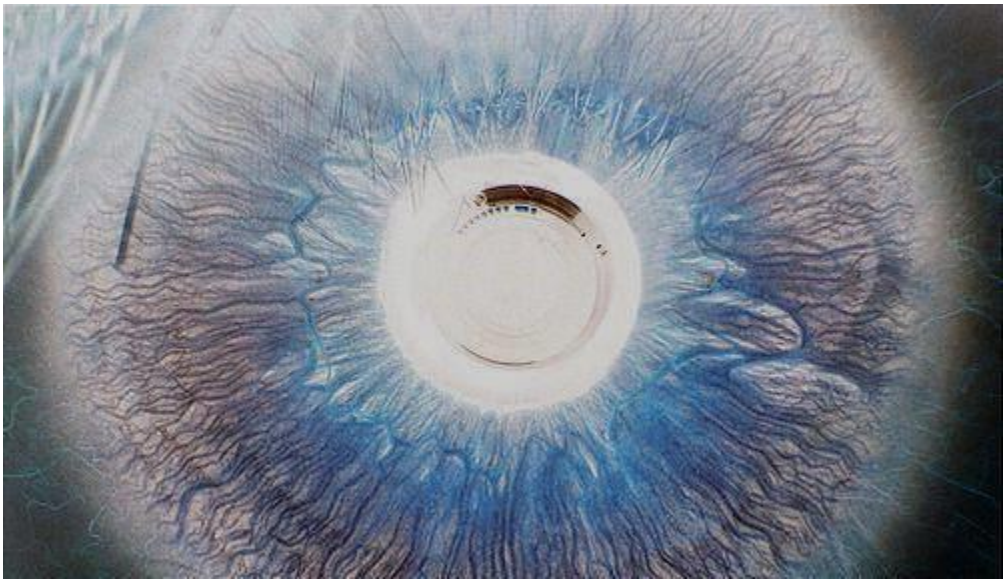
```
} else if (g < 0) {
g = 0;
}

if (b >= 256) {
b = 255;
} else if (b < 0) {
b = 0;
}

int colorInverse = toRGB(255 - r, 255 - g,        255 - b);

img.setRGB(x, y, colorInverse);
```

An output picture is below:



3 Brightness modification

In order to make the image brighter (closer to white) it is necessary to get brightness modification.

Code implementation:

```
int factor = 85;

//subtracting factor to rgb values
 int r = getR(p) - factor;
 int g = getG(p) - factor;
 int b = getB(p) - factor;
```

```
int brightness = toRGB(r, g, b);
img.setRGB(x, y, brightness);
```

Here is a result image:

4 Thresholding

Thresholding is e easiest form of binarization which means that we want to "classify" each pixel as either black or white and nothing in between. We can select some threshold T and if the value of a pixel (in a grayscale) is greater than the T classify it as white and if it's not – as black. The main idea is to separate out regions of an image corresponding to objects which we want to analyze. This separation is based on the variation of intensity between the object pixels and the background pixels.

Code implementation:

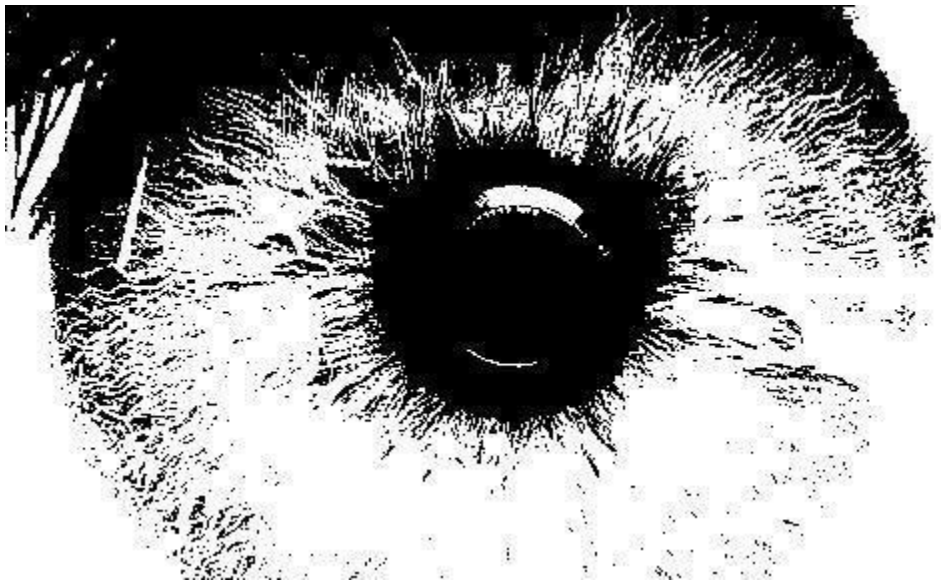
```
int threshold = 55;

if(green < threshold)
 {
 newPixel = 0;
 }
 else
 {
 newPixel = 255;
```

```
 }
 newPixel = colorToRGB(alpha, newPixel, newPixel, newPixel);
 binarized.setRGB(i, j, newPixel);
 }

private static int colorToRGB(int alpha, int red, int green, int blue) {
 int newPixel = 0;
 newPixel += alpha;
 newPixel = newPixel << 8;
 newPixel += red; newPixel = newPixel << 8;
 newPixel += green; newPixel = newPixel << 8;
 newPixel += blue;
 return newPixel;
 }
```

Here is a result picture:



5 Horizontal/Vertical projection

Horizontal or vertical projection may be used for analyzing the images. The idea is to count all pixels for each column (vertical projection) or row (horizontal projection).

6 Histogram manipulation

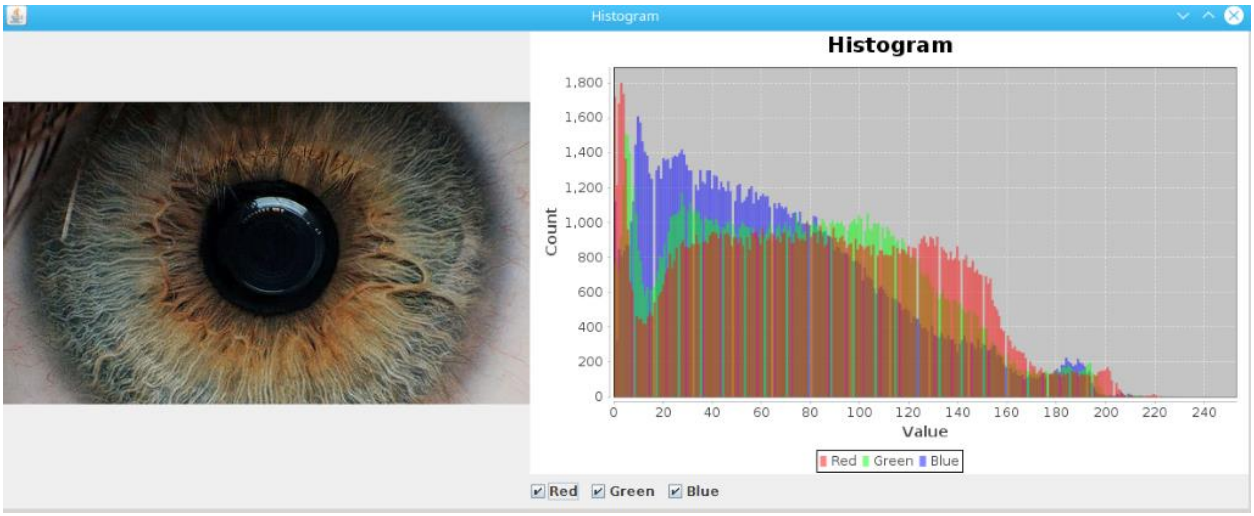
Histogram is basically a graphical representation of pixel values distribution. In order to build a histogram, it is necessary to find for each possible pixel value count how many pixels in the image equal to this value.

Code implementation:

```
private ChartPanel createChartPanel() {
    // dataset
    dataset = new HistogramDataset();
    Raster raster = image.getRaster();
    final int w = image.getWidth();
    final int h = image.getHeight();
    double[] r = new double[w * h];
    r = raster.getSamples(0, 0, w, h, 0, r);
    dataset.addSeries("Red", r, BINS);
    r = raster.getSamples(0, 0, w, h, 1, r);
    dataset.addSeries("Green", r, BINS);
    r = raster.getSamples(0, 0, w, h, 2, r);
    dataset.addSeries("Blue", r, BINS);
    // chart
    JFreeChart chart = ChartFactory.createHistogram("Histogram", "Value",
            "Count", dataset, PlotOrientation.VERTICAL, true, true, false);
    XYPlot plot = (XYPlot) chart.getPlot();
    renderer = (XYBarRenderer) plot.getRenderer();
    renderer.setBarPainter(new StandardXYBarPainter());
    // translucent red, green & blue
    Paint[] paintArray = {
            new Color(0x80ff0000, true),
            new Color(0x8000ff00, true),
            new Color(0x800000ff, true)
    };

    ChartPanel panel = new ChartPanel(chart);
    panel.setMouseWheelEnabled(true);
    return panel;
}
```

Here is an output image:



6.1 Histogram manipulation

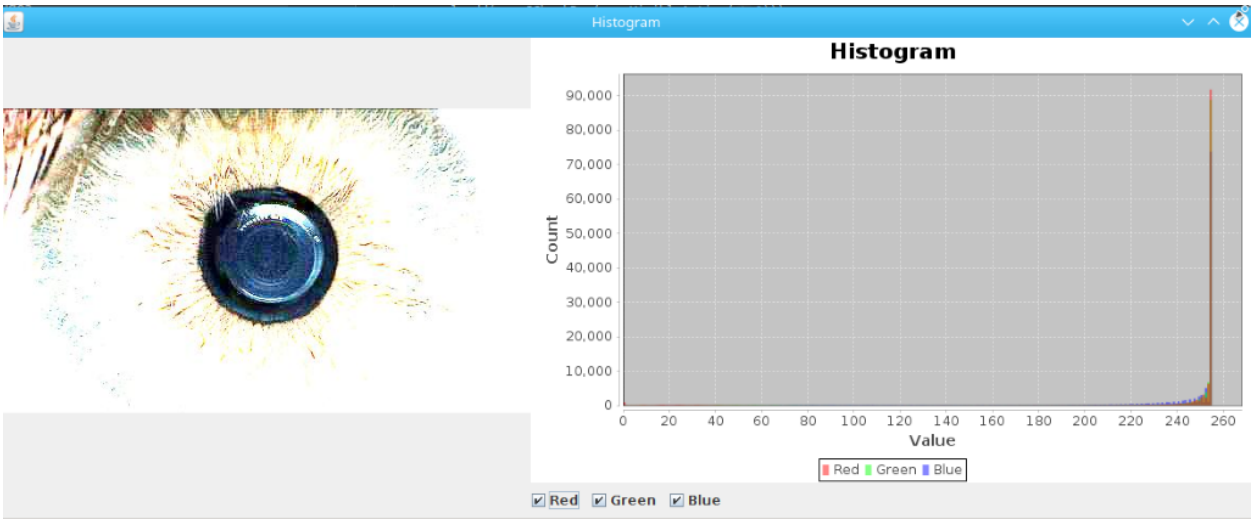
General way of manipulating the histogram is using a Lookup Table. Such table defines how to map each pixel in input image to pixel in output image by value. In case of RGB mages we would have three separate Lookup Tables for each channel. In general, Lookup Tables will have 256 elements.

6.2 Histogram stretching

Code implementation:

```
//stretch them to 0 to 255
 r[i] = (int) (1.0*( r[i] - min) / (max - min) * 255);
 g[i] = (int) (1.0*( g[i] - min) / (max - min) * 255);
 b[i] = (int) (1.0*( b[i] - min) / (max - min) * 255);
```

Here is an output image:



6.3 Histogram equalization

Code implementation:

```
int anzpixel = width * height;

int sum = 0;
// build a Lookup table LUT containing scale factor
int [] lut = new int[anzpixel];
int [] d = new int [anzpixel];
   for (i = 0; i < 255; ++i) {
   sum += histogram[i];
   d[i] = sum / anzpixel;
   lut[i] = (d[i] - d[0])/(1-d[0]) * 255;
```

7 Filters

Image filters, also known as kernels are simple, usually small matrices (e.g. 3x3).

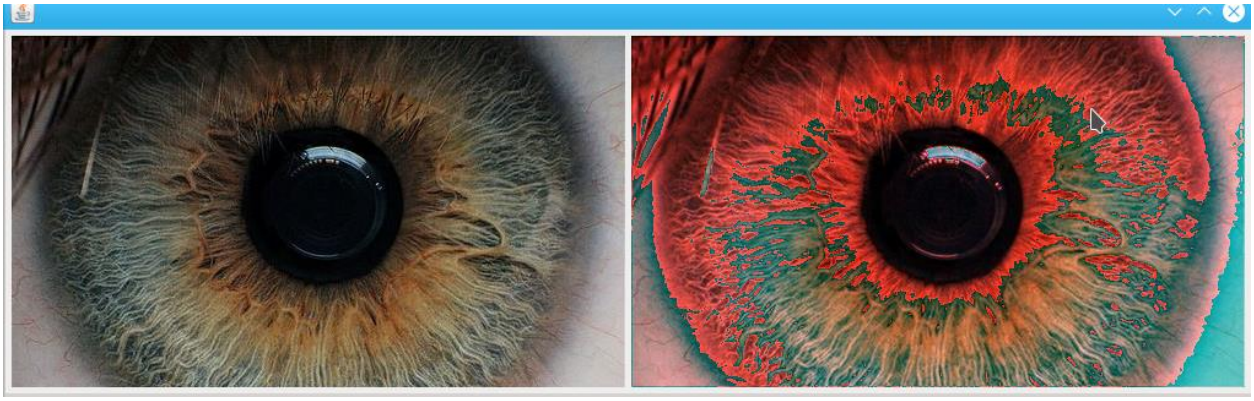
7.1 Averaging filter

Code implementation:

```
public static Matrix filter(int M, int N) {
   Matrix A = new Matrix(M, N);
   for (int i = 0; i < M; i++)
      for (int j = 0; j < N; j++)
         A.data[i][j] = (1.0 / 9.0);
   return A;
```

```
}
```

Here is an output image:



7.2 Gauss filter

Code implementation:

```
float[] matrix = {
        1/16f, 1/8f, 1/16f,
        1/8f, 1/4f, 1/8f,
        1/16f, 1/8f, 1/16f,
};
```

Here is an output image:

7.3 Sharpening filter

Code implementation:

```
Kernel kernel = new Kernel(3, 3, new float[] { -1, -1, -1, -1, 9, -1, -1,       -1, -1 });
```

7.4 Edge detection filter

7.4.2 Sobel filter

Code implementation:

```
int gy=(pixelMatrix[0][0]*-1)+(pixelMatrix[0][1]*-2)+(pixelMatrix[0][2]*-
1)+(pixelMatrix[2][0])+(pixelMatrix[2][1]*2)+(pixelMatrix[2][2]*1);
int gx=(pixelMatrix[0][0])+(pixelMatrix[0][2]*-1)+(pixelMatrix[1][0]*2)+(pixelMatrix[1][2]*-
2)+(pixelMatrix[2][0])+(pixelMatrix[2][2]*-1);
```

Conclusion:

During the performance of this lab the different techniques of image processing was applied on the image of human eye. The project is concentrated on “JAVA” on Linux platform. For grayscale conversion the luminance method was used since humans do not perceive all colors equally. In order to get a brightness modification, the value of 85 was taken for brightness at this project. During the binarization of the image a threshold value was equal to 55 since it gives the best separation of our object from the image background.