CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

1. Implementation Contrast Stretching command indo dihilangkan dan tambah gui ss

```
62      public void ContrasStretchh() {
            new Thread(new Runnable() {
                public void run() {
65
66
67              int width = img.getWidth();
68              int height = img.getHeight();
69
70              int min = 100;
71              int max = 200;
72
73              System.out.println("width=" + width + " height=" + height);
74              System.out.println("min=" + min + " max=" + max);
75
```

Illustration 5.1: Input Min Max Contrast Stretching

This is a part of the input image that will be processed, we use getWidth and getHeight to get an image input size. Then on line 70 and 71 are the min and max parameters to determine the stretching level, in this part we input min = 100 and max = 200.

```
77                  try {
78
79                      int[] r = new int[width * height];
80                      int[] g = new int[width * height];
81                      int[] b = new int[width * height];
82                      int[] e = new int[width * height];
83                      int[] data = new int[width * height];
84                      img.getRGB(0, 0, width, height, data, 0, width);
85
86                      for (int i = 0; i < (height * width); i++) {
87                          r[i] = (int) ((data[i] >> 16) & 0xff);
88                          g[i] = (int) ((data[i] >> 8) & 0xff);
89                          b[i] = (int) (data[i] & 0xff);
```

Illustration 5.2: Get RGB Image

To do Contrast Stretching method, first we make an array to load rgb value in line 79-81 then we make an array for load an image after processing/output int[] e and int[] data for load rgb data below on line 86-89.

```
93                      r[i] = (int) (1.0 * (r[i] - min) / (max - min) * 255);
94                      g[i] = (int) (1.0 * (g[i] - min) / (max - min) * 255);
95                      b[i] = (int) (1.0 * (b[i] - min) / (max - min) * 255);
96
97                      if (r[i] > 255) {
98                          r[i] = 255;
99                      }
100                     if (g[i] > 255) {
101                         g[i] = 255;
102                     }
103                     if (b[i] > 255) {
104                         b[i] = 255;
105                     }
106
107                     if (r[i] < 0) {
108                         r[i] = 0;
109                     }
110                     if (g[i] < 0) {
111                         g[i] = 0;
112                     }
113                     if (b[i] < 0) {
114                         b[i] = 0;
115                     }
```

Illustration 5.3: Contrast Stretching Formula

This is the contrast stretching formula on lines 93-95, we have previously determined the min and max values min = 100 and max = 200 if we want to stretch the intensity value in the range 0-255 each pixel must be reduced by the min value of 100 so that forming a range of 0-100. Then each pixel intensity multiplied by 255/100 to make the stretching range 0-255.

```
120
121                     }
122                     img.setRGB(0, 0, width, height, e, 0, width);
123     ImageIO.write(img, "jpg" ,new File("CStiger.jpg"));
124
125
```

Illustration 5.4: Output Image

This section is used to print the image output after we have contrasted the original image.

2. Implementation Histogram Equalization

```
64  public float [] getRGB(File file) throws IOException{
        BufferedImage buf = ImageIO.read(file);
66      int width = buf.getWidth();
67      int height = buf.getHeight();
68      int size = width * height;
        int c = 0, counter = 0,r,g,b;
70      float [] rgb = new float[size];
71      for(int i = 0; i<width; i++){
72          for(int j = 0; j<height ; j++){
73              c = buf.getRGB(i,j);
74              r = (c&0x00ff0000)>>16;
75              g= (c&0x0000ff00)>>8;
76              b = c&0x000000ff;
77              counter++;
78          }
79      }
80      return rgb;
81  }
```

Illustration 5.5: Read Image RGB

In this section it is used to get RGB image data by determining the size of input image then on line 71-76 used to obtain RGB values.

```
85      public float [] GRAYSCALE (File file) throws IOException{
            BufferedImage buf = ImageIO.read(file);
87          int width = buf.getWidth();
88          int height = buf.getHeight();
89          int size = width * height;
            int c = 0, counter = 0, r, g, b;
91          float [] grayScale = new float[size];
92          for(int i = 0; i<width; i++){
93              for(int j = 0; j<height ; j++){
94                  c = buf.getRGB(i,j);
95                  r = (c&0x00ff0000)>>16;
96                  g = (c&0x0000ff00)>>8;
97                  b = c&0x000000ff;
98                  grayScale[counter] = (float) (r+g+b)/3;
99                  counter++;
100             }
101         }
102         return grayScale;
103     }
```

Illustration 5.6: Read Image and Convert Grayscale

This code is used to get grayscale image data by determining the size of input image then on line 95-97 used to obtain an RGB value. After that, to produse grayscale image the rgb value is divided by 3 on line 98.

```
108     public int [] histogram(float[] grayScale){ //panjang histogram array gr
109         int [] pixNum = new int [256];
110         int size = grayScale.length;
111 
112         for(int c = 0; c<256; c++){
113             int sum = 0;
114             for(int i = 0; i<size; i++) if(grayScale[i]==c) sum++;
115             pixNum[c] = sum;
116             //System.out.println("picNum:"+pixNum);
117         }
118         return pixNum;
119     }
```

Illustration 5.7: Get Histogram and Pixel Frecuency.

The top part is used to make the range of grayscale histograms in the range 0-255. int size = grayscale.length is used to count the total frecuency of pixel then on line 112-115 to count the frecuency of each pixel.

```
124     public int [] getCDF(float[] grayScale,int [] histogram){
125         int [] cdf = new int[256];
            double [] Cp = new double [256];
            double [] floorRounding = new double [256];
128         int cum = 0;
            double kali=0;
            double frek =0;
            double probability =0;
            double pembulatan=0;
133            double size = grayScale.length;
134         for(int i = 0; i<256; i++){
135              probability = histogram[i]/size;
136             frek = histogram[i];
137             cum += probability;//
138             cdf[i] = cum;
139             kali = cdf[i]*20;
140             Cp[i]= kali;
141             pembulatan = cdf[i]*20;
142             floorRounding[i]= pembulatan;
143           System.out.println("pixel ke:"+i);
144           System.out.println("frek:"+frek);
145           System.out.println("probability:"+probability);
146           System.out.println("probability:"+String.format("%2.6f",probability));
147           System.out.println("total frek:"+size);
148           System.out.println("cumulative:"+cum);
149           System.out.println("CP:"+kali);
150           System.out.println("Pembulatan:"+String.format("%2.0f",pembulatan));
151         }
152         return cdf;
153     }
```

Illustration 5.8: Get Probability and Cumulative Probability.

The next part is to find the probability of each value on line 135 histogram[i] known as thepixel divided by size the total of pixel frequency. After that, to find the cumulative probability we count the probability plus next probability on line 137-138 then to upgrade the pixel value we can multiple the cumulative probability by 20 or etc.

```
176     public float[] ekualisasi(int [] cdf, int size,float [] probabilities,float [] cumulative){
177         int min = getMinCDF(cdf);
178             int max = getMaxCDF(cdf);
179 
180         float e [] = new float[256];
181        System.out.println("minimum: "+min);
182        System.out.println("maximum: "+max);
183        System.out.println("pictSize: "+size);
184         for(int i = 0; i<256; i++){
185             e[i] = (float)((((float)cdf[i]-min)/(float)size)*255);
186         }
187         for(int i = 0; i<256; i++){
188             if(e[i]<0) e[i]=0;
189             if(e[i]>255) e[i]=255;
190         }
191         return e;
192     }
```

Illustration 5.9: Histogram Equalization Formula.

This part is the histogram equalization formula on line 185, after we got the cdf or probabilty value it will minus by minimum probability then divided by the total pixel frecuency(size), to improve the pixel value to range 255 we can multiple it by 255.

```
207     public void buatGambar(float [] newGS, int w, int h)throws IOException{
208         int size = w*h;
209         int counter = 0;
210         //int c=0;
211         BufferedImage im = new BufferedImage(w,h,BufferedImage.TYPE_BYTE_GRAY);
212         WritableRaster raster = im.getRaster();
213         for(int i = 0; i<w; i++){
214             for(int j = 0; j<h; j++){
215                 raster.setSample(i, j, 0, newGS[counter]);
216                 counter++;
217             }
218 
219         ImageIO.write(buf, "jpg", new File("/home/gabriel/Desktop/Skripsi/CSHE/HEgirl9.jpg"));
220         }
221     }
```

Illustration 5.10: Make Output Image.

This code using to call the new image from histogram equalization then after processing it we got image output.

```
223     public void HE() throws IOException{
224         HE he = new HE();
225         File file = new File("/home/gabriel/Desktop/Skripsi/CSHE/girl.jpg");
226         BufferedImage x = ImageIO.read(file);
227         int width = x.getWidth();
228         int height = x.getHeight();
229         int size = width * height;
230         float getrgb [] = new float[size];
            float grayScale [] = new float[size];
            int histogram [] = new int[256];
            int cdf [] = new int[256];
            float probabilities [] =new float[256];
            float cumulative [] =new float[256];
            float ekualisasi [] = new float[256];
            float citraEkualisasi [] = new float[size];
238         getrgb = he.getRGB(file);
239         grayScale = he.GRAYSCALE(file);
240         histogram = he.histogram(grayScale);
241         cdf = he.getCDF(histogram);
242         probabilities=he.getProbabilities(cdf,size);
243         cumulative=he.getCumulative(cdf,probabilities,size);
244         ekualisasi = he.ekualisasi(cdf, size,probabilities,cumulative);
245         citraEkualisasi =he.citraEkualisasi(grayScale, ekualisasi, width, height);
246         he.buatGambar(citraEkualisasi, width, height);
247         int counter = 0;
248
249
250     }
```

Illustration 5.11: Display Programs.

In this section it is used to display all the results of the coding command made from lines 233-250, which is an array in the storage request and in lines 238-247 it is useful to require / display data in an array.

3. Implementasi PSNR

```
64  public void msepsnr(){
65
66
67  try{
68                  File f1 = new File("/home/gabriel/Desktop/Skripsi/coc3.jpg");
69                  File f2 = new File("/home/gabriel/Desktop/Skripsi/pubg.jpg");
70          img =ImageIO.read(f1);
71          img2 =ImageIO.read(f2);
                        } catch (Exception e) {
73    System.err.println("Error: " + e);
    Thread.dumpStack();
75
76  }
```

Illustration 5.12: Input Image PSNR

The above code is used to call and read original images and output images on line 68-71.

```
78      for (int i = 0; i < img.getHeight(); i++) {
79          for (int j = 0; j < img.getWidth(); j++) {
80          //  System.out.println("x,y: " + j + ", " + i); //
81            int pixel = img.getRGB(j, i);
82            //System.out.println("");
83          }
84      }
85
86
87      for (int i = 0; i < img2.getHeight(); i++) {
88          for (int j = 0; j < img2.getWidth(); j++) {
89          //  System.out.println("x,y: " + j + ", " + i); //
90            int pixel = img2.getRGB(j, i);
91            //System.out.println("");
92          }
93      }
```

Illustration 5.13: Get pixel from image 1 and image 2.

The above section is used to read the input and output images on the line 81 of the input image and line 90 for the output image.

```
95
96
97              assert (img.getType() == img2.getType()
98                      && img.getHeight() == img2.getHeight()
99                      && img.getWidth() == img2.getWidth());
100
101             double mse = 0;
102             int width = img.getWidth();
103             int height = img.getHeight();
104             Raster r1 = img.getRaster();
105             Raster r2 = img2.getRaster();
106             for (int j = 0; j < width; j++) {
107                 for (int i = 0; i < height; i++) {
108                     mse += Math.pow(r1.getSample(j, i, 0) - r2.getSample(j, i, 0), 2);
109                 }
110             }
111             System.out.println("MSE = " + mse);
112             mse /= (double) (width * height);
113
114             double psnr = Math.log10(Math.pow(255, 2) / mse); //angka,pangkat math pow
115             System.out.println("PSNR = " + psnr);
116
```

Illustration 5.14: MSE and PSNR formula.

This code above is to call image 1 and image 2 for finding psnr value, then before we execute in psnr formula we must find mse value because psnr needs the value of mse.

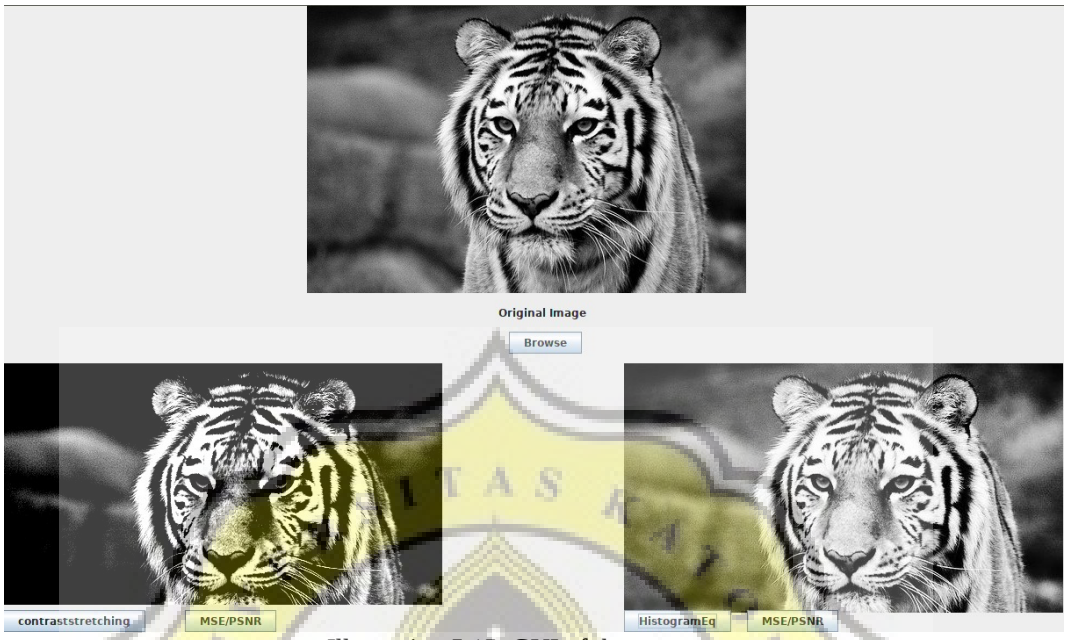
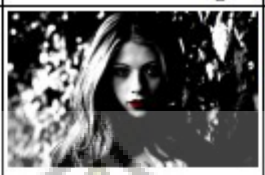
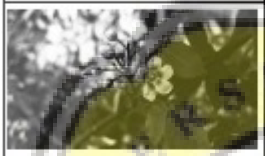
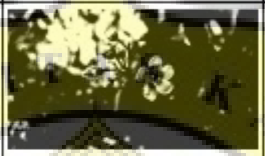
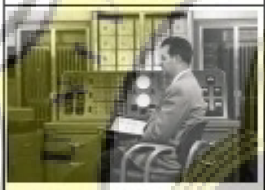
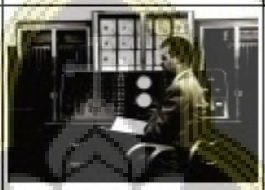
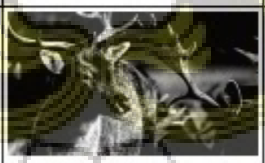


Illustration 5.15: GUI of the programs.

That's the user interface for Histogram Equalization and Contrast Stretching programs.The table above is the testing of contrast stretching program, we need an original image before the processing then we execute it in contrast stretching program.

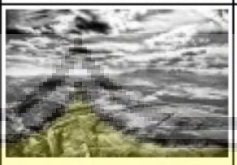
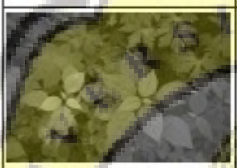
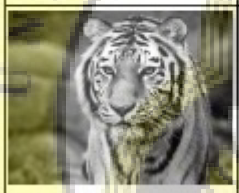
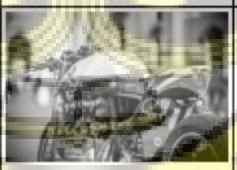
5.2 Testing

Table 5.1: Contrast Stretching Testing.

Original Image	Contrast Stretching	Statement
		CS = 100-200 to 0-255 CS PSNR =14.2996
		CS = 100-200 to 0-255 CS PSNR =14.0580
		CS = 100-200 to 0-255 CS PSNR =14.3375
		CS = 100-200 to 0-255 CS PSNR =12.5272
		CS = 100-200 to 0-255 CS PSNR =15.2729

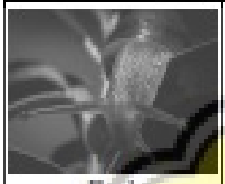
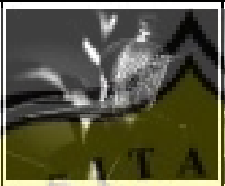
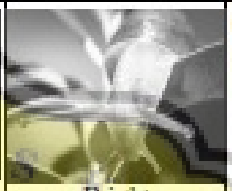
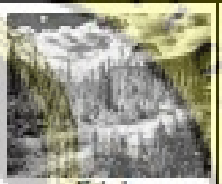
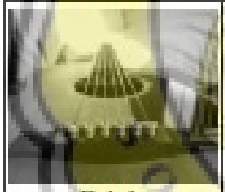
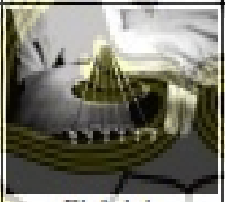
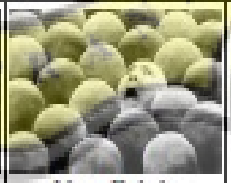
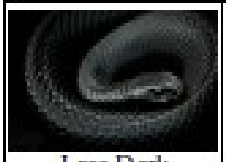
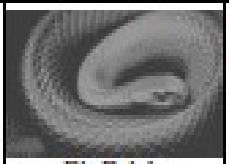
The table above is the testing of contrast stretching program, we need an original image grayscale before the processing then we execute it in contrast stretching program also the calculation program PSNR. The contrast stretching method making the original image more dark and the color sharp because the original image have dark quality and it stretched into range 0-255 so the pixel spread.

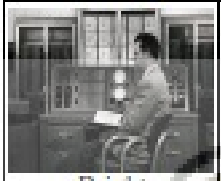
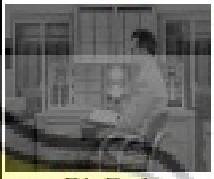
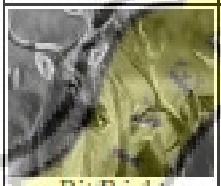
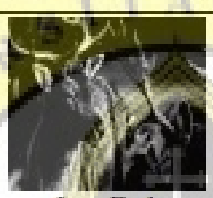
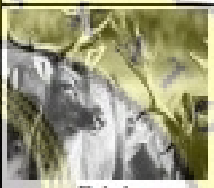
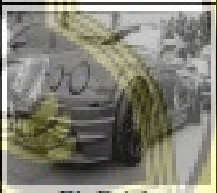
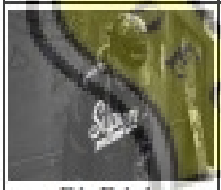
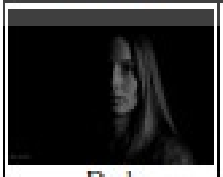
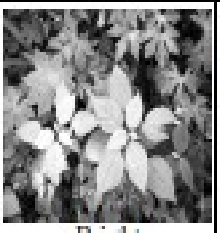
Table 5.2: Histogram Equalization Testing

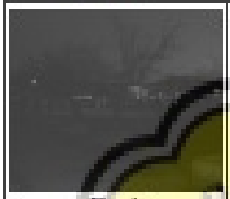
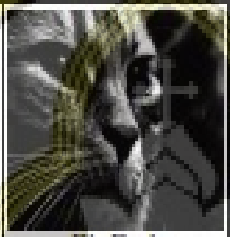
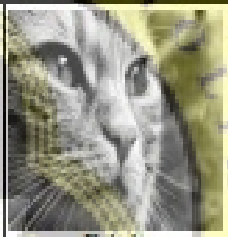
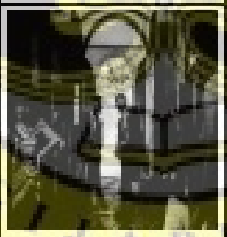
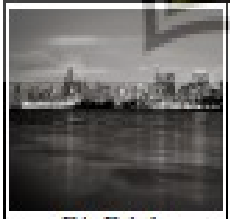
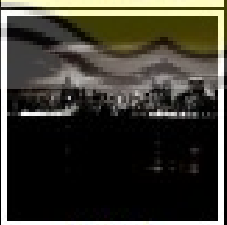
Original Image	Histogram Equalization	Statement
		HE = 0-255 PSNR = 9.8737
		HE = 0-255 PSNR = 14.3085
		HE = 0-255 PSNR = 10.1092
		HE = 0-255 PSNR = 19.3684
		HE = 0-255 PSNR = 20.5968

The table above is the testing of histogram equalization program, we need an original image grayscale before the processing then we execute it into histogram equalization program. The result of histogram equalization program compared with the original image to calculate the quality of image in Peak Signal Noise Ratio (PSNR). The historam equalization method making the original image more bright because the histogram equalization making pixel value intensity relative same then we improve the pixel range into 0-255.

Table 5.3: Contrast Stretching and Histogram Equalization Testing.

Original Image	Contrast Stretching (0-255)	Histogram Equalization (0-255)	Image statement (original image to CS/HE = PSNR)
 Dark	 Very Dark	 Bright	CS = 18.9234 HE = 8.2927
 Bit Bright	 Less Dark	 Bright	CS = 13.5666 HE = 18.0182
 Bright	 Bit bright	 Bright	CS = 15.5007 HE = 24.1742
 Less Dark	 Very Dark	 Very Bright	CS = 14.6607 HE = 8.5228
 Less Dark	 Very Dark	 Bit Bright	CS = 16.8913 HE = 10.2118
 Bit Bright	 Less Dark	 Bright	CS = 14.2996 HE = 17.6479

 Bit Bright	 Less Dark	 Bright	CS = 14.0580 HE = 13.0215
 Bright	 Less Dark	 Bit Dark	CS = 14.3375 HE = 14.4989
 Bit Bright	 Less Dark	 Bright	CS = 12.5272 HE = 15.6099
 Less Dark	 Dark	 Bit Bright	CS = 15.2729 HE = 12.2912
 Bit Bright	 Dark	 Bright	CS = 13.6363 HE = 8.2927
 Dark	 Very Dark	 Bit Bright	CS = 21.4673 HE = 2.4622
 Less Dark	 Bit Bright	 Bright	CS = 18.1581 HE = 10.1092

 Dark	 Very Dark	 Bright	CS = 22.9379 HE = 5.8686
 Dark	 Very Dark	 Bright	CS = 23.1814 HE = 5.7070
 Bit Bright	 Bit Dark	 Bright	CS = 13.8439 HE = 13.1167
 Bit Bright	 Bit Dark	 Bright	CS = 14.11951 HE = 12.6289
 Bit Bright	 Bit Dark	 Dark	CS = 11.9817 HE = 10.7636
 Bit Bright	 Bit Dark	 Bright	CS = 13.6799 HE = 13.0271

			CS = 14.1132 HE = 11.4518
Bit Bright	Dark	Bright	

The Testing above is to compare Contrast Stretching and Histogram Equalization image quality from the original image that having a bright or dark image. We know that Histogram Equalization is better than Contrast Stretching method becuse the output from Histogram Equalization have a bright quality. But when we calculate the quality of image to PSNR, Histogram Equalization cannot be said that is better than Contrast Stretching because Contrast Stretching have a high PSNR than Histogram Equalization. All depends on the original image having a dark or bright quality. This method Contrast Stretching and Histogram Equalization is compatible with dark image because the differences about both of method visible.