CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

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

```java
public void ContrasStretchh() {
    new Thread(new Runnable() {
        public void run() {
            int width = img.getWidth();
            int height = img.getHeight();
            int min = 100;
            int max = 200;
            System.out.println("width" + width + " height" + height);
            System.out.println("min" + min + " max" + max);

            try {
                int[] r = new int[width * height];
                int[] g = new int[width * height];
                int[] b = new int[width * height];
                int[] e = new int[width * height];
                int[] data = new int[width * height];
                img.getRGB(0, 0, width, height, data, 0, width);
                for (int i = 0; i < (height * width); i++) {
                    r[i] = (int) ((data[i] >> 16) & 0xff);
                    g[i] = (int) ((data[i] >> 8) & 0xff);
                    b[i] = (int) (data[i] & 0xff);
                }
            }
        }
    }).start();
}
```

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.
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.

```java
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);

if (r[i] > 255) {
    r[i] = 255;
}
if (g[i] > 255) {
    g[i] = 255;
}
if (b[i] > 255) {
    b[i] = 255;
}
if (r[i] < 0) {
    r[i] = 0;
}
if (g[i] < 0) {
    g[i] = 0;
}
if (b[i] < 0) {
    b[i] = 0;
}
```

Illustration 5.3: Contrast Stretching Formula

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

```java
img.setRGB(0, 0, width, height, e, 0, width);
ImageIO.write(img, "jpg", new File("CStiger.jpg"));
```

Illustration 5.4: Output Image
2. Implementation Histogram Equalization

public float [] getRGB(File file) throws IOException{
    BufferedImage buf = ImageIO.read(file);
    int width = buf.getWidth();
    int height = buf.getHeight();
    int size = width * height;
    int c = 0, counter = 0,r,g,b;
    float [] rgb = new float[size];
    for(int i = 0; i<width; i++){
        for(int j = 0; j<height ; j++){
            c = buf.getRGB(i,j);
            r = (c&0x0000ff0000)>>16;
            g = (c&0x0000ff0000)>>8;
            b = c&0x000000ff;
            counter++;
        }
    }
    return rgb;
}

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.

public float [] GRAYSCALE (File file) throws IOException{
    BufferedImage buf = ImageIO.read(file);
    int width = buf.getWidth();
    int height = buf.getHeight();
    int size = width * height;
    int c = 0, counter = 0, r, g, b;
    float [] grayScale = new float[size];
    for(int i = 0; i<width; i++){
        for(int j = 0; j<height ; j++){
            c = buf.getRGB(i,j);
            r = (c&0x0000ff0000)>>16;
            g = (c&0x0000ff0000)>>8;
            b = c&0x000000ff;
            grayScale[counter] = (float) (r+g+b)/3;
            counter++;
        }
    }
    return grayScale;
}

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.
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 frequency of pixel then on line 112-115 to count the frequency of each pixel.

Illustration 5.7: Get Histogram and Pixel Frequency.

The next part is to find the probability of each value on line 135 histogram[i] known as the pixel 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.

Illustration 5.8: Get Probability and Cumulative Probability.
This part is the histogram equalization formula on line 185, after we got the cdf or probability value it will minus by minimum probability then divided by the total pixel frequency(size), to improve the pixel value to range 255 we can multiple it by 255.

Illustration 5.9: Histogram Equalization Formula.

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

Illustration 5.10: Make Output Image.
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

```java
public void HE() throws IOException{
    HE he = new HE();
    File file = new File("/home/gabriel/Desktop/Skripsi/GSHM/girl.jpg");
    BufferedImage x = ImageIO.read(file);
    int width = x.getWidth();
    int height = x.getHeight();
    int size = width * height;
    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[size];
    float citraEkualisasi [] = new float[size];
    getrgb = he.getRGB(file);
    grayScale = he.GRAYSCALE(file);
    histogram = he.histogram(grayScale);
    cdf = he.getCDF(histogram);
    probabilities = he.getProbabilities(cdf, size);
    cumulative = he.getCumulative(cdf, probabilities, size);
    ekualisasi = he.ekualisasi(cdf, size, cumulative);
    citraEkualisasi = he.citraEkualisasi(grayScale, ekualisasi, width, height);
    he.buatGambar(citraEkualisasi, width, height);
    int counter = 0;
}
```

Illustration 5.11: Display Programs.

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

Illustration 5.12: Input Image PSNR
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.

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

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.
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.

<table>
<thead>
<tr>
<th>Original Image</th>
<th>Contrast Stretching</th>
<th>Statement</th>
</tr>
</thead>
</table>
| ![Image](image1.png) | ![Image](image2.png) | CS = 100-200 to 0-255  
CS PSNR = 14.2996 |
| ![Image](image3.png) | ![Image](image4.png) | CS = 100-200 to 0-255  
CS PSNR = 14.0580 |
| ![Image](image5.png) | ![Image](image6.png) | CS = 100-200 to 0-255  
CS PSNR = 14.3375 |
| ![Image](image7.png) | ![Image](image8.png) | CS = 100-200 to 0-255  
CS PSNR = 12.5272 |
| ![Image](image9.png) | ![Image](image10.png) | 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.
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 histogram 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>
<thead>
<tr>
<th>Original Image</th>
<th>Histogram Equalization</th>
<th>Statement</th>
</tr>
</thead>
</table>
| ![Original Image](image1) | ![Histogram Equalization](image2) | \( HE = 0-255 \)  
  \( PSNR = 9.8737 \) |
| ![Original Image](image3) | ![Histogram Equalization](image4) | \( HE = 0-255 \)  
  \( PSNR = 14.3085 \) |
| ![Original Image](image5) | ![Histogram Equalization](image6) | \( HE = 0-255 \)  
  \( PSNR = 10.1292 \) |
| ![Original Image](image7) | ![Histogram Equalization](image8) | \( HE = 0-255 \)  
  \( PSNR = 19.3684 \) |
| ![Original Image](image9) | ![Histogram Equalization](image10) | \( HE = 0-255 \)  
  \( PSNR = 20.5968 \) |
Table 5.3: Contrast Stretching and Histogram Equalization Testing.

<table>
<thead>
<tr>
<th>Original Image</th>
<th>Contrast Stretching (0-255)</th>
<th>Histogram Equalization (0-255)</th>
<th>Image statement (original image to CS/HE = PSNR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dark</td>
<td>Very Dark</td>
<td>Bright</td>
<td>CS = 18.9234, HE = 8.2927</td>
</tr>
<tr>
<td>Bit Bright</td>
<td>Less Dark</td>
<td>Bright</td>
<td>CS = 13.5666, HE = 18.0182</td>
</tr>
<tr>
<td>Bright</td>
<td>Bit bright</td>
<td>Bright</td>
<td>CS = 15.5007, HE = 24.1742</td>
</tr>
<tr>
<td>Less Dark</td>
<td>Very Dark</td>
<td>Very Bright</td>
<td>CS = 14.6607, HE = 8.5228</td>
</tr>
<tr>
<td>Less Dark</td>
<td>Very Dark</td>
<td>Bit Bright</td>
<td>CS = 16.8913, HE = 10.2118</td>
</tr>
<tr>
<td>Bit Bright</td>
<td>Less Dark</td>
<td>Bright</td>
<td>CS = 14.2996, HE = 17.6479</td>
</tr>
<tr>
<td>Bright</td>
<td>Less Dark</td>
<td>Bright</td>
<td>CS = 14.0580  HE = 13.0215</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Bright</td>
<td>Less Dark</td>
<td>Bit Dark</td>
<td>CS = 14.3375  HE = 14.4989</td>
</tr>
<tr>
<td>Bright</td>
<td>Less Dark</td>
<td>Bit Dark</td>
<td>CS = 12.5272  HE = 15.6099</td>
</tr>
<tr>
<td>Bit Bright</td>
<td>Less Dark</td>
<td>Bright</td>
<td>CS = 15.2729  HE = 12.2912</td>
</tr>
<tr>
<td>Less Dark</td>
<td>Dark</td>
<td>Bit Bright</td>
<td>CS = 13.6363  HE = 8.2527</td>
</tr>
<tr>
<td>Bit Bright</td>
<td>Dark</td>
<td>Bright</td>
<td>CS = 21.4673  HE = 2.4622</td>
</tr>
<tr>
<td>Dark</td>
<td>Very Dark</td>
<td>Bit Bright</td>
<td>CS = 18.1581  HE = 10.1092</td>
</tr>
<tr>
<td>Less Dark</td>
<td>Bit Bright</td>
<td>Bright</td>
<td>CS = 18.1581  HE = 10.1092</td>
</tr>
</tbody>
</table>
| Dark | Very Dark | Bright | CS = 22.9379  
HE = 5.8686 |
|------|----------|--------|---------------|
| Dark | Very Dark | Bright | CS = 23.1814  
HE = 5.7070 |
| Dark | Very Dark | Bright | CS = 13.8439  
HE = 13.1167 |
| Bright | Bit Dark | Bright | CS = 14.11951  
HE = 12.6289 |
| Bright | Bit Dark | Dark | CS = 11.9817  
HE = 10.7636 |
| Bright | Bit Dark | Bright | CS = 13.6799  
HE = 13.0271 |
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 because 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.