

CHAPTER 5

IMPLEMENTATION AND TESTING

5.1 Implementation

In this study using java programming that started with digital image input leaves. For this digital image input I use bufferedimage library, then convert the digital image into grayscale imagery.

```
1. File input = new File
   ("/home/yohan/Pictures/Project/Images/daun1.jpg");
2. img = ImageIO.read(input);
3. width = img.getWidth();
4. height = img.getHeight();
```

First and Second Lines contain commands for inputting the original image from the folder where the image is stored, while for third and fourth lines, it is useful to read the size of the inputted image starting from the width and length of the image.

```
5. for(int i=0; i<height; i++){
6.   for(int j=0; j<width; j++){
7.     Color c = new Color(img.getRGB(j, i));
8.     int red = (int) (c.getRed()*0.299);
9.     int green = (int) (c.getGreen()*0.587);
10.    int blue = (int) (c.getBlue()*0.114);
11.    int gray = (red + green + blue);
12.    Color newcolor = new Color(gray, gray, gray);
13.    img.setRGB(j, i, newcolor.getRGB());
14.  }
15. }
```

On fifth until seventh lines, it is useful to take each pixel on the image. For lines eighth to tenth, it is useful to take Red, Green and Blue colors, and for eleventh to thirteenth lines is the process by which to convert red, green and blue into grayscale shapes. It is then stored with an RGB composition value after divided by three.

1. Thresholding Process.

Then enter the Threshold process, in this Threshold begins by looking for the degree of grayness (Thres value) which is useful to separate between the object and the background.

```
1. public static int RGBVal(BufferedImage img, int width, int
   height, int i, int j) {
2.     i = Math.max(0, Math.min(width - 1, i));
3.     j = Math.max(0, Math.min(height - 1, j));
4.     return img.getRGB(i, j);
5. }
```

In rows 1 through 5 is a method to get every pixel on the image. In the 2nd row to get the pixel width while for row 3 to get the pixel length.

```
6. public static int[] histo(BufferedImage img,int width, int
   height) {
7.     int interval[] = new int[256];
8.     for (int i = 0; i < img.getWidth(); i++) {
9.         for (int j = 0; j < img.getHeight(); j++) {
10.            int p = RGBVal( img, width, height, i, j);
11.            int r = (p >> 16) & 255;
12.            interval[r]++;
13.        }
14.    }
15.    return interval;
16. }
```

For rows 6 through 16 is a method to search for histograms, in search of histograms required intervals, this interval is searched in rows 8 to row 12, after which the interval is returned for processing in the next to find the boundary between the background and foreground.

```
17. for (int i = 0; i < 256; i++) {
18.     wg_back += histogram[i];
19.     if (wg_back == 0) {
20.         continue;
21.     }
22.     wg_fore = total - wg_back;
23.     if (wg_fore == 0) {
24.         break;
25.     }
26.     sum_back += (float) (i * histogram[i]);
27.     float mean_back = sum_back / wg_back;
28.     float mean_fore = (sum - sum_back) / wg_fore;
29.     float varBetween = (float) wg_back * (float) wg_fore *
       (mean_back - mean_fore) * (mean_back - mean_fore);
30.     if (varBetween > varMax) {
31.         varMax = varBetween;
32.         threshold = i;
33.     }
```

34. }

In rows 18 through 35 this is a method of finding grayish values that are useful for the border between the background and the object you want to display (foreground). On the 18th to 26th row is sorting one by one per pixel to get the value of the pixel. Rows 20 and rows 21 are useful for finding weight background values. While row 23 sampai line 25 is to find the value of weight foreground. In rows 27 to 35 is the process by which the threshold values are searched by compared to one by one with the other pixels.

```

35. for (int i = 0; i < img.getWidth(); i++) {
36.     for (int j = 0; j < img.getHeight(); j++) {
37.         val = RGBVal(img,width,height, i, j);
38.         val = ((val >> 16) & 0xff);
39.         if (val > thresValue) {
40.             th = 255;
41.         } else {
42.             th = 0;
43.         }
44.         th = (th << 16) | (th << 8) | (th);
45.         Output.setRGB(i, j, th);
46.     }
47. }

```

Rows 36 through 48 are methods to change RGB values to black and white. The RGB value here is symbolized by the variable 'val' with the condition that if the val value exceeds the Threshold value then the color is changed to white while for val value less than threshold value will be changed to black. The method for changing the color is in rows 40 through 44. will then be stored in RGB with the new composition. This method is on rows 45 and 46.

2. Proses Morphological (Closing)

In this morphology process is the same as thresholding, starting with making the image into grayscale which then goes into the selection of the shape of the image structure to be tested.

```

1. int size = 2 * shpSize + 1;
2. short[][] struct = new short[size][size];
3. switch (shape) {
4.     case SQUARE:
5.         for (int i = 0; i < size; i++) {
6.             for (int j = 0; j < size; j++) {
7.                 struct[i][j] = 1;
8.             }

```

```

9.     }
10.    break;
11.    case VERTICAL:
12.        for (int i = 0; i < size; i++) {
13.            struct[i][shpSize] = 1;
14.        }
15.    break;
16.    case HORIZONTAL:
17.        for (int i = 0; i < size; i++) {
18.            struct[shpSize][i] = 1;
19.        }
20.    break;
21.    default:
22.        for (int i = 0; i < size; i++) {
23.            for (int j = 0; j < size; j++) {
24.                struct[i][j] = 1;
25.            }
26.        }
27.    }
28.    return struct;

```

In this method is a form of structure that will be used to add or subtract pixels to be executed.

```

1. for (int x = 0; x <= filWidth; x++) {
2.     for (int y = 0; y <= filHeight; y++) {
3.         window = (byte[]) oldData.getDataElements(x, y, sSize,
4.             sSize, null);
5.         newValue = max(window);
6.         newData.setSample(x + shpSize, y + shpSize, 0, newValue);
7.     }

```

In rows 1 through 7 is the image dilation method in the middle. On the 3rd row to retrieve the old data element starting from the width and length and also the forming element. After dilation in the middle of the data will be saved with a new composition in the 5th row.

```

8. for (int x = 0; x < shpSize; x++) {
9.     for (int y = 0; y <= filHeight; y++) {
10.        window = (byte[]) oldData.getDataElements(0, y, sSize,
11.            sSize, null);
12.        newValue = max(window);
13.        newData.setSample(x, y + shpSize, 0, newValue);
14.    }
15.    newData.setSamples(0, LoSide, shpSize, shpSize, 0,
16.        fillArray(shpSize * shpSize, newValue));
17.    window = (byte[]) oldData.getDataElements(0, 0, sSize, sSize,
18.        null);
19.    newValue = max(window);
20.    newData.setSamples(0, 0, shpSize, shpSize, 0, fillArray(shpSize
21.        * shpSize, newValue));

```

Rows 8 through 18 are methods for dilation on the left edge of an object. In the 10th row it is useful to retrieve the old data element starting from the length and width to then stored in the 12th row. And for the 15th row it is useful to add data and in line to 16 it is useful to retrieve the old data and coupled with the new data to be stored in the 18th row.

```
19. for (int x = RSide; x < imgWidth; x++) {
20.   for (int y = 0; y <= filHeight; y++) {
21.     window = (byte[]) oldData.getDataElements(filWidth, y,
22.       sSize, sSize, null);
23.     newValue = max(window);
24.     newData.setSample(x, y + shpSize, 0, newValue);
25.   }
26.   newData.setSamples(RSide, LoSide, shpSize, shpSize, 0,
27.     fillArray(shpSize * shpSize, newValue));
```

Rows 19 through 26 are methods for dilation on the right edge of an object. On the 21st row it is useful to retrieve old data and then stored in the 23rd row. after that the 26th line is useful for adding data by modifying arrays in new data.

```
27. for (int y = LoSide - 1; y < imgHeight; y++) {
28.   for (int x = 0; x <= filWidth; x++) {
29.     window = (byte[]) oldData.getDataElements(x, filHeight,
30.       sSize, sSize, null);
31.     newValue = max(window);
32.     newData.setSample(x + shpSize, y, 0, newValue);
33.   }
```

In lines 27 through 33 is a method to dilation on the bottom edge of the object. In row 29 is a method to retrieve old data and stored in method in row 31.

```
34. for (int y = 0; y < shpSize; y++) {
35.   for (int x = 0; x <= filWidth; x++) {
36.     window = (byte[]) oldData.getDataElements(x, 0, sSize, sSize,
37.       null);
38.     newValue = max(window);
39.     newData.setSample(x + shpSize, y, 0, newValue);
40.   }
41.   newData.setSamples(RSide, 0, shpSize, shpSize, 0,
42.     fillArray(shpSize * shpSize, newValue));
```

Rows 34 through 41 are methods for dilasiing the upper edges of objects. On the 36th row it is useful to retrieve the old data as well as the structur element data and then stored in the 38th row. then lined up to 41 is useful for adding data by modifying arrays already stored in the new data.


```

1.   for (int x = 0; x <= filWidth; x++) {
2.       for (int y = 0; y <= filHeight; y++) {
3.           window = (byte[]) oldData.getDataElements(x, y, sSize,
4.               sSize,null);
5.           newValue = min(window);
6.           newData.setSample(x + shpSize, y + shpSize, 0, newValue);
7.       }
8.   }

```

In rows 1 to 7 is a method to do erosion in the middle of the image. On the 3rd row it is useful to retrieve old data and be combined with the forming element and then stored in the 5th row.

```

8.   for (int x = 0; x < shpSize; x++) {
9.       for (int y = 0; y <= filHeight; y++) {
10.          window = (byte[]) oldData.getDataElements(0, y, sSize,
11.              sSize, null);
12.          newValue = min(window);
13.          newData.setSample(x, y + shpSize, 0, newValue);
14.      }
15.  }
16.  newData.setSamples(0, LoSide, shpSize, shpSize,
17.      0,fillArray(shpSize * shpSize, newValue));
18.  window = (byte[]) oldData.getDataElements(0, 0,
19.      sSize,sSize,null);
20.  newValue = min(window);
21.  newData.setSamples(0, 0, shpSize, shpSize,0,
22.      fillArray(shpSize*shpSize,newValue));

```

In lines 8 to 18 is a method used to erosion the edge of the drawing line on the left. On the 10th row it is useful to retrieve old data by adding data from the structur element to be subsequently stored in the 12th row. For the 15th row is new data by adding pixels and then stored in the 18th row.

```

19. for (int x = RSide; x < imgWidth; x++) {
20.     for (int y = 0; y <= filHeight; y++) {
21.         window = (byte[]) oldData.getDataElements(filWidth, y,
22.             sSize, sSize, null);
23.         newValue = min(window);
24.         newData.setSample(x, y + shpSize, 0, newValue);
25.     }
26. }
27. newData.setSamples(RSide, LoSide, shpSize, shpSize, 0,
28.     fillArray(shpSize * shpSize, newValue));

```

In lines 19 to 26 is a method used to erosion the right edge. In row 21 is a method to retrieve old data with structur element and stored in the 23rd row. on the 26th row is new data to add pixels on the right side.

```

27. for (int y = LoSide - 1; y < imgHeight; y++) {
28.     for (int x = 0; x <= filWidth; x++) {

```

```

29.     window = (byte[]) oldData.getDataElements(x, filHeight,
        sSize, sSize, null);
30.     newValue = min(window);
31.     newData.setSample(x + shpSize, y, 0, newValue);
32. }
33. }

```

In rows 27 to 33 is a method to do erosion on the lower edges. On the 29th row it is useful to retrieve the old data then stored back in the 31st row.

```

34. for (int y = 0; y < shpSize; y++) {
35.     for (int x = 0; x <= filWidth; x++) {
36.         window = (byte[]) oldData.getDataElements(x, 0, sSize,
            sSize, null);
37.         newValue = min(window);
38.         newData.setSample(x + shpSize, y, 0, newValue);
39.     }
40. }
41. newData.setSamples(RSide, 0, shpSize, shpSize, 0,
    fillArray(shpSize * shpSize, newValue));
42. return erodedImg;

```

In rows 34 to 42 is a method to do erosion at the top edge. On the 36th row it is useful to retrieve old data with structur element and stored in the 38th row. line 41 is useful for creating new data by adding pixels.

```

43. BufferedImage dilatedImg, closedImg;
44. Dilation dilation = new Dilation(shape, shapeSize);
45. Erosion erosion = new Erosion(shape, shapeSize);
46. dilatedImg = dilation.execute(img);
47. closedImg = erosion.execute(dilatedImg);

```

Line 44 is a method for creating dilation classes, and row 45 is for creating erosion classes. On line 46 is a method to call the dilation class and 47 to call the erosion class with the dilation image.

3. PSNR dan MSE

After both images are entered into the algorithm, it will be compared to determine which algorithm is good for segmenting the image of the leaves.

```

1.   for(int i=0;i<height;i++)
2.   {
3.       for(int j=0;j<width;j++)
4.       {
5.           Color P1 = new Color(Img1.getRGB(j,i));
6.           Color P2 = new Color(Img2.getRGB(j,i));
7.           double R_P1 = (double) (P1.getRed());
8.           double R_P2 = (double) (P2.getRed());
9.           double G_P1 = (double) (P1.getGreen());
10.          double G_P2 = (double) (P2.getGreen());
11.          double B_P1 = (double) (P1.getBlue());
12.          double B_P2 = (double) (P2.getBlue());
13.          if(R_P1>R_img1)
14.          {
15.              R_img1=R_P1;
16.          }
17.          if(G_P1>G_img1)
18.          {
19.              G_img1=G_P1;
20.          }
21.          if(B_P1>B_img1)
22.          {
23.              B_img1=B_P1;
24.          }
25.          if(R_P2>R_img2)
26.          {
27.              R_img2=R_P2;
28.          }
29.          if(G_P2>G_img2)
30.          {
31.              R_img2=R_P2;
32.          }
33.          if(B_P2>B_img2)
34.          {
35.              R_img2=R_P2;
36.          }
37.          R_MSE=R_MSE+Math.pow((R_P2-R_P1),2);
38.          G_MSE=G_MSE+Math.pow((G_P2-G_P1),2);
39.          B_MSE=B_MSE+Math.pow((B_P2-B_P1),2);
40.      }
41.  }

```

In Rows 5 to 12 is a method to get RGB from each image 1 or 2. For lines 13 to 36 is a comparison of RGB between image 1 and image 2. And on the 37th to 39th rows is the result of saving the MSE value of each RGB.

```

42.   R_MSE = R_MSE/(width*height);
43.   G_MSE = G_MSE/(width*height);

```

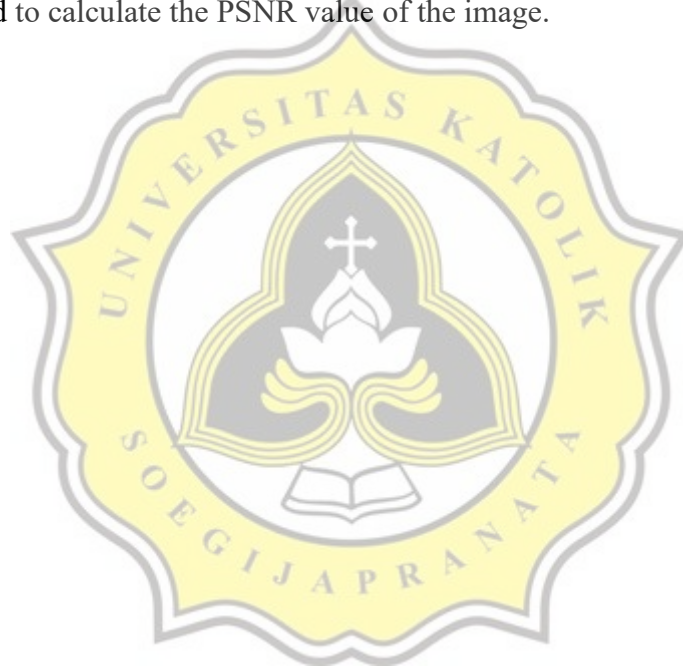


```

44.   B_MSE = B_MSE/(width*height);
45.   MSE = (R_MSE+G_MSE+B_MSE)/3;
46. R_PSNR = 10.0 * logbase10(Math.pow(Math.max(R_img1,R_img2), 2)
   / R_MSE);
47. G_PSNR = 10.0 * logbase10(Math.pow(Math.max(G_img1,G_img2), 2)
   / G_MSE);
48. B_PSNR = 10.0 * logbase10(Math.pow(Math.max(B_img1,B_img2), 2)
   / B_MSE);
49.   PSNR = (R_PSNR+G_PSNR+B_PSNR)/3;

```






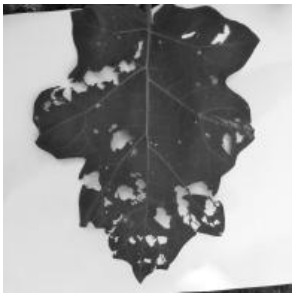
on lines 42 through 44 is the MSE value of each RGB divided by the size of the image. While on the 45th row is the MSE value of the image. On the 46th to 48th row is a method to calculate the PSNR value of each RGB and on the 49th row is a method to calculate the PSNR value of the image.

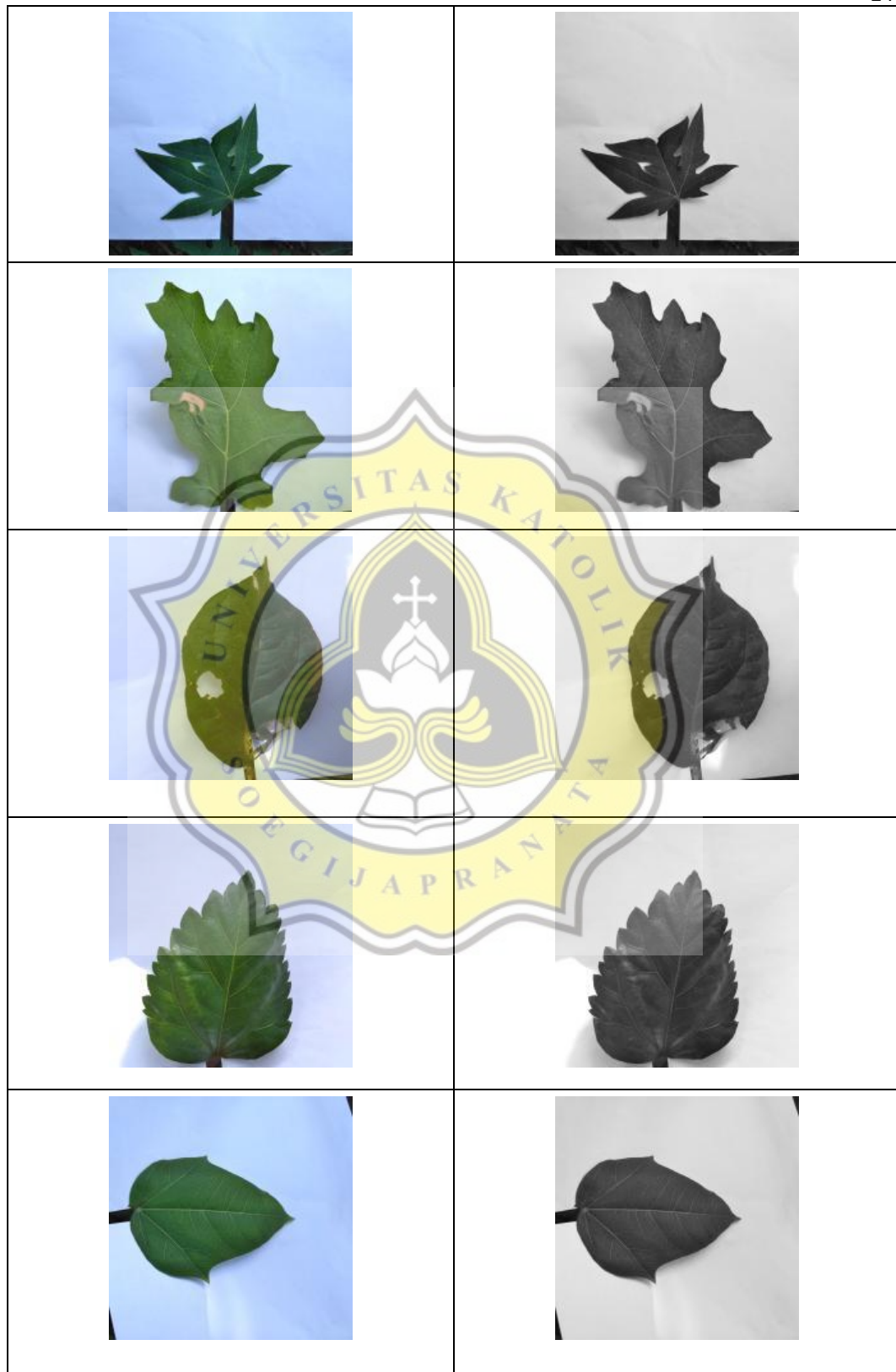


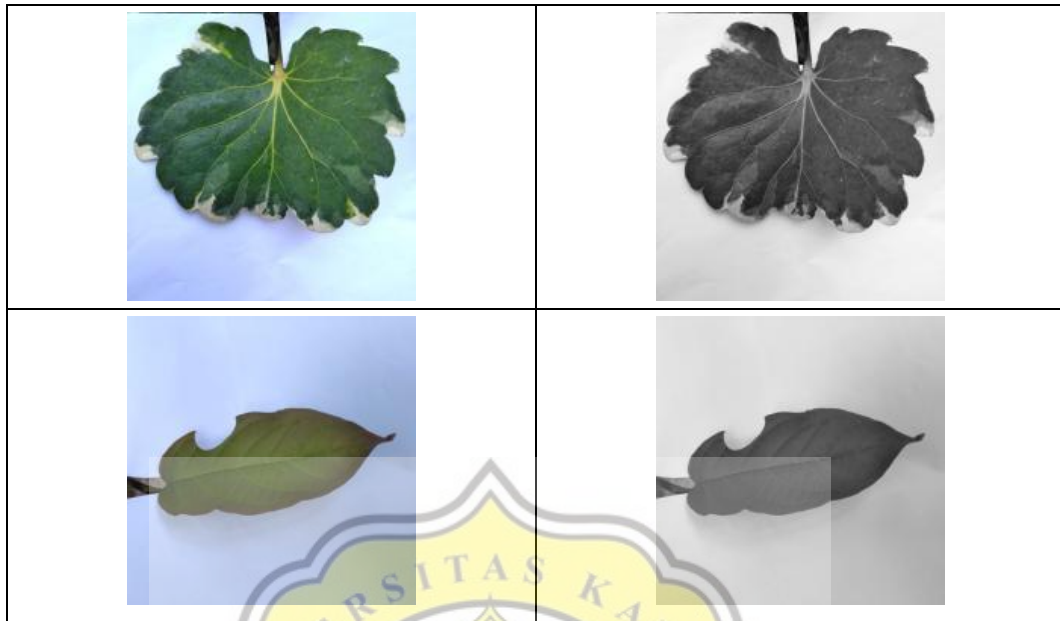
5.2 Testing

5.2.1. Grayscale

The first process that will be done is to convert the original image to grayscale image. This process starts with the original image input to be further detected the value of the RGB and will be summed and red pixel * 0,299, Green pixel * 0,587 and Blue pixel * 0,114.

Original Image	Grayscale Image
	
	
	



















5.2.1. Original Image to Grayscale Image

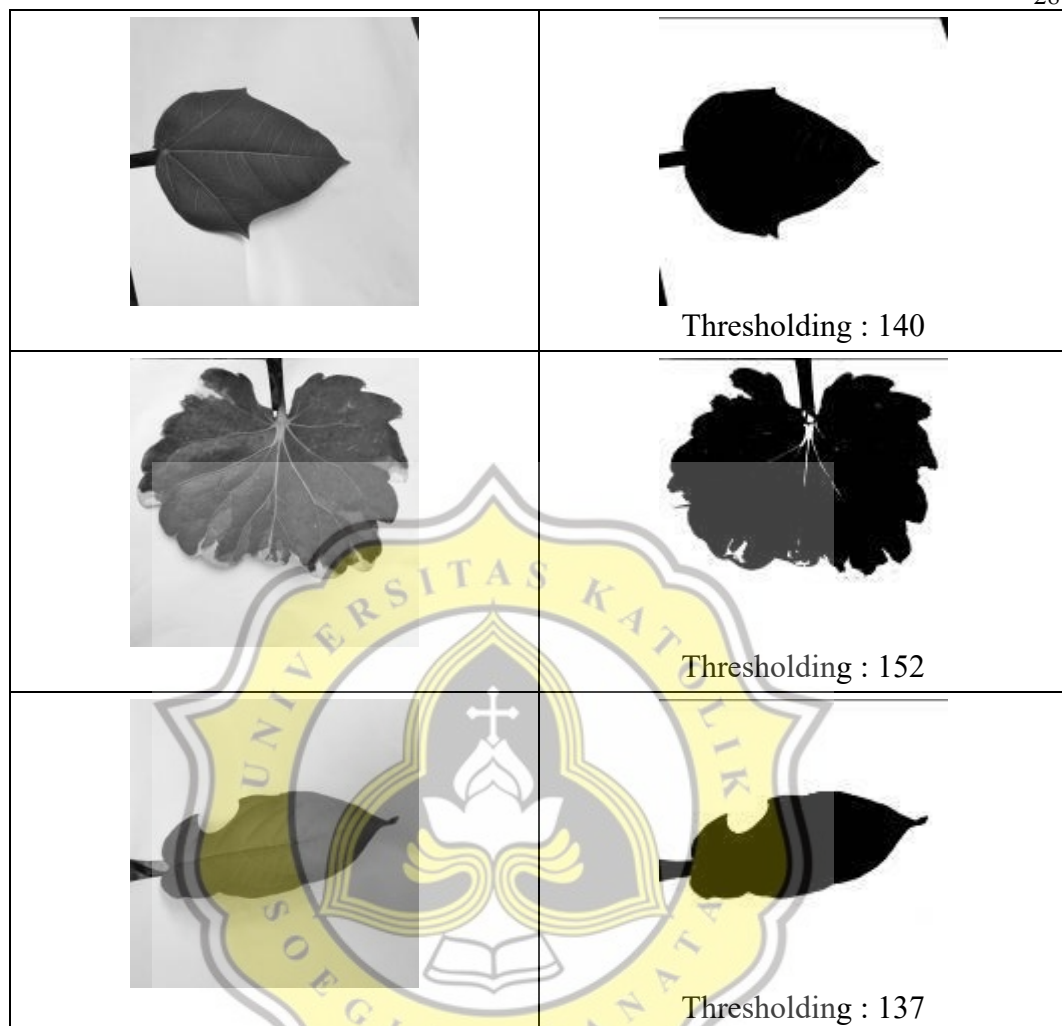
5.2.2. Thresholding

For the next step go to the Threshold algorithm. The first thing to do is to look for threshold value (degree of grayness) as the boundary between the background and the object. In looking for this Threshold value, it takes a histogram calculation to know the frequency of RGB values from images. After obtaining the number of frequencies of these values, the limit will be calculated by calculating weight background, weight foreground, mean background, mean foreground and variance background and also variance foreground. After that it will be looped in each pixel to find the minimum frequency value as the boundary between the background and also foreground or often referred to as the degree of Grayness.

After obtaining a degree of grayness or often referred to as the T value, the value will be the limit and if the RGB value exceeds the T value then the RGB value will turn white, while the RGB value less than T will change to black.

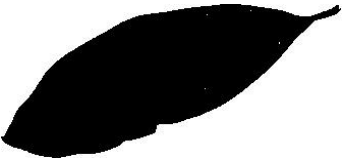
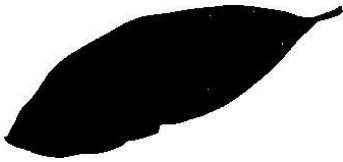






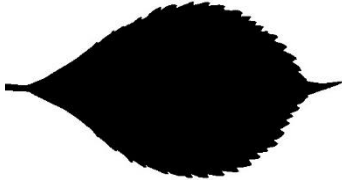
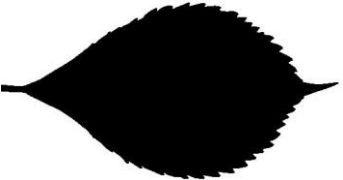
Grayscale Image	Thresholding Image
	 <p data-bbox="986 869 1228 902">Thresholding : 151</p>
	 <p data-bbox="986 1301 1228 1335">Thresholding : 132</p>
	 <p data-bbox="986 1646 1228 1680">Thresholding : 144</p>

	 Thresholding : 126
	 Thresholding : 144
	 Thresholding : 136
	 Thresholding : 144



5.2.2. Grayscale Image to Thresholding Image

In the experiment, a writer too uses biner images. A was undertaken almost similar but not through the grayscale, or like directly to the algorithm. A binary image is done to enter into algorithms. The next image will be handled by the algorithms and images are going out with the new and different kept folder.


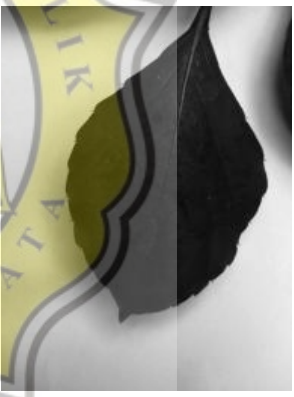


Binary Images	Threshold Image
	 Thresholding : 12
	 Thresholding : 12
	 Thresholding : 12
	 Thresholding : 11
	 Thresholding : 11

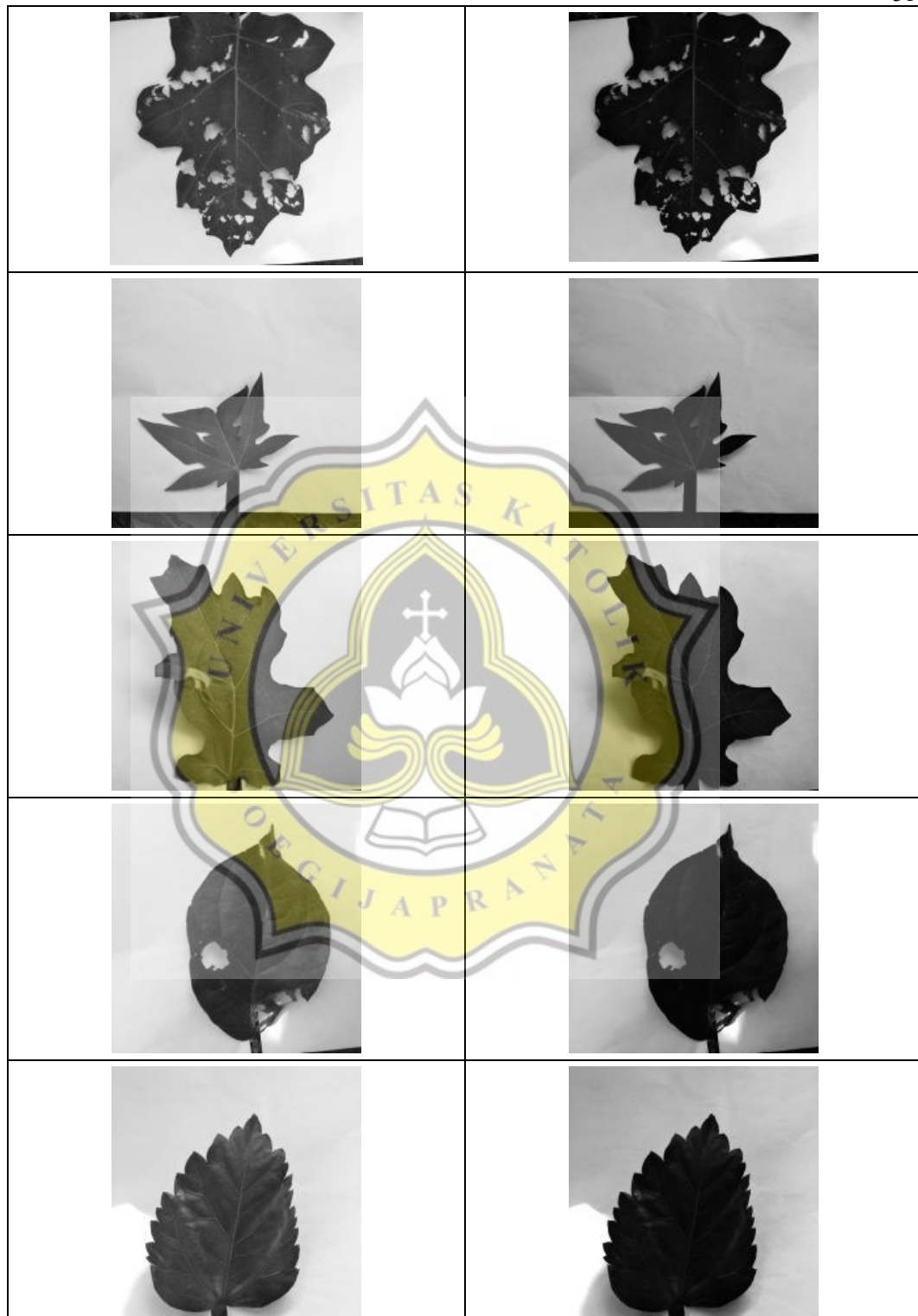
5.2.3. Binary Image to Thresholding Image

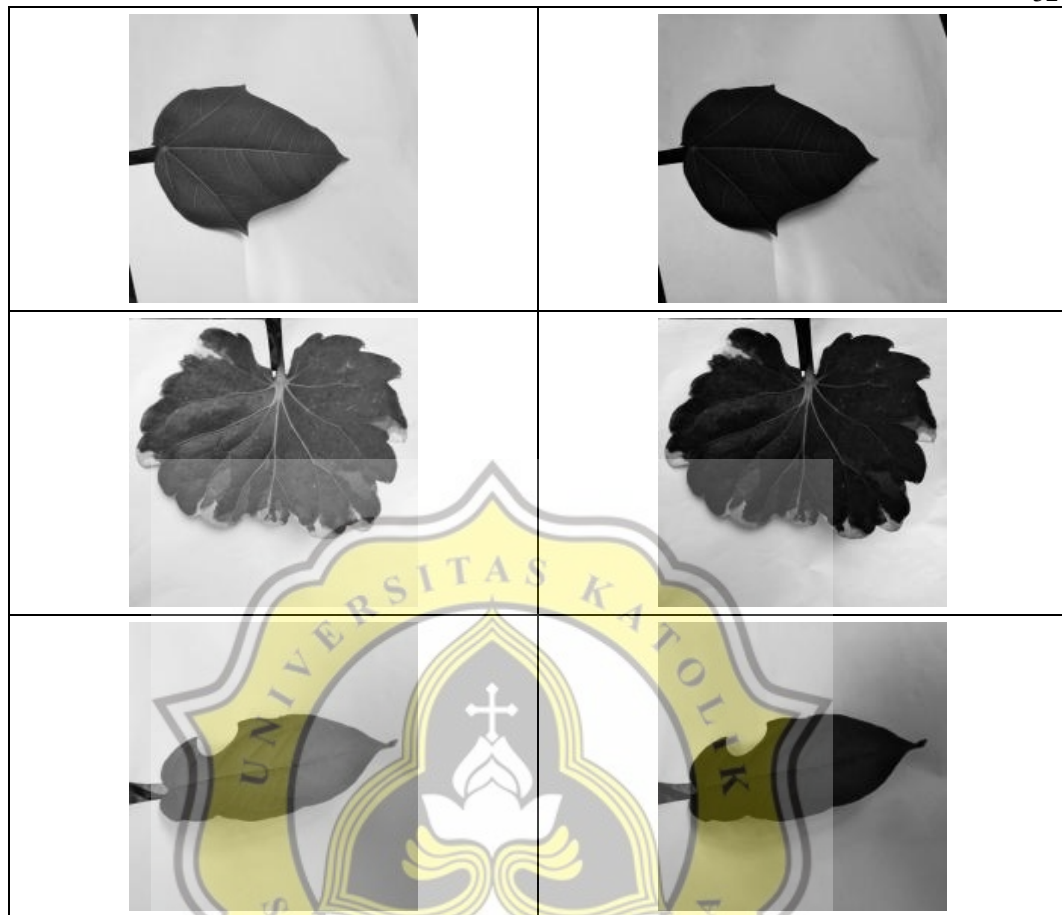
5.2.3. Morphologi (Closing)

In morphology algorithm, the first step that is done is the same as Thresholding is to convert digital imagery into grayscale imagery to then be reprinted with grayscale image output.

When finished with grayscale output, the image will be input back into the morphological process (Closing). Morphological Closing process is a combination of morphological dilasi followed by Morphological Erosion, meaning that the image undergoes pixel addition around the image called Morphological Dilasi and subsequently undergoes pixel reduction or so-called Morphological Erosion based on the arrangement of structur elements.



Grayscale Image	Morphological Image (Closing)
	
	

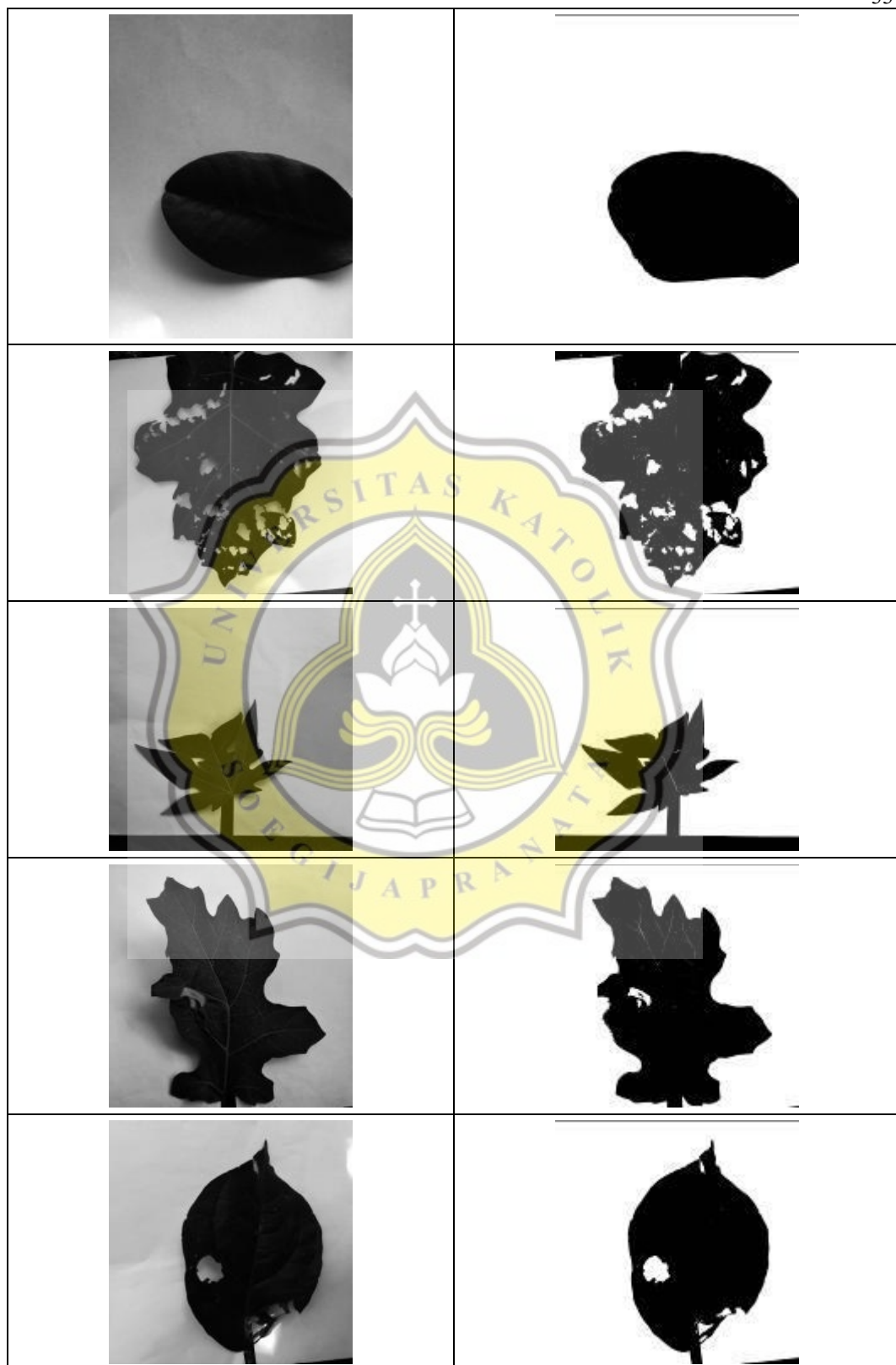


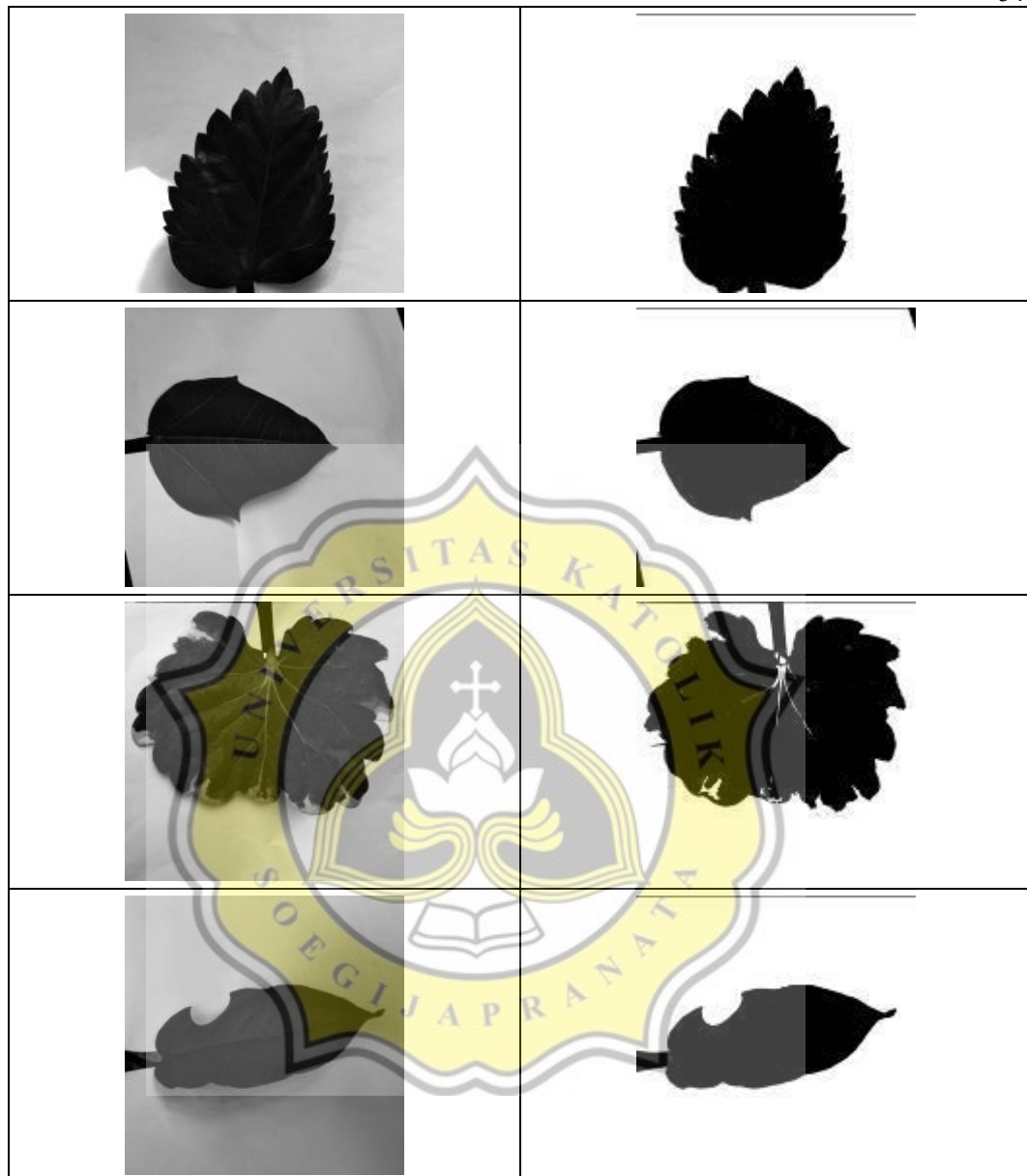


5.2.4 Grayscale Image to Morphological Image

After that the Morphological image will be saved with the new folder. Then the Morphological image will be changed to binary imagery or black and white imagery, then it will be saved in a different folder.

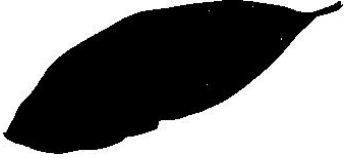
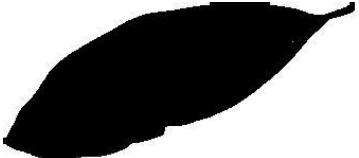

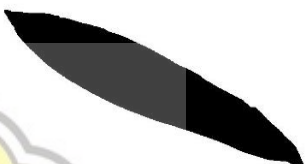




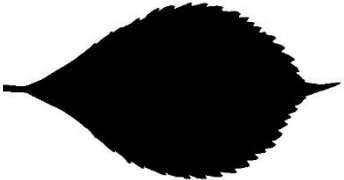
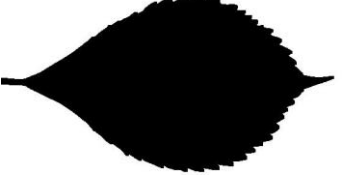
Morphological Image (Closing)	Biner Image
	





5.2.5. Morphological Image to Binary Image

After performing with colored pictures, followed by the binary. The one conducted together with colored pictures, it is just a direct shot into the algorithm.

Binary Images	Morphological Closing Images
	
	
	
	
	

5.2.6. Binary Image to Morphological Image

5.2.4. PSNR dan MSE

The way PSNR and MSE work is to insert the initial image to be further compared with thresholding imagery and also Morphological (Closing) imagery. In this case, the way to calculate MSE is to reduce between the pixels of the initial image with thresholding imagery and morphology imagery to be further raised to 2 and then divided by the image area. Then in calculating PSNR using log 10 with the highest pixel value and then divided by MSE value. MSE value is considered good if approaching with a value of 0 while for PSNR is said to be good if approaching the value of 30 db., here I use 40 times the image to be compared.

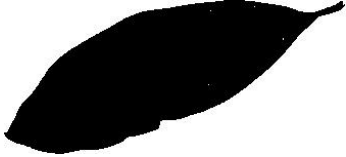

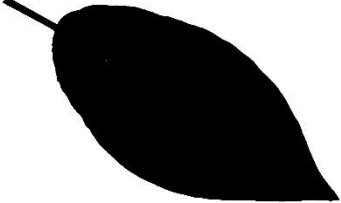
Original Image	Image Comparison
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 4020.465102458687 PSNR = 12.34 dB</pre>
	Thresholding
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 4002.6347667134182 PSNR = 12.36 dB</pre>
	Morphological (Closing)
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3667.4440720169637 PSNR = 12.88 dB</pre>
	Thresholding
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3661.0331885988003 PSNR = 12.89 dB</pre>
	Morphological (Closing)



	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3175.632865999952 PSNR = 13.7 dB</pre>
	Thresholding
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3139.5554683146324 PSNR = 13.77 dB</pre>
	Morphological (Closing)
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3225.8474147606503 PSNR = 14.17 dB</pre>
	Thresholding
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3216.521113056868 PSNR = 14.21 dB</pre>
	Morphological (Closing)
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3990.89661986914 PSNR = 12.74 dB</pre>
	Thresholding
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3950.954849054784 PSNR = 12.79 dB</pre>
	Morphological (Closing)
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 2546.8349131866303 PSNR = 14.57 dB</pre>
	Thresholding
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 2533.604702447613 PSNR = 14.6 dB</pre>
	Morphological (Closing)

	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 2723.65236002046 PSNR = 14.29 dB</pre> <p>Thresholding</p> <pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 2711.718334279978 PSNR = 14.32 dB</pre> <p>Morphological (Closing)</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3207.4460536013144 PSNR = 14 dB</pre> <p>Thresholding</p> <pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3195.319402945003 PSNR = 14.03 dB</pre> <p>Morphological (Closing)</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 4363.557686455565 PSNR = 11.92 dB</pre> <p>Thresholding</p> <pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 4320.365844307939 PSNR = 11.96 dB</pre> <p>Morphological (Closing)</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 4351.115054590602 PSNR = 12.4 dB</pre> <p>Thresholding</p> <pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 4343.154337034849 PSNR = 12.42 dB</pre> <p>Morphological (Closing)</p>

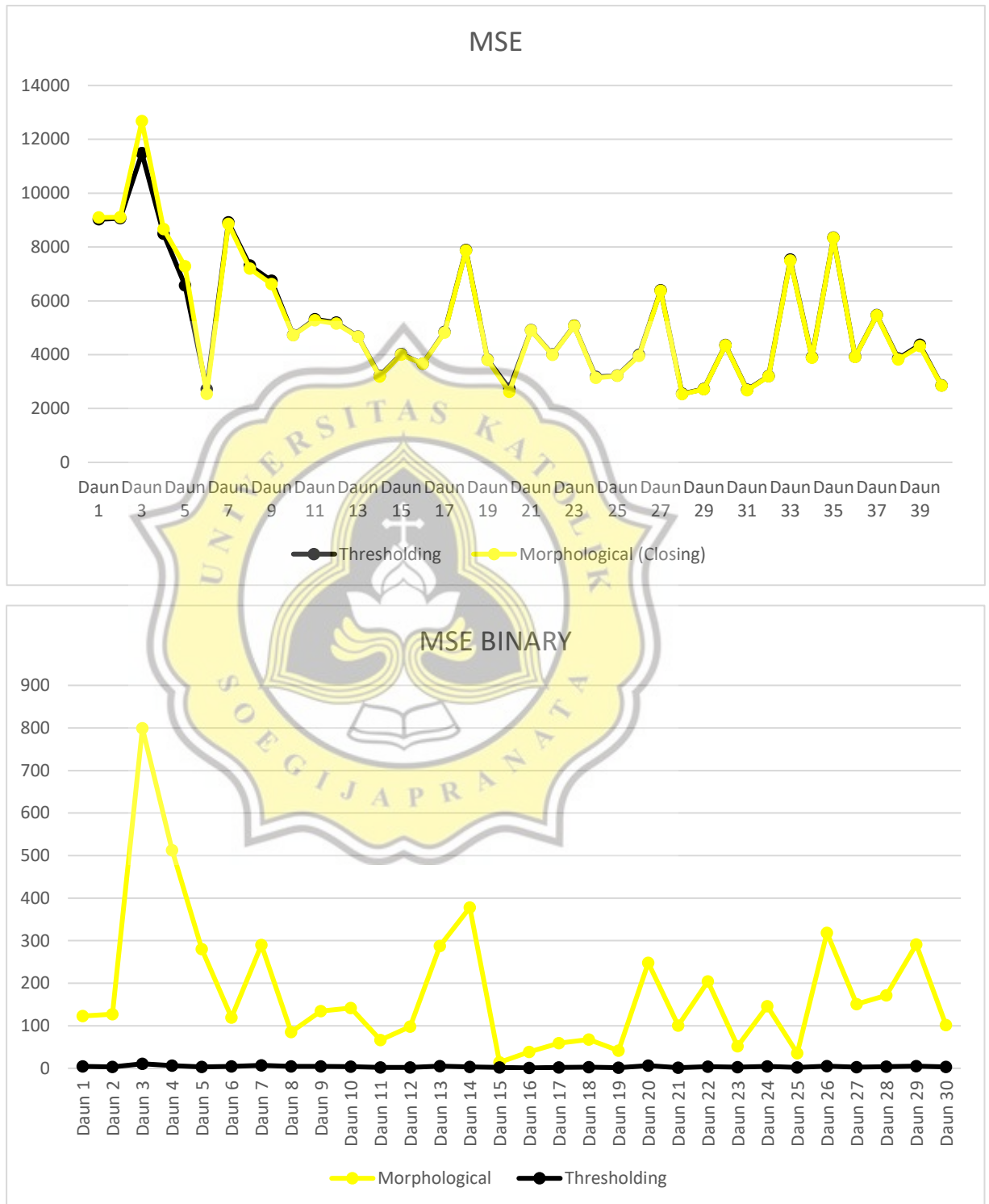
5.2.7. MSE and PSNR color images

The following is the PSNR and MSE in figure binary. A was aimed to examine the pictures binary equal to colored pictures.

Binary Images	PSNR and MSE
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 5.378368159204 PSNR = 40.82 dB</pre> <p>Thresholding</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 287.7604776119403 PSNR = 23.54 dB</pre> <p>Morphological Closing</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 2.1486779448622 PSNR = 44.81 dB</pre> <p>Thresholding</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 13.9989661654135 PSNR = 36.67 dB</pre> <p>Morphological Closing</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 0.9893890471668 PSNR = 48.18 dB</pre> <p>Thresholding</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 38.6382288698955 PSNR = 32.26 dB</pre> <p>Morphological Closing</p>

	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 5.3763902439024 PSNR = 40.83 dB</pre> <p>Thresholding</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 291.0037630662021 PSNR = 23.49 dB</pre> <p>Morphological Closing</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 3.1593781842884 PSNR = 43.13 dB</pre> <p>Thresholding</p>
	<pre>yohan@Adi:~/Pictures/Project/Operation\$ java MSE_PSNR MSE = 101.4890763743995 PSNR = 28.07 dB</pre> <p>Morphological Closing</p>
<p>5.2.8. MSE and PSNR Binary images</p>	

5.3. Experiment Results



5.3.1. MSE

