CHAPTER V

IMPLEMENTATION AND TESTING

5.1. Implementation

5.1.1. Implementation Program

The display of this program is shown below:

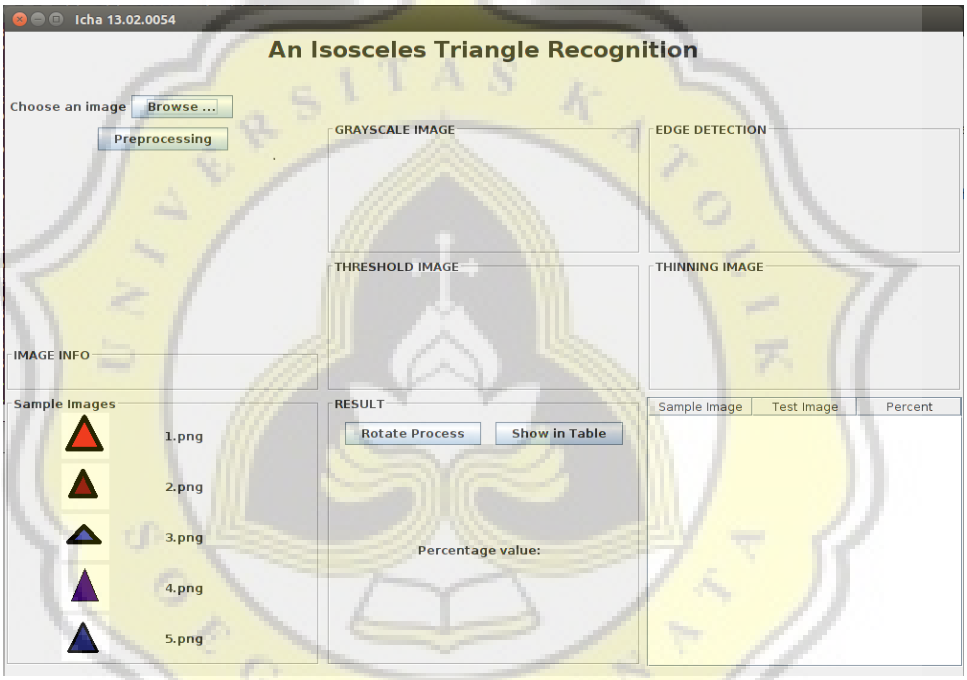

Figure 4 : Main Display

This program display allow user to input the triangle image that will be tested using Browse button. Then the processing button is to process the triangle image to grayscale, filtering, threshold, edge detection and thinning before the image will be rotate and matching. After that, if the rotate process is pressed, the program will run the rotate process and matching the triangle image that will be tested based on the reference triangle images.

5.1.1.1. Grayscale

The first preprocessing step is grayscale. In this step, the program change the triangle image that will be tested become a grayscale image. The grayscale value is gained from the average of the total value of the red, green and blue layer in each pixel coordinates.

```
for(int y=0;y<height;y++)
{
        for(int x=0;x<width;x++)
        {
                Color c = new Color(image.getRGB(x,y));
                avg = (c.getRed()+c.getGreen()+c.getBlue())/3;

                //set grayscale image
                Color gray = new Color(avg,avg,avg);
                image.setRGB(x,y,gray.getRGB());
        }
}
```

Figure 5 : Grayscale Code

5.1.1.2. Filtering

The second step is filtering the grayscale image known as grayscale process output. The purpose of this step is to decrease the noise at the image so no noise detected as edge when the next step running. This program use mean filter by adding all of pixel value in matrix 5x5 and it will divided by 25, start at initial coordinate (0,0) then shift to the right until the end of pixel coordinate.

```
rata = (temp[0]+temp[1]+temp[2]+temp[3]+temp[4]
        +temp[5]+temp[6]+temp[7]+temp[8]+temp[9]
        +temp[10]+temp[11]+temp[12]+temp[13]+temp[14]
        +temp[15]+temp[16]+temp[17]+temp[18]+temp[19]
        +temp[20]+temp[21]+temp[22]+temp[23]+temp[24]) / 25;
curr[y][x]=rata;

Color filter = new Color(curr[y][x],curr[y][x],curr[y][x]);
image.setRGB(x,y,filter.getRGB());
```

Figure 6 : Filtering Code

5.1.1.3. Threshold

After filtering, the output of it will through the threshold process. In this step, the program change the image into a binary image. The program will detect all of the pixel value when the value is less than 128 so that pixel value will change into 0 (black), whereas when the pixel value is greater than or equal to 128 so that pixel value will change into 255 (white).

```
//treshold pixel
for(int j=0;j<height;j++)
{
        for(int i=0;i<width;i++)
        {
                if(pixelku[j][i]<128)
                {
                        pixelbaru[j][i] = 0;
                }
                if(pixelku[j][i]>=128)
                {
                        pixelbaru[j][i] = 255;
                }
                Color threshold = new Color(pixelbaru[j][i],pixelbaru[j][i],pixelbaru[j][i]);
                image.setRGB(i,j,threshold.getRGB());
        }
}
```

Figure 7 : Threshold Code

5.1.1.4. Edge Detection

The next step is edge detection that is to detect the edge of object in the image. In this program, to detect the edge of object using Sobel method. All of the pixel value in matrix 3x3 will be calculate by Sobel kernel that is Gx and Gy and will produce the absolute value.

```
for(int y=1;y<height-1;y++)
{
        for(int x=1;x<width-1;x++)
        {
                //gy
                tempy[0] = filter[y-1][x-1] * sobely[0][0];
                tempy[1] = filter[y-1][x] * sobely[0][1];
                tempy[2] = filter[y-1][x+1] * sobely[0][2];
                tempy[3] = filter[y][x-1] * sobely[1][0];
                tempy[4] = filter[y][x] * sobely[1][1];
                tempy[5] = filter[y][x+1] * sobely[1][2];
                tempy[6] = filter[y+1][x-1] * sobely[2][0];
                tempy[7] = filter[y+1][x] * sobely[2][1];
                tempy[8] = filter[y+1][x+1] * sobely[2][2];

                jumlahy[y][x] = Math.abs(tempy[0]+tempy[1]+tempy[2]+tempy[3]+tempy[4]+tempy[5]+tempy[6]+tempy[7]
+tempy[8]);

                //gx
                tempx[0] = filter[y-1][x-1] * sobelx[0][0];
                tempx[1] = filter[y-1][x] * sobelx[0][1];
                tempx[2] = filter[y-1][x+1] * sobelx[0][2];
                tempx[3] = filter[y][x-1] * sobelx[1][0];
                tempx[4] = filter[y][x] * sobelx[1][1];
                tempx[5] = filter[y][x+1] * sobelx[1][2];
                tempx[6] = filter[y+1][x-1] * sobelx[2][0];
                tempx[7] = filter[y+1][x] * sobelx[2][1];
                tempx[8] = filter[y+1][x+1] * sobelx[2][2];

                jumlahx[y][x] = Math.abs(tempx[0]+tempx[1]+tempx[2]+tempx[3]+tempx[4]+tempx[5]+tempx[6]+tempx[7]
+tempx[8]);

                total[y][x] = jumlahx[y][x] + jumlahy[y][x];
                if(total[y][x] > 255)
                {
                        total[y][x] = 255;
                }
```

Figure 8 : Edge Detection Code

5.1.1.5. Thinning

The last step of preprocessing is thinning the pixel image until it only has one pixel thickness. All of pixel value of 255 will through two sub iteration. The pixel coordinate of 255 will marked as to be deleted if it satisfy all of this requirement:

P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

Figure 9 : Illustration Pixel Position

1. $5 <= NP <= 6$, where NP is the amount of neighbor of P_1 that is not 0
2. SP = 1, where the SP is transition from 0 to 255 in order P_2, P_3, P_4, P_5, etc.
3. $P_2 \times P_4 \times P_5 = 0$
4. $P_4 \times P_5 \times P_8 = 0$

```
if(pixeltemp[j][i]==255)
{
        //syarat 1 (5<=NP<=6)
        //NP = jumlah piksel tetangga 8 arah yang bukan 0
        if(pixeltemp[j-1][i]==255)
        {
                NP++;
        }
        if(pixeltemp[j-1][i+1]==255)
        {
                NP++;
        }
        if(pixeltemp[j][i+1]==255)
        {
                NP++;
        }
        if(pixeltemp[j+1][i+1]==255)
        {
                NP++;
        }
```

Figure 10 : One of Thinning Requirement

5.1.1.6. Translation and Rotation

After all of preprocessing is completely done, all the pixel coordinates of triangle shape in the image will be processed in translation and rotation process. Based on the combine of translation and rotation formula that has been discussed at the chapter II, in this program the triangle will be rotated as far as a certain angle.

```
        //translasi masing-masing koordinat dengan sumbu y0 dan x0
        for(int a=0;a<hitung255;a++)
        {
                geser[a][0] = awal[a][0] - x_min;
                geser[a][1] = awal[a][1] - y_max;
        }
/*
        rumus :
        x' = x cos sudut + y sin sudut
        y' = y cos sudut - x sin sudut
*/

        radian = sudut * Math.PI/180;
        cos_sudut = Math.cos(radian);
        sin_sudut = Math.sin(radian);

        for(int b=0;b<hitung255;b++)
        {
                //putar ke kiri (sudut terhadap sumbu x positif)
                baru[b][0] = Math.floor(geser[b][0] * cos_sudut + geser[b][1] * sin_sudut);
                baru[b][1] = Math.floor(geser[b][1] * cos_sudut - geser[b][0] * sin_sudut);
        }
```

Figure 11 : Translation and Rotation Code

5.1.1.7. Matching Shape

The next step is matching the shape of triangle that will be tested with the reference images. In this program, the similarity of two triangle are determined by length of base and the height of its.

```
x_max1 = awal1[hitung255-1][0];
y_min1 = awal1[0][1];
y_max1 = awal1[hitung255-1][1];

tinggi1 = y_max1 - y_min1;
System.out.println("tinggi1 : "+y_max1+" - "+y_min1+" = "+tinggi1);

//cari x_min1
k = 0;
while(k!=hitung255-1)
{
        if(awal1[k][1]==y_max1 && awal1[k][0]<x_max1)
        {
                x_min1 = awal1[k][0];
                break;
        }
        k++;
}

pjgalas1 = x_max1 - x_min1;
System.out.println("panjang alas 1= "+x_max1+" - "+x_min1+" = "+pjgalas1);
```

Figure 12 : Calculation Length of Base and Height of Triangle Code

After the length of base and the height are calculated, so calculate each percentage value. Then the final result is average of two percentage value above.

```
persen = ((double)pjgalas2/(double)pjgalas1)*100;
persen1 = ((double)tinggi2/(double)tinggi1)*100;

if(persen>100)
{
        persen = ((double)pjgalas1/(double)pjgalas2)*100;
}
if(persen1>100)
{
        persen1 = ((double)tinggi1/(double)tinggi2)*100;
}

System.out.printf("%.2f",persen);
System.out.println();
System.out.printf("%.2f",persen1);
System.out.println();

if(persen>45 && persen1>45)
{
        status = 1;
}
else
{
        status = 0;
}

persentotal = (persen+persen1)/2;
```

Figure 13 : Calculation Percent of Similarity Code

5.1.1.8. Storage the percentage value in the linked list

After matching shape and get the percentage value, then the percentage value will be stored in data structured that is linked list.

```
public void addLast(double p)
{
        Link curr;
        Link nw = new Link(p);

        if(isEmpty())
        {
                head=nw;
        }
        else
        {
                curr=head;
                while (curr.next!=null)
                {
                        curr=curr.next;
                }

                curr.next=nw;
        }

}
```

Figure 14 : add Item to Linked List

The last step in this program is searching for the greatest value of the percentage value of similarity between triangle that will be tested and the reference images.

```
public double[] cariMax(double [] a)
{
        index=0;
        max = 0;
        for(int i=0;i<5;i++)
        {
                if(a[i]>max)
                {
                        max = a[i];
                }
        }
        //index
        for(int i=0;i<5;i++)
        {
                if(a[i]==max)
                {
                        index = i+1;
                        break;
                }
        }
        return new double[] {max,index};
}
```

Figure 15 : Search the Greatest Percent Value

5.1.2. Program using Image

First, user input the triangle image that will be tested by press the ‘browse’ button.

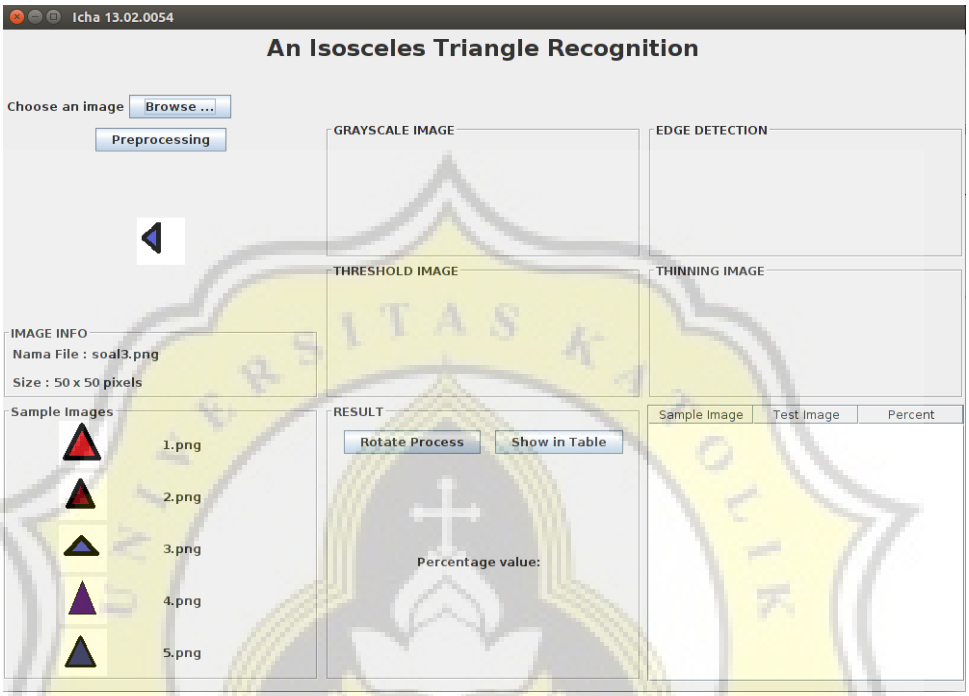

Figure 16 : Input Triangle Image

After that the image will be processed into grayscale, filtering, threshold, edge detection and thinning by press the ‘preprocessing’ button and will show output of each process.

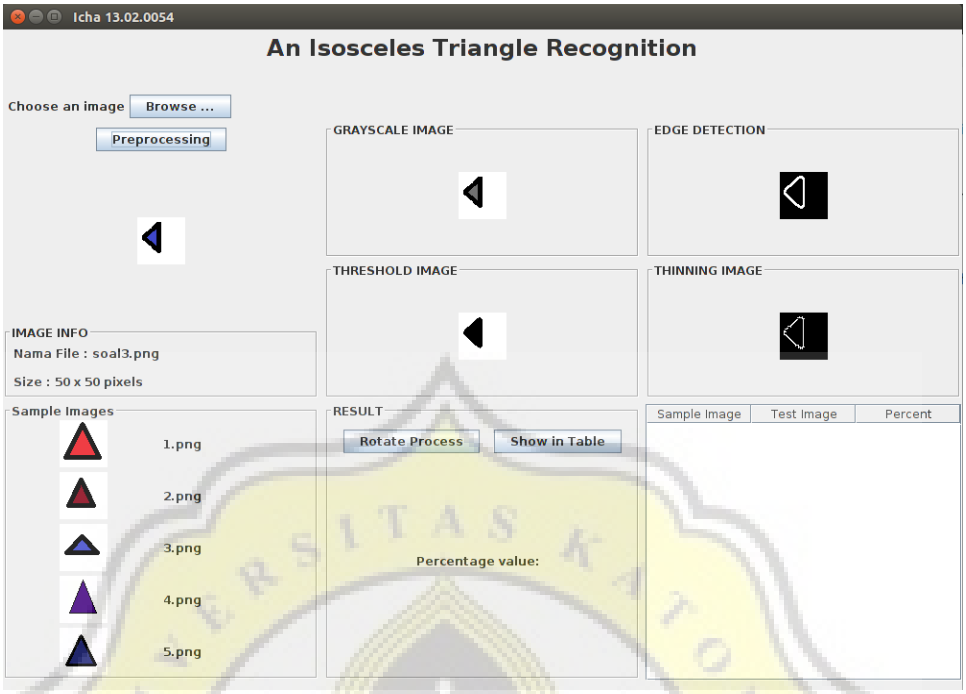

Figure 17 : Preprocessing Image

Then the image will be rotate and matching based on reference images. This process will give display the output in form percentage value in the middle of the bottom. User can see all of the percent result of triangle that will be tested and each reference images on the table at the right of bottom.

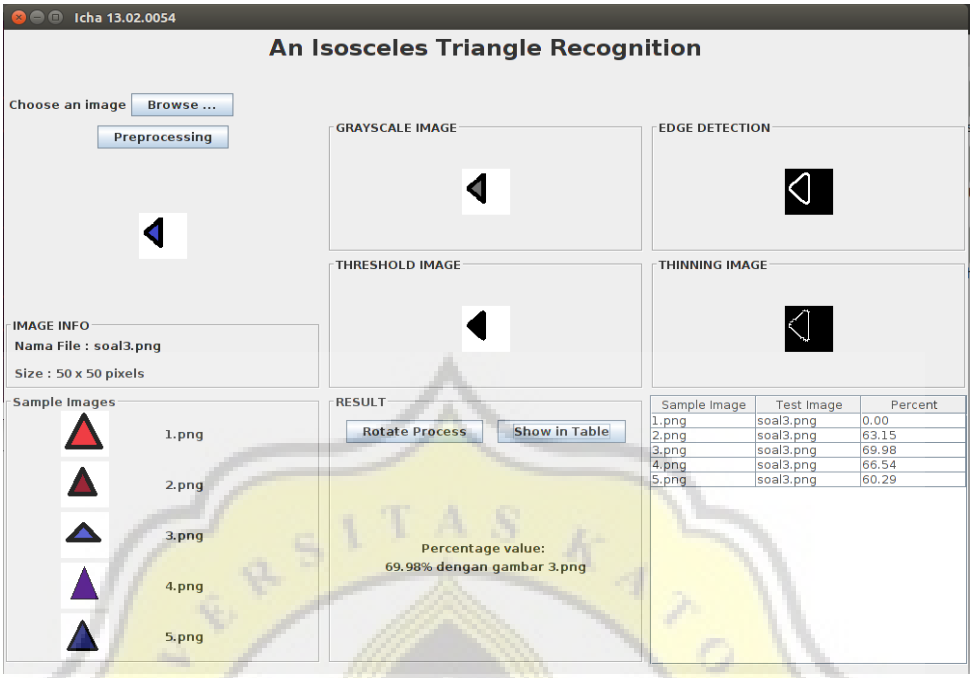

Figure 18 : The Result of the Testing

5.2. Testing Program

In this program there are 5 reference images as follows:

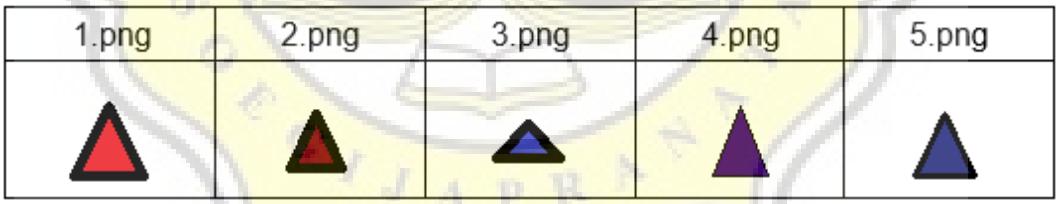

Figure 19 : Reference Triangle Images

5.2.1. Testing using red triangle that has been rotated as far as a certain angle

Table 1 : Testing using soal1.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal1.png		49.66%
2.png			62.18%
3.png			70.65%
4.png			66.30%
5.png			59.22%

According to the result of the testing with image soal1.png above, the highest percentage value of similarity between tested image and reference images obtained by the image 3.png that is 70.65%. So the soal1.png is most similar to image 3.png.

5.2.2. Testing using maroon triangle that has been rotated as far as a certain angle

Table 2 : Testing using soal2.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal2.png		0%
2.png			58.33%
3.png			58.33%
4.png			61.52%
5.png			55.69%

According to the result of the testing with image soal2.png above, the highest percentage value of similarity between tested image and reference images obtained by the image 4.png that is 61.52%. So the soal2.png is most similar to image 4.png.

5.2.3. Testing using blue triangle that has been rotated as far as a certain angle

Table 3 : Testing using soal3.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal3.png		0%
2.png			63.15%
3.png			69.98%
4.png			66.54%
5.png			60.29%

According to the result of the testing with image soal3.png above, the highest percentage value of similarity between tested image and reference images obtained by the image 3.png that is 69.98%. So the soal3.png is most similar to image 3.png.

5.2.4. Testing using purple triangle that has been rotated as far as a certain angle

Table 4 : Testing using soal4.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal4.png		0%
2.png			50.76%
3.png			50.76%
4.png			54.17%
5.png			48.33%

According to the result of the testing with image soal4.png above, the highest percentage value of similarity between tested image and reference images obtained by the image 4.png that is 54.17%. So the soal4.png is most similar to image 4.png.

5.2.5. Testing using dark blue triangle that has been rotated as far as a certain angle

Table 5 : Testing using soal5.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal5.png 		50.47%
2.png			63.69%
3.png			72.83%
4.png			67.77%
5.png			60.69%

According to the result of the testing with image soal5.png above, the highest percentage value of similarity between tested image and reference images obtained by the image 3.png that is 72.83%. So the soal5.png is most similar to image 3.png.

5.2.6. Testing using gray triangle that has been rotated as far as a certain angle

Table 6 : Testing using soal6.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal6.png 		0%
2.png			48.97%
3.png			48.97%
4.png			52.08%
5.png			52.08%

According to the result of the testing with image soal6.png above, the highest percentage value of similarity between tested image and reference images obtained by the image 4.png and 5.png that is 52.08%. So the soal6.png is similar to image 4.png and 5.png.

5.2.7. Testing using red triangle that has been rotated as far as a certain angle less than 90 degrees

Table 7 : Testing using soal7.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal7.png		0%
2.png			72.51%
3.png			63.18%
4.png			75.98%
5.png			69.31%

According to the result of the testing with image soal7.png above, the highest percentage value of similarity between tested image and reference images obtained by the image 4.png that is 75.98%. So the soal7.png is most similar to image 4.png.

5.2.8. Testing using blue triangle that has been rotated as far as a certain angle more than 90 degrees

Table 8 : Testing using soal8.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal8.png		0%
2.png			49.51%
3.png			49.51%
4.png			49.51%
5.png			49.51%

According to the result of the testing with image soal8.png above, there are no highest percentage value of similarity between tested image and reference images, because it has same percentage value with all of the reference images except image 1.png whose percentage value 0%. So the soal8.png similar to image 2.png, 3.png, 4.png and 5.png.

5.2.9. Testing using purple triangle that has been rotated as far as a certain angle less than 90 degrees

Table 9 : Testing using soal9.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal9.png		0%
2.png			65.91%
3.png			65.91%
4.png			68.87%
5.png			63.04%

According to the result of the testing with image soal9.png above, the highest percentage value of similarity between tested image and reference images obtained by the image 4.png that is 68.87%. So the soal9.png is most similar to image 4.png.

5.2.10. Testing using green triangle that has been rotated as far as a certain angle

Table 10 : Testing using soal10.png

Sample Image	Test Image	Rotated test image	Percentage value of similarity
1.png	soal10.png		76.79%
2.png			90.61%
3.png			83.77%
4.png			82.65%
5.png			92.65%

According to the result of the testing with image soal10.png above, the highest percentage value of similarity between tested image and reference images obtained by the image 5.png that is 92.65%. So the soal10.png is most similar to image 5.png.