CHAPTER 5
IMPLEMENTATION AND TESTING

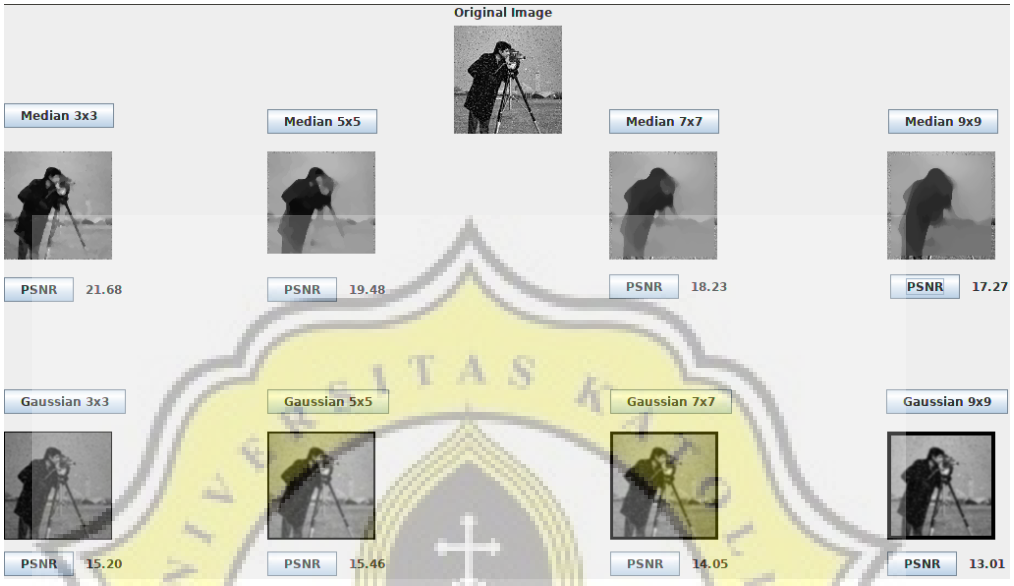

Illustration 5.1: Main Program

5.1 Implementation

This project use Java programming languange and NetBeans IDE for the user interfaces. This project use 2 methods for reducing a noise in the images , that is Gaussian Filtering and Median Filtering with different size of kernels (3x3 , 5x5 ,7x7 and 9x9). Gaussian Filtering and Median Filtering chosen beacuse based on the journals stated that both of them are the best method for reducing noise in image. Also , this project use PSNR (Peak Signal to Noise Ratio) as a parameter to see the result of filtered images. The object of this project is 20 images with noise (10 salt & pepper noise , 5 gaussian noise , and 5 speckle noise) which obtained from the web. After getting the images , then the images filtered using Median Filtering and Gaussian Filtering. Each images filtered using both of methods with different size of kernels (3x3 , 5x5 , 7x7 , and 9x9). After that count the filtered image with PSNR , the PSNR works comparising the original image with the filtered image using it formula

Below is the code of Median Filtering :

```
23      public static void main(String[] args) throws Throwable {
24          // TODO code application logic here
25          File f = new File("/home/ega/Ega/Skripsi/snoise5.jpeg");
26          Color[] pixel = new Color[9];
27          int[] R = new int[9];
28          int[] B = new int[9];
29          int[] G = new int[9];
30          File output = new File("/home/ega/Ega/Skripsi/median3x3snoise5.png");
31          BufferedImage img = ImageIO.read(f);
32          for (int i = 1; i < img.getWidth() - 1; i++) {
33              for (int j = 1; j < img.getHeight() - 1; j++) {
34                  pixel[0] = new Color(img.getRGB(i - 1, j - 1));
35                  pixel[1] = new Color(img.getRGB(i - 1, j));
36                  pixel[2] = new Color(img.getRGB(i - 1, j + 1));
37                  pixel[3] = new Color(img.getRGB(i, j - 1));
38                  pixel[4] = new Color(img.getRGB(i, j));
39                  pixel[5] = new Color(img.getRGB(i, j + 1));
40                  pixel[6] = new Color(img.getRGB(i + 1, j - 1));
41                  pixel[7] = new Color(img.getRGB(i + 1, j));
42                  pixel[8] = new Color(img.getRGB(i + 1, j + 1));
43                  for (int k = 0; k < 9; k++) {
44                      R[k] = pixel[k].getRed();
45                      B[k] = pixel[k].getBlue();
46                      G[k] = pixel[k].getGreen();
47                  }
48                  Arrays.sort(R);
49                  Arrays.sort(G);
50                  Arrays.sort(B);
51
52                  img.setRGB(i, j, new Color(R[4], B[4], G[4]).getRGB());
53              }
54          }
55          ImageIO.write(img, "png", output);
56      }
```

Illustration 5.2: Code 3x3 Median

The code above show the process of reducing a noise in original image with Median Filtering method , Median filtering works by sorting the pixel value of original image from the lowest pixel value to the highest pixel value , then finding the median value of the pixel after sorting process. After get the median value then , the result value change the original pixel value

Below is the code of Gaussian Filtering :

```
24          private Gaussian3x3() {
25          }
26
27          private static Gaussian3x3 gauss;
28
29          public static Gaussian3x3 getInstance() {
30              if (gauss == null) {
31                  gauss = new Gaussian3x3();
32              }
33
34              return gauss;
35          }
```

```
37      public double[][] generateKernel(int radius, double variance) {
38          double[][] kernel = new double[radius][radius];
39          for (int i = 0; i < kernel.length; i++) {
40              for (int j = 0; j < kernel[i].length; j++) {
41                  kernel[i][j] = gaussianModel(i - radius / 2, j - radius / 2, variance);
42
43                  System.out.print(kernel[i][j] + " ");
44              }
45              System.out.println();
46          }
47
48          return kernel;
49      }
50
51      public BufferedImage createGaussianImage(BufferedImage source_image, double kernel[][], int radius) {
52          BufferedImage value = new BufferedImage(source_image.getWidth(), source_image.getHeight(),
    BufferedImage.TYPE_INT_RGB);
53          for (int x = 1; x < source_image.getWidth() - 1; x++) {
54              for (int y = 1; y < source_image.getHeight() - 1; y++) {
55
56                  double[][] redProcess = new double[radius][radius];
57                  double[][] greenProcess = new double[radius][radius];
58                  double[][] blueProcess = new double[radius][radius];
59
60                  for (int kernelX = 0; kernelX < kernel.length; kernelX++) {
61                      for (int kernelY = 0; kernelY < kernel[kernelX].length; kernelY++) {
62
63                          try {
64                              int valueX = x + kernelX - (kernel.length / 2);
65                              int valueY = y + kernelY - (kernel.length / 2);
66
67                          double currentKernel = kernel[kernelX][kernelY];
68
69                          Color color = new Color(source_image.getRGB(valueX, valueY));
70
71                          redProcess[kernelX][kernelY] = currentKernel * color.getRed();
72                          greenProcess[kernelX][kernelY] = currentKernel * color.getGreen();
73                          blueProcess[kernelX][kernelY] = currentKernel * color.getBlue();
74                      } catch (Exception e) {
75                          System.out.println("Out of Bonds");
76                      }
77                  }
78              }
79
80              value.setRGB(x, y, new Color(getKernelValue(redProcess), getKernelValue(greenProcess), getKernelValue
    (blueProcess)).getRGB());
81              }
82          }
83          return value;
84      }
85
86      private int getKernelValue(double[][] kernelColor) {
87          double sum = 0;
88
89          for (int i = 0; i < kernelColor.length; i++) {
90              for (int j = 0; j < kernelColor[i].length; j++) {
91                  sum += kernelColor[i][j];
92              }
93          }
94          return (int) sum;
95      }
```

```
97          public double gaussianModel(double x, double y, double variance) {
98              return (1 / (2 * Math.PI * Math.pow(variance, 2)) * Math.exp(-(Math.pow(x, 2) + Math.pow(y, 2)) / (2 * Math.pow
    (variance, 2))));
99          }
100
101         public static void main(String[] args) {
102             // TODO code application logic here
103
104             double[][] kernel = Gaussian3x3.getInstance().generateKernel(3, 1);
105
106             BufferedImage value = null;
107             try {
108                 BufferedImage source_image = ImageIO.read(new File("/home/ega/Ega/Skripsi/snoise5.jpeg"));
109                 value = Gaussian3x3.getInstance().createGaussianImage(source_image, kernel, 3);
110             } catch (IOException e) {
111                 e.printStackTrace();
112             }
113
114             try {
115                 ImageIO.write(value, "PNG", new File("/home/ega/Ega/Skripsi/gaus3x3snoise5.png"));
116             } catch (IOException e) {
117                 e.printStackTrace();
118             }
119             System.out.println("DONE");
120         }
121
```

Illustration 5.3: 3x3 Gaussian

The code above show the process of reducing a noise in original image with Gaussian Filtering method , Gaussian Filtering works by finding the kernel Gaussian first , using it's formula , after getting the kernel value then do the convolution process

Below is the code of PSNR :

```
23      public static void main(String[] args) throws Throwable {
24          BufferedImage img1 = null;
25          BufferedImage img2 = null;
26
27          try {
28              File f1 = new File("/home/ega/Ega/Skripsi/snoise5.jpeg");
29              File f2 = new File("/home/ega/Ega/Skripsi/gaus5x5snoise5.png");
30
31              img1 = ImageIO.read(f1);
32              img2 = ImageIO.read(f2);
33          } catch (IOException e) {
34              System.out.println(e);
35          }
36          for (int i = 0; i < img1.getWidth(); i++) {
37              for (int j = 0; j < img1.getHeight(); j++) {
38                  Color c = new Color(img1.getRGB(i,j));
39                  int red = c.getRed();
40                  int green = c.getGreen();
41                  int blue = c.getBlue();
42              }
43          }
44          for (int i = 0; i < img2.getWidth(); i++) {
45              for (int j = 0; j < img2.getHeight(); j++) {
46                  Color c = new Color(img2.getRGB(i,j));
47                  int red = c.getRed();
48                  int green = c.getGreen();
49                  int blue = c.getBlue();
50              }
51          }
52      assert(img1.getType() == img2.getType() && img1.getWidth() == img2.getWidth() && img1.getHeight() == img2.getHeight
    ());
53      double mse = 0;
54      int width = img1.getWidth();
55      int height = img1.getHeight();
56      Raster r1 = img1.getRaster();
57      Raster r2 = img2.getRaster();
58      for (int i = 0; i < width; i++)
59          for (int j = 0; j < height; j++)
60              mse += Math.pow(r1.getSample(i, j, 0) - r2.getSample(i, j, 0), 2);
61
62      mse /= (double) (width * height);
63      double psnr = 10.0 * Math.log10(Math.pow(255, 2) / mse);
64      System.out.println("PSNR = " + psnr);
65  }
```

Illustration 5.4: PSNR

The code above show the process of counting PSNR value of original image and processed image , PSNR works by comparising the original image and processed image with it’s formula , but original image and processed image must have the same type , height , and weight. The higher PSNR value the better quality, but the higher PSNR value can’t determine the best image according to vision

5.2 Testing

The result of this research sort by size of kernels (3x3 , 5x5 , 7x7 , 9x9) and Filtering method (Median Filtering and Gaussian Filtering)

By kernel size :

Table 5.1: Kernel 3x3

Image	PSNR (dB) (Median)	PSNR (dB) (Gaussian)
noise.png (8 bit)	21.68	15.20
noise2.png (8 bit)	17.50	12.55
noise3.png (8 bit)	13.35	12.47
noise4.png (8 bit)	22.04	15.50
noise5.png (8 bit)	20.66	16.13
noise6.png (8 bit)	18.48	14.02
noise7.png (8 bit)	21.60	18.51
noise8.png (24 bit)	21.66	16.02
noise9.png (8 bit)	23.47	14.90
noise10.png (8 bit)	21.21	17.09
gnoise1.jpeg (24 bit)	21.93	12.26
gnoise2.jpg (24 bit)	20.98	16.44
gnoise3.png (24 bit)	16.98	14.28
gnoise4.jpg (24 bit)	22.33	12.19
gnoise5.jpeg (24 bit)	17.63	14.57
snoise1.jpeg (24 bit)	19.09	13.58
snoise2.jpeg (24 bit)	18.10	11.42
snoise3.jpeg (24 bit)	16.83	14.76
snoise4.jpg (24 bit)	28.31	18.04
snoise5.jpeg (24 bit)	15.98	13.12

Table 5.2: Average Kernel 3x3

Method	Average PSNR Value
Median Filtering	19.99 dB
Gaussian Filtering	14.65 dB

According to kernel size 3x3 Median Filtering has a minimum value of PSNR 13.35 dB and maximum value of PSNR 28.31 dB while Gaussian Filtering has a minimum value of PSNR 12.19 and the maximum value of PSNR 18.51 dB. The average for kernel size 3x3 Median Filtering has 19.99 dB and Gaussian Filtering has 14.65 dB

Table 5.3: Kernel 5x5

Image	PSNR (dB) (Median)	PSNR (dB) (Gaussian)
noise.png (8 bit)	19.48	15.46
noise2.png (8 bit)	17.27	12.92
noise3.png (8 bit)	13.03	12.58
noise4.png (8 bit)	20.18	16.02
noise5.png (8 bit)	18.72	16.53
noise6.png (8 bit)	16.01	14.53
noise7.png (8 bit)	20.02	19.18
noise8.png (24 bit)	20.28	16.14
noise9.png (8 bit)	21.71	15.30
noise10.png (8 bit)	18.94	18.59
gnoise1.jpeg (24 bit)	20.71	11.15
gnoise2.jpg (24 bit)	18.54	16.52
gnoise3.png (24 bit)	15.08	14.58
gnoise4.jpg (24 bit)	20.56	12.69
gnoise5.jpeg (24 bit)	16.91	15.09
snoise1.jpeg (24 bit)	17.22	13.91
snoise2.jpeg (24 bit)	16.94	10.84
snoise3.jpeg (24 bit)	15.82	15.02
snoise4.jpg (24 bit)	26.61	22.43
snoise5.jpeg (24 bit)	14.39	13.78

Table 5.4: Average Kernel 5x5

Method	Average PSNR Value
Median Filtering	18.42 dB
Gaussian Filtering	15.16 dB

According to kernel size 5x5 Median Filtering has a minimum value of PSNR 13.03 dB and maximum value of PSNR 26.61 dB while Gaussian Filtering has a minimum value of PSNR 10.84 and the maximum value of PSNR 22.43 dB. The average for kernel size 5x5 Median Filtering has 18.42 dB and Gaussian Filtering has 15.16 dB

Table 5.5: Kernel 7x7

Image	PSNR (dB) (Median)	PSNR (dB) (Gaussian)
noise.png (8 bit)	18.23	14.05
noise2.png (8 bit)	17.27	12.45
noise3.png (8 bit)	13.03	12.58
noise4.png (8 bit)	18.79	14.68
noise5.png (8 bit)	17.91	15.34
noise6.png (8 bit)	15.11	13.45
noise7.png (8 bit)	19.42	18.23
noise8.png (24 bit)	19.65	14.83
noise9.png (8 bit)	20.17	13.77
noise10.png (8 bit)	17.48	17.63
gnoise1.jpeg (24 bit)	19.98	9.56
gnoise2.jpg (24 bit)	17.79	15.30
gnoise3.png (24 bit)	13.97	13.72
gnoise4.jpg (24 bit)	19.87	12.40
gnoise5.jpeg (24 bit)	16.87	14.09
snoise1.jpeg (24 bit)	16.38	12.87
snoise2.jpeg (24 bit)	16.73	9.78
snoise3.jpeg (24 bit)	15.55	14.20
snoise4.jpg (24 bit)	25.92	21.01
snoise5.jpeg (24 bit)	13.68	12.90

Table 5.6: Average Kernel 7x7

Method	Average PSNR Value
Median Filtering	17.69 dB
Gaussian Filtering	14.14 dB

According to kernel size 7x7 Median Filtering has a minimum value of PSNR 13.03 dB and maximum value of PSNR 25.92 dB while Gaussian Filtering has a minimum value of PSNR 9.56 and the maximum value of PSNR 21.01 dB. The average for kernel size 7x7 Median Filtering has 17.69 dB and Gaussian Filtering has 14.14 dB

Table 5.7: Kernel 9x9

Image	PSNR (dB) (Median)	PSNR (dB) (Gaussian)
noise.png (8 bit)	17.27	13.01
noise2.png (8 bit)	17.36	12.01
noise3.png (8 bit)	13.25	11.57
noise4.png (8 bit)	18.23	13.63
noise5.png (8 bit)	17.23	14.51
noise6.png (8 bit)	14.36	12.65
noise7.png (8 bit)	19.08	17.46
noise8.png (24 bit)	19.32	13.88
noise9.png (8 bit)	19.28	12.63
noise10.png (8 bit)	16.40	16.83
gnoise1.jpeg (24 bit)	19.25	8.64
gnoise2.jpg (24 bit)	17.51	14.36
gnoise3.png (24 bit)	13.21	13.02
gnoise4.jpg (24 bit)	19.24	12.07
gnoise5.jpeg (24 bit)	17.03	13.33
snoise1.jpeg (24 bit)	15.86	12.06
snoise2.jpeg (24 bit)	16.50	8.96
snoise3.jpeg (24 bit)	15.65	13.48
snoise4.jpg (24 bit)	25.51	19.88
snoise5.jpeg (24 bit)	12.81	12.15

Table 5.8: Average Kernel 9x9

Method	Average PSNR Value
Median Filtering	17.22 dB
Gaussian Filtering	13.30 dB

According to kernel size 9x9 Median Filtering has a minimum value of PSNR 12.81 dB and maximum value of PSNR 25.51 dB while Gaussian Filtering has a minimum value of PSNR 8.64 and the maximum value of PSNR 19.88 dB. The average for kernel size 9x9 Median Filtering has 17.22 dB and Gaussian Filtering has 13.30 dB

By filtering method :

Table 5.9: Median FIltering

Image	PSNR (dB) 3x3	PSNR (dB) 5x5	PSNR (dB) 7x7	PSNR (dB) 9x9
noise.png (8 bit)	21.68	19.48	18.23	17.27
noise2.png (8 bit)	17.50	17.27	17.28	17.36
noise3.png (8 bit)	13.35	13.03	13.09	13.25
noise4.png (8 bit)	22.04	20.18	18.79	18.23
noise5.png (8 bit)	20.66	18.72	17.91	17.23
noise6.png (8 bit)	18.48	16.01	15.11	14.36
noise7.png (8 bit)	21.60	20.02	19.42	19.08
noise8.png (8 bit)	21.66	20.28	19.65	19.32
noise9.png (8 bit)	23.47	21.21	20.17	19.28
noise10.png (8 bit)	21.21	18.94	17.48	16.40
gnoise1.jpeg(24bit)	21.93	20.71	19.98	19.25
gnoise2.jpg (24bit)	20.98	18.54	17.79	17.51
gnoise3.png(24bit)	16.98	15.08	13.97	13.21
gnoise4.jpg(24 bit)	22.33	20.56	19.87	19.24
gnoise5.jpeg(24bit)	17.63	16.91	16.87	17.03
snoise1.jpeg (8 bit)	19.09	17.22	16.38	15.86
snoise2.jpeg (8 bit)	18.10	16.94	16.73	16.50
snoise3.jpeg (8 bit)	16.83	15.82	15.55	15.65
snoise4.jpg (24bit)	28.31	26.61	25.92	25.51
snoise5.jpeg(24bit)	15.98	14.39	13.68	12.81

According to data testing from Median Filtering with size of kernels 3x3 , 5x5 , 7x7 , and 9x9 , it has a maximum value of PSNR 28.31 dB and a minimum value of PSNR 12.81 dB , the best kernel size is 3x3 because it has the higher PSNR than others size , and the lowest is 9x9 kernel size

Table 5.10: Gaussian Filtering

Image	PSNR (dB) 3x3	PSNR (dB) 5x5	PSNR (dB) 7x7	PSNR (dB) 9x9
noise.png (8 bit)	15.20	15.46	14.05	13.01
noise2.png (8 bit)	12.55	12.92	12.45	12.01
noise3.png (8 bit)	12.47	12.58	12.05	11.57
noise4.png (8 bit)	15.50	16.02	14.68	13.63
noise5.png (8 bit)	16.13	16.53	15.34	14.51
noise6.png (8 bit)	14.02	14.53	13.45	12.65
noise7.png (8 bit)	18.51	19.18	18.23	17.46
noise8.png (8 bit)	16.02	16.14	14.83	13.88
noise9.png (8 bit)	14.90	15.30	13.77	12.63
noise10.png (8 bit)	17.09	18.59	17.63	16.83
gnoise1.jpeg(24bit)	12.26	11.15	9.56	8.64
gnoise2.jpg (24bit)	16.44	16.52	15.30	14.30
gnoise3.png(24bit)	14.28	14.58	13.72	13.02
gnoise4.jpg(24 bit)	12.19	12.09	12.40	12.07
gnoise5.jpeg(24bit)	14.57	15.09	14.09	13.33
snoise1.jpeg (8 bit)	13.58	13.91	12.87	12.06
snoise2.jpeg (8 bit)	11.42	10.84	9.78	8.96
snoise3.jpeg (8 bit)	14.76	15.02	14.20	13.48
snoise4.jpg (24bit)	18.04	22.43	21.01	19.88
snoise5.jpeg(24bit)	13.12	13.78	12.90	12.15

According to data testing from Gaussian Filtering with size of kernels 3x3 , 5x5 , 7x7 , and 9x9 , it has a maximum value of PSNR 22.43 dB and a minimum value of PSNR 8.64 dB , the best kernel size is 5x5 because it has the higher PSNR than others size , and the lowest is 9x9 kernel size

Table 5.11: Average Filter

Method	Average PSNR Value
Median Filtering	18.33 dB
Gaussian Filtering	14.31 dB

According to data testing based on filtering method stated that Median filtering has the better PSNR value than Gaussian Filtering , Median Filtering has 18.33 dB of PSNR value and Gaussian Filtering has 14.31 dB of PSNR value