BAB IV

IMPLEMENTASI DAN PENGUJIAN SISTEM

Pada bab ini akan dijelaskan mengenai implementasi "Perbaikan Kontras Citra Retinopati Diabetes Menggunakan Teknik *Global Local Contrast Enhancement Berbasis Harmony Search*".

4.1 Perangkat Implementasi

Perangkat keras *(hardware)* dan perangkat lunak *(software)* yang digunakan dalam implementasi "Perbaikan Kontras Citra Retinopati Diabetes Menggunakan Teknik *Global Local Contrast Enhancement Berbasis Harmony Search*" adalah sebagai berikut.

A. Spesifikasi Perangkat Keras *(Hardware)*

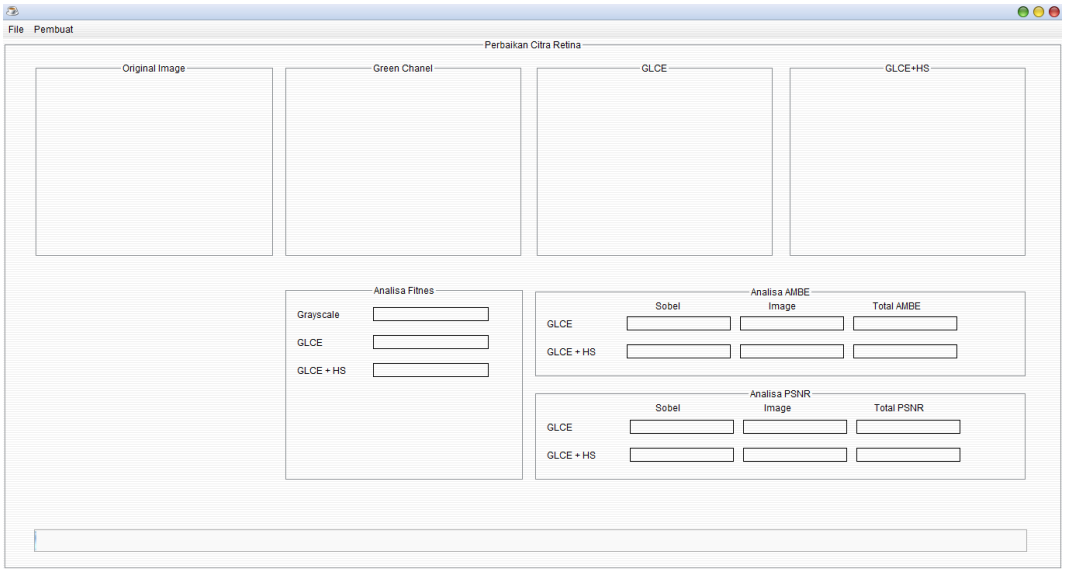
1. Laptop Lenovo G40 AMD A6.
2. Processor AMD A6-6310 1,8 GHZ.
3. RAM DDR3 4 GB.
4. Harddisk 500GB SATA.

B. Spesifikasi Perangkat Lunak *(Software)*

1. Operating System Microsoft Windows 7 Ultimate.
2. Netbean IDE.

4.2 Implementasi Sistem

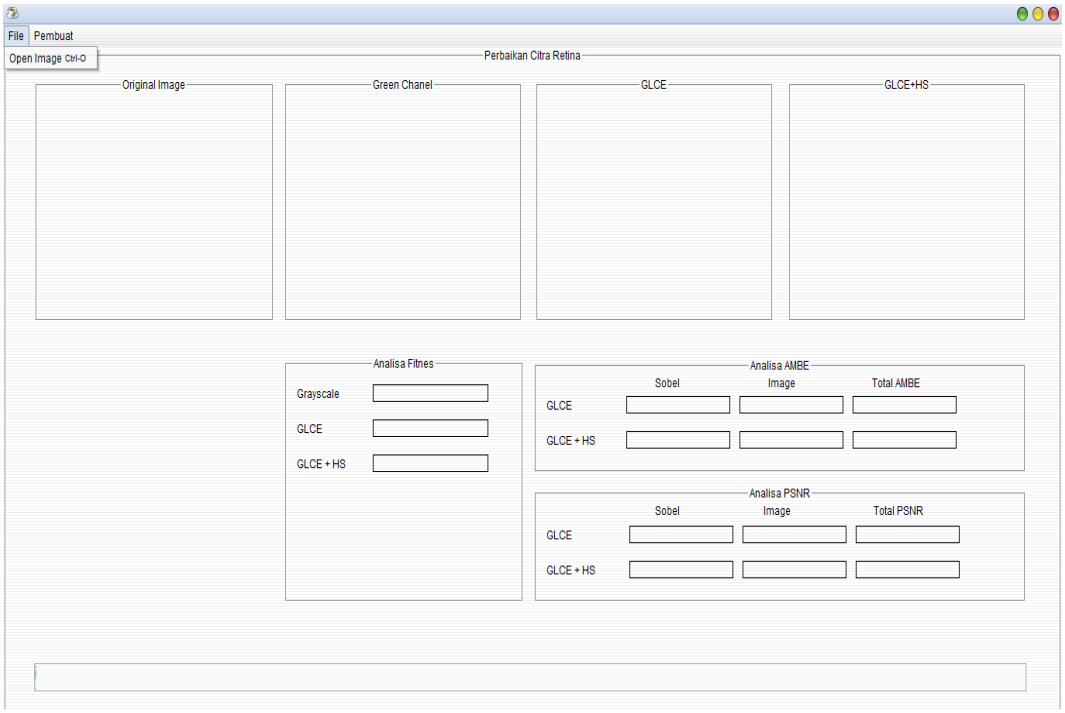
Implementasi dari sistem ini telah didasarkan pada arsitektur sistem yang telah dirancang pada Gambar 3.2. Gambar 4.1 merupakan hasil visualisasi halaman utama dari implementasi sistem.



Gambar 4.1 Visualisasi Implementasi Sistem

Pada halaman utama yang direpresentasikan pada Gambar 4.1 terdapat *panel OriginalImage, Green Chanel, GLCE, GLCE+HS, Analisis Fitness, Analisis AMBE,* dan *Analisis PSNR*. *Panel OriginalImage* difungsikan untuk menampilkan citra inputan retinopati diabetes yang diambil dari database *stare*. *Panel Green Chanel* difungsikan untuk menampilkan citra *grayscale* retinopati diabetes. *Panel GLCE* difungsikan untuk menampilkan citra *enhancement* retinopati diabetes menggunakan metode *GLCE* yang nantinya akan dijadikan pembanding untuk hasil dari metode *GLCE+HS*. *Panel Analisis Fitness, Analisis AMBE,* dan *Analisis PSNR* difungsikan untuk menampilkan hasil pengujian citra *enhancement* dari metode *GLCE,* dan *GLCE+HS*. Selain *panel-panel* yang telah disebutkan diatas, didalam halaman utama terdapat menu *open image*. Menu *open image* ini berfungsi untuk menjalankan semua proses yang telah dirangcang dalam sistem perbaikan kontras citra retinopati diabetes. Proses yang akan dijalankan ketika menekan menu open image adalah proses pemilihan citra, proses mengubah citra inputan menjadi

grayscale, proses *filter mean* dan *global mean*, proses *standard deviation*, proses melakukan perbaikan citra menggunakan metode *GLCE*, proses *harmony search*, proses melakukan perbaikan citra menggunakan metode *GLCE+HS* dan yang terakhir adalah proses analisis menggunakan metode *AMBE, PSNR,* dan *fitness*. Gambar 4.2 merupakan menu open image yang terdapat pada halaman utama.



Gambar 4.2 *Menu File*

4.2.1 Implementasi *Input Image*

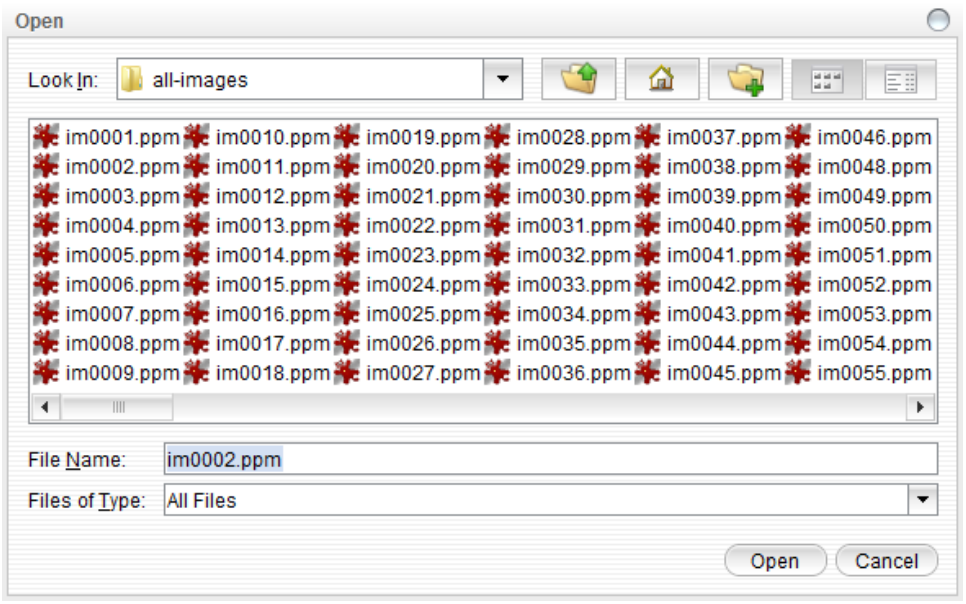
Pada bagian ini, citra akan diakuisisi untuk kemudian disimpan dalam sebuah variabel agar bisa diproses lebih lanjut. Gambar 4.3 adalah potongan proses akuisisi citra lalu disimpan dalam *Jpanel*.

```
public void openImage() {
    int option = fileChooser.showOpenDialog(getContentPane());
    if (option == JFileChooser.APPROVE_OPTION) {
        file = fileChooser.getSelectedFile();
        try {
            FileInputStream in = new FileInputStream(file);
            FileChannel channel = in.getChannel();
            ByteBuffer buffer = ByteBuffer.allocate((int) channel.size());
            channel.read(buffer);
            try {
                image = load(buffer.array());
            } catch (Exception e) {
                System.out.println(e);
            }

        } catch (IOException ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), "Exception", JOptionPane.ERROR_MESSAGE);
        }
        Image dimg = image.getScaledInstance(jLOriginal.getWidth(), jLOriginal.getHeight(), Image.SCALE_SMOOTH);
        jLOriginal.setIcon(new ImageIcon(dimg));
    } else {
        JOptionPane.showMessageDialog(getContentPane(), "File Not Found");
    }
}
```

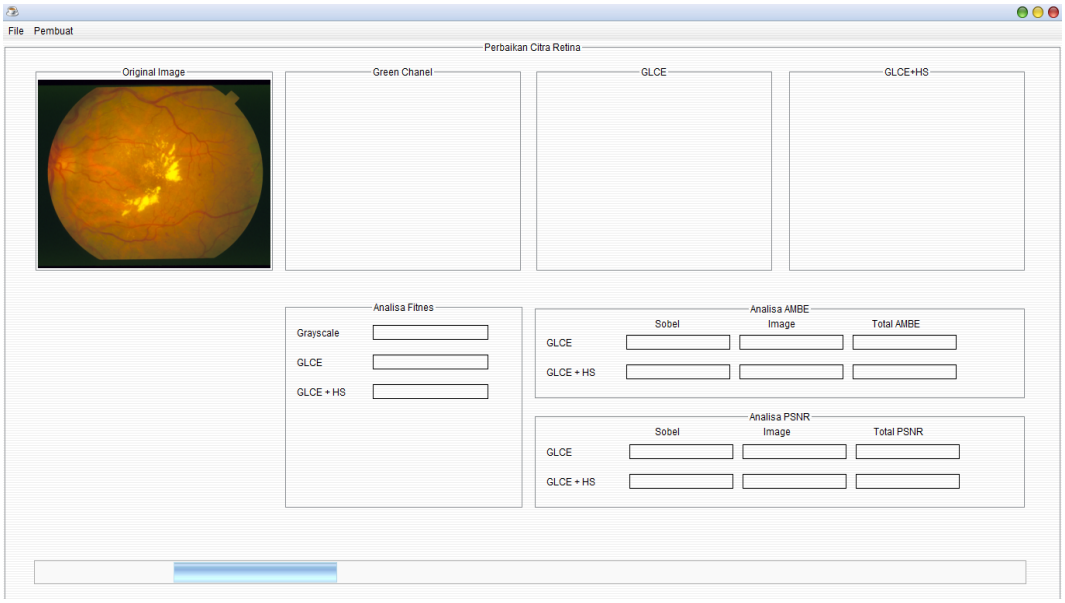
Gambar 4.3 Kode Program Akuisisi citra

Kode program yang diblok dengan warna biru berfungsi untuk membuka file manajer dari komputer. Hasil visualisasi dari kode program yang di blok warna biru dapat dilihat pada Gambar 4.4.



Gambar 4.4 Visualisasi *Open File Manager*

Kode program yang diblok dengan warna *orange* berfungsi untuk membaca file berdasarkan ukuran *bytes* dan kode program yang diblok warna merah muda berfungsi untuk menampilkan file yang telah dipilih kedalam *Jpanel*. Hasil visualisasi dari kode program yang diblok warna merah muda dapat dilihat pada Gambar 4.5.



Gambar 4.5 Visualisasi *Input Image*

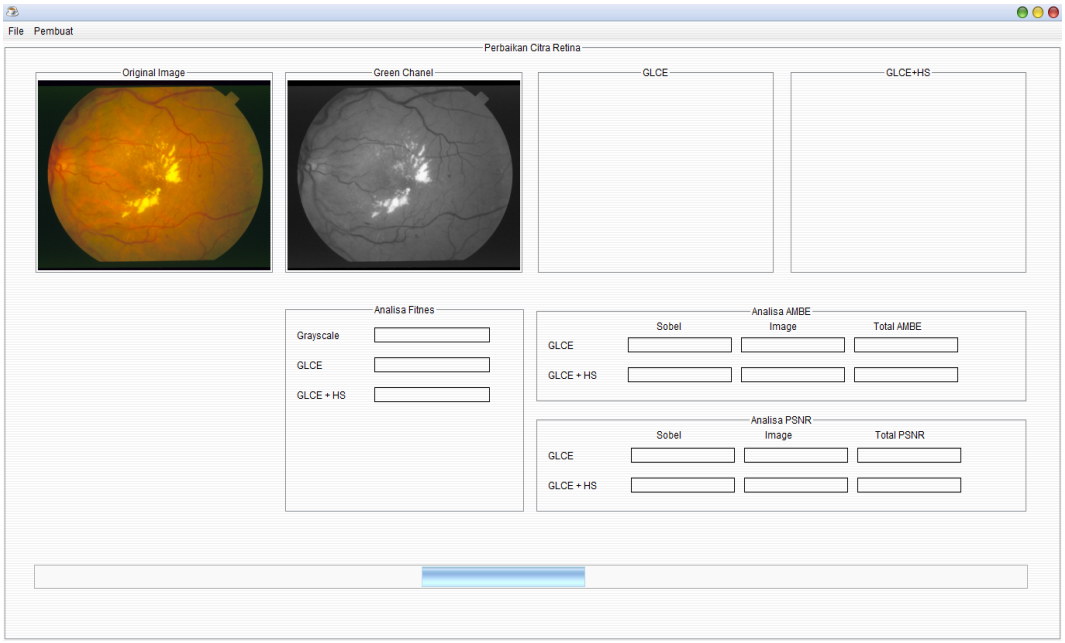
4.2.2 Impementasi *Grayscale*

Setelah citra diakuisisi, maka akan dilanjutkan pada langkah selanjutnya yaitu mengubah citra dalam bentuk *grayscale*. Gambar 4.6 adalah potongan kode program untuk melakukan konversi *grayscale*. Kode program yang diblok dengan warna kuning berfungsi untuk mengambil nilai dari piksel sebuah citra dengan warna hijau dan kode program yang diblok warna merah muda berfungsi untuk menampilkan citra *grayscale* yang didapatkan dari warna hijau sebuah citra

kedalam *Jpanel*. Hasil visualisasi dari kode program yang diblok warna merah muda dapat dilihat pada Gambar 4.7.

```
public void greenChanel(int matGreen[][]) {
        resetGambar();
        try {
            for (int i = 0; i < image.getWidth(); i++) {
                for (int j = 0; j < image.getHeight(); j++) {
                    int color = image.getRGB(i, j);
                    int green = (color & 0x0000ff00) >> 8;
                    matGreen[i][j] = green;
                    jmlGlobalMean(green);
                    Color newColor = new Color(green, green, green);
                    image.setRGB(i, j, newColor.getRGB());
                }
            }
            Image dimg = image.getScaledInstance(jLOriginal.getWidth(), jLOriginal.getHeight(), Image.SCALE_SMOOTH);
            jLGreen.setIcon(new ImageIcon(dimg));
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

Gambar 4.6 Kode Program *Grayscale*



Gambar 4.7 Visualisasi *Grayscale*

4.2.3 Implementasi *Local Mean* dan *Global Mean*

Setelah konversi *grayscale*, maka akan dilanjutkan pada langkah selanjutnya yaitu melakukan proses *filter mean* dan *global mean*. Gambar 4.8 adalah potongan kode program untuk *global mean.*

```
public void jmlGlobalMean(double globalMean) {
        try {
            this.globalMean += globalMean / (image.getWidth() * image.getHeight());
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

Gambar 4.8 Kode Program *Global Mean*

Gambar 4.8 adalah potongan kode program berupa fungsi untuk menghitung nilai rata-rata keseluruhan dari citra *grayscale*. Langkah selanjutnya adalah melakukan proses *filter mean* yang direpresentasikan pada Gambar 4.9.

```
public void meanFilter(int matriksGreen[][], double mean[][]) {
    try {
        for (int i = 0; i < image.getWidth(); i++) {
            for (int j = 0; j < image.getHeight(); j++) {
                mean[i][j] = matriksGreen[i][j];
            }
        }
        for (int i = 1; i < image.getWidth() - 1; i++) {
            for (int j = 1; j < image.getHeight() - 1; j++) {
                double baris1 = matriksGreen[i - 1][j - 1] + matriksGreen[i - 1][j] + matriksGreen[i - 1][j + 1];
                double baris2 = matriksGreen[i][j - 1] + matriksGreen[i][j] + matriksGreen[i][j + 1];
                double baris3 = matriksGreen[i + 1][j - 1] + matriksGreen[i + 1][j] + matriksGreen[i + 1][j + 1];
                mean[i][j] = (baris1 + baris2 + baris3) / 9;
            }
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Gambar 4.9 Kode Program *Filter Mean*

Kode program yang berwarna merah muda pada Gambar 4.9 berfungsi untuk menduplikasi matriks yang berisikan nilai dari citra *grayscale*. Setelah menduplikasi matriks langkah selanjutnya adalah menyusun spatial domain dari

matriks dengan ukuran 3x3 kemudian menjumlahkan semua nilai piksel matriks tersebut dan dibagi dengan luas matriks untuk mendapatkan nilai rata-rata, implementasi proses tersebut ditunjukkan oleh kode program dengan blok warna kuning. Proses selanjutnya adalah mengganti nilai yang terdapat pada matriks duplikat citra *grayscale* dengan nilai tengah dari matriks spatial domain dengan ketentuan untuk matriks duplikat dengan indek piksel mulai dari 1 dan diakhiri dengan indeks terkahir dikurang dengan nilai 1.

4.2.4 Implementasi *Standard Deviation*

Setelah proses *filter mean* dan *global mean*, maka akan dilanjutkan pada langkah selanjutnya yaitu melakukan proses *standard deviation*. Gambar 4.10 adalah potongan kode program untuk *standard deviation.*

```
public void sdvFilter(int matriksGreen[][], double sdv[][], double mean[][]) {
    try {
        sdv = new double[image.getWidth()][image.getHeight()];
        for (int i = 0; i < image.getWidth(); i++) {
            for (int j = 0; j < image.getHeight(); j++) {
                sdv[i][j] = matriksGreen[i][j];
            }
        }
        for (int i = 1; i < image.getWidth() - 1; i++) {
            for (int j = 1; j < image.getHeight() - 1; j++) {
                double baris1 = Math.pow(matriksGreen[i - 1][j - 1] - mean[i - 1][j - 1], 2) +
                                Math.pow(matriksGreen[i - 1][j] - mean[i - 1][j], 2) +
                                Math.pow(matriksGreen[i - 1][j + 1] - mean[i - 1][j + 1], 2);
                double baris2 = Math.pow(matriksGreen[i][j - 1] - mean[i][j - 1], 2) +
                                Math.pow(matriksGreen[i][j] - mean[i][j], 2) +
                                Math.pow(matriksGreen[i][j + 1] - mean[i][j + 1], 2);
                double baris3 = Math.pow(matriksGreen[i + 1][j - 1] - mean[i + 1][j - 1], 2) +
                                Math.pow(matriksGreen[i + 1][j] - mean[i + 1][j], 2) +
                                Math.pow(matriksGreen[i + 1][j + 1] - mean[i + 1][j + 1], 2);
                sdv[i][j] = Math.sqrt((baris1 + baris2 + baris3) / 8);
            }
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Gambar 4.10 Kode Program *Standard Deviation*

Kode program yang berwarna merah muda pada Gambar 4.10 berfungsi untuk menduplikasi matriks yang berisikan nilai dari citra *grayscale*. Setelah menduplikasi matriks langkah selanjutnya adalah menyusun spatial domain dari matriks dengan ukuran 3x3 kemudian melakukan proses perhitungan *standard deviation,* implementasi proses tersebut ditunjukkan oleh kode program dengan blok warna kuning. Proses selanjutnya adalah mengganti nilai yang terdapat pada matriks duplikat citra *grayscale* dengan nilai tengah dari matriks spatial domain dengan ketentuan untuk matriks duplikat dengan indek piksel mulai dari 1 dan diakhiri dengan indeks terkahir dikurang dengan nilai 1.

4.2.5 Impelemtasi GLCE

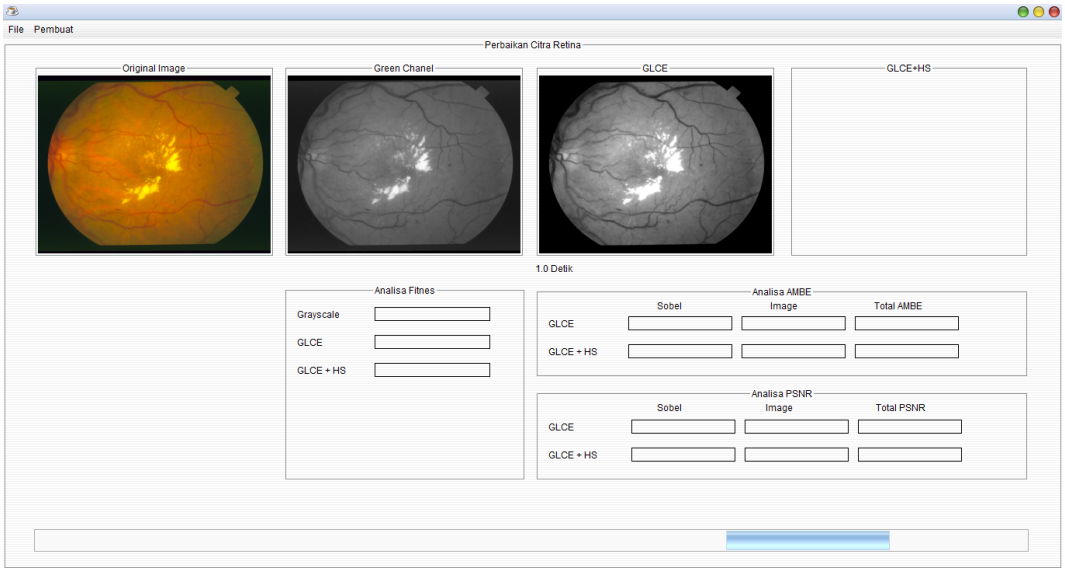
```
public void GLCE(int matGreen[][], double local[][]) {
        try {
            resetGambar();
            matrikGLCE = new double[image.getWidth()][image.getHeight()];
            for (int i = 0; i < image.getWidth(); i++) {
                for (int j = 0; j < image.getHeight(); j++) {
                    double glce = matGreen[i][j] + (local[i][j] - globalMean);
                    if (glce > 255) {
                        glce = 255;
                    }
                    if (glce < 0) {
                        glce = 0;
                    }
                    matrikGLCE[i][j] = glce;
                    Color newColor = new Color((int) glce, (int) glce, (int) glce);
                    image.setRGB(i, j, newColor.getRGB());
                }
            }
            Image dimg = image.getScaledInstance(jLOriginal.getWidth(), jLOriginal.getHeight(), Image.SCALE_SMOOTH);
            jLGLCE.setIcon(new ImageIcon(dimg));
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

Gambar 4.11 Kode Program GLCE

Setelah melakukan proses *grayscale, filter mean, dan standard deviation,* maka akan dilanjutkan pada langkah selanjutnya yaitu melakukan proses *global local contrast enhancement (GLCE).* Pengimplementasian *GLCE* pada tugas akhir

ini difungsikan untuk pembanding dari metode *GLCE+HS* yang diusulkan. Gambar 4.11 adalah potongan kode program untuk metode *GLCE*.

Kode program dengan blok warna kuning pada Gambar 4.11 merupakan rumus dari metode *GLCE*. Hasil yang didapatkan dari perumusan metode *GLCE* kemudian akan ditampung pada matriks, proses ini bertujuan untuk memudahkan dalam proses pengujian dari citra yang dihasilkan oleh metode *GLCE*. Kode program dengan blok merah muda merupakan implementasi penampungan hasil dari perumusan metode *GLCE*. Kode program dengan blok biru berfungsi untuk menampilkan hasil dari metode *GLCE* kedalam *Jpanel*. Gambar 4.12 merupakan hasil visualisasi dari metode *GLCE*.



Gambar 4.12 Visualisasi GLCE

4.2.6 Implementasi *Harmony Search*

Setelah melakukan proses *grayscale, filter mean, standard deviation,* dan *GLCE* maka akan dilanjutkan pada langkah selanjutnya yaitu melakukan proses

harmony search. proses *harmony search* untuk mencari parameter *a, b, c, k* yang terdapat dalam formula *GLCE+HS*. Untuk nilai *a* terdiri dari rentang nilai (0 - 1,5), nilai b (0 – *globalMean* / 3), nilai c (0 - 1), dan k (0,5 – 1,5) *Harmony search* diimplementasikan berdasarkan *flowchart* pada Gambar 3.3 yaitu *inisialisasi parameter (HMS, NI, HMCR,* dan *PAR), inisialisasi harmony memory, menghitung fitness,* dan *improvisasi vector harmony memory*. Gambar 4.13 dan Gambar 4.14 adalah potongan kode program untuk *inisialisasi parameter* dan *inisialisasi harmony memory*.

```
public int hms = 100, iterasi = 200, banyakVariabel = 4, generation = 0;

public void inisialisasiHS() {
    try {

        hmcr = 0.9;
        par = 0.6;
        bwMin = 0.001;

    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Gambar 4.13 Kode Program Inisialisasi Parameter

Pada kode program yang diblok dengan warna kuning pada Gambar 4.13 dan Gambar 4.14 merupakan inisialisasi parameter dan *harmony memory* untuk metode *GLCE+HS.* Proses inisialisasi *harmony memory* merupakan pembentukan kandidat awal pada proses *harmony search*. Implementasi dari pembentukan kandidat awal ditunjukkan oleh kode program dengan blok warna hijau pada Gambar 4.14. Setelah terbentuk kandidat awal, langkah selanjutnya adalah memasukkan kandidat tersebut kedalam formula *global local contrast enhancement* untuk dihitung nilai *fitness* atau *fungsi objektif* dari setiap kandidat.

```
public void harmonyMemori(int banyakVariabel, double nilaiMaxmin[][], RandomGenerator random) {
    try {
        for (int i = 0; i < hms; i++) {
            for (int j = 0; j < banyakVariabel; j++) {
                harmoniMemori[i][j] = nilaiMaxmin[0][j] + ((nilaiMaxmin[1][j] - nilaiMaxmin[0][j]) * random.ran1());
            }

            matrikEnhance1 = new double[0][0];
            matrikEnhance1 = new double[image.getWidth()][image.getHeight()];
            fungsiEnhancemen(matrikEnhance1, harmoniMemori[i]);
            nilaiFitnes = hitungFitnes(matrikEnhance1);

            harmoniMemori[i][banyakVariabel] = nilaiFitnes;
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Gambar 4.14 Kode Program Inisialisasi *Harmony Memory*

```
public void fungsiEnhancemen(double matriks[][], double matrik[]) {
    try {
        resetGambar();
        for (int i = 0; i < image.getWidth(); i++) {
            for (int j = 0; j < image.getHeight(); j++) {
                int color = image.getRGB(i, j);
                int green = (color & 0x0000ff00) >> 8;
                double nilai1 = matrik[3] * globalMean / (standartDeviasi[i][j] + matrik[1]);
                double nilai2 = green - (matrik[2] * matrikFilter[i][j]);
                double nilai3 = Math.pow(matrikFilter[i][j], matrik[0]);
                double hasil = 0;

                hasil = (nilai1 * (nilai2 + nilai3));
                if (hasil < 0) {
                    hasil = 0;
                } else if (hasil > 255) {
                    hasil = 255;
                }


                matriks[i][j] = hasil;
            }
        }
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Gambar 4.15 Kode Program Fungsi *Enhancement*

```
public void itensitasGreen(double matrik[][]) {
        try {
            mapMatrik = new HashMap<>();
            for (int i = 0; i < image.getWidth(); i++) {
                for (int j = 0; j < image.getHeight(); j++) {
                    int green = (int) matrik[i][j];
                    Integer count = mapMatrik.get(green);
                    if (count == null) {
                        mapMatrik.put(green, 1);
                    } else {
                        mapMatrik.put(green, ++count);
                    }
                }
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }

public void entropy() {
        try {
            double probabilitas = 0;
            jumlahMN = image.getWidth() * image.getHeight();
            nilaiEntropi = 0;
            for (int i : mapMatrik.values()) {
                probabilitas = i / jumlahMN;
                nilaiEntropi += -1 * (probabilitas * Math.log(probabilitas) / Math.log(2));
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

Gambar 4.16 Kode Program Nilai *Entropy*

Gambar 4.15 merupakan kode program formulasi dari kedua metode. Kode program dengan blok warna biru merupakan implementasi rumus dari metode *GLCE+HS*. Kode program dengan blok warna kuning pada Gambar 4.15 merupakan detail dari perumusan variabel nilai1, nilai2, dan nilai3.

Setelah memasukan kandidat awal, langkah selanjutnya adalah melakukan perhitungan nilai *fitness* atau *fungsi objektif* yang didapatkan berdasarkan formula nomor 20 yaitu dengan perpaduan dari nilai *entropy*, jumlah itensitas, serta informasi *edge*. Gambar 4.16 merupakan kode program *entropy*. Kode program dengan blok warna kuning merupakan nilai yang dihasilkan dari matriks kandidat

awal yang telah dimasukan kedalam proses sebelumnya. Kode program dengan blok warna merah muda merupakan proses menghitung banyak itensitas yang dihaslikan dari matriks kandidat awal yang telah dimasukan kedalam proses sebelumnya. Kode program dengan blok warna biru merupakan implementasi dari perhitungan *entropy*.

Langkah selanjunya adalah melakukan proses perhitungan informasi *edge* yang meliputi jumlah itensitas dari *edge*, dan banyak *edge* yang terdeteksi melalui perhitungan *operator sobel* tersebut. Gambar 4.17 adalah kode program dari *operator sobel*. Kode program dengan blok warna kuning pada Gambar 4.17 merupakan proses duplikasi dari nilai matriks kandidat awal yang telah dimasukan kedalam proses sebelumnya. Kode program dengan blok warna merah muda merupakan implementasi dari perumusan *operator sobel*.

```
public void sobel(double matrix[][]) {
    try {
        double sml[][] = new double[image.getWidth()][image.getHeight()];
        double snl[][] = new double[image.getWidth()][image.getHeight()];
        matrikSobel = new double[image.getWidth()][image.getHeight()];
        double tampung = 0;
        jmlIntensitas = 0;
        banyakGaris = 0;
        for (int i = 0; i < image.getWidth(); i++) {
            for (int j = 0; j < image.getHeight(); j++) {
                matrikSobel[i][j] = matrix[i][j];
            }
        }

        for (int i = 1; i < image.getWidth() - 1; i++) {
            for (int j = 1; j < image.getHeight() - 1; j++) {
                sml[i][j] = matrix[i + 1][j - 1] + (2 * matrix[i + 1][j]) + matrix[i + 1][j + 1] - (matrix[i - 1][j - 1] + (2 * matrix[i - 1][j]) + matrix[i - 1][j + 1]);
                snl[i][j] = matrix[i + 1][j - 1] + (2 * matrix[i][j - 1]) + matrix[i - 1][j - 1] - (matrix[i + 1][j + 1] + (2 * matrix[i][j + 1]) + matrix[i - 1][j + 1]);
                tampung = Math.round(Math.sqrt(Math.pow(sml[i][j], 2) + Math.pow(snl[i][j], 2)));
                if (tampung > 255) {
                    tampung = 255;
                } else if (tampung < 0) {
                    tampung = 0;
                }
                matrikSobel[i][j] = tampung;
            }
        }

    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Gambar 4.17 Kode Program Operator Sobel

Selanjutnya adalah melakukan perhitungan jumlah itensitas *edge*, serta menghitung banyak edge yang direpresentasikan pada Gambar 4.18 dan 4.19.

```
public double intensitasSobel() {
    double nilai = 0;
    for (int i = 0; i < image.getWidth(); i++) {
        for (int j = 0; j < image.getHeight(); j++) {
            nilai += matrikSobel[i][j];
        }
    }
    return nilai;
}
```

Gambar 4.18 Kode Program Itensitas *Edge*

Kode program dengan blok warna kuning pada Gambar 4.18 merupakan implementasi dari perhitungan atau penjumlahan keseluruhan dari nilai yang dihasilkan melalui proses operator sobel.

Kode program dengan blok warna kuning pada Gambar 4.19 merupakan implementasi dari ekstraksi *edge* dari *operator sobel*. Setelah melakukan perhitungan nilai *entropy*, jumlah itensitas, dan banyak *edge*, langkah selanjutnya adalah memasukkan ketiga nilai tersebut ke dalam formulasi nomor 20. Gambar 4.20 merupakan kode program perhitungan *fitness*.

```
public void TH_Sobel(double matrikSobel[][]) {
    try {
        int loop = 0;
        for (int i = 1; i < image.getWidth() - 1; i++) {
            for (int j = 1; j < image.getHeight() - 1; j++) {
                int tampung = (int) Math.round(matrikSobel[i][j] / 255);
                if (tampung > 0) {
                    loop++;
                }
            }
        }
        banyakGaris = loop;
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Gambar 4.19 Kode Program Banyak *Edge*

```
public double hitungFitnes(double matrix[][]) {
        double nilai = 0;
        jmlIntensitas = 0;
        mapMatrik = new HashMap<>();
        sobel(matrix);
        itensitasGreen(matrix);
        entropy();
        TH_Sobel(matrikSobel);
        jmlIntensitas = intensitasSobel();
        if (jmlIntensitas != 0 && banyakGaris != 0 && nilaiEntropi != 0) {
            nilai = (Math.log(Math.log(jmlIntensitas)) * (banyakGaris / jumlahMN) * nilaiEntropi);
        } else {
            nilai = 0;
        }
        return nilai;
    }
```

Gambar 4.20 Kode Program Hitung *Fitness*

```
public void harmonySearch(double nMaxmin[][], int banyakVariabel, RandomGenerator random) {
        try {
            generation = 0;
            inisialisasiHS(kode);
            while (stopCondition()) {
                for (int j = 0; j < banyakVariabel; j++) {
                    if (random.ran1() < hmcr) {
                        memoriConsideration(j, random);
                        if (random.ran1() < par) {
                            bwMax = (1.0 / 20.0) * (nMaxmin[1][j] - nMaxmin[0][j]);
                            bw = bwMax * Math.exp((Math.log(bwMin / bwMax) / iterasi) * generation);
                            double temp = newHM[j];
                            temp += random.ran1() * bw;
                            if (temp < nMaxmin[1][j] && temp > nMaxmin[0][j]) {
                                newHM[j] = temp;
                            }
                        }
                    } else {
                        newHM[j] = nMaxmin[0][j] + ((nMaxmin[1][j] - nMaxmin[0][j]) * random.ran1());
                    }
                }
                matrikEnhance1 = new double[0][0];
                matrikEnhance1 = new double[image.getWidth()][image.getHeight()];
                fungsiEnhancemen(matrikEnhance1, newHM);
                nilaiFitnes = hitungFitnes(matrikEnhance1);

                updateHarmony(nilaiFitnes, banyakVariabel);
                generation++;
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

Gambar 4.21 Kode Program Improvisasi *Harmony Memory*

Kode program dengan blok warna kuning pada Gambar 4.20 merupakan implementasi dari perhitungan *fitness* atau fungsi obejektif. Setelah mendapatkan kandidat awal serta nilai *fitness* dari setiap kandidat, langkah selanjutnya adalah melakukan proses *improvisasi harmony memory* untuk mendapatkan kandidat yang terbaik. Gambar 4.21 adalah kode program *improvisasi harmony memory*.

Kode program dengan blok warna kuning dari Gambar 4.21 merupakan implementasi dari proses penentuan frekuensi *memory consideration*. Cara kerja dari proses penentuan frekuensi *memory consideration* adalah apabila nilai random lebih dari nilai *HMCR* maka membangkitkan kandidat baru yang diimplementasikan pada kode program dengan blok warna *orange* dan sebaliknya apabila nilai random kurang dari nilai *HMCR* maka akan melanjutkan ke proses berikutnya yaitu mengacak nilai dari kandidat parameter yang dihasilkan oleh proses sebelumnya. Proses selanjutnya adalah menentukan frekuensi *pitch adjustment* yang ditunjukkan oleh kode program dengan blok warna merah muda. Cara kerja dari proses penentuan frekuensi *pitch adjustment* adalah apabila nilai random kurang dari nilai *PAR* maka nilai dari kandidat parameter yang dihasilkan melalui proses *memory consideration* dijumlahkan dengan hasil perkalian dari nilai random dengan *bandwith. Bandwith* berfungsi untuk mengontrol pencarian lokal disekitar *harmony memory*. Kode program dengan blok warna biru merupakan implementasi dari perumusan *bandwith.* Setelah improvisasi kandidat parameter telah selesai dilakukan proses selanjutnya adalah menghitung nilai *fitness* dari kandidat tersebut, dengan aturan kandidat yang memiliki nilai *fitness* yang rendah

akan dieleminasi oleh kandidat *fitness* yang lebih tinggi. Proses ini terus diulang hingga mencapai batas iterasi yang telah ditentukan.

Kode program fungsi memori Consideration dan UpdateHarmony yang terdapat pada Gambar 4.21 direpresentasikan pada Gambar 4.22 dan 4.23.

```
public void memoriConsideration(int indek, RandomGenerator random) {
    newHM[indek] = harmoniMemori[random.randVal(0, hms - 1)][indek];
}
```

Gambar 4.22 Kode Program *Memory Consideration*

```
public void updateHarmony(double newFitnes, int banyakVariabel) {
    try {
        double worst = harmoniMemori[0][banyakVariabel];
        int worstIndex = 0;
        for (int i = 0; i < hms; i++) {
            if (harmoniMemori[i][banyakVariabel] < worst) {
                worst = harmoniMemori[i][banyakVariabel];
                worstIndex = i;
            }
        }
        worstFitHistory[generation] = worst;
        if (newFitnes > worst) {
            for (int i = 0; i < banyakVariabel; i++) {
                harmoniMemori[worstIndex][i] = newHM[i];
            }
            harmoniMemori[worstIndex][banyakVariabel] = newFitnes;
        }

        double best = harmoniMemori[0][banyakVariabel];
        int bestIndex = 0;
        for (int i = 0; i < hms; i++) {
            if (harmoniMemori[i][banyakVariabel] > best) {
                best = harmoniMemori[i][banyakVariabel];
                bestIndex = i;
            }
        }
        bestFitHistory[generation] = best;
        for (int i = 0; i < banyakVariabel; i++) {
            bestHarmony[i] = harmoniMemori[bestIndex][i];
        }
        bestHarmony[banyakVariabel] = best;
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

Gambar 4.23 Kode Program *Update Harmony*

Kode program dengan blok warna kuning pada Gambar 4.22 merupakan implementasi rumus dari penentuan kandidat *harmony memory* secara acak. Pengambilan kandidat *harmony memory* secara acak dimulai dari indek ke 0 sampai indek terakhir dari kandidat *harmony memory*.

Kode program pada Gambar 4.23 dengan blok warna kuning merupakan implementasi untuk pengambilan nilai *fitness* dari kandidat awal. Kode program dengan blok warna merah muda merupakan proses pencarian nilai *fitness* terendah.Kode program dengan blok warna *orange* merupakan proses eleminasi untuk kandidat yang mempunyai nilai *fitness* paling rendah. Kode program dengan blok warna biru merupakan proses mencari kandidat dengan nilai *fitness* yang terbaik. Kandidat terbaik didapatkan dengan cara memilih kandidat yang mempunyai nilai *fitness* tertinggi. Hal ini disebabkan apabila memilih kandidat dengan *fitness* paling rendah, maka hasil keluaran citra memiliki pencahayaan yang berlebihan. Gambar 4.24 adalah perbandingan hasil metode *GLCE+HS* pada sampel citra im0032.ppm yang menggunakan kandidat dengan *fitness* yang rendah dan kandidat dengan *fitness* yang tinggi.

Grayscale Kandidat *Fitness* Tinggi Kandidat *Fitness* Rendah

Gambar 4.24 Perbandingan Hasil GLCE+HS berdasarkan *Fitness*

Setelah menemukan kandidat fitness terbaik lankah selanjutnya adalah menyimpan kandidat dengan *fitness* tertinggi kedalam sebuah matriks yang direpresentasikan pada kode program yang berwarna abu-abu dari Gambar 4.23.

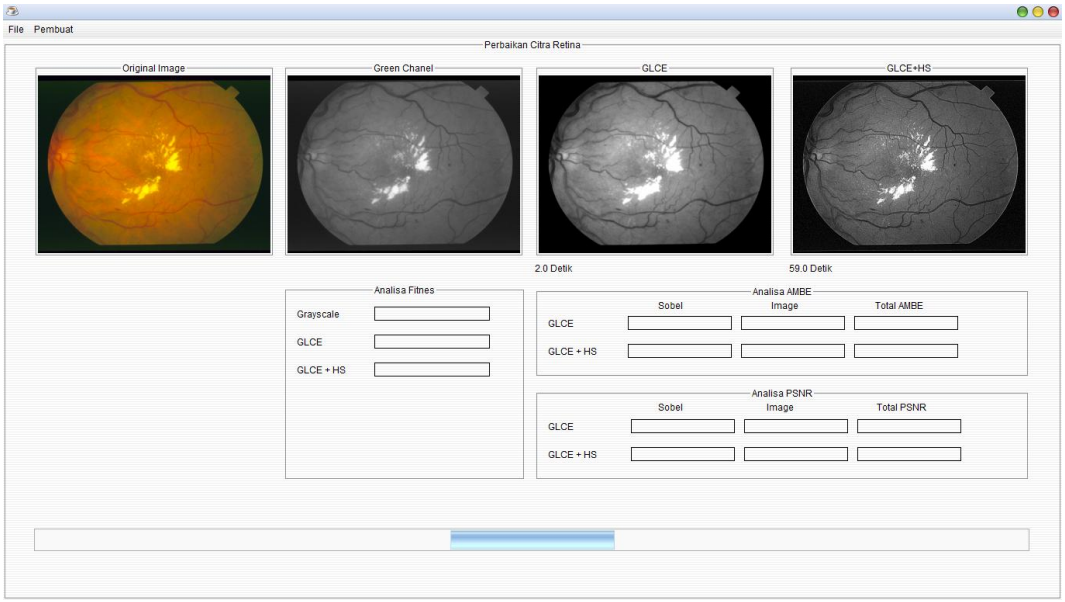
4.2.7 Implementasi GLCE+HS

Langkah selanjutnya adalah memasukkan kandidat yang dihasilkan melalui proses *harmony search* ke dalam formula *GLCE+HS*. Gambar 4.25 merupakan kode program dari proses transformasi. Kode program dengan blok warna ungu pada Gambar 4.25 merupakan perumusan dari *GLCE+HS*, dan kode program dengan blok hijau berfungsi untuk menampilkan hasil trasformasi citra kedalam *Jpanel*. Hasil visualisasi *GLCE+HS* ditunjukan pada Gambar 4.26.

```
public void transformasiCitra(double matrikEnhance[][], JLabel pilihFrame) {
        try {
            resetGambar();
            for (int i = 0; i < image.getWidth(); i++) {
                for (int j = 0; j < image.getHeight(); j++) {
                    int color = image.getRGB(i, j);
                    int green = (color & 0x0000ff00) >> 8;
                    double nilai1 = bestHarmony[3] * globalMean / (standartDeviasi[i][j] + bestHarmony[1]);
                    double nilai2 = green - (bestHarmony[2] * matrikFilter[i][j]);
                    double nilai3 = Math.pow(matrikFilter[i][j], bestHarmony[0]);
                    double hasil = 0;
                    hasil = (nilai1 * nilai2 + nilai3) + ((matrikFilter[i][j] - globalMean) / 2);
                    if (hasil > 255) {
                        hasil = 255;
                    } else if (hasil < 0) {
                        hasil = 0;
                    }
                    matrikEnhance[i][j] = Math.round(hasil);
                    Color newColor = new Color((int) matrikEnhance[i][j], (int) matrikEnhance[i][j], (int) matrikEnhance[i][j]);
                    image.setRGB(i, j, newColor.getRGB());
                }
            }
            Image dimg = image.getScaledInstance(jLOriginal.getWidth(), jLOriginal.getHeight(), Image.SCALE_SMOOTH);
            pilihFrame.setIcon(new ImageIcon(dimg));

        } catch (Exception e) {
            System.out.println(e);
        }
    }
```

Gambar 4.25 Kode Program Transformasi Citra



Gambar 4.26 Visualisasi GLCE+HS

4.2.8 Implementasi Analisis PSNR, AMBE, dan *Fitness*

Langkah terakhir dalam implementasi sistem ini adalah dengan melakukan proses analisis sistem dari metode *GLCE* dan *GLCE+HS* (metode yang diusulkan). Analisis sistem dilakukan menggunakan tiga metode yang pertama dari kombinasi nilai *entropy*, banyak garis, dan jumlah itensitas atau disebut dengan nilai *fitness*, pengujian kedua dilakukan menggunakan metode *PSNR (Peak Signal to Noise Ratio),* dan pengujian yang terakhir menggunakan metode *AMBE (Absolute Mean Brigthness Error).* Implementasi pengujian fitness ditunjukkan pada Gambar 4.20 sedangkan untuk implementasi pengujian *PSNR* dan *AMBE* ditunjukkan pada Gambar 4.27. Pada kode program dengan blok berwarna kuning merupakan perumusan dari metode *AMBE*, Metode ini dihasilkan berdasarkan perbedaan rata-rata itensitas dari citra asli dengan citra *enhance.* Pada kode program dengan blok

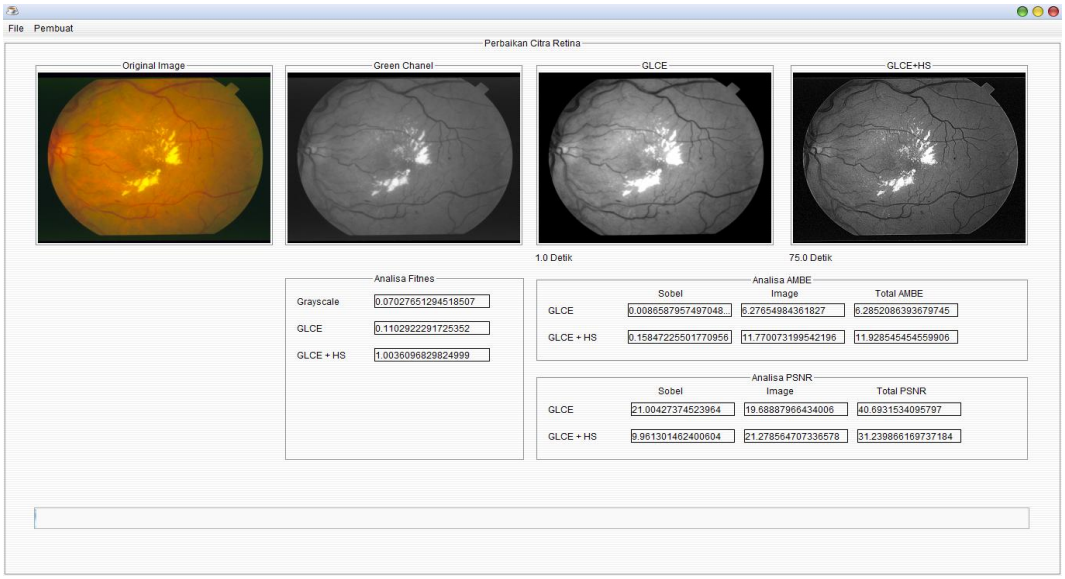
berwarna merah muda merupakan perumusan dari metode *PSNR*, Metode ini dihasilkan berdasarkan perbedaan nilai itensitas citra perpiksel.

```
for (int i = 0; i < image.getWidth(); i++) {
      for (int j = 0; j < image.getHeight(); j++) {
          meanEnhance += matrikEnhance1[i][j] / luasCitra;
          }
 }
globalMean = Math.abs(meanAsli - meanEnhance);
```

```
psnr = 0;
double temp = 0, mse = 0;
double luasCitra = image.getWidth() * image.getHeight();
for (int i = 0; i < image.getWidth(); i++) {
   for (int j = 0; j < image.getHeight(); j++) {
      temp += Math.pow(matrikA[i][j] - matrikB[i][j], 2);
   }
}
mse = temp / luasCitra;
psnr = 10 * Math.log10(Math.pow(255, 2) / mse);
```

Gambar 4.27 Analisis Sistem

Hasil visualisasi dari keseluruhan implementasi sistem direpresentasikan pada Gambar 4.28.



Gambar 4.28 Visualisasi Akhir

4.3 Skenario Uji Coba

Tahap pengujian bertujuan untuk mengetahui kinerja dari metode *global local enhancement berbasis harmony search (GLCE+HS)* yang diusulkan dalam tugas akhir ini. Secara garis besar, pengujian akan dilakukan dengan dua tahap yaitu pengujian terhadap algoritma *harmony search* serta pengujian dari metode yang diusulkan. Pengujian metode yang diusulkan, dilakukan dengan cara menghitung nilai *PSNR, AMBE,* dan kombinasi antara *entropy,* jumlah itensitas *edge* dan banyak garis yang terdeteksi. Kemudian akan dibandingkan dengan hasil dari metode *global local enhancement (GLCE),* dan *optimum green plane masking berbasis adaptive genetic algorithm (OPGM+AGA).* Data uji yang digunakan dalam pengujian *global local contrast enhancement,* dan *global local contrast enhancement berbasis harmony search* adalah citra diabetes retinopati yang berjumlah 362 citra, sementara data pengujian terhadap metode *optimum green plane masking berbasis adaptive genetic algorithm* sesuai dengan *paper* atau *jurnal* (Daniel, Ebenezer dan Anita, J , 2015).

4.3.1 Hasil Pengujian *Harmony Search*

Pengujian dibawah ini bertujuan untuk melihat kinerja dari algoritma *harmony search* dalam mencari perpaduan parameter yang terdapat pada perumusan dari metode yang diusulkan. Pada tugas akhir ini parameter *harmony search* yang digunakan adalah sesuai dengan penelitian yang dilakukan Albetar, dkk (2016) yaitu *HMS* = 100, *PAR* = 0,6, *HMCR* = 0,9, *NI* = 200, banyak variabel = 4.

Pengujian algoritma *harmony search* dilakukan dengan cara mencari manual perpaduan parameter *a, b, c, k* yang akan dicari, kemudian dibandingkan dengan hasil pencarian menggunakan algoritma *harmony search.* Tabel 4.1 merupakan hasil pengujian dari parameter *harmony search* menggunakan *sampel* citra retinopati diabetes im0001.ppm yang telah diubah ukurannya menjadi 3 x 4 piksel. Perubahan ukuran citra ini dilakukan untuk memudahkan proses perhitungan secara manual.

Tabel 4.1 Pengujian *Harmony Search* Sampel Citra im0001.ppm

Jenis Pencarian	Parameter a	Parameter b	Parameter c	Parameter k
Manual	1,099	39,200	0,300	0,5
Harmony search	0,883	39,501	0,363	0,646

Berdasarkan informasi pada Tabel 4.1 menunjukan bahwa algoritma *harmony search* telah bekerja dengan baik. Hal ini terlihat dari selisih antara parameter hasil pencarian manual dengan parameter hasil pencarian *harmony search* mempunyai selisih yang cukup rendah. Tabel 4.2 merupakan *error* parameter yang dihasilkan melalui algoritma *harmony search* pada citra im0001.ppm.

Tabel 4.2 *Error* Parameter Pada Citra im0001.ppm

Parameter	*Error* (Selisih Pencarian Manual Dengan *Harmony Search*)
A	0,216
B	0,301

Parameter	*Error* (Selisih Pencarian Manual Dengan *Harmony Search*)
C	0,063
K	0,146

4.3.2 Hasil Pengujian Metode GLCE+HS

Pengujian dibawah ini untuk melihat kinerja metode *global local contrast enhancement berbasis harmony search* untuk peningkatan kontras citra retinopati diabetes serta membandingkan hasil dari metode tersebut dengan metode *global local contrast enhancement,* dan *optimum green plane masking berbasis adaptive genetic algorithm*. Pengujian dilakukan dengan menggunakan metode *PSNR, AMBE,* dan *fitness* (kombinasi nilai *entropy*, jumlah itensitas *edge*, dan banyak *edge*). *PSNR* merupakan metode pengujian yang dilakukan untuk menguji tingkat *noise* dari sebuah citra. Semakin besar nilai *PSNR* yang dihasilkan maka citra *enhancement* memiliki kualitas yang semakin mendekati citra aslinya, begitu juga sebaliknya. *AMBE* merupakan metode pengujian yang dilakukan untuk menguji *preventing brightness* dari sebuah citra. Semakin kecil nilai *AMBE* yang dihasilkan maka citra *enhancement* memiliki kualitas yang semakin mendekati citra aslinya, begitu juga sebaliknya. *Fitness* (kombinasi nilai *entropy*, jumlah itensitas *edge*, dan banyak *edge*) merupakan pengujian yang dilakukan untuk mengukur *preventing edge* dan penyebaran itensitas pada citra. Semakin rendah nilai *fitness* maka penyebaran itensitas dan *preventing edge* yang dihasilkan oleh citra *enhancement* semakin mendekati citra aslinya, begitu juga sebaliknya. Citra asli retinopati

diabetes yang diambil dari database *stare* adalah citra retinopati dengan kontras yang rendah serta ber*noise*.

1. Pengujian *PSNR*

Pengujian *PSNR* dilakukan dengan tiga tahap, yang pertama dengan cara melakukan perhitungan *PSNR* berdasarkan *edge* yang dihasilkan melalui algoritma *sobel,* kemudian melakukan perhitungan *PSNR* berdasarkan gambar (citra) yang dihasilkan oleh setiap metode, dan yang terakhir menjumlahkan nilai *PSNR edge* dan nilai *PSNR image(citra)*. Tabel 4.3 merupakan hasil rata-rata pengujian *PSNR* citra retinopati diabetes dari metode *GLCE* dan *GLCE+HS.*

Tabel 4.3 Rata-rata Pengujian PSNR

Metode	$PSNR_{edges}$	$PSNR_{image}$	$Total\ PSNR$
GLCE	22,51727	19,13642	41,65369
GLCE+HS	10,13642	20,33002	30,46644

2. Pengujian *AMBE*

Pengujian *AMBE* dilakukan dengan tiga tahap, yang pertama dengan cara melakukan perhitungan *AMBE* berdasarkan *edge* yang dihasilkan melalui algoritma *sobel,* kemudian melakukan perhitungan *AMBE* berdasarkan gambar (citra) yang dihasilkan oleh setiap metode, dan yang terakhir menjumlahkan nilai *AMBE edge* dan nilai *AMBE image(citra)*. Tabel 4.4 merupakan hasil rata-rata pengujian *AMBE* citra retinopati diabetes dari metode *GLCE* dan *GLCE+HS.*

Tabel 4.4 Rata-rata Pengujian AMBE

Metode	$AMBE_{edge}$	$AMBE_{image}$	$Total\ AMBE$
GLCE	0,112647	10,73535	10,78768
GLCE+HS	0,16663	13,68185	13,84848

3. Pengujian *Fitness*

Pengujian *Fitness* merupakan pengujian yang didapatkan dari kombinasi nilai *entropy*, jumlah itensitas *edge*, dan banyak *edge*. Tabel 4.5 merupakan hasil rata-rata pengujian *fitness* citra retinopati diabetes dari metode *GLCE*, dan *GLCE+HS.*

Tabel 4.5 Rata-rata Pengujian *fitness*

Metode	$Fitnes$
GLCE	0,124373
GLCE+HS	1,028202

4. Pengujian *GLCE+HS* dibandingkan *OPGM+AGA*

Pengujian ini adalah pengujian yang dilakukan dengan membandingkan hasil *PSNR*, dan *AMBE* dari hasil citra enhancement menggunakan metode *GLCE+HS* dan *OPGM+AGA.* Tabel 4.6 adalah pengujian antara metode *GLCE+HS* dan metode *OPGM+AGA* yang diambil dari penelitian Daniel, Ebenezer dan Anita, J (2015) menggunakan sampel citra retinopati diabetes im0001.ppm, im0002.ppm, im0003.ppm, im0004.ppm, im0005.ppm, im0008.ppm, dan im0402.ppm.

Tabel 4.6 Pengujian *GLCE+HS* dibandingkan *OPGM+AGA*

Nama Citra	Metode	$AMBE_{edge}$	$AMBE_{image}$	$PSNR_{edges}$	$PSNR_{image}$
Im0001	*OPGM+AGA*	0,0052	0,1500	66,8579	63,7372
	GLCE+HS	0,158	11,805	9,961	21,278
Im0002	*OPGM+AGA*	0,0055	0,0936	66,6925	67,6668
	GLCE+HS	0,154	8,671	9,941	21,358
Im0003	*OPGM+AGA*	0,0031	0,1280	68,2612	65,1365
	GLCE+HS	0,149	12,058	10,522	21,205
Im0004	*OPGM+AGA*	0,0037	0,2073	68,6036	60,8677
	GLCE+HS	0,067	10,406	11,844	22,356
Im0005	*OPGM+AGA*	0,0072	0,0451	65,7182	73,9254
	GLCE+HS	0,196	10,543	9,575	20,923
Im0008	*OPGM+AGA*	0,0015	0,1299	70,8312	64,2877
	GLCE+HS	0,088	7,04	12,217	23,079
Im0402	*OPGM+AGA*	0,0156	0,0617	63,7585	71,2514
	GLCE+HS	0,173	10,817	9,845	20,939

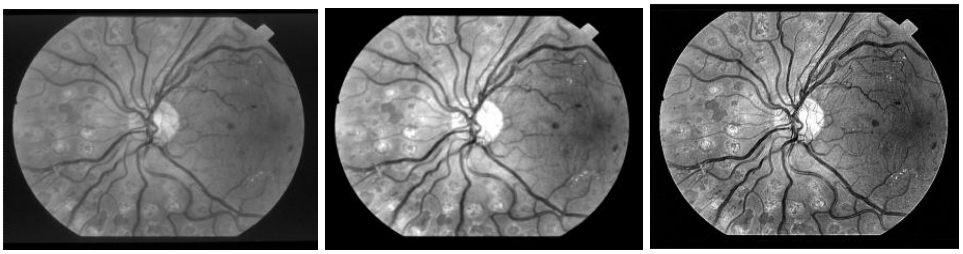
4.4 Analisis Sistem

Pada bab ini akan dibahas analisa hasil percobaan sistem yang telah dilakukan.

4.4.1 Analisis Sistem Hasil Metode GLCE+HS dengan GLCE

Berdasarkan informasi pada Tabel 4.3, Tabel 4.4 dan Tabel 4.5 metode *GLCE* memiliki nilai rata-rata *PSNR* yang lebih tinggi, serta memiliki nilai rata-rata *AMBE*

dan *fitness* yang lebih rendah dibandingkan metode *GLCE+HS.* Nilai rata-rata *PSNR, fitness* dan nilai rata-rata *AMBE* dari metode *GLCE* adalah 41,65369, 0,124373, dan 10,78768 sedangkan metode *GLCE+HS* adalah 30,46644, 1,028202, dan 13,84848. Dengan hasil tersebut, metode *GLCE* menampilkan citra keluaran yang dapat mereduksi *noise,* hal ini disebabkan oleh kecerahan warna pada objek meningkat sehingga *noise* pada citra asli tidak terlihat, akan tetapi peningkatan kecerahan warna yang dihasilkan menggunakan metode *GLCE* pada beberapa bagian citra cukup berlebihan sehingga mengakibatkan *edge* pada citra menjadi tidak terlihat, sedangkan metode metode *GLCE+HS* menampilkan citra keluaran yang dapat mereduksi *noise* serta dapat mempertahankan *edge,* hal ini disebabkan kecerahan warna pada objek meningkat sehingga *noise* pada citra asli tidak terlihat, dan penambahan pencahayaan *(brightness)* pada objek tidak berlebihan. Hasil visualisasi citra asli retinopati diabetes im0347.ppm dan citra hasil *enhancement* metode *GLCE,* dan *GLCE+HS* direpresentasikan pada Gambar 4.29.



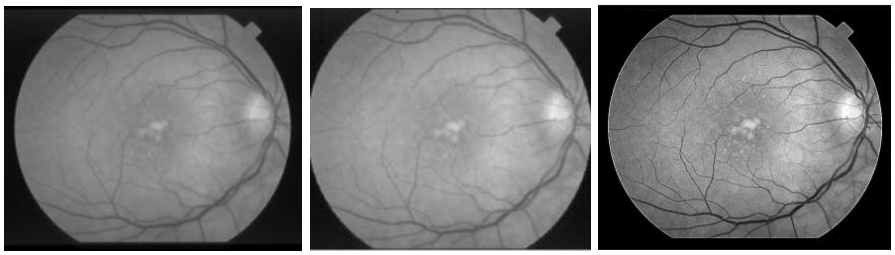
A B C

Gambar 4.29 Visualisasi Citra : (A). *Grayscale* Citra Asli, (B). GLCE, (C). GLCE+HS

4.4.2 Analisis Sistem Hasil Metode GLCE+HS dengan OPGM+AGA

Berdasarkan informasi dari Tabel 4.6 metode *OPGM+AGA* memiliki nilai *PSNR* yang lebih tinggi dan nilai *AMBE* yang lebih rendah dibandingkan metode

GLCE+HS. Oleh karena itu, citra keluaran yang dihasilkan menggunakan metode *OPGM+AGA* adalah citra dengan kualitas yang hampir sama dengan citra aslinya. Citra asli retinopati diabetes yang diambil dari database *stare* adalah citra retinopati dengan kontras yang rendah serta ber*noise* sehingga citra keluaran yang dihasilkan menggunakan metode *GLCE+HS* lebih baik dibandingkan citra keluaran dari metode *OPGM+AGA*. Hasil visualisasi citra asli retinopati diabetes im0402.ppm dan citra hasil *enhancement* metode *OPGM+AGA* yang diambil dari penelitian Daniel, Ebenezer dan Anita, J (2015) dan metode *GLCE+HS* direpresentasikan pada Gambar 4.30.



A B C

Gambar 4.30 Visualisasi Citra : (A). *Grayscale* Citra Asli, (B). OPGM+AGA, (C). GLCE+HS