PRAKTIKUM 8
Pengambangan (Thresholding)

8.1. Tujuan :

Mahasiswa mengetahui cara membuat program segmentasi citra dengan metode pengambangan (thresholding) menggunakan metode otsu untuk penentuan nilai ambang / threshold secara otomatis.

8.2. Dasar Teori :

8.2.1. Segmentasi

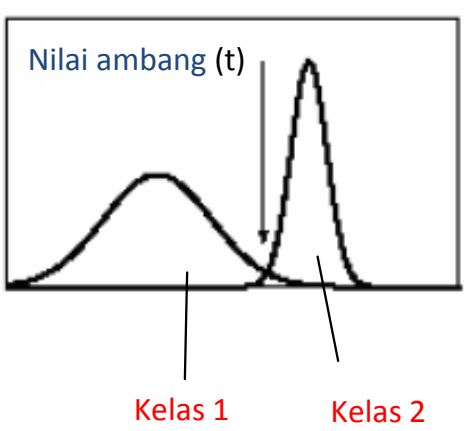
Segmentasi citra merupakan proses yang ditujukan untuk mendapatkan objek-objek yang terkandung di dalam citra atau membagi citra ke dalam beberapa daerah dengan setiap objek atau daerah memiliki kemiripan atribut. Pada citra yang mengandung hanya satu objek, objek dibedakan dari latarbelakangnya.

8.2.2. Deteksi tepi

Deteksi tepi berfungsi untuk memperoleh tepi objek. Deteksi tepi memanfaatkan perubahan nilai intensitas yang drastis pada batas dua area. Definisi tepi di sini adalah “himpunan piksel yang terhubung yang terletak pada batas dua area” (Gonzalez & Woods, 2002). Perlu diketahui, tepi sesungguhnya mengandung informasi yang sangat penting. Informasi yang diperoleh dapat berupa bentuk maupun ukuran objek.

8.2.3. Metode Otsu

Metode Otsu dipublikasikan oleh Nobuyuki Otsu pada tahun 1979. Metode ini menentukan nilai ambang dengan cara membedakan dua kelompok, yaitu objek dan latarbelakang, yang memiliki bagian yang saling bertumpukan, berdasarkan histogram (lihat Gambar 8.1).



Gambar 8.1 Penentuan nilai ambang
untuk memperoleh hasil yang optimal

Prinsip metode *Otsu* dijelaskan berikut ini. Pertama-tama, probabilitas nilai intensitas *i* dalam histogram dihitung melalui

$$p(i) = \frac{n_i}{N}, p(i) \geq 0, \sum_1^{256} p(i) = 1 \qquad (8.1)$$

dengan n_i menyatakan jumlah piksel berintensitas *i* dan *N* menyatakan jumlah semua piksel dalam citra. Jika histogram dibagi menjadi dua kelas (objek dan latarbelakang), pembobotan pada kedua kelas dinyatakan sebagai berikut:

$$w_1(t) = \sum_{i=1}^{t} p(i) \qquad (8.2)$$

$$w_2(t) = \sum_{i=t+1}^{L} p(i) = 1 - w_1(t) \qquad (8.3)$$

Dalam hal ini, L menyatakan jumlah aras keabuan. Rerata kedua kelas dihitung melalui:

$$m_1(t) = \sum_{i=1}^{t} i.p(i) \,/W_1(t) \qquad (8.4)$$

$$m_2(t) = \sum_{i=1}^{t} i.p(i) \,/W_2(t) \qquad (8.5)$$

Varians kedua kelas dinyatakan dengan rumus:

$$\sigma_1{}^2(t) = \sum_{i=1}^{t}(1 - m_1)^2 . \frac{p(i)}{W_1(t)} \quad (8.6)$$

$$\sigma_2{}^2(t) = \sum_{i=t+1}^{L}(1 - m_2)^2 . \frac{p(i)}{W_2(t)} \quad (8.7)$$

Varians total dapat dinyatakan dengan

$$\sigma^2(t) = \sigma_W{}^2(t) + \sigma_B{}^2(t) \quad (8.8)$$

Dalam hal ini, $\sigma_W{}^2$ dinamakan sebagai *within-class variance* (WCV) dan $\sigma_B{}^2$ disebut sebagai *between-class variance* (BCV). WCV dapat dinyatakan dengan

$$\sigma_W{}^2(t) = W_1(t) . \sigma_1(t)^2 + W_2(t) . \sigma_2(t)^2 \quad (8.9)$$

Rumus di atas menunjukkan bahwa WCV adalah jumlah varians kelas secara individual yang telah diboboti dengan probabilitas kelas masing-masing. Adapun BCV dinyatakan dengan

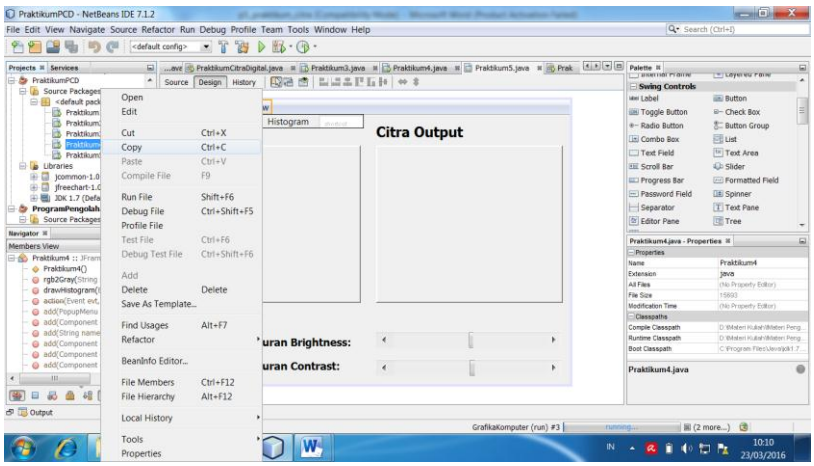
$$\sigma_B{}^2(t) = W_1 . [m_1(t) - m_T]^2 + W_2 . [m_2(t) - m_T]^2 \quad (8.10)$$

Dalam hal ini, m_T adalah rerata total ($m_T = \sum_{i=1}^{N} i . p(i)$).

Nilai ambang optimum dapat diperoleh dengan dua cara. Cara pertama dilaksanakan dengan meminimumkan WCV. Cara kedua dilaksanakan dengan memaksimumkan BCV. Namun, berdasarkan kedua cara tersebut, cara yang kedua lebih menghemat komputasi.

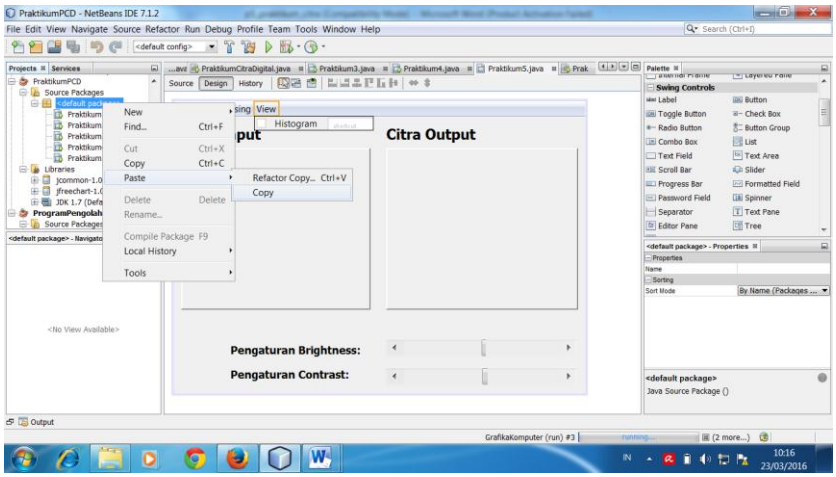
8.2. Langkah Praktikum :

1). Copy class JFrame Form pada praktikum sebelumnya dengan cara klik kanan pada folder Source Package Praktikum7 lalu pilih Copy seperti tampak pada gambar 8.2 berikut:



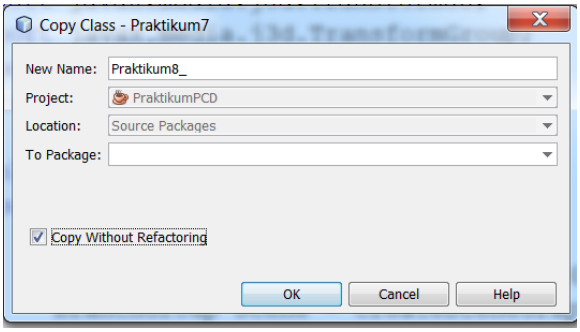
Gambar 8.2. Copy class JFrame Form

2). Kemudian klik kanan pada folder source packages pada <default Packages>, kemudian pilih Paste kemudian pilih Copy, seperti tampak pada gambar 8.3.



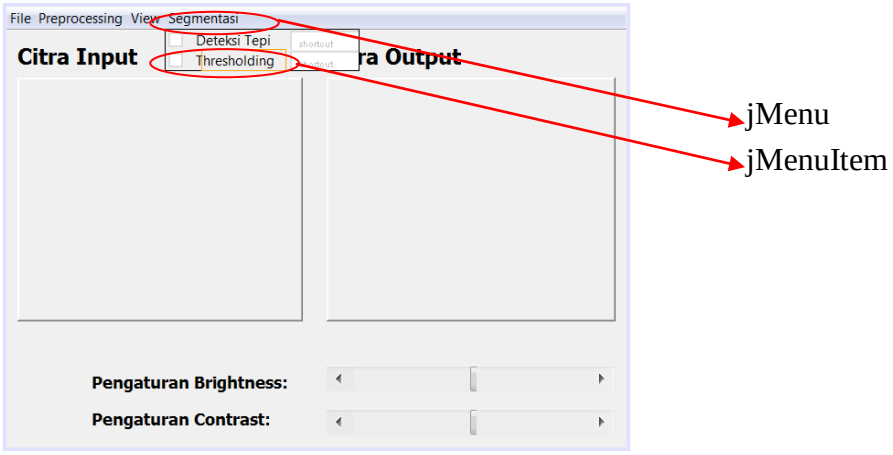
Gambar 8.3. Paste ke class JFrame Form baru

3). Rubah nama class menjadi Praktikum8 kemudian centang pilihan Copy Without Refactoring. Kemudian klik OK. Seperti tampak pada gambar 8.4. kemudian rubah nama class dari Praktikum7_1 menjadi Praktikum8.



Gambar 8.4. Jendela Dialog Copy Class

4). Klik Design, kemudian tambahkan 1 buah jMenuItem baru pada Menu Segmentasi tersebut kemudian rubah text nya menjadi Thresholding. Seperti tampak dalam gambar 8.5. berikut:



Gambar 8.5. Design Form

5). Tambahkan Kode metode untuk proses segmentasi pada bagian Source Praktikum8 pada event method jMenuItem Thresholding sebagai berikut.

```
private void jMenuItem7ActionPerformed(java.awt.event.ActionEvent evt) {
  BufferedImage grayscale = thresholding(sumber);
  int x = jLabel2.getWidth();
  int y = jLabel2.getHeight();
  ImageIcon imageIcon = new ImageIcon(resize(grayscale, x, y));
  jLabel2.setIcon(imageIcon);
}
```

6). Tambahkan methode baru untuk proses thresholding dengan metode otsu dengan kode sebagai berikut:

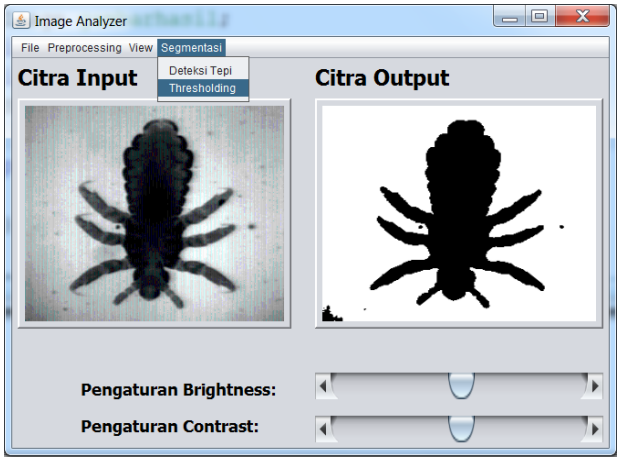
```
public BufferedImage thresholding(String sumber) {
int [] histogram = new int[256];
double variance=0, totalMean=0, maxVariance=0, firstCumuMoment=0,
       zerothCumuMoment=0;
int threshold=0;
BufferedImage prosesGambar;
BufferedImage loadIng = loadImage(sumber);
ukuranX = loadIng.getWidth();
ukuranY = loadIng.getHeight();
prosesGambar = new BufferedImage(ukuranX, ukuranY, BufferedImage.TYPE_BYTE_GRAY);
Graphics g = prosesGambar.getGraphics();
g.drawImage(loadIng, 0, 0, null);
WritableRaster raster = prosesGambar.getRaster();
for (int i=0; i < 256; i++){
    histogram[i]=0;
}

for (int x = 0; x < ukuranX; x++) {
    for (int y = 0; y < ukuranY; y++) {
        int rgb = loadIng.getRGB(x, y);
```

```
            int hijaug = (rgb >> 8) & 0xff;
            histogram[hijaug]++;
        }
    }

    double area= (ukuranX * ukuranY);
    for (int k=0; k< 256; k++) {
       totalMean = totalMean + ((k * histogram[k]) / area);
    }
    for (int j=0; j<256; j++) {
       zerothCumuMoment = zerothCumuMoment + histogram[j] / area;
       firstCumuMoment =  firstCumuMoment + (j * histogram[j] / area);
       variance = totalMean * zerothCumuMoment -  firstCumuMoment;
       variance = variance * variance;
       if ((zerothCumuMoment != 0) && (zerothCumuMoment != 1)) {
          variance = variance /(zerothCumuMoment * (1 - zerothCumuMoment));
          if (maxVariance < variance) {
              maxVariance = variance;
              threshold = j;
          }
       }
    }
    for (int x = 0; x < ukuranX; x++) {
        for (int y = 0; y < ukuranY; y++) {
            int rgb = loadIng.getRGB(x, y);
            int hijaug = (rgb >> 8) & 0xff;
            if (hijaug >= threshold) {
                rgb=255;
            }
            else if (hijaug < threshold){
                rgb=0;
            }
            raster.setSample(x, y, 0, rgb);
        }
    }
    return prosesGambar;
}
```

7). Jalankan program sehingga tampilannya seperti gambar 8.6. berikut:



Gambar 8.6. Tampilan aplikasi hasil proses thresholding menggunakan metode otsu