TUGAS

IMAGE ENHANCEMENT DENGAN METODE *HISTOGRAM MATCHING*

Oleh:

Almas Fauzia Wibawa

17/409427/PA/17734

PROGRAM STUDI ILMU KOMPUTER

DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVESITAS GADJAH MADA

YOGYAKARTA

2019

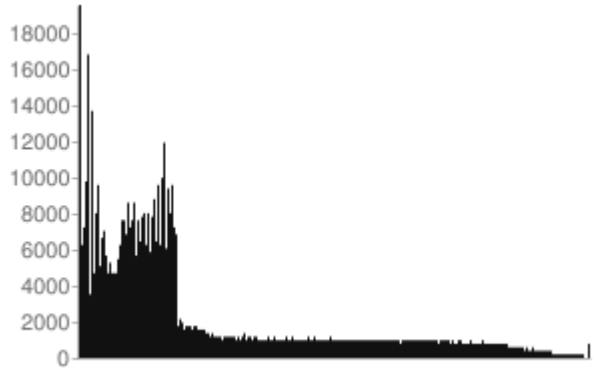
BAB I

PENDAHULUAN

1. Metode *Histogram Matching*

Histogram Matching merupakan salah satu metode penyesuaian citra dengan histogram yang kita inginkan. Dalam pembuatan histogram yang diinginkan tersebut, dilakukan *histogram equalization*, yaitu merupakan salah satu metode dalam proses *image enhancement*. Metode ini digunakan untuk mengurangi kontras pada citra. Pendekatan yang dilakukan adalah pendekatan terhadap frekuensi intensitas.

Pertama-tama, perlu dibangun histogram yang merepresentasikan frekuensi setiap intensitas pada citra yang muncul di setiap pikselnya. Pada citra 8-bit, *range* frekuensi yang mungkin muncul ada 0-255 setiap warnanya (jika digunakan RGB, maka 0-255 untuk masing-masing Red, Green, dan Blue). Contoh histogram dari suatu citra:



Tidak ratanya tinggi dari histogram yang tergambar, mengartikan bahwa kontras pada citra tersebut besar. Pada *histogram matching*, dilakukan terlebih dahulu normalisasi histogram, yaitu meratakan tinggi dari histogram tersebut, kemudian dilakukan *matching*, yaitu mencocokkan histogram yang diinginkan pada citra yang ingin diolah.

Terdapat beberapa rumus yang digunakan dalam *histogram matching*. Pada proses normalisasi, diterapkan rumus pencarian *cumulative distribution function* (CDF) pada setiap frekuensi intensitas, dimana CDF memiliki rumus:

$$cdf(i) = \sum_{j=0}^{i} p(j)$$

$$p(i) = \frac{n_i}{n}, \qquad 0 \leq i < L$$

cdf(i) = CDF dari intensitas i
p(i) = probabilitas intensitas i pada citra
n_i = frekuensi intensitas i
n = jumlah piksel keseluruhan pada citra

CDF tersebut kemudian dikalikan dengan intensitas maksimal yang mungkin, yaitu 255, untuk melahirkan frekuensi baru. Apabila hasil kali itu melebihi 255, intensitas baru ditetapkan 255. Sampai sini, akan dihasilkan histogram baru dengan frekuensi yang lebih merata.

Tahap selanjutnya adalah melakukan *histogram matching*. Terdapat 255 intensitas yang dicatat frekuensinya. Telah diset juga frekuensi agar tidak melebihi 255 (intensitas maksimal yang mungkin pada citra tersebut). Maka, pada *histogram matching*, kita dapat lakukan perubahan nilai intensitas dengan cara: piksel dengan intensitas i diubah menjadi berintensitas sebesar frekuensi intensitas i pada histogram.

BAB II
PEMBAHASAN

Pada eksperimen kali ini, akan dicoba dilakukan *image enhancement* dengan metode *histogram equalization* dan *histogram matching*. Kedua metode tersebut akan dikenai pada tiga citra di bawah ini:

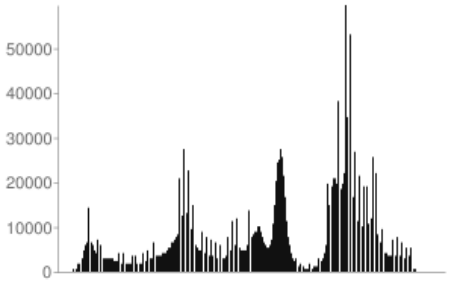
(1)

(3)

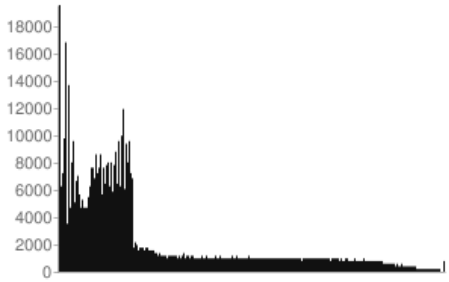
(2)

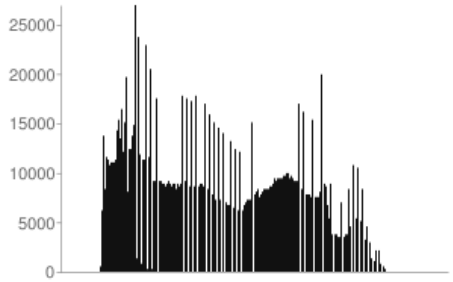
Ketiga citra tersebut memiliki histogram yang relatif beda jauh sebagai berikut:

(1)



(2)

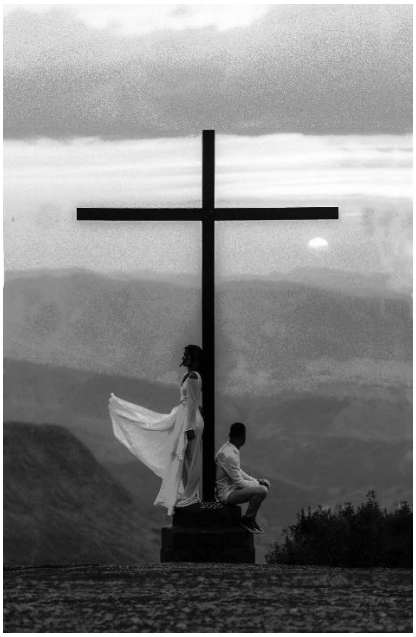




(3)

1. *Histogram Equalization*

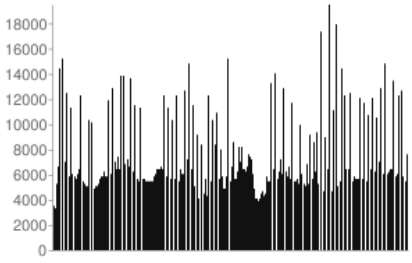
Setelah dikenai *histogram equalization*, ketiga citra tersebut menjadi:

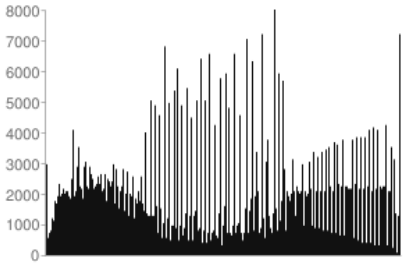
(1) 

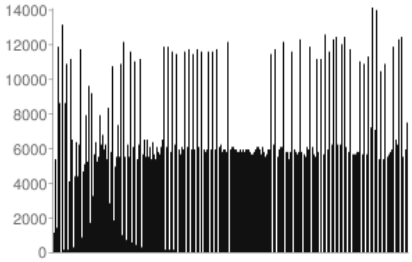
(3) 

(2) 

Ketiga citra tersebut dihasilkan dari histogram yang telah diratakan sebagai berikut:

(1) 

(2) 

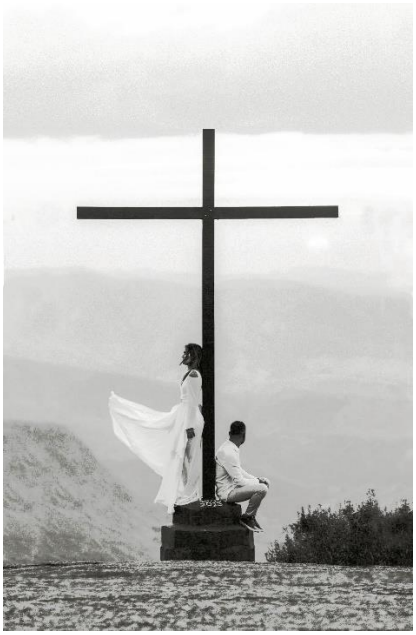
(3) 

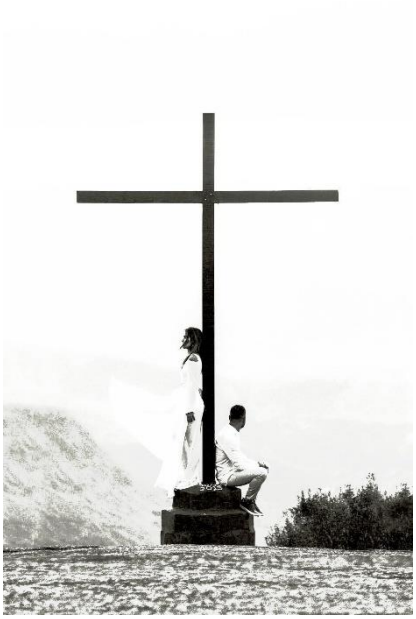
Perataan frekuensi pada histogram tersebut menghasilkan gambar dengan kontras yang lebih rendah.

2. *Histogram Matching*

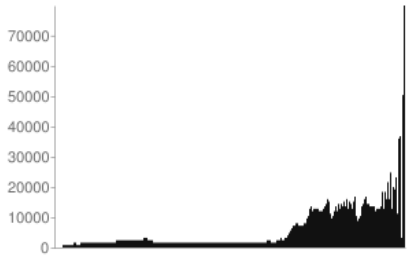
Setelah dilakukan *histogram equalization*, dilakukan eksperimen *histogram matching* pada setiap citra. Histogram yang digunakan adalah histogram dari 2 citra lain.

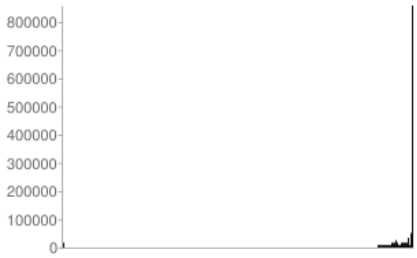
Berikut perubahan yang terjadi pada citra pertama setelah dikenai *histogram matching* dengan histogram citra kedua dan ketiga secara berturut-turut:

(1) 

(2) 

Kedua citra di atas memiliki histogram sebagai berikut:

(1) 

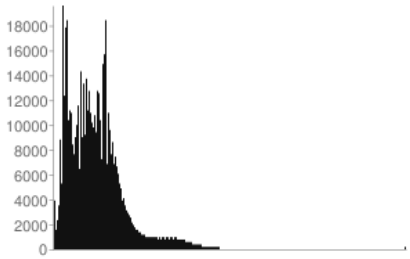
(2) 

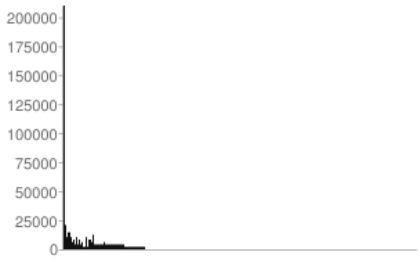
Selanjutnya, berikut ini merupakan perubahan yang terjadi pada citra kedua setelah dilakukan *histogram matching* dengan histogram yang dimiliki citra pertama dan ketiga secara berturut-turut:

(1) 

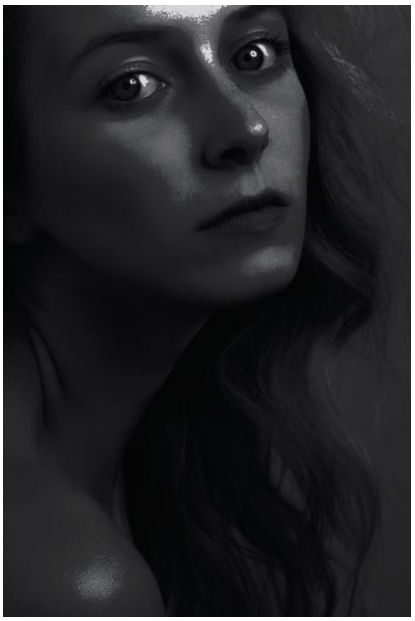
(2) 

Histogram dari kedua citra tersebut adalah sebagai berikut:

(1) 

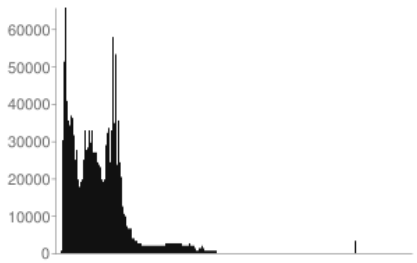
(2) 

Dan yang terakhir, berikut adalah perubahan yang terjadi pada citra ketiga setelah dilakukan *histogram matching* dengan histogram yang dimiliki oleh citra pertama dan kedua secara berturut-turut:

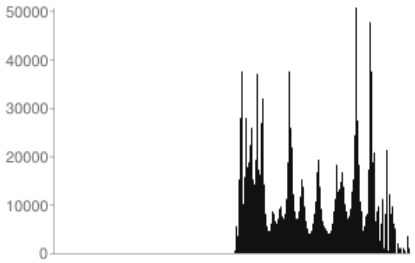
(1) 

(2) 

Dan berikut ini merupakan histogram dari kedua citra tersebut:



(1)



(2)

Dari semua eksperimen di atas, dapat disimpulkan bahwa dengan histogram citra lain, sebuah citra tidak akan bisa melalui proses *histogram equalization* sebagaimana tujuan proses tersebut dilakukan, yaitu menghasilkan gambar dengan kontras yang kecil. Sebaliknya, menggunakan histogram citra lain dalam proses ini malah menaikkan nilai kontras yang dimiliki citra tersebut dari awal.

BAB III

LAMPIRAN

1. Kode Program *Histogram Equalization*

```
48      public static void histogramEqualization(BufferedImage img, int n) {
49          File file;
50          int red, green, blue, alpha, pixel;
51          ArrayList<int[]> histLookup = histogramLookup(img);
52
53          int width = img.getWidth();
54          int height = img.getHeight();
55
56          for (int y = 0; y < height; y++) {
57              for (int x = 0; x < width; x++) {
58                  pixel = img.getRGB(x, y);
59                  alpha = (pixel >> 24)&0xff;
60                  red = (pixel >> 16)&0xff;
61                  green = (pixel >> 8)&0xff;
62                  blue = pixel&0xff;
63
64                  red = histLookup.get(0)[red];
65                  green = histLookup.get(1)[green];
66                  blue = histLookup.get(2)[blue];
67
68                  pixel = (alpha << 24) | (red << 16) | (green << 8) | blue;
69                  img.setRGB(x, y, pixel);
70              }
71          }
72
73          try {
74              file = new File("D:\\Kuliah\\Semester 5\\Tugas\\"
75                      + "Pengolahan Citra Digital (2)\\ImageEqualization\\"
76                      + "output" + n + ".jpg");
77              ImageIO.write(img, "jpg", file);
78          } catch(IOException e) {
79              System.out.println(e);
80          }
81      }
```

2. Kode Program *Histogram Matching*

```
83      public static void swapHistogram(
84              BufferedImage img1, BufferedImage img2, int n, int m) {
85          //img1 sbg image yang akan diubah histogramnya
86          //img2 sbg image yang akan digunakan histogramnya
87          File file;
88          int red, green, blue, alpha, pixel;
89          ArrayList<int[]> histLookup = histogramLookup(img2);
90
91          for (int y = 0; y < img1.getHeight(); y++) {
92              for (int x = 0; x < img1.getWidth(); x++) {
93                  pixel = img1.getRGB(x, y);
94                  alpha = (pixel >> 24)&0xff;
95                  red = (pixel >> 16)&0xff;
96                  green = (pixel >> 8)&0xff;
97                  blue = pixel&0xff;
98
99                  red = histLookup.get(0)[red];
100                 green = histLookup.get(1)[green];
101                 blue = histLookup.get(2)[blue];
102
103                 pixel = (alpha << 24) | (red << 16) | (green << 8) | blue;
104                 img1.setRGB(x, y, pixel);
105             }
106         }
107
108         try {
109             file = new File("D:\\Kuliah\\Semester 5\\Tugas\\"
110                     + "Pengolahan Citra Digital (2)\\ImageEqualization\\"
111                     + "outputSwap" + n + m + ".jpg");
112             ImageIO.write(img1, "jpg", file);
113         } catch(IOException e) {
114             System.out.println(e);
115         }
116     }
```

3. Kode Fungsi Mencari Lookup Table

```
118     public static ArrayList<int[]> histogramLookup(BufferedImage img) {
119         ArrayList<int[]> arrayResult = new ArrayList<>();
120         ArrayList<int[]> histogramOri = histogramOfOriImg(img);
121 
122         int[] redHistogram = new int[256];
123         int[] greenHistogram = new int[256];
124         int[] blueHistogram = new int[256];
125 
126         for (int i = 0; i < 256; i++) {
127             redHistogram[i] = 0;
128             greenHistogram[i] = 0;
129             blueHistogram[i] = 0;
130         }
131 
132         long sumRed = 0, sumGreen = 0, sumBlue = 0;
133 
134         float scaleFactor = (float)(255.0 / (img.getWidth() * img.getHeight()));
135         for (int i = 0; i < 256; i++) {
136             sumRed += histogramOri.get(0)[i];
137             int valRed = (int) (sumRed * scaleFactor);
138             if (valRed > 255) {
139                 redHistogram[i] = 255;
140             } else {
141                 redHistogram[i] = valRed;
142             }
143 
144             sumGreen += histogramOri.get(1)[i];
145             int valGreen = (int) (sumGreen * scaleFactor);
146             if (valGreen > 255) {
147                 greenHistogram[i] = 255;
148             } else {
149                 greenHistogram[i] = valGreen;
150             }
151 
152             sumBlue += histogramOri.get(2)[i];
153             int valBlue = (int) (sumBlue * scaleFactor);
154             if (valBlue > 255) {
155                 blueHistogram[i] = 255;
156             } else {
157                 blueHistogram[i] = valBlue;
158             }
159         }
160 
161         arrayResult.add(redHistogram);
162         arrayResult.add(greenHistogram);
163         arrayResult.add(blueHistogram);
164 
165         return arrayResult;
166     }
```

4. Kode Fungsi Menghitung Frekuensi Intensitas

```
168     public static ArrayList<int[]> histogramOfOriImg(BufferedImage img) {
169         ArrayList<int[]> arrayResult = new ArrayList<>();
170 
171         int[] redFreq = new int[256];
172         int[] greenFreq = new int[256];
173         int[] blueFreq = new int[256];
174         for (int i = 0; i < 256; i++) {
175             redFreq[i] = 0;
176             greenFreq[i] = 0;
177             blueFreq[i] = 0;
178         }
179 
180         int pixel, alpha, red, green, blue;
181         for (int y = 0; y < img.getHeight(); y++) {
182             for (int x = 0; x < img.getWidth(); x++) {
183                 pixel = img.getRGB(x, y);
184                 alpha = (pixel >> 24)&0xff;
185                 red = (pixel >> 16)&0xff;
186                 green = (pixel >> 8)&0xff;
187                 blue = pixel&0xff;
188 
189                 redFreq[red]++;
190                 greenFreq[green]++;
191                 blueFreq[blue]++;
192             }
193         }
194 
195         arrayResult.add(redFreq);
196         arrayResult.add(greenFreq);
197         arrayResult.add(blueFreq);
198 
199         return arrayResult;
200     }
201 }
```