BAB 5 IMPLEMENTASI

Pada Bab 5, dibahas mengenai implementasi berdasarkan perancangan yang telah disusun pada Bab Analisis dan Perancangan. Pada Bab ini terdiri dari penjelasan lingkungan implementasi, batasan implementasi dan implementasi aplikasi. Lingkungan implementasi berisi tentang spesifikasi kebutuhan yang digunakan dalam penelitian. Lingkungan implementasi tersebut terdiri dari lingkungan *hardware* dan lingkungan *software*. Batasan implementasi menjelaskan mengenai batas-batas yang diterapkan pada penelitian. Sedangkan implementasi aplikasi menjabarkan mengenai implementasi *code* yang terdiri dari *Pre-processing*, GLCM dan *manhattan distance*.

5.1 Lingkungan Implementasi

5.1.1 Lingkungan *Hardware*

Berikut ini merupakan lingkungan *hardware* yang digunakan dalam pembuatan sistem temu kembali citra lubang jalan aspal berdasarkan tingkat kerusakan:

1. *Processor*: Intel® Core™ i5-7200U CPU @ 2.50GHz 2.71
2. *Memory:* 4 GB
3. VGA: NVIDIA GEFORCE 930MX
4. *Harddisk* 500 GB
5. Monitor 14 *inch*
6. *Keyboard*
7. *Mouse*

5.1.2 Lingkungan *Software*

Berikut ini merupakan lingkungan *hardware* yang digunakan dalam pembuatan sistem temu kembali citra lubang jalan aspal berdasarkan tingkat kerusakan:

1. *Operating System* Microsoft Windows 10 Pro
2. Microsoft Office Word 2010 yang digunakan untuk membuat laporan penelitian
3. Microsoft Office Excel 2010 yang digunakan untuk membuat perhitungan manual dari metode, menyimpan data hasil GLCM dan pengumpulan data hasil pengujian
4. Microsoft Office PowerPoint 2010 yang digunakan untuk membuat presentasi penelitian
5. Notepad digunakan untuk menyimpan hasil GLCM sebagai data yang digunakan untuk perhitungan kemiripan dengan *manhattan distance*
6. WEKA 3.8 digunakan untuk penyeleksian fitur GLCM

5.2 Batasan Implementasi

Berikut merupakan batasan dalam pengimplementasian sistem temu kembali citra lubang jalan aspal:

1. Jumlah data yang digunakan pada penelitian yaitu basis data sebanyak 100 data yang terbagi menjadi 3 kelas yaitu 33 data rusak ringan (L), 42 data rusak sedang (M) dan 25 data rusak berat (H). Sedangkan data uji sebanyak 17 data.
2. Dataset berupa citra dengan berbagai tingkat kerusakan lubang jalan aspal diperoleh dari observasi langsung pada jalan raya yang terdapat di wilayah Kediri dan Malang antara pukul 09.00 WIB hingga 12.00 dan 15.00 WIB hingga 18.00 WIB.
3. Metode *pre-processing* yang digunakan yaitu *grayscaling*, *Bilateral Filtering,* Deteksi tepi sobel, *Thresholding*, *Closing* dan *Erosion*.
4. Metode ekstraksi fitur yang digunakan yaitu GLCM. Jumlah fitur ekstraksi keseluruhan sebanyak 52 fitur.
5. Seleksi fitur dilakukan menggunakan WEKA dengan metode CFS dan *Wrapper*.
6. Ukuran citra yang digunakan yaitu 3264 x 2320, kemudian dilakukan *resize* menjadi 816 x 580. Citra hasil *resize* tersebut yang digunakan hingga proses ekstraksi fitur tekstur GLCM.
7. Metode pelatihan data yang digunakan yaitu *manhattan distance*.

5.3 Implementasi Aplikasi

Pada sub bab ini, dijelaskan mengenai implementasi *code* berdasarkan perancangan pada BAB 4. Pada sub bab ini terbagi menjadi 3 sub-sub bab yaitu *pre-processing*, ekstraksi fitur GLCM, dan *manhattan distance*. Pada sub sub bab *pre-processing* terbagi menjadi subsubsub bab *grayscaling*, *Bilateral Filtering,* Deteksi tepi sobel, *Thresholding*, *Closing* dan *Erosion*.

5.3.1 *Pre-Processing*

5.3.1.1 Implementasi *Grayscaling*

Langkah awal dalam pengimplementasian sistem temu kembali citra lubang jalan aspal yaitu mengubah citra berwarna (RGB) menjadi citra keabuan. Implementasi *grayscaling* digunakan sebagai proses segmentasi lubang jalan dan sebagai masukan perhitungan ekstraksi fitur GLCM. Implementasi *code grayscaling* disimpan kedalam fungsi grayscaling() pada *Sourcecode* 5.1.

```
1  def grayscaling (rgbimg):
2      t, l = rgbimg.size
3      grayimg = np.zeros((l, t))
```

```
4       for y in range (t):
5           for x in range (l):
6               Red, Green, Blue = rgbimg.getpixel((y, x))
7               gray=float(Red + Green+ Blue)/3
8               grayimg[x,y]= gray
9       return grayimg
```

***Sourcecode* 0.1 Implementasi fungsi grayscaling()**

Berikut merupakan penjelasan *Sourcecode* 5.1 :

1. Baris 1 merupakan deklarasi fungsi grayscaling() dengan parameter masukan berupa citra RGB bertipe data Image
2. Baris 2 digunakan untuk mendapatkan nilai panjang (t) dan lebar (l) dari citra RGB
3. Baris 3 digunakan untuk inisialisasi array bernama grayimg yang berisi 0 dengan panjang dan lebar yang sudah didapatkan dari citra RGB
4. Baris 4-8 digunakan untuk melakukan proses perhitungan nilai keabuan (gray)tiap *pixel* yang diperoleh dari jumlah nilai Red, Green dan Blue kemudian dibagi 3.
5. Baris 9 digunakan untuk mengembalikan nilai variabel grayimg dengan tipe data float64

5.3.1.2 Implementasi *Bilateral Filter*

```
1   def bilateralFilter(img, diameter, sigmaR, sigmaS):
2       uPadding=padding(img, diameter)
3       t, l = img.shape
4       tGray=t+(diameter-1)
5       lGray=l+(diameter-1)
6       mid=diameter/2
7       bilateral = np.zeros((t,l))
8       for y in range (mid,tGray-mid):
9           for x in range (mid,lGray-mid):
10              totFilter=0
11              Wp=0
12              for b in range (-mid,mid+1):
13                  for k in range (-mid,mid+1):
14                      Gi=  float(gaussian(uPadding[y+b,x+k]  -
    uPadding[y,x], sigmaR))
15                      Gs=  float(gaussian(jarak(b,  k,  y,  x),
    sigmaS))
16                      w = Gi * Gs
```

```
17                          totFilter += uPadding[y+b,x+k]*w
18                          Wp += w
19                  totFilter = totFilter / float(Wp)
20                  bilateral[y-mid,x-mid]= int(totFilter)
21  return bilateral
```

***Sourcecode* 0.2 Implementasi fungsi bilateralFilter()**

Berikut merupakan penjelasan Sourcecode 5.2 :

1. Baris 1 merupakan deklarasi fungsi bilateralfilter() dengan parameter masukan berupa citra *grayscale* bertipe data float64, diameter bertipe data *integer*, sigmaR bertipe data *integer* dan sigmaS beripe data *integer*
2. Baris 2 digunakan untuk melakukan proses penambahan *padding*
3. Baris 3 digunakan untuk mendapatkan nilai panjang (t) dan lebar (l) dari citra *grayscaling*
4. Baris 4-5 digunakan inisialisasi tGray dan lGray yang digunakan sebagai batas perhitungan *pixel* pada citra yang telah dilakukan penambahan *padding*
5. Baris 6 digunakan untuk inisialisasi nilai mid yang digunakan untuk *kernel*
6. Baris 7 digunakan untuk inisialisasi array bernama bilateral yang berisi 0 dengan panjang dan lebar yang sudah didapatkan dari citra *grayscaling*
7. Baris 10-11 digunakan untuk inisialisasi nilai totFilter dan Wp
8. Baris 12-19 merupakan proses perhitungan *bilateral filter* tiap *pixel* dari citra yang telah ditambahkan *padding*
9. Baris 20 digunakan untuk menyimpan hasil perhitungan bilateral filter kedalam array list bilateral
10. Baris 9 digunakan untuk mengembalikan nilai variabel bilateral dengan tipe data float64

Implementasi perhitungan jarak ditunjukkan pada Gambar 5.3 .

```
1  def jarak(x, y, i, j):
2      return np.sqrt((x-i)**2 + (y-j)**2)
```

***Sourcecode* 0.3 Implementasi Perhitungan Jarak**

Berikut merupakan penjelasan Sourcecode 5.3 :

1. Baris 1 merupakan deklarasi fungsi jarak() dengan parameter masukan x, y, I, j beripe data *integer*
2. Baris 2 digunakan untuk mengembalikan nilai hasil dari perhitungan jarak

Implementasi perhitungan *gaussian* ditunjukkan pada Gambar 5.4 .

```
1  def gaussian(x, sigma):
2      return (1.0 / (2 * math.pi * (sigma ** 2))) * math.exp(-
(x ** 2) / (2 * sigma ** 2))
```

Sourcecode* 0.4 Implementasi Perhitungan *Gaussian

Berikut merupakan penjelasan Sourcecode 5.4 :

1. Baris 1 merupakan deklarasi fungsi gaussian() dengan parameter masukan x dan sigma beripe data *integer*
2. Baris 2 digunakan untuk mengembalikan nilai hasil dari perhitungan *gaussian*

5.3.1.3 Implementasi *Deteksi Tepi Sobel*

Dari hasil filter bilateral yang didapatkan, kemudian dilakukan pendeteksian tepi menggunakan sobel. Metode deteksi tepi sobel dilakukan untuk memperjelas tepi area lubang jalan. Implementasi *code* deteksi tepi sobel terbagi menjadi 2 bagian fungsi yaitu fungsi padding dan perhitungan sobel. Bagian implementasi yang pertama yaitu padding yang digunakan untuk menambahkan padding dari hasil citra keabuan. Fungsi padding() ditunjukkan pada *Sourcecode* 5.5.

```
1  def padding (img, ukuranMask):
2    tImg, lImg = img.shape
3    tTot=tImg+(ukuranMask-1)
4    lTot=lImg+(ukuranMask-1)
5    batasBaris1= (ukuranMask-1)/2
6    batasBaris2= tImg + batasBaris1 - 1
7    batasKolom1= (ukuranMask-1)/2
8    batasKolom2= lImg + batasKolom1 - 1
9    newimg = np.zeros((tTot,lTot))
10   for baris in range (0,tTot):
11    for kolom in range (0,lTot):
12     if baris < batasBaris1 or baris > batasBaris2 or kolom <
batasKolom1 or kolom > batasKolom2 :
13       if baris <= batasBaris1 and kolom <=batasKolom1:
#pojok kiri atas
14        newimg[baris,kolom]=img[0,0]
15       elif baris >= batasBaris2 and kolom <=batasKolom1:
#pojok kiri bawah
16         newimg[baris,kolom]=img[tImg-1,0]
17       elif baris >= batasBaris2 and kolom >=batasKolom2:
#pojok kanan bawah
18         newimg[baris,kolom]=img[tImg-1,lImg-1]
19       elif baris <= batasBaris1 and kolom >=batasKolom2:
```

```
    #pojok kanan atas
20          newimg[baris,kolom]=img[0,lImg-1]
21        elif baris > batasBaris1 and baris <batasBaris2 and
    kolom <batasKolom1:  #tepi kiri
22          newimg[baris,kolom]=img[baris-batasBaris1,0]
        elif baris > batasBaris1 and baris <batasBaris2 and
23  kolom >batasKolom2:  #tepi kanan
24          newimg[baris,kolom]=img[baris-batasBaris1,lImg-1]
25        elif kolom > batasKolom1 and kolom <batasKolom2 and
    baris <batasBaris1:  #tepi atas
26          newimg[baris,kolom]=img[0,kolom-batasKolom1]
27        elif kolom > batasKolom1 and kolom <batasKolom2 and
    baris >batasBaris1:  #tepi bawah
28          newimg[baris,kolom]=img[tImg-1,kolom-batasKolom1]
29        else:
30          newimg[baris,kolom]=img[baris-batasBaris1,kolom-
    batasKolom1]
31      return newimg
```

***Sourcecode* 0.5 Implementasi fungsi padding()**

Berikut merupakan penjelasan dari *Sourcecode* 5.5:

1. Baris 1 merupakakan deklarasi fungsi padding () dengan parameter img bertipe data float64 dan ukuranMask bertipe data int untuk menentukan penambahan padding pada citra
2. Baris 2 digunakan untuk mendapatkan nilai panjang (tImg) dan lebar (lImg) dari citra img
3. Baris 3-4 digunakan untuk mendapatkan ukuran panjang (tTot) dan lebar (lTot) setelah penambahan padding
4. Baris 5-8 digunakan untuk menginisialisasi batas penambahan padding
5. Baris 9 digunakan untuk inisialisasi array bernama newimg yang berisi 0 dengan panjang dan lebar yang sudah didapatkan dari citra img
6. Baris 10-30 merupakan proses padding terhadap citra img yang disimpan kedalam variabel newimg
7. Baris 31 digunakan untuk mengembalikan variabel newimg dengan tipe data float64

Implementasi bagian yang kedua yaitu *code* proses perhitungan deteksi tepi sobel dengan nama fungsi perhitunganSobel() yang ditunjukkan pada *Source*code 5.6.

```
1   def sobel (img, ukuranMask):
2       t, l = img.shape
```

```
    # kernel
3   horizontal = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  # s2
4   vertical = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])  # s1
5   uPadding=padding(img, ukuranMask)
6   tMed=t+(ukuranMask-1)
7   lMed=l+(ukuranMask-1)
8   sobelimg = np.zeros((t,l))
9   for i in range(1, tMed - 1):
10    for j in range(1, lMed - 1):
11      horizontalGrad = (horizontal[0, 0] * uPadding[i - 1, j - 1]) + \
12        (horizontal[0, 1] * uPadding[i - 1, j]) + \
          (horizontal[0, 2] * uPadding[i - 1, j + 1]) + \
13        (horizontal[1, 0] * uPadding[i, j - 1]) + \
14        horizontal[1, 1] * uPadding[i, j]) + \
15        (horizontal[1, 2] * uPadding[i, j + 1]) + \
16        (horizontal[2, 0] * uPadding[i + 1, j - 1]) + \
17        (horizontal[2, 1] * uPadding[i + 1, j]) + \
18        (horizontal[2, 2] * uPadding[i + 1, j + 1])
19      verticalGrad = (vertical[0, 0] * uPadding[i - 1, j - 1]) + \
20        (vertical[0, 1] * uPadding[i - 1, j]) + \
21        (vertical[0, 2] * uPadding[i - 1, j + 1]) + \
22        (vertical[1, 0] * uPadding[i, j - 1]) + \
23        (vertical[1, 1] * uPadding[i, j]) + \
24        (vertical[1, 2] * uPadding[i, j + 1]) + \
25        (vertical[2, 0] * uPadding[i + 1, j - 1]) + \
26        (vertical[2, 1] * uPadding[i + 1, j]) + \
27        (vertical[2, 2] * uPadding[i + 1, j + 1])
            # Edge Magnitude
28          mag = np.sqrt(pow(horizontalGrad, 2.0) + pow(verticalGrad, 2.0))
29          sobelimg[i - 1, j - 1] = mag
30  maks=0
31  for i in range(t):
32      for j in range(l):
33          if sobelimg[i,j] > maks:
34              maks=sobelimg[i,j]
```

```
35      sobelimg *= 255.0/maks
36      return sobelimg
```

***Sourcecode* 0.6 Implementas fungsi perhitunganSobel()**

Berikut merupakan penjelasan dari *Sourcecode* 5.6:

1. Baris 1 merupakan deklarasi fungsi perhitunganSobel() dengan parameter img bertipe data float64 dan ukuranMask bertipe data int untuk menentukan penambahan padding pada citra. ukuranMask digunakan ketika pemanggilan fungsi padding
2. Baris 2 digunakan untuk mendapatkan nilai panjang (t) dan lebar (l) dari citra img
3. Baris 3-4 digunakan untuk menginisialisasi nilai dari *kernel* sobel yang disimpan kedalam variabel horizontal (Gx)dan vertical (Gy)
4. Baris 5 digunakan untuk memanggil fungsi padding yang diterapkan pada citra img
5. Baris 6-7 digunakan untuk inisialisasi panjang (tMed) dan lebar (lMed) proses perhitungan sobel
6. Baris 8 digunakan untuk inisialisasi array bernama sobelimg yang berisi 0 dengan panjang dan lebar yang sudah didapatkan dari citra img
7. Baris 9-19 merupakan proses perhitungan operator sobel Gx
8. Baris 20-27 merupakan proses perhitungan operator sobel Gy
9. Baris 28 merupakan perhitungan besaran tepi sobel
10. Baris 29 digunakan untuk menyimpan nilai besaran tepi kedalam variabel sobelimg bertipe data float64
11. Baris 30-35 digunakan untuk mengubah variabel sobelimg bertipe data float64 menjadi uint8
12. Baris 36 digunakan untuk mengembalikan nilai variabel sobelimg dengan tipe data uint8

5.3.1.4 Implementasi *Thresholding*

Langkah selanjutnya, hasil dari citra yang telah dilakukan deteksi tepi sobel dilakukan *thresholding*. Dalam pengimplementasian *code* nilai ambang batas masing-masing citra didapatkan dari meminimalkan bobot *variance* dalam suatu kelas. *Thresholding* dilakukan untuk mempertegas hasil dari citra deteksi tepi sobel. Implementasi *code thresholding* terbagi menjadi 2 bagian, yaitu penentuan nilai ambang batas dengan nama fungsi nilaiThresh() dan proses *thresholding* dengan nama fungsi thresholding(). Implementasi bagian pertama dengan nama fungsi nilaiThresh() ditunjukkan pada *Sourcecode* 5.7.

```
1  def nilaiThresh(sobelimg):
2      p,l=sobelimg.shape
3      sobelimg = np.uint8(sobelimg)
4      hist = cv2.calcHist([sobelimg],[0],None,[256],[0,256])
5      hist_norm = hist.ravel()/hist.max()
6      Q = hist_norm.cumsum()
7      bins = np.arange(256)
8      fn_min = np.inf
9      thresh = -1
10     for i in xrange(1,256):
11         p1,p2 = np.hsplit(hist_norm,[i])
12         q1,q2 = Q[i],Q[255]-Q[i]
13         b1,b2 = np.hsplit(bins,[i])
           # finding means and variances
14         m1,m2 = np.sum(p1*b1)/q1, np.sum(p2*b2)/q2
15         v1,v2    =    np.sum(((b1-m1)**2)*p1)/q1,np.sum(((b2-
m2)**2)*p2)/q2
           # calculates the minimization function
16         fn = v1*q1 + v2*q2
17         if fn < fn_min:
18             fn_min = fn
19             thresh = i
20     return thresh
```

***Sourcecode* 0.7 Implementasi fungsi nilaiThresh()**

Berikut merupakan penjelasan dari *Sourcecode* 5.7:

1. Baris 1 merupakan deklarasi fungsi nilaiThresh() dengan parameter sobelimg bertipe data float64
2. Baris 2 digunakan untuk mendapatkan nilai panjang (t) dan lebar (l) dari citra sobelimg
3. Baris 3 digunakan untuk mengubah tipe data sobelimg dari floar64 menjadi uint8
4. Baris 4-9 merupakan proses perhitungan normalisasi histogram dan juga fungsi distribusi kumulatif
5. Baris 11 merupakan proses perhitungan probabilitas
6. Baris 12 merupakan proses menghitung jumlah kumulatf
7. Baris 13 merupakan proses untuk menghitung bobot
8. Baris 14-14 merupakan proses menghitung *mean* dan *variance*

9. Baris 16-19 merupakan proses menghitung fungsi minimisasi untuk mendapatkan nilai ambang
10. Baris 20 digunakan untuk mengembalikan nilai ambang batas dengan tipe data int

Implementasi bagian yang kedua yaitu code proses *tresholding* dengan nama fungsi thresholding() dengan nilai ambang batas yang telah didapatkan dari fungsi nilaiThresh(). Fungsi thresholding() ditunjukkan pada Sourcecode 5.8.

```
1  def thresholding (img, threshold):
2      t, l = img.shape
3      thresholdimg=np.zeros((t,l))
4      for b in range (t):
5          for k in range (l):
6              if img[b,k] >= threshold:
7                  thresholdimg[b,k] = 1
8              else:
9                  thresholdimg[b,k] = 0
10     return thresholdimg
```

***Sourcecode* 0.8 Implementasi fungsi thresholding()**

Berikut merupakan penjelasan dari *Sourcecode* 5.8:

1. Baris 1 merupakan deklarasi fungsi thresholding() dengan parameter img bertipe data float64 dan threshold bertipe data int
2. Baris 2 digunakan untuk mendapatkan nilai panjang (t) dan lebar (l) dari citra img
3. Baris 3 digunakan untuk inisialisasi array bernama thrsholdimg yang berisi 0 dengan panjang dan lebar yang sudah didapatkan dari citra img
4. Baris 4-9 merupakan proses *thresholding* pada citra img dengan nilai ambang parameter threshold yang didapatkan dari hasil proses fungsi nilaiThresh()
5. Baris 37 digunakan untuk mengembalikan nilai variabel thresholdimg dengan tipe data float64

5.3.1.5 Implementasi *Closing*

Metode *closing* diterapkan pada citra hasil *thresholding* guna melebarkan *pixel* bernilai 1 yang berada di area lubang jalan dan menghilangkan lubang-lubang kecil di area selain lubang jalan. Implementasi *code* opening terbagi menjadi 2 bagian yaitu *dilate* yang merupakan proses pertama dari operasi *closing*, *erosion* yang merupakan proses kedua dari operasi *closing*. Implementasi code bagian pertama yaitu fungsi dilate() yang ditunjukkan pada *Sourcecode* 5.9.

```
1  def dilate (img, ukuran_Kernel):
2      _kernel_ = np.ones((ukuran_Kernel_,ukuran_Kernel_))
```

```
3       tkernel, lkernel = kernel.shape
4       xkernel = np.round(lkernel/2)
5       ykernel = np.round(tkernel/2)
6       arrayXk=[]
7       arrayYk=[]
8       jumlahNilai1=0
9       for b in range (tkernel):
10          for k in range (lkernel):
11              if kernel[b,k]==1:
12                  jumlahNilai1=jumlahNilai1 + 1
13                  arrayXk.append(-xkernel + k)
14                  arrayYk.append(-ykernel + b)
15      t, l = img.shape
16      dilateimg = np.zeros((t,l))
17      for b in range (t):
18          for k in range (l):
19              for i in range (jumlahNilai1):
20                  if img[b,k] == 1:
21                      xpos= k + arrayXk[i]
22                      ypos= b + arrayYk[i]
23                      if (xpos >= 0) & (xpos <= l-1) & \
24                         (ypos >= 0) & (ypos <= t-1) :
25                             dilateimg[ypos, xpos] = 1
26      return dilateimg
```

***Sourcecode* 0.9 Implementasi fungsi dilate()**

Berikut merupakan penjelasan dari *Sourcecode* 5.9:

1. Baris 1 merupakan deklarasi fungsi dilate() dengan parameter img bertipe data float64 dan ukuran*Kernel* bertipe data int
2. Baris 2 digunakan untuk inisialisasi array bernama *kernel* yang berisi 1 dengan ukuran panjang dan lebar sebesar parameter ukuran*Kernel*
3. Baris 3 digunakan untuk mendapatkan nilai panjang (t*kernel*) dan lebar (l*kernel*) dari variabel *kernel*
4. Baris 4-5 digunakan untuk menentukan nilai panjang dan lebar *kernel* ketika titik 0,0 berada di tengah
5. Baris 6-7 digunakan untuk deklarasi array kosong dengan nama arrayXk dan arrayYk yang digunakan untuk menyimpan posisi *kernel* yang bernilai 1 ketika titik 0,0 berada di tengah
6. Baris 9-14 merupakan proses peletakan posisi isi *kernel* yang bernilai 1

7. Baris 15 digunakan untuk mendapatkan nilai panjang (t) dan lebar (l) dari citra img
8. Baris 16 digunakan untuk inisialisasi array bernama dilateimg yang berisi 1 dengan ukuran panjang dan lebar yang sudah didapatkan dari citra img
9. Aris 17-25 merupakan proses *dilation* pada citra img yang hasilnya disimpan kedalam array dilateimg
10. Baris 36 digunakan untuk mengembalikan nilai variabel dilateimg dengan tipe data float64

Implementasi bagian yang kedua yaitu *code* proses *erosion* dengan nama fungsi erosion(). Fungsi erosion() ditunjukkan pada Sourcecode 5.10.

```
1  def erosion (img, ukuranKernel):
2      kernel = np.ones((ukuranKernel,ukuranKernel))
3      tkernel, lkernel = kernel.shape
4      xkernel = np.round(lkernel/2)
5      ykernel = np.round(tkernel/2)
6      arrayXk=[]
7      arrayYk=[]
8      jumlahNilai1=0
9      for b in range (tkernel):
10         for k in range (lkernel):
11             if kernel[b,k]==1:
12                 jumlahNilai1=jumlahNilai1 + 1
13                 arrayXk.append(-xkernel + k)
14                 arrayYk.append(-ykernel + b)
15     t, l = img.shape
16     erosionimg = np.zeros((t,l))
17     for b in range (t):
18         for k in range (l):
19             cocok = 1
20             for i in range (jumlahNilai1):
21                 xpos= k + arrayXk[i]
22                 ypos= b + arrayYk[i]
23                 if (xpos >= 0) & (xpos <= l-1) & \
24                    (ypos >= 0) & (ypos <= t-1) :
25                     if img[ypos, xpos] != 1:
26                         cocok = 0
27                         break
```

```
28                  else:
29                      cocok= 0
30              if cocok==1:
31                  erosionimg[b,k] =1
32      return erosionimg
```

***Sourcecode* 0.10 Implementasi fungsi erosion()**

Berikut merupakan penjelasan dari *Sourcecode* 5.10:

11. Baris 1 merupakan deklarasi fungsi erosion() dengan parameter img bertipe data float64 dan ukuran*Kernel* bertipe data int
12. Baris 2 digunakan untuk inisialisasi array bernama *kernel* yang berisi 1 dengan ukuran panjang dan lebar sebesar parameter ukuran*Kernel*
13. Baris 3 digunakan untuk mendapatkan nilai panjang (t*kernel*) dan lebar (l*kernel*) dari variabel *kernel*
14. Baris 4-5 digunakan untuk menentukan nilai panjang dan lebar *kernel* ketika titik 0,0 berada di tengah
15. Baris 6-7 digunakan untuk deklarasi array kosong dengan nama arrayXk dan arrayYk yang digunakan untuk menyimpan posisi *kernel* yang bernilai 1 ketika titik 0,0 berada di tengah
16. Baris 9-14 merupakan proses peletakan posisi isi *kernel* yang bernilai 1
17. Baris 15 digunakan untuk mendapatkan nilai panjang (t) dan lebar (l) dari citra img
18. Baris 16 digunakan untuk inisialisasi array bernama erosionimg yang berisi 1 dengan ukuran panjang dan lebar yang sudah didapatkan dari citra img
19. BAris 17-31 merupakan proses *erosion* pada citra img yang hasilnya disimpan kedalam array erosionimg
20. Baris 36 digunakan untuk mengembalikan nilai variabel erosionimg dengan tipe data float64

5.3.1.6 Implementasi *Erosion*

Operasi *erosion* diterapkan pada citra hasil operasi *closing*. Operasi *erosion* digunakan untuk menghilangkan lubang-lubang kecil yang masih ada setelah dilakukannya operasi *closing*. Implementasi *code* operasi *closing* ditunjukkan pada *Sourcecode* 5.10 yang telah tertera pada sub sub sub bab *Closing* dengan nama fungsi erosion().

Setelah didapatkan objek lubang jalan tanpa adanya *noise* berupa lubang-lubang disekitarnya, dilakukan proses pengubahan nilai citra biner pada bagian lubang menjadi citra nilai keabuan. Proses ini dilakukan untuk memisahkan antara objek lubang jalan dengan daerah sekitarnya atau jalan yang tidak

mengalami kerusakan.Implementasi *code* hasil *pre-processing* disimpan ke dalam fungsi hasilPrepocessing() yang ditunjukkan pada *Sourcecode* 5.11.

```
1  def hasilPreprocessing (img1, img2, nilai):
2      t, l = img1.shape
3      segmenimg = Image.new('L', (l,t))
4      pixel = segmenimg.load()
5      con = Image.fromarray(img2)
6      for b in range (l):
7          for k in range (t):
8              if img1[k,b] == nilai:
9                  gray = con.getpixel((b, k))
10                 pixel[b,k] = int(gray)
11             else:
12                 pixel[b,k] = 0
13     return segmenimg
```

***Sourcecode* 0.11 Implementasi fungsi hasilPreprocessing()**

Berikut merupakan penjelasan dari *Sourcecode* 5.11:

1. Baris 1 merupakan deklarasi fungsi hasilPreprocessing() dengan parameter img1 bertipe data float64, img2 bertipe data float64 dan nilai bertipe data int. Parameter img1 merupakan citra dari hasil *erosion* dan img2 merupakan citra keabuan. Sedangkan paremeter nilai merupakan nilai dari area citra lubang jalan
2. Baris 2 digunakan untuk mendapatkan nilai panjang (t) dan lebar (l) dari citra img1
3. Baris 3 digunakan untuk deklarasi Image baru bernama segmenimg yang berwarna keabuan dengan panjang dan lebar sebesar ukuran img1
4. Baris 5 digunakan untuk mengubah citra array img2 bertipe data float64 menjadi Image
5. Baris 6-12 merupakan proses segmentasi citra lubang jalan yang disimpan kedalam variabel Image segmenimg
6. Baris 36 digunakan untuk mengembalikan nilai variabel segmentasi dengan tipe data Image

5.3.2 Implementasi GLCM

Proses ekstraksi fitur GLCM dilakukan pada hasil citra lubang jalan yang tersegmentasi dengan warna keabuan untuk mendapatkan nilai tekstur dari lubang jalan. Proses penentuan awal matriks GLCM menggunakan 4 sudut yaitu 0^o, 45^o, 90^o dan 135^o. Jarak antar *pixel* tetangga yang digunakan yaitu 1 *pixel*. Fitur ekstraksi GLCM yang digunakan sebanyak 13 yaitu *Angular Second Moment*

(ASM), *Contrast*, *Correlation, Sum of Squares: Variance, Inverse Difference Moment* (IDM), *Sum Average* (AVER)*, Sum Entropy* (SENT)*, Sum Variance* (SVAR)*, Entropy, Difference Entropy* (DENT)*, Difference Variance* (DVAR)*, Information Measure of Correlation 1* dan *Information Measure of Correlation 2*. Implementasi *code* ekstraksi fitur GLCM terdapat 2 fungsi yaitu fungsi untuk menentukan nilai awal masing-masing matriks GLCM berdasarkan sudut hingga menjadi matriks GLCM ternormalisasi dan proses perhitungan 13 fitur GLCM.

5.3.2.1 Implementasi Penentuan Matriks Awal GLCM

Fungsi bagian yang pertama yaitu matriksglcm() yang ditunjukan pada *Sourcecode* 5.12.

```
1  def matriksglcm(img, sudut, d):
2      glcmimg= np.array(img).astype(np.uint8)
3      t, l = glcmimg.shape
4      G=256
5      GLCM = np.zeros((G, G))
       #PENENTUAN AWAL MATRIKS GLCM
6      if sudut == 0:
7          for b in range (t-d):
8              for k in range (l-d):
9                  if (glcmimg[b,k] != 0) and (glcmimg[b,k+d]
   != 0):
11                     t1= glcmimg[b,k]
12                     t2= glcmimg[b,k+d]
13                     GLCM[t1,t2] += 1
14     elif sudut == 45:
15         for b in range (t-d):
16             for k in range (l-d):
17                 if (glcmimg[b,k] != 0) and (glcmimg[b-1,k+d]
   != 0):
18                     t1= glcmimg[b,k]
19                     t2= glcmimg[b-d,k+d]
20                     GLCM[t1,t2] += 1
21     elif sudut == 90:
22         for b in range (t-d):
23             for k in range (l-d):
24                 if (glcmimg[b,k] != 0) and (glcmimg[b-d,k]
   != 0):
25                     t1= glcmimg[b,k]
26                     t2= glcmimg[b-d,k]
27                     GLCM[t1,t2] += 1
```

```
28      elif sudut == 135:
29          for b in range (t-d):
30              for k in range (l-d):
31                  if (glcmimg[b,k] != 0) and (glcmimg[b-d,k-d]
!= 0):
32                      t1= glcmimg[b,k]
33                      t2= glcmimg[b-d,k-d]
34                      GLCM[t1,t2] += 1
        #HASIL TRANSPOSE DAN MATRIKS SIMETRIS
35      Tranpose = np.zeros((G, G))
36      totalPixel= 0
37      for baris in range (G-1):
38          for kolom in range (G-1):
39              Tranpose[baris,kolom]=GLCM[baris,kolom]+
GLCM[kolom,baris]
41              totalPixel += Tranpose[baris,kolom]
42      normalisasi = np.zeros((G, G))
43      normalisasi =Tranpose/totalPixel
44      return normalisasi
```

***Sourcecode* 0.12 Implementasi fungsi matriksglcm()**

Berikut merupakan penjelasan dari *Sourcecode* 5.12:

1. Baris 1 merupakan deklarasi fungsi matriksglcm() dengan parameter img bertipe data Image dan sudut bertipe data int. Parameter img merupakan citra dari hasil segmentasi dan sudut digunakan untuk menentukan arah perhitungan matriks awal GLCM
2. Baris digunakan untuk mengubah citra img bertipe data Image menjadi array dengan tipe data uint8 dan disimpan ke dalam variabel glcmimg
3. Baris 2 digunakan untuk mendapatkan nilai panjang (t) dan lebar (l) dari citra glcmimg
4. Baris 5 digunakan untuk inisialisasi array bernama GLCM yang berisi 0 dengan panjang dan lebar yang sudah didapatkan dari citra glcmimg
5. Baris 6-13 digunakan untuk melakukan perhitungan penentuan matriks awal GLCM apabila masukan parameter sudut bernilai 0
6. Baris 14-20 digunakan untuk melakukan perhitungan penentuan matriks awal GLCM apabila masukan parameter sudut bernilai 45
7. Baris 21-27 digunakan untuk melakukan perhitungan penentuan matriks awal GLCM apabila masukan parameter sudut bernilai 90
8. Baris 28-34 digunakan untuk melakukan perhitungan penentuan matriks awal GLCM apabila masukan parameter sudut bernilai 135

9. Baris 35 digunakan untuk inisialisasi array bernama Trapose yang berisi 0 dengan panjang dan lebar sebesar G (256)
10. Baris 37-41 merupakan proses penjumlahan antara matriks penentuan awal dengan matriks transpose agar menghasilkan matriks yang simetris. Hasil penjumlahan tersebut disimpan ke dalam variabel Tranpose
11. Baris 42-43 merupakan proses normalisasi matriks dengan melakukan pembagian antara variabel Tranpose dengan totalPixel yang didapatkan dari jumlah keseluruhan nilai *pixel* pada array Tranpose
12. Baris 44 digunakan untuk mengembalikan nilai variabel normalisasi dengan tipe data float64

5.3.2.2 Implementasi Ekstraksi Fitur

Fungsi bagian ketiga yaitu fiturglcm(). Pada fungsi fiturglcm() memberikan hasil perhitungan 13 fitur yang disimpan kedalam array. Salah satu fitur disimpan kedalam fungsi yaitu fungsi entropy() karena digunakan lebih dari satu kali. Implementasi fungsi fitur entropy() ditunjukkan pada *Sourcecode* 5.13.

```
1   def entropy(arr):
        #4. Entropy
2       G=256
3       Entropy=0
4       for i in range (G-1):
5           for j in range (G-1):
6               Entropy += ( (arr[i,j]) * np.log(arr[i,j]+np.finfo(float).eps) )
7       Entropy=Entropy*(-1)
8       return Entropy
```

***Sourcecode* 0.13 Implementasi fungsi entropy()**

Berikut merupakan penjelasan dari *Sourcecode* 5.13:

1. Baris 1 merupakan deklarasi fungsi entropy() dengan parameter arr bertipe data float64. Parameter arr digunakan untuk perhitungan fitur entropy.
2. Baris 2-7 merupakan proses perhitungan fitur *Entropy*
3. Baris 8 digunakan untuk mengembalikan nilai variabel Entropy dengan tipe data float64

Keseluruhan proses perhitungan fitur GLCMter dapat pada fungsi fiturglcm() yang ditunjukkan pada *Sourcecode* 5.14.

```
1   def fiturglcm(arr):
2       fitur = np.zeros(13,np.double)
3       G=256
        #1. Angular Second Moment
4       ASM=0
5       for i in range (G-1):
6           for j in range (G-1):
7               ASM += (arr[i,j]**2)
8       fitur[0]=ASM
        #2. Contrast
9       Contrast=0
10      for i in range (G-1):
11          for j in range (G-1):
12                  Contrast += (((i-j)**2) * arr[i,j])
13      fitur[1]=Contrast
        #3. Correlation
14      uX=0
15      uY=0
16      stdX=0
17      stdY=0
18      Correlation=0
19      for i in range (G-1):
20          for j in range (G-1):
21              uX += i*arr[i,j]
22              uY += j*arr[i,j]
23      for i in range (G-1):
24          for j in range (G-1):
25              stdX += arr[i,j] * ((i-uX)**2)**0.5
26              stdY += arr[i,j] * ((j-uY)**2)**0.5
27      for i in range (G-1):
28          for j in range (G-1):
29             Correlation += ( ((i*j) * arr[i,j]) - (uX*uY) )/
    (stdX*stdY)
30      fitur[2]=Correlation
        #4. Sum of Square, Variance
31      jumlah=0
32      tot=0
33      Variance=0
34      for i in range (G-1):
```

```
35          for j in range (G-1):
36              jumlah += arr[i,j]
37              tot += 1
38      mean=jumlah/tot
39      for i in range (G-1):
40          for j in range (G-1):
41              Variance += ((i-mean)**2) * arr[i,j]
42      fitur[3]=Variance
        #5. Inverse Difference Moment
43      IDM=0
44      for i in range (G-1):
45          for j in range (G-1):
46                  IDM += ((1/(1+((i-j)**2))) * (arr[i,j]))
47      fitur[4]=IDM
        #6. Sum Average
48      AVER=0
49      pXplusY= np.zeros((2*G))
50      for i in range(G-1):
51             for j in range(G-1):
52                 pXplusY[i+j] += arr[i,j]
53      for i in range (2, 2*G):
54          AVER += i*pXplusY[i]
55      fitur[5]=AVER
        #7. Sum Entropy
56      SENT=0
57      for i in range (2, 2*G):
58          SENT                                                    +=
    i*pXplusY[i]*np.log(pXplusY[i]+np.finfo(float).eps)
59      SENT = -1*SENT
60      fitur[6]=SENT
        #8.Sum Variance
61      SVAR=0
62      for i in range(2, 2*G):
63          SVAR += (((i-SENT)**2)*pXplusY[i])
64      fitur[7]=SVAR
        #9. Entropy
65      fitur[8]=entropy(arr)
        #10. Difference Entropy
66      DENT=0
```

```
67      pXminY= np.zeros((G))
68      for i in range(G-1):
69             for j in range(G-1):
70                 pXminY[np.abs(i-j)] += arr[i,j]
71      for i in range (G-1):
72          DENT                                                    +=
i*pXminY[i]*np.log(pXminY[i]+np.finfo(float).eps)
73      DENT = -1*DENT
74      fitur[9]=DENT
        #11. Difference Variance
75      DVAR=0
76      for i in range(G-1):
77          DVAR += ((i**2)*pXminY[i])
78      fitur[10]=DVAR
        #12 dan 13. Information Measure of Correlation
79      pX=np.zeros((G))
80      pY=np.zeros((G))
81      HXY=entropy(arr)
82      HX=0
83      HY=0
84      HXY1=0
85      HXY2=0
86      for i in range(G-1):
87          for j in range(G-1):
88              pX[i] += arr[i,j]
89              pY[j] += arr[i,j]
            #HX dan HY dihitung dari entropy pX dan pY
90      for i in range (G-1):
91          for j in range (G-1):
92              HX        +=        (        (pX[i])        *
np.log(pX[i]+np.finfo(float).eps) )
93              HY        +=        (        (pY[j])        *
np.log(pY[j]+np.finfo(float).eps) )
94      HX=HX*-1
95      HY=HY*-1
96      maks=0
97      if HX > HY:
98          maks = HX
99      else:
```

```
100         maks = HY
101     for i in range (G-1):
102         for j in range (G-1):
103             HXY1        +=        (        (arr[i,j])        *
    np.log((pX[i]*pY[j])+np.finfo(float).eps) )
104             HXY2     +=     (     pX[i]     *     pY[j]     *
    np.log((pX[i]*pY[j])+np.finfo(float).eps) )
105     HXY1=HXY*-1
106     HXY2=HXY*-1
107     F12 = (HXY - HXY1)/maks
108     F13 = np.sqrt(np.abs(1 - np.exp( -2 * (HXY2 - HXY))))
        ##IMoF 1
109     fitur[11]=F12
        ##IMoF 2
110     fitur[12]=F13
111     return fitur
```

***Sourcecode* 0.14 Implementasi fungsi fiturglcm()**

Berikut merupakan penjelasan dari *Sourcecode* 5.14:

1. Baris 1 merupakan deklarasi fungsi fiturglcm() dengan parameter arr bertipe data float64. Parameter arr digunakan untuk perhitungan fitur GLCM.
2. Baris 4-8 merupakan proses perhitungan fitur *Angular Second Moment* yang hasilnya disimpan kedalam array fitur ke 0
3. Baris 9-13 merupakan proses perhitungan fitur *Contrast* yang hasilnya disimpan kedalam array fitur ke 1
4. Baris 14-30 merupakan proses perhitungan fitur *Correlation* yang hasilnya disimpan kedalam array fitur ke 2
5. Baris 31-42 merupakan proses perhitungan fitur *Variance* yang hasilnya disimpan kedalam array fitur ke 3
6. Baris 43-47 merupakan proses perhitungan fitur *Inverse Difference Moment* yang hasilnya disimpan kedalam array fitur ke 4
7. Baris 48-55 merupakan proses perhitungan fitur *Sum Average* yang hasilnya disimpan kedalam array fitur ke 5
8. Baris 46-60 merupakan proses perhitungan fitur *Sum Entropy* yang hasilnya disimpan kedalam array fitur ke 6
9. Baris 61-64 merupakan proses perhitungan fitur *Sum Variance* yang hasilnya disimpan kedalam array fitur ke 7
10. Baris 65 digunakan untuk memanggil fungsi entropy() yang kemudian hasilnya disimpan kedalam array fitur ke 8. Proses perhitungan entropy ditunjukkan pada *Sourcecode* 5.12

11. Baris 66-74 merupakan proses perhitungan fitur *Difference Entropy* yang hasilnya disimpan kedalam array fitur ke 9
12. Baris 75-78 merupakan proses perhitungan fitur *Difference Variance* yang hasilnya disimpan kedalam array fitur ke 10
13. Baris 79-110 merupakan proses perhitungan fitur *Information Measure of Correlation* 1 dan 2, *Information Measure of Correlation* 1 disimpan kedalam array fitur ke 11 dan *Information Measure of Correlation* 2 disimpan kedalam array fitur ke 12
14. Baris 111 digunakan untuk mengembalikan nilai variabel fitur bertipe list

5.3.3 Implementasi *Manhattan Distance*

Implementasi *manhattan distance* digunakan sebagai pelatihan data dalam sistem temu kembali citra. Implementasi *code manhattan distance* dilakukan setelah didapatkan nilai ekstraksi fitur GLCM. Implementasi *code manhattan* terbagi menjadi 2 fungsi. Pertama, fungsi manhattan() yang digunakan untuk menghitung jarak *manhattan* data citra uji dengan data citra latih. Kedua, fungsi sort() digunakan untuk mengurutkan hasil perhitungan jarak *manhattan*, menyimpan hasil pengurutan yang paling kecil sebanyak *n*. Hasil dari fungsi sort() yaitu citra sebanyak *n* yang memiliki kemiripan dengan citra *query*. Fungsi manhattan() ditunjukkan pada *Sourcecode* 5.15.

```
1  def manhattan(dataUji, dataLatih):
2      jarak = 0
3      for x in range((len(dataUji)-1)):
4          jarak+=abs(dataUji[x]-dataLatih[x])
5  return jarak
```

***Sourcecode* 0.15 Implementasi fungsi manhattan()**

Berikut merupakan penjelasan dari *Sourcecode* 5.15:

1. Baris 1 merupakan deklarasi fungsi manhattan() dengan parameter dataUji yang merupakan list sekumpulan data uji berupa hasil perhitungan fungsi GLCM dan dataLatih yang juga merupakan list sekumpulan basis data berupa hasil perhitungan fungsi GLCM
2. Baris 2-4 merupakan proses perhitungan jarak *manhattan* antara data uji dengan basis data yang disimpan kedalam variabel jarak
3. Baris 5 digunakan untuk mengembalikan nilai jarak dengan tipe data float

Fungsi kedua sort() ditunjukkan pada *Sourcecode* 5.16.

```
1  def sort(dataLatih, dataUji, n):
2      jarak = []
3      for bdataLatih in range(len(dataLatih)):
```

```
4           jarakEu = manhattan(dataUji, dataLatih[bdataLatih])
5           jarak.append((dataLatih[bdataLatih], jarakEu))
6       jarak = sorted(jarak, key=lambda x: x[1])
7       tetangga = []
8       for x in range(k):
9           tetangga.append(jarak[x][0])
10      return tetangga
```

***Sourcecode* 0.16 Implementasi fungsi sort()**

Berikut merupakan penjelasan dari *Sourcecode* 5.16:

1. Baris 1 merupakan deklarasi fungsi sort() dengan parameter dataUji yang merupakan list sekumpulan data uji berupa hasil perhitungan fungsi GLCM, dataLatih yang juga merupakan list sekumpulan basis data berupa hasil perhitungan fungsi GLCM dan k dengan tipe data int untuk menentukan banyak data yang diambil sebelum penentuan kelas
2. Baris 3-5 merupakan pemanggilan fungsi manhattan() dengan parameter list dataUji dan list dataLatih yang disimpan kedalam variabel jarak
3. Baris 6 digunakan untuk mengurutkan besarnya jarak dari kecil ke besar
4. Baris 7-9 digunakan untuk mengambil data sebanyak k dengan jarak terkecil yang disimpan kedalam variabel list tetangga
5. Baris 10 digunakan untuk mengembalikan hasil citra sebanyak n yang memiliki kemiripan dengan *query*