Nama : Athief Naufal Ridhotullah

NIM : 2009106101

**Bab 7: Dilasi, Erosi, Opening, Closing**

Pada bab 7, materi yang diberikan adalah proses morfologi. Operasi morfologi dalam citra merupakan suatu proses yang bertujuan untuk mengubah bentuk objek pada citra asli. Proses tersebut dapat dilakukan pada citra grayscale maupun citra biner. Jenis-jenis operasi morfologi di antaranya adalah dilasi, erosi, closing, dan opening

1. **Dilasi** (biasa disimbolkan dengan **⊕**) adalah salah satu dari dua operasi dasar dalam [morfologi pengolahan citra](https://id.wikipedia.org/wiki/Morfologi_matematis" \o "Morfologi matematis). Operasi ini memperluas (memperbesar) objek latar depan.
2. **Erosi** (biasa disimbolkan dengan **⊖**) adalah salah satu dari dua operasi dasar dalam [morfologi pengolahan citra](https://id.wikipedia.org/wiki/Morfologi_matematis" \o "Morfologi matematis). Operasi ini mempersempit (memperkecil) objek latar depan.
3. **Opening**

Pembukaan citra biner A oleh B didapatkan dari erosi A oleh B, lalu diikuti dengan dilasi oleh B.

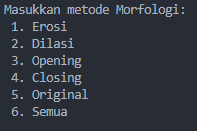


1. **Closing**

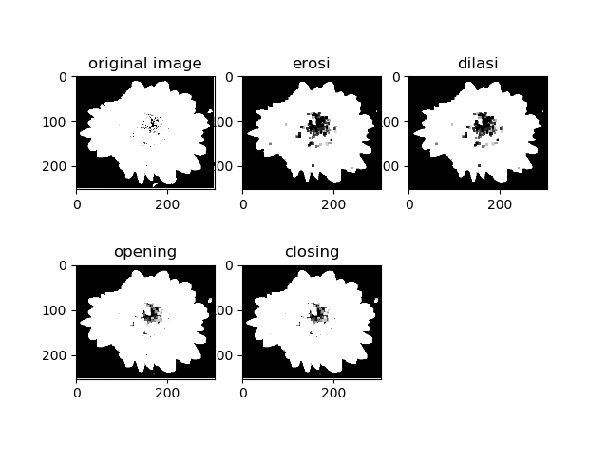
Penutupan A oleh B didapatkan dari dilasi A oleh B, lalu diikuti dengan erosi oleh B.



Di dalam program, user akan diberi kesempatan untuk menginput dari metode morfologi



Berikut adalah output jika user memilih pilihan no 6



**Source Code:**

|  |
| --- |
| import cv2  img = cv2.imread('red.png')  def **main**():      erosi = cv2.erode(img,kernel,iterations = 1)      dilasi = cv2.dilate(erosi,kernel,iterations = 1)      opening = cv2.morphologyEx(dilasi, cv2.MORPH\_OPEN, kernel)      closing = cv2.morphologyEx(opening, cv2.MORPH\_CLOSE, kernel)      options()      cv2.waitKey(0)      cv2.destroyAllWindows()  def **options**():      show\_morf = input("Masukkan metode Morfologi:\n 1. Erosi\n 2. Dilasi\n 3. Opening\n 4. Closing\n 5. Original \n 6. Semua")      if show\_morf == 1:          cv2.imshow('erosi', erosi)      elif show\_morf == 2:          cv2.imshow('dilasi', dilasi)      elif show\_morf == 3:          cv2.imshow('opening', opening)      elif show\_morf == 4:          cv2.imshow('closing', closing)      elif show\_morf == 5:          cv2.imshow('original', img)      elif show\_morf == 6:          cv2.imshow('original', img)          cv2.imshow('erosi', erosi)          cv2.imshow('dilasi', dilasi)          cv2.imshow('opening', opening)          cv2.imshow('closing', closing)      else:          print("Error")          options()  main()  #referensi  #https://www.ivanjul.com/dilation-pelebaran-dengan-opencv-python/  #https://www.youtube.com/watch?v=DUq7IPLi8TA&t=511s  #https://github.com/jjone36/vision\_4\_beginners |

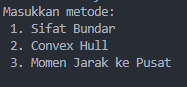
**Bab 12: Dilasi, Erosi, Opening, Closing**

Pada bab 12, Ekstraksi Fitur Bentuk dan Kontur dibahas secara detail, Beberapa fitur berdasarkan bentuk dan kontur telah dibahas pada Bab 8. Sebagai contoh, fitur kekompakan diperoleh melalui perimeter dan luas objek. Beberapa fitur lain yang telah dikupas antara lain berupa dispersi dan kerampingan.

Pada bab ini, beberapa fitur lain yang terkait dengan bentuk dan kontur akan dibahas.

1. **Sifat bundar** atau circularity adalah perbandingan antara rerata jarak Euclidean dari sentroid terhadap tepi area dan deviasi standar jarak dari sentroid ke tepi area.
2. **Convex hull** adalah himpunan cembung terkecil yang memuatnyam bentuk Convex hull yang ideal adalah seperti karet gelang yang dipasang di tepi objek.
3. **Momen Jarak ke Pusat,** menurut(Rangayyan, 2005) Apabila objek berupa kontur saja (misalnya berupa bentuk suatu pulau), momen dapat dihitung melalui jarak titik pada kontur terhadap pusat massa.

Sama seperti program sebelumnya, menggunakan user input



Hasil Convex hull



**Source Code:**

|  |
| --- |
| from \_\_future\_\_ import print\_function  import cv2  import numpy as np  import argparse  import random as rng  rng.seed(12345)  #main  def **main**():      conveks = input("Masukkan metode:\n 1. Sifat Bundar\n 2. Convex Hull\n 3. Momen Jarak ke Pusat")      if conveks == 1:          bundar()      elif conveks == 2:          convexHull()      elif conveks == 3:          momen()      else:          print("Error")          main()  #Sifat Bundar  def **bundar**():      [px, py] = centroid(BW);        [m, n] = size(BW);      Kontur = inbound\_tracing(BW);        jum =length(Kontur);        total = 0;      for i in jum:          i = 1          total =  total + sqrt( (Kontur(i,1)-py)^2 + (Kontur(i,2)-px)^2);      mu = total / jum;      total = 0;        for i in jum:          i = 1          total =  total + (sqrt( (Kontur(i,1)-py)^2 + (Kontur(i,2)-px)^2) - mu) ^ 2;        sigma = total / jum;        #hasil      c = mu / sigma;  #Convex hull  def **convexHull**():      def **thresh\_callback**(val):          threshold = val          canny\_output = cv2.Canny(src\_gray, threshold, threshold \* 2)          contours, \_ = cv2.findContours(canny\_output, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)          hull\_list = []          for i in range(len(contours)):              hull = cv2.convexHull(contours[i])              hull\_list.append(hull)          drawing = np.zeros((canny\_output.shape[0], canny\_output.shape[1], 3), dtype=np.uint8)          for i in range(len(contours)):              color = (rng.randint(0,256), rng.randint(0,256), rng.randint(0,256))              cv2.drawContours(drawing, contours, i, color)              cv2.drawContours(drawing, hull\_list, i, color)          cv2.imshow('Contours', drawing)      parser = argparse.ArgumentParser(description='Code for Convex Hull tutorial.')      parser.add\_argument('--input', help='Path to input image.', default='bung-A.jpg')      args = parser.parse\_args()      src = cv2.imread(cv2.samples.findFile(args.input))      if src is None:          print('Could not open or find the image:', args.input)          exit(0)      src\_gray = cv2.cv2tColor(src, cv2.COLOR\_BGR2GRAY)      src\_gray = cv2.blur(src\_gray, (3,3))      source\_window = 'Source'      cv2.namedWindow(source\_window)      cv2.imshow(source\_window, src)      max\_thresh = 255      thresh = 100 # treshold awal      cv2.createTrackbar('Canny thresh:', source\_window, thresh, max\_thresh, thresh\_callback)      thresh\_callback(thresh)      cv2.waitKey()  #Momen jarak ke pusat  def **momen**():      Fitur.F1 = sqrt(momen\_pusat\_ke\_p(F, 2)) / momen\_ke\_p(F, 1);      Fitur.F2 = momen\_pusat\_ke\_p(F, 3) / (momen\_ke\_p(F, 2)^1.5);      Fitur.F3 = momen\_pusat\_ke\_p(F, 4) / (momen\_ke\_p(F, 2)^2);        Fitur.F2a = momen\_pusat\_ke\_p(F, 3)^(1/3) / momen\_ke\_p(F, 1);      Fitur.F3a = momen\_pusat\_ke\_p(F, 4)^(1/4) / momen\_ke\_p(F, 1);      Fitur.mf = Fitur.F3a - Fitur.F1;        function [momen] = momen\_pusat\_ke\_p(F, p)        momen\_p = momen\_ke\_p(F, p);      momen\_1 = momen\_ke\_p(F, 1);        Kontur = inbound\_tracing(F);        [m, n] = size(F);      [xp, yp] = centroid(F);      jum = length(Kontur);        momen = 0;      for i in jum:          i=1          jarak = sqrt((Kontur(i,2)-xp)^2 + (Kontur(i,1)-yp)^2);            momen = momen + abs(jarak - momen\_1) ^ p;      momen = momen / jum;      function [momen] = momen\_ke\_p(F, p)        Kontur = inbound\_tracing(F);      [m, n] = size(F);      [xp, yp] = centroid(F);      jum = length(Kontur);      momen = 0;      for i in jum:          jarak = sqrt((Kontur(i,2)-xp)^2 + (Kontur(i,1)-yp)^2);          momen = momen + jarak ^ p;      momen = momen / jum;  main()  #referensi  #https://www.researchgate.net/publication/358220979\_Pengolahan\_Citra\_Digital\_Menggunakan\_Python  #https://www.researchgate.net/profile/Abdul-Kadir-14/publication/236673073\_Teori\_dan\_Aplikasi\_Pengolahan\_Citra/links/00b49518db5b7608fd000000/Teori-dan-Aplikasi-Pengolahan-Citra.pdf  #https://www.youtube.com/watch?v=W7YKrMynXpE |

**Bab 14: Jarak Euclidean, Manhattan/City Blok**

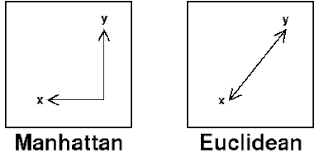
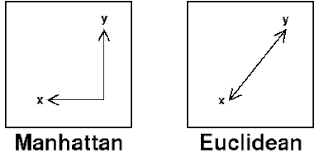
Aplikasi Temu Kembali Citra dibahas dalam bab 14, Temu kembali citra (image retrieval) merupakan proses untuk mendapatkan sejumlah citra berdasarkan masukan satu citra. Istilah yang lebih spesifik lagi adalah content based image retrieval (CBIR) atau temu kembali citra berdasarkan isinya.

1. Jarak Euclidean jarak ukur antara dua vektor dengan menghitung akar kuadrat dari jumlah selisih kuadrat antara keduanya.
2. Jarak Manhattan atau city-block adalah jarak ukur antara dua titik adalah jumlah dari panjang ruas garis kedua titik tersebut terhadap tiap sumbu dalam koordinat Kartesius, jarak dapat dibayangkan sebagai jarak vertikal plus horizontal dari vektor pertama ke vektor kedua

Pilihan dalam program:



Euclidean dan Manhattan



**Source Code:**

|  |
| --- |
| from \_\_future\_\_ import print\_function  import cv2  import numpy as np  import argparse  import random as rng  rng.seed(12345)  parser = argparse.ArgumentParser(description='Code for Image Segmentation with Distance Transform and Watershed Algorithm.\      Sample code showing how to segment overlapping objects using Laplacian filtering, \      in addition to Watershed and Distance Transformation')  parser.add\_argument('--input', help='Path to input image.', default='red.png')  args = parser.parse\_args()  src = cv2.imread(cv2.samples.findFile(args.input))  def **main**():      if src is None:          print('Error:', args.input)          exit(0)      options()      cv2.imshow('Source Image', src)      src[np.all(src == 255, axis=2)] = 0        cv2.imshow('Black Background Image', src)      kernel = np.array([[1, 1, 1], [1, -8, 1], [1, 1, 1]], dtype=np.float32)        imgLaplacian = cv2.filter2D(src, cv2.CV\_32F, kernel)      sharp = np.float32(src)      imgResult = sharp - imgLaplacian        imgResult = np.clip(imgResult, 0, 255)      imgResult = imgResult.astype('uint8')      imgLaplacian = np.clip(imgLaplacian, 0, 255)      imgLaplacian = np.uint8(imgLaplacian)        cv2.imshow('New Sharped Image', imgResult)      bw = cv2.cvtColor(imgResult, cv2.COLOR\_BGR2GRAY)      \_, bw = cv2.threshold(bw, 40, 255, cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)      cv2.imshow('Binary Image', bw)      dist = cv2.distanceTransform(bw, cv2.DIST\_L2, 3)        cv2.normalize(dist, dist, 0, 1.0, cv2.NORM\_MINMAX)      cv2.imshow('Distance Transform Image', dist)      \_, dist = cv2.threshold(dist, 0.4, 1.0, cv2.THRESH\_BINARY)      # Dilate a bit the dist image      kernel1 = np.ones((3,3), dtype=np.uint8)      dist = cv2.dilate(dist, kernel1)      cv2.imshow('Peaks', dist)      dist\_8u = dist.astype('uint8')      # Find total markers      \_, contours, \_ = cv2.findContours(dist\_8u, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)        markers = np.zeros(dist.shape, dtype=np.int32)      # Draw the foreground markers      for i in range(len(contours)):          cv2.drawContours(markers, contours, i, (i+1), -1)      # Draw the background marker      cv2.circle(markers, (5,5), 3, (255,255,255), -1)      markers\_8u = (markers \* 10).astype('uint8')      cv2.imshow('Markers', markers\_8u)      cv2.watershed(imgResult, markers)        mark = markers.astype('uint8')      mark = cv2.bitwise\_not(mark)        colors = []      for contour in contours:          colors.append((rng.randint(0,256), rng.randint(0,256), rng.randint(0,256)))        dst = np.zeros((markers.shape[0], markers.shape[1], 3), dtype=np.uint8)        for i in range(markers.shape[0]):          for j in range(markers.shape[1]):              index = markers[i,j]              if index > 0 and index <= len(contours):                  dst[i,j,:] = colors[index-1]        cv2.imshow('Final Result', dst)      cv2.waitKey()  def **euclid**():      return (((x2 - x1)\*\*2 +(y2 - y1)\*\*2)\*\*0.5)    def **manhat**():      dist = Math.fabs(X2 - X1) + Math.fabs(Y2 - Y1)      return (int)(dist)  print(distancesum(x, y, n) )  def **options**():      distant = input("Masukkan metode Jarak:\n 1. Euclidean\n 2. Manhattan/City Block")      if distant == 1:          euclid()      elif distant == 2:          manhat()      else:          print("Error")          options()  main()  #referensi  #https://www.youtube.com/watch?v=\_3S3eTvBEns |