**VISI KOMPUTER DAN PENGOLAHAN CITRA**

**S2 TEKNIK INFOMATIKA DAN KOMPUTER**

**Script Program Jawaban UAS**



NRP : 1223800002

NAMA : Nur Rizky Romadhon

**Teknik Informatika dan Komputer Politeknik Elektronika Negeri Surabaya**

**2023**

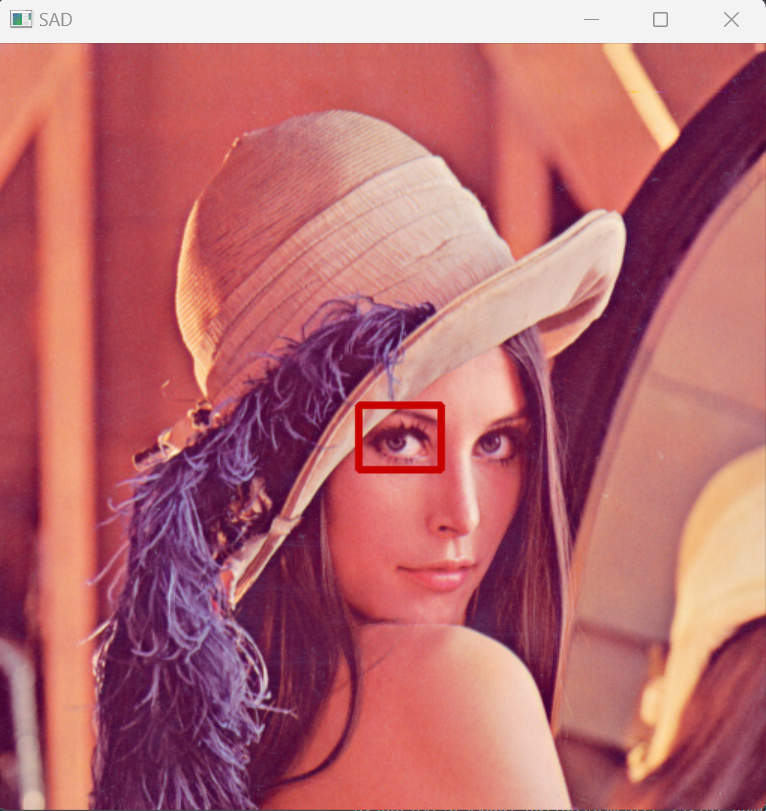
Praktikum 5Filter

**PROSEDUR PERCOBAAN :**

1. **Script program sederhana untuk SAD dan SSD**
2. **Program SAD**

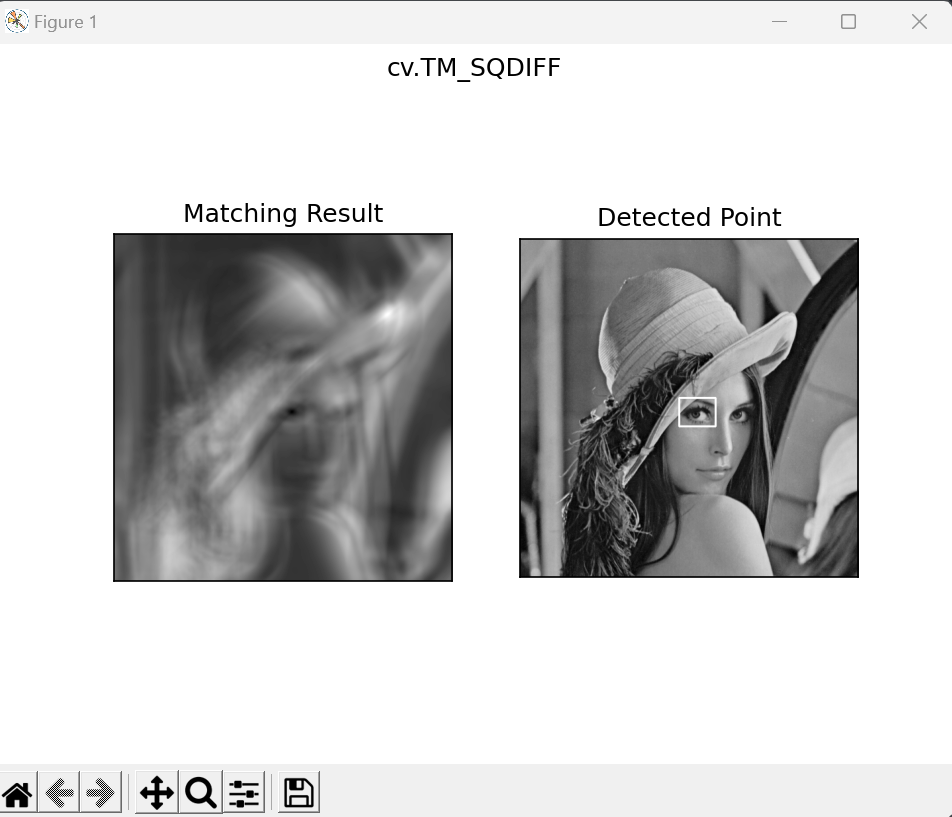
|  |
| --- |
| import cv2  import numpy as np  def template\_matching\_sad(src, temp):      h, w = src.shape      ht, wt = temp.shape      score = np.empty((h-ht, w-wt))      for dy in range(0, h - ht):          for dx in range(0, w - wt):              diff = np.abs(src[dy:dy + ht, dx:dx + wt] - temp)              score[dy, dx] = diff.sum()      pt = np.unravel\_index(score.argmin(), score.shape)      return (pt[1], pt[0])  def main():      img = cv2.imread("./Lenna.png")      temp = cv2.imread("./Lenna\_mata.png")      gray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)      temp = cv2.cvtColor(temp, cv2.COLOR\_RGB2GRAY)      h, w = temp.shape      pt = template\_matching\_sad(gray, temp)  *#match = cv2.matchTemplate(gray, temp, cv2.TM\_SQDIFF\_NORMED)*  *#min\_value, max\_value, min\_pt, max\_pt = cv2.minMaxLoc(match)*  *#pt = min\_pt*      cv2.rectangle(img, (pt[0], pt[1]), (pt[0] + w, pt[1] + h), (0,0,200), 3)      cv2.imshow('Deteksi Lingkaran', img)      cv2.waitKey(0)      cv2.destroyAllWindows()  if \_\_name\_\_ == "\_\_main\_\_":      main() |

****

****

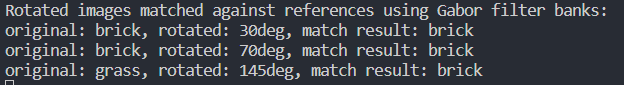
1. **Program SSD**

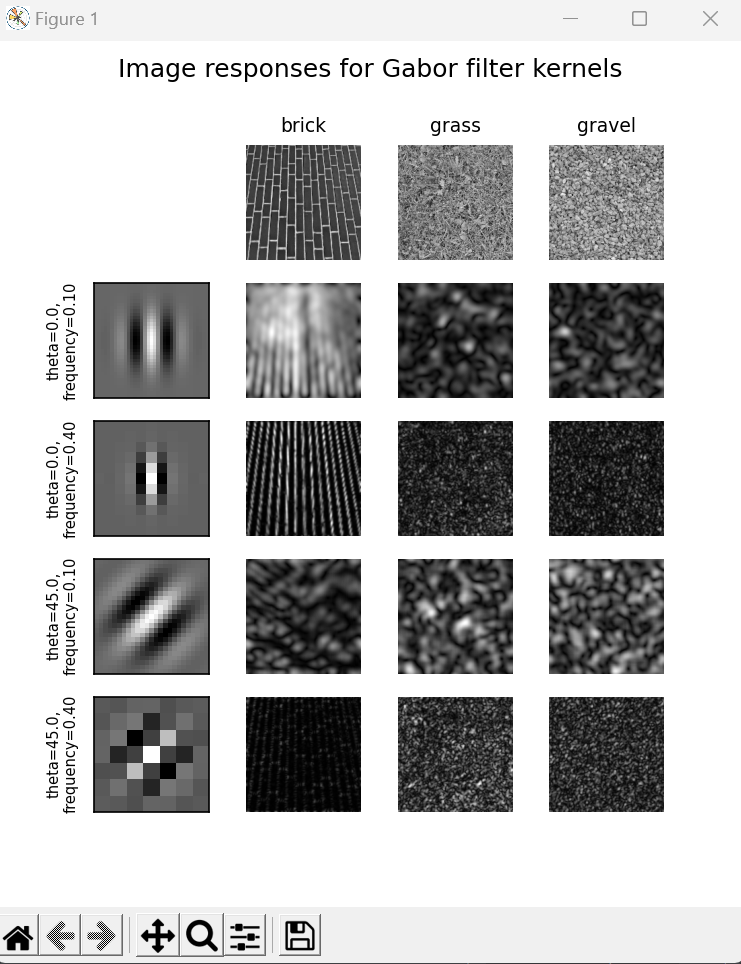
|  |
| --- |
| import cv2 as cv  import numpy as np  import sys  from matplotlib import pyplot as plt  img = cv.imread(sys.path[0]+"/Lenna.png", cv.IMREAD\_GRAYSCALE)  assert img is not None, "file could not be read, check with os.path.exists()"  img2 = img.copy()  template = cv.imread(sys.path[0]+"/Lenna\_mata.png", cv.IMREAD\_GRAYSCALE)  assert template is not None, "file could not be read, check with os.path.exists()"  w, h = template.shape[::-1]  *# All the 6 methods for comparison in a list*  *#methods = ['cv.TM\_CCOEFF', 'cv.TM\_CCOEFF\_NORMED', 'cv.TM\_CCORR',*  *#            'cv.TM\_CCORR\_NORMED', 'cv.TM\_SQDIFF', 'cv.TM\_SQDIFF\_NORMED']*  methods = ['cv.TM\_SQDIFF']  for meth in methods:      img = img2.copy()      method = eval(meth)  *# Apply template Matching*      res = cv.matchTemplate(img,template,method)      min\_val, max\_val, min\_loc, max\_loc = cv.minMaxLoc(res)  *# If the method is TM\_SQDIFF or TM\_SQDIFF\_NORMED, take minimum*      if method in [cv.TM\_SQDIFF, cv.TM\_SQDIFF\_NORMED]:          top\_left = min\_loc      else:          top\_left = max\_loc      bottom\_right = (top\_left[0] + w, top\_left[1] + h)      cv.rectangle(img,top\_left, bottom\_right, 255, 2)      plt.subplot(121),plt.imshow(res,cmap = 'gray')      plt.title('Matching Result'), plt.xticks([]), plt.yticks([])      plt.subplot(122),plt.imshow(img,cmap = 'gray')      plt.title('Detected Point'), plt.xticks([]), plt.yticks([])      plt.suptitle(meth)      plt.show() |

****

1. **Script program sederhana untuk filter bank dan image pyramid**
2. **Program Filter bank**

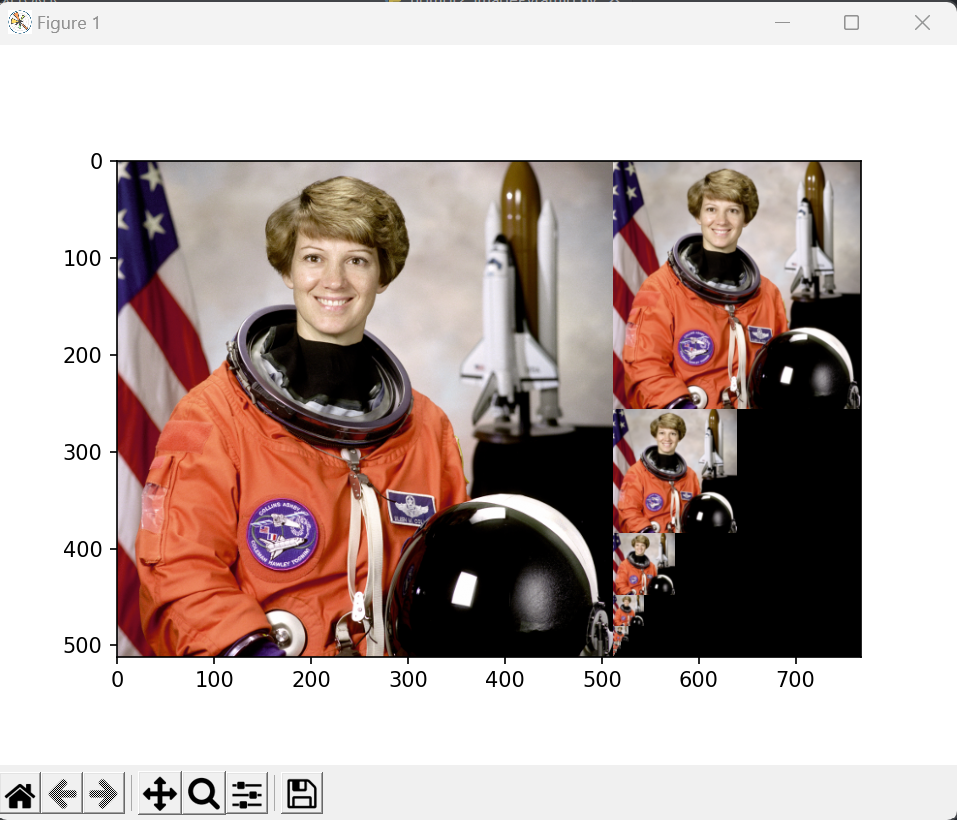
|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  from scipy import ndimage as ndi  from skimage import data  from skimage.util import img\_as\_float  from skimage.filters import gabor\_kernel  def compute\_feats(image, kernels):      feats = np.zeros((len(kernels), 2), dtype=np.double)      for k, kernel in enumerate(kernels):          filtered = ndi.convolve(image, kernel, mode='wrap')          feats[k, 0] = filtered.mean()          feats[k, 1] = filtered.var()      return feats  def match(feats, ref\_feats):      min\_error = np.inf      min\_i = None      for i in range(ref\_feats.shape[0]):          error = np.sum((feats - ref\_feats[i, :])\*\*2)          if error < min\_error:              min\_error = error              min\_i = i      return min\_i  *# prepare filter bank kernels*  kernels = []  for theta in range(4):      theta = theta / 4. \* np.pi      for sigma in (1, 3):          for frequency in (0.05, 0.25):              kernel = np.real(gabor\_kernel(frequency, theta=theta,                                            sigma\_x=sigma, sigma\_y=sigma))              kernels.append(kernel)  shrink = (slice(0, None, 3), slice(0, None, 3))  brick = img\_as\_float(data.brick())[shrink]  grass = img\_as\_float(data.grass())[shrink]  gravel = img\_as\_float(data.gravel())[shrink]  image\_names = ('brick', 'grass', 'gravel')  images = (brick, grass, gravel)  *# prepare reference features*  ref\_feats = np.zeros((3, len(kernels), 2), dtype=np.double)  ref\_feats[0, :, :] = compute\_feats(brick, kernels)  ref\_feats[1, :, :] = compute\_feats(grass, kernels)  ref\_feats[2, :, :] = compute\_feats(gravel, kernels)  print('Rotated images matched against references using Gabor filter banks:')  print('original: brick, rotated: 30deg, match result: ', end='')  feats = compute\_feats(ndi.rotate(brick, angle=190, reshape=False), kernels)  print(image\_names[match(feats, ref\_feats)])  print('original: brick, rotated: 70deg, match result: ', end='')  feats = compute\_feats(ndi.rotate(brick, angle=70, reshape=False), kernels)  print(image\_names[match(feats, ref\_feats)])  print('original: grass, rotated: 145deg, match result: ', end='')  feats = compute\_feats(ndi.rotate(grass, angle=145, reshape=False), kernels)  print(image\_names[match(feats, ref\_feats)])  def power(image, kernel):  *# Normalize images for better comparison.*      image = (image - image.mean()) / image.std()      return np.sqrt(ndi.convolve(image, np.real(kernel), mode='wrap')\*\*2 +                     ndi.convolve(image, np.imag(kernel), mode='wrap')\*\*2)  *# Plot a selection of the filter bank kernels and their responses.*  results = []  kernel\_params = []  for theta in (0, 1):      theta = theta / 4. \* np.pi      for frequency in (0.1, 0.4):          kernel = gabor\_kernel(frequency, theta=theta)          params = f"theta={theta \* 180 / np.pi},\nfrequency={frequency:.2f}"          kernel\_params.append(params)  *# Save kernel and the power image for each image*          results.append((kernel, [power(img, kernel) for img in images]))  fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(5, 6))  plt.gray()  fig.suptitle('Image responses for Gabor filter kernels', fontsize=12)  axes[0][0].axis('off')  *# Plot original images*  for label, img, ax in zip(image\_names, images, axes[0][1:]):      ax.imshow(img)      ax.set\_title(label, fontsize=9)      ax.axis('off')  for label, (kernel, powers), ax\_row in zip(kernel\_params, results, axes[1:]):  *# Plot Gabor kernel*      ax = ax\_row[0]      ax.imshow(np.real(kernel))      ax.set\_ylabel(label, fontsize=7)      ax.set\_xticks([])      ax.set\_yticks([])  *# Plot Gabor responses with the contrast normalized for each filter*      vmin = np.min(powers)      vmax = np.max(powers)      for patch, ax in zip(powers, ax\_row[1:]):          ax.imshow(patch, vmin=vmin, vmax=vmax)          ax.axis('off')  plt.show() |

****

****

1. **Program Filter image pyramid**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from skimage import data  from skimage.transform import pyramid\_gaussian  image = data.astronaut()  rows, cols, dim = image.shape  pyramid = tuple(pyramid\_gaussian(image, downscale=2, channel\_axis=-1))  *# determine the total number of rows and columns for the composite*  composite\_rows = max(rows, sum(p.shape[0] for p in pyramid[1:]))  composite\_cols = cols + pyramid[1].shape[1]  composite\_image = np.zeros((composite\_rows, composite\_cols, 3),                             dtype=np.double)  *# store the original to the left*  composite\_image[:rows, :cols, :] = pyramid[0]  *# stack all downsampled images in a column to the right of the original*  i\_row = 0  for p in pyramid[1:]:      n\_rows, n\_cols = p.shape[:2]      composite\_image[i\_row:i\_row + n\_rows, cols:cols + n\_cols] = p      i\_row += n\_rows  fig, ax = plt.subplots()  ax.imshow(composite\_image)  plt.show() |

****

1. **Script program sederhana untuk metode ohlander**

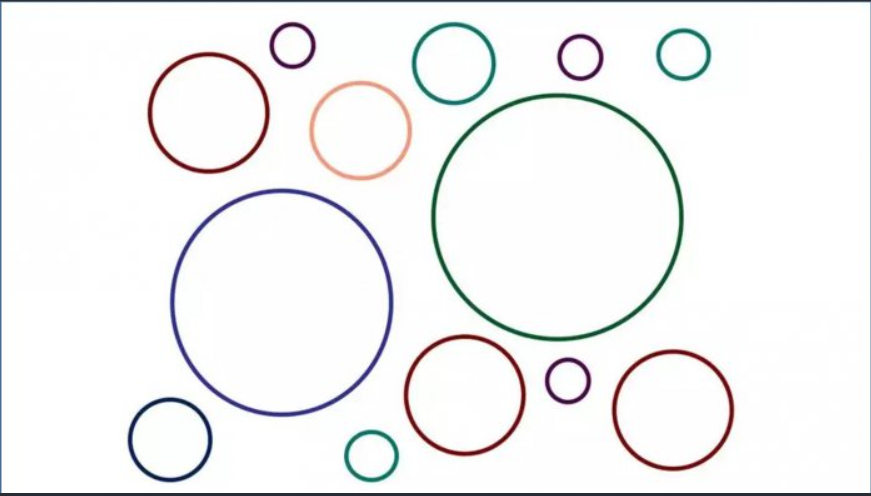
|  |
| --- |
| import cv2  import numpy as np  import sys  *# Fungsi untuk melakukan segmentasi menggunakan metode Ohlander's Recursive Histogram-Based Clustering*  def ohlander\_clustering(image, threshold):      if len(image.shape) > 2:          gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)      else:          gray\_image = image.copy()  *# Mendapatkan histogram dari gambar*      hist = cv2.calcHist([gray\_image], [0], None, [256], [0, 256])  *# Mencari nilai untuk clustering*      split\_value = 0      max\_val = np.max(hist)      for i in range(255, 0, -1):          if hist[i] > threshold \* max\_val:              split\_value = i              break  *# Segmentasi gambar*      segmented\_image = np.zeros\_like(gray\_image)      segmented\_image[gray\_image >= split\_value] = 255      return segmented\_image  input\_image = cv2.imread(sys.path[0]+'/bola2.jpg')  threshold\_value = 0.7  segmented\_image = ohlander\_clustering(input\_image, threshold\_value)  cv2.imshow('Original Image', input\_image)  cv2.imshow('Segmented Image', segmented\_image)  cv2.waitKey(0)  cv2.destroyAllWindows() |

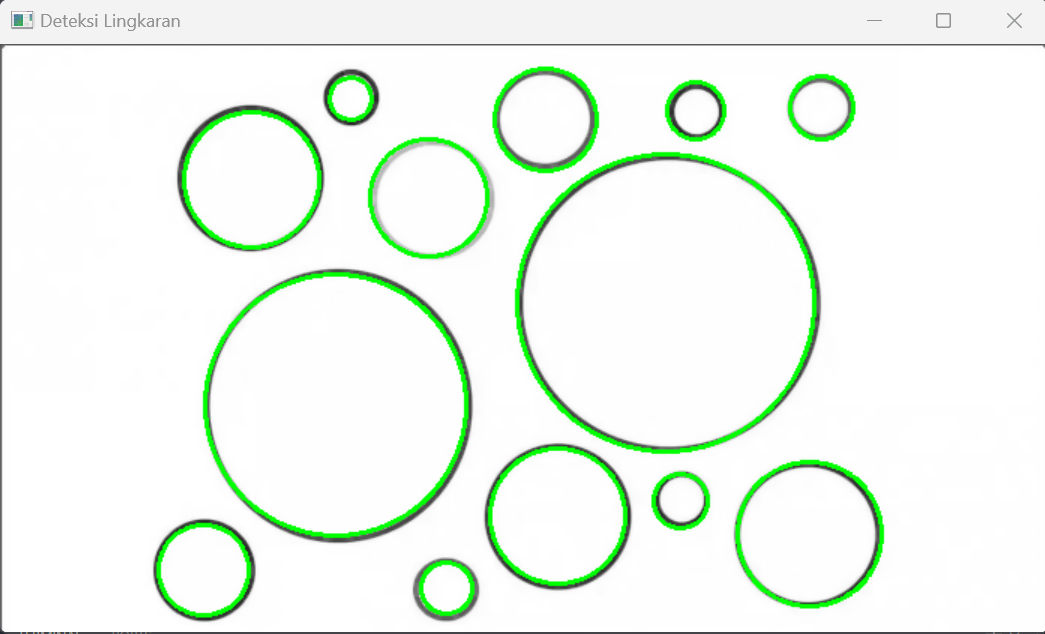
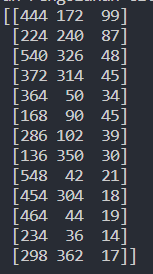
****

****

1. **Script program sederhana untuk transformasi hough untuk mendeteksi lingkaran**

|  |
| --- |
| import cv2  import numpy as np  import sys  *# Baca gambar*  img = cv2.imread(sys.path[0]+"/lingkaran.jpg", cv2.IMREAD\_GRAYSCALE)  *# pre-processing*  imgm = cv2.medianBlur(img,5)  imgb = cv2.cvtColor(imgm, cv2.COLOR\_GRAY2BGR)  *# Deteksi lingkaran menggunakan Hough Transform*  circles = cv2.HoughCircles(      imgm,      cv2.HOUGH\_GRADIENT,      1,  *# Resolusi ruang akumulator yang diinginkan (semakin kecil semakin akurat, tapi juga membutuhkan lebih banyak waktu)*      50,  *# Jarak minimum antara pusat dua lingkaran yang dideteksi*      param1=100,  *# Parameter deteksi tepi (semakin tinggi, semakin ketat)*      param2=50,  *# Parameter ambang batas untuk memilih pusat lingkaran (semakin kecil, semakin ketat)*      minRadius=0,  *# Radius minimum lingkaran yang akan dideteksi*      maxRadius=100  *# Radius maksimum lingkaran yang akan dideteksi*  )  *# Jika lingkaran ditemukan, gambar mereka pada gambar asli*  if circles is not None:  *# cari koordinat x,y dan radius (r)*     circles = np.round(circles[0, :]).astype("int")     print(circles)  *# cari terus dlm loop*     for (x, y, r) in circles:        cv2.circle(imgb, (x, y), r, (0, 255, 0), 2)  *# Tampilkan hasil*  cv2.imshow('Deteksi Lingkaran', imgb)  cv2.waitKey(0)  cv2.destroyAllWindows() |

****

****  ****