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

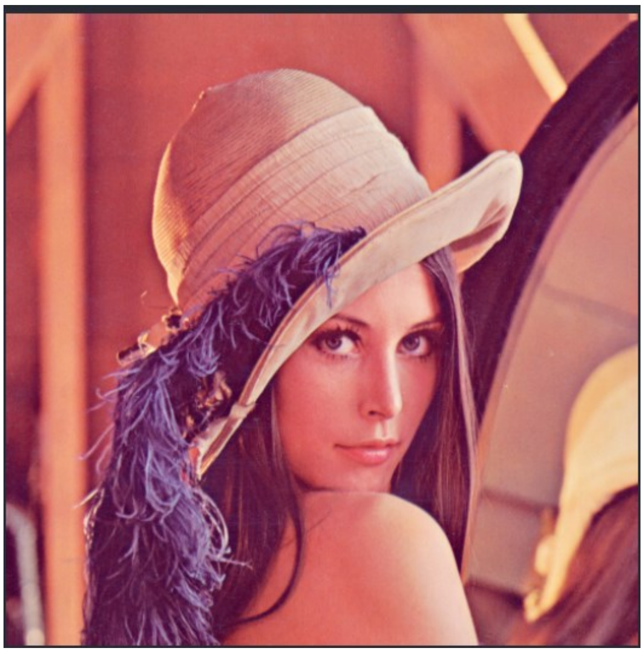
2023

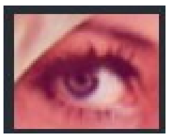
Praktikum 5
Filter

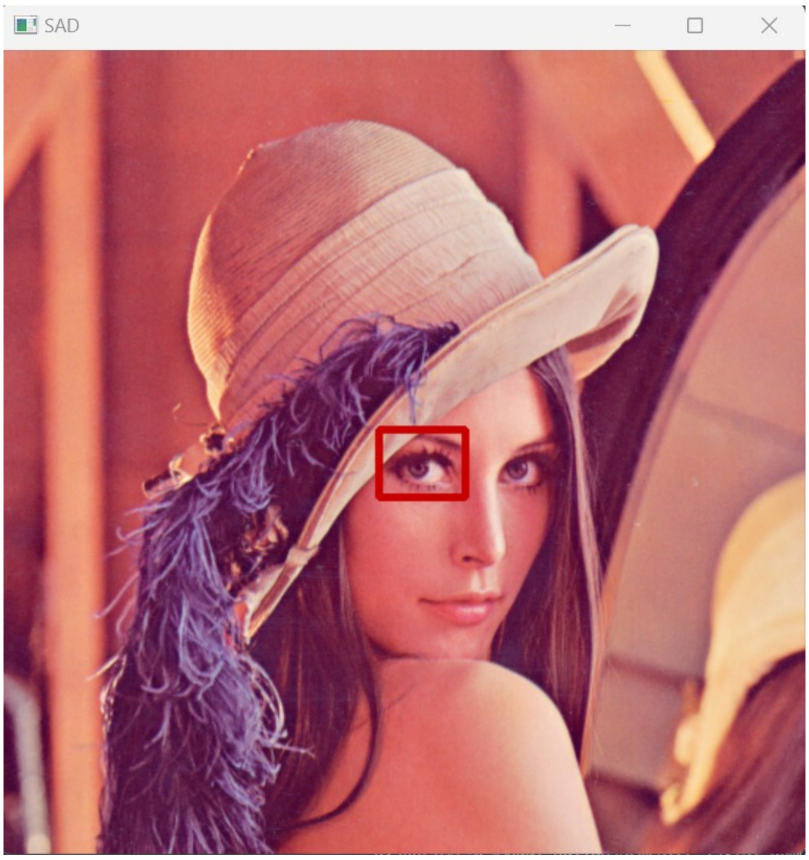
PROSEDUR PERCOBAAN :

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

```python
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()
```

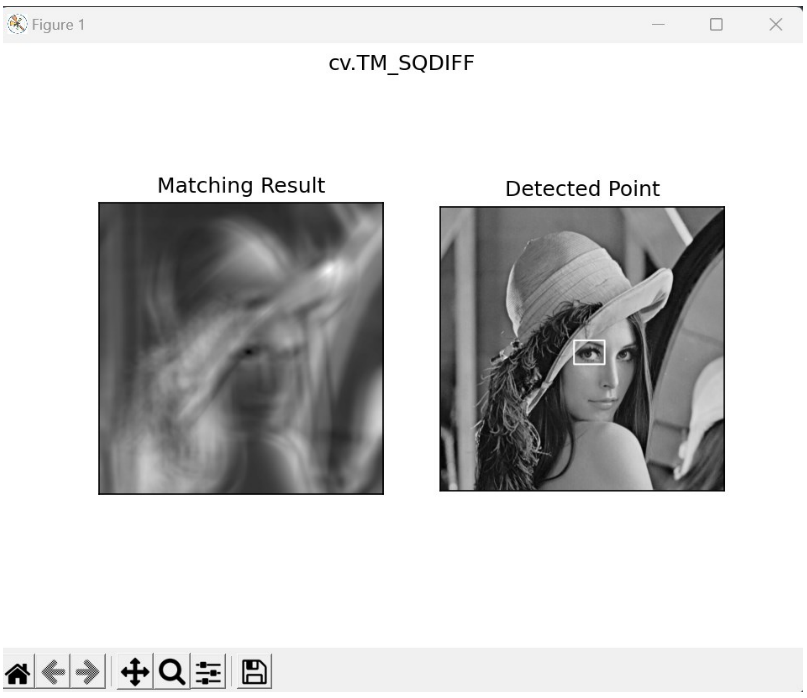






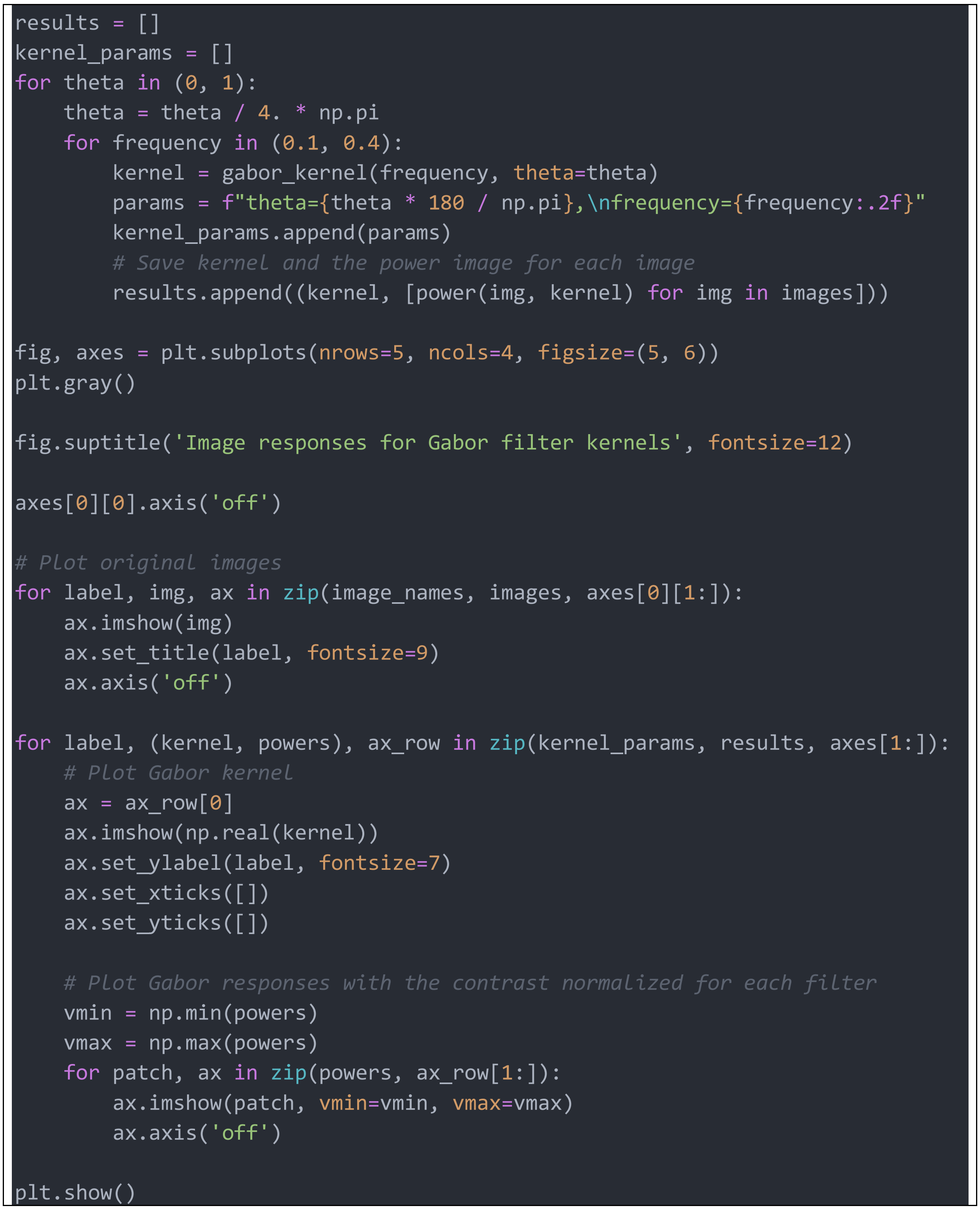
b. Program SSD

```python
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()
```

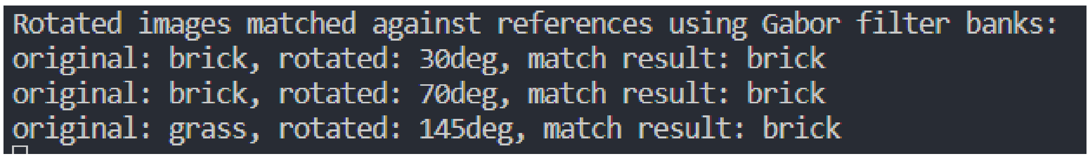


2. **Script program sederhana untuk filter bank dan image pyramid**
   a. **Program Filter bank**

```python
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.
```



```python
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()
```



```
Rotated images matched against references using Gabor filter banks:
original: brick, rotated: 30deg, match result: brick
original: brick, rotated: 70deg, match result: brick
original: grass, rotated: 145deg, match result: brick
```

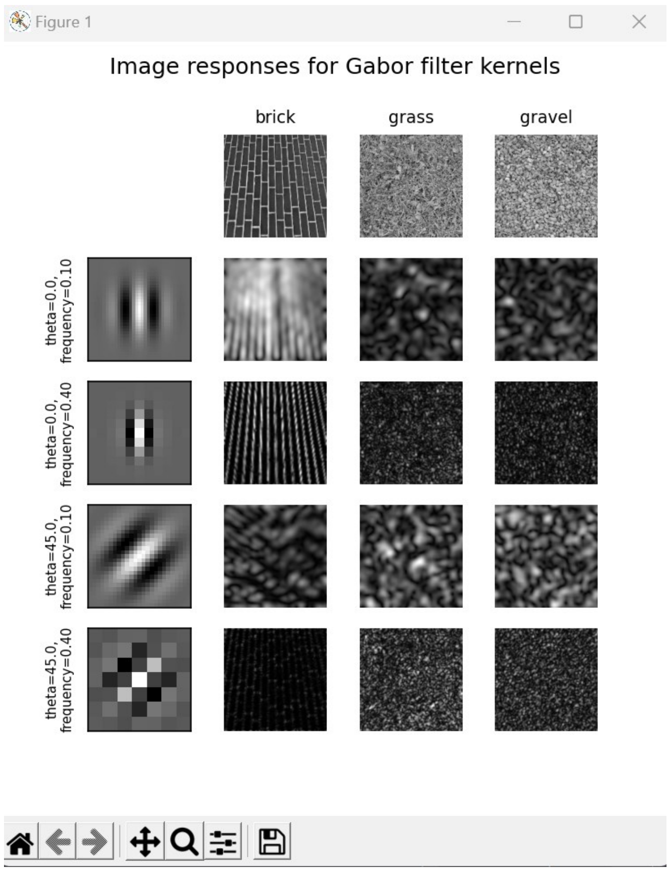
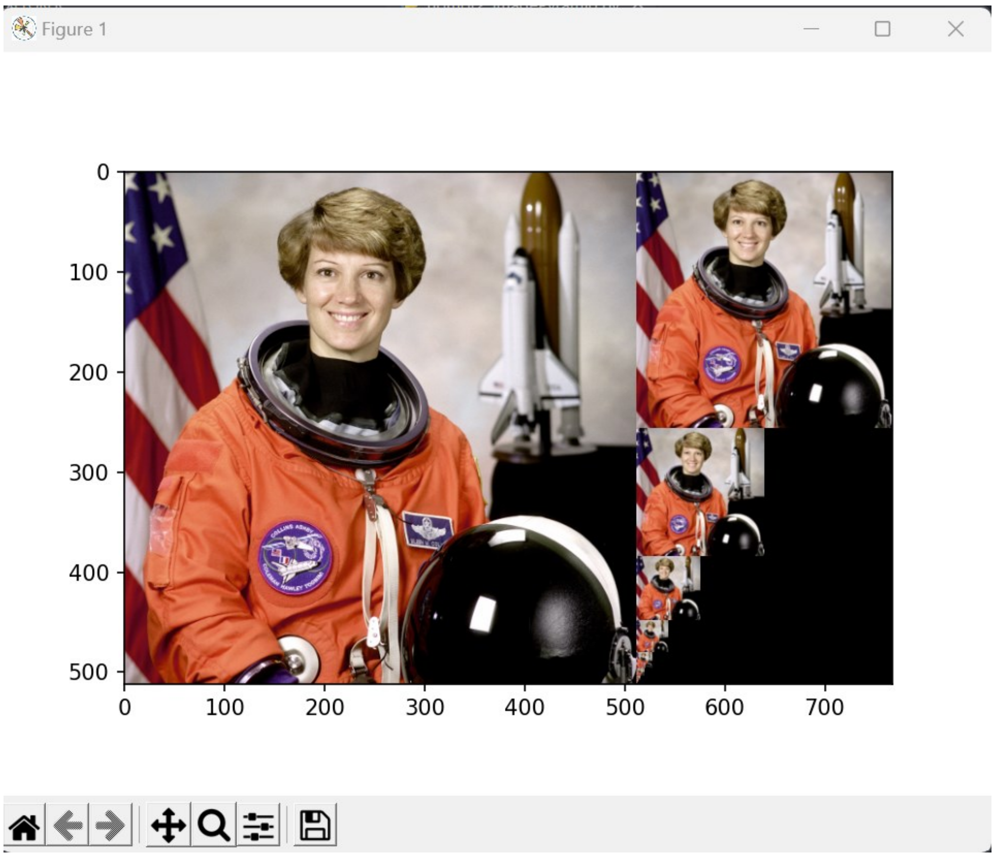


Figure 1
Image responses for Gabor filter kernels
brick
grass
gravel
theta=0.0, frequency=0.10
theta=0.0, frequency=0.40
theta=45.0, frequency=0.10
theta=45.0, frequency=0.40

b. Program Filter image pyramid

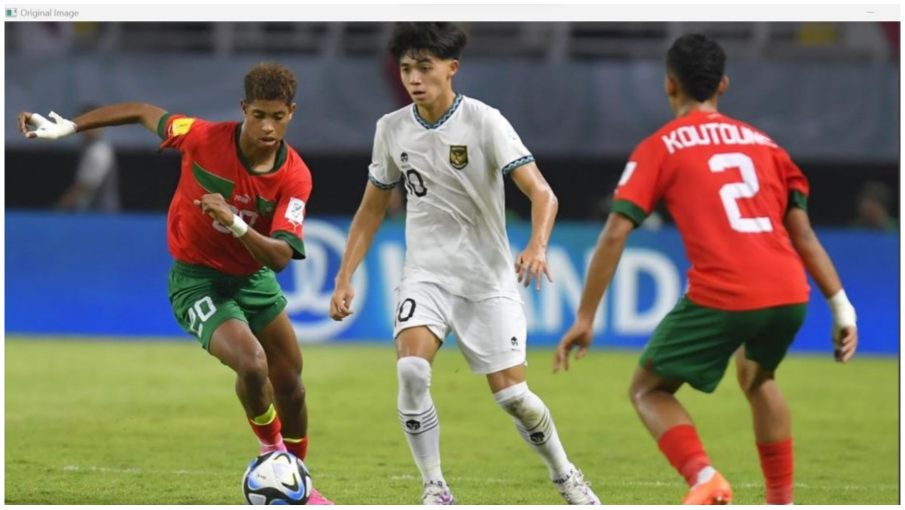
```
import numpy as np
import matplotlib.pyplot as plt

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
```

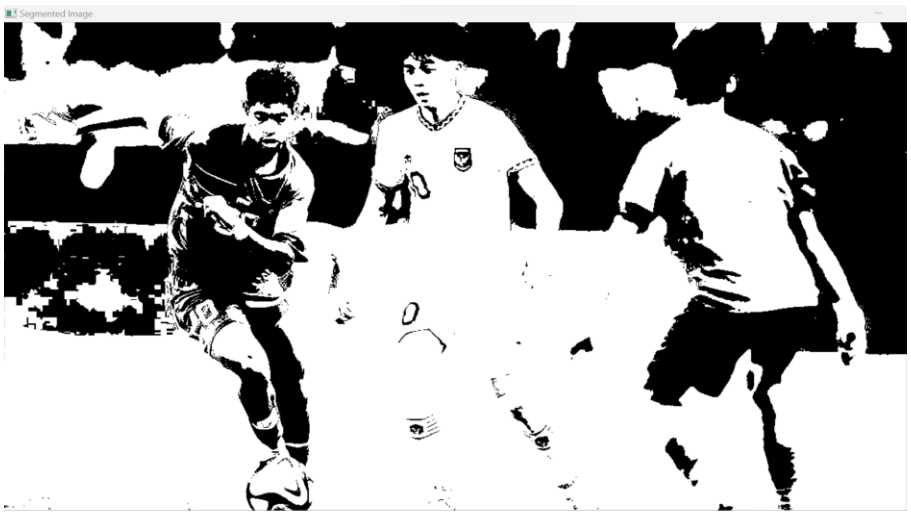
```
i_row = 0
for p in pyramid[1:]:
    n_rows, n_cols = p.shape[:2]
    composite_image[i_row:i_row + n_rows, cols:cols + n_cols] = p
    i_row += n_rows

fig, ax = plt.subplots()
ax.imshow(composite_image)
plt.show()
```



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

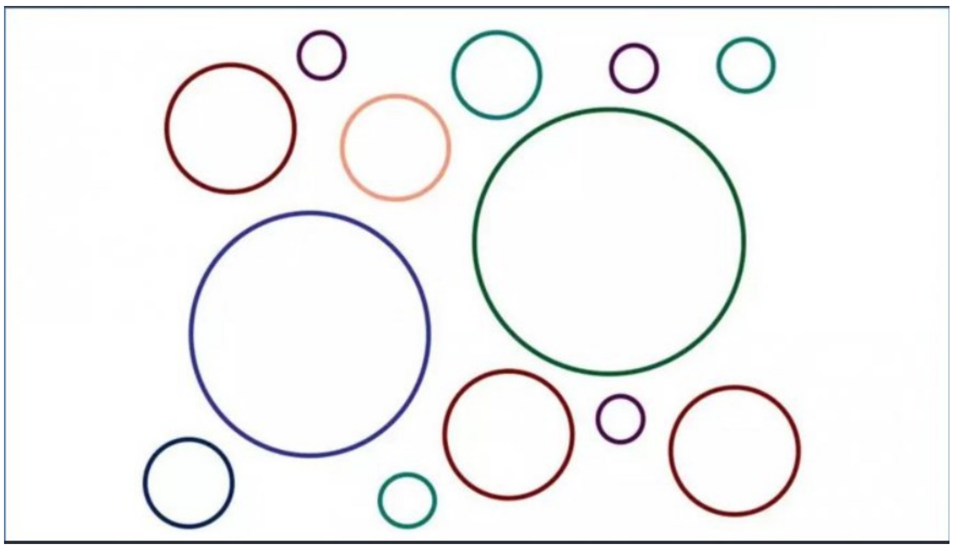
```
import cv2
import numpy as np
import sys
# Fungsi untuk melakukan segmentasi menggunakan metode Ohlander's Recursive
Histogram-Based Clustering
def ohlander_clustering(image, threshold):
    if len(image.shape) > 2:
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        gray_image = 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()
```

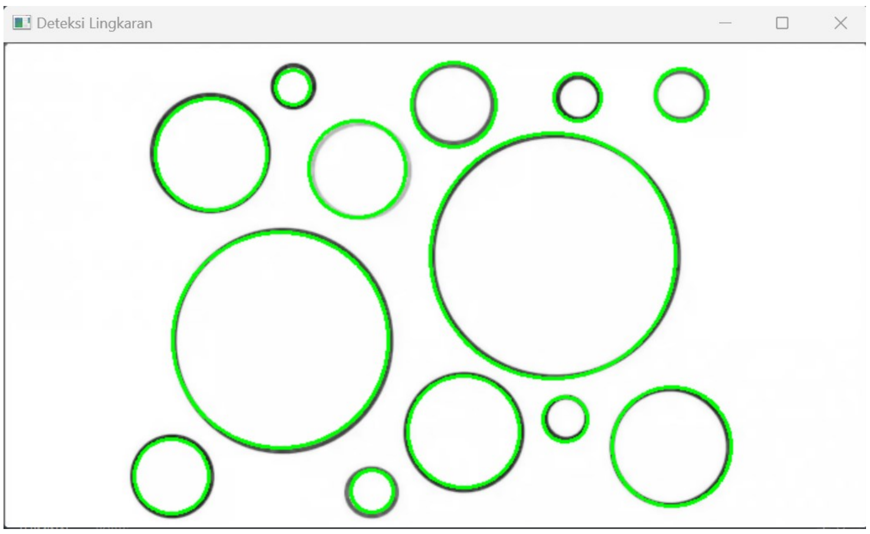


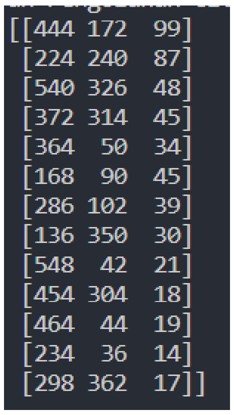


4. **Script program sederhana untuk transformasi hough untuk mendeteksi lingkaran**

```python
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()
```







```
[[[444 172  99]
  [224 240  87]
  [540 326  48]
  [372 314  45]
  [364  50  34]
  [168  90  45]
  [286 102  39]
  [136 350  30]
  [548  42  21]
  [454 304  18]
  [464  44  19]
  [234  36  14]
  [298 362  17]]
```