TEORI VISI KOMPUTER DAN PENGOLAHAN CITRA

S2 TEKNIK INFOMATIKA DAN KOMPUTER

Image Segmentation

NRP	: 1223800002
NAMA	: Nur Rizky Romadhon

Teknik Informatika dan Komputer

Politeknik Elektronika Negeri Surabaya

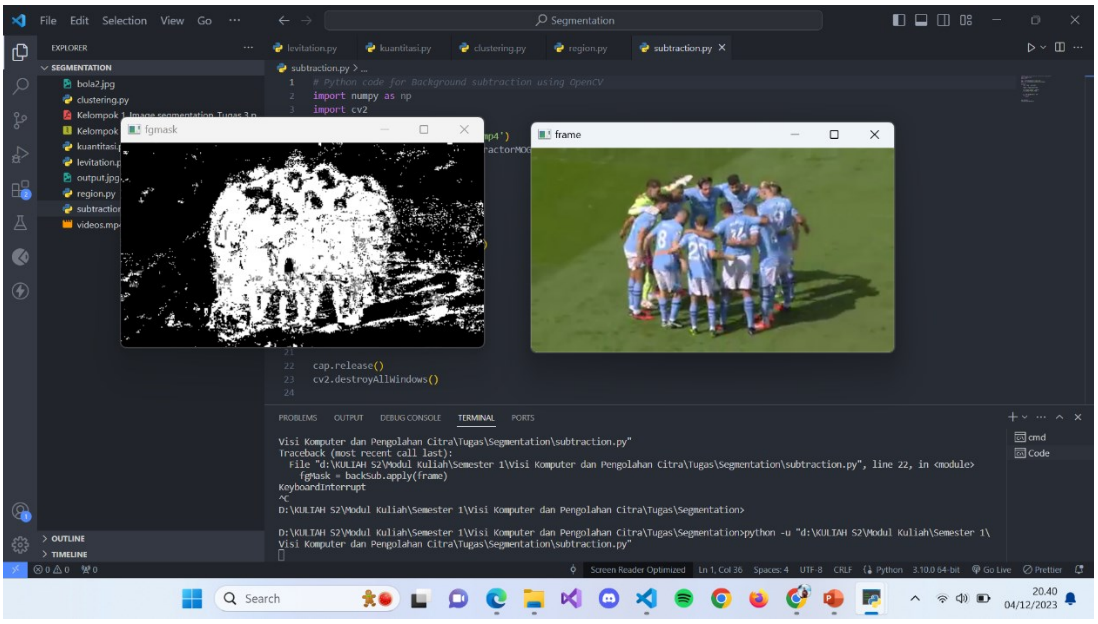
2023

Praktikum Pertemuan 10
Image Segmentation

1. Background Subtraction

```python
# Python code for Background subtraction using OpenCV
import numpy as np
import cv2

cap = cv2.VideoCapture('videos.mp4')
fgbg = cv2.createBackgroundSubtractorMOG2()

while(1):
    ret, frame = cap.read()

    fgmask = fgbg.apply(frame)

    cv2.imshow('fgmask', fgmask)
    cv2.imshow('frame',frame )

    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Output :



```python
# Python code for Background subtraction using OpenCV
import numpy as np
import cv2
```

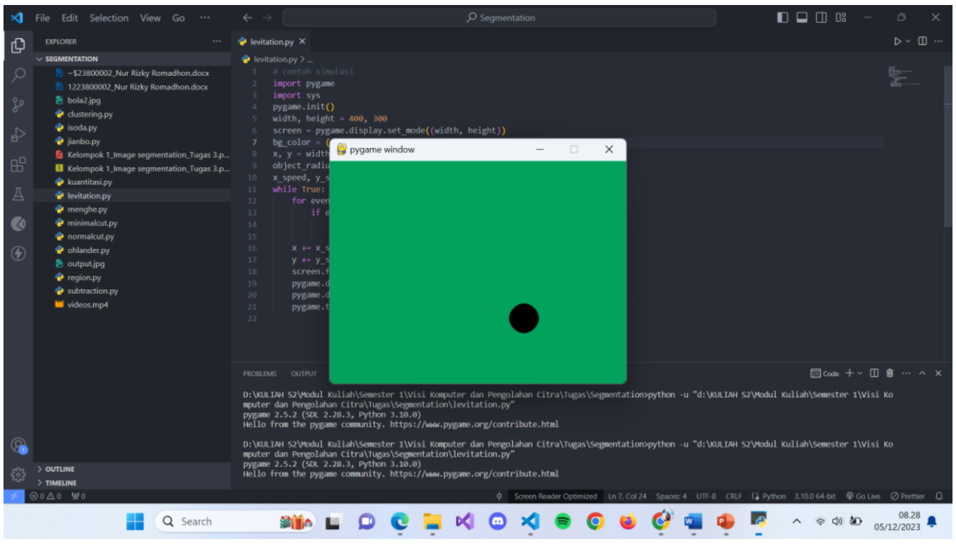
```python
cap.release()
cv2.destroyAllWindows()
```

```
Visi Komputer dan Pengolahan Citra\Tugas\Segmentation\subtraction.py"
Traceback (most recent call last):
  File "d:\KULIAH S2\Modul Kuliah\Semester 1\Visi Komputer dan Pengolahan Citra\Tugas\Segmentation\subtraction.py", line 22, in <module>
    fgMask = backSub.apply(frame)
KeyboardInterrupt
^C
D:\KULIAH S2\Modul Kuliah\Semester 1\Visi Komputer dan Pengolahan Citra\Tugas\Segmentation>

D:\KULIAH S2\Modul Kuliah\Semester 1\Visi Komputer dan Pengolahan Citra\Tugas\Segmentation>python -u "d:\KULIAH S2\Modul Kuliah\Semester 1\
Visi Komputer dan Pengolahan Citra\Tugas\Segmentation\subtraction.py"
```

2. Levitation

```python
# contoh simulasi
import pygame
import sys
pygame.init()
width, height = 400, 300
screen = pygame.display.set_mode((width, height))
bg_color = (0, 162, 92)
x, y = width // 2, height // 2
object_radius = 20
x_speed, y_speed = 1, 1
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    x += x_speed
    y += y_speed
    screen.fill(bg_color)
    pygame.draw.circle(screen, (0, 0, 0), (x, y), object_radius)
    pygame.display.flip()
    pygame.time.delay(10)
```

Output :



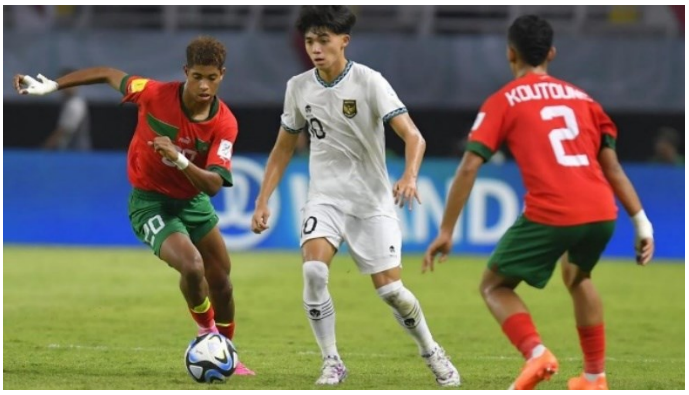
```
D:\KULIAH S2\Modul Kuliah\Semester 1\Visi Komputer dan Pengolahan Citra\Tugas\Segmentation>python -u "d:\KULIAH S2\Modul Kuliah\Semester 1\Visi Ko
mputer dan Pengolahan Citra\Tugas\Segmentation\levitation.py"
pygame 2.5.2 (SDL 2.28.3, Python 3.10.0)
Hello from the pygame community. https://www.pygame.org/contribute.html

D:\KULIAH S2\Modul Kuliah\Semester 1\Visi Komputer dan Pengolahan Citra\Tugas\Segmentation>python -u "d:\KULIAH S2\Modul Kuliah\Semester 1\Visi Ko
mputer dan Pengolahan Citra\Tugas\Segmentation\levitation.py"
pygame 2.5.2 (SDL 2.28.3, Python 3.10.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

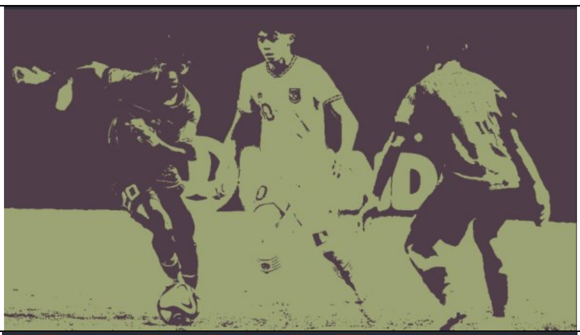
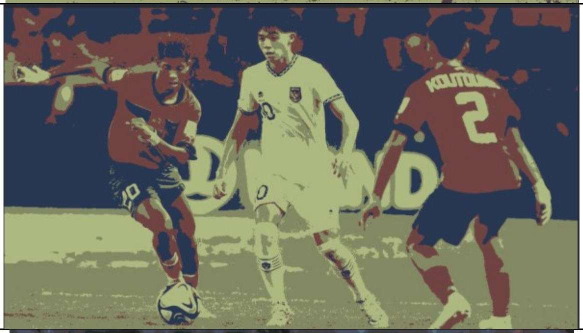
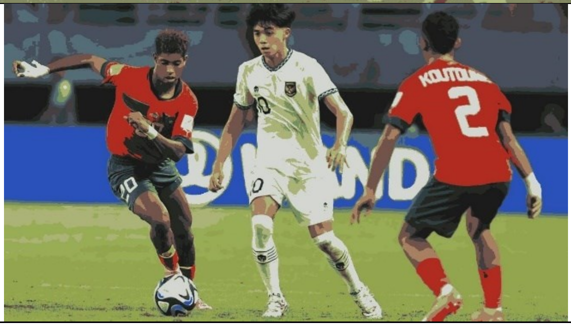
3. Image Kuantisasi

```python
from PIL import Image
def quantize_image(input_image_path, output_image_path,num_colors):
    gambar = Image.open(input_image_path)
    quantized = gambar.convert("P", palette=Image.ADAPTIVE,colors=num_colors)
    quantized = quantized.convert("RGB")
    quantized.save(output_image_path, format="JPEG")
```

```python
if __name__ == "__main__":
    input_image_path = "bola2.jpg"
    output_image_path = "output.jpg"
    num_colors = 16
    quantize_image(input_image_path, output_image_path,num_colors)
```

Input :

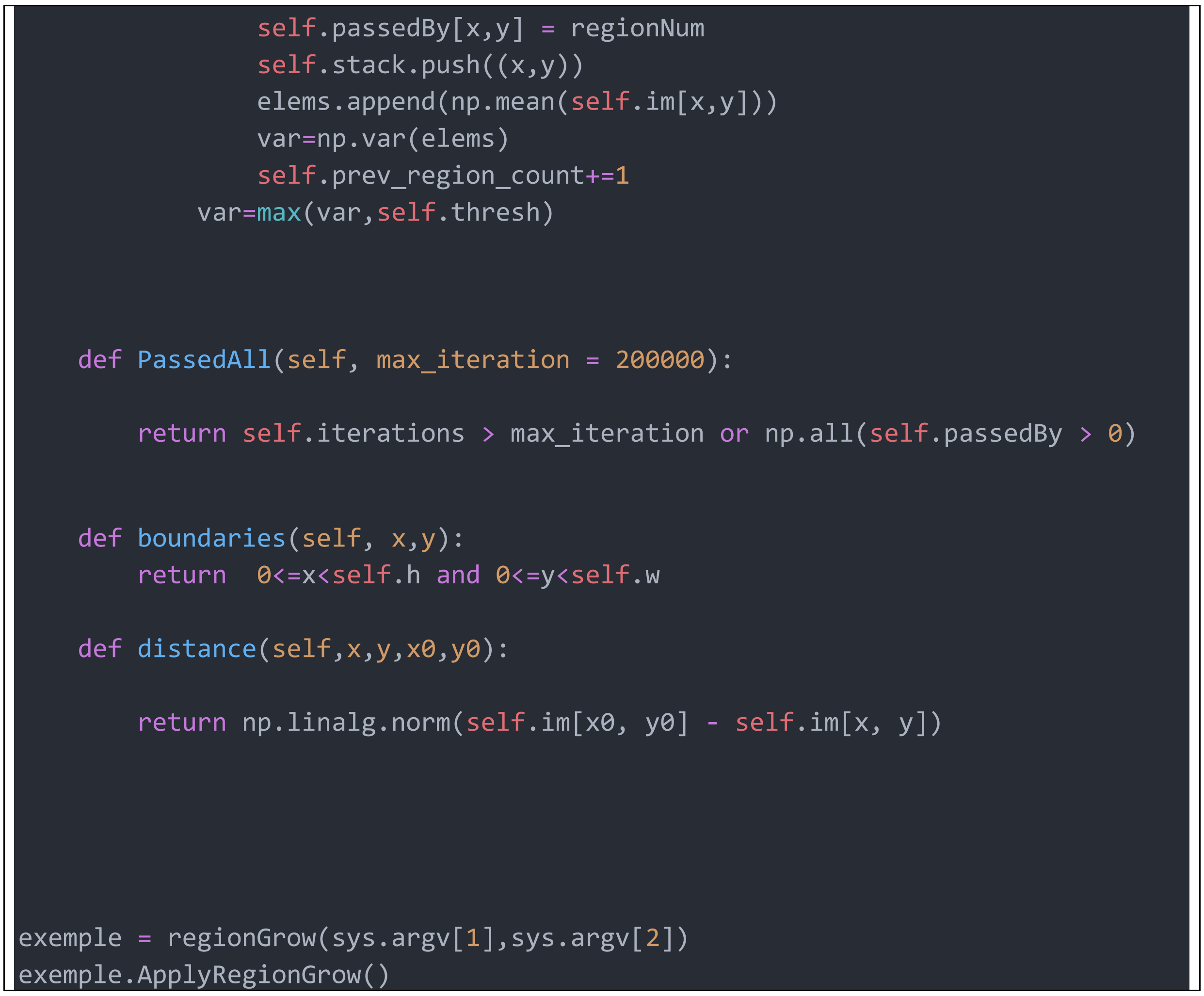


num_color = 2	
num_color = 4	
num_color = 16	

num_color = 256	

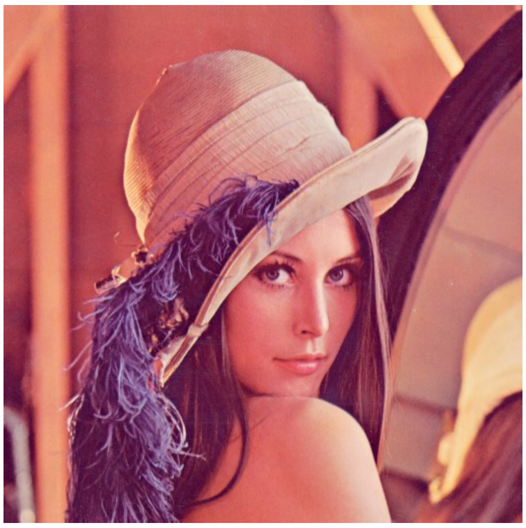
4. Region Growing

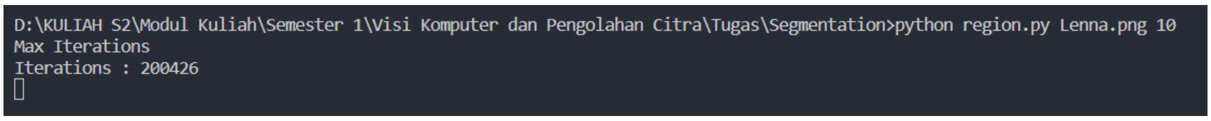
```python
import cv2
import itertools
import numpy as np
import random
import sys

#class pour une pile
class Stack():
    def __init__(self):
        self.item = []
        self.obj=[]
    def push(self, value):
        self.item.append(value)

    def pop(self):
        return self.item.pop()

    def size(self):
        return len(self.item)

    def isEmpty(self):
        return self.size() == 0

    def clear(self):
        self.item = []

class regionGrow():

    def __init__(self,im_path,th):
        self.readImage(im_path)
        self.h, self.w,_ =  self.im.shape
        self.passedBy = np.zeros((self.h,self.w), np.double)
        self.currentRegion = 0
        self.iterations=0
        self.SEGS=np.zeros((self.h,self.w,3), dtype='uint8')
        self.stack = Stack()
        self.thresh=float(th)
    def readImage(self, img_path):
        self.im = cv2.imread(img_path,1).astype('int')


    def getNeighbour(self, x0, y0):
        return [
            (x, y)
            for i, j in itertools.product((-1, 0, 1), repeat=2)
            if (i, j) != (0, 0) and self.boundaries(x := x0 + i, y := y0 + j)
        ]


    def create_seeds(self):
        return [
            [self.h/2,self.w/2],
            [self.h/3,self.w/3],[2*self.h/3,self.w/3],[self.h/3-10,self.w/3],
            [self.h/3,2*self.w/3],[2*self.h/3,2*self.w/3],[self.h/3-
10,2*self.w/3],
            [self.h/3,self.w-10],[2*self.h/3,self.w-10],[self.h/3-10,self.w-
10]
                    ]
    def ApplyRegionGrow(self, cv_display = True):

        randomseeds = self.create_seeds()
        np.random.shuffle(randomseeds)

        for x0 in range (self.h):
            for y0 in range (self.w):

                if self.passedBy[x0,y0] == 0 : #and (np.all(self.im[x0,y0] >
0)) :
                    self.currentRegion += 1
                    self.passedBy[x0,y0] = self.currentRegion
                    self.stack.push((x0,y0))
                    self.prev_region_count= 0

                    while not self.stack.isEmpty():
                        x,y = self.stack.pop()
                        self.BFS(x,y)
                        self.iterations+=1

                    if self.PassedAll():
                        break

                    if self.prev_region_count< 8*8 :
                       x0, y0 = self.reset_region(x0, y0)
        if self.iterations>200000:
            print("Max Iterations")
        print(f"Iterations : {str(self.iterations)}")

        if cv_display:
            [self.color_pixel(i,j) for i, j in
itertools.product(range(self.h), range (self.w))]
            self.display()

    def reset_region(self, x0, y0):

        self.passedBy[self.passedBy==self.currentRegion] = 0
        x0=random.randint(x0-4,x0+4)
        y0=random.randint(y0-4,y0+4)
        x0 = np.clip(x0, 0, self.h - 1)
        y0 = np.clip(y0, 0, self.w - 1)
        self.currentRegion-=1
        return x0, y0

    def color_pixel(self, i, j):
        val = self.passedBy[i][j]
        self.SEGS[i][j] = (255, 255, 255) if (val==0) else (val*35, val*90,
val*30)

    def display(self):
        cv2.imshow("",self.SEGS)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    def BFS(self, x0,y0):

        regionNum = self.passedBy[x0,y0]

        elems = [np.mean(self.im[x0, y0])]

        var=self.thresh

        neighbours=self.getNeighbour(x0,y0)

        for x,y in neighbours:
            if self.passedBy[x,y] == 0 and self.distance(x,y,x0,y0) < var:

                if self.PassedAll():
                    break
```



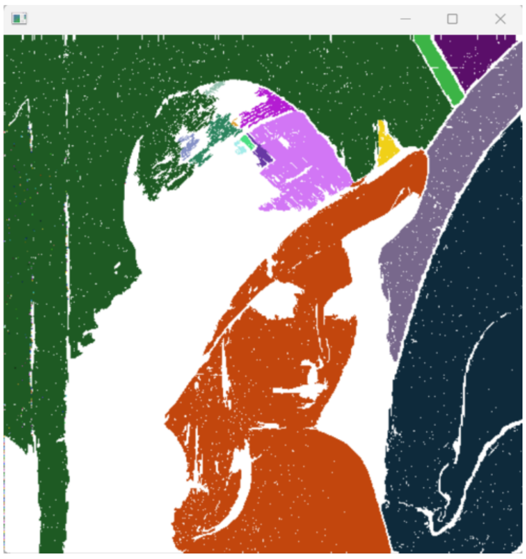
```python
                self.passedBy[x,y] = regionNum
                self.stack.push((x,y))
                elems.append(np.mean(self.im[x,y]))
                var=np.var(elems)
                self.prev_region_count+=1
            var=max(var,self.thresh)



    def PassedAll(self, max_iteration = 200000):

        return self.iterations > max_iteration or np.all(self.passedBy > 0)


    def boundaries(self, x,y):
        return  0<=x<self.h and 0<=y<self.w

    def distance(self,x,y,x0,y0):

        return np.linalg.norm(self.im[x0, y0] - self.im[x, y])





exemple = regionGrow(sys.argv[1],sys.argv[2])
exemple.ApplyRegionGrow()
```

Input :





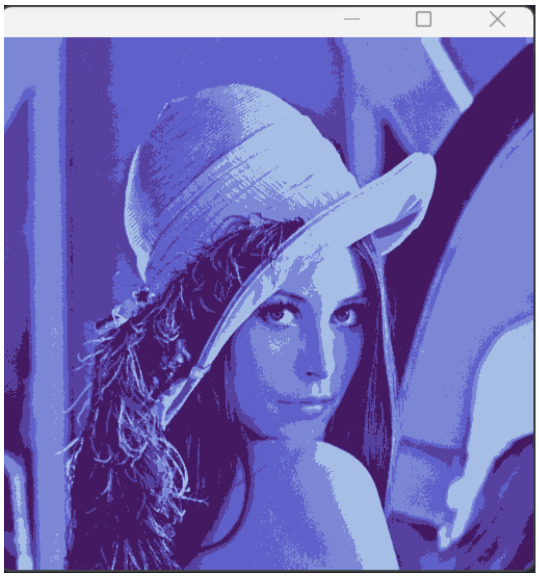
```
D:\KULIAH S2\Modul Kuliah\Semester 1\Visi Komputer dan Pengolahan Citra\Tugas\Segmentation>python region.py Lenna.png 10
Max Iterations
Iterations : 200426
```



5. Clustering

```
# Loading required libraries
import numpy as np
import matplotlib.pyplot as plt
import cv2

# load image from images directory
image = cv2.imread('Lenna.png')

# Change color to RGB (from BGR)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Reshaping the image into a 2D array of pixels and 3 color values (RGB)
pixel_vals = image.reshape((-1,3)) # numpy reshape operation -1 unspecified

# Convert to float type only for supporting cv2.kmean
pixel_vals = np.float32(pixel_vals)

#criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)

# Choosing number of cluster
k = 5

retval, labels, centers = cv2.kmeans(pixel_vals, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)

# convert data into 8-bit values
centers = np.uint8(centers)

segmented_data = centers[labels.flatten()] # Mapping labels to center points(
RGB Value)

# reshape data into the original image dimensions
segmented_image = segmented_data.reshape((image.shape))

cv2.imshow('res2',segmented_image)
plt.imshow(segmented_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

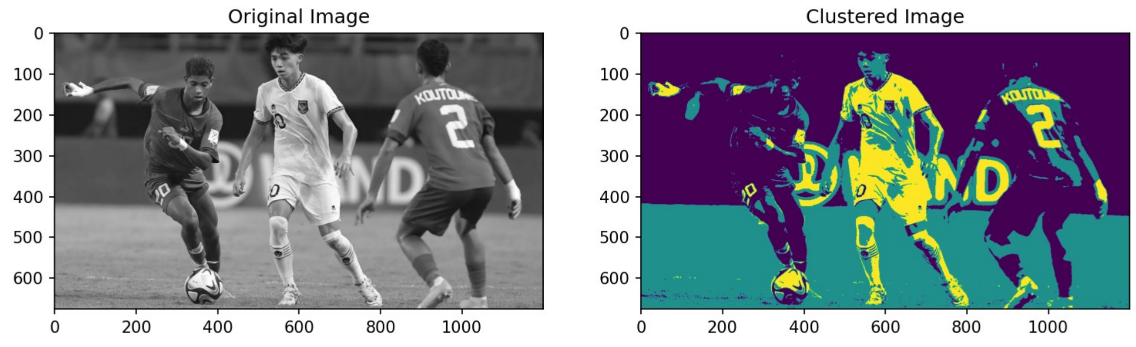
Input : **Output :**



6. Meng-Hee Heng's K-Means

```
import cv2
import numpy as np
from sklearn.cluster import KMeans
from skimage import feature
import matplotlib.pyplot as plt

# Load an image (replace 'your_image.jpg' with the actual image file)
image_path = 'bola2.jpg'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Extract Local Binary Pattern (LBP) features
radius = 3
n_points = 8 * radius
lbp_features = feature.local_binary_pattern(image, n_points, radius,
method="uniform")
```

```
# Reshape the LBP features to a 1D array
lbp_features_1d = lbp_features.flatten()

# Reshape the image to a 1D array for clustering
image_1d = image.flatten()

# Concatenate LBP features with pixel values
data = np.column_stack((image_1d, lbp_features_1d))

# Apply K-Means clustering
num_clusters = 3  # Adjust the number of clusters as needed
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
labels = kmeans.fit_predict(data)

# Reshape the labels to the shape of the original image
clustered_image = labels.reshape(image.shape)

# Display the original and clustered images
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.imshow(image, cmap='gray')
ax1.set_title('Original Image')

ax2.imshow(clustered_image, cmap='viridis')
ax2.set_title('Clustered Image')

plt.show()
```
Output :

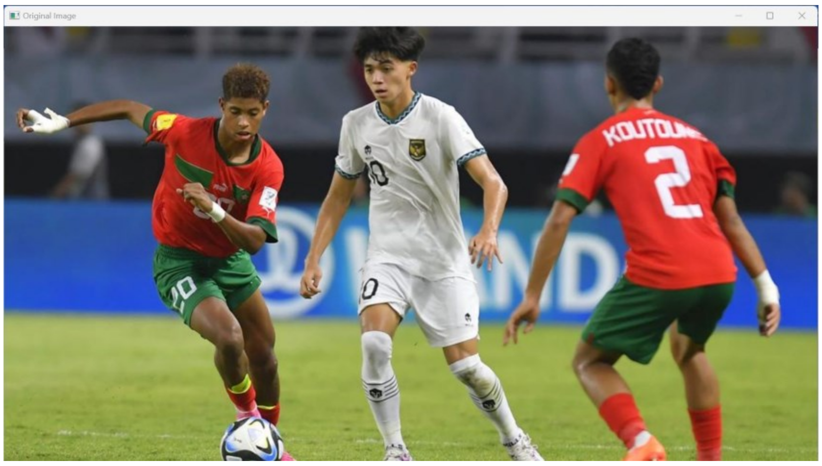


7. Isodata Clusturing

```
import numpy as np
import cv2
from sklearn.cluster import KMeans

# Load an image (replace 'your_image.jpg' with the actual image file)
image_path = 'bola2.jpg'
```

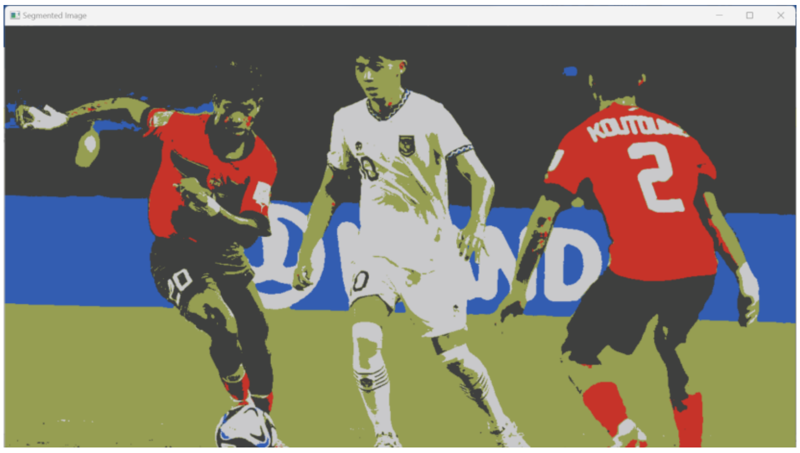
```
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Reshape the image to a 2D array of pixels
pixels = image.reshape((-1, 3))

# Apply ISODATA clustering (K-Means with dynamic cluster adaptation)
def isodata_clustering(data, max_clusters, min_samples=5, max_iter=100):
    kmeans = KMeans(n_clusters=max_clusters, random_state=42)
    kmeans.fit(data)
    labels = kmeans.labels_
    centers = kmeans.cluster_centers_

    for _ in range(max_iter):
        cluster_sizes = np.bincount(labels)
        mask = cluster_sizes > min_samples
        if mask.sum() == 0:
            break

        kmeans = KMeans(n_clusters=mask.sum(), random_state=42,
init=centers[mask])
        kmeans.fit(data)
        labels = kmeans.labels_
        centers = kmeans.cluster_centers_

    return labels, centers

# Apply ISODATA clustering
num_clusters = 5  # Adjust the number of clusters as needed
isodata_labels, isodata_centers = isodata_clustering(pixels, num_clusters)

# Replace each pixel with its corresponding cluster center
segmented_image = isodata_centers[isodata_labels].reshape(image.shape)

# Display the original and segmented images
cv2.imshow('Original Image', cv2.cvtColor(image, cv2.COLOR_RGB2BGR))
cv2.imshow('Segmented Image', cv2.cvtColor(segmented_image.astype(np.uint8),
cv2.COLOR_RGB2BGR))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input :

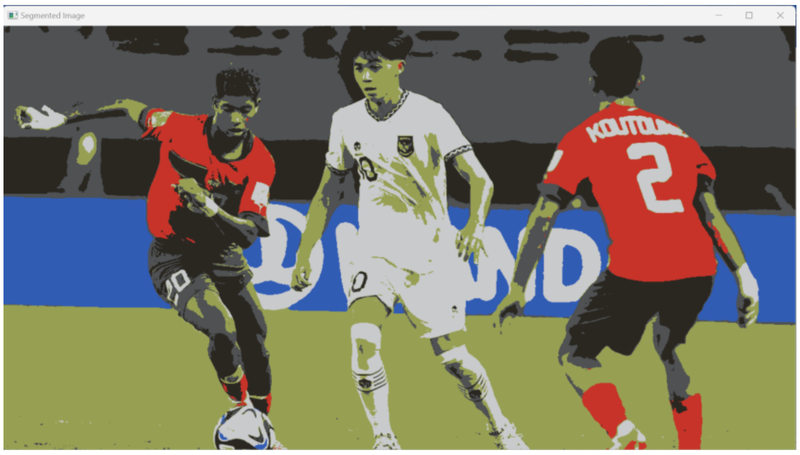


Output :

K= 5

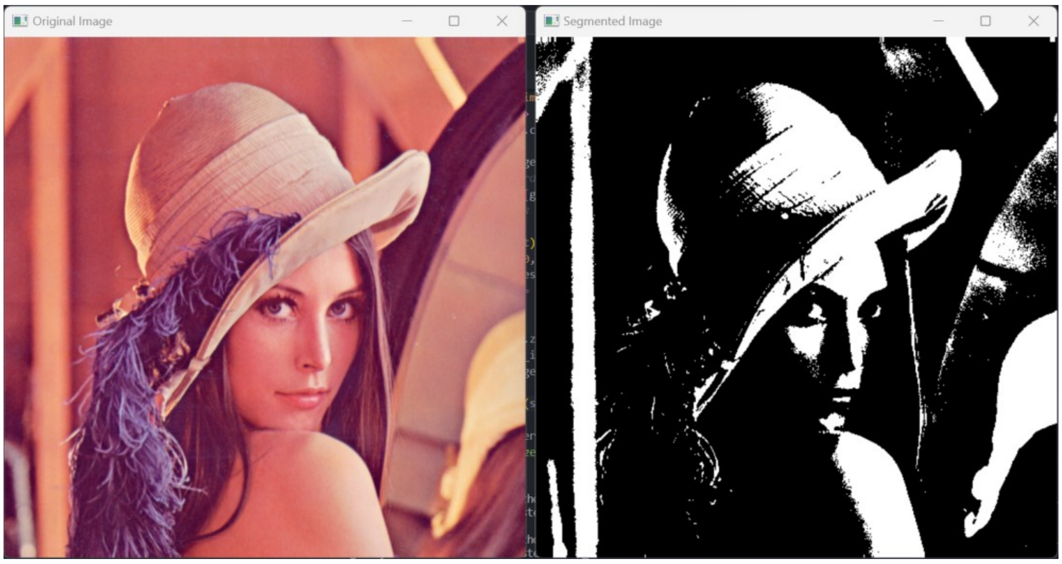


K=6



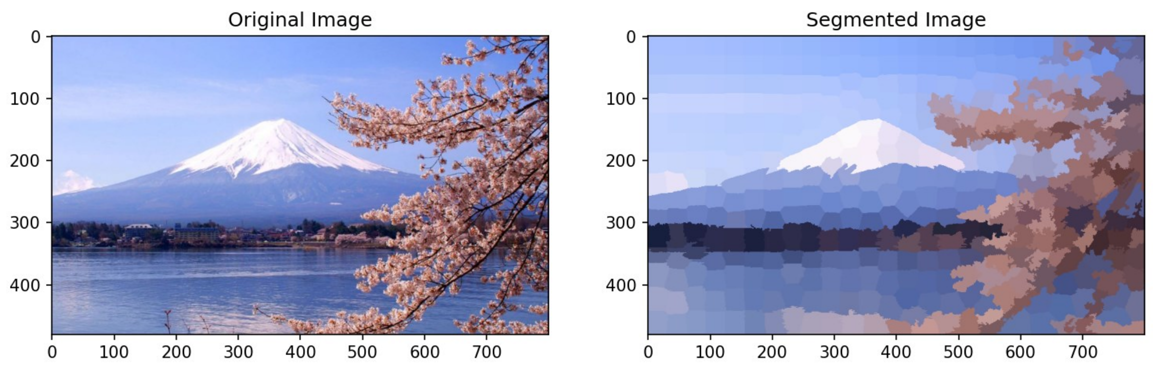
8. Ohlander's Recursive Histogram-Based Clustering

```python
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]+'/Lenna.png')
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()
```



9. Jianbo Shi's Graph-Partitioning

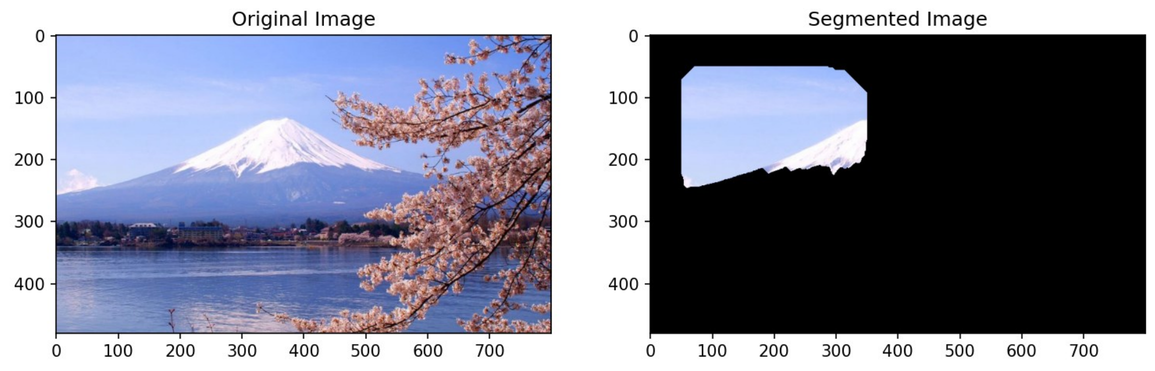
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, segmentation, color, io

# Load an image (replace 'your_image.jpg' with the actual image file)
image_path = image1.jpg'
image = io.imread(image_path)

# Convert the image to a 2D array (graph representation)
graph = color.rgb2gray(image)

# Apply Normalized Cut for image segmentation
labels = segmentation.slic(image, compactness=30, n_segments=400)
segmented_image = color.label2rgb(labels, image, kind='avg')

# Display the original and segmented images
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.imshow(image)
ax1.set_title('Original Image')

ax2.imshow(segmented_image)
ax2.set_title('Segmented Image')

plt.show()
```



10. Minimal Cuts

Program berikut ini menunjukkan cara menggambar sebuah object (lines dan shapes) dan menulis (text) pada window visualisasi.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Load an image (replace 'your_image.jpg' with the actual image file)
image_path = image1.jpg'
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Create a mask (0: sure background, 2: sure foreground)
mask = np.zeros(image.shape[:2], np.uint8)

# Define a rectangle around the object of interest (rect = (start_x, start_y,
width, height))
rect = (50, 50, 300, 200)

# Initialize the background and foreground models
bgd_model = np.zeros((1, 65), np.float64)
fgd_model = np.zeros((1, 65), np.float64)

# Apply GrabCut algorithm
cv2.grabCut(image, mask, rect, bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)

# Modify the mask to get the segmented image
segmentation_mask = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
segmented_image = image * segmentation_mask[:, :, np.newaxis]

# Display the original and segmented images
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.imshow(image)
ax1.set_title('Original Image')

ax2.imshow(segmented_image)
ax2.set_title('Segmented Image')

plt.show()
```



11. Normalized Cut

```python
import numpy as np
import matplotlib.pyplot as plt
from skimage import segmentation, color, io

# Load an image (replace 'your_image.jpg' with the actual image file)
image_path = 'image1.jpg'
image = io.imread(image_path)

# Convert the image to a 2D array (graph representation)
graph = color.rgb2gray(image)

# Apply Normalized Cut for image segmentation
labels = segmentation.slic(image, compactness=30, n_segments=400)
segmented_image = color.label2rgb(labels, image, kind='avg')

# Display the original and segmented images
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.imshow(image)
ax1.set_title('Original Image')

ax2.imshow(segmented_image)
ax2.set_title('Segmented Image')

plt.show()
```