

**TUGAS 3**  
**VISI KOMPUTER DAN PENGOLAHAN CITRA**



**Disusun Oleh Kelompok 1 :**

|                                    |                     |
|------------------------------------|---------------------|
| <b>Silfiana Nur Hamida</b>         | <b>(1223800005)</b> |
| <b>Firnanda Pristiana N</b>        | <b>(1222800004)</b> |
| <b>Ambarwati Rizkia Putri</b>      | <b>(1123800008)</b> |
| <b>Mochamad Riswandha Lazuardi</b> | <b>(1123800006)</b> |

**Membahas tentang:**  
***“Image Segmentation”***

**PROGRAM PASCASARJANA TEKNIK INFORMATIKA DAN KOMPUTER**  
**POLITEKNIK ELEKTRONIKA NEGERI SURABAYA**

**2023/2024**

## A. Levitation

Pada implementasi di bawah ini saya mencoba menggunakan library pygame untuk melakukan simulasi levitation, dimana levitation ini adalah suatu teknik yang digunakan dalam fotografi kreatif dan seni digital untuk menciptakan suatu gambar yang seolah olah melayang di udara. Dan berikut ini merupakan implementasi codenya.

- Source Code

```
# contoh simulasi

import pygame
import sys

pygame.init()
width, height = 400, 300
screen = pygame.display.set_mode((width, height))
bg_color = (0, 255, 255)

x, y = width // 2, height // 2
object_radius = 20

x_speed, y_speed = 1, 1

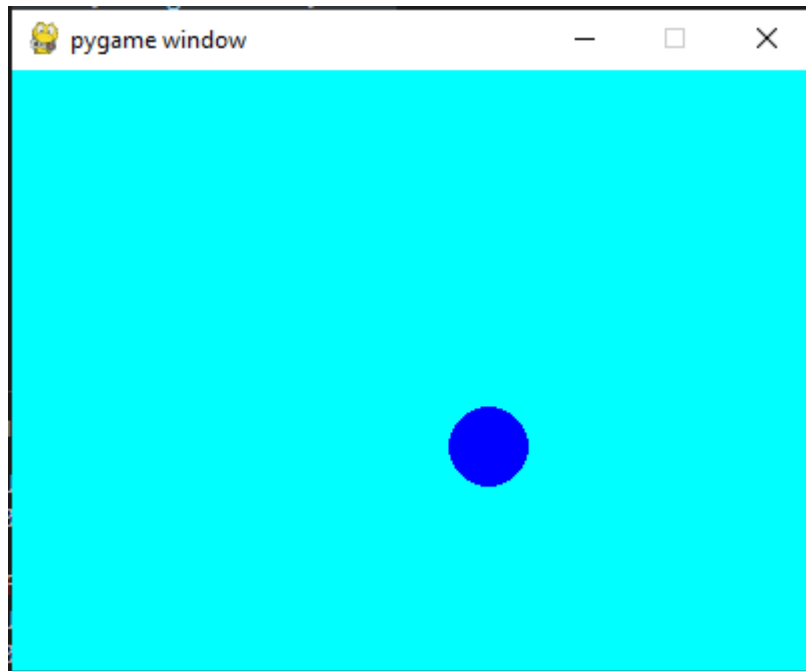
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    x += x_speed
    y += y_speed
    screen.fill(bg_color)
    pygame.draw.circle(screen, (0, 0, 255), (x, y), object_radius)

    pygame.display.flip()

    pygame.time.delay(10)
```

- **Output Program**



## B. Image Kuantisasi

Pada percobaan kodingan di bawah ini saya menggunakan teknik kuantisasi yang dimana teknik akan membuat skala dari nilai r,g dan b menjadi lebih kecil sehingga jumlah warna yang ada pada sebuah image akan menjadi lebih sedikit.

- **Source Code**

```
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")

if __name__ == "__main__":
    input_image_path = "download.jpg"
    output_image_path = "skala256.jpg"
    num_colors = 256

    quantize_image(input_image_path, output_image_path,
num_colors)
```

- **Output Program**

- Gambar Asli



- Ketika menggunakan num\_color = 2



- Ketika menggunakan num\_color = 4



- Ketika menggunakan num\_color = 16



- Ketika menggunakan num\_color = 256



### C. Region Growing

Teknik Region Growing ini dimulai dari satu titik di area potensial, Teknik ini berfokus pada identifikasi dan pengelompokan piksel (pixel) yang memiliki karakteristik atau sifat serupa ke dalam wilayah (region) yang lebih besar dan berikut ini implementasi source code dari teknik Region Growing.

- Source Code

```
import cv2
import numpy as np

def region_growing(image, seed):
    tolerance = 10
    rows, cols = image.shape
    visited = np.zeros_like(image, dtype=np.uint8)
    stack = []

    result = np.zeros_like(image)
    def is_within_tolerance(p1, p2):
        return abs(int(p1) - int(p2)) < tolerance

    stack.append(seed)
    visited[seed] = 1
    while len(stack) > 0:
        current_point = stack.pop()
        result[current_point] = image[current_point]

        for i in range(-1, 2):
            for j in range(-1, 2):
                if current_point[0] + i >= 0 and
current_point[1] + j >= 0 and current_point[0] + i < rows and
current_point[1] + j < cols:
                    if visited[current_point[0] + i,
current_point[1] + j] == 0 and
is_within_tolerance(image[current_point], image[current_point[0]
+ i, current_point[1] + j]):
                        stack.append((current_point[0] + i,
current_point[1] + j))
                        visited[current_point[0] + i,
```

```
current_point[1] + j] = 1

    return result

if __name__ == "__main__":
    image = cv2.imread("kayu.jpg", 0)
    seed = (50, 50)

    result = region_growing(image, seed)

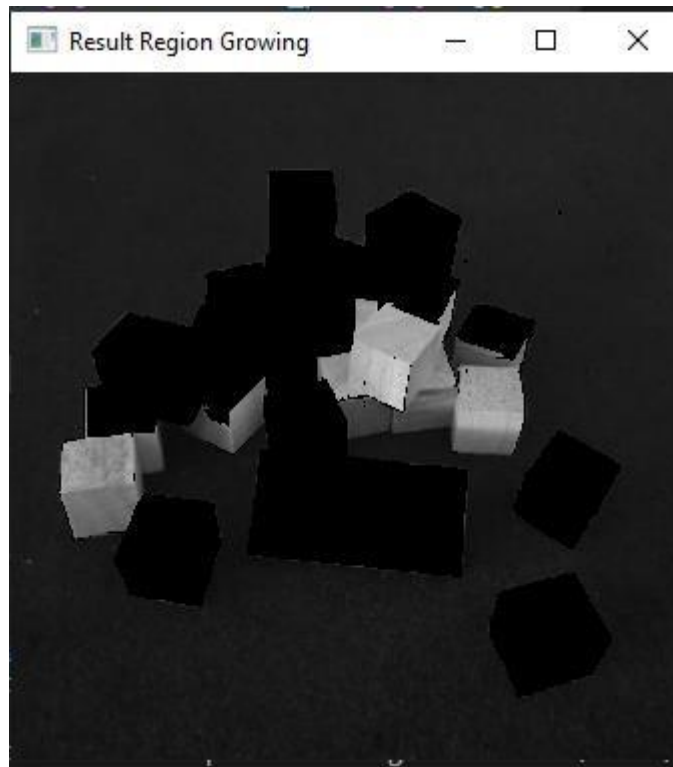
    cv2.imshow("Result Region Growing", result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

- **Output Program**

- Gambar sebelum dilakukan teknik region growing



- Gambar setelah dilakukan teknik region growing



#### D. Clustering

Pada teknik clustering ini saya mencoba menggunakan K-Means, tujuannya untuk mengidentifikasi struktur, pola, atau objek dalam citra dengan mengelompokkan piksel-piksel yang mirip berdasarkan properti tertentu seperti intensitas warna atau tekstur. Dan berikut ini hasil implementasinya.

- **Source Code**

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_sample_image
from sklearn.utils import shuffle
from PIL import Image
gambar = np.array(Image.open('favorit.jpg'))

gambar = gambar / 255.0

w, h, d = original_shape = tuple(gambar.shape)
assert d == 3

image_array = np.reshape(gambar, (w * h, d))

n_colors = 3
kmeans = KMeans(n_clusters=n_colors,
random_state=0).fit(image_array)
labels = kmeans.predict(image_array)
centers = kmeans.cluster_centers_
```

```

compressed_image = centers[labels].reshape(gambar.shape)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.axis('off')
plt.title('Citra Gambar Asli')
plt.imshow(gambar)

plt.subplot(1, 2, 2)
plt.axis('off')
plt.title('Citra Gambar Menggunakan K-means')
plt.imshow(compressed_image)
plt.show()

```

- **Output Program**



### E. Meng-Hee Heng's K-means

Pada teknik Meng-Hee Heng's K-Means ini merupakan salah satu variasi K-Means yang memperkenalkan beberapa perubahan dalam algoritma dasar K-Means untuk meningkatkan hasil segmentasi pada citra. Berikut ini merupakan implementasi codenya.

- **Source Code**

```

import numpy as np
from sklearn.cluster import KMeans
from skimage.feature import graycomatrix, graycoprops
from skimage import io, color, img_as_ubyte

def extract_texture_features(image_path):
    image = io.imread(image_path)
    gray_image = img_as_ubyte(color.rgb2gray(image))
    glcm = graycomatrix(gray_image, [1], [0], symmetric=True,
normed=True)

```





```

        centroids = new_centroids
        cluster_variances = np.array([data[labels == i].var() for i
in range(num_clusters)])
        mean_variance = cluster_variances.mean()
        low_variance_clusters = np.where(cluster_variances <
max_variance * mean_variance)[0]
        for cluster in low_variance_clusters:
            labels[labels == cluster] = labels[labels ==
cluster].min()
        segmented_image = labels.reshape(m, n)

    return segmented_image

if __name__ == "__main__":
    image = io.imread('favorit.jpg')
    num_clusters = 3
    max_iterations = 100
    min_samples = 0.1
    max_variance = 0.5

    segmented_image = isodata_segmentation(image, num_clusters,
max_iterations, min_samples, max_variance)
    plt.figure(figsize=(8, 4))
    plt.subplot(121)
    plt.imshow(image, cmap='gray')
    plt.axis('off')
    plt.title('Gambar Citra Asli')

    plt.subplot(122)
    plt.imshow(segmented_image, cmap='nipy_spectral')
    plt.axis('off')
    plt.title('Hasil Gambar Segmentasi ISODATA')

    plt.show()

```

- **Output Program**



### G. Ohlander's Recursive Histogram-Based Clustering

Pada percobaan menggunakan teknik Ohlander's Recursive Histogram-Based Clustering yang dimana teknik ini memanfaatkan representasi histogram citra untuk melakukan clustering. Dan berikut ini merupakan code hasil dari implementasinya.

- Source Code

```
import cv2
import numpy as np
image = cv2.imread('favorit.jpg')
lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2Lab)
l_channel, a_channel, b_channel = cv2.split(lab_image)
hist_l = cv2.calcHist([l_channel], [0], None, [256], [0, 256])
hist_a = cv2.calcHist([a_channel], [0], None, [256], [0, 256])
hist_b = cv2.calcHist([b_channel], [0], None, [256], [0, 256])
def histogram_distance(hist1, hist2):
    return cv2.compareHist(hist1, hist2,
cv2.HISTCMP_BHATTACHARYYA)
histograms_and_masks = [(hist_l, l_channel), (hist_a,
a_channel), (hist_b, b_channel)]
def recursive_clustering(hist_and_mask_list, cluster_list):
    if len(hist_and_mask_list) == 0:
        return
    max_distance = -1
    cluster_index = -1
    for i in range(len(hist_and_mask_list)):
        for j in range(i + 1, len(hist_and_mask_list)):
            distance =
histogram_distance(hist_and_mask_list[i][0],
hist_and_mask_list[j][0])
            if distance > max_distance:
                max_distance = distance
```

```

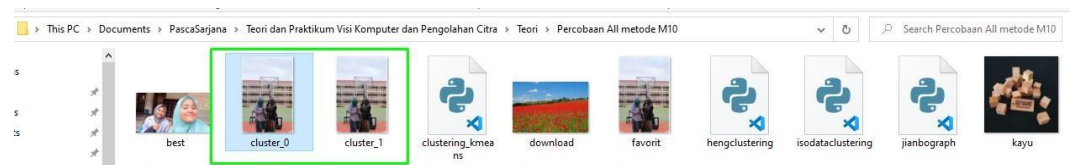
        cluster_index = (i, j)

        if cluster_index != -1:
            i, j = cluster_index
            new_histogram = hist_and_mask_list[i][0] +
hist_and_mask_list[j][0]
            new_mask = np.maximum(hist_and_mask_list[i][1],
hist_and_mask_list[j][1])
            del hist_and_mask_list[j]
            del hist_and_mask_list[i]
            hist_and_mask_list.append((new_histogram, new_mask))
            cluster_list.append(new_mask)
            recursive_clustering(hist_and_mask_list, cluster_list)
clusters = []
recursive_clustering(histograms_and_masks, clusters)

for i, cluster in enumerate(clusters):
    result_image = cv2.bitwise_and(image, image, mask=cluster)
    cv2.imwrite(f'cluster_{i}.jpg', result_image)

```

- **Output Program**



## H. Jianbo Shi's Graph-Partitioning

Pada percobaan dengan menggunakan teknik Jianbo Shi's Graph-Partitioning yang merupakan teknik segmentasi berbasis grafik yang mengandalkan pemisahan piksel menjadi segmen dengan mengoptimalkan pemotongan normalized pada grafik yang dihasilkan dari citra. Dan berikut ini merupakan implementasi dari teknik tersebut.

- **Source Code**

```

import networkx as nx
import numpy as np
from sklearn.cluster import spectral_clustering

G = nx.karate_club_graph()

matrix_conv = nx.to_numpy_matrix(G)

array_conv = np.array(matrix_conv)

num_clusters = 2
labels = spectral_clustering(array_conv,
n_clusters=num_clusters)

```

```
# Hasil partisi graf
for node, label in enumerate(labels):
    print(f"Node {node} berada di Kelompok {label}")
```

- **Output Program**

```
PS C:\Users\ASUS\Documents\PascaSarjana\Teori dan Praktikum Visi Komputer dan Pengolahan Citra\Teori\Percobaan All met
ode M10> python .\jianbograph.py
Node 0 berada di Kelompok 1
Node 1 berada di Kelompok 1
Node 2 berada di Kelompok 0
Node 3 berada di Kelompok 1
Node 4 berada di Kelompok 1
Node 5 berada di Kelompok 1
Node 6 berada di Kelompok 1
Node 7 berada di Kelompok 1
Node 8 berada di Kelompok 0
Node 9 berada di Kelompok 0
Node 10 berada di Kelompok 1
Node 11 berada di Kelompok 1
Node 12 berada di Kelompok 1
Node 13 berada di Kelompok 1
Node 14 berada di Kelompok 0
Node 15 berada di Kelompok 0
Node 16 berada di Kelompok 1
Node 17 berada di Kelompok 1
Node 18 berada di Kelompok 0
Node 19 berada di Kelompok 1
Node 20 berada di Kelompok 0
Node 21 berada di Kelompok 1
Node 22 berada di Kelompok 0
Node 23 berada di Kelompok 0
Node 24 berada di Kelompok 0
Node 25 berada di Kelompok 0
```

## I. Minimal Cuts

Pada percobaan dengan menggunakan teknik minimal cuts yang merupakan teknik yang digunakan dalam pengolahan citra untuk segmentasi citra dalam konteks tidak mencari minimal cuts melainkan mencari pemotongan yang dihasilkan segmen yang seimbang dan saling berkaitan. Dan berikut ini merupakan hasil dari implementasi code yang menggunakan teknik tersebut.

- **Source Code**

```
import random

def min_cut(graph):
    while len(graph) > 2:
        u = random.choice(list(graph.keys()))
        v = random.choice(graph[u])
        graph[u].extend(graph[v])

        for node in graph[v]:
            graph[node].remove(v)
            graph[node].append(u)
        del graph[v]
    min_cuts = len(list(graph.values())[0])
    return min_cuts

graph = {
    'A': ['B', 'C'],
```

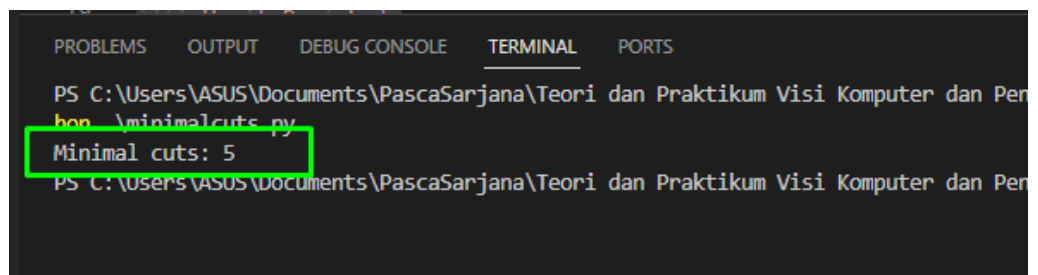
```

    'B': ['A', 'C', 'D'],
    'C': ['A', 'B', 'D'],
    'D': ['B', 'C']
}

# Contoh pengujian
min_cut_result = min_cut(graph)
print("Minimal cuts:", min_cut_result)

```

- **Output Program**



```

PS C:\Users\ASUS\Documents\PascaSarjana\Teori dan Praktikum Visi Komputer dan Per...
> python minimalcuts.py
Minimal cuts: 5
PS C:\Users\ASUS\Documents\PascaSarjana\Teori dan Praktikum Visi Komputer dan Per...

```

## J. Normalized Cut

Pada percobaan dengan menggunakan teknik mormalized cuts yang bertujuan untukmembagi citra menjadi segmen atau wilayah yang memiliki kesamaan dalam suatu fitur dan kesamaan dalam konteks lingkunagn sekitarnya. Dan berikut ini merupakan hasil dari implementasi code yang menggunakan teknik tersebut.

- **Source Code**

```

import numpy as np
from skimage import io
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt

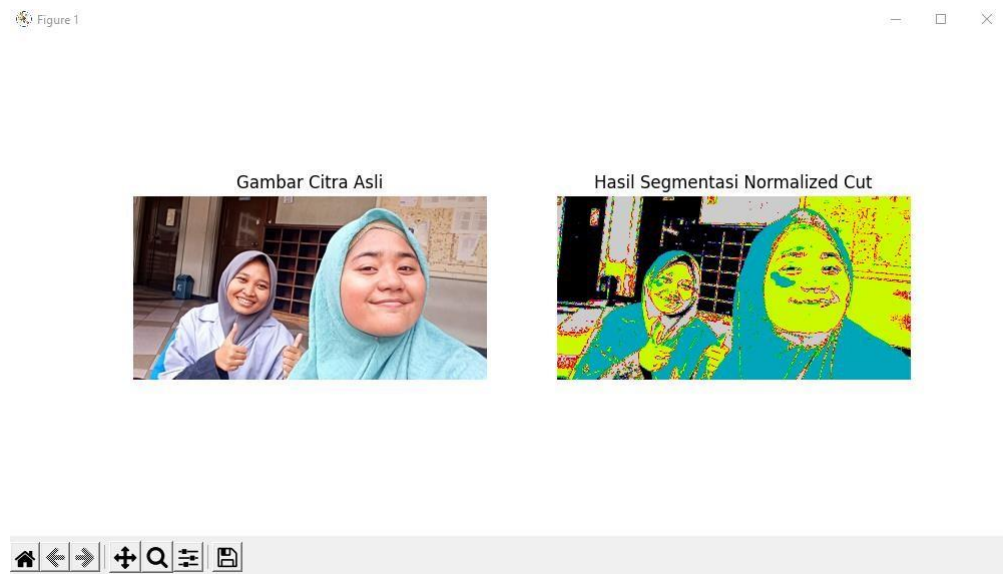
# Membaca citra
image = io.imread("best.jpg")
height, width, _ = image.shape
image_2d = image.reshape(-1, 3)
num_segments = 4
kmeans = KMeans(n_clusters=num_segments,
random_state=0).fit(image_2d)
labels = kmeans.labels_
segmented_image = labels.reshape(height, width)
plt.figure(figsize=(10, 5))
plt.subplot(121)
plt.imshow(image)
plt.axis('off')
plt.title('Gambar Citra Asli')

```

```
plt.subplot(122)
plt.imshow(segmented_image, cmap='nipy_spectral')
plt.axis('off')
plt.title('Hasil Segmentasi Normalized Cut')

plt.show()
```

- **Output Program**



## K. Shi Clustering

Pada percobaan dengan menggunakan teknik shi clustering yang bertujuan untuk mendeteksi sudut dalam citra. Dan berikut ini merupakan hasil dari implementasi code yang menggunakan teknik tersebut.

- **Source Code**

```
import cv2
import numpy as np

image = cv2.imread('download.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
corners = cv2.goodFeaturesToTrack(gray, maxCorners=100,
qualityLevel=0.01, minDistance=10)
corners = np.int0(corners)
for corner in corners:
    x, y = corner.ravel()
    cv2.circle(image, (x, y), 3, 255, -1)
cv2.imshow('Hasil Deteksi Corner', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Output Program**

