LAPORAN
Visi Komputer dan Pengolahan Citra

Image Segmentation

Disusun oleh:

1123800010 - Ahmada Haiz Zakiyil Ilahi
1123800012 - Zulfikar Davbi Mahendra F
1222800008 - Achmad Torikul Huda
1222800012 - Rudi Kurniawan

PROGRAM PASCASARJANA TERAPAN

POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

2023/2024

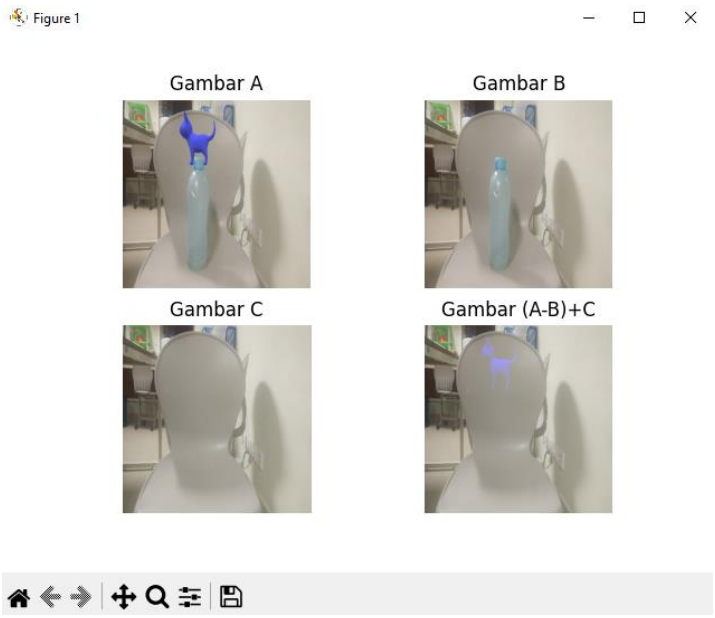
IMAGE SEGMENTATION

1) Levitation

- **Kode Program:**

```
import cv2
import matplotlib.pyplot as plt


# Baca gambar A, B, dan C
image_A = cv2.imread('imgA.jpg')
image_B = cv2.imread('imgB.jpg')
image_C = cv2.imread('imgA-B.jpg')

# Resize semua gambar menjadi dimensi yang lebih kecil
new_dimensions = (500, 500)  # Ganti dimensi ini sesuai kebutuhan

image_A_resized = cv2.resize(image_A, new_dimensions)
image_B_resized = cv2.resize(image_B, new_dimensions)
image_C_resized = cv2.resize(image_C, new_dimensions)

# Hitung X = (A - B) + C pada gambar yang sudah diresize
image_X = cv2.add(cv2.subtract(image_A_resized, image_B_resized),
image_C_resized)

# Menampilkan gambar dalam subplot 2 baris 2 kolom
plt.subplot(2, 2, 1)
plt.imshow(cv2.cvtColor(image_A_resized, cv2.COLOR_BGR2RGB))
plt.title('Gambar A')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(cv2.cvtColor(image_B_resized, cv2.COLOR_BGR2RGB))
plt.title('Gambar B')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(cv2.cvtColor(image_C_resized, cv2.COLOR_BGR2RGB))
plt.title('Gambar C')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(cv2.cvtColor(image_X, cv2.COLOR_BGR2RGB))
plt.title('Gambar (A-B)+C')
plt.axis('off')

plt.show()
```

- **Output**:



- **Analisa:**

  Program tersebut adalah implementasi operasi aritmetika pada gambar menggunakan Python dengan library OpenCV dan Matplotlib. Pertama, tiga gambar (A, B, dan C) dibaca menggunakan OpenCV. Selanjutnya, gambar-gambar tersebut diubah ukurannya menjadi dimensi yang lebih kecil sesuai dengan nilai yang ditentukan. Proses selanjutnya melibatkan operasi aritmetika pada gambar yang sudah diresize, dengan mengurangkan gambar B dari gambar A, dan hasilnya ditambahkan dengan gambar C. Hasil operasi ini disimpan dalam variabel image_X. Kemudian, keseluruhan gambar (A, B, C, dan X) ditampilkan dalam subplot 2 baris dan 2 kolom menggunakan Matplotlib. Setiap subplot menampilkan gambar yang sesuai dengan judul dan sumbu yang dinonaktifkan untuk memberikan presentasi visual yang lebih bersih. Dengan demikian, program ini memberikan ilustrasi tentang bagaimana melakukan manipulasi gambar menggunakan operasi aritmetika dengan bantuan OpenCV dan memvisualisasikannya menggunakan Matplotlib.

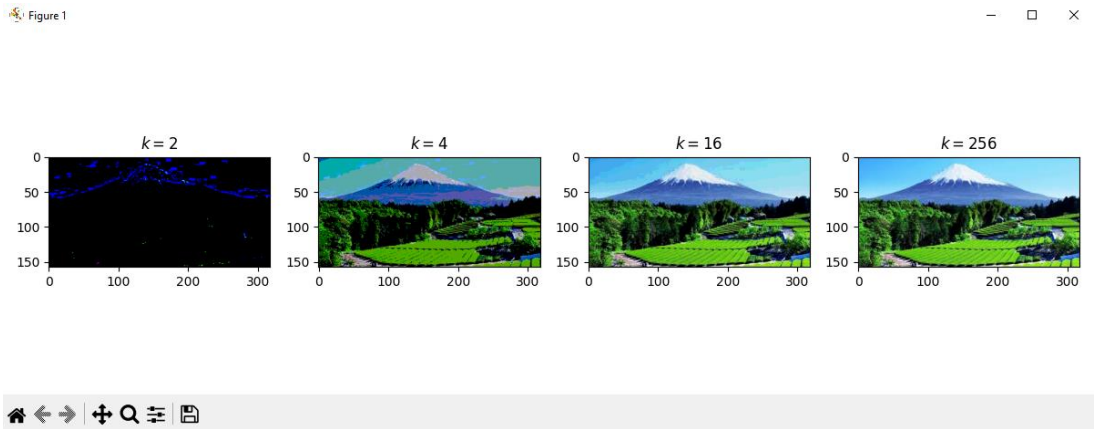
2) Image Kuantisasi

- **Kode Program:**

```
import matplotlib.pyplot as plt
import numpy as np
from skimage.io import imshow, imread


def quantize_image(image_path, k_values):
    # Load the original image
```

```
    img = imread(image_path)

    # Create subplots for different k values
    fig, ax = plt.subplots(1, len(k_values), figsize=(12, 4))

    for i, k in enumerate(k_values):
        # Create k bins of equal width between 0 and the maximum
intensity value
        bins = np.linspace(0, img.max(), k)

        # Map the pixel values of the original image to the nearest
bin
        quantized_image = np.digitize(img, bins)

        # Convert the binned values back to the original range of
intensity values
        reconstructed_image =
(np.vectorize(bins.tolist().__getitem__)(quantized_image-
1).astype(int))

        # Display the quantized image with title showing the number
of bins (k)
        ax[i].imshow(reconstructed_image)
        ax[i].set_title(r'$k = %d$' % k)

    # Adjust the layout of the subplots to prevent overlap
    plt.tight_layout()

    # Show the final plot
    plt.show()

# Quantize the image
k_values = [2,  4, 16, 256]
quantize_image('gambar2.png', k_values)
```

- **Output**:



- **Analisa:**

  Program tersebut merupakan implementasi dari metode kuantisasi citra. Fungsi quantize_image menerima path gambar dan daftar nilai k (jumlah bin) sebagai input. Pertama, citra asli dimuat menggunakan imread dari scikit-image. Selanjutnya, program membuat subplot untuk setiap nilai k dalam daftar k_values. Dalam setiap iterasi, program membuat k bin dengan lebar yang sama antara 0 dan nilai intensitas maksimum citra. Nilai piksel citra asli kemudian dipetakan ke bin terdekat menggunakan fungsi digitize dari NumPy. Selanjutnya, nilai-nilai yang terbinning dikonversi kembali ke rentang intensitas asli. Citra kuantisasi yang dihasilkan ditampilkan di subplot dengan judul yang menunjukkan jumlah bin (k). Setelah seluruh iterasi selesai, tata letak subplot disesuaikan untuk mencegah tumpang tindih, dan hasil akhirnya ditampilkan menggunakan plt.show(). Program ini memberikan pemahaman visual tentang pengaruh jumlah bin pada citra kuantisasi dan dapat digunakan untuk mengamati perubahan tingkat detail citra ketika k bertambah.

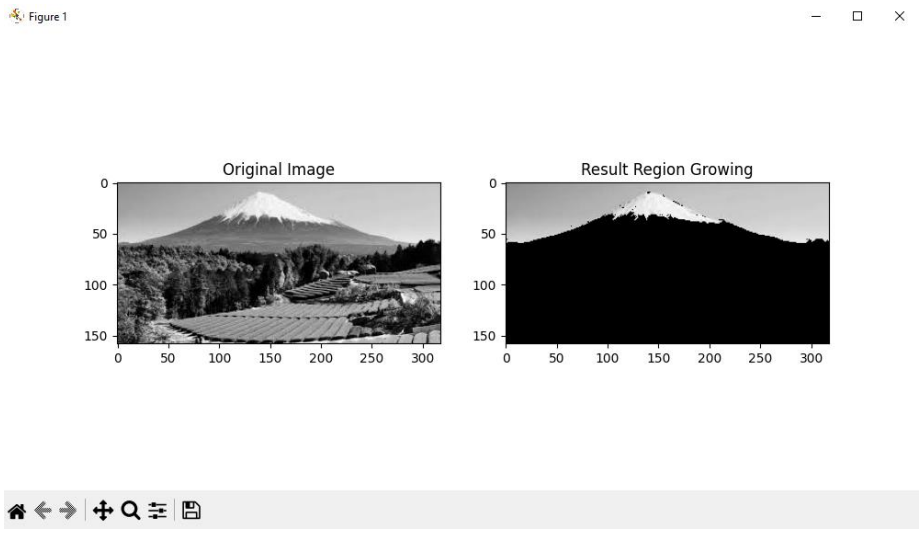
3) Region Growing

- **Kode Program:**

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

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("gambar2.png", 0)
    seed = (50, 50)

    result = region_growing(image, seed)

    # Menampilkan gambar asli dan hasil region growing dalam satu
window Matplotlib
    plt.figure(figsize=(10, 5))

    # Gambar Asli
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')

    # Hasil Region Growing
    plt.subplot(1, 2, 2)
    plt.imshow(result, cmap='gray')
    plt.title('Result Region Growing')

    plt.show()
```

- **Output**:



- **Analisa:**

  Program tersebut merupakan implementasi dari metode segmentasi citra menggunakan algoritma Region Growing. Fungsi region_growing menerima citra grayscale dan titik biji (seed) sebagai input. Pada awalnya, program menginisialisasi beberapa parameter, seperti toleransi warna, ukuran citra, serta array untuk menandai pixel yang telah dikunjungi dan hasil segmentasi. Algoritma Region Growing dimulai dari seed, di mana nilai intensitas piksel seed diambil sebagai nilai awal segmentasi. Program menggunakan tumpukan (stack) untuk melacak piksel yang akan ditelusuri. Selama proses iteratif, program memeriksa tetangga piksel dari piksel saat ini dan memeriksa apakah nilai intensitasnya berada dalam toleransi warna yang ditentukan. Jika iya, piksel tersebut ditambahkan ke hasil segmentasi dan dimasukkan ke dalam tumpukan untuk pemeriksaan lebih lanjut. Proses ini berlanjut hingga tumpukan kosong, dan citra hasil segmentasi ditampilkan menggunakan matplotlib. Dengan memilih titik biji yang sesuai, program dapat menghasilkan segmentasi citra yang memisahkan wilayah dengan nilai intensitas yang mirip.

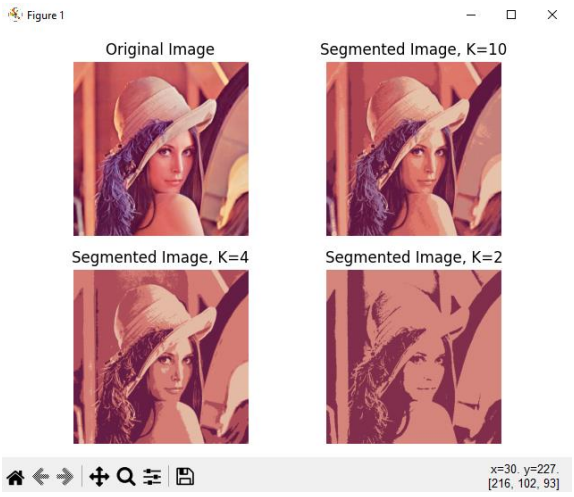
4) K-means Clustering

- **Kode Program:**

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

# Read in the image
image = cv2.imread('gambar1.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))

# Convert to float type
pixel_vals = np.float32(pixel_vals)

# The below line of code defines the criteria for the algorithm to
stop running,
# which will happen if 100 iterations are run or the epsilon (which
is the required accuracy)
# becomes 85%
```

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

# Display the original image
plt.subplot(2, 2, 1)
plt.imshow(image)
plt.title('Original Image')
plt.axis('off')

# Perform k-means clustering for different values of K
for i, K in enumerate([10, 4, 2], start=2):
    # Then perform k-means clustering with the number of clusters
defined as K
    # Also, random centers are initially chosen for k-means
clustering
    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()]

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

    # Display the segmented image
    plt.subplot(2, 2, i)
    plt.imshow(segmented_image)
    plt.title(f'Segmented Image, K={K}')
    plt.axis('off')

plt.tight_layout()
plt.show()
```

- **Output**:



- **Analisa:**

  Program tersebut merupakan implementasi segmentasi citra menggunakan algoritma K-means clustering dengan menggunakan Python dan beberapa library seperti OpenCV, NumPy, dan Matplotlib. Pertama, gambar dibaca menggunakan OpenCV, dan konversi warna dilakukan dari format BGR ke RGB. Selanjutnya, piksel gambar diubah menjadi array dua dimensi yang merepresentasikan matriks piksel dan tiga nilai warna (RGB). Array ini diubah menjadi tipe data float32 untuk mempersiapkannya sebagai input untuk algoritma K-means. Kriteria berhenti algoritma K-means ditentukan dengan menggunakan epsilon (0.85) dan jumlah iterasi maksimum (100). Program kemudian menampilkan gambar asli dan melakukan segmentasi untuk beberapa nilai K (jumlah kluster), yaitu 10, 4, dan 2. Setiap iterasi K-means menghasilkan label kluster dan nilai pusat kluster, yang digunakan untuk mendapatkan gambar hasil segmentasi. Hasil segmentasi untuk setiap nilai K ditampilkan dalam subplot yang sesuai menggunakan Matplotlib. Dengan memilih jumlah kluster yang berbeda, program memberikan gambaran tentang efek segmentasi citra pada tingkat kluster yang berbeda.

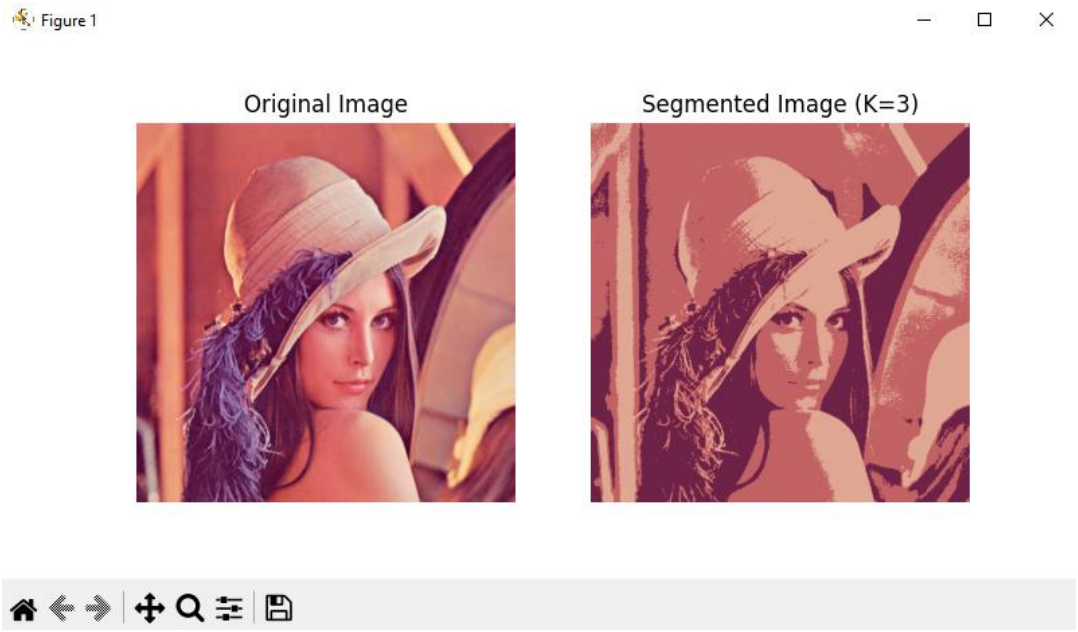
5) Meng-Hee Heng's K-Means Clustering

- **Kode Program:**

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

def kmeans_segmentation(image_path, k):
    # Load the image
    image = cv2.imread(image_path)

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

    # Convert the pixel values to float
    pixels = np.float32(pixels)

    # Define criteria and apply kmeans()
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
100, 0.2)
    _, labels, centers = cv2.kmeans(pixels, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)

    # Convert back to 8-bit values
```

```
    centers = np.uint8(centers)

    # Map the labels to the centers
    segmented_image = centers[labels.flatten()]

    # Reshape back to the original image shape
    segmented_image = segmented_image.reshape(image.shape)

    # Display the original and segmented images using matplotlib
    plt.figure(figsize=(8, 4))

    plt.subplot(1, 2, 1)
    plt.title('Original Image')
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.title('Segmented Image (K=' + str(k) + ')')
    plt.imshow(cv2.cvtColor(segmented_image, cv2.COLOR_BGR2RGB))
    plt.axis('off')

    plt.show()

# Example usage
image_path = 'gambar1.png'
k = 3  # Number of clusters (adjust as needed)
kmeans_segmentation(image_path, k)
```

- **Output**:



- **Analisa:**

  Program tersebut menggunakan algoritma Meng-Hee Heng's K-Means Clustering untuk melakukan segmentasi citra. Fungsi kmeans_segmentation menerima path gambar dan jumlah kluster (k) sebagai parameter. Pertama, gambar dimuat menggunakan OpenCV dan diubah menjadi array dua dimensi yang merepresentasikan matriks piksel dan tiga nilai warna (RGB). Nilai piksel dikonversi menjadi tipe data float32 untuk persiapan input algoritma K-Means. Kriteria berhenti algoritma K-Means ditentukan, dan algoritma diterapkan pada data piksel menggunakan metode cv2.kmeans(). Hasilnya berupa label kluster dan nilai pusat kluster. Kemudian, nilai pusat kluster diubah kembali ke tipe data uint8 dan digunakan untuk memetakan label ke nilai piksel yang sesuai. Hasil segmentasi dikembalikan ke bentuk asli gambar dan ditampilkan bersamaan dengan gambar asli menggunakan Matplotlib. Pengguna dapat mengatur jumlah kluster (k) sesuai kebutuhan untuk melihat dampak segmentasi pada citra dengan tingkat kluster yang berbeda.

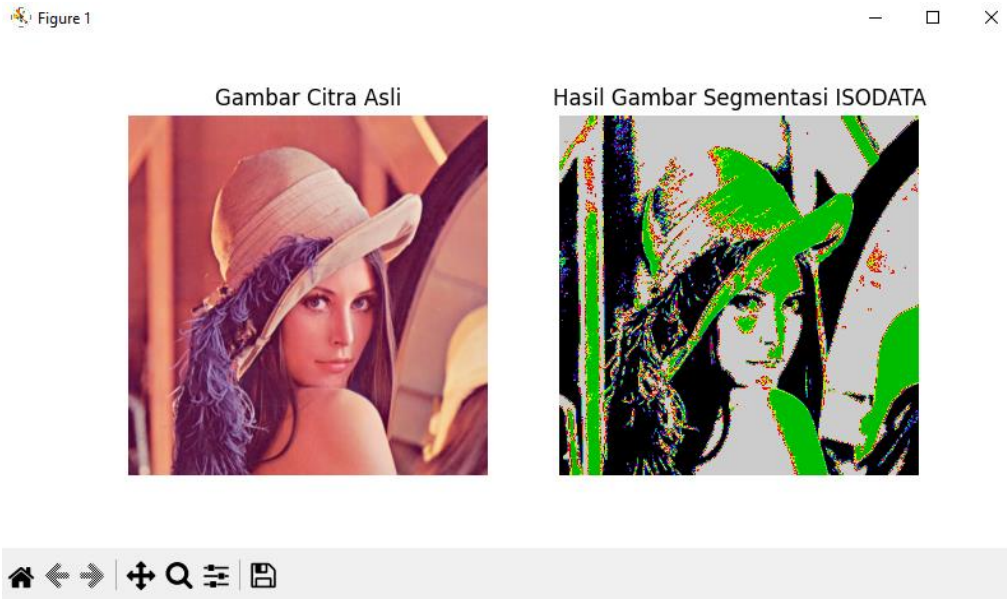
6) Isodata Clustering

- **Kode Program:**

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

def isodata_segmentation(image, num_clusters, max_iterations,
min_samples, max_variance):
    gray_image = color.rgb2gray(image)
    m, n = gray_image.shape
    data = gray_image.reshape(-1, 1)
    np.random.seed(0)
    centroids = np.random.rand(num_clusters, 1)

    for _ in range(max_iterations):
        distances = np.abs(data - centroids.T)
        labels = np.argmin(distances, axis=1)
        new_centroids = np.array([data[labels == i].mean() for i in
range(num_clusters)])
        if np.all(np.isclose(new_centroids, centroids, atol=1e-2)):
            break

        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('gambar1.png')
    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**:



- **Analisa:**

  Program tersebut adalah implementasi metode segmentasi citra menggunakan algoritma clustering ISODATA (Iterative Self-Organizing Data Analysis Technique Algorithm). Pertama, citra berwarna dimuat dan diubah menjadi citra grayscale. Kemudian, citra grayscale diubah menjadi array satu dimensi untuk mempermudah pemrosesan. Algoritma kemudian menginisialisasi pusat cluster secara acak dan melakukan iterasi sebanyak yang ditentukan oleh parameter max_iterations. Pada setiap iterasi, jarak antara setiap piksel dengan pusat cluster dihitung, dan piksel tersebut diatributkan ke cluster dengan pusat terdekat. Pusat cluster baru dihitung sebagai rata-rata piksel dalam setiap cluster. Proses ini berlanjut hingga konvergensi atau mencapai batas iterasi maksimum. Setelah iterasi selesai, varian cluster dihitung, dan cluster-cluster dengan varian di bawah batas tertentu dibakukan. Hasil akhirnya adalah citra yang telah di-segmentasi, di mana setiap piksel diberi label sesuai dengan cluster yang telah ditentukan.

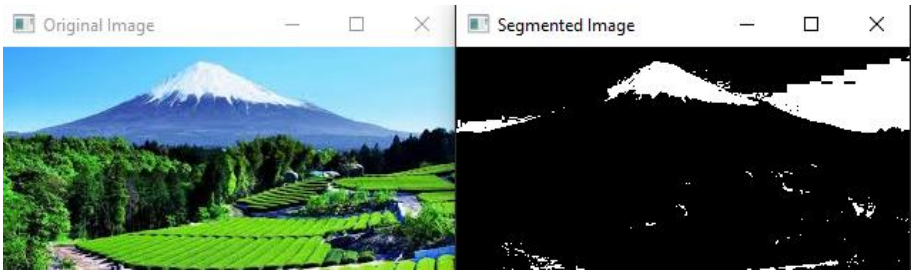
7) Ohlander's Recursive Histogram-Based Clustering

- **Kode Program:**

```
import cv2
import numpy as np

# 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('gambar2.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()
```

- **Output**:



- **Analisa:**

  Program tersebut adalah implementasi segmentasi citra menggunakan metode Ohlander's Recursive Histogram-Based Clustering. Pertama, gambar dimuat dan jika gambar berwarna, diubah menjadi citra grayscale. Histogram citra dihitung untuk mendapatkan distribusi intensitas piksel. Selanjutnya, program mencari nilai pemisah (split value) dengan iterasi mundur melalui histogram. Pemisah tersebut dipilih berdasarkan ambang batas tertentu (threshold) dari nilai maksimum histogram. Setelah mendapatkan nilai pemisah, citra di-segmentasi dengan menetapkan nilai piksel di atas nilai pemisah menjadi 255 (putih) dan sisanya menjadi 0 (hitam). Hasil akhirnya adalah citra biner yang telah di-segmentasi. Program menampilkan citra asli dan citra hasil segmentasi menggunakan OpenCV, memungkinkan visualisasi dan evaluasi segmentasi secara langsung.

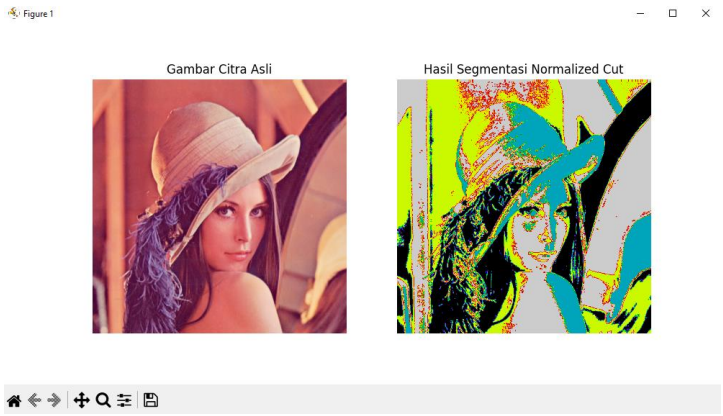
8) Jianbo Shi's Graph-Partitioning (Normalized Cut)

- **Kode Program:**

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


# Membaca citra
```

```
image = io.imread("gambar1.png")
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**:



- **Analisa:**

  Program tersebut menerapkan metode segmentasi citra menggunakan Normalized Cut, Pertama, citra dimuat dan diubah menjadi array dua dimensi. Kemudian, algoritma K-Means diterapkan pada data citra untuk mengelompokkan piksel menjadi sejumlah klaster yang ditentukan oleh parameter num_segments. Hasil klaster tersebut diubah kembali menjadi citra dua dimensi. Metode K-Means membantu mengidentifikasi struktur dan pola dalam citra. Pemilihan jumlah klaster memengaruhi tingkat segmentasi dan kejelasan hasil akhir. Citra asli dan citra hasil segmentasi ditampilkan menggunakan matplotlib untuk memudahkan evaluasi visual. Pendekatan ini menggambarkan prinsip Normalized Cut dalam mengelompokkan piksel-piksel yang saling terkait berdasarkan kesamaan fitur, sehingga memungkinkan segmentasi yang baik dalam gambar.