**TUGAS 3**

**VISI KOMPUTER DAN PENGOLAHAN CITRA**



**Oleh :**

**Silfiana Nur Hamida**

**(1223800005)**

**Membahas tentang:**

***“Image Segmentation”***

**PROGRAM PASCASARJANA TEKNIK INFORMATIKA DAN KOMPUTER**

POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

2023/2024

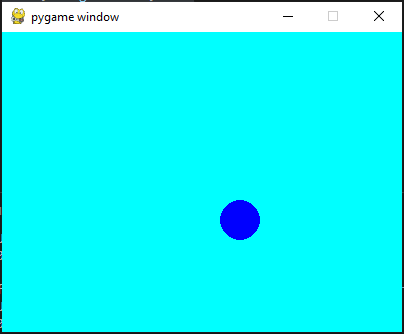
1. **Levitation**

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 menciptkan 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**



1. **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



1. **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



1. **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**



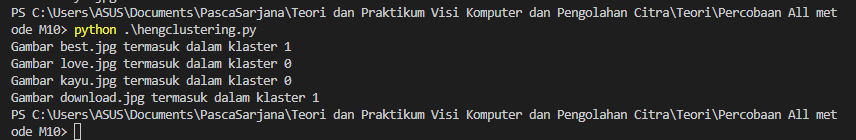
1. **Meng-Hee Heng’s K-means**

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)      contrast = graycoprops(glcm, prop='contrast')      correlation = graycoprops(glcm, prop='correlation')        return [contrast[0, 0], correlation[0, 0]]  image\_paths = ["best.jpg", "love.jpg", "kayu.jpg", "download.jpg"]  features = [extract\_texture\_features(path) for path in image\_paths]  kmeans = KMeans(n\_clusters=2)  kmeans.fit(features)  labels = kmeans.labels\_  for i, path in enumerate(image\_paths):      print(f"Gambar {path} termasuk dalam klaster {labels[i]}") |

* **Output Program**



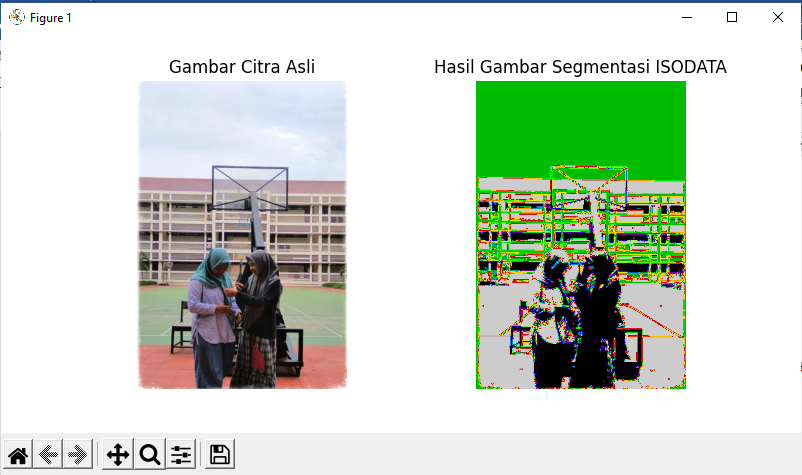
1. **Isodata Clustering**

Pada percobaan dengan menggunakan Isodata Clustering yang dimana teknik ini bertujuan untuk membagi citra menjadi beberapa kelompok atau segmen yag homogen berdasarkan kesamaan intensitas piksel. Dan berikut ini merupakan implementasi codenya.

* **Source Code**

|  |
| --- |
| 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('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**



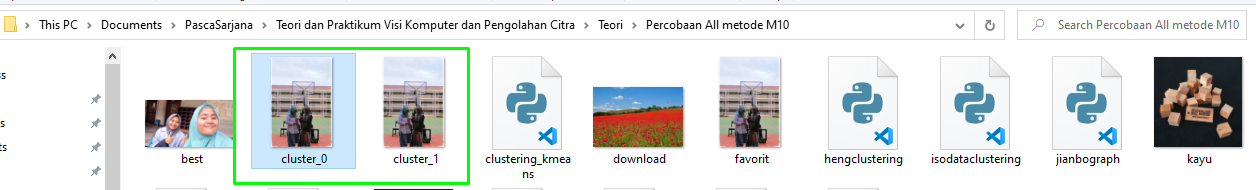
1. **Ohlander’s Recursive Histogram-Based Clustering**

Pada percobaan menggunakan teknik Ohlander’s Recursive Histogram-Based Clustering yang dimana teknik ini memanfaatka 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**



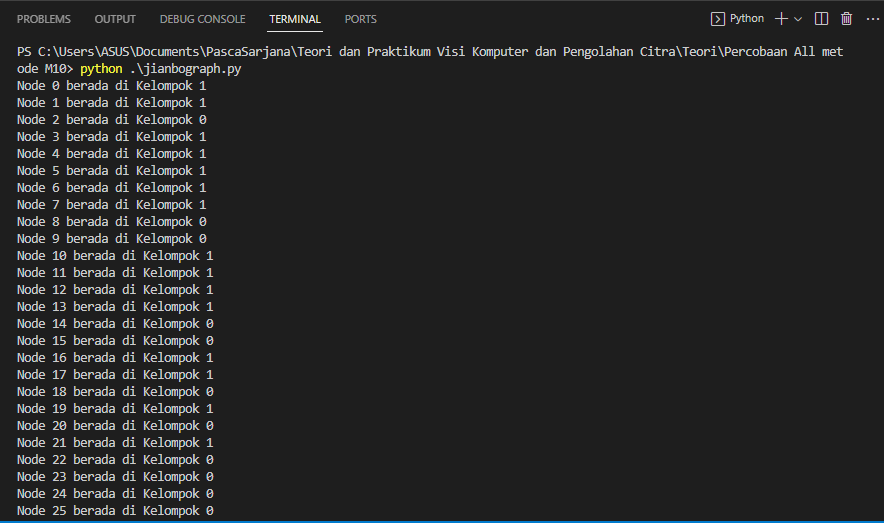
1. **Jianbo Shi’s Graph-Partitioning**

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**



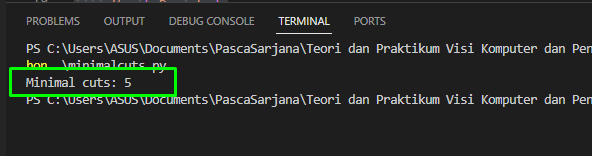
1. **Minimal Cuts**

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**



1. **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**



1. **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**

