Uji Coba Program Pengolahan Citra Digital
Untuk Ujian Akhir Semester

Aurio Rajaa (2207412016)
Jurusan Teknik Informatika, Politeknik Negeri Jakarta, Kelas TI-CCIT4A

Link Source Code:

auriorajaa/Uji-Coba-Program-Pengolahan-Citra-Digital-Untuk-Ujian-Akhir-Semester (github.com)

Thresholding

Thresholding adalah teknik segmentasi gambar sederhana untuk memisahkan objek dari latar belakang berdasarkan intensitas pixel. Prinsipnya membandingkan intensitas setiap pixel dengan nilai ambang (threshold) yang ditentukan. Pixel diberi nilai tertentu jika intensitasnya di atas atau di bawah threshold. OpenCV menyediakan fungsi cv::threshold() dengan berbagai jenis operasi thresholding untuk kebutuhan segmentasi gambar yang berbeda-beda.

Program:

```
from __future__ import print_function
import cv2 as cv
import argparse

# Constants
max_value = 255
max_type = 4
max_binary_value = 255
trackbar_type = 'Type: \n 0: Binary \n 1: Binary Inverted \n 2: Truncate \n
3: To Zero \n 4: To Zero Inverted'
trackbar_value = 'Value'
window_name = 'Threshold Demo'

# Function for thresholding demo
def Threshold_Demo(val):
    # Get the current positions of the trackbars
    threshold_type = cv.getTrackbarPos(trackbar_type, window_name)
    threshold_value = cv.getTrackbarPos(trackbar_value, window_name)

    # Apply the thresholding
     _, dst = cv.threshold(src_gray, threshold_value, max_binary_value,
threshold_type)
    cv.imshow(window_name, dst)

# Argument parser to get input image
parser = argparse.ArgumentParser(description='Code for Basic Thresholding
Operations tutorial.')
parser.add_argument('--input', help='Path to input image.',
default='gambar.jpg')
args = parser.parse_args()

# Read the input image
src = cv.imread(args.input)
if src is None:
    print('Could not open or find the image:', args.input)
    exit(0)

# Convert the image to gray
src_gray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)

# Create a window and trackbars
cv.namedWindow(window_name)
cv.createTrackbar(trackbar_type, window_name, 3, max_type, Threshold_Demo)
cv.createTrackbar(trackbar_value, window_name, 0, max_value, Threshold_Demo)

# Initialize
```

```
Threshold_Demo(0)

# Wait until the user finishes the program
cv.waitKey()
```

Kode diatas adalah skrip Python menggunakan OpenCV untuk melakukan operasi thresholding dasar pada sebuah gambar. Skrip ini memungkinkan pengguna untuk mengatur jenis dan nilai threshold secara interaktif melalui trackbar.

Pertama, skrip mengimpor pustaka yang diperlukan (`cv2` untuk OpenCV dan `argparse` untuk argumen baris perintah). Kemudian, beberapa konstanta didefinisikan untuk nilai maksimum dan label trackbar.

Fungsi `Threshold_Demo` digunakan untuk mendapatkan posisi trackbar saat ini dan menerapkan thresholding pada gambar grayscale, lalu menampilkan hasilnya di jendela. Skrip menggunakan `argparse` untuk mendapatkan path gambar input dari argumen baris perintah. Gambar yang dimasukkan dibaca dan dikonversi ke grayscale.

Jika gambar tidak dapat dibuka, skrip akan mencetak pesan kesalahan dan keluar. Selanjutnya, sebuah jendela dan dua trackbar dibuat untuk mengontrol jenis dan nilai threshold. Fungsi `Threshold_Demo` dipanggil untuk inisialisasi dan menampilkan gambar yang sudah di-threshold. Skrip ini menunggu input dari pengguna (`cv.waitKey()`) agar jendela tetap terbuka hingga pengguna menutupnya.

Hasil Program:

Gambar original vs gambar output dari program


RGB to HSV Conversion

RGB dan HSV adalah dua model warna utama dalam pengolahan citra digital. RGB menggunakan campuran merah, hijau, dan biru, sementara HSV menggunakan hue, saturation, dan value. HSV lebih intuitif untuk analisis dan pengeditan gambar.

Konversi RGB ke HSV penting dalam pemrosesan gambar, memudahkan identifikasi dan penyesuaian atribut warna. Ini berguna untuk berbagai aplikasi seperti koreksi warna, deteksi objek, segmentasi gambar, dan pencarian gambar berbasis warna. HSV juga unggul dalam analisis histogram warna dan thresholding gambar.

Program:

```
import colorsys
(r, g, b) = (0.2, 0.4, 0.4)
(h, s, v) = colorsys.rgb_to_hsv(r, g, b)
print(f"Hue: {h}, Saturation: {s}, Value: {v}")
```

Kode ini menggunakan modul colorsys untuk mengkonversi nilai warna dari format RGB ke format HSV. Pertama, tiga variabel r, g, dan b diatur dengan nilai merah, hijau, dan biru masing-masing 0.2, 0.4, dan 0.4. Fungsi colorsys.rgb_to_hsv kemudian digunakan untuk mengubah nilai RGB tersebut menjadi nilai Hue, Saturation, dan Value (HSV). Hasil konversi ini disimpan dalam variabel h, s, dan v, yang kemudian dicetak ke layar.

Program:

```
import colorsys
# Input RGB values
(r, g, b) = (142, 244, 85)

# Normalize the RGB values
(r, g, b) = (r / 255, g / 255, b / 255)

# Convert RGB to HSV
(h, s, v) = colorsys.rgb_to_hsv(r, g, b)

# Expand HSV range
(h, s, v) = (int(h * 179), int(s * 255), int(v * 255))

# Print the HSV values
print('HSV : ', h, s, v)
```

Kode ini mengambil nilai warna dalam format RGB, menormalkannya, dan kemudian mengonversinya ke format HSV. Nilai RGB awalnya adalah (142, 244, 85). Nilai-nilai ini dinormalisasi dengan membagi setiap komponen dengan 255. Selanjutnya, fungsi colorsys.rgb_to_hsv digunakan untuk mengubah nilai RGB yang sudah dinormalisasi menjadi

nilai Hue, Saturation, dan Value (HSV). Untuk memperluas rentang HSV, nilai Hue dikalikan dengan 179 dan nilai Saturation dan Value masing-masing dikalikan dengan 255, kemudian dikonversi ke integer. Hasil konversi ini dicetak ke layar dalam format HSV.

Program:

```
import cv2

# Read the input RGB image (which is read as BGR by OpenCV)
bgr_img = cv2.imread('antony.jpg')

# Convert the BGR image to HSV format
hsv_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2HSV)

# Save the HSV image
cv2.imwrite('hsv_image.jpg', hsv_img)

# Display the HSV image
cv2.imshow('HSV image', hsv_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Kode ini membaca gambar RGB dengan nama antony.jpg menggunakan OpenCV, yang secara default membacanya dalam format BGR. Gambar kemudian dikonversi ke format HSV menggunakan fungsi cv2.cvtColor. Hasil konversi ini disimpan sebagai file hsv_image.jpg. Selanjutnya, gambar HSV ditampilkan dalam sebuah jendela menggunakan cv2.imshow. Program akan menunggu hingga ada input tombol dari pengguna (cv2.waitKey(0)) sebelum menutup jendela tersebut dan menghancurkan semua jendela yang dibuka (cv2.destroyAllWindows).

Output:

Image Compression

Image compression adalah proses pengurangan ukuran file gambar dengan meminimalkan dampak pada kualitas visual. Terdapat dua jenis utama: lossless (mempertahankan data asli)

dan lossy (menghapus sebagian informasi). Kompresi gambar penting untuk menghemat ruang penyimpanan, mempercepat transmisi, dan mengurangi penggunaan bandwidth.

Teknik kompresi memanfaatkan redundansi spasial dan warna dalam gambar, menggunakan metode seperti Run-Length Encoding, Huffman Encoding, Discrete Cosine Transform (DCT), dan quantization untuk mencapai ukuran file yang lebih kecil dengan tetap menjaga kualitas gambar.

Program:

```
import cv2
import os

# Read the input image
img = cv2.imread('antony.jpg')

# Save the image as a lossless PNG
cv2.imwrite('lossless_compressed_image.png', img)

# Set JPEG quality (0-100, higher means better quality but larger file size)
jpeg_quality = 90

# Save the image as a lossy JPEG
cv2.imwrite('lossy_compressed_image.jpg',  img,  [cv2.IMWRITE_JPEG_QUALITY,
jpeg_quality])

# Get the file sizes of the original, lossless, and lossy compressed images
original_size = os.path.getsize('antony.jpg')
lossless_size = os.path.getsize('lossless_compressed_image.png')
lossy_size = os.path.getsize('lossy_compressed_image.jpg')

# Print the file sizes
print(f'Original image size: {original_size} bytes')
print(f'Lossless compressed image size: {lossless_size} bytes')
print(f'Lossy compressed image size: {lossy_size} bytes')

# Read the compressed images
lossless_img = cv2.imread('lossless_compressed_image.png')
lossy_img = cv2.imread('lossy_compressed_image.jpg')

# Display the original and compressed images
cv2.imshow('Original Image', img)
cv2.imshow('Lossless Compressed Image', lossless_img)
cv2.imshow('Lossy Compressed Image', lossy_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

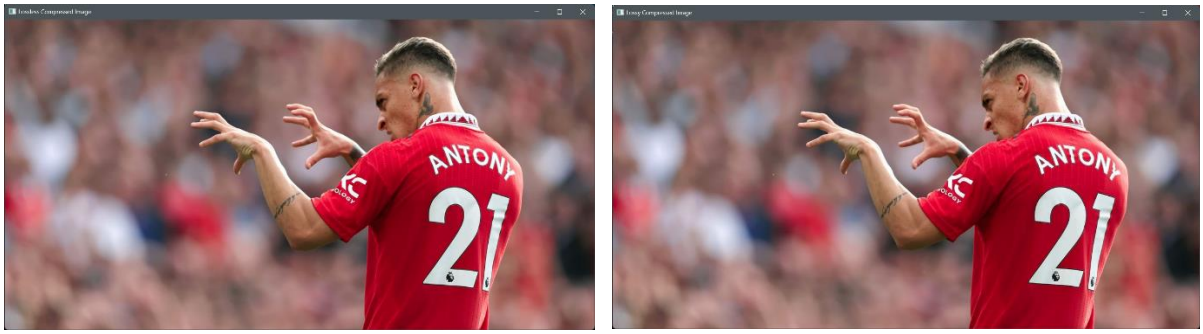
Kode diatas membaca gambar antony.jpg menggunakan OpenCV, kemudian menyimpan gambar tersebut dalam dua format: PNG (lossless) dan JPEG (lossy) dengan kualitas yang ditentukan. Kualitas JPEG diatur ke 90, di mana nilai lebih tinggi berarti kualitas lebih baik, tetapi ukuran file lebih besar.

Ukuran file gambar asli, gambar kompresi lossless, dan gambar kompresi lossy dihitung dan dicetak. Gambar asli dan gambar yang dikompresi kemudian dibaca kembali dan ditampilkan menggunakan OpenCV. Program akan menunggu hingga ada input tombol dari pengguna sebelum menutup semua jendela yang dibuka.

Output:

```
Original image size: 70952 bytes
Lossless compressed image size: 844305 bytes
Lossy compressed image size: 94776 bytes
```

Lossless vs Lossy Compressed Image



Morphological Transformations

Morphology adalah operasi pemrosesan gambar berdasarkan bentuk. Operasi ini menggunakan elemen penataan pada gambar input untuk menghasilkan gambar output dengan ukuran sama. Nilai pixel output didasarkan pada perbandingan pixel input dengan tetangganya.

Morphology terkait dengan Image Segmentation, bisa digunakan untuk pre-process atau post-process. Prosesnya mirip spatial filtering, dengan structuring element yang dipindahkan ke setiap pixel gambar asli. Dua operasi utama adalah Erosi (mengecilkan atau menghilangkan pixel batas objek) dan Dilasi (memperluas atau menambah pixel batas objek).

Program:

```
import cv2 as cv
import numpy as np

# Read the input image in grayscale
img = cv.imread('antony.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "File could not be read, check with
os.path.exists()"

# Define a 5x5 kernel of ones
kernel = np.ones((5,5),np.uint8)

# Perform various morphological operations
```

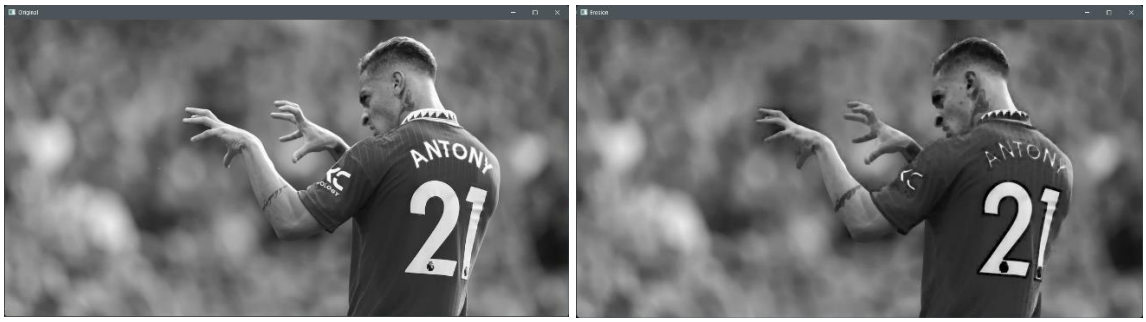
```
erosion = cv.erode(img, kernel, iterations=1)           # Erosion
dilation = cv.dilate(img, kernel, iterations=1)         # Dilation
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)   # Opening
closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)  # Closing
gradient = cv.morphologyEx(img, cv.MORPH_GRADIENT, kernel) # Morphological
Gradient
tophat = cv.morphologyEx(img, cv.MORPH_TOPHAT, kernel)     # Top Hat
blackhat = cv.morphologyEx(img, cv.MORPH_BLACKHAT, kernel) # Black Hat

# Create different types of kernels
rect_kernel = cv.getStructuringElement(cv.MORPH_RECT, (5,5))      #
Rectangular Kernel
ellipse_kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5)) #
Elliptical Kernel
cross_kernel = cv.getStructuringElement(cv.MORPH_CROSS, (5,5))     # Cross-
shaped Kernel

# Display the results
cv.imshow("Original", img)
cv.imshow("Erosion", erosion)
cv.imshow("Dilation", dilation)
cv.imshow("Opening", opening)
cv.imshow("Closing", closing)
cv.imshow("Gradient", gradient)
cv.imshow("Top Hat", tophat)
cv.imshow("Black Hat", blackhat)

# Print the kernels
print("Rectangular Kernel:")
print(rect_kernel)
print("Elliptical Kernel:")
print(ellipse_kernel)
print("Cross-shaped Kernel:")
print(cross_kernel)

cv.waitKey(0)
cv.destroyAllWindows()
```

Kode ini memuat gambar antony.jpg dalam format grayscale menggunakan OpenCV dan kemudian melakukan berbagai operasi morfologi pada gambar tersebut. Berbagai operasi yang dilakukan termasuk erosi, dilasi, pembukaan, penutupan, gradien morfologi, top hat, dan black hat, semuanya menggunakan kernel ukuran 5x5.

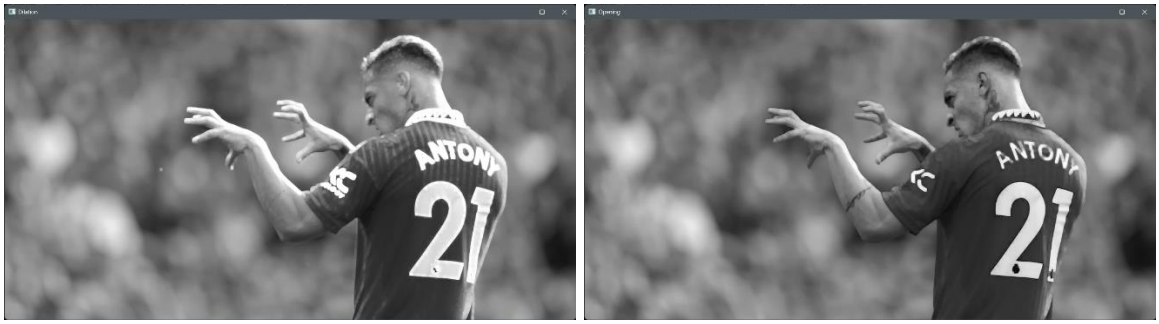
Selain itu, kode ini juga membuat dan mencetak tiga jenis kernel berbeda: persegi panjang, elips, dan berbentuk silang. Hasil dari setiap operasi ditampilkan menggunakan OpenCV. Program akan menunggu hingga ada input tombol dari pengguna sebelum menutup semua jendela yang dibuka.

Output:

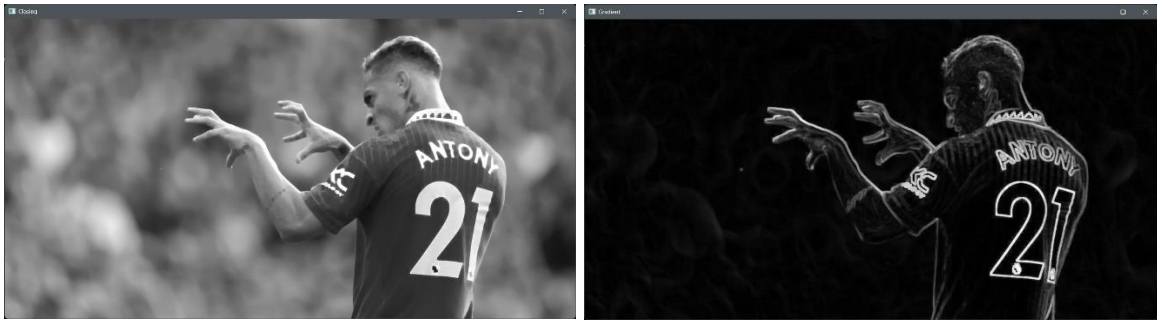
Original vs Erosion



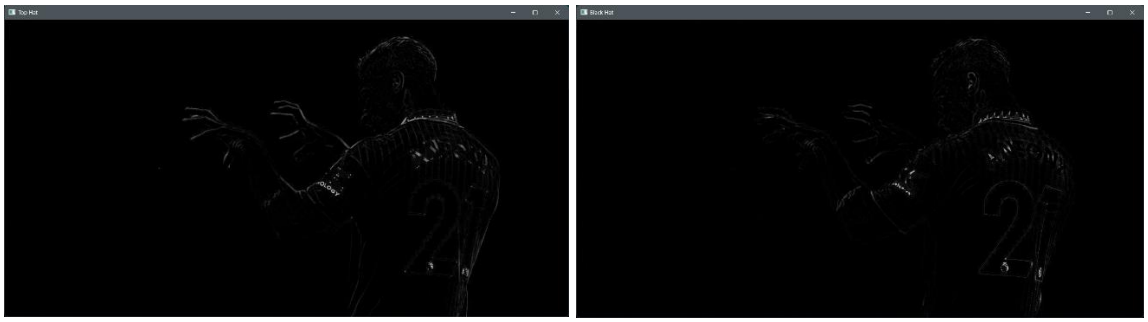
Dilation vs Opening:



Closing vs Gradient:



Top hat vs Black hat:



Element structural:

```
Rectangular Kernel:
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
Elliptical Kernel:
[[0 0 1 0 0]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [0 0 1 0 0]]
Cross-shaped Kernel:
[[0 0 1 0 0]
 [0 0 1 0 0]
 [1 1 1 1 1]
 [0 0 1 0 0]
 [0 0 1 0 0]]
```

Segmentasi Citra

Image segmentation adalah proses membagi gambar menjadi beberapa wilayah berdasarkan karakteristik pixel yang sama. Teknik ini penting untuk berbagai aplikasi seperti deteksi objek, analisis gambar medis, dan navigasi kendaraan otomatis. Beberapa metode populer untuk image segmentation meliputi:

1. Algoritma K-means: Mengelompokkan pixel berdasarkan kemiripan.
2. Contour detection: Mengidentifikasi kurva dan poligon dengan intensitas atau warna yang sama.
3. Masking: Menerapkan gambar biner sebagai "topeng" pada gambar asli.
4. Color detection: Menganalisis nilai ruang warna RGB untuk mengklasifikasikan warna.

Teknik-teknik ini sering digunakan bersama dengan metode lain seperti edge detection atau thresholding untuk mengungkap struktur dan pola dalam gambar.

Program:

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

# Load the image
sample_image = cv2.imread('antony.jpg')
img = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)

# Reshape the image to a 2D array of pixels
twoDimage = img.reshape((-1, 3))
twoDimage = np.float32(twoDimage)

# Create a figure with 4 subplots
```

```
fig, axs = plt.subplots(2, 2, figsize=(12, 12))

# Perform K-Means clustering with different K values
for i, k in enumerate([3, 4, 5, 6]):
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    attempts = 10
    ret, label, center = cv2.kmeans(twoDimage, k, None, criteria, attempts,
cv2.KMEANS_PP_CENTERS)
    center = np.uint8(center)
    res = center[label.flatten()]
    result_image = res.reshape((img.shape))

    # Plot the results
    row = i // 2
    col = i % 2
    axs[row, col].imshow(result_image)
    axs[row, col].set_title(f'K={k}')
    axs[row, col].axis('off')

# Show the figure
plt.show()
```

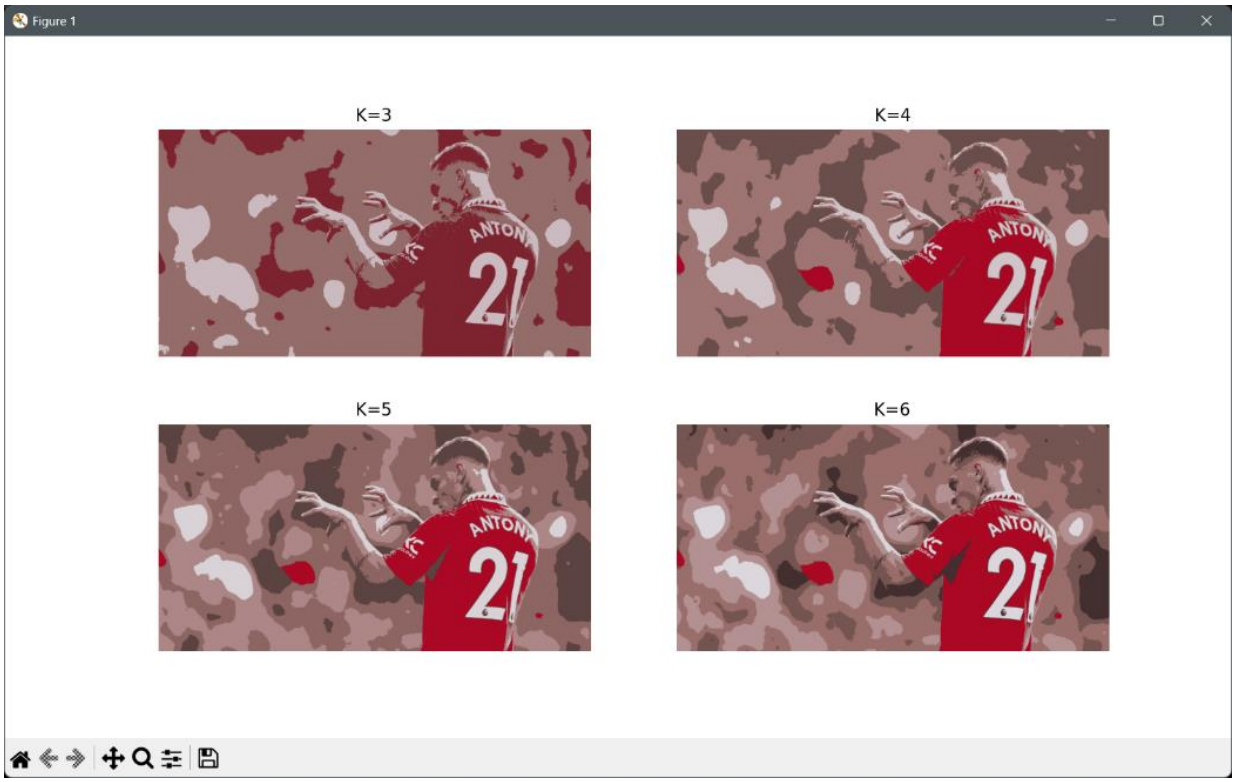
Kode ini memuat gambar antony.jpg, mengonversinya dari format BGR ke RGB, dan kemudian mengubahnya menjadi array 2D dari piksel. Kode ini melakukan clustering K-Means pada gambar tersebut dengan nilai K yang berbeda (3, 4, 5, dan 6) untuk mempartisi piksel gambar ke dalam K kelompok.

Hasil clustering untuk setiap nilai K ditampilkan dalam subplot yang terpisah pada sebuah figur dengan ukuran 2x2. Subplot diatur sedemikian rupa sehingga setiap gambar hasil clustering memiliki judul yang menunjukkan nilai K yang digunakan, dan sumbu subplot dinonaktifkan untuk tampilan yang lebih bersih.

Output:

Original

Output dari program



Program:

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

# Load and preprocess the image
sample_image = cv2.imread('antony.jpg')
img = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (256, 256))

# Create a figure with 4 subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 12))

# Image Thresholding
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
_, thresh = cv2.threshold(gray, np.mean(gray), 255, cv2.THRESH_BINARY_INV)
axes[0, 0].imshow(thresh, cmap='gray')
axes[0, 0].set_title('Image Thresholding')
axes[0, 0].axis('off')

# Detecting Edges
edges = cv2.dilate(cv2.Canny(thresh, 0, 255), None)
axes[0, 1].imshow(edges, cmap='gray')
axes[0, 1].set_title('Detecting Edges')
axes[0, 1].axis('off')

# Detecting Contours To Create Mask
cnt = sorted(cv2.findContours(edges, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)[-2], key=cv2.contourArea)[-1]
```

```
mask = np.zeros((256, 256), np.uint8)
masked = cv2.drawContours(mask, [cnt], -1, 255, -1)
axes[1, 0].imshow(masked, cmap='gray')
axes[1, 0].set_title('Detecting Contours To Create Mask')
axes[1, 0].axis('off')

# Segmenting Region
dst = cv2.bitwise_and(img, img, mask=mask)
segmented = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
axes[1, 1].imshow(segmented)
axes[1, 1].set_title('Segmenting Region')
axes[1, 1].axis('off')

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```

Kode diatas memuat gambar antony.jpg, mengonversinya ke format RGB, dan mengubah ukurannya menjadi 256x256 piksel. Proses dimulai dengan menerapkan **Image Thresholding** untuk mengubah gambar menjadi grayscale, kemudian menerapkan threshold untuk menghasilkan gambar biner terbalik, yang menonjolkan area yang lebih terang.

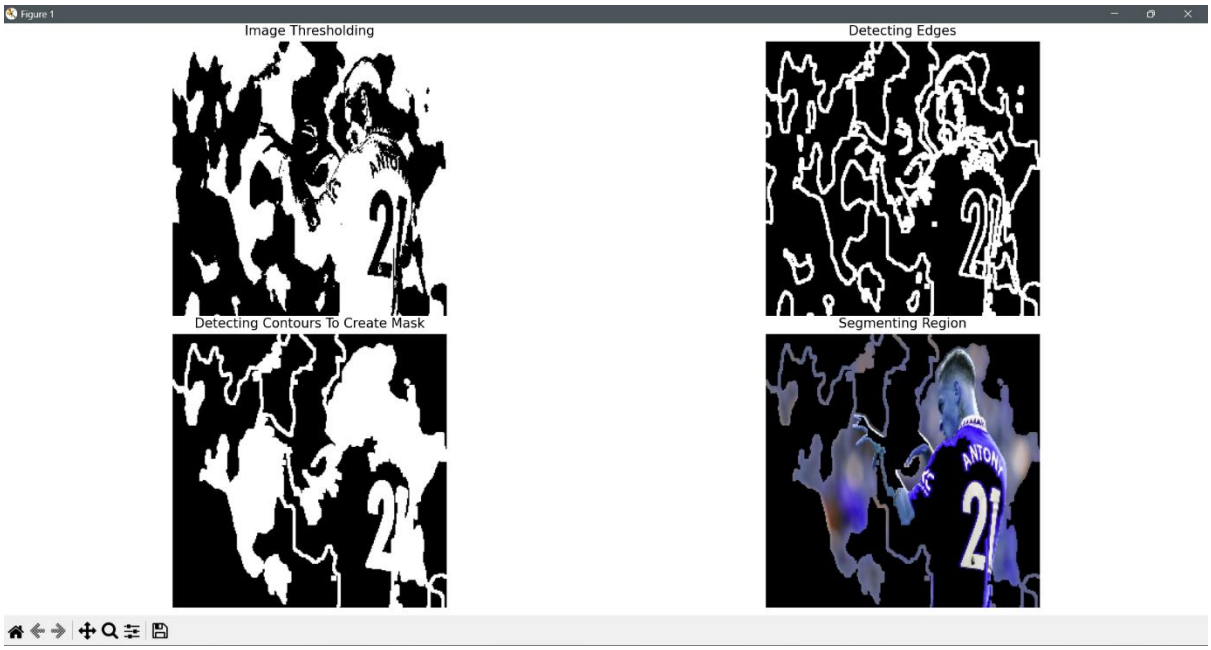
Selanjutnya, **Detecting Edges** menggunakan deteksi tepi Canny yang dilanjutkan dengan dilasi untuk menonjolkan tepi-tepi pada gambar biner. Setelah itu, **Detecting Contours To Create Mask** menemukan kontur terbesar pada gambar tepi dan menggambar kontur tersebut pada masker untuk membuat area yang tersegmentasi.

Akhirnya, **Segmenting Region** menggunakan masker untuk mengekstrak dan menampilkan area yang tersegmentasi dari gambar asli. Semua langkah ini ditampilkan dalam empat subplot yang terpisah, masing-masing dengan judul yang menjelaskan tahap pemrosesan gambar.

Output:

Original

Output dari program



Program diatas melakukan pemrosesan gambar menggunakan serangkaian teknik untuk segmentasi wilayah yang diminati:

1. Konversi gambar ke grayscale
2. Penerapan binary threshold
3. Penggunaan deteksi tepi Canny
4. Pelebaran edge map
5. Pencarian kontur terbesar
6. Pembuatan binary mask
7. Penerapan mask pada gambar RGB asli

Proses ini secara efektif menyegmentasikan wilayah yang diinginkan, dan hasilnya dapat dilihat pada gambar output.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.filters import threshold_otsu
import cv2

# Load and preprocess the image
sample_image = cv2.imread('antony.jpg')
```

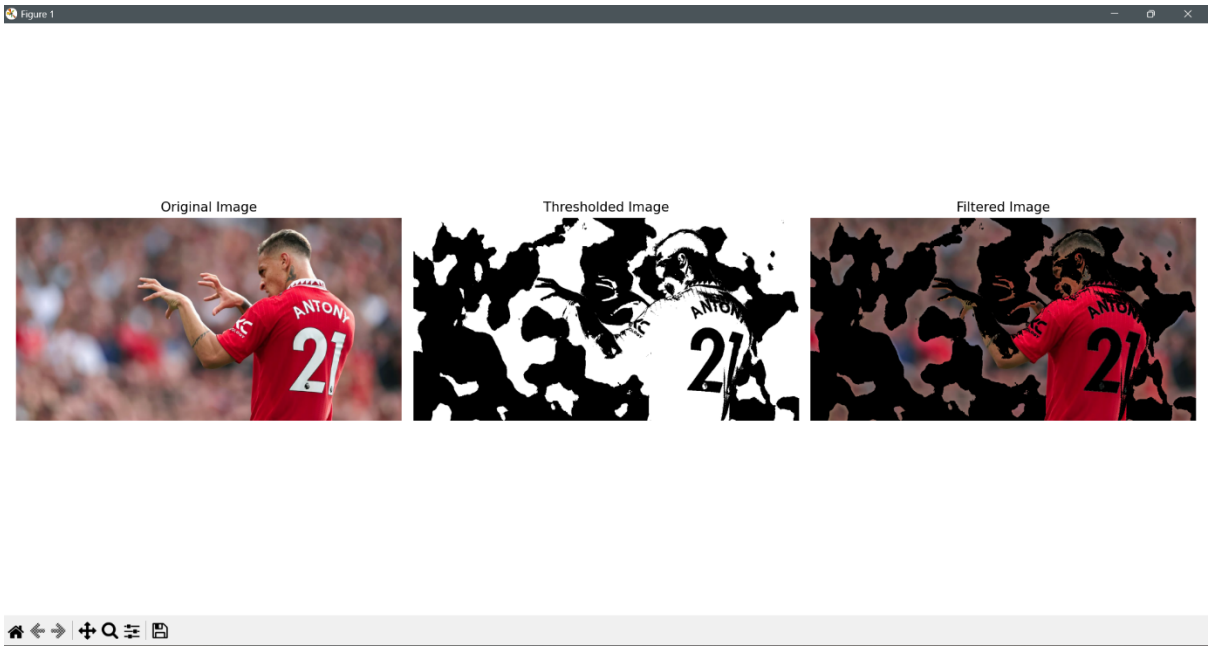
```
img = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)

# Create a figure with 3 subplots
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Original image
axes[0].imshow(img)
axes[0].set_title('Original Image')
axes[0].axis('off')

# Thresholded image
img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
thresh = threshold_otsu(img_gray)
img_otsu = img_gray < thresh
axes[1].imshow(img_otsu, cmap='gray')
axes[1].set_title('Thresholded Image')
axes[1].axis('off')

# Filtered image
def filter_image(image, mask):
    r = image[:, :, 0] * mask
    g = image[:, :, 1] * mask
    b = image[:, :, 2] * mask
    return np.dstack([r, g, b])

filtered = filter_image(img, img_otsu)
axes[2].imshow(filtered)
axes[2].set_title('Filtered Image')
axes[2].axis('off')

# Adjust layout and display the plots
plt.tight_layout()
plt.show()
```

Kode diatas memuat gambar antony.jpg, mengonversinya dari format BGR ke RGB, dan menyiapkannya untuk pemrosesan lebih lanjut. Gambar asli ditampilkan di subplot pertama. Selanjutnya, gambar dikonversi menjadi grayscale, dan threshold Otsu diterapkan untuk menghasilkan gambar biner berdasarkan ambang batas otomatis yang dihitung.

Gambar hasil thresholding ditampilkan di subplot kedua. Fungsi filter_image kemudian digunakan untuk mengaplikasikan masker biner ke gambar RGB asli, menghasilkan gambar terfilter yang menonjolkan area yang dipilih.

Gambar terfilter ini ditampilkan di subplot ketiga. Semua gambar ditata dalam satu baris subplot dengan judul yang sesuai dan sumbu yang dinonaktifkan untuk tampilan yang lebih bersih.

Output:



Algoritma Otsu thresholding digunakan untuk membuat gambar threshold, memisahkan foreground dan background. Pixel di bawah threshold menjadi hitam (0), di atas threshold menjadi putih (1). Gambar biner yang dihasilkan menyoroti area yang diinginkan dari gambar asli.

Binary mask diterapkan ke gambar RGB asli dengan mengalikan setiap channel warna dengan mask. Proses ini mempertahankan informasi warna di wilayah foreground dan menghapus di wilayah yang tidak dipilih. Hasilnya memungkinkan analisis dan visualisasi terfokus pada area yang diminati dalam gambar asli.

Program:

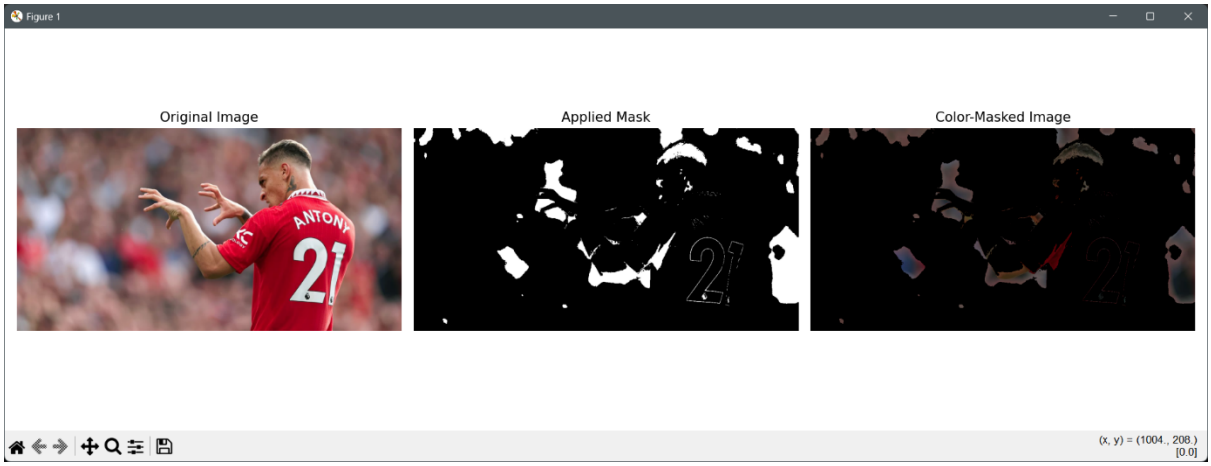
```
import numpy as np
import matplotlib.pyplot as plt
from skimage.filters import threshold_otsu
import cv2

# Load and preprocess the image
sample_image = cv2.imread('antony.jpg')
img = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)

# Define the color mask
low = np.array([0, 0, 0])
high = np.array([100, 255, 100])
mask = cv2.inRange(img, low, high)

# Apply color masking
result = cv2.bitwise_and(img, img, mask=mask)
```

```
# Create a figure with 3 subplots
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

# Plot the original image
ax1.imshow(img)
ax1.set_title('Original Image')
ax1.axis('off')

# Plot the applied mask
ax2.imshow(mask, cmap='gray')
ax2.set_title('Applied Mask')
ax2.axis('off')

# Plot the color-masked image
ax3.imshow(result)
ax3.set_title('Color-Masked Image')
ax3.axis('off')

# Adjust layout and display the plots
plt.tight_layout()
plt.show()
```

Kode ini memuat gambar antony.jpg, mengonversinya dari format BGR ke RGB, dan menerapkan pemrosesan untuk menyorot area tertentu berdasarkan warna. Masker warna didefinisikan dengan rentang warna dari [0, 0, 0] hingga [100, 255, 100], dan diterapkan ke gambar untuk mengekstrak area yang sesuai.

Hasil dari proses ini adalah gambar yang menampilkan hanya bagian yang sesuai dengan masker warna, ditampilkan bersamaan dengan gambar asli dan masker dalam tiga subplot. Semua subplot ditampilkan dalam satu baris dengan judul yang sesuai dan sumbu yang dinonaktifkan untuk tampilan yang bersih.

Output:



Mask warna dalam program ini berfungsi untuk memilih dan menyoroti area spesifik pada gambar berdasarkan rentang warna yang telah ditentukan. Proses ini secara efektif menutupi seluruh bagian gambar kecuali area yang sesuai dengan warna yang dipilih.

Hasilnya adalah sebuah gambar yang hanya menampilkan bagian-bagian yang cocok dengan warna yang diinginkan, sementara bagian lainnya dihilangkan, memungkinkan fokus visual pada elemen-elemen tertentu dalam gambar.

Feature Extraction

Feature extraction adalah proses penting dalam pengolahan citra dan computer vision yang mengidentifikasi dan merepresentasikan struktur-struktur pembeda dalam gambar. Proses ini mengubah data gambar mentah menjadi fitur-fitur numerik yang dapat diproses, sambil mempertahankan informasi penting.

Fitur-fitur ini, yang bisa berupa elemen sederhana seperti edge dan corner atau yang lebih kompleks seperti tekstur dan bentuk, sangat penting untuk tugas-tugas lanjutan seperti deteksi objek, klasifikasi, dan pencocokan gambar.

Tujuan utamanya adalah menciptakan representasi yang lebih ringkas dan bermakna dibandingkan data pixel mentah, sehingga memudahkan analisis dan pemrosesan lebih lanjut.

Program:

```
import cv2
import numpy as np

# Load the image
img = cv2.imread('antony.jpg')

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect edges using Canny method
edges = cv2.Canny(gray, 150, 300)

# Highlight the edges on the original image
img[edges == 255] = (255, 0, 0)  # Color the edges in red

# Display the image with edges highlighted
cv2.imshow('Canny Edges', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Kode diatas memuat gambar antony.jpg dan mengonversinya menjadi grayscale untuk mempermudah deteksi tepi. Metode Canny digunakan untuk mendeteksi tepi dalam gambar grayscale dengan ambang batas 150 dan 300.

Tepi yang terdeteksi kemudian digarisbawahi dengan warna merah pada gambar asli. Hasil akhir ditampilkan menggunakan OpenCV dalam jendela yang disebut "Canny Edges". Setelah pengguna menekan sembarang tombol, jendela tersebut ditutup.

Output:

Program:

```
import cv2
import numpy as np

# Load the image
img = cv2.imread('antony.jpg')

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect corners using the Harris method
dst = cv2.cornerHarris(gray, 3, 5, 0.1)

# Create a boolean bitmap of corner positions
corners = dst > 0.05 * dst.max()

# Find the coordinates from the boolean bitmap
coord = np.argwhere(corners)

# Draw circles on the coordinates to mark the corners
```

```
for y, x in coord:
    cv2.circle(img, (x, y), 3, (0, 0, 255), -1)

# Display the image with corners
cv2.imshow('Harris Corners', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Kode diatas memuat gambar antony.jpg, mengonversinya menjadi grayscale untuk analisis, dan mendeteksi sudut menggunakan metode Harris. Hasil deteksi sudut disaring dengan ambang batas, dan koordinat sudut yang terdeteksi digambarkan sebagai lingkaran merah pada gambar asli. Gambar yang menampilkan sudut-sudut tersebut kemudian ditampilkan dalam jendela dengan nama "Harris Corners". Setelah menekan sembarang tombol, jendela tersebut akan ditutup.

Output:

Program ini menggunakan algoritma Harris Corner Detection untuk mengidentifikasi titik-titik sudut pada gambar awal. Hasilnya adalah sebuah gambar yang menampilkan sudut-sudut terdeteksi, yang ditandai dengan lingkaran merah di sekitar koordinat masing-masing sudut. Proses ini memungkinkan visualisasi yang jelas dari fitur-fitur sudut penting dalam gambar asli.

Watermark

Watermark adalah tanda atau logo yang disisipkan dalam citra digital untuk identifikasi kepemilikan atau sumber. Fungsinya mencegah penyalahgunaan dan pembajakan melalui beberapa metode seperti watermark transparan, tersembunyi, atau sulit dihapus. Teknik ini membantu pemilik citra mempertahankan kontrol dan kepemilikan atas karya digital mereka, bahkan jika citra dimodifikasi.

Program:

```
# import required libraries
import cv2

# Read the image on which we are going to apply watermark
img = cv2.imread("antony.jpg")

# Read the watermark image
wm = cv2.imread("wm-2.jpg")

# height and width of the watermark image
h_wm, w_wm = wm.shape[:2]

# height and width of the image
h_img, w_img = img.shape[:2]

# calculate coordinates of center of image
center_x = int(w_img / 2)
center_y = int(h_img / 2)

# calculate region of interest (ROI) from top, bottom, right, and left
top_y = center_y - int(h_wm / 2)
left_x = center_x - int(w_wm / 2)
bottom_y = top_y + h_wm
right_x = left_x + w_wm

# add watermark to the image
roi = img[top_y:bottom_y, left_x:right_x]
result = cv2.addWeighted(roi, 1, wm, 0.3, 0)
img[top_y:bottom_y, left_x:right_x] = result

# display watermarked image
cv2.imshow("Watermarked Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Kode ini menambahkan watermark ke gambar antony.jpg. Pertama, gambar dan watermark dibaca, kemudian ukuran watermark dan gambar dihitung. Koordinat untuk menempatkan watermark di tengah gambar ditentukan.

Watermark ditambahkan ke area yang sesuai pada gambar menggunakan fungsi cv2.addWeighted untuk menggabungkan watermark dengan gambar asli dengan transparansi.

Hasil akhir ditampilkan dengan nama "Watermarked Image" dalam jendela yang akan ditutup setelah menekan sembarang tombol.

Output:

Steganografi

Steganografi adalah teknik untuk menyembunyikan pesan rahasia dalam pesan yang lebih besar, sehingga keberadaan dan isinya tidak terdeteksi. Tujuan utamanya adalah menjaga komunikasi rahasia antara dua pihak.

Berbeda dengan kriptografi, yang menyembunyikan isi pesan, steganografi menyembunyikan keberadaan pesan itu sendiri. Meskipun berbeda, steganografi dan kriptografi memiliki kesamaan, dan beberapa penulis bahkan menganggap steganografi sebagai bentuk kriptografi karena pesan rahasia tampak seperti pesan biasa.

Program:

```
import cv2
import numpy as np

# Convert various types to binary
def msg_to_bin(msg):
    if type(msg) == str:
        return ''.join([format(ord(i), "08b") for i in msg])
    elif type(msg) == bytes or type(msg) == np.ndarray:
        return [format(i, "08b") for i in msg]
    elif type(msg) == int or type(msg) == np.uint8:
```

```
        return format(msg, "08b")
    else:
        raise TypeError("Input type not supported")

# Hide secret message into the image
def hide_data(img, secret_msg):
    # Calculate the maximum bytes for encoding
    nBytes = img.shape[0] * img.shape[1] * 3 // 8
    print("Maximum Bytes for encoding:", nBytes)

    # Check if the number of bytes for encoding is less than the maximum
bytes in the image
    if len(secret_msg) > nBytes:
        raise ValueError("Error encountered: insufficient bytes, need a
bigger image or less data!!")

    secret_msg += '#####'  # Add delimiter to the message
    dataIndex = 0

    # Convert the input data to binary format
    bin_secret_msg = msg_to_bin(secret_msg)
    dataLen = len(bin_secret_msg)

    for values in img:
        for pixels in values:
            r, g, b = msg_to_bin(pixels)

            # Modify the LSB only if there is data remaining to store
            if dataIndex < dataLen:
                pixels[0] = int(r[:-1] + bin_secret_msg[dataIndex], 2)
                dataIndex += 1
            if dataIndex < dataLen:
                pixels[1] = int(g[:-1] + bin_secret_msg[dataIndex], 2)
                dataIndex += 1
            if dataIndex < dataLen:
                pixels[2] = int(b[:-1] + bin_secret_msg[dataIndex], 2)
                dataIndex += 1

            if dataIndex >= dataLen:
                break
        if dataIndex >= dataLen:
            break

    return img

# Extract hidden message from the image
def show_data(img):
    bin_data = ""

    for values in img:
        for pixels in values:
            r, g, b = msg_to_bin(pixels)
            bin_data += r[-1]
            bin_data += g[-1]
            bin_data += b[-1]

    allBytes = [bin_data[i: i + 8] for i in range(0, len(bin_data), 8)]
    decodedData = ""

    for bytes in allBytes:
        decodedData += chr(int(bytes, 2))
```

```
    if decodedData[-5:] == "#####":
        return decodedData[:-5]

# Encode data into the image
def encodeText():
    img_name = input("Enter image name (with extension): ")
    img = cv2.imread(img_name)
    print("The shape of the image is: ", img.shape)
    print("The original image is as shown below: ")

    resizedImg = cv2.resize(img, (500, 500))
    cv2.imshow("Original Image", resizedImg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    data = input("Enter data to be encoded: ")
    if len(data) == 0:
        raise ValueError('Data is Empty')

    file_name = input("Enter the name of the new encoded image (with
extension): ")
    encodedImage = hide_data(img, data)
    cv2.imwrite(file_name, encodedImage)

# Decode data from the image
def decodeText():
    img_name = input("Enter the name of the Steganographic image that has to
be decoded (with extension): ")
    img = cv2.imread(img_name)
    print("The Steganographic image is as follows: ")

    resizedImg = cv2.resize(img, (500, 500))
    cv2.imshow("Steganographic Image", resizedImg)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    text = show_data(img)
    return text

# Steganography menu
def steganography():
    n = int(input("Image Steganography \n1. Encode the data \n2. Decode the
data \nSelect the option: "))
    if n == 1:
        print("\nEncoding...")
        encodeText()
    elif n == 2:
        print("\nDecoding...")
        print("Decoded message is " + decodeText())
    else:
        raise Exception("Inserted value is incorrect!")

steganography()  # Run the steganography function
```

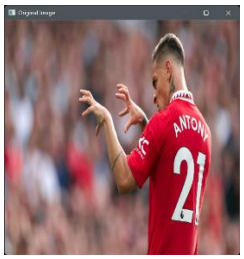
Program ini menggunakan teknik steganografi untuk menyembunyikan pesan rahasia dalam gambar. Fungsi msg_to_bin mengubah pesan rahasia menjadi format biner. Fungsi hide_data menyembunyikan pesan biner dalam bit terakhir dari nilai RGB setiap piksel gambar.

Sebelum menyembunyikan pesan, program memeriksa apakah ukuran pesan melebihi kapasitas gambar. Jika tidak, pesan ditambahkan ke gambar dengan modifikasi bit LSB (Least Significant Bit) dari piksel gambar.

Fungsi show_data digunakan untuk mengekstrak dan mengonversi kembali pesan yang disembunyikan dari bit LSB gambar. Program memiliki dua fungsi utama: encodeText untuk menyembunyikan pesan dalam gambar, dan decodeText untuk mengungkap pesan tersebut. Menu interaktif memungkinkan pengguna memilih antara encoding dan decoding pesan dalam gambar.

Output:

```
Image Steganography
1. Encode the data
2. Decode the data
Select the option: 1

Encoding...
Enter image name (with extension): antony.jpg
The shape of the image is:  (630, 1200, 3)
The original image is as shown below:
```



```
Image Steganography
1. Encode the data
2. Decode the data
Select the option: 2

Decoding...
Enter the name of the Steganographic image that has to be decoded (with extension): antony.jpg
The Steganographic image is as follows:
```