1. Hough Transform

```
import cv2
import numpy as np
import requests
from io import BytesIO
from google.colab.patches import cv2_imshow  # Fungsi khusus
Colab untuk menampilkan gambar

# URL gambar
url = 'https://cdn2.thecatapi.com/images/MTY3ODIyMQ.jpg'

# Mengambil gambar dari URL
response = requests.get(url)
image_bytes = BytesIO(response.content)
image_array = np.asarray(bytearray(image_bytes.read()),
dtype=np.uint8)
img = cv2.imdecode(image_array, cv2.IMREAD_COLOR)

# Konversi gambar ke skala abu-abu
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Deteksi tepi menggunakan Canny Edge Detection
edges = cv2.Canny(gray, 50, 150, apertureSize=3)

# Deteksi garis menggunakan Hough Line Transform
lines = cv2.HoughLines(edges, rho=1, theta=np.pi / 180,
threshold=150)

# Menampilkan garis pada gambar
if lines is not None:
    for rho, theta in lines[:, 0]:
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho
        x1 = int(x0 + 1000 * (-b))
        y1 = int(y0 + 1000 * (a))
        x2 = int(x0 - 1000 * (-b))
        y2 = int(y0 - 1000 * (a))
        cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)

# Menampilkan hasil
cv2_imshow(img)    # Gambar dengan garis
cv2_imshow(edges)  # Gambar hasil deteksi tepi
```

Penjelasan :

Kode ini bertujuan untuk mendeteksi garis pada sebuah gambar yang diambil dari URL. Berikut langkah-langkahnya:

Mengambil gambar: Gambar diambil dari URL menggunakan requests dan diubah menjadi format yang dapat diproses oleh OpenCV.

Konversi ke grayscale: Gambar diubah ke skala abu-abu menggunakan cv2.cvtColor.

Deteksi tepi: Deteksi tepi dilakukan menggunakan algoritma Canny Edge Detection (cv2.Canny).

Deteksi garis: Garis-garis dideteksi menggunakan Hough Line Transform (cv2.HoughLines).

Menampilkan garis: Garis-garis yang terdeteksi digambar pada gambar asli menggunakan cv2.line.

Menampilkan hasil: Gambar asli dengan garis yang terdeteksi dan gambar hasil deteksi tepi ditampilkan menggunakan cv2_imshow.

---

2. Template Matching

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow  # Untuk Google
Colab

# Path gambar utama dan template
main_image_path = '/content/drive/MyDrive/Colab
Notebooks/1.jpg'  # Gambar utama
template_image_path = '/content/drive/MyDrive/Colab
Notebooks/template1.jpg'  # Gambar template (objek yang
ingin dideteksi)

# Membaca gambar utama dan template
main_image = cv2.imread(main_image_path, cv2.IMREAD_COLOR)
template = cv2.imread(template_image_path, cv2.IMREAD_COLOR)

# Konversi ke grayscale
main_gray = cv2.cvtColor(main_image, cv2.COLOR_BGR2GRAY)
template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
```

```
# Ukuran template
w, h = template_gray.shape[::-1]

# Template Matching menggunakan metode cv2.TM_CCOEFF_NORMED
result = cv2.matchTemplate(main_gray, template_gray,
cv2.TM_CCOEFF_NORMED)

# Threshold untuk mencocokkan hasil
threshold = 0.8
loc = np.where(result >= threshold)

# Gambar kotak persegi di lokasi yang cocok
for pt in zip(*loc[::-1]):
    cv2.rectangle(main_image, pt, (pt[0] + w, pt[1] + h),
(0, 255, 0), 2)

# Menampilkan hasil
cv2_imshow(main_image)
```

Penjelasan

Kode ini melakukan Template Matching, yaitu teknik untuk menemukan lokasi suatu objek (template) dalam gambar yang lebih besar.

Langkah-langjah :

Membaca gambar: Kode membaca gambar utama dan gambar template menggunakan cv2.imread.

Konversi ke grayscale: Kedua gambar diubah ke skala abu-abu untuk mempermudah proses matching.

Template Matching: Fungsi cv2.matchTemplate digunakan untuk melakukan template matching. Metode yang digunakan adalah cv2.TM_CCOEFF_NORMED.

Thresholding: Hasil template matching difilter menggunakan threshold. Lokasi dengan nilai di atas threshold dianggap sebagai lokasi yang cocok.

Menggambar kotak: Kotak persegi digambar pada lokasi yang cocok untuk menandai objek yang ditemukan.

Menampilkan hasil: Gambar utama dengan kotak yang menandai objek ditampilkan menggunakan cv2_imshow.

Singkatnya, kode ini mencari objek (template) dalam gambar utama dan menandainya dengan kotak. Objek tersebut didefinisikan dalam gambar template.

Penjelasan tambajan:

cv2.TM_CCOEFF_NORMED adalah salah satu metode template matching yang tersedia di OpenCV. Metode ini menghitung korelasi antara template dan bagian-bagian gambar utama.

Nilai threshold menentukan seberapa mirip template dan bagian gambar utama agar dianggap sebagai kecocokan. Nilai yang lebih tinggi menghasilkan kecocokan yang lebih ketat.

---

3. Pyramid Gambar

```
import cv2
import numpy as np
import requests
from io import BytesIO
from google.colab.patches import cv2_imshow  # Untuk
menampilkan gambar di Colab

# Step 1: Ambil gambar dari URL
url = 'https://cdn2.thecatapi.com/images/MTY3ODIyMQ.jpg'  #
Ganti dengan URL gambar Anda
response = requests.get(url)
image_bytes = BytesIO(response.content)
image_array = np.asarray(bytearray(image_bytes.read()),
dtype=np.uint8)
image = cv2.imdecode(image_array, cv2.IMREAD_COLOR)

# Step 2: Tampilkan gambar asli
print("Gambar Asli:")
cv2_imshow(image)
```

```
# Step 3: Gaussian Pyramid - Mengurangi ukuran gambar
print("Gaussian Pyramid:")
layer = image.copy()
gp = [layer]  # List untuk menyimpan semua layer Gaussian
Pyramid

for i in range(3):  # Buat 3 level pyramid
    layer = cv2.pyrDown(layer)  # Perkecil gambar
    gp.append(layer)
    print(f"Level {i+1}:")
    cv2_imshow(layer)

# Step 4: Laplacian Pyramid - Deteksi detail antar level
print("Laplacian Pyramid:")
layer = gp[-1]  # Ambil level terakhir dari Gaussian Pyramid
lp = [layer]  # List untuk menyimpan Laplacian Pyramid

for i in range(3, 0, -1):  # Mulai dari level terakhir ke
level awal
    size = (gp[i-1].shape[1], gp[i-1].shape[0])  # Ukuran
gambar sebelumnya
    gaussian_expanded = cv2.pyrUp(gp[i], dstsize=size)  #
Perbesar gambar
    laplacian = cv2.subtract(gp[i-1], gaussian_expanded)  #
Kurangi dengan gambar sebelumnya
    lp.append(laplacian)
    print(f"Level {4-i}:")
    cv2_imshow(laplacian)

# Step 5: Rekonstruksi gambar menggunakan Laplacian Pyramid
print("Rekonstruksi Gambar:")
reconstructed = lp[0]
for i in range(1, len(lp)):
    size = (lp[i].shape[1], lp[i].shape[0])
    reconstructed = cv2.pyrUp(reconstructed, dstsize=size)
    reconstructed = cv2.add(reconstructed, lp[i])

cv2_imshow(reconstructed)
print("Selesai.")
```

Gaussian Pyramid dibentuk dengan mengurangi ukuran gambar secara bertahap menggunakan cv2.pyrDown. Setiap level pyramid menyimpan gambar yang lebih kecil dari level sebelumnya.

Laplacian Pyramid dibentuk dengan menghitung perbedaan antara level-level pada Gaussian Pyramid. Ini dilakukan dengan memperbesar gambar dari level bawah menggunakan cv2.pyrUp dan menguranginya dengan gambar dari level atas. Laplacian Pyramid menyimpan detail gambar pada setiap level.

Terakhir, gambar asli direkonstruksi dari Laplacian Pyramid dengan membalik proses pembentukannya. Gambar dari level teratas Laplacian Pyramid diperbesar dan ditambahkan dengan gambar dari level berikutnya, dan seterusnya hingga mencapai ukuran gambar asli.

Singkatnya, kode ini membuat representasi multi-skala dari gambar menggunakan Gaussian dan Laplacian Pyramid, kemudian merekonstruksi gambar asli dari representasi tersebut.

4. Hough Cirlce Transform

```
# Import library
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Path gambar lokal (ganti dengan path gambar Anda)
image_path = "/content/drive/MyDrive/Colab
Notebooks/HoughCircles.jpg"  # Ganti dengan nama atau path
gambar Anda

# Baca gambar
image = cv2.imread(image_path, cv2.IMREAD_COLOR)  # Baca
gambar dalam mode warna
output = image.copy()

# Ubah ke format grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Terapkan Gaussian Blur untuk mengurangi noise
gray_blurred = cv2.GaussianBlur(gray, (9, 9), 2)

# Hough Transform untuk deteksi lingkaran
detected_circles = cv2.HoughCircles(
    gray_blurred,           # Input gambar
    cv2.HOUGH_GRADIENT,     # Metode Hough Transform
    dp=1,                   # Resolusi akurasi
    minDist=30,             # Jarak minimal antar lingkaran
    param1=50,              # Threshold untuk edge detection
(Canny)
```

```
    param2=30,              # Threshold sensitivitas deteksi
lingkaran
    minRadius=10,           # Radius minimal lingkaran
    maxRadius=100           # Radius maksimal lingkaran
)

# Jika ada lingkaran yang terdeteksi, gambar lingkaran pada
gambar asli
if detected_circles is not None:
    detected_circles =
np.uint16(np.around(detected_circles))  # Bulatkan koordinat
lingkaran
    for circle in detected_circles[0, :]:
        x, y, radius = circle
        # Gambar lingkaran tepi
        cv2.circle(output, (x, y), radius, (0, 255, 0), 2)
        # Gambar titik pusat lingkaran
        cv2.circle(output, (x, y), 2, (0, 0, 255), 3)

# Tampilkan gambar hasil menggunakan Matplotlib
plt.figure(figsize=(10, 6))
plt.subplot(1, 2, 1)
plt.title("Gambar Asli")
plt.axis("off")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

plt.subplot(1, 2, 2)
plt.title("Deteksi Lingkaran")
plt.axis("off")
plt.imshow(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))
plt.show()
```

Kode ini mendeteksi lingkaran dalam gambar menggunakan Hough Circle Transform. Pertama, gambar dibaca dan diubah ke skala abu-abu, kemudian Gaussian Blur diterapkan untuk mengurangi noise. Selanjutnya, Hough Circle Transform digunakan untuk mendeteksi lingkaran dalam gambar. Jika lingkaran terdeteksi, lingkaran tersebut digambar pada gambar asli dengan warna hijau untuk tepinya dan merah untuk titik pusatnya. Terakhir, gambar asli dan gambar dengan lingkaran yang terdeteksi ditampilkan berdampingan.

5. Ekstrasi Warna dominan

```python
# Import library
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import os

# Path gambar lokal
image_path = "/content/drive/MyDrive/Colab
Notebooks/template1.jpg"  # Ganti dengan path gambar Anda

# Cek keberadaan gambar
if not os.path.exists(image_path):
    raise FileNotFoundError(f"Gambar tidak ditemukan di
path: {image_path}")

# Baca gambar
image = cv2.imread(image_path)
if image is None:
    raise ValueError("Gambar gagal dibaca. Periksa path
gambar Anda.")

# Konversi ke RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Fungsi untuk ekstraksi warna dominan
def get_dominant_colors(image, k=3):
    resized_image = cv2.resize(image, (100, 100),
interpolation=cv2.INTER_AREA)
    reshaped_image = resized_image.reshape(-1, 3)  # Ubah
menjadi array 2D

    # Cek jika jumlah unik warna lebih kecil dari k
    unique_colors = np.unique(reshaped_image, axis=0)
    if len(unique_colors) < k:
        raise ValueError(f"Jumlah warna unik
({len(unique_colors)}) lebih kecil dari k ({k})")

    # K-Means Clustering
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(reshaped_image)

    # Warna dominan dan jumlah piksel per kluster
    dominant_colors = kmeans.cluster_centers_.astype(int)
    counts = np.bincount(kmeans.labels_)

    # Urutkan warna dominan berdasarkan frekuensi
```

```
    sorted_indices = np.argsort(-counts)
    dominant_colors = dominant_colors[sorted_indices]
    counts = counts[sorted_indices]

    return dominant_colors, counts

# Ekstraksi warna dominan
k = 5
try:
    dominant_colors, counts = get_dominant_colors(image_rgb,
k)
except ValueError as e:
    print(e)
    exit()

# Visualisasi hasil
plt.figure(figsize=(14, 7))

# Tampilkan gambar asli
plt.subplot(1, 2, 1)
plt.title("Gambar Asli")
plt.axis("off")
plt.imshow(image_rgb)

# Tampilkan warna dominan dengan label
plt.subplot(1, 2, 2)
plt.title("Warna Dominan")
plt.axis("off")
palette = np.zeros((100, k * 100, 3), dtype=int)

for i in range(k):
    palette[:, i * 100:(i + 1) * 100] = dominant_colors[i]

plt.imshow(palette / 255)

# Tambahkan label untuk setiap warna dominan
for i in range(k):
    plt.text(i * 100 + 10, 50,
f"{dominant_colors[i]}\n({counts[i]} px)", color='white',
fontsize=10, ha='left')

plt.show()

# Print warna dominan
print("Warna Dominan (RGB) dan Jumlah Piksel:")
for i, (color, count) in enumerate(zip(dominant_colors,
counts)):
```

```
    print(f"Warna {i+1}: {color}, Jumlah Piksel: {count}")
```

Kode bertujuan untuk mengekstrak dan menampilkan warna dominan dari suatu gambar. Pertama, kode membaca gambar dan mengubahnya ke format RGB. Kemudian, fungsi get_dominant_colors dijalankan untuk melakukan proses ekstraksi warna. Di dalam fungsi ini, gambar diubah ukurannya dan di-reshape untuk mempermudah pemrosesan. Algoritma K-Means Clustering diterapkan untuk mengelompokkan piksel-piksel gambar berdasarkan warna, dan pusat-pusat cluster tersebut dianggap sebagai warna dominan. Terakhir, kode menampilkan gambar asli beserta palet warna yang merepresentasikan warna-warna dominan yang telah diekstrak, disertai informasi jumlah piksel untuk setiap warna.

6. Deteksi kontur

```
# Import library
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os

# Path gambar lokal
image_path = "/content/drive/MyDrive/Colab
Notebooks/template1.jpg"  # Ganti dengan path gambar Anda

# Cek keberadaan gambar
if not os.path.exists(image_path):
    raise FileNotFoundError(f"Gambar tidak ditemukan di
path: {image_path}")

# Baca gambar
image = cv2.imread(image_path)
if image is None:
    raise ValueError("Gambar gagal dibaca. Periksa path
gambar Anda.")

# Konversi gambar ke grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
# Lakukan blurring untuk mengurangi noise
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

# Deteksi tepi menggunakan Canny Edge Detection
edges = cv2.Canny(blurred_image, threshold1=50,
threshold2=150)

# Temukan kontur
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Buat salinan gambar asli untuk menggambar kontur
image_with_contours = image.copy()
cv2.drawContours(image_with_contours, contours, -1, (0, 255,
0), 2)  # Warna hijau untuk kontur

# Konversi ke RGB untuk ditampilkan dengan matplotlib
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_with_contours_rgb = cv2.cvtColor(image_with_contours,
cv2.COLOR_BGR2RGB)

# Visualisasi hasil
plt.figure(figsize=(14, 7))

# Tampilkan gambar asli
plt.subplot(1, 3, 1)
plt.title("Gambar Asli")
plt.axis("off")
plt.imshow(image_rgb)

# Tampilkan tepi hasil Canny
plt.subplot(1, 3, 2)
plt.title("Deteksi Tepi (Canny)")
plt.axis("off")
plt.imshow(edges, cmap='gray')

# Tampilkan gambar dengan kontur
plt.subplot(1, 3, 3)
plt.title("Gambar dengan Kontur")
plt.axis("off")
plt.imshow(image_with_contours_rgb)

plt.show()
```

Kode bertujuan untuk mendeteksi tepi objek dalam gambar dan menampilkannya beserta kontur objek tersebut. Pertama, gambar dibaca dan diubah ke skala abu-abu. Kemudian, Gaussian Blur diterapkan untuk mengurangi noise dan memperjelas tepi objek. Selanjutnya, algoritma Canny Edge Detection digunakan untuk mendeteksi tepi-tepi objek dalam gambar. Setelah tepi terdeteksi, fungsi cv2.findContours digunakan untuk menemukan kontur objek berdasarkan tepi-tepi tersebut. Terakhir, kode menampilkan tiga gambar secara berdampingan: gambar asli, gambar hasil deteksi tepi (Canny), dan gambar asli dengan kontur objek yang telah digambar di atasnya.