Analisis Simulasi Image Processing, Feature Detection, dan Feature Description Menggunakan Python dan OpenCV

Simulasi ini dilakukan menggunakan Python dan OpenCV di Google Colab dengan fokus pada tiga aspek utama: Image Processing, Feature Detection, dan Feature Description. Berikut adalah langkah-langkah analisisnya.

Langkah pertama adalah membaca gambar dalam format grayscale menggunakan fungsi `cv2.imread`. Grayscale digunakan untuk mengurangi dimensi data dan memudahkan pemrosesan citra berikutnya seperti deteksi tepi.

```
import cv2
import matplotlib.pyplot as plt

# Membaca gambar
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
plt.imshow(image, cmap='gray')
plt.title("Grayscale Image")
plt.show()
```

Kemudian, deteksi tepi dilakukan dengan algoritma Canny Edge Detection (`cv2.Canny`) untuk mengidentifikasi batas objek dalam gambar berdasarkan gradien intensitas. Hasil dari proses ini berupa gambar dengan garis-garis tepi yang jelas, yang menandai perubahan signifikan pada intensitas cahaya.

```
edges = cv2.Canny(image, 100, 200)
plt.imshow(edges, cmap='gray')
plt.title("Canny Edge Detection")
plt.show()
```

Selanjutnya, pada tahap Feature Detection, ORB (Oriented FAST and Rotated BRIEF) digunakan untuk mendeteksi fitur unik (keypoints) pada gambar. ORB diinisialisasi menggunakan `cv2.ORB_create`, dan fitur utama pada gambar dideteksi menggunakan metode `orb.detect`. Keypoints ini kemudian digambarkan pada gambar asli menggunakan fungsi `cv2.drawKeypoints` untuk memberikan visualisasi fitur yang terdeteksi.

```
# Inisialisasi detektor ORB
orb = cv2.ORB_create()
```

```
# Deteksi keypoints
keypoints = orb.detect(image, None)

# Gambarkan keypoints pada gambar asli
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None, color=(0, 255, 0))
plt.imshow(image_with_keypoints, cmap='gray')
plt.title("ORB Keypoints")
plt.show()
```

Pada tahap Feature Description, deskriptor fitur dihasilkan menggunakan metode `orb.detectAndCompute`. Deskriptor ini adalah representasi numerik dari fitur yang dapat digunakan untuk pencocokan gambar. Hasilnya meliputi lokasi keypoints dan deskriptor fitur yang masing-masing menggambarkan atribut unik dari setiap fitur yang terdeteksi.

```
# Deteksi dan deskripsi keypoints
keypoints, descriptors = orb.detectAndCompute(image, None)

print("Number of Keypoints Detected:", len(keypoints))
print("Descriptor Shape:", descriptors.shape)
```

Terakhir, pada tahap Feature Matching, algoritma Brute Force Matcher (`cv2.BFMatcher`) digunakan untuk mencocokkan deskriptor dari dua gambar atau dalam gambar yang sama. Pencocokan dilakukan dengan membandingkan jarak antar deskriptor, dan hasil pencocokan terbaik digambarkan menggunakan `cv2.drawMatches`. Proses ini penting dalam berbagai aplikasi seperti deteksi objek, rekonstruksi 3D, dan analisis citra medis.

```
# Membuat objek matcher
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
# Membandingkan fitur dengan dirinya sendiri (contoh)
matches = bf.match(descriptors, descriptors)

# Urutkan berdasarkan jarak
matches = sorted(matches, key=lambda x: x.distance)
# Gambarkan 10 pencocokan terbaik
image_matches = cv2.drawMatches(image, keypoints, image, keypoints, matches[:10], None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
plt.imshow(image_matches)
plt.title("Feature Matching")
plt.show()
```

Analisis Simulasi Webots: Image Processing, Feature Detection, dan Feature Description

Simulasi ini dilakukan untuk mengeksplorasi kemampuan Webots bersama Python dan OpenCV dalam bidang Image Processing, Feature Detection, dan Feature Description. Berikut adalah analisis untuk tiga tugas utama yang disimulasikan.

1. Visual Tracking dengan OpenCV

Simulasi ini bertujuan untuk melacak bola merah menggunakan thresholding HSV untuk mendeteksi warna dan pengendali P sederhana untuk mengarahkan robot. Kode berikut menunjukkan segmentasi warna merah dalam ruang HSV:

```
mask = cv2.inRange(hsv, np.array([0, 120, 70]), np.array([10, 255, 255]))
```

Segmentasi ini menghasilkan sebuah mask yang hanya menyimpan piksel dalam rentang warna merah. Setelah itu, kontrol berbasis P digunakan untuk mengarahkan robot mengikuti bola merah berdasarkan posisi error dalam gambar kamera.

```
error = (center_x - camera.getWidth() / 2)
correction = Kp * error
motor_left.setVelocity(5 - correction)
motor_right.setVelocity(5 + correction)
```

Kode di atas menghitung koreksi berdasarkan jarak horizontal bola dari pusat kamera, lalu menyesuaikan kecepatan roda robot.

2. Document Scanner Simulation

Tujuan dari simulasi ini adalah untuk mendeteksi dokumen pada conveyor belt, kemudian mengubahnya menjadi tampilan top-down menggunakan transformasi perspektif. Deteksi tepi dilakukan dengan algoritma Canny, seperti ditunjukkan di bawah ini:

```
edges = cv2.Canny(gray, 50, 150)
```

Hasil deteksi tepi ini digunakan untuk menemukan kontur dokumen terbesar. Kemudian, transformasi perspektif dilakukan untuk meluruskan dokumen ke tampilan top-down.

```
matrix = cv2.getPerspectiveTransform(src_pts, dst_pts)
warped = cv2.warpPerspective(image, matrix, (500, 700))
```

Kode ini menggunakan matriks transformasi untuk mentransformasikan dokumen menjadi bentuk persegi panjang.

3. Fruit Detection Robot

Simulasi ini mendeteksi buah berdasarkan warna (hijau untuk apel dan oranye untuk jeruk) dan menggunakan lengan robot untuk mengambil serta memindahkan buah ke lokasi yang sesuai. Segmentasi warna dilakukan dengan thresholding HSV, seperti berikut:

```
mask_green = cv2.inRange(hsv, np.array([35, 50, 50]), np.array([85, 255, 255]))
mask_orange = cv2.inRange(hsv, np.array([10, 100, 20]), np.array([25, 255, 255]))
```

Hasil segmentasi digunakan untuk menentukan apakah ada buah hijau atau oranye dalam gambar. Jika buah terdeteksi, gripper robot diaktifkan untuk mengambil buah tersebut, seperti ditunjukkan di bawah ini:

```
gripper.setPosition(1)  # Mengaktifkan gripper untuk mengambil buah
```

Setelah itu, lengan robot diarahkan ke lokasi penempatan yang sesuai dengan jenis buah yang terdeteksi.