HASIL ANALISIS

A. COLAB

1. Ekstraksi Garis dengan Hough Transform

Ekstraksi garis menggunakan Hough Transform adalah salah satu teknik dasar dalam pengolahan citra untuk mendeteksi garis lurus pada gambar.

Penjelasan Kode:

cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE):

- Fungsi ini membaca gambar dari disk dan mengonversinya ke grayscale.
- Grayscale digunakan untuk memudahkan deteksi tepi dan garis, karena hanya mempertimbangkan intensitas cahaya.

cv2.Canny(image, 50, 150):

- Deteksi tepi menggunakan algoritma Canny. Metode ini mencari tepi dengan mendeteksi perubahan intensitas cahaya yang tajam dalam gambar.

cv2.HoughLines(edges, 1, np.pi / 180, 150):

- Menerapkan Hough Transform untuk mendeteksi garis. Fungsi ini membutuhkan gambar tepi sebagai input.
- Parameter 1 adalah resolusi akumulator, np.pi / 180 adalah resolusi sudut, dan 150 adalah threshold untuk deteksi garis.

cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 2):

- Menggambar garis yang terdeteksi pada gambar dengan warna merah (BGR (0, 0, 255)), dengan ketebalan 2 piksel.

cv2.cvtColor(image, cv2.COLOR_BGR2RGB):

- Mengonversi gambar dari format BGR ke RGB untuk kompatibilitas dengan matplotlib.

2. **Template Matching untuk Deteksi Objek**

Template Matching adalah teknik untuk mencari objek di dalam gambar berdasarkan template yang sudah ada.

Penjelasan Kode:

cv2.cvtColor(image, cv2.COLOR_BGR2GRAY):

- Mengonversi gambar utama dan template menjadi grayscale untuk memudahkan pencocokan.

cv2.matchTemplate(gray_image,gray_template,cv2.TM_CCOEFF_NORMED):

- Fungsi matchTemplate() mencari kecocokan antara gambar utama dan template menggunakan metode TM_CCOEFF_NORMED, yang menghasilkan hasil dengan skor korelasi tertinggi.

np.where(result >= threshold):

- Menemukan lokasi kecocokan template yang memiliki skor lebih besar dari threshold (0.8), yang berarti area yang cocok.

cv2.rectangle(image, pt, (pt[0] + template.shape[1], pt[1] + template.shape[0]), (0, 255, 0), 2):

- Menandai lokasi kecocokan dengan persegi panjang hijau untuk menunjukkan objek yang terdeteksi.

3. **Pembuatan Pyramid Gambar**
   Pyramid gambar digunakan untuk memperkecil gambar secara bertahap untuk analisis multi-skala.

   Penjelasan Kode:

   cv2.pyrDown(layer):

   - Fungsi ini membuat salinan gambar dengan ukuran yang lebih kecil, menurunkan resolusi gambar secara bertahap.

   plt.subplots(1, 6, figsize=(20, 10)):

   - Membuat subplots untuk menampilkan 6 layer gambar pyramid.

   axes[i].imshow(cv2.cvtColor(gp[i], cv2.COLOR_BGR2RGB)):

   - Mengonversi gambar dari format BGR ke RGB untuk visualisasi menggunakan matplotlib.

4. **Deteksi Lingkaran Menggunakan Hough Transform**
   Deteksi lingkaran dalam gambar menggunakan Hough Circle Transform untuk menemukan lingkaran yang ada.

   Penjelasan Kode:

   cv2.GaussianBlur(image, (5, 5), 0):

   - Menggunakan filter Gaussian untuk mengurangi noise dalam gambar sebelum melakukan deteksi lingkaran.

   cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, dp=1.2, minDist=30, param1=50, param2=30, minRadius=10, maxRadius=100):

   - Mendeteksi lingkaran dengan menggunakan Hough Transform. Parameter dp, minDist, dan lainnya mengontrol akurasi dan rentang lingkaran yang terdeteksi.

   cv2.circle(image, (x, y), r, (0, 255, 0), 4):

   - Menggambar lingkaran yang terdeteksi pada gambar.

5. **Ekstraksi Warna Dominan pada Gambar**
   Menggunakan **K-means clustering** untuk mengekstraksi warna dominan pada gambar.

   Penjelasan Kode:
   image.reshape((-1, 3)):
   - Mengubah gambar menjadi bentuk 2D, dengan setiap baris mewakili satu piksel dan kolom mewakili nilai RGB.

   KMeans(n_clusters=1):
   - Menggunakan algoritma **K-means** untuk mengelompokkan piksel berdasarkan warna dan menemukan warna dominan.

6. **Deteksi Kontur pada Gambar**
   Deteksi kontur digunakan untuk mengidentifikasi batas objek dalam gambar.

   Penjelasan Kode:
   cv2.findContours(thresholded,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE):
   - Mencari kontur dalam gambar biner yang diperoleh dari proses thresholding.

   cv2.drawContours(output, contours, -1, (0, 255, 0), 2):
   - Menggambar kontur yang terdeteksi dengan warna hijau pada gambar.

B. SIMULASI

Simulasi yang diberikan adalah sebuah program untuk mengendalikan robot di Webots, menggunakan **Lidar (Light Detection and Ranging)** dan **sensor ultrasonik** untuk penghindaran tabrakan, serta pengaturan kecepatan roda untuk navigasi.

Penjelasan Kode Program :

A. Impor dan Inisialisasi

from controller import Robot: Mengimpor kelas Robot dari library controller Webots. Kelas ini memungkinkan pengendalian robot dalam simulasi Webots.
TIME_STEP = 32: Menetapkan konstanta TIME_STEP yang menentukan interval waktu simulasi (dalam milidetik) di mana kontrol robot diperbarui. Ini mengatur frekuensi eksekusi langkah dalam simulasi, di sini adalah 32 ms.

B. Inisialisasi Sensor

`LEFT = 0, RIGHT = 1`: Indeks untuk sensor jarak kiri dan kanan. Variabel ini akan digunakan untuk merujuk sensor mana yang akan diakses.
`robot = Robot()`: Membuat objek `robot` yang memungkinkan kita untuk mengakses dan mengontrol robot dalam simulasi.
`lidar = robot.getDevice('lidar')`: Menginisialisasi perangkat **Lidar** pada robot. Lidar digunakan untuk mendeteksi jarak objek di sekitar robot dan memberikan data untuk navigasi.
`lidar.enable(TIME_STEP)`: Mengaktifkan Lidar agar bisa mulai mengirimkan data pada setiap langkah simulasi. Parameter `TIME_STEP` digunakan untuk mengatur seberapa sering data lidar diperbarui.
`lidar.enablePointCloud()`: Mengaktifkan fitur Point Cloud dari Lidar, yang mengizinkan untuk mendapatkan data 3D dari lingkungan sekitar.
`us = [robot.getDevice('us0'), robot.getDevice('us1')]`: Inisialisasi dua sensor ultrasonik (sensor jarak), satu untuk sisi kiri (`us0`) dan satu untuk sisi kanan (`us1`).
`for sensor in us: sensor.enable(TIME_STEP)`: Mengaktifkan kedua sensor ultrasonik tersebut pada setiap langkah simulasi.

C. Inisialisasi Motor

`left_motor = robot.getDevice('left wheel motor')`: Menginisialisasi motor roda kiri.

`right_motor = robot.getDevice('right wheel motor')`: Menginisialisasi motor roda kanan.

`setPosition(float('inf'))`: Mengatur posisi motor menjadi "tak terbatas", yang berarti motor beroperasi dalam mode kecepatan (bukan mode posisi).

`setVelocity(0.0)`: Mengatur kecepatan awal motor kiri dan kanan ke 0.0 (artinya robot diam pada awalnya).

D. Koefisien Empiris untuk Penghindaran Tabrakan

`coefficients = [[12.0, -6.0], [-10.0, 8.0]]`: Matriks koefisien yang digunakan untuk mengubah pembacaan sensor menjadi kecepatan roda. Koefisien ini empiris, berarti ditentukan berdasarkan percobaan untuk mencapai kinerja yang optimal dalam menghindari tabrakan.

`base_speed = 6.0`: Kecepatan dasar motor, di mana kecepatan roda kiri dan kanan akan disesuaikan berdasarkan pembacaan sensor.

E. Fungsi untuk Membaca Data Lidar

`extract_lidar_data()`: Fungsi ini digunakan untuk mendapatkan data dari Lidar.

`lidar.getRangeImage()`: Mengambil gambar jarak dari Lidar yang berisi data tentang jarak objek di sekitar robot dalam bentuk array.

`print(f"Lidar Data {lidar_data[10]}...")`: Menampilkan data Lidar untuk indeks ke-10 sebagai contoh (untuk mengurangi jumlah data yang ditampilkan).

F. Fungsi untuk Membaca Data Sensor Jarak (Ultrasonik)

`read_distance_sensors()`: Fungsi ini membaca nilai dari sensor ultrasonik untuk sisi kiri dan kanan.

`distances = [sensor.getValue() for sensor in us]`: Membaca nilai dari kedua sensor jarak dan menyimpannya dalam list `distances`.

`print(f"Distance Sensor Readings Left={distances[LEFT]:.2f}, Right={distances[RIGHT]:.2f}")`: Menampilkan hasil pembacaan sensor jarak kiri dan kanan dengan format dua angka desimal.

G. Fungsi untuk Menghitung Kecepatan Berdasarkan Data Sensor

`compute_speeds(us_values)`: Fungsi ini digunakan untuk menghitung kecepatan roda berdasarkan nilai dari sensor jarak (ultrasonik).

`speed[i] += us_values[k] * coefficients[i][k]`: Iterasi melalui nilai sensor dan koefisien empiris untuk menghitung kecepatan kiri dan kanan dengan mengalikan pembacaan sensor dengan koefisien yang sesuai.

`return speed`: Mengembalikan nilai kecepatan kiri dan kanan yang dihitung.

H. Loop Utama untuk Simulasi

`while robot.step(TIME_STEP) != -1:`: Ini adalah loop utama yang dijalankan pada setiap langkah simulasi. `robot.step(TIME_STEP)` memajukan simulasi sebesar `TIME_STEP` dan mengecek apakah simulasi masih berjalan.

`lidar_data = extract_lidar_data()`: Membaca data Lidar setiap iterasi.

`us_values = read_distance_sensors()`: Membaca data sensor jarak setiap iterasi.

`speeds = compute_speeds(us_values)`: Menghitung kecepatan berdasarkan data sensor.

`left_motor.setVelocity(base_speed + speeds[LEFT])`: Mengatur kecepatan roda kiri berdasarkan kecepatan yang dihitung.

`right_motor.setVelocity(base_speed + speeds[RIGHT])`: Mengatur kecepatan roda kanan berdasarkan kecepatan yang dihitung.

`robot.cleanup()`: Membersihkan dan mengatur ulang status robot setelah simulasi selesai.