Nama : Hamdan Syaifuddin Zuhri
NIM : 1103220220
Kelas : TK-45-G09

Analisis Tugas Week 11

A. Simulasi Information Extraction

1. Ekstraksi Garis dengan Hough Transform

Simulasi ini bertujuan untuk mendeteksi garis pada gambar menggunakan algoritma Transformasi Hough. Langkah pertama adalah membaca gambar input menggunakan cv2.imread(), lalu mengubahnya menjadi gambar skala abu-abu menggunakan cv2.cvtColor(). Konversi ini penting karena deteksi garis lebih efektif pada gambar hitam-putih daripada pada gambar berwarna. Selanjutnya, algoritma deteksi tepi seperti Canny Edge Detection diterapkan pada gambar skala abu-abu menggunakan fungsi cv2.Canny(). Hasil deteksi tepi ini adalah gambar biner yang menunjukkan bagian gambar yang memiliki perubahan intensitas yang tajam.

Setelah itu, kami menggunakan metode Transformasi Hough untuk mendeteksi garis pada gambar. Fungsi cv2.HoughLines() digunakan dengan parameter ruang Hough, di mana parameter pertama adalah resolusi ruang akumulator, yang kedua adalah sudut dalam radian, dan yang ketiga adalah ambang deteksi. Garis-garis yang terdeteksi kemudian digambar pada gambar asli menggunakan cv2.line(), yang memungkinkan kita untuk melihat garis-garis yang ditemukan. Hasil dari simulasi ini adalah gambar dengan garis-garis yang terdeteksi, yang ditampilkan pada gambar dengan warna merah.

2. Template Matching untuk Deteksi Objek

Template Matching adalah teknik yang digunakan untuk mendeteksi lokasi objek dalam sebuah gambar input yang memiliki kesamaan dengan gambar template. Dalam simulasi ini, langkah pertama adalah membaca gambar template dan gambar input menggunakan fungsi cv2.imread(). Gambar input kemudian diubah menjadi grayscale untuk mempermudah proses pencocokan. Selanjutnya, fungsi cv2.matchTemplate() digunakan untuk mencari lokasi template dalam gambar input. Fungsi ini menghasilkan citra yang merepresentasikan hasil perhitungan korelasi antara template dan area-area dalam gambar. Teknik ini memanfaatkan metode normalisasi koefisien korelasi agar proses pencocokan template lebih akurat. Untuk menentukan posisi terbaik template dalam gambar, kita menggunakan fungsi cv2.minMaxLoc(), yang mengidentifikasi lokasi dengan nilai

maksimum dari hasil perhitungan korelasi. Posisi ini menunjukkan letak optimal objek yang sesuai dengan template. Setelah posisi tersebut ditemukan, kotak pembatas digambar di sekitar objek terdeteksi menggunakan fungsi cv2.rectangle(). Hasil akhir simulasi adalah gambar dengan kotak yang menunjukkan lokasi objek berdasarkan kecocokan dengan template.

3. **Pembuatan Pyramid Gambar**

Pyramid Gambar adalah metode yang digunakan untuk menghasilkan gambar dengan berbagai tingkat resolusi, yang berguna untuk analisis multi-skala atau saat bekerja dengan gambar dalam tingkat detail yang berbeda. Dalam simulasi ini, kita memanfaatkan dua fungsi dari OpenCV, yaitu cv2.pyrDown() dan cv2.pyrUp(). Fungsi cv2.pyrDown() digunakan untuk mengecilkan ukuran gambar, menghasilkan versi gambar input dengan resolusi lebih rendah. Sebaliknya, fungsi cv2.pyrUp() digunakan untuk memperbesar kembali gambar yang telah dikecilkan, sehingga ukurannya kembali seperti semula. Teknik ini memungkinkan pembuatan representasi gambar pada berbagai resolusi, yang sangat berguna dalam pemrosesan gambar lanjutan, seperti mendeteksi objek pada skala yang berbeda.

4. **Deteksi Lingkaran Menggunakan Hough Transform**

Deteksi lingkaran pada gambar dilakukan dengan menggunakan metode Hough Transform, yaitu teknik matematika untuk mendeteksi objek melingkar. Dalam simulasi ini, kami menggunakan fungsi cv2.HoughCircles() yang memungkinkan kami mendeteksi lingkaran pada gambar skala abu-abu. Fungsi ini memerlukan beberapa parameter penting, seperti dp yang merupakan rasio resolusi akumulator dan minDist yang menentukan jarak minimum antara pusat lingkaran yang terdeteksi. Setelah lingkaran terdeteksi, kami menggunakan cv2.circle() untuk menggambar lingkaran pada gambar asli dengan warna hijau. Hasil simulasi ini adalah gambar yang menampilkan lingkaran yang terdeteksi pada gambar masukan.

5. **Ekstraksi Warna Dominan pada Gambar**

Untuk mendeteksi warna dominan pada citra, simulasi ini menggunakan metode color clustering dengan K-Means. Citra terlebih dahulu diubah menjadi array dua dimensi dengan reshape(), sehingga setiap piksel pada citra menjadi satu baris pada array tersebut. Kemudian, algoritma K-Means dari library sklearn.cluster digunakan untuk mengelompokkan piksel-piksel citra berdasarkan warnanya. Dengan menentukan jumlah cluster, misalnya 3 untuk mengidentifikasi tiga warna dominan, algoritma K-Means akan

mengelompokkan piksel-piksel yang memiliki warna serupa. Setelah proses clustering, warna dominan pada citra dapat diidentifikasi berdasarkan pusat cluster yang terdeteksi. Hasilnya adalah identifikasi warna-warna utama pada citra yang menggambarkan warna dominan tersebut.

6. **Deteksi Kontur pada Gambar**

Deteksi kontur pada citra merupakan salah satu teknik penting untuk mengenali bentuk objek pada citra. Pada simulasi ini, kami menggunakan fungsi cv2.findContours() yang memungkinkan kami menemukan kontur berdasarkan hasil deteksi tepi yang telah dilakukan sebelumnya. Metode ini bekerja dengan cara menemukan batas objek pada citra, yang kemudian direpresentasikan sebagai garis kontur. Setelah kontur ditemukan, kami dapat menggambar kontur tersebut menggunakan cv2.drawContours(). Dengan menggambar kontur menggunakan warna tertentu, kami dapat dengan mudah mengidentifikasi bentuk objek pada citra. Hasil simulasi ini berupa citra yang menampilkan kontur objek yang terdeteksi berwarna hijau di sepanjang batas objek.

Kesimpulan:

Simulasi Information Extraction yang dilakukan pada minggu ke-11 mencakup berbagai teknik penting dalam pengolahan citra digital, yang memberikan kemampuan untuk mengekstrak informasi dan mendeteksi objek dalam citra. Hough Transform digunakan untuk mendeteksi garis dan lingkaran, sedangkan Template Matching membantu dalam mendeteksi objek berdasarkan template. Teknik Image Pyramid berguna untuk analisis multiskala, sedangkan dominant color extraction dilakukan dengan menggunakan metode clustering K-Means untuk mengenali warna-warna utama dalam citra. Selain itu, contour detection berperan penting dalam mengidentifikasi bentuk objek. Semua teknik ini memungkinkan ekstraksi informasi yang lebih kaya dari citra dan sangat berguna untuk aplikasi seperti pengenalan pola, pengolahan citra, dan analisis visual lebih lanjut.

B. Simulasi Webots Berfokus Lidar Data Extraction and Obstacle Detection

Program ini bertujuan untuk menggunakan data LIDAR untuk mendeteksi rintangan di sekitar robot dan menghindarinya dengan pengendali berbasis jarak sederhana

1. **Inisialisasi Robot e-puck dan Sensor LIDAR**

- Robot Initialization: Program dimulai dengan menginisialisasi objek Robot() dari pustaka Webots, yang memungkinkan kita mengontrol robot selama simulasi. robot =

Robot() menciptakan instance robot yang akan digunakan untuk mengakses perangkat-perangkat di robot, seperti sensor dan motor. robot = Robot()

- LIDAR Initialization: LIDAR adalah sensor yang digunakan untuk mengukur jarak ke objek di sekitar robot. Dalam kode ini, LIDAR diakses dengan robot.getDevice('lidar'), dan diaktifkan dengan metode enable(TIME_STEP) yang menentukan langkah waktu simulasi yang digunakan untuk pembacaan sensor. Fungsi enablePointCloud() memungkinkan kita untuk mendapatkan data dalam bentuk titik 3D, yang sangat berguna untuk visualisasi lebih lanjut atau analisis lanjutan. lidar = robot.getDevice('lidar') lidar.enable(TIME_STEP) lidar.enablePointCloud()
- Motor Initialization: Motor kiri dan kanan diinisialisasi dengan robot.getDevice('left wheel motor') dan robot.getDevice('right wheel motor'). Untuk pengendalian kecepatan, mode posisi motor disetel ke 'inf' untuk mengaktifkan kontrol kecepatan motor (bukan posisi). Kecepatan motor diatur menggunakan setVelocity(), dengan kecepatan awal diatur ke 0.0 (tidak bergerak) sampai kecepatan dihitung dan diatur lebih lanjut dalam simulasi. left_motor = robot.getDevice('left wheel motor') right_motor = robot.getDevice('right wheel motor') left_motor.setPosition(float('inf')) # Aktifkan mode kecepatan right_motor.setPosition(float('inf')) # Aktifkan mode kecepatan left_motor.setVelocity(0.0) right_motor.setVelocity(0.0)

2. Pembacaan Data LIDAR

- Fungsi extract_lidar_data(): Fungsi ini digunakan untuk membaca data dari sensor LIDAR. lidar.getRangeImage() mengembalikan gambar jarak dari sensor LIDAR, yang merupakan array dari nilai jarak yang diukur oleh LIDAR dalam bentuk citra dua dimensi. Dalam kode ini, hanya data jarak pertama yang ditampilkan dengan print(fLidar Data {lidar_data[10]}...) untuk memberi gambaran kasar mengenai data yang dibaca dari sensor LIDAR. def extract_lidar_data(): lidar_data = lidar.getRangeImage() print(f"Lidar Data {lidar_data[10]}...") # Menampilkan 10 data pertama return lidar_data.
- Fungsi read_distance_sensors(): Fungsi ini digunakan untuk membaca nilai dari sensor jarak ultrasonic. Pembacaan dilakukan dengan sensor.getValue(), dan hasilnya adalah daftar dua nilai yang menunjukkan jarak yang terdeteksi oleh masing-masing sensor kiri dan kanan. Hasil ini dicetak ke layar dengan format yang jelas agar kita dapat memonitor pembacaan sensor secara real-time. def read_distance_sensors(): distances

= [sensor.getValue() for sensor in us] print(f"Distance Sensor Readings Left={distances[LEFT]:.2f}, Right={distances[RIGHT]:.2f}") return distances

3. **Perhitungan Kecepatan Berdasarkan Data Sensor**

   Fungsi compute_speeds(): Fungsi ini digunakan untuk menghitung kecepatan motor kiri dan kanan berdasarkan pembacaan dari sensor jarak. Koefisien empiris coefficients digunakan untuk mengubah nilai sensor menjadi perubahan kecepatan. Setiap nilai sensor diperhitungkan untuk menentukan kecepatan masing-masing motor, dengan menggunakan persamaan yang melibatkan koefisien yang telah ditentukan. def compute_speeds(us_values): speed = [0.0, 0.0] for i in range(2): for k in range(2): speed[i] += us_values[k] * coefficients[i][k] return speed

4. **Loop Utama**

   Loop Kontrol Robot: Dalam loop utama while robot.step(TIME_STEP) != -1:, program akan terus berulang selama simulasi berjalan. Pada setiap langkah simulasi, program akan: o Membaca data dari sensor LIDAR dengan memanggil extract_lidar_data(). o Membaca data dari sensor jarak dengan memanggil read_distance_sensors(). o Menghitung kecepatan motor berdasarkan pembacaan sensor menggunakan compute_speeds(). o Mengatur kecepatan motor kiri dan kanan dengan left_motor.setVelocity() dan right_motor.setVelocity(), sehingga robot dapat bergerak dengan kecepatan yang disesuaikan dengan jarak terdeteksi oleh sensor. while robot.step(TIME_STEP) != -1: lidar_data = extract_lidar_data() us_values = read_distance_sensors() speeds = compute_speeds(us_values) left_motor.setVelocity(base_speed + speeds[LEFT]) right_motor.setVelocity(base_speed + speeds[RIGHT])

5. **Pembersihan Setelah Simulasi**

   Setelah simulasi selesai, fungsi robot.cleanup() dipanggil untuk membersihkan semua perangkat dan membebaskan sumber daya yang digunakan oleh robot selama simulasi. Ini memastikan bahwa sumber daya dibebaskan dengan benar setelah simulasi berakhir. robot.cleanup()