ANALISIS TUGAS ROBOTIK WEEK 11

1. Simulasi Information Extraction di Google Colab

1.1 Ekstraksi Garis dengan Hough Transform

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

Setelah itu, kita menggunakan metode Hough Transform untuk mendeteksi garis-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 batas deteksi. Fungsi ini menghasilkan koordinat parameter garis dalam ruang Hough. 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.

1.2 Template Matching untuk Deteksi Objek

Template Matching adalah metode yang digunakan untuk menemukan lokasi objek yang sama dengan template gambar dalam sebuah gambar input. Dalam simulasi ini, kita pertama-tama membaca gambar template dan gambar input menggunakan cv2.imread(). Gambar input kemudian dikonversi menjadi grayscale untuk memudahkan proses pencocokan. Selanjutnya, kita menggunakan fungsi cv2.matchTemplate() untuk mencari keberadaan template dalam gambar. Fungsi ini akan mengembalikan citra hasil perhitungan korelasi antara template dan bagian-bagian gambar. Teknik ini menggunakan metode normalisasi koefisien korelasi untuk mencocokkan template secara efektif. Untuk menemukan posisi terbaik dari template dalam gambar, kita menggunakan cv2.minMaxLoc(), yang memberikan posisi nilai maksimum dalam hasil perhitungan korelasi. Posisi ini menunjukkan lokasi terbaik dari objek yang sesuai dengan template. Setelah lokasi ditemukan, kita menggambar kotak pembatas di sekitar objek yang terdeteksi menggunakan cv2.rectangle(). Hasil dari simulasi ini adalah gambar dengan kotak yang menandai posisi objek yang terdeteksi berdasarkan kecocokan dengan template.

1.3 Pembuatan Pyramid Gambar

Pyramid Gambar adalah teknik untuk membuat gambar dengan resolusi yang berbeda, yang berguna untuk analisis multi-skala atau saat bekerja dengan gambar pada berbagai tingkat detail. Dalam simulasi ini, kita menggunakan dua fungsi dari OpenCV, yaitu cv2.pyrDown() dan cv2.pyrUp(). Fungsi cv2.pyrDown() digunakan untuk mengurangi ukuran gambar, menghasilkan versi yang lebih kecil dari gambar input dengan resolusi yang lebih rendah. Fungsi cv2.pyrUp() digunakan untuk memperbesar kembali gambar yang telah dikecilkan, mengembalikannya ke ukuran aslinya. Proses ini membantu dalam menciptakan representasi gambar pada berbagai tingkat resolusi, yang dapat bermanfaat dalam pemrosesan gambar lebih lanjut, seperti deteksi objek pada skala yang berbeda.

1.4 Deteksi Lingkaran Menggunakan Hough Transform

Deteksi lingkaran dalam gambar dilakukan menggunakan metode Hough Transform, yang merupakan teknik matematis untuk mendeteksi objek berbentuk lingkaran. Pada simulasi ini, kita menggunakan fungsi cv2.HoughCircles(), yang memungkinkan kita untuk mendeteksi lingkaran pada gambar grayscale. Fungsi ini memerlukan beberapa parameter penting, seperti dp, yang merupakan rasio resolusi akumulator, dan minDist, yang menentukan jarak minimum antar pusat lingkaran yang terdeteksi. Setelah lingkaran terdeteksi, kita menggunakan cv2.circle() untuk menggambar lingkaran pada gambar asli dengan warna hijau. Hasil dari simulasi ini adalah gambar yang menampilkan lingkaran-lingkaran yang terdeteksi dalam gambar input.

1.5 Ekstraksi Warna Dominan pada Gambar

Untuk mendeteksi warna dominan dalam gambar, simulasi ini menggunakan metode clustering warna dengan K-Means. Gambar pertama-tama diubah menjadi array dua dimensi dengan reshape(), sehingga setiap piksel gambar menjadi satu baris dalam array. Kemudian, algoritma K-Means dari pustaka sklearn.cluster digunakan untuk mengelompokkan piksel-piksel gambar berdasarkan warna mereka. Dengan menentukan jumlah cluster, misalnya 3 untuk mengidentifikasi tiga warna dominan, algoritma K-Means akan mengelompokkan piksel yang memiliki warna serupa. Setelah proses clustering, warna dominan dalam gambar dapat diketahui berdasarkan pusat cluster yang terdeteksi. Hasilnya adalah identifikasi warna-warna utama dalam gambar yang menggambarkan warna dominan.

1.6 Deteksi Kontur pada Gambar

Deteksi kontur pada gambar merupakan teknik penting untuk mengenali bentuk-bentuk objek dalam citra. Dalam simulasi ini, kita menggunakan fungsi cv2.findContours() yang memungkinkan kita untuk menemukan kontur berdasarkan hasil deteksi tepi yang sudah dilakukan sebelumnya. Metode ini bekerja dengan mencari batas-batas objek dalam gambar, yang kemudian diwakili sebagai garis kontur. Setelah kontur ditemukan, kita dapat menggambar kontur tersebut menggunakan cv2.drawContours(). Dengan menggambar kontur menggunakan warna tertentu, kita dapat dengan mudah mengidentifikasi bentuk-bentuk objek yang ada dalam gambar. Hasil dari simulasi ini adalah gambar yang menampilkan kontur objek yang terdeteksi dengan warna hijau di sepanjang batas objek.

Kesimpulan: Simulasi Information Extraction yang dilakukan pada minggu ke-11 ini mencakup berbagai teknik penting dalam pengolahan citra digital, yang memberikan kemampuan untuk mengekstrak informasi dan mendeteksi objek dalam gambar. Hough Transform digunakan untuk mendeteksi garis dan lingkaran, sementara Template Matching membantu dalam mendeteksi objek berdasarkan template. Teknik Pyramid Gambar berguna untuk analisis multi-skala, sementara ekstraksi warna dominan dilakukan menggunakan metode clustering dengan K-Means untuk mengenali warna utama dalam gambar. Selain itu, deteksi kontur memainkan peran penting dalam mengidentifikasi bentuk objek. Semua teknik ini memungkinkan ekstraksi informasi yang lebih kaya dari gambar dan sangat bermanfaat untuk aplikasi seperti pengenalan pola, pemrosesan gambar, dan analisis visual lebih lanjut.

2. **simulasi Webots** berfokus pada **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. Saya akan menjelaskan setiap bagian kode dengan rinci.

   2.1 Inisialisasi Robot dan Perangkat Sensor

   - **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()

   - **Sensor Jarak (Ultrasonic):** Dua sensor jarak ultrasonic digunakan untuk mendeteksi keberadaan objek di sekitar robot. Sensor-sensor ini diakses menggunakan robot.getDevice('us0') dan robot.getDevice('us1'). Sensor-sensor ini diaktifkan dengan sensor.enable(TIME_STEP), yang memungkinkan sensor memberikan pembacaan pada setiap langkah simulasi.

     us = [robot.getDevice('us0'), robot.getDevice('us1')]

     for sensor in us:

       sensor.enable(TIME_STEP)

- **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 dan Sensor Jarak

- **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. Penghitungan 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:
  - Membaca data dari sensor LIDAR dengan memanggil extract_lidar_data().
  - Membaca data dari sensor jarak dengan memanggil read_distance_sensors().
  - Menghitung kecepatan motor berdasarkan pembacaan sensor menggunakan compute_speeds().
  - 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

- **Pembersihan Memori:** 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()
```

Kesimpulan: Program ini memungkinkan robot untuk mendeteksi rintangan di sekitarnya menggunakan sensor LIDAR dan sensor jarak ultrasonic. Berdasarkan pembacaan sensor, robot dapat menghitung kecepatan motor untuk menghindari tabrakan dengan objek di sekitar. Kecepatan motor dihitung

menggunakan koefisien empiris yang tergantung pada nilai pembacaan sensor. Program ini menciptakan sistem penghindaran tabrakan berbasis jarak yang sederhana dan efisien dalam simulasi Webots.