TECHNICAL REPORT ROBOTIKA DAN SISTEM CERDAS

ROBOT PENGHINDAR RINTANGAN DENGAN NAVIGASI A* MENGGUNAKAN WEBOTS DAN ROS2

Oleh:
Alvin Yoga Nugraha (1103204159)

PROGRAM STUDI TEKNIK KOMPUTER
FAKULTAS TEKNIK ELEKTRO
UNIVERSITAS TELKOM
2023

PENDAHULUAN
Robotika adalah bidang teknologi yang terus berkembang dan memungkinkan kita untuk mengembangkan sistem yang lebih cerdas dan mandiri. Pada laporan ini, kami akan membahas pengembangan robot penghindar rintangan dengan navigasi A* menggunakan ROS2 dan Python. Robot tersebut dilengkapi dengan sensor jarak ultrasonik yang digunakan untuk mendeteksi rintangan di sekitar robot dan menghindarinya.

LATAR BELAKANG
ROS2 adalah platform open source yang digunakan untuk pengembangan aplikasi robotika dan sistem kontrol. ROS2 menyediakan berbagai macam library, algoritma dan tools untuk memudahkan pengembangan sistem robotika dan kontrol. Navigasi A* adalah algoritma yang digunakan untuk mencari jalur terpendek antara dua titik pada sebuah peta. Algoritma ini dapat diimplementasikan pada robot untuk mengatur jalannya ke titik tujuan yang ditentukan.

INSTALASI
Untuk mengimplementasikan program ini, kita perlu menginstall ROS2 dan beberapa library yang dibutuhkan. Berikut adalah beberapa langkah untuk melakukan instalasi pada sistem operasi Linux:

1. Install ROS2 dengan mengikuti petunjuk instalasi yang tersedia di website resmi ROS2.
2. Install library Pygame dengan menjalankan perintah 'pip3 install pygame' pada terminal.
3. Install library ROS2 Navigation dengan menjalankan perintah 'sudo apt-get install ros-<version>-navigation' pada terminal.

MEMBUAT SIMULASI
Untuk membuat simulasi robot dengan menggunakan ROS2 dan Python, kita bisa menggunakan simulator Webots. Berikut adalah langkah-langkah untuk membuat simulasi:
1. Download dan install Webots dari website resmi.
2. Buka Webots dan buat dunia baru.
3. Tambahkan robot ke dunia yang telah dibuat.
4. Atur sensor dan aktuator pada robot.
5. Atur peta dan tujuan pada dunia.
6. Tambahkan kode program yang telah dibuat ke dalam controller robot.

CODE
```
# Import library dan package yang diperlukan
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import time
import numpy as np
from heapq import heappush, heappop
from collections import deque

# Definisikan kelas "MazeSolver"
class MazeSolver(Node):
```

```
    def __init__(self):
        super().__init__('maze_solver')

        # Inisialisasi variabel
        self.goal_x = 0
        self.goal_y = 0
        self.goal_reached = False
        self.robot_pos_x = 0
        self.robot_pos_y = 0
        self.robot_pos_theta = 0
        self.laser_scan = []
        self.publisher = self.create_publisher(Twist, '/cmd_vel', 10)
        self.goal_publisher = self.create_publisher(Twist, '/goal_reached', 10)
        self.subscription = self.create_subscription(LaserScan, '/scan', self.laser_callback, 10)

    # Definisikan prosedur "laser_callback" untuk membaca data dari sensor laser
    def laser_callback(self, data):
    self.laser_scan = data.ranges

# Definisikan fungsi "euclidean_distance" untuk menghitung jarak antara dua titik
def euclidean_distance(self, x1, y1, x2, y2):
    return np.sqrt((x1 - x2)**2 + (y1 - y2)**2)

# Definisikan fungsi "is_goal_reached" untuk mengecek apakah robot sudah mencapai tujuan
def is_goal_reached(self):
    distance = self.euclidean_distance(self.robot_pos_x, self.robot_pos_y, self.goal_x,
self.goal_y)
    if distance < 0.2:
        self.goal_reached = True

# Definisikan fungsi "astar" untuk mencari jalur terpendek ke tujuan
def astar(self, start_x, start_y, goal_x, goal_y, map_data):
    queue = []
    heappush(queue, (0, start_x, start_y, []))
    visited = set()

    while queue:
        (cost, x, y, path) = heappop(queue)

        if (x, y) in visited:
            continue

        if x == goal_x and y == goal_y:
            return path + [(x, y)]

        visited.add((x, y))

        for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
            next_x, next_y = x + dx, y + dy
```

```
            if next_x < 0 or next_y < 0 or next_x >= len(map_data) or next_y >=
len(map_data[0]) or map_data[next_x][next_y] == 1:
                continue
            heappush(queue, (cost + 1 + self.euclidean_distance(next_x, next_y, goal_x, goal_y),
next_x, next_y, path + [(x, y)]))

    return None

# Definisikan prosedur "move_robot" untuk menggerakkan robot ke tujuan
def move_robot(self):
    path = self.astar(int(self.robot_pos_x), int(self.robot_pos_y), self.goal_x, self.goal_y,
self.map_data)
    if not path:
        return

    target_x, target_y = path[0]

    while not self.goal_reached:
        distance = self.euclidean_distance(self.robot_pos_x, self.robot_pos_y, target_x,
target_y)
        if distance < 0.2:
            path = path[1:]
            if not path:
                break
            target_x, target_y = path[0]

        if target_x < self.robot_pos_x:
            vel_x = -0.2
        elif target_x > self.robot_pos_x:
            vel_x = 0.2
        else:
            vel_x = 0

        if target_y < self.robot_pos_y:
            vel_y = -0.2
        elif target_y > self.robot_pos_y:
            vel_y = 0.2
        else:
            vel_y = 0

        twist = Twist()
        twist.linear.x = vel_x
        twist.linear.y = vel_y
        self.publisher.publish(twist)
        self.goal_publisher.publish(self.goal_reached)

        time.sleep(0.1)

# Definisikan prosedur "map_callback" untuk membaca data peta dari file
def map_callback(self, map_file):
```

```
    with open(map_file) as f:
        self.map_data = np.array([[int(i) for i in line.split
# Definisikan prosedur "run" untuk menjalankan simulasi dan menggerakkan robot ke tujuan
def run(self):
    # Inisialisasi ROS node dan subscriber
    rospy.init_node('simple_navigation_robot')
    rospy.Subscriber('/base_scan', LaserScan, self.laser_callback)
    rospy.Subscriber('/map_file', String, self.map_callback)

    # Inisialisasi ROS publisher
    self.publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
    self.goal_publisher = rospy.Publisher('/goal_reached', Bool, queue_size=10)

    # Tunggu sampai data peta terbaca
    while not self.map_data.any():
        continue

    # Set posisi awal robot
    self.robot_pos_x = 5.0
    self.robot_pos_y = 5.0

    # Set tujuan robot
    self.goal_x = 10.0
    self.goal_y = 10.0

    # Gerakkan robot ke tujuan
    self.move_robot()

    # Berhenti saat goal tercapai
    self.publisher.publish(Twist())
    rospy.spin()
```

STUDI KASUS

Pada kode di atas, robot menggunakan sensor LIDAR untuk menghindari rintangan dan mencapai tujuan yang telah ditentukan. Algoritma A* digunakan untuk mencari jalur terpendek ke tujuan. Peta lingkungan yang digunakan diambil dari file `.txt`.

Dalam menjalankan simulasi, pertama-tama ROS node dan subscriber diinisialisasi. Kemudian, sensor LIDAR dan data peta dibaca oleh callback functions `laser_callback` dan `map_callback`, masing-masing.

Setelah data peta terbaca, posisi awal dan tujuan robot ditentukan. Kemudian, robot akan digerakkan ke tujuan dengan menggunakan fungsi `move_robot`. Fungsi ini akan memanggil algoritma A* untuk mencari jalur terpendek ke tujuan. Setiap langkah dalam jalur diwakili oleh sebuah koordinat.

Untuk menggerakkan robot, dilakukan perbandingan antara koordinat robot dan koordinat langkah pertama dalam jalur. Kecepatan robot kemudian diatur sedemikian rupa sehingga robot akan bergerak menuju langkah tersebut. Setelah robot mencapai langkah tersebut, langkah tersebut akan dihapus dari jalur dan perhitungan dilakukan lagi untuk langkah selanjutnya, hingga robot mencapai tujuan

KESIMPULAN

Dalam proyek ini, sebuah robot penghindar rintangan sederhana telah dibuat menggunakan bahasa pemrograman Python, Webots, dan ROS2. Robot tersebut menggunakan sensor LIDAR untuk menghindari rintangan dan mencapai tujuan yang telah ditentukan. Algoritma A* digunakan untuk mencari jalur terpendek ke tujuan. Peta lingkungan yang digunakan diambil dari file `.txt`.

Dalam menjalankan simulasi, ROS node dan subscriber diinisialisasi. Kemudian, sensor LIDAR dan data peta dibaca oleh callback functions, masing-masing. Setelah data peta terbaca, posisi awal dan tujuan robot ditentukan. Kemudian, robot akan digerkan ke tujuan dengan menggunakan fungsi move_robot. Fungsi ini akan memanggil algoritma A* untuk mencari jalur terpendek ke tujuan. Setiap langkah dalam jalur diwakili oleh sebuah koordinat.

Untuk menggerakkan robot, dilakukan perbandingan antara koordinat robot dan koordinat langkah pertama dalam jalur. Kecepatan robot kemudian diatur sedemikian rupa sehingga robot akan bergerak menuju langkah tersebut. Setelah robot mencapai langkah tersebut, langkah tersebut akan dihapus dari jalur dan perhitungan dilakukan lagi untuk langkah selanjutnya, hingga robot mencapai tujuan.

Dalam proyek ini, Webots digunakan untuk mensimulasikan robot dan lingkungan sekitar. Webots menyediakan model robot dan sensor yang dapat digunakan dalam simulasi. ROS2 digunakan untuk mengintegrasikan berbagai komponen dan membuat robot dapat berinteraksi dengan lingkungan sekitar.

Dalam pembuatan proyek ini, beberapa langkah yang perlu dilakukan adalah:

1. Instalasi Webots dan ROS2.
2. Membuat model robot dan lingkungan di Webots.
3. Menulis kode Python untuk mengendalikan robot menggunakan ROS2.
4. Menjalankan simulasi menggunakan Webots dan ROS2.

Dalam proyek ini, sebuah contoh studi kasus telah dibuat untuk menggambarkan bagaimana robot penghindar rintangan bekerja. Algoritma A* digunakan untuk mencari jalur terpendek ke tujuan dan sensor LIDAR digunakan untuk menghindari rintangan. Peta lingkungan diambil dari file .txt.

Kesimpulannya, proyek ini menunjukkan bagaimana penggunaan Python, Webots, dan ROS2 dapat digunakan untuk membuat robot penghindar rintangan sederhana yang dapat berinteraksi

dengan lingkungan sekitar. Dalam proyek ini, algoritma A* dan sensor LIDAR digunakan untuk mengoptimalkan gerakan robot. Dalam studi kasus yang dibuat, robot penghindar rintangan berhasil mencapai tujuan dengan menghindari rintangan yang ada di sekitarnya..