Project 4c Report Group 38 : Robotics and Spatial Intelligence

Part 1(A) Description of the Overall Approach:

System Overview

The system is designed for a differential drive robot to navigate autonomously in an environment while avoiding obstacles. It primarily consists of two ROS nodes:

Differential Drive Simulator Node (same as project 4b): This node stimulates the movement of a differential drive robot in a virtual environment. It uses parameters from a loaded robot model and world file to simulate the robot's physical characteristics and the environment it navigates. The node calculates the robot's pose (position and orientation) based on velocity commands, and checks for collisions using an occupancy grid representing the environment. Additionally, it simulates laser scan data to mimic the robot's perception of its surroundings.

Navigation Controller Node: This node controls the robot's movements in response to its environment. It subscribes to the laser scan data provided by the simulator, **dividing this data into different regions around the robot (front, front-left, front-right, left, right)**. Based on the proximity of obstacles in these regions, the controller decides the robot's motion. **It uses a reactive control strategy, adjusting the robot's linear and angular velocities to navigate while avoiding collisions.** The controller maintains a safe distance from obstacles and is capable of handling narrow corridors by adjusting the robot's speed.

Operational Flow

Sensing and Perception: The Differential Drive Simulator generates laser scan data, simulating how the robot perceives obstacles around it.

Decision Making: The Navigation Controller processes this data, determining the distance to obstacles in different directions. It then decides the robot's next movement to avoid obstacles while aiming to move forward.

Motion Control: Based on the controller's decision, velocity commands (linear and angular) are published. These commands dictate how the robot moves in the simulated environment.

Feedback Loop: The simulator updates the robot's pose based on the received commands and checks for collisions, feeding back the updated position and simulated laser data to the controller, thus closing the feedback loop.

Project 4c Report Group 38 : Robotics and Spatial Intelligence

Key Features

Reactive Control: The system reacts in real-time to its environment, making it suitable for dynamic and unpredictable settings.
Safety and Adaptability: It maintains a safe distance from obstacles and can adapt its behavior for different scenarios like narrow corridors.

Part 1(B) The choices made in designing the program and the rationale behind those decisions:

When designing the navigation program for the differential drive robot, several key choices were made to ensure effective and safe navigation. Here's an overview of these choices and the rationale behind them:

Reactive Control Strategy:

Choice: Implementing a reactive control system that responds in real-time to sensor data.
Why: This approach is well-suited for dynamic environments where obstacles can appear unpredictably. It allows the robot to make immediate decisions based on current sensor readings, enhancing its ability to avoid collisions.

```
## Implementing the reactive controller

if regions['front'] > 1 and regions['fleft'] > 1 and regions['fright'] > 1:
    state = 'case 1 - nothing'
    linear_x = scaling_factor/regions['front']
    angular_z = 0.0
elif regions['front'] < 1 and regions['fleft'] > 1 and regions['fright'] > 1:
    state = 'case 2 - front'
    linear_x = 0.0
    angular_z = scaling_factor/regions['front']
    # angular_z = 0.5

elif regions['front'] > 1 and regions['fleft'] > 1 and regions['fright'] < 1:
    state = 'case 3 - front_right'
    linear_x = 0.0
    angular_z = scaling_factor/regions['fright']
    # angular_z = 0.5

elif regions['front'] > 1 and regions['fleft'] < 1 and regions['fright'] > 1:
    state = 'case 4 - fornt_left'
    linear_x = 0.0
    angular_z = -scaling_factor/regions['fleft']
    # angular_z = -0.5

elif regions['front'] < 1 and regions['fleft'] > 1 and regions['fright'] < 1:
    state = 'case 5 - front and fornt_right'
    linear_x = 0.0
    angular_z = scaling_factor/regions['fright']
    # angular_z = 0.5

elif regions['front'] < 1 and regions['fleft'] < 1 and regions['fright'] > 1:
    state = 'case 6 - front and front_left'
    linear_x = 0.1  # Small forward motion
    angular_z = -angular_control_factor * side_diff
    # angular_z = -0.5

elif regions['front'] < 1 and regions['fleft'] < 1 and regions['fright'] < 1:
    state = 'case 7 - front and front_left and front_right'
    # Small forward motion, adjust angular_z based on side_diff
    linear_x = 0.1
    angular_z = angular_control_factor * side_diff

elif regions['front'] > 1 and regions['fleft'] < 1 and regions['fright'] < 1:
    state = 'case 8 - front_left and front_right'
    # Adjust linear_x and angular_z based on side_diff
    linear_x = scaling_factor/regions['front']
    angular_z = 0.0
else:
    state = 'unknown case'
```

Project 4c Report Group 38 : Robotics and Spatial Intelligence

Division of Laser Scan Data into Regions:

Choice: Dividing the laser scan data into distinct regions (front, front-left, front-right, left, right).
Why: This segmentation simplifies the decision-making process. By evaluating the proximity of obstacles in these key directions, the robot can more effectively determine how to adjust its course.

```
#  Dividing the laser scan data into distinct regions (front, front-left, front-right, left, right).
regions = {
    'right':  min(min(msg.ranges[0:region_size]), 40.0),  # Closest obstacle in the right slice
    'fright': min(min(msg.ranges[region_size:2*region_size]), 40.0),  # Closest obstacle in the front-right slice
    'front':  min(min(msg.ranges[2*region_size:3*region_size]), 40.0),  # Closest obstacle in the front slice
    'fleft':  min(min(msg.ranges[3*region_size:4*region_size]), 40.0),  # Closest obstacle in the front-left slice
    'left':   min(min(msg.ranges[4*region_size:]), 40.0),  # Closest obstacle in the left slice
}
```

Proportional Control for Angular Velocity:

Choice: Using proportional control for determining the robot's angular velocity.
Why: Proportional control provides a smooth and responsive adjustment of the robot's turning rate, which is crucial for agile maneuvering in environments with variable obstacle density.

```
# Proportional control factor for angular velocity
angular_control_factor = 0.2  # Adjusted based on robot's responsiveness
```

```
    # Scaling factor for proportional control
    scaling_factor = 2.0  # adjusted this based on testing
```

Safety Thresholds and Parameters:

Choice: Setting safety thresholds such as minimum safe distance from obstacles and speed limits.
Why: These thresholds ensure the robot maintains a safe distance from obstacles, reducing the risk of collisions, and operates within safe operational speeds.

```
    # Minimum safe distance from any obstacle
    safe_distance = 0.16 # Adjusted based on  requirements
```

Handling Narrow Corridors:

Choice: Special logic to handle navigation in narrow corridors.
Why: Narrow corridors pose a unique challenge due to limited maneuvering space. Adjusting the robot's behavior in these scenarios helps prevent getting stuck or colliding with walls.

```
    narrow_corridor_threshold = 1.0  # Threshold to detect narrow corridors

    # Check if the robot is in a narrow corridor
    in_narrow_corridor = all(region < narrow_corridor_threshold for region in regions.values())
```

```
    ## Ensure the robot is not too close to any side
    if regions['left'] < safe_distance or regions['right'] < safe_distance:
        angular_z = angular_control_factor * side_diff
```

Flexible and Scalable Design:

Choice: Making the system parameters (like safe distance, speed thresholds) adjustable.
Why: This flexibility allows the navigation system to be adapted for different robots or environments, enhancing the system's scalability and versatility.

```
# Calculates the difference between left and right distances
side_diff = regions['left'] - regions['right']

# Proportional control factor for angular velocity
angular_control_factor = 0.2  # Adjusted based on robot's responsiveness

# Minimum safe distance from any obstacle
safe_distance = 0.16 # Adjusted based on  requirements
state = ''
# Scaling factor for proportional control
scaling_factor = 2.0  # adjusted this based on testing

min_speed_threshold = 1.0 # Minimum speed threshold
narrow_corridor_threshold = 1.0  # Threshold to detect narrow corridors

# Check if the robot is in a narrow corridor
in_narrow_corridor = all(region < narrow_corridor_threshold for region in regions.values())
```

Collision Checking:

Choice: Implementing collision detection in the simulator.
Why: This ensures that the simulated robot's interactions with the environment are realistic, providing valuable feedback for refining the navigation strategy.

These choices were guided by the goals of ensuring safety, adaptability, and effective navigation in a variety of environments.

Project 4c Report Group 38 : Robotics and Spatial Intelligence

Part 1(B): Did the results meet our expectations?

Yes, the results met expectations, primarily due to the well-structured design and implementation of the reactive navigation controller. The reactive control strategy, which is pivotal in dynamic environments, functioned effectively by making real-time decisions based on sensor data, thus adeptly managing unforeseen obstacles. The segmentation of laser scan data into distinct regions allowed for precise obstacle detection and avoidance, ensuring the robot could navigate through various scenarios without collisions.

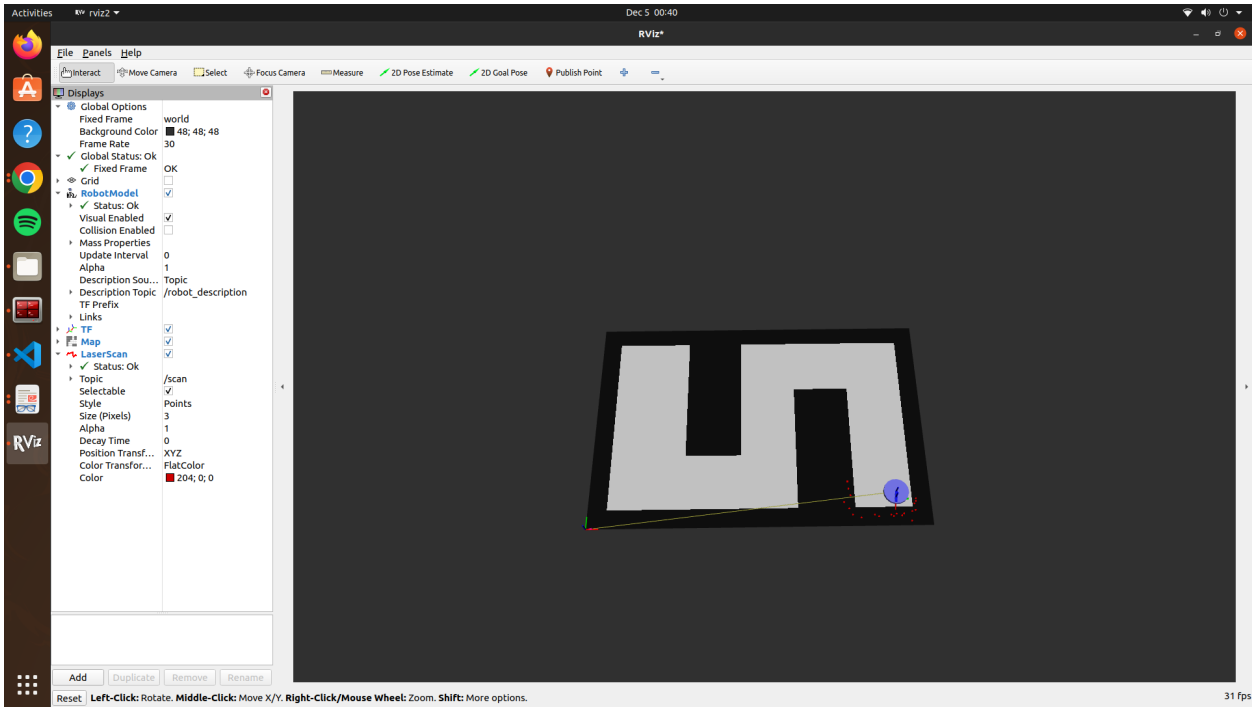
The implementation of proportional control for angular velocity provided the necessary balance between maneuverability and stability, crucial for smooth navigation. Additionally, the safety measures, such as maintaining a minimum safe distance from obstacles and setting speed limits, significantly contributed to avoiding potential collisions. The successful handling of narrow corridors, often challenging in robotic navigation, further underscores the controller's efficacy.

Overall, the system's adaptability, coupled with rigorous testing, ensured it performed up to expectations, providing collision-free paths across all test cases.

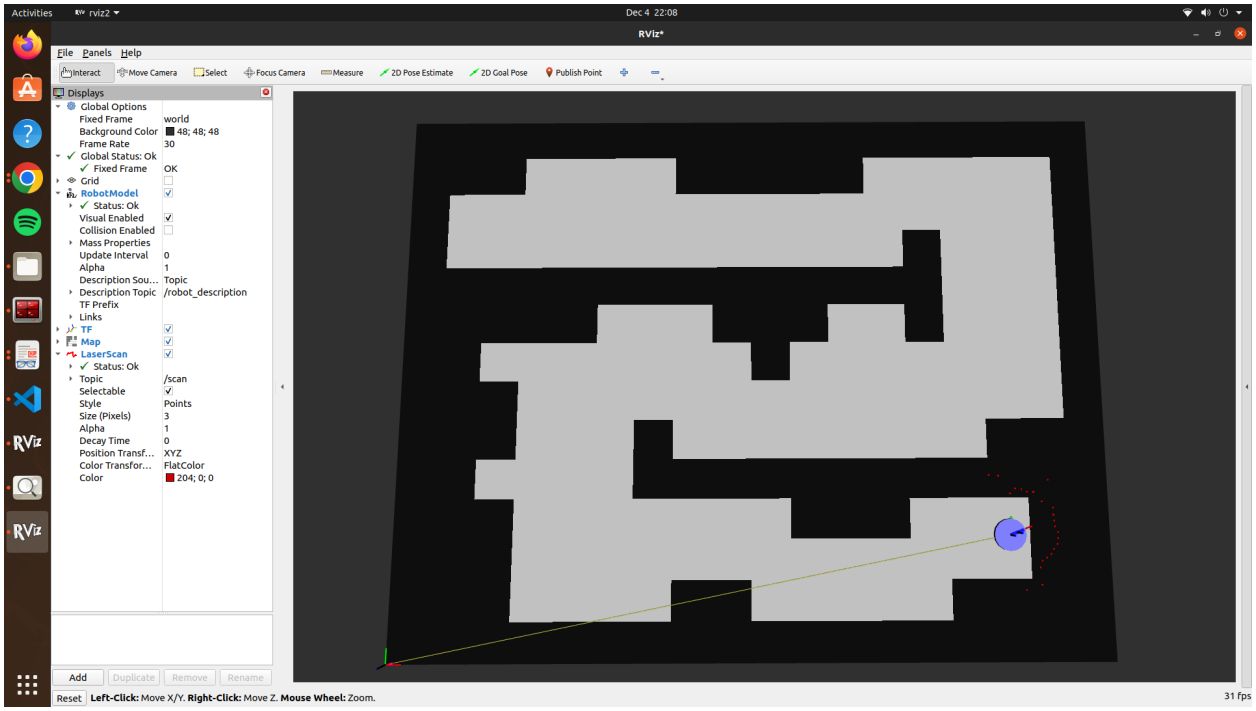
Project 4c Report Group 38 : Robotics and Spatial Intelligence

Screenshots showing the robot near the furthest travel down the corridor that it achieves

Test case 1 robot name: normal.robot world name: windy.world

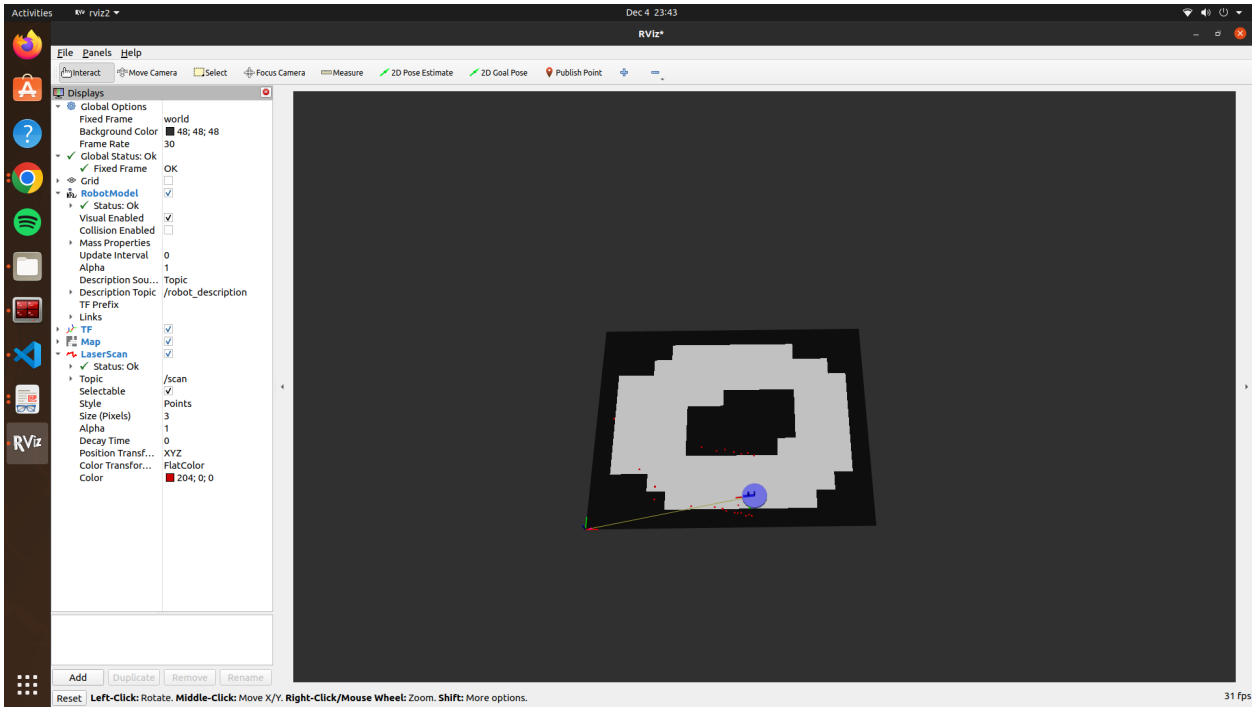


Test case 2 robot name: normal.robot world name: cave.world



Project 4c Report Group 38 : Robotics and Spatial Intelligence

Test case 3 robot name: normal.robot world name: donut.world



Test case 4 robot name: normal.robot world name: snake.world