Final Project Write-Up: Mobile Robotics - ECE 5340

Author: Nate Herbst - A02307138

Course: ECE 5340 - Introduction to Planning and Control for Mobile Robots

Outline

This write-up is organized into the following sections:

Introduction

Provides an overview of the project's objectives and scope, including the focus on simulating a differential drive robot in RVIZ and Gazebo and implementing a path-planning algorithm.

Methodology

Explains the step-by-step process undertaken to achieve the project's goals. This includes creating the URDF model, configuring RVIZ and Gazebo, implementing ROS nodes for publishing and subscribing, and linking the path-planning algorithm to the robot's control logic.

Results

Details the outcomes of the project, highlighting successful implementation aspects such as visualizing the robot in RVIZ and Gazebo and noting areas where functionality was incomplete, such as path-following in the simulation.

Challenges

Describes the technical and logistical difficulties encountered during the project. These include the extensive effort required to configure simulation environments and the struggles with integrating pose publishing and subscribing in a CMake-based configuration.

Basic Simulation Abilities (50 Points)

This section highlights the foundational capabilities achieved in the project.

Control Capabilities (25 Points)

Focuses on the robot's ability to follow a planned trajectory

Going Beyond the Classroom (60 Points)

This section explores advanced features implemented beyond basic requirements:

Conclusion

Summarizes the project's achievements, limitations, and potential future directions to enhance functionality and address unresolved issues.

Introduction

This project aims to simulate a differential drive robot capable of trajectory following and interactive destination selection within ROS2 Jazzy. The project integrates Python-based simulations, visualization tools like RVIZ and Gazebo, and ROS2 utilities to demonstrate planning and control techniques. Each milestone extends fundamental concepts of mobile robotics, ensuring compliance with static analysis tools, robust visualization, and dynamic adaptability.

Methodology

The implementation of this project involved integrating several components to achieve the goals of visualizing a differential drive robot and simulating its movement using path planning algorithms. The process began with designing the robot's URDF model to define its physical attributes and kinematics. This model was then configured to work within RVIZ and Gazebo, ensuring compatibility with the tools required for simulation.

To bring the simulation to life, I created Python-based ROS nodes for publishing and subscribing to topics, including pose and control commands. A Voronoi path planning algorithm was implemented to generate trajectories, and the robot's control logic was designed to follow these trajectories. Launch files were created to streamline running the different components simultaneously, reducing manual setup complexity. The entire setup was tested using ROS 2 Jazzy, with dependencies managed via a virtual environment and a `final_project.rosinstall` file.

Results

The project successfully achieved partial functionality. The robot's 3D model was visually represented in both RVIZ and Gazebo, providing a clear view of its structure and layout. The model was able to spawn correctly in both environments, demonstrating accurate implementation of the URDF and simulation setup. However, while the robot appeared in RVIZ and Gazebo, it was unable to follow the path generated by the path planner. This limitation highlighted areas for improvement in linking the control and planning components with the robot's simulation environment.

Challenges

This project posed several significant challenges. A substantial amount of time—approximately 18 hours—was spent configuring RVIZ and Gazebo to correctly implement the robot and integrate it with the path planner. These difficulties stemmed from both the complexity of the tools and the nuances of aligning the simulation with the control logic, as well as different command for different versions of RVIZ and Gazebo.

Another major challenge was publishing and subscribing to the robot's position within the URDF implementation. Initially, I struggled with the format, as I was using a CMake-based setup instead of a Python-based configuration. This unfamiliarity made it difficult to ensure that the robot model could subscribe to pose topics and dynamically adjust its position in the simulation.

Despite these challenges, the progress made highlights the potential for further development, particularly by addressing the integration of the robot model with pose topics and refining the path-following

capabilities in future iterations.

1. **Basic Simulation Abilities (50 Points)**

This section focuses on the foundational capabilities of the robot simulation, ensuring that the setup adheres to coding standards, provides a cohesive workspace, and delivers interactive functionality.

Static Analysis Compliance: Mypy and Pylint (15 Points)

The Python code that I wrote adheres to robust coding standards and has been verified using **mypy** and **pylint** static analysis tools.
The project follows Python best practices by passing `mypy` and `pylint` checks.

- `mypy` ensures type safety, preventing runtime errors due to type mismatches.
- `pylint` enforces code quality and maintainability through style checks.

To run, these checks, first navigate your terminal to the following directory:

```
cd FinalProject/src/py_sim/py_sim
```

- **`mypy` Execution**: Now, run the following command to initiate the `mypy` check:

  ```
  python3 -m mypy py_sim --explicit-package-bases
  ```

- **`pylint` Execution**: Run the following command to initiate the `pylint` check:

  ```
  python3 -m pylint --jobs 0 --rcfile .pylintrc py_sim/
  ```

This adherence ensures the code is maintainable, reliable, and free of common errors, setting a foundation for extensibility.

Workspace Configuration with .rosinstall (10 Points)

A `.rosinstall` file has been provided to streamline workspace creation and setup for the project.

- **Purpose**: The `final_project.rosinstall` file enables users to quickly recreate the exact workspace configuration used during development. By running the following command, all required repositories and dependencies are cloned and configured in a consistent manner.
- **Usage**:
  1. Make sure the `final_project.rosinstall` file in the desired workspace directory of the repository.
  2. Run the following command:

```
vcs import --input finalProject.rosinstall src
```

3. Complete the setup by running:

```
build
```

This approach ensures reproducibility across systems, minimizing environment setup errors.

Unified Simulation Launch File (10 points)

Each major functionality is encapsulated in a single launch file, streamlining the process of running the simulation. These launch files consolidate all necessary components, including Gazebo, the robot description, and path planning modules.

- **Execution**: See the `README.md` for this repository on executing the launch files.

Launch files are extremely convent as they allow the execution of multiple nodes simultaneously with just a single `launch` command.

Interactive Goal Selection via GUI

The `3D_bot_path_follow_pysim.launch.py` file integrates with a ROS node and RVIZ that interprets mouse clicks as pose goals for the robot.

- **Steps to Use**:
    1. Launch the simulation using the `3D_bot_path_follow_pysim.launch.py` launch file (see the `README.md` for the command).
    2. Navigate to the RVIZ GUI window that opened.
    3. Use the "2D Goal Pose" tool (in RVIZ) at the top to click on a point in the map or environment.
    4. The robot will immediately recalculate its path using a `Voronoi` plan type algorithm and begin navigating to the selected destination.

This feature significantly enhances the user experience by allowing real-time adjustments without requiring manual parameter updates or re-launching processes.

Adaptable Look-Ahead Horizon (5 points)

The planning architecture incorporates an adaptable look-ahead horizon, enabling dynamic adjustments to the robot's planning behavior based on its current state and environment.

- **Usage in Planning Architecture**: The look-ahead horizon determines the extent of the search space considered during path planning. A shorter horizon focuses on local obstacles, ensuring fine-grained navigation in cluttered environments, while a longer horizon prioritizes global path optimization.
- **Dynamic Adjustment**: The look-ahead horizon is recalculated during motion, factoring in parameters such as the robot’s speed, proximity to obstacles, and goal distance. This adaptability ensures the

robot maintains a balance between reactive and deliberate path planning, optimizing both safety and efficiency.

This feature enables the robot to navigate complex environments effectively, accommodating dynamic changes while maintaining robust planning performance.

---

2. **Control Capabilities (25 Points)**

This section demonstrates the robot's ability to follow complex paths using a trajectory-following control law.

Trajectory Following Control Law (25 Points)

The implemented control law ensures the robot accurately tracks paths generated by planners like Voronoi.

- **Implementation**: The `09_planning_service` node integrates with path-planning algorithms using a `Voronoi` plan type and publishes a sequence of poses using the node `09_pysim_pose_pub` to the `/pose` topic for execution.

---

3. **Going Beyond the Classroom (60 Points)**

These features showcase the visualization and simulation capabilities in RVIZ and Gazebo.

3D Visualization in RVIZ (30 Points)

A CAD model of the differential drive robot is visualized in RVIZ. Users can interactively set goals for the robot.

- **Launch File**: `3D_RViz_Bot_FP.launch.py`
- **Implementation**: A `robot.urdf.xacro` file describes the robot's structure and properties. The `robot_state_publisher` node broadcasts the robot's state for visualization in RVIZ.

Simulation in Gazebo (30 Points)

The robot spawns into a Gazebo simulation environment where it moves without getting stuck.

- **Launch File**: `3D_Gazebo_Bot_FP.launch.py`
- **Implementation**: The `gazebo_control.xacro` file defines the robot's dynamic properties and interacts with Gazebo physics plugins to simulate realistic movements.

Conclusion

The Mobile Robotics Final Project provided an opportunity to apply theoretical knowledge and practical skills to simulate and control a differential drive robot in both RVIZ and Gazebo environments. The project demonstrated essential capabilities, including static analysis compliance, workspace reproducibility, interactive simulation control, and trajectory-following control laws. Furthermore, advanced visualization and simulation in RVIZ and Gazebo showcased the ability to go beyond classroom requirements.

Despite notable successes, such as visualizing the robot in two different simulation environments and integrating ROS components, some challenges persisted. The robot could not fully execute path-following functionality in Gazebo due to difficulties in integrating pose subscriptions with the URDF model using a CMake-based configuration. These challenges highlight areas for further learning and refinement, particularly in harmonizing ROS components and formats.

This project underscores the complexities of robot simulation and control while showcasing the potential of ROS for robotics development. The outcomes pave the way for future enhancements, such as optimizing the interaction between simulation environments and robot control logic, ensuring the robot achieves full path-planning capabilities in simulation. Through these experiences, this project has strengthened skills in robotics software engineering, simulation configuration, and problem-solving, providing a solid foundation for continued exploration in mobile robotics.