Computational Robotics Warmup Project

Cole Marco, Luke Witten

September 19, 2023

1 Introduction

The intention of this project was to provide a practical introduction to learning how to interact with Neatos (a Roomba like robot), using ROS2, the Robot Operating System. The goal was to write scripts in Python using rclpy that will allow the Neato to accomplish a certain set of behaviors. The five primary behaviors we developed were teleop control, driving in a square, wall following, obstacle avoidance, and person following. Finite state control was also implemented using a combination of the aforementioned behaviors. We focused on developing our code in a modular format, so that we could better understand what parts of ROS (and computational robotics in general) might be repeatedly useful.

2 Teleop

Teleoperation (teleop) control in the context of a robot using ROS (Robot Operating System) refers to the ability to remotely control and maneuver the robot using a human-operated interface, such as a joystick or a computer keyboard. This allows an operator to guide the robot's movements and actions from a distance, making it a valuable feature for tasks that require real-time human supervision or when exploring unstructured environments where autonomous navigation may be challenging. Our implementation enables manual control through keyboard input. The code is based on this repository which we used to understand how teleop works and to scaffold our implementation.

2.1 Keybindings and Control

The teleoperation script interprets keyboard input to control the robot's movement. It does so in a non-blocking format by listening for key presses in an infinite loop until the user presses an exit key (in our case ctrl+c). Users can employ a variety of keybindings to dictate linear and angular velocity, resulting in different motion behaviors. Keybindings are divided into two primary categories: movement control and speed adjustments.

2.1.1 Movement Control

The following keys are used for controlling the robot's movement:

- `u`, `i`, `o`, `j`, `k`, `l`, `m`, `,`, `.`: These keys enable movement in various directions, allowing the robot to move forward, backward, turn left, turn right, and more.
- `O`, `I`, `J`, `L`, `U`, `<`, `>`, `M`: These keys provide additional movement options, including diagonal movement and strafing.
- `t` and `b`: These keys control the robot's vertical movement, allowing it to move up and down along the Z-axis.
- Anything else: Pressing any other key results in the robot coming to a complete stop.

2.1.2 Speed Adjustments

Users have the flexibility to adjust the robot's speed on-the-fly using the following keys:

- `q` and `z`: These keys increase or decrease both linear and angular speed by 10
- `w` and `x`: These keys specifically adjust the linear speed while keeping angular speed constant.
- `e` and `c`: These keys specifically adjust the angular speed while keeping linear speed constant.

2.2 User Interaction

The code provides real-time feedback to users, displaying the current telemetry with speed and turn values. It ensures that users have a clear understanding of the robot's behavior during teleoperation.

2.3 Restults and Performance

The teleoperation control code serves as an essential building block in our robot's software stack, facilitating manual control and enabling users to navigate the robot with ease. The intuitive keybindings and real-time telemetry feedback enhance the user experience, making it a valuable tool for various applications.

3 Driving in a Square

This exercise had us place the Neato on the ground and have it draw a perfect square with its movement. Because we hadn't learned how to localize our robot in space yet, the challenge was intended to teach us how to use the Neato's

odometry. To track the neato's odometry, we created a subscription to the 'odom' topic within our square_driver ROS2 node, which tracks the position and orientation of the neato along with its covariances, quantifying the belief of the robot in its current stored state.

3.1 Quaternions

The orientation of the Neato was stored as a quaternion, which encodes 3 dimensional orientation and rotation as an implicit result of transforming a 4 dimensional hypersphere.

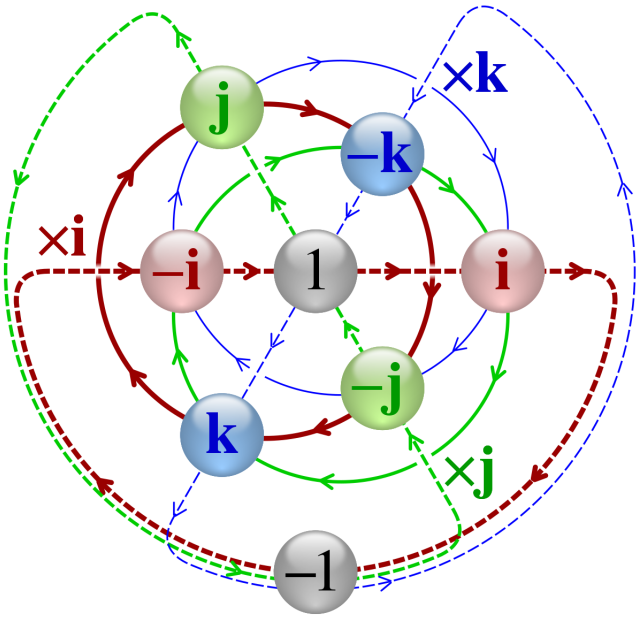

Figure 1: Map of Quaternion Multiplication

Using quaternions imposed a mathematical learning curve for us as neither of us had ever used them before, but after some research, we were able to implement a way to consistently track the orientation of our robot in space.

3.2 Results

Using quaternions were we able to confidently proportionally control to any arbitrary size turn every time, completing the module. Controlling the length driven by the Neato was also accomplished using proportional control, though the implementation path was much simpler. Not only can we drive the neato in a square, but we can confidently traverse any known map given a set of distances and turns because of our odometry tracking ability.

4 Person Following

The person following behavior has the Neato use its sensors to find a person, and then attempts to follow that person while maintaining a minimum distance. We implemented person following behavior primarily with the laser rangefinder. After subscribing to the 'scan' topic, we were able to disregard data from points which were either too far away to be considered or invalid scans. Averaging the positions of nearby, valid scans, we were able to track and follow nearby entities. Proportional control was used to stop before bumping into a person as well as to ensure that the Neato finished pointing towards the object it is actively tracking.
Odometry data was used once again but only for the purposes of visualization and debugging in Rviz.

5 Wall Following

The wall following behavior makes the Neato find a wall, match its course until it's parallel with the wall, and then follow alongside it. The naive approach to this is to grab a laser scan 45 degrees clockwise of the robot, get another scan from an additional 90 degrees clockwise, and then compare those two distances.

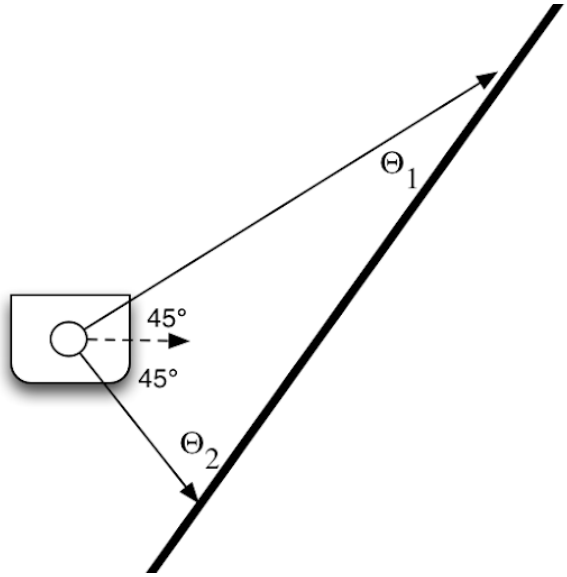

Figure 2: Diagram of Naive Wall Following Approach

If Θ_1 is greater, the robot needs to turn clockwise, otherwise it needs to turn counter-clockwise. This approach does work quite well, but we decided we wanted to explore more advanced options that might yield more robust results

in complicated systems. One of these ideas was to use the RANSAC algorithm to find a wall for the robot to match.

5.1 Random Sample Consensus (RANSAC) Algorithm for Wall Detection

The Random Sample Consensus (RANSAC) algorithm played a pivotal role in accurately identifying walls in the robot's environment. RANSAC is a widely recognized algorithm utilized for model fitting in the presence of outliers. In our context, it efficiently discerned linear segments representing walls within the sensor data.

1. **Random Sample Selection**: Initially, a small set of random data points was chosen from the sensor data, hypothesizing a potential wall.
2. **Model Fitting**: Using the selected points, a linear model (representing a wall) was fitted.
3. **Inlier Detection**: All data points consistent with the model (within a specified tolerance) were identified as inliers.
4. **Model Evaluation**: The quality of the model was assessed based on the number of inliers it had.
5. **Iteration and Convergence**: Steps 1 to 4 were repeated iteratively to find the best model (wall) that explained the maximum number of data points.

This robust approach to wall detection provided the foundation for our subsequent wall-following control strategy.

5.2 Proportional Control for Wall Following

With the walls detected using the RANSAC algorithm, we employed a proportional control strategy to enable the robot to follow a detected wall while maintaining a constant distance from it. Proportional control, a fundamental technique in control theory, adjusts the robot's turning behavior in proportion to the error between the desired distance from the wall and the actual distance measured by the sensors.

The control loop functioned as follows:

1. **Error Calculation**: The error, representing the deviation from the desired distance to the wall, was computed based on sensor readings.
2. **Proportional Gain Application**: The error was multiplied by a proportional gain, determining the magnitude of the control signal.
3. **Control Signal Generation**: The resulting product was used to generate a control signal, dictating the robot's turning angle and velocity.

4. **Adjustment for Wall Alignment**: The control signal was applied to ensure that the robot maintained a constant distance from the wall while moving parallel to it.

5.3 Results and Performance

The integration of the RANSAC algorithm for wall detection with the proportional control strategy yielded a bit of a finicky system due to the laser scans picking up obstacles that aren't walls, like chairs or people. The proportional control was also hard to tune correctly, and led to the Neato jolting around while trying to align to the wall. Our approach did still work however, and gave us insight into considerations to make when implementing a more complicated approach.

6 Obstacle Avoidance

The goal of this challenge is to enable a robot to seek out some goal, whether that is moving towards a specific point in space or generally heading in a direction, while avoiding arbitrary obstacles it encounters. A main challenge of implementing this behavior is deciding whether obstacle avoidance should be a continuous, fluid process or if the behavior requires discretely changing robot operations. We decided to continuously alter the path of the robot by avoiding obstacles with potential fields.

6.1 Potential Fields

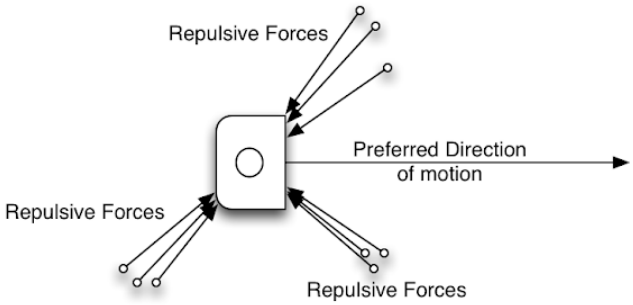

Figure 3: Repellent Forces Diagram

If one has some idea where obstacles in the environment are, then one can steer themselves away from obstacles by imposing a repulsive "force" from each individual obstacle and an attractive force pulling the Neato in the preferred direction. This direction can be a specific point, an orientation with respect

to the global frame, or just forward. By summing the attractive and repellent forces of the obstacles and goals, one can find an overall direction to head in, avoiding obstacles while still moving towards the goal. The repellent forces become more powerful the closer one gets to an obstacle to prioritize avoiding nearby hazards. Some tuning is required for this approach.

6.2 Implementation and Results

By modeling each valid scan made by the laser scanner as a weak priority obstacle, we were able to implement an obstacle avoidance algorithm using potential fields. Because we modeled obstacles on the contours of larger obstacles instead of the centroid of the large obstacle itself, we were able to move along the contours of obstacles instead of just avoiding a large region of space. Removing the necessity to aggregate scans and then classify them into obstructions also aided in computational efficiency. Overall this implementation successfully avoided obstacles while moving towards its goal.

7 Finite State Machine

The final challenge was to implement a Finite State Machine (FSM) that enables dynamic switching between two primary behaviors: teleoperation control and person following. This exercise was intended to teach us how to combine multiple small behaviors together, in the hope that we can abstract this in the future to control complex and capable robot systems. This FSM is designed to respond to specific triggers and conditions, allowing the robot to seamlessly transition between these behaviors.

7.1 Teleoperation Control

The teleoperation mode is the default behavior when the robot is initialized. It enables manual control of the robot's movement through keyboard input. Keybindings are defined for various movement directions and speed adjustments. The primary triggers of the teleoperation mode include:

- Escape Key Trigger: The FSM monitors for the Escape key (ASCII code 27), and when pressed, initiates the transition to the person following mode.
- Exit: Users can exit the teleoperation mode by pressing the 'p' key, which terminates the teleoperation loop.

7.2 Person Following Mode

The person following mode is an advanced behavior that relies on sensor data to track and follow a person within its environment. This mode is initiated in response to the Escape key trigger during teleoperation. Key features of the person following mode include:

- Distance Threshold: To ensure the robot doesn't lose track of the person, a distance threshold is set. If the person moves more than 1 meter away, the FSM triggers a switch back to teleoperation mode.
- Visualization: A visualization marker is used to display the robot's estimated position relative to the person's position, aiding in monitoring and debugging.

7.3 State Transition and Control

The FSM efficiently manages the transitions between teleoperation and person following modes, ensuring smooth and responsive behavior changes. The robot's control commands are published as Twist messages to control its movement.

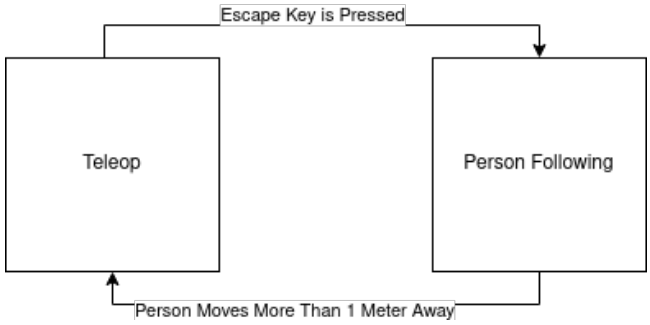

Figure 4: State Machine Diagram

7.4 Results and Performance

This FSM-based approach to robot control allows for versatile and responsive behavior switching, enhancing the robot's capabilities in diverse scenarios. The combination of teleoperation for manual control and person following for autonomous tracking offers flexibility and adaptability in real-world environments.

Further details on the code implementation, algorithms, and control strategies can be found in the source code and accompanying documentation.

8 Discussion and Conclusion

Having some robotics experience, but never having used ROS before we consider this project a huge success. Setting up ROS infrastructure, visualisation, and debugging tools was invaluable to quickly iterating, implementing, and testing these behaviors as we gained familiarity with ROS. We found the Rviz visualisation platform as well as publishing custom markers to be extremely valuable as debugged our programs.

Overall we found the square driver program most difficult to implement, not because it was the hardest behavior, but because it was the first. Grappling with these new tools as well as new mathematical concepts such as quaternion multiplication proved a challenging but rewarding experience. Implementing RANSAC on the wall follower was also a particularly proud moment as we proved we could cause a Neato to somewhat understand the environment around it using messy and inconsistent data.

There are some behaviors that we would have liked to explore more thoroughly such as adding even more behaviors to the finite state machine, performing RANSAC on obstacles as well as walls to create exportable maps of the traversed environments, and following curved walls in non rectangular spaces. All of these behaviors now seem well within reach as our challenge has changed from understanding the landscape of ROS to creatively solving computational problems in robotics.