Computational Robotics Warmup Project

By Nate Sampo and MJ McMillen 09/21

Introduction

This project is part of Olin College's Fall 2018 Computational Robotics Class. In it we use modified Neato robotic vacuum cleaners to demonstrate six basic robotic behaviors. The behaviors demonstrated include teleop control, autonomous path tracing, wall following, obstacle avoidance, person following, and intelligent teleop control. All code is written in Python and ROS (robot operating system) is used to help control the robot.

Creating these behaviors polished up our previously acquired skills, acquainted us with object oriented programming, and introduced us to ROS.

The Neatos

Figure 1: Bob ROS, the Modified Neato

Modified Neato Vacuum cleaners are used in this project. Their dustbins have been removed and replaced with a Raspberry Pi and a portable battery.

Neato Sensing Capabilities

Neatos are equipped with an array of useful sensors but this project focuses on only two of them. These sensors are crucial to enacting our behaviors. The position data from the wheel encoders is used in all behaviors. It helps determine where our robot is and where it is pointed. The 360° lidar scanner scans the surrounding area returning the distance to the nearest point and the degree at which it was found. This data is used in the Wall Following, Obstacle Avoidance, Person Tracking, and Intelligent Teleop behaviors.

Software

The scripts used to run the Neato are written in Python 2.7.2. ROS (Robot Operating System) is used to interface with the Neato and to debug the code. Several ROS nodes and python scripts were written to achieve the desired behaviors.

Structure

Regardless of the desired behavior, all of our code followed the same basic structure. We create an object for each behavior that contains a ROS node, a velocity publisher, and the desired behavior. Multithreading allows us to toggle between different control schemes and interrupt rouge robot behavior. We also use several helper functions that aid in basic calculations.

Behavior Classes

Each behavior is contained within its own class. All behaviors start with an _init_ function that defines the ros node, subscribes to the desired sensor values, and creates the velocity publisher. All behaviors also contain a run function that calculates the needed x and rotational velocities then publishes them to the robot. The classes also contain behavior-specific helper functions. These will be discussed in greater detail later in this report.

Helper Functions

Due to the similarity of several behaviors, universal helper functions are convenient. Functions performing basic trig calculations, angle conversions, and angle subtractions are used in several different classes.

Behavior 1: Teleop Control

The first Behavior we tackled was Teleop control. Teleop control allows us to manually adjust the rotational and linear speed using the keys on our keyboards. This is a very useful and fun feature.

Basic Teleop Control

In basic Telop Control, the I,J,K, and L keys determine the direction of the robot. When the I key is pressed, the robot moves forward. When the K key is pressed, it reverses. When the L key is pressed it turns clockwise. When the J key is pressed it turns counterclockwise. The robot keeps enacting the behavior associated with the last pressed key until a new key is pressed. Because of this, we also added a stop command mapped to the s key.

The Program

The program is very basic. It is a series of if statements that sense the key strokes and translate them into linear and angular velocities. It is very simple and underwhelming.

Results

The program works like a charm. After a key is pressed, the robot shoots forward at top speed performing the action determined by the keystroke.

Advanced Teleop Control

Since the teleop control is nice but underwhelming, we decided to expand upon the basic keyboard control to a mouse based teleop control. When this teleop control is active, the screen space is mapped onto the floor and the robot turns towards wherever the mouse is positioned then travels to that point in space.

The Program

When this program begins, it sets the robot's origin to the current mouse position. When the mouse moves, it calculates the angle to the desired target, turns to that angle, and drives to the new target. The robot periodically checks its trajectory while driving straight towards the target slightly turning back towards if it has drifted from its goal.

The robot slows down as it approaches both its angular target and it linear translation target. We used proportional control to slow the robot as it approaches the target position. This initially made the robot very slow but with a bit of tuning, we have sped it up. It now smoothly navigates to the mouse position.

We also added a toggle mapped to the w key that switches us in and out of advanced teleop mode. This was done because we initially struggled to stop the robot and prevent it from running into obstacles. By allowing us to switch into keyboard mode, we are able to navigate the robot away from any hazards before resuming our mouse control.

What we Learned

Enacting this behavior taught us many useful skills. To create the advanced mouse-based Teleop control, we had to brush up on our trigonometry and learn about the odometry positions published by the neato. We discovered the Neato uses its encoders to automatically keep track of its x, y, and z positions relative to its initial origin. This information was extremely useful because it allowed us to accurately turn and go forward. This laid the groundwork for future behaviors like Autonomous shape driving.

Video

[Teleop Demonstration]

Behavior 2: Driving in a Square

In this behavior, the robot autonomously drives in a perfect 1 m by 1 m square. To create this behavior, we relied on the angular and linear odometry values to report the robot's position. We also created a proportional control loop that reduces overshooting and smooths out the behavior.

The Program

The Drive Square autonomous program is very simple but unique. There is an infinite loop where the robot first turns 90 degrees then drives one meter forwards then resets the origin. By resetting the origin, the robot thinks it is at 0 degrees and 0 meters again. This throws it back into the previous 2 if statements causing it to turn another 90 degrees then drive forward 1 meter. The robot continues driving the same square until either the mode is changed or the program is interrupted. This is a simple yet effective loop.

What we Learned

Programming this behavior taught us about the value of simplicity. We initially tried to program a longer, more complex routine but messed up the angle conversion several times. After realizing that the behavior can be abstracted to a simple turn and drive forward repeated four times, we wrote the new bug-free loop. This simple behavior helped us tackle complex behaviors by showing us that even seemingly simply movements can be broken down to even more simple motions. This thinking helped us structure and debug more complex behaviors.

Video

Drive Square Demonstration

Behavior 3: Wall Following

The third behavior we enacted was wall following. In this behavior, the robot uses its Lidar scanner to locate the nearest wall-like object, then navigates towards it. It stops a predetermined distance away from the wall, turns to position itself parallel to the wall, and then drives forward correcting its course so it continues to remain a stable distance from the wall.

The Program

The code uses the lidar to construct a rough location of several objects in the surrounding area. It considers an object to be a closely grouped cluster of five or more points. The robot looks for a large relatively flat surface and decides that object is the wall and not a trashcan, a chair or another robot. It uses the lidar to calculate the distance to the wall then it travels in a straight

line to the wall. It then turns and becomes parallel to the wall then continues on its way. While it is following the wall, it adjusts its rotational speed so it turns slightly left or right if it wanders too far or too close to the wall.

What we Learned

Creating this behavior taught us how to use the lidar and how to roughly detect objects. It was initially difficult to distinguish between the objects but we refined our algorithm to detect only walls. This was a good introduction into the lidar and its limitation and also introduced us to basic object detection.

Video

Wall Follower Demonstration

Behavior 4: Obstacle Avoidance

This behavior uses the lidar to navigate in a certain direction while avoiding obstacles. The Lidar senses the obstacle and the robot reacts depending on where the obstacle is and how far it is from the robot.

The Program

This program uses the lidar to detect the objects around it. All points detected within one meter from the robot apply a force on the robot forcing it to turn away and seek an alternate path. Using the object detection, it can create a vector field around it. The strength of the vector decreases with distance. The closer the object is to the robot, the faster it tries to get away from it. The robot navigates through the vector fields seeking the path of least resistance.

What We Learned

Creating this behavior taught us how to use vector fields to control the robot behavior. It also taught us the importance of object recognition. We initially did not group the points into objects and had the robot just avoid everything it saw this paralyzed the robot with indecision. After grouping the points into objects it was easier to figure out what to avoid so the robot was abe to both move and avoid hitting anything.

Video

Obstacle Avoidance Demonstration

Behavior 5: Person Following

This behavior is the opposite of obstacle avoidance. Instead of driving away from an obstacle, it moves towards it and tries to follow it.

The Behavior

The robot completes this behavior by locating a human like object then following the cluster of points as it moves around the space. The robot will only imprint on the closest roughly human leg sized object. It watches the object move then if it moves at a reasonably human like speed, it runs towards it stopping at a set distance away. This behavior, like the others, uses proportional control to slow down the robot as it approaches a target.

What We Learned

This behavior taught us how to detect moving objects. It also showed us how to turn the repulsive obstacle avoidance code into an attractive Person following behavior. We initially struggled to keep the robot following a single target. It often grew distracted and imprinted on a static object like a door or a trash can. We changed the code so it favors following objects that move on their own. This improved the design.

Video

Person Follower Demonstration

Behavior 6: Intelligent Teleop Control

For our last behavior, we combined the obstacle avoidance code with the advanced teleop control. When we were controlling the robot with our mouse, it automatically takes the shortest path to reach its destination regardless of what is in its way. We discovered that this makes it hard to use this teleop control because it would often try to drive through objects to get to its point. This is impossible. We decided to improve this by adding obstacle avoidance.

The Behavior

This behavior treats the mouse as an attractive object and anything else as a repulsive object. The neato is ultimately drawn towards the desired target but it is also repulsed by the things it passes. This helps it avoid hitting any obstacles in its path.

What We Learned

We learned how to balance the robot's repulsive and attractive forces. Initially, it was very difficult to manage this. The robot would either charge obstacles or it would never reach its ultimate destination. After careful tuning, we were able to balance the attractive and repulsive forces so the robot reaches its destination without running into anything. This behavior taught us the value, importance, and difficulty of tuning a robot.

Future Improvements

Throughout this project, we used propositional control to slow the robot down as it approaches its target. While this is effective, it slowed down our robot considerably. In the future, we would like to implement a full PID loop so the robot hits its target expeditiously.

Video

[Finite State Demonstration]