ME 280X Final Project

Joseph Rozell and Will Burken

Abstract

Cyber-physical systems are a pivotal part of modern society that shapes many industries from energy to manufacturing to transportation. These systems are incorporated to make processes both more efficient and safer. Some foundational concepts that are encapsulated with this subject are the communication and control of these systems. This project aids to give students a better grasp on the material that they are being taught using the Gazebo simulator. Tasks include optimized routes, leader-follower situations, and collision avoidance. There are many issues that were encountered and elegant ways to solve these problems.

Introduction

Cyber-physical systems (CPS) is a growing area of study that is invading many aspects of modern society including cyber security, transportation, and health care. With integrated computing, communication, and control, physical systems have the ability to interact with the surrounding environment. Systems such as these allow for tasks to be completed in an efficient manner; additionally, they can be optimized with decision-making strategies to ensure safety. In order to develop an overall understanding of these concepts, students have been challenged to complete three tasks.

Problem Setup

The first of these three tasks involved programming each simulator robot moving to four separate positions on the coordinate plane. They are controlled using both linear and angular PID controllers that's values must be optimized to provide the quickest and most efficient results. The next scenario challenged students to program the robots to create a *follow-the-leader* situation. For instance, if robot 2 were following robot 1, robot 1 would move its designated coordinates while robot 2 follows at a *safe* distance to follow, not cause a collision. Again, PID controllers will be utilized and their values optimized to complete in a timely and efficient manner. Finally, the last task asks students to modify the robot's programming with a collision-avoidance system and maneuver around obstacles to their designated spots. Similarly, to the first task, each robot must have its own unique position. All of these tasks should be completed using the Gazebo simulator and each robot is programmed with its own python script.

Methodology

PID Tuning

The PID controller was used to take in data from the robot's current position and velocity, compare that to the destination, and translate that to PID control. There were separate controllers for both the angular velocity as well as the linear velocity.

To tune the PID controller of the robot's movement, two approaches were taken. One approach was to automate the task with a python script. First, an instance of Gazebo would be created, then the turtlebot controller script would be started by the automation script. The turtlebot controller script took 8 inputs: The P, I, and D values of both the angular velocity controller as well as for the linear velocity controller, dubbed P_A, I_A, and D_A, and P_L, I_L, and D_L, respectively. The last two inputs were the x and y coordinates of the destination. To find the best PID values the automation script would run through each possible value of P_L, I_L, and D_L as well as each possible value of P_A, I_A, and D_A from 0 to 2 in steps of 0.1. For example, tests would run with the parameters of this pattern:

1st Run	2nd Run	3rd Run	… 21st Run
$P_A = 0.1$	$P_A = 0.2$	$P_A = 0.3$	$P_A = 0.0$
$I_A = 0.0$	$I_A = 0.0$	$I_A = 0.0$	$I_A = 0.1$
$D_A = 0.0$	$D_A = 0.0$	$D_A = 0.0$	$D_A = 0.0$
$P_L = 0.0$	$P_L = 0.0$	$P_L = 0.0$	$P_L = 0.0$
$I_L = 0.0$	$I_L = 0.0$	$I_L = 0.0$	$I_L = 0.0$
$D_L = 0.0$	$D_L = 0.0$	$D_L = 0.0$	$D_L = 0.0$

The input for the x and y destination was always (5, 5) for comparable results, although this point was chosen arbitrarily.

To determine the efficiency of each PID combination, a timer was run once each turtlebot started. The timer ends once the turtlebot has hit its destination. The best choice is then the one with the shortest runtime. Since a turtlebot could fly off course and never recover, the script would time out after 1 minute of runtime, and those PID values would be recorded as unfavorable values.

The other method of PID tuning was by manual inspection. First, All values were set to 0 except for P_L, which was set to 0.1. Then, P_L was slowly changed until the turtlebot performed more efficiently and approached the target closer and closer. Once this occurred, the other values were

tuned based on insight. For example, if the angular velocity would change too quickly and cause the turtlebot to spin off course, the D_A value was increased due to the fact that the D multiplier within a PID controller reduces overshoot in the total output.

Leader/Follower Methods

The main objective of the leader/follower logic was to create a system in which a turtlebot robot would try to reach and then follow another turtlebot that moved to reach its own position.

The logic is as follows: The leader publishes his position, and the follower subscribes to it. The goal position for the follower is then constantly changing, trying to keep up with the leader. Once the follower gets within a certain distance (0.75) of the leader, it no longer works on object avoidance and instead just follows exactly what the leader is doing, and the leader will then take care of object avoidance. This eliminates the issue of the follower seeing the leader as an object.

The follower robot has decided it's reached its goal position once it is within the 0.75 range, and the leader's velocity is less than 0.05. The reasoning for the velocity check is because we don't want the follower to decide it has reached its goal just because it gets close to the leader because the leader might still be trying to reach its goal. Instead, we want the follower to continue following until they both reach the goal together.

Object Avoidance Logic

The object avoidance system was a vital section of the script that allows the simulator robot to maneuver safely around the coordinate system. By utilizing the distance-laser system, a hundred and eighty-degree section in the front of the robot can be analyzed for obstacles. This buffer zone extends out in 0.75 which provides adequate spacing for the robot to take the necessary actions in order to avoid collisions. The first iteration of scripting had the robot move backward with the linear PID controls while the angular PID rotates the robot. This implementation accomplished its goal; however, there was a lot of unnecessary forward and backward movement which was not optimal for the robot to reach its position in a timely manner. With the second implementation, only the angular PID controller has non-zero values. This version provided a more optimal solution because the robot would not be continually moving in and out of the buffer zone; instead, the robot would just rotate instead of moving linearly. Overall, this allows for more optimal movement to the designated position.

Results

PID Tuning Results

The automated method of PID tuning ultimately failed to work. The main issues that arose were in the opening and closing of Gazebo, more specifically the rosrun file that launched it. It opened multiple processes that could not be easily closed, and due to a general lack of python experience, we were unable to make it work.

Values have been taken from other instances of using the PID controllers aided in making educated guesses about optimal values for this project. After continually attempting different combinations, the optimal values were found to be 0.25, 0, and 0 for the linear PID controller while the angular values were 5, 0, and 10.

Leader-Follower Result
The leader follower logic performed exactly as expected once it was tuned just right. No matter where the leader robot was, a follower could turn to it, and start driving, continually updating it's goal and therefore it's path. The run time for a bot to hit it's destination was not increased.

Object Avoidance Results
The first version of the object avoidance algorithm worked for three out of the four robots, but robot 3 experienced an issue that caused it to get stuck near the corner of an obstacle. This problem was remedied by tuning the PID values of that individual robot.

Discussion

PID Tuning Discussion
In the future, it would prove useful to put more time and effort into the automatic PID tuning script. If this were to be completed, we could tune it very precisely, and find a great solution. It would prove especially useful if it could be used in multiple areas, such as if we had multiple types of robots and the issue needed to be solved multiple times.

However, since this was a one-off project that required PID tuning for just one type of robot, the time it would take to fully implement automatic tuning would be greater than the time it took to find a solution manually that was sufficiently effective. Therefore, in this case, it was better to have a manual search as opposed to an automatic one.

Another important thing to consider is how we manually found our PID values. In the beginning, we did not have a routine way of finding it. Many values were plugged in, more or less at random. Once we determined this wasn't working, we then moved into a more methodical approach, which gave us much better results.

Manually tuning the PID was a hassle due to the infinite number of combinations that are present. However, understanding what each value controls (proportionally, integral, and derivative) helps with making educated decisions about the optimal set of values. Knowing this, the best option was to utilize a small integral linear value so that the robot will not be moving in an erratic manner. Additionally, the large proportional and derivative in the angular PID allow for easy rotation to the line of the most direct path. When using this combination, the simulator robots were about to get to their position in around thirty seconds.

Leader-Follower Discussion

At first, the minimum distance the follower had to be to the leader to successfully stop was too small and didn't take into account that the distance was being measured from the center of each robot. Therefore, the follower would just ram into the leader and push it around the map. This issue was solved by increasing the minimum distance to a larger size (From 0.25 to 0.75).
At first, the follower only looked at the distance between the two robots, and didn't check to see if the leader had reached its destination. As you can see, this caused issues in cases where the follower reached the leader faster than the leader reached its destination. The follower would get close, and stop moving while the leader kept going. This was avoided by requiring the follower to look at both the leader's speed and their relative distance.

When the follower would get close to the leader, it would see it as an object, and try to avoid it. To fix this issue, the logic of the follower was changed to use object avoidance only when it was far away from the leader. Once it reached the 0.75 threshold, the follower would no longer enact object avoidance and would just follow the leaders movements, relying on the leaders object avoidance.

Object Avoidance Discussion

The first implementation of the object avoidance system provided feedback that there were better ways to program the system instead of the constant back and forth. After receiving feedback, the decision to make only the angular PID values be non-zero improved the overall function of the system by stopping the robot's forward motion and letting it turn away from obstacles. Additionally, with this change, this is now a less likely chance that the robot will back up into anything during its avoiding process which overall improves the safety of the collision avoidance system.

Control Signal Smoothing

Control signal smoothing was implemented by using an EMA on the control signal, with the equation $EMA_{new} = 0.75*Input + 0.25 * EMA_{old}$. This was done because the EMA takes new input and places most of the importance on that, but also takes input from the past and uses that to smooth the signal. As can be seen in the videos, this does not affect the movement much at all. Maybe if our robots PID inputs made them more erratic and jerky, this would assist, but since our robots move fairly slowly, there is not much benefit from control signal smoothing.

Conclusion

In conclusion, our team worked on three major overall tasks: developing PID control for four turtlebot robots, implementing object avoidance logic to make sure they avoided obstacles, and leader-follower logic so robots can optionally follow other robots.

In PID control, two approaches were tried: an automated approach and a manual one. The manual approach is the one that proved to be fruitful due to issues with closing processes on the automatic one. The leader-follower logic worked well after some fine-tuning of end goals and following distance and worked well even when objects were placed in between. The object avoidance was initially random, and after adding in some extra logic to determine how it would turn, performed as desired.

Future Direction

In the future, there are multiple things that can be done to improve this project. First, more sophisticated logic and planning can go into object avoidance. Currently, it sees an object, reverses and turns whichever way is opposite of the object. In the future, it could scan further ahead in distance, and see objects sooner and try to avoid them while still following the path laid out in front of it. This would be more efficient than simply stopping and turning every time an object is seen.

Another method that could be implemented is Simultaneous Localization and Mapping, or SLAM. SLAM is a method for route planning and map making. Basically what it does is takes in sensor readings of everything around it, and every thing it sees, and makes a map of all the objects in its view and builds it as it goes. With this information in mind, the robot then is able to plan an optimized route around all the objects to complete it's goal using different algorithms like DFS, BFS, and Dijkstras. Using the SLAM method, we would be able to more efficiently get to our destination with a host of objects in the way. Although, something to be taken into account is the computational cost and memory requirements of this type of system. To create a 3D map, and run all these algorithms, a large cost is incurred compared to randomly bumping around until you reach your destination. In our case, for our problem at hand, neither of these methods is going to be very much faster than the other, but the method implemented has a much lower computational cost.

Works Cited

Martin, Scott. “What Is Simultaneous Localization and Mapping? What Is Simultaneous Localization and Mapping?” *NVIDIA Blog*, 25 July 2019, https://blogs.nvidia.com/blog/2019/07/25/what-is-simultaneous-localization-and-mapping-nvidia-jetson-isaac-sdk/. Accessed 2 December 2021.

Sarkar, Soumik, et al. *Lecture 13 and 14 Introduction to Robot Operating System (ROS)*. October 2021.

Sarkar, Soumik, and Xian Yeow Lee. *Lecture 20 Introduction to Gazebo + ROS Gazebo*. November 2021.

“What Is SLAM (Simultaneous Localization and Mapping) – MATLAB & Simulink - MATLAB & Simulink.” *MathWorks*, MathWorks, 2021, https://www.mathworks.com/discovery/slam.html. Accessed 2 December 2021.