University of Colorado - Boulder

Lab 2: Odometry

20 September, 2022

Lab Section 012, Group: name_group

Authors:
Caleb Bristol
Aidan Jones

Professor:
Nikolaus Correll

I. Robot Behavior

Q: What happens (in terms of the robot's behavior) during the robot.step(TIME_STEP) statement?

A: During each time step, the ground sensor values are read in, and checked against known values to determine the robot's course of action. The wheel speeds are then set based on the sensor values, as according to the lab document. Then, the inputted wheel speeds are passed into the odometry equations to update the robot's pose. Finally, there is a check for the starting line, which flags the robot as being on the starting line or not. If it is, then the pose values are zeroed out. Finally, the current pose is printed to the command window and the motor speeds are sent to the motors.

II. Variance in TIME_STEP

Q: What happens if your robot's time step is not exactly TIME_STEP long, but slightly varies?

A: If the robot's time step is slightly less than the simulation time step, the robot will wait after issuing the robot.step command for the simulation time step to finish and "re-synchronize" with the robot. If the robot's time step is slightly longer than the simulation time step, the simulation will pause until the robot.step command passes. This would make it inconsequential if there was variance in the time step as the simulation would automatically correct for it.

III. ePuck Average Speed

Q: What is the ePuck's average speed (in m/s) from Part 1?

A: The average speed was measured to be 0.1265 m/s.

IV. Ideal Pose

Q: In an ideal world, what should the ePuck's pose show each time it crosses the starting line?

A: The ePuck's pose at the starting line, given ideal conditions, should return to its initial pose, or (x_i, y_i, θ_i) = (0 m, 0 m, 0 rads) in the world coordinate frame { i }.

V. Loop Closure

Q: How did you implement loop closure in your controller?

A: The closed loop controller used a flag which would be set to 0 by default, and would get flagged if the right conditions were met that the robot should be at the starting line. First it checks if all three ground sensors register a line, thus indicating a horizontal line. This flagged occasionally during the turning sections, so it was updated to also check the angle of the ePuck, as it would face South while it crossed over the start line. This included a margin of about 45 degrees in either direction to compensate for the odometry errors.

VI. Time Spent

Q: Roughly how much time did you spend programming this lab?

A: About two hours were spent doing the programming for the lab.

VII. Problems Encountered

Q: Does your implementation work as expected? If not, what problems do you encounter?

A: The implementation mostly works as expected. The odometry isn't as accurate as it could be, as the robot spends a fair bit of time jittering on turns, and by the time a lap is completed, it consistently overestimates its pose. This is remedied by the closed loop system, however, so it would only become a real problem if the implementation were expanded beyond the scope of a closed track. However, the implementation wouldn't work outside of a closed track anyways, so this problem is inconsequential.