Peter Huynh

September 9, 2016

CSCI 3302

Lab 2

Robot Odometry

The goals of this exercise are to implement the forward kinematics of a differential wheel robot on an embedded system to experience and quantify sensor noise from wheel-slip, discretization, and real-time violations to understand the concept of "loop closure"

You'll need:

- A Sparki robot
- A working implementation of a line-following algorithm
- The ArcBotics line following poster

Overview

Odometry integrates the wheel speeds of your robot to calculate its 3D pose (x, y, θ) on the plane. In order to do this, you need to write down the forward kinematics of your robot. As your robot is non-holonomic, you need to calculate the velocity along the robots x and y-axis, as well as the rotational speed around its z-axis. You then need to transform these velocities into a global coordinate frame and add them up to calculate the robot's pose. You will find out that you cannot do this without error as the robot slips, time discretization introduces inaccuracies, and your computer might not be fast enough to perform calculations fast enough.

Instructions

1. Inspect the line following code provided by ArcBotics. What happens during the "`delay(100)`" statement and where would you want to introduce your odometry code?

   At the function call `delay(100)`, the robot will pause for 100 milliseconds, allowing for enough time for the previous call(s) to finish. In my odometry code, it might be best to call `delay` when my robot completes all task prior to 100 milliseconds, calling for a delay of remainder of the time. This is done in order to ensure each loop runs exactly for the specified timeframe and gives time to complete tasks.

2. Think about what would happen if calculating a position update would take longer than 100ms. Think about ways to measure the time the execution of a command takes and making sure every loop takes exactly 100ms. Hint: check out the millis library for Arduino.

One possible way to ensure the position update doesn't exceed 100ms is to record for the loop's starting time and ending time by setting two separate variables to `millis()`, subtract their difference from 100ms and delay for the remaining value. This ensures that the program will run for a varying time, and delay for whatever remains to sum up to 100ms. However, I must also take into account for if the loop exceeds 100ms, and should set a hard loop exit if it takes too long.

3. Use the millis library to measure how long it takes the robot to move 30cm and calculate its speed from there. This will allow you to calculate its speed in m/s without actually measuring the wheel diameter.

   Average: 10773 milliseconds to travel 30 centimeters ➔
   0.3m / 10.773s = **0.02785 m/s**

4. Implement odometry code following Chapter 3 in the textbook. Calculate the actual distance by multiplying the speeds with the time that the robot actually moves. Make sure the robot actually moves for exactly 100ms, not the time of the delay plus the time it takes to execute your code! Initialize your pose with (0, 0, 0). Using equation 3.41 to integrate your speeds into positions.

   Check source code: Completed with x, y, and theta float values.

5. Display the robot's pose on its display. (Make sure this operation does not destroy your timing!). What do you expect the display to show when the robot arrives at the start line at the second (the third, the fourth) time? What actually happens?

   I expected the robot to simply draw over the same pixels on the display, however it started deviating and draw under the initial pixels. This is most likely because the drawing is not entirely accurate to the line it is traversing.

6. Use the robots sensors to identify the start line. How could you exploit this information to reduce your error?

   I simply set the math variables and display to reset to ensure that Sparki will redraw the robot in the correct position again after one lap. Resetting the display was unnecessary, just additional, as the variables being set to 0 allowed for the proper redrawing.