Lab 2: Odometry

Lab Instructions

Objective

To determine the accuracy of the implemented odometry system, and implement a simple correction using a light sensor.

Method

1. In the file `Odometer.java`, implement code that performs the task of an odometer as described in the Odometry tutorial and in class. You should only need to add member variables to and modify the `run()` method of the `Odometer` class.

2. Run the robot in a 3-by-3 tile square (where one tile is 30.48 cm in linear dimension) using the provided code and tweak the `leftRadius`, `rightRadius`, and `width` parameters passed to the `SquareDriver.drive()` method in `Lab2.java` until the robot returns (approximately) to its starting position. If your left and right wheel motors are not connected to motor ports A and B respectively, you may need to also modify those parameters. The call to `SquareDriver.drive()` is shown below.

   ```
   SquareDriver.drive(leftMotor, rightMotor, leftRadius, rightRadius, width);
   ```

3. Now, run the robot in a 3-by-3 tile square ten (10) times and measure the *signed* x and y distances from the actual location of the robot on the field (recalling that its starting position is the origin) to that reported by its odometer. Feel free to calibrate your odometer prior to this step. Record these results. (Note: Your odometer is reporting the position of the point directly between your wheel axles).

4. In the file `OdometryCorrection.java`, implement code that, using the light sensor, detects grid lines and updates/corrects the odometer's position data as needed. Your robot should start its motion in the center of a tile, and thus grid lines will occur at x = 15 cm, 45 cm, 75 cm, … and y = 15 cm, 45 cm, 75 cm, …. You do not need to account for grid line intersections in this case.

5. Repeat step (3) with your odometry correction enabled. Do not recalibrate your odometer.

6. Demonstrate to a TA your code. The TA will first 'float' your motors (i.e. the motors will be shut off in such a way that they do not provide resistance to being backdriven) and push your robot on the field to confirm that your odometer works correctly. The TA will then start your robot off-center in a tile and look for it to report its correct position after running in a 3-by-3 tile square.

Data

Provide two tables (with units clearly indicated) with the results from steps (3) and (5).

Data Analysis

a) What was the standard deviation of the results without correction (compute it for x and y separately, and provide the four (4) values in a table)? Did it decrease when correction was introduced? Explain why/why not. (Note: If you are unfamiliar with the concept of standard deviation, read the Error tutorial posted on WebCT).

b) With correction, do you expect the error in the x position or the y position to be smaller? Explain.

Observations and Conclusion

Is the error you observed in the odometer (without correction) tolerable for larger distances (i.e. circumnavigating the field requires a travel distance five (5) times larger than that used for this lab)? Do you expect the error to grow linearly with respect to travel distance? Explain briefly.

Further Improvements

a) Propose a means of, in software, reducing the slip of the robot's wheels (do not provide code).

b) Propose a means of, in software, correcting the angle reported by the odometer, when (do not provide code):

   i. The robot has two light sensors.

   ii. The robot has only one light sensor.

To Submit

- One document in .pdf format containing the lab report

FAQ

In past semesters some questions have commonly arose. Hopefully this will answer most of the questions that students have. Of course TAs can explain further these questions and will be available to do so.

a) **My odometer reports that my x-position is increasing, when I want my robot to be moving forward in the y-direction. Why?**
The odometry tutorial (posted on WebCT) assumes that the robot's initial orientation is 0 radians, which by convention is pointing along the positive x-axis. Furthermore, it assumes that angles are increasing counterclockwise (i.e. a positively oriented coordinate system). Use the `setPosition()` method to set your robot's initial orientation to 90 degrees or $\pi/2$ radians, or modify your odometer code for the alternate convention, which is clockwise-increasing angles starting at the positive y-axis (like a compass).

b) **Do I need to use the `synchronized()` blocks?**
Yes. Get used to using them, because for your final project they will be essential to minimizing the number of bugs in your code. Available online is the Java Tutorials Lesson on Concurrency, and it strongly advised that students struggling with the concepts of multithreading and synchronization read this *before* asking TAs for help.