E190Q – Autonomous Robot Navigation

Lab 2

Odometry

INTRODUCTION

Odometry is a useful method for predicting the position of a robot after it has moved. The prediction is accomplished by counting the number of wheel revolutions that each wheel rotated, then converting this to motion to coordinates a global coordinate frame. Unfortunately, this method is prone to errors from slipping, and poor modeling of the system (e.g. wheel dimensions).

This lab requires students to implement odometry in the Jaguar Lite platform, and characterize the types of errors that can be encountered.

BACKGROUND

The encoders used on the Jaguar Lite are called incremental encoders. They measure the signal from photodetectors located on one side of a transparent disk. The disk rotates around the motor shaft. Located on the other side of the disk are photoemitters. Printed on the disk are tracks of black lines that shield detectors from emitters.

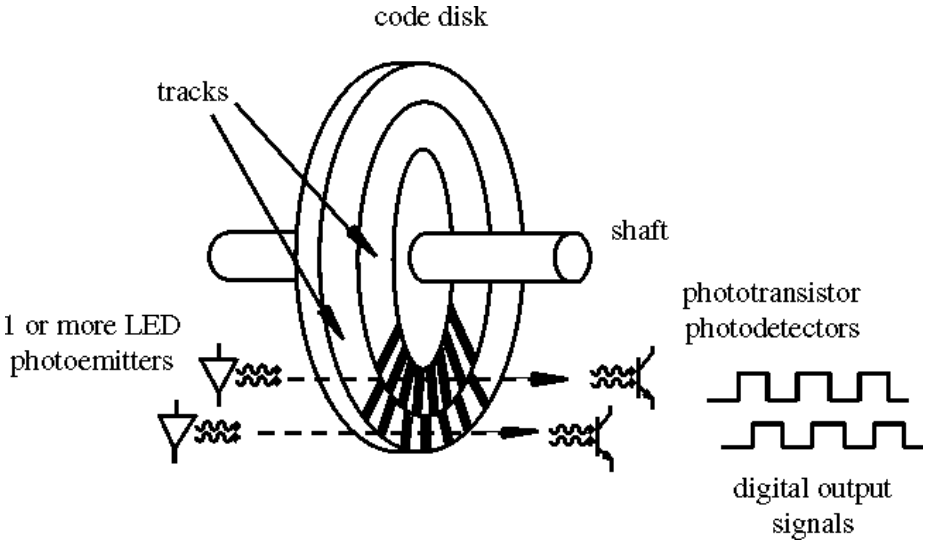

Figure 1: An incremental Encoder (from http://www.kavlico.com/index_home.html)

As the disk rotates, the output of the photodetectors oscillates between LOW and HI, as shown in Fig. 1. By counting these signal changes, the amount of rotation can be measured. The resolution of the measurement is dependent on the number of lines in the grating.

Build from the lab base code, (see web site lab page). The code to be modified is located in the file Navigation.cs. Within this file there is function called from the main control thread. This function, `runControlLoop()`, is called at every time step of the thread.

To localize the robot, `runControlLoop()` will call two functions that you will modify. The first is called `MotionPrediction()`. The second is called `LocalizeRealWithOdometry()`. You can comment out the call to `LocalizeEstStateWithParticleFilter()`.

Within `MotionPrediction()`, you are required to calculate the `distanceTravelled` and `angleTravelled` since the last time step. With these two variables calculated, the new state *x, y, t* will be calculated in `LocalizeRealWithOdometry()`.

Note that in this particular lab, we will set the actual state *x, y, t* to be the same as the estimated state *x_est, y_est, t_est*. The 2D graphics window will display the robot at the actual state.

EXPERIMENTS

1. Read the Sensors

The Jaguar Lite odometry lab uses two encoders, one located on each drive wheel to measure the wheel distances. To access these measurements, the following functions are used.

```
GetEncoderPulse4()
GetEncoderPulse5()
```

These functions are used to set the variables `currentEncoderPulseR` and `currentEncoderPulseL`. Track down where and how this is done. Make sure it is done at each iteration of the control loop.

2. Calculate the Encoder Count Difference

To get the motion of the past time step, we must difference the current sensor reading with the last sensor reading. This will reflect the number of encoder pulses that were counted during the last time step. We call these differences `diffEncoderPulseR` and `diffEncoderPulseL`.

Within the `MotionPrediction()` function, set these encoder differences using the variables `lastEncoderPulseL` and `lastEnoderPulseR`.

3. Eliminate Encoder Rollover

The encoder signal sent from the robot, ranges from 0 to 32767. This must be taken into account when taking the difference of two consecutive encoder measurements.

For example, consider a case where the encoder measurement at one time step is 32763, and 11 at the next time step. The actual difference between the two time steps is 16 pulses, but taking the difference between these two values would result in -32752.

Check that this rollover problem did not occur when calculating the encoder count differences, and correct for the problem where it occurs. Note that 32767 is stored as a constant in Navigation.cs named `encoderMax`.

4. Update the Last Encoder Count Variables

Add code to remember the current encoder measurements for use in the next time step, (i.e. set `lastEncoderPulseR` and `lastEncoderPulseL`).

5. Calculate Wheel Distances

Based on the number of pulses counted over the last time step, calculate the distance traveled by each wheel: `wheelDistanceR` and `wheelDistanceL`.

You will need to make use of the fact that the encoder has maximum 190 pulses, set as constant `pulsesPerRevolution` in Navigation.cs. You will also need the wheel's radius, set as the constant `wheelRadius`. You may want to double check this value to make sure it is accurate for your robot.

6. Calculate the Angle and Distance Travelled

Using the distance each wheel travelled, we can calculate the distance the center of the robot travelled `distanceTravelled`, as well as the change in orientation `angleTravelled`. The equations required are presented in lecture. You will need the constant `robotRadius`.

This is the last code to be added to the function `MotionPrediction()`.

7. Update Robot States

Within the function `LocalizeRealWithOdometry()` that is called after the function `MotionPrediction()`, update the new position and orientation of the robot: x, y, t. At this point, you can also set the estimated states to be equal to the actual. Make sure the angles remains between $-\pi$ and π.

The 2D graphics window should show the robot's position with respect to the Cartesian coordinate frame. Grid cells are 1 m x 1 m in size. Drive the robot in simulation and hardware modes to ensure the odometry is working. Check that no encoder rollovers occur.

8. Characterize Errors

Setup a set of experiments to determine the types of errors that are usually encountered with your robot. For example run the robot through several tests that move the robot straight ahead d meters, where d takes on values 0.5, 1.0, 1.5, 2.0, 2.5, …, 5.0. Repeat the test many times for each value of d. At the end of each test, record the predicted position of the robot (from odometry), and the actual measured position (use a ruler). For each value of d, calculate the mean error in the *x*, *y*, *t* coordinate directions. Be sure to log your data.

Run similar tests where *d* remains constant at 0, but there are various changes in orientation.

This part of the lab is meant to be open ended. You may perform different tests and get different results than other groups.

DELIVERABLES

1. Demonstration

Before the end of the final day of this lab, you must demonstrate to the Professor that your odometry is working properly. In both simulation and hardware mode, the 2D graphics window should show the robot and estimate movement according to the motion control commands.

2. Submit

Write a report (2-10 pages) describing your findings from the error characterization step. Be sure to include the following sections: abstract, introduction, background, problem definition, control design, results, conclusion. Be sure that your results section includes plots of robot trajectories, odometry errors, etc.

Note, all lab documents in this class will follow the template found at:
http://www.ieee.org/conferences_events/conferences/publishing/templates.html

The report is due 1 week after the final day of this lab (09:00am on Friday, Feb. 13th).