E160 Lab 02

Odometry Spring 2018

1. 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, measurement resolution, and poor modeling of the system (e.g. wheel dimensions).

This lab requires students to implement odometry in the pololu romi robot platform, and characterize the types of errors that can be encountered. The sensor used will be the romi encoder kit, (see Fig. 1 below).

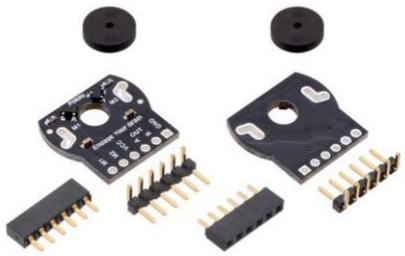

Fig. 1: Pololu Romi Encoder kit.

As described on the Pololu website, “The encoder board senses the rotation of the magnetic disc and provides a resolution of 12 counts per revolution of the motor shaft when counting both edges of both channels, which corresponds to approximately 1440 counts per revolution of the Romi’s wheels.”

2. Background

Download the lab 2 base code, (see web site lab page). The code to be modified is located in the file E160_robot.py. Within this file there is function called `update()`, that is called at every time step of the main thread in the E160_gui.py file.

To localize the robot, the `update()` function calls `localize()` on line 46 which is reponsible for updating the robot’s state estimate to be stored the variable name `state_est`. Within the `localize()` function, there are two function calls, the first function called is `update_odometry()`. The second is `update_state()`. All your code for lab 2 should be written in these two functions.

Within `update_odometry()`, you are required to calculate the `delta_s` and `delta_theta` since the last time step. With these two variables calculated, the new state x, y, t will be calculated in `update_state()`.

Note 1: that the 2D graphics window will display the robot at the actual state.

Note 2: in this lab, you will need to toggle back and forth between using the real robot hardware and simulating the hardware. To do this, you must modify the line 25 of the file environment.py.

3. Experiments

1. Read the Sensors

The odometry lab uses two encoders, one located on each drive wheel to measure the wheel distances. To access these measurements, the following function is used on line 43 of E160_robot.py.

```
update_sensor_measurements()
```

This function returns the two encoder measurements that are then assigned to `self.encoder_measurements`. Depending the orientation of your robot, figure out which of the two array element belongs to which wheel.

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. To do so, set these encoder differences as variables `diffEncoder0` and `diffEncoder1` within the `update_odometry()` function. You will need to use and then record the previous encoder measurements `self.last_encoder_measurements`, which is 1x2 array variable.

3. Start up

When the code is started, the hardware will send whatever encoder measurements were logged by the teensy the last time the robot was used. This will cause the diffEncoder variables to jump at the beginning since the last_encoder_measurements variable will not be set properly. An easy hack is to set the diffEncoder variables to 0 if the calculated jump is > 1000.

4. Calculate Wheel Distances

Within `update_odometry()`, calculate the distance traveled by each wheel and store them in variables named `wheelDistanceR` and `wheelDistanceL`.

You will need to make use of the fact that the encoder has maximum 1440 pulses, set as constant `self.encoder_resolution`. You will also need the wheel's radius, set as the constant `self.wheel_radius`. You may want to double check this value to make sure it is accurate for your robot.

5. Calculate the Angle and Distance Travelled

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

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

6. Update Robot States

The function `self.update_state()` should return the new state (position and orientation) of the robot: x, y, $theta$. Make sure all angles remain between $-\pi$ and π. The 2D graphics window should reflect the robot's movement.

7. Characterize Errors

Setup a series 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 (Feb. 11th 2018), you must demonstrate to the Instructor 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 here. The report is due 1 week after the final day of this lab (09:00am on Friday, Feb. 18th).