Thursday Laboratory Exercise 2: Proportional Control Implementation

Objectives

In this laboratory exercise, you will modify your controller to create a proportional controller that will improve the RACECAR's ability to follow the wall.

Deliverables

- ❑ Demonstrate understanding of proportional control introduced in lectures
- ❑ Learn to fit lines to data and to get a better estimate of angle and distance
- ❑ Build software that has dynamically controllable behaviors
- ❑ Plots from your recorded data that show the improved performance of your controller

Challenge Problems

- ❑ Improve your controller to more accurately estimate its position
- ❑ Use a System Model to develop and validate your algorithms before building the system

Proportional Controller Implementation

Now that you have all the pieces in place, it will be easy to upgrade your controller to implement the proportional control you have seen in the lectures. You already have the error estimate that you calculated from the perpendicular distance to the wall. That just needs to be scaled by a constant amount, the proportional gain (G_p), that you will determine empirically. So go ahead and make those modifications to your code and test them in simulation first, if possible, before running on the RACECAR to make final tweaks.

Controllable Behaviors

For the end-of-week Drag Race, you will be told which wall to follow and it will likely be different in each heat. That choice needs to be one you can easily make on demand at the start of each race. There are several ways that you can adjust parameters at run-time in ROS, you've already used command line parameters in the Bang-Bang controller. We would like all teams to do something much more dynamic than that and use the Gamepad buttons to trigger which wall to follow. The blue X button and red Square buttons could be used to cause the RACECAR to follow the left and right walls respectively. You have already experimented with the joypad as a sensor, now you can put that button map you made to good use.

1. You may want to setup a callback function that is invoked whenever any message is published on the /joy topic (and remap it as appropriate when running with either the RACECAR or simulator).
2. When neither button is pressed, you should make sure to send a "Stop" message to the car.
3. Consider what might happen if both buttons are pressed (do the right, or at least a known thing)

Improving the Result

Up to this point, we have only been using two data points from the laser scanner to estimate the distance from the wall (and its relative angle). Errors in those measurements introduced by noise or "imperfections" in the environment can dramatically affect the results. We can use more points from the laser scanner to help with that problem. One possibility is to use more points in the local area of l_{-90} and l_{-60} and filter them (perhaps taking the median). Another is to use all the points between them in a more "optimum" way. This can be done with the Python function polyfit in the numpy module. Experiment in Python from the command line with some data you make up to get a feel for how it works. Basically, numpy.polyfit(x, y, 1) returns the slope and intercept of the line that best fits the x and y data.

An ipython example session:

```
In [1]: x=[0,1]
In [2]: y=[0,2]
In [3]: import numpy as np
In [4]: np.polyfit(x, y, 1)
Out[4]: array([ 2.,  0.])
```

Developing Algorithms with a System Model

Once you are comfortable with the function itself, you can verify your algorithm and assumptions by creating a representative set of data that looks more like what you will get from the laser scanner. Since you build the model, you know exactly what the results should be and it can be as simple or complex as you need. In Python:

1. Generate a data set the same size as what you will be processing with the same angle spacing
2. You should be able to control the distance to the wall and vehicle orientation in your model
3. Use simple trig functions to calculate the range to each point on the wall from [-90°..-60°]
4. The first step in your processing will be to convert them (back) to Cartesian coordinates
5. Then you can use the polyfit function to extract the line parameters as above
6. Standard "closest point on the line" techniques will give the perpendicular distance to the line you can use in the error calculation for your proportional controller
7. Once you are convinced it works as you expect debugging on the RACECAR will be much easier