Comp790-058 Homework 3

Fun with PID and Robot Kinematics

DUE: 11/17/2017 by 11:59:59 PM

This homework consists of three problems related to planning for mobile robots. For each question, you will need to write code and answer several questions. Your final submission should consist of a zip of your code as well as your written report with the requested details for each question. You are allowed to complete the homework in the language of your choice; however, it is highly recommended you complete this homework in Python as code is provided for visualization and simulation in Python. If you strongly prefer another language, please email me at best@cs.unc.edu so we can setup a robot model and water heater in your language. A complete set of sample code and resources for the project can be found here **[**https://goo.gl/Q4WjVf**]**. For each problem, it is critical that you not modify the properties of the provided systems. Do not alter the water heater code, nor the differential-drive robot code. Your submission will be checked for modification of these files. If you modify the systems in an unapproved way to make the problems easier it will be considered **cheating.**

Problem 1: PID Control

Problem 1A) Implement a PID controller for a single input, single output system. Your controller should support the following:

Setting each gain at runtime

Setting / adjusting the time factor (you can use the same time for I and D)

Setting a dead zone tolerance (range within which PID controller will not update, just return the prior value)

Setting upper and lower bounds on the Control Output

The single input / output system you will be controlling is a heating-element for a simulated water heater. Your Process Variable (PV) will be a temperature in Celsius and your Control Output is the gas proportion from 0 to 1. I have provided the simulated water heater in Python in the sample code provided above in the **Problem1A** folder. The water heater has the following methods:

SetGasInput(float) - sets the gas input for the next step

AdvanceTime() - updates the gas input, advances time by 1/4 second

GetTemp() - returns the temperature of the water in the tank (in **c**)

You will need to demonstrate that your PID controller can hold the water temperature steady at **35c**, **55c,** and **75c**.

Specific Deliverables:

A) Write a PID controller class as described above.

B) Using your PID controller class and the simulated water heater I have provided, create a main file which uses your PID controller to control the water heater at the set points described (35c, 55c, 75c). **You do not need to write a simulator; the sample code contains all you need to simulate the heater for this problem.**

C) Write-up: Provide your gains for P, I, and D for your controller. You should have one value for the entire simulation, not one for each set point. Provide a visualization of your controller's convergence for the three set points. A graph is sufficient. I have attached a sample graph from lecture slide 64 at the end of the assignment. Also, indicate the rise time between setting a target temperature and achieving that temperature consistently (within 5% of set point).

Hint: As we discussed in class, tuning a PID controller is somewhat trial and error. You will do yourself a favor to define an interface which allows you to do so quickly or at runtime. Python has a multitude of visualization and interaction libraries, including QT in which the visual code samples are provided. Devising a PID tuner should not take a substantial amount of time so you may find it worth the effort to write a tuning program.

Problem 1B) Using your PID controller class from **1A**, define a controller for a differential drive mobile robot (think Roomba, as we discussed these in class). The simulated differential drive robot you will work with has two control inputs which represent the power to each wheel (u_L and u_R). The state space of the robot is position and orientation(x, y, Θ). Your controller class should take as input a desired linear speed and turning rate while providing the necessary controls for the robot to achieve these. The robot's max linear speed is +/- 1.5 m/s for each wheel. As a caveat, the robot's power will vary slightly every second, causing minor changes in the wheel motor response rates. Your controller will need to compensate for the changes (hint… PID).

The kinematic equations of motion for a differential drive robot are provided below:

$$\dot{x} = \frac{\boldsymbol{r_w}}{\mathbf{2}}(\boldsymbol{\omega_L} + \boldsymbol{\omega_R})\boldsymbol{cos}(\boldsymbol{\Theta}) \qquad \dot{y} = \frac{\boldsymbol{r_w}}{\mathbf{2}}(\boldsymbol{\omega_L} + \boldsymbol{\omega_R})\boldsymbol{sin}(\boldsymbol{\Theta}) \qquad \dot{\boldsymbol{\Theta}} = \frac{\boldsymbol{r_w}}{\mathbf{2}\boldsymbol{r_r}}(\boldsymbol{\omega}_{\boldsymbol{R}} - \boldsymbol{\omega}_{\boldsymbol{L}})$$

The robot in question has the following model properties:

Radius (r_r): .25 meters **Wheel radius(r_w):** .102 meters **max linear wheel speed:** 1.5 m/s

A basic visualizer and simulator is provided in the sample code in the folder Simulator. This code includes a base robot model which can be sub-classed to create your robot. The methods of the robot are documented in the code. Instructions for running the simulator are provided at the end of the document.

Specific Deliverables:

A) Write a robot controller which accepts as input target speed and turning rate. You can subclass the provided differential drive robot, and use your PID controller from problem 1A to accomplish this. **You will not need to write a simulator. The simulator provided is sufficient for this problem.**

B) **Write-up**: Provide the P, I, and D gains for your controller(s), however many you used for the problem. Explain how your controller works and how you chose to tackle the problem. Provide a visualization of your differential drive robot moving at the target speed and turning rate (graphs similar to those in 1A are sufficient for this problem).

Hint: There are multiple ways to tackle this problem. Differential drive robots turn by imbalance in the wheel speeds. One approach is to use a PID controller for each wheel. Another is to use a PID controller for linear speed and a second for heading mixed after (like a quadrotor). You can combine these approaches as well.

Problem 2: Waypoint Following for a Differential Drive Robot

Implement a waypoint follower for your differential drive robot from problem 1. Given the heading and distance to the next waypoint, determine appropriate set points for the PID controllers you implemented. For Problems 2 and 3, you can assume a waypoint has been successfully visited if the robot travels within 0.5 meters of the waypoint. I have provided two test sets of waypoints in the **Resources/Problem2** folder. The waypoints should be followed in order and are in the form X [space] Y with one waypoint per line.

Specific Deliverables:

A) You will need to write a planner for your robot. You can sub-class your robot from Problem1B to accomplish this, or write a controller which determines the appropriate set controls for the robot. It will also need to determine when the robot has reached a waypoint and set the next waypoint as the goal. In either case, **you will not need to write a simulator for this problem.** You can subclass the simulator to add extra features, but you should not need to implement an entire new simulator.

B) Provide a visualization of your robot travelling through the two sets of waypoints. This should be a video. The provided simulator visualization is sufficient.

C) Write-up: provide a short description of your waypoint follower code, maneuver choice (if used), and the time your robot took to complete each set of waypoints. The fastest robot time (with video verification) will receive bonus points.

Hint: There are multiple ways to proceed to a goal. Your robot can choose different maneuvers to accomplish the task. For example, the robot can turn in place to face the goal then travel towards the goal on a direct heading, or simultaneously correct heading and speed. Choosing the right maneuver for a given set of steering and speed is the key to quickly and effectively completing the course. Also, when making tight turns, lower speed can be advantageous. Experiment with your robot's maneuver choices to find the best combination.

Problem 3 [Updated 11/12/2017 BONUS PROBLEM]: Implement a car-like robot and pure-pursuit controller

For this problem, you will implement a simulated car-like robot using the single-track bicycle equations of motion we discussed in class. Your robot must accept steering and speed as control (u_s, u_ϕ). You do not need to account for dynamics, nor do you need to implement acceleration or steering rates. Assume the robot can instantly achieve changes to steering and speed. Also, choose reasonable values for the length, max speed, and max steering of the vehicle. Your robot does not need to be capable of travelling at car-like speeds, but should support a max speed of at least 5 m/s. Your robot should also support assignment of a goal position to which it will travel by pure-pursuit (i.e. heading towards the goal at all times on an arc). You can use the robot provided for problem 1B as a model for your car-like robot should you choose to use the provided simulator.

The single-track bicycle equations of motion are provided below:

$$\dot{x} = \boldsymbol{u_s cos(\Theta)} \qquad \dot{y} = \boldsymbol{u_s sin(\Theta)} \qquad \dot{\Theta} = \frac{\boldsymbol{u_s}}{\boldsymbol{L}} \boldsymbol{tan(u_\phi)}$$

You will need to choose reasonable values for:

Interaxle length (L)   max/min steering ($\phi_{\min} \leq u_{\phi} \leq \phi_{max}$)   max/min speed ($s_{min} \leq u_s \leq s_{max}$)

Specific Deliverables:

A) Using the differential-drive robot as a guide, write a new robot class for your car-like robot. **You do not need to create a simulator for this problem, only a robot.** You can ignore the functions related to power shift. You will need to implement the UpdateConfig function to evolve the robot forward according to its equations of motion. **You do not need a PID controller for this problem.** The robot can instantly achieve new controls and has no uncertainty, therefore a PID controller is not needed, but can be used.

B) Using the QTDifferentialDriveRobot class as a guide, write a visual representation for your car-like robot. This can be as simple as a single rectangle, but is needed to visualize the robot.

C) Write a waypoint follower for your car-like robot. This is similar to problem 2.

D) Provide a visualization of your car-like robot travelling through two sets of waypoints you have created. These should be videos. The provided visualizer is sufficient for these videos.

D) Write-up: Describe your controller. How did you decide the appropriate steering and speed for reaching a waypoint? Describe your choice of relevant model parameters. Provide two sets of waypoints you chose with your code submission. Provide the time required for your robot to travel each set of waypoints.

Hint: I am not providing waypoints for this problem. You will need to choose reasonable waypoints for your car to track. Remember, there are kinematic limits to the maneuvers your vehicle can perform. Do not choose waypoints which are within the minimum radius the vehicle can turn, or your car will never achieve the waypoints!

Supplemental Materials

Python Simulator Code

To use the python tools provided, you will need the following python version and packages:

Python 3+ (tested on 3.6)

pyQT5

Numpy

You may also need

Pyqt5-tools

The easiest way to accomplish this in windows is to install the latest WinPython(http://winpython.github.io/).

Most linux distributions provide packages for pyQT and numpy. Mac-os may be more complicated, but there are homebrew packages available.

UPDATED 11/1/2017

I have provided a more thorough simulator, as I found the first version difficult to work with. You can use either for your project, but I believe the newer simulator is much easier.

To run KinematicSim2d:

See the readme file in the simulation. But in general, from the SampleCode folder, run the Following:

```
python -m KinematicSim2d.scenes.[your_scene_name]
```

To run the old sim:

To run the simulator, navigate to the **SampleCode\PreSimulator** folder. Run the following command:

Python QTSimulator.py

By default the differential drive robot will be attempting to trace a circle.

Sample graph

Example of an error over time graph for a PID controller from the lecture. Yours should show different set points.

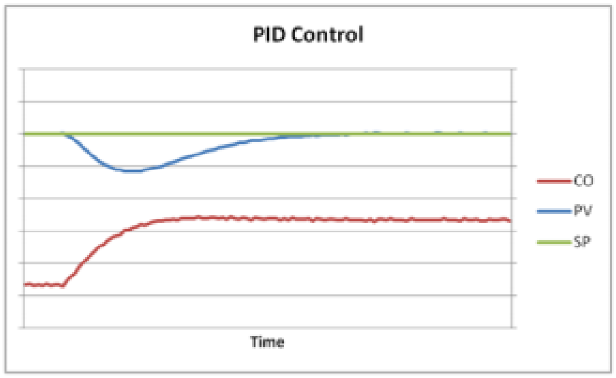

Figure 13
A PID Controller's Response to a Disturbance

Help & Relevant tutorials / materials

More information on relevant topics can be found here:

PID:

https://www.dataforth.com/introduction-to-pid-control.aspx

https://en.wikipedia.org/wiki/PID_controller

Differential-drive and car-like kinematics:

http://planning.cs.uiuc.edu/

Chapter 13