docs.duckietown.org

Part 1: Altitude PID in Simulation

4-6 minutes

Modified 2020-11-04 by Sean Hastings

In this part of the project, you will be implementing a PID controller for a simulated drone that can only move in one dimension, the vertical dimension. You can control the speed the motors spin on the drone, which sets the thrust being generated by the propellers. In this system, the process variable is the drone's altitude, the setpoint is the desired altitude, and the error is the distance in meters between the setpoint and the drone's altitude. The output of the control function is a PWM (pulse-width modulation) value between 1100 and 1900, which is sent to the flight controller to set the drone's throttle.

You should implement the discretized version of the PID control function in *student_pid_class.py*:

Notice that there is an extra offset term added to the control function. This is the base PWM value/throttle command before the three control terms are applied to correct the error in the system.

To tune your PID, set the parameters ( ) in *z_pid.yaml*.

To test your PID, run `python sim.py` on your base station or a department computer but not on your drone, since it requires a graphical user interface to visualize the output. The PID class in *student_pid_class.py* will automatically be used to control the simulated drone. The *up* and *down* arrow keys will change the setpoint, and *r* resets the simulation.

You will need *numpy*, *matplotlib*, and *yaml* to run the simulation. To install these dependencies, run `pip install numpy matplotlib pyyaml`.

Write brief answers to all exercises in *answers_pid.md*.

Problem 1: Implement an Idealized PID

Modified 2019-10-29 by andrewkpeterson

Exercises

1. Implement the `step` method to return the constant . At what value of does the drone takeoff? Set to 1300 for the remainder of the questions.
2. Implement the P term. What happens when is 50? 500? 5000?

3. Implement the D term. Set to zero. What happens when is 50? 500? 5000?
4. Now tune and so that the drone comes to a steady hover.

   Describe the trade-off as you change the ratio of to .
5. Implement the I term and observe the difference between PD and PID control. What role does the I term play in this system? What

   happens when and are set to zero?
6. Implement the `reset` method and test its behavior. If implemented incorrectly, what problems can you anticipate reset causing?
7. Finally, tune the constants in your PID controller to the best of your abilities. When the setpoint is moving, the drone should chase the setpoint very closely. When the setpoint is still, the drone should converge exactly at the setpoint and not oscillate. Report your tuning values.

Problem 2: Tuning a PID with Latency

Modified 2018-09-01 by Sophie Yang

Now, we introduce latency! Run the simulation as `python sim.py -l 6` to introduce 24 milliseconds of latency (six steps of latency running at 25 hz).

Exercises

1. Tune the constants in your PID controller to the best of your abilities. The drone should chase the setpoint very closely, but will converge more slowly when the setpoint is still. Report your tuning values.

2. Compare your tuning values to the values you obtained in problem 1.
3. Explain the effect of latency on each control term.

Problem 3: Tuning a PID with Latency, Noise, and Drag

Modified 2018-09-01 by Sophie Yang

In the most realistic mode, you will tune a controller with latency, noise, and a drag coefficient. You can do this with the command line arguments `python sim.py -l 3 -n 0.5 -d 0.02` to be most realistic to real-world flight.

Exercises

1. Tune with these arguments to be as good as possible. Report your tuning values.
2. Compare your tuning values to the values from problems 1 and 2.

   Run `python sim.py -h` to see the other simulator parameters. We encourage you to experiment with those and observe their effects on your controller.

   After you finish this part of the project, make sure that you push the final versions of the files that you modified to your Github repo.