Introduction to Feedback Control Systems 16-299 - The Robotics Institute

Homework 1: Feedack Controllers

Prof. George Kantor

Due: 18 February 2021

1 Introduction

This homework is designed to help you master the intuition behind feedback controls, in particular the proportional, derivative, and integral components that make the PID controller. After this homework, you should be able to tune a controller for most systems.

You will design a controller for pendulum rotating about a vertical axis; however, it is analogous to the point mass and mass-spring systems (as modeled by a bead on a wire and a bead constrained by a spring) that were presented in class. The problem set starts with proportional control and proportional-derivative control applied to a spring-less system, and, culminates in a proportional-derivative-integral control for a spring-loaded system.

2 Questions

In the provided zip file, you will complete four sections marked `TODO` in the `controllers.py`. Assuming you have installed Mujoco as instructed in the previously supplied guide, you can run controllers on a pendulum as follows:

```
python3 controller.py --sim_type TYPE --controller CONTROLLER
    TYPE: one_link, spring
    CONTROLLER: open, p, pd, pid (default is random)

other options:
        --plot_torque 1      # plots the torque signal (disabled by default)
        --plot 0             # disables position & velocity plotting
        --sim_length 10000   # set the simulation duration (default is 5000)
```

Use the plot cursor & zoom features to read the plot values.

2.1 Mass System

1. **Open Loop Control**: this controller requires a step up then down in the applied torque τ. The torque should last for a duration d (the provided code has a variable `step` that counts the simulation steps). In the `TODO` section of the `open_loop(...)` method, the logic for an pulse input has been written for you, set a duration of 50, and input of 8.5. Report the settling time, maximum percentage overshoot, and rise time, and submit the position-time plot.

2. **Low Gain Proportional Control**: this is the simplest feedback controller (note the previous problem did not use feedback). For this `TODO` (in the `prop_controller(...)` method), the error is the difference between the `reference` (also known as the *desired outcome*) and the current angle of the pendulum.

The control signal is proportional to the error by a coefficient K_p. With $K_p = 7.0$, report rise time, settling time, and the maximum percentage overshoot[1][2][3]. Submit the position-time plot.

3. **High Gain Proportional Control**: adjust the K_p value until you achieve a rise time of 175. Report settling time and the maximum percentage overshoot. Submit the position-time plot.

4. **Low Derivative Gain Controller**: Since the proportional controller is ringing (i.e., not settling quickly, a sign of too much energy in the system), we need to introduce a derivative controller. This controller is of the form $u = K_p * e + K_d * \frac{de}{dt}$. In `pd_controller(...)`, set $K_p = 10.0$ and $K_d = 0.75$. Submit the position-time plot, and report the report rise time, settling time, and the maximum percentage overshoot.
Note: $\frac{de}{dt}$ can be expressed as the difference between the reference velocity and the velocity of the system.

5. **High Derivative Gain Controller**: To demonstrate the derivative controller's effectiveness in dissipating energy, increase K_p gain until you achieve a settling time of 750 or better. Submit the position-time plot.

2.2 Mass-Spring System

In the previous section, when the error is eliminated, and the system is at rest, the input u is zero since the error and its time-derivative are also zero. Imagine if the system has an external force (say, due to gravity or an electric field), then the PD controller will no longer suffice. The final controller can be written as

$$u(t) = K_p * e(t) + K_d * \frac{de}{dt} + K_i * \int_0^t e(\tau)\, d\tau$$

If any residual error remains, the integral controller will progressively increase until $u(t)$ is equal to the external force, and the steady-state error is zero.

1. **Open Loop Control**: apply a pulse controller with a duration of 50 and input of 100. What is the steady-state output?

2. **PID Control**: set $K_p = 8.0$ and $K_d = 1.75$

   (a) apply $K_i = 0.0$. What type of a controller is this? And what is the steady-state error? Submit the position-time plot.

   (b) apply a low $K_i = 0.003$ and observe the steady-state error slowly get eliminated.

   ```
   ./controller.py --controller pid --sim_type spring --sim_length 12000
   ```

   Submit the position-time plot.

   (c) **PID Control**: tune your K_i until you achieve a maximum overshoot (not percentage overshoot) less than 2.0, and the settling time less than 2000. Submit the position-time plot.

---

3 What to Submit

Your submission should consist of just one pdf file, submitted via Canvas. For each controller, submit a position-time plot that shows the settling dynamics.

[1] `https://en.wikipedia.org/wiki/PID_controller`
[2] `https://youtu.be/zGp-_qJ58kQ?t=431`
[3] `https://ctms.engin.umich.edu/CTMS/index.php?aux=Home`