cpsc 543 Lab 4: Controlled Actuation using the PID

The PID (Proportional + Integral + Derivative) controller is the most garden-variety method of making a DC motor (Torque = Kt * Current) follow a command, whether the command is a desired position trajectory θ(t), or produced by a haptic virtual environment. Let's talk about the position trajectory. The core problem here is that while you would like to make your DC motor move to a desired position, you only have control over its torque output[1]. If you additionally have an estimate of its angular position, then you can used closed-loop control to drive the measured position to the desired position using torque commands (send torque until error=0). If you do this using just simple proportional control [command = a constant * (measured-actual position] it will usually be unstable. PID is the solution to this.

Learning Goals:

By the end of this lab, you will be able to:

- State several reasons why (and when) controlled actuation could be useful
- Define the core operating principles of a PID controller (defining equations, and the impact of each component)
- Tune a PID controller to be stable and effective
- Describe and fix control issues like uniform updates, integrator windup, delay
- Construct a simple Arduino-driven haptic display consisting of a motor with a handle and position sensing, using the Twiddlerino/Arduino setup supplied in class
- Write a simple Arduino program that drives your motor to implement some simple haptic environments, stably

Setup:

This assignment uses some skeleton code to get you started. It's written for Arduino, which you already know, and Processing, a similar programming environment for writing graphical sketches on your laptop or desktop computer. The Processing sketch communicates with Arduino over the serial port, letting you use your laptop to control your Arduino.

You will need:

- Arduino: http://arduino.cc/en/Main/Software
- Processing: http://processing.org/download/
- PIDLibrary v1.1.1 for **Arduino**. Either download from: http://playground.arduino.cc/Code/PIDLibrary and extract it to your Documents/Arduino/libraries folder to install, or go to Sketch > Include Libraries > Manage Libraries, search for PID and click install.

[1] Why do we use DC motors, when a stepper motor lets you control position directly? It's not nice to feel (steppy!); but more importantly, what we need more generally for force feedback is a *relationship* between the impedance supplied by the human and the motor pushing back, not a position trajectory. A stepper motor can't do this.

- Twiddlerino library for **Arduino**. Download from:
  http://www.cs.ubc.ca/~cs543/2015W2/del/Twiddlerino.zip
  and extract it to your Documents/Arduino/libraries folder to install.
- ControlP5 v2.2.5, a UI library for **Processing**. Either download from:
  http://www.sojamo.de/libraries/controlP5/
  and extract it to your Documents/Processing/libraries folder to install, or go to Sketch > Import Library > Add Libraries, search for controlP5 and click install.
- The Lab4 skeleton code (Arduino sketch) from:
  http://www.cs.ubc.ca/~cs543/2015W2/del/lab4-PID-codeLibraries.zip

To run, upload the Arduino sketch (.ino) to your Twiddler (**change the board to be Duemilanove**). Once it's running, **change the serialPortName** in the Processing sketch (.pde).

The Assignment:

The skeleton code includes an implemented PID controller that only uses the P part (also known, unsurprisingly, as a P controller). This is analogous to a linear spring, where P is the spring constant. The output of the motor is proportional to the error of the motor's current position from the target position. Your assignment is to first get familiar with the PID, then employ it to build a virtual wall, then experiment with sampling rate.

The supplied sketch implements a simple virtual model: a linear spring that moves the Twiddler to a desired "neutral" position. **These are the items you'll need to accomplish and reflect on:**

1. Run the code and try out the P controller.
   How does it feel? What happens when you change the target position? What does changing the P parameter do? Do you notice any problems?
2. Add the D component to your controller.
   Tip: you'll need to modify a parameter in Lab4Arduino.ino, and work it up/down to find an effective value.
   How does this change the behaviour of the Twiddlerino? Are there any problems?
3. Add the I component to your controller.
   How does this change the behaviour of the Twiddlerino? Can you create a stable system that reaches the target?
4. Implement a virtual wall, as a one-sided spring (like Step 1) but active only for $\theta>0$.)
   What happens? By adding/tweaking PID parameters, can you make it more wall-like?
5. Play with the controller update rates, and with introducing delays.
   Hint: use the PID's SetSamplingRate() command. *How does this change the system? What happens if you sample faster or slower? Can you make the interrupt service routine run faster, and see better performance that way*
6. Think about different ways of timing the loop (polling "run as fast as you can", vs. clocked "wake up at a specified time").
   Hint: Look at the following Arduino examples: Blink (01 Basics), BlinkWithoutDelay (02 Digital), and FlashLED (MsTimer2).
   What are the impact of each of these? Which method is implemented in the sample code? Can you make a theory about impact on the Twiddlerino sketch, and test it?

Suggested Steps

Preparation:

1) Read through the Arduino PID tutorial, including the individual "improvements" steps beginning with *sample time.* This will be a handy reference when doing the assignment.
2) Read the introduction of the Twiddlerino documentation (HardwareDocumentationV2.pdf, included with the example code).
3) Make sure you know how a PWM (pulse width modulated power amplifier) and an encoder (position sensor) work – see earlier course notes.
4) Bring your laptop to class.

At lab and on your own:

5) Get your Twiddlerino setup and the example code working. Spend some time to become familiar with what is happening, as you'll have to modify this code for your lab.
6) Do the assignment! Answer the questions as you go, so that you can put them on your final blog post.

The usual rules apply:

7) Work in pairs or individually.
8) Video/photo document your work and describe on your blog (every individual creates own documentation). IN YOUR BLOG, please proceed sequentially through each assignment question, clearly indicating which one you are addressing. You might find screenshots or photos/scans of hand sketches more useful than videos in this case; and words (succinct) may also be sufficient.
9) Post the blog link onto the course twiki: https://bugs.cs.ubc.ca/cgi-bin/twiki/view/CS543/

Resources

Find more resources on the course twiki.

Lab 4 Mark Sheet (completed by instructor)

Name: _____

Term: 2015/16 W2

- ☐ (15%) PID / Virtual spring implemented
- ☐ (15%) Virtual wall implemented
- ☐ (15%) Experiment with controller update rate modifications

- ☐ (40%) Reflection on above items
- ☐ (15%) Documentation adequate (visual and words)

- ☐ Multiplier applied for late hand-ins, as described on course homepage.

OVERALL MARK:

☐	**Great (100%)**	Entirely satisfied and exceptionally well done
☐	**Good (85%)**	Entirely satisfied and well done
☐	**Fair (70%)**	Largely satisfied, few major issues
☐	**Poor (54%)**	Some worthwhile, comprehensible effort, with substantial issues
☐	**Zero (0%)**	Little to no real comprehensible effort; not handed in or otherwise unacceptable