Lab: Cruise Control

CSI 3305: Introduction to Computational Thinking

October 31, 2010

1 Introduction

Almost every area of life involves some aspect of **Control**, where some property or quantity must be regulated to remain within reasonable bounds. This regulation is exercised by varying any number of control variables or processes which, in turn, affect the property or quantity being regulated. Regulation typically involves a form of feedback, where a sensing mechanism detects the level of the property or quantity, and activates variation in the control variables or processes. For example, if you exercise outdoors on a hot summer day your body will sense a rise in body temperature and stimulate the sweat glands to release fluid onto the skin. The fluid then evaporates, taking heat with it and cooling your body. If you forget your coat on a cold winter day your body may detect the drop in body temperature and stimulate your skin to produce goose bumps. The goose bumps raise the small hairs on your skin, which trap air against your body and keep it from carrying off your body heat, thereby helping to keep you warm. For another example consider look up and describe how so many aspects of the Earth's environment are regulated with feedback and control in order to be appropriate for life (oxygen, etc.). There are countless examples in the realm of Mechanical Engineering where control is required. Consider how a toilet bowl fills the tank only up to a certain level, then the float detects this level and shuts off the water valve. The thermostat contains a thermometer that detects temperature, and if the temperature gets too cold turns on the heater, or if it gets too hot turns on the air conditioner. Gasoline engines, such as you may see on your lawnmower, have a governor that detects the rotational speed of the engine, and if it gets too fast reduces the flow of air and gas to the engine, or if it gets too slow increases it.

2 Problem Statement

The problem we will address is that of the cruise control system (CCS) of an automobile. Most every automobile these days is equipped with a CCS that maintains the automobile at a certain speed designated by the driver. The driver drives the car up to the speed that he wants to maintain, and then hits the "Set" button on the cruise control. The CCS takes over and the driver may remove his foot from the accelerator. The CCS monitors the automobile speed through the speedometer, and adjusts the accelerator to maintain the target speed. If the car begins slowing down as it goes up a hill, the CCS will increase the accelerator. Or, if the car begins speeding up as it goes down a big hill, the CCS will decrease the accelerator. The aim of the CCS is to maintain the automobile at the target speed no matter what else may be impacting the automobiles performance. If the driver wants to regain control of the cars accelerator, he may tap the brakes or turn the CCS off. In this assignment you will design and modify a cruise control system for a simulated automobile driving on a simulated roadway course with varying environmental conditions.

3 Background Reading

We will create a simple PID system to serves as the brains of our cruise control system. Please read the Wikipedia page on PID controllers to understand the theory behind them:

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

Do not worry if you can't follow all of the math, just try to understand what the three main components of a PID system are and how they work together to produce an output.

4 Tools

You will be using the Python programming language and Python interpreter for this lab, as well as the provided Car Simulator model. The code will be written using a text editor, and the interpreter will be called using the command line.

5 Setup and Programming

5.1 PID file

1. Download and save the files needed for the simulator from the course website. Unzip the file and save to a local folder.

2. Within that folder, create a new file called pid.py

3. Open the pid.py in the text editor of your choice (such as Notepad or Vim.)

4. Enter the following code at the beginning of the file on the first two lines:

   ```
   class Control:
       Kp, Ki, Kd = 0.5, 0.5, 0.5
   ```

   Notice the indentation on the second line. (Python is whitespace sensitive and you will get an error if the second line is not indented.) The first line creates a class called Control that will be the brains of our cruise control system. The second line defines three variables within that class, which are the parameters we will need to set for our particular system. (See Wikipedia page for details on these parameters.)

5. Next, we will create an initialization method for our class. Below the two lines entered above, enter the following code:

   ```
       def __init__(self, target, current):
           self._previousError = 0.0
           self._target = target
           self._error = target - current
           self._integral = 0.0
   ```

   Again, notice the indentation. (The line beginning with "def" should be indented, as well as the lines beneath that one.)

This method initializes the private variables we will be using in our PID controller, from a user supplied target value and current value. (In our case, this is our target speed and current speed.)

6. Next, we create methods for setting our parameters and for setting our target:

```
def setParameters(self, Kp, Ki, Kd):
    self.Kp = Kp
    self.Ki = Ki
    self.Kd = Kd

def setTarget(self, target):
    self._target = target
```

7. After creating the above methods, we will create the method that does the actual work for our controller. Add the following update method:

```
def update(self, current):
    self._error = self._target - current
    self._integral += self._error
    derivative = (self._error - self._previousError)
    self._previousError = self._error
    return (self.Kp * self._error) + \
           (self.Ki * self._integral) + \
           (self.Kd * derivative)
```

This method takes in the current value, calculates the error based on the target value, then calculates the three components of our PID calculation. It then sums those components, weighted by the parameters K_p, K_i and K_d, and returns this value. It also updates the previous error variable, for the next calculation.

8. After this, add one last method, useful for checking the error:

```
def error(self, current):
    self._error = self._target - current
    return self._error
```

9. Save the file, and close it.

5.2 Testing the Controller

After creating our PID controller, we want to test it and find parameters that allow it to work best. To do so, we will create another file, called tester.py, that will allow us to test our controller.

1. Within the same folder, create a new file called tester.py
2. Open the tester.py in the text editor of your choice (such as Notepad or Vim.)
3. Enter the following code at the beginning of the file:

```
from math import fabs
from pid import *
```

These lines import a math function allowing us to calculate the absolute value and the PID controller class we just created.

4. Next we will create local variables to keep track of our speed and acceleration. Enter the following code:

```
speed = 100
acceleration = 0.0
```

5. We next create a controller object, which we will use to adjust our speed. Enter the following lines below what is already in your file:

```
c = Control(target = 60, current = speed)
c.setParameters(0.5, 0.5, 0.5)
```

The c variable will refer to our PID controller, which we set with a target speed of 60 and a current speed equal to our speed variable. Next, we need to set the K parameters for our controller. Your job is to find values that allow the system to quickly zoom in on the correct value. You will do this through trial and error, after adding the rest of the code below and running this file.

6. The next step is to add a loop that will call your controller and update the acceleration value based on the output of your PID controller. Enter the following code:

```
while fabs(c.error(speed)) > 0.01:
    print c.error(speed), speed
    acceleration = c.update(speed)
    speed += acceleration

print c.error(speed), speed
```

While the error is greater than 0.01, the loop will iterate. First, it prints to the screen the current error as well as the current speed. Next, it gets a new acceleration value from the PID controller, by calling its update method, passing in the current speed. It then uses this acceleration to update the current speed. Finally, after the loop is exited, the final values are printed out.

7. The tester.py file should have the following final form:

```
from math import fabs
from pid import *

speed = 100
acceleration = 0.0

c = Control(target = 60, current = speed)
c.setParameters(0.1, 0.1, 0.1)

while fabs(c.error(speed)) > 0.01:
    print c.error(speed), speed
    acceleration = c.update(speed)
    speed += acceleration

print c.error(speed), speed
```

8. Save and close this file.
9. Open the command line (From the Windows menu bar: Start > Run... > cmd). Navigate to the folder where you created your file (use the following command: cd "C:\Folder\where file\is".)
10. From the command line, type the following to run your program and hit enter:

    ```
    python tester.py
    ```

    You should see a stream of output printed to your screen. Notice that the error should decrease, until your speed is very close to the target speed.
11. Open and edit your tester.py file to find parameter values that minimize the number of steps it takes to settle on the target. When you feel you have good values (it converges in ten or less steps), move on to the next section.

5.3 Cruise Control Simulator

We will now use the PID controller we created, with the parameter values you found through experimentation, to control a cruise control system.

1. Within the same folder, create a new file called cruise_control.py
2. Open the cruise_control.py in the text editor of your choice (such as Notepad or Vim.)
3. Enter the following code and save the file:

    ```
    from simulator.simulator import CarSimulator
    from pid import *

    acceleration = 0.0
    speed = 0.0
    cs = CarSimulator()

    pid = Control(target = 60, current = speed)
    pid.setParameters(0.5, 0.5, 0.5)

    while 1:
        speed = cs.update(acceleration)
        acceleration = pid.update(speed)
    ```

    Again, pay attention to indentation.

    The first line imports the pre-built car simulator class, and the second imports our PID controller class. We next set our acceleration and speed variables, as well as creating a CarSimulator object, called cs. Next we create an instance of our PID class, setting the target speed at 60 and passing in the current speed.

    Using the parameter values you found through experimentation, replace the "0.5, 0.5, 0.5" values with your own, such as:

    ```
    pid.setParameters(0.3, 2, 1.5)
    ```

After this, we have a simple while loop that passes the acceleration to our car simulator, which uses this acceleration to control the car speed. It returns the updated speed of the car (taking into account road conditions, such as wind drag and any hills in the road which may affect speed.) We then pass this new speed to our PID controller, in order to calculate what the next acceleration adjustment should be, repeating the process.

4. Save and close this file.

5. From the command line, type the following to run your program and hit enter:

```
python cruise_control.py
```

   You should see a window open up, with a car driving along a road. If your controller is working properly, the car's speed should remain around 60 mile per hour, regardless of what the road conditions are. There may be some slight oscillation around the 60 mph mark, but you should not see wild oscillations in speed. If you do, then you need to adjust your parameters until you see a nice, controlled, fairly constant speed on your vehicle.

6 Questions

1. Briefly explain what a PID controller is and why we would want to use one for controlling a cruise control system.

2. What are the three main components of a PID controller and what do they each do?

3. What parameter values did you find that allowed for best control of the car speed?

4. Did you see oscillations caused by your controller? What do oscillations mean, from a control point of view? (In other words, what causes them?)