CS148
Building Intelligent Robots

Week 4 : PID Control

Out: 4 Oct 2005

Preliminary Tasks

before 6 Oct 2005, 9am

The purpose of this week's lab and project is to implement a simple but powerful control system called a PID controller. Your task will be to create a wall following robot.

PID Review

PID stands for **P**roportional **I**ntegral **D**erivative control (after the formula that uses these components to calculate the control parameter.) It is used in a wide variety of real life situations - cruise control in cars is an example that comes to mind. When talking about PID control, there are some terms we need to define. The *set point* is defined as the goal of the system or the position where you *want* your system to be, this is denoted as $s(t)$, the set point at time t. For cruise control, it would be the desired speed you want. The *process variable* is the position where your system currently is (denoted as $v(t)$), the process variable at time t. The difference between the set point (goal) and process variable is your *error* (denoted as $e(t)$), the error at time t. Ideally your controller should force the error to zero.

The basic idea is that the output of a PID controller is, in theory a linear combination of the error, the integral of the error, and the derivative of the error. In practice, however, we usually take the derivative of the process variable instead. We do this because the derivative of the difference between two variables is the same as the sum of the derivatives, and the derivative of a constant ($s(t)$ in this case) is zero.

The PID controller attempts to control the process variable (current state of the system) according to the formula:

$$\text{output}(t) = K_{proportional}\left[e(t) + \frac{1}{K_{integral}}\int_0^t e(t)\,\mathrm{d}t - \frac{1}{K_{derivative}}\frac{\mathrm{d}}{\mathrm{d}t}v(t)\right]$$

The term output(t) is the output applied to your system at time t. The constants K are used to weigh the sum of the three terms to produce an output that steadily drives the error to 0.

The tricky part of PID control is *tuning* these constants appropriately so they eventually eliminate the error. The output is applied to the process variable, hopefully correcting the error. In the case of cruise control, the output would be applied to the motors to either speed up or slow down the car. The proportional part of the formula ($e(t)$) helps to correct the error by applying an output that is proportional to the amount of error but does not reduce the *steady-state error*, that is the error that has accumulated over time. That is why we have the part of the equation, $(\frac{1}{K_{integral}}\int_0^t e(t))$, which sums (integrates) the error over time. The differential part, $(\frac{1}{K_{derivative}}\frac{\mathrm{d}}{\mathrm{d}t}v(t))$ is mainly there to tame down the overshoot (after proportional and integral correction), since the differential reacts to the rate of change of the process variable.

Lab 4: PID Control

6 Oct 2005

Task:

- This week's project is to build a wallfollowing robot. You must use PID control (actually just the proportional and derivative parts) in your wallfollowing algorithm. You will want to implement the proportional part first before adding the derivative part to your controller. The motors should react depending on whether the sonar sensor detects that the robot is going too close or moving away from the wall.

  When finished your robot should be able to

  - Move parallel along a wall after some error correcting (PID control)
  - Be able to turn corners and keep following the wall.

Some things to think about before doing this part of the lab:

Sonar Sensor. Consider the positive and negative sonar values and what they indicate about the robot's position relative to the wall.

As a further adjustment you may now have to tweak the proportional constant. Now why might you need to tweak the proportional constant? Consider the different scenarios shown in Appendix A. (The horizontal line labeled SP is the *set point* (target)).

In the first figure the velocity fluctuates wildly for a long time before reaching the set point. This is clearly not desirable. In the second figure the velocity does not fluctuate that wildly but it takes a long time to settle down and in the third figure, the velocity fluctuates wildly but settles in a short time. You want your robot to display, ideally, behavior which could cause it to quickly reach the set point without much deviation. Practically it will probably be somewhere between figure 2 and figure 3. The way to manipulate the graphs is by adjusting the proportionality constant. You can narrow your search down to the perfect proportionality constant by doing a kind of a binary search. Start with a large extremes of values (a very large value and a very low value) and keep decreasing the difference between the values until the two values become the same. That will be your desired proportionality constant.

One way to see how long it takes to settle on the set point is to print out the process variable to the lcd of the RCX and see how long it takes to get to the set point. You can also display the error and see how long it takes to reach 0. Doing this will also show how the control sometimes corrects itself but then overshoots the goal.

Corners: You might want to use a bumper to detect when the robot has to turn a corner. For the other type of corner, you will know when you are at a corner when you have been traveling parallel along a wall and then suddenly your sonar readings change drastically.

PID: Calculating the PID values might seem a bit tricky as you might wonder how would you take the integral of the error or the derivative of the process variable. An easy way to approximate the integral of the error would be to keep a running sum of the error. This is essentially what taking the integral of the error does, it takes the sum of the error over a certain period of time. The derviative of the process variable measures the rate of change; you can calulate this by taking the difference between the current process variable and the last process variable. This will give you how much the process variable changed from the last measurement.