CS148
Building Intelligent Robots

Week 5 : PID Control *Out: 2 Mar 2004*

Preliminary Tasks *before 4 Mar 2004, 9am*

The purpose of this week's lab and project is implement a simple but powerful control system called a PID controller. You will be implmenting a cruise control for your robot in lab and then a wall following robot for your project. You should complete the follwing before coming to lab on Wednesday.

- Read *Robotic Explorations*, pp. 174–190 by Fred Martin
- Build a robust tankbot with treads/wheels with a motor controlling each tread/wheel. Attach a rotation sensor to each motor by using some gear ratio.
- Bring a ruler that can measure *cm*. (We won't have enough rulers for the entire class.)

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. You would like for your controller to force the error to zero.

The basic idea is that the output of a PID controller should be, in theory a linear combination of the error, the integral of the error, and the derivative of the error. But in practice, 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 and 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

In this week's lab you will implement a cruise control for your tankbot. You will need to measure the velocity of your tankbot by using the rotation sensors and then adjust the robot's motor power accordingly.

0.1

We first need to find a goal or *set point* for our robot. In this case, it will be some set velocity. We will measure the speed of our tankbot by measuring it in *clicks/sec.* We can measure this by using our rotation sensors that are attached to the motors of the tankbot.

a) We will first need to calculate the distance traveled for one rotation of the wheel shaft. You will need to measure how much distance the robot travels—you can assume no slipping and use the formula $2\pi r$ where r is the radius of the wheel, thereby figuring out what distance your robot travels for a single rotation of the wheel shaft. Then you need to figure out your gear ratio between the wheel shaft and the rotation sensor. From here, you can use simple arithmetic to calculate how many clicks of the rotation sensor occurred for one rotation of the drive shaft. This will give you the number of clicks of the rotation sensor for one rotation of the wheel shaft in *clicks/cm.*

b) Now we must figure out what velocity the robot can maintain. The best way to determine this is to make a trial run of the robot. Let the robot travel a distance of say a meter (100 cm) and record the time it takes. This will be the velocity the robot can maintin which is measured in *cm/sec.* Remember to set the motor at a medium power level. You do not want the power level to be too high because then the robot will not be able to increase the power level to maintain the same velocity when challenged with a ramp.

c) Now take the value from part *b* and mulitply it by your answer from part *a.* This will give you your desired set point in the appropriate units *clicks/sec.*

0.2

You are given a stencil to write your cruise control system in. Copy it into your directory.

```
cp l:/asgn/pid/* .
```

The code given is a basic skeleton for writing your cruise control for your robot. Pay particular attention to the PID portion of the code (actually just the proportional part), where it calculates the output for the motor. Replace the `TARGET_VELOCITY` with your set point that you got in step 1.

0.3

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

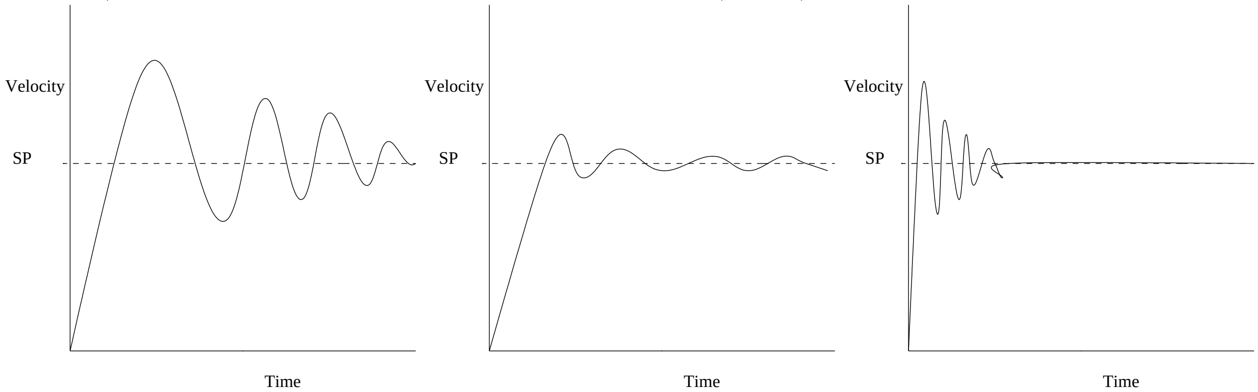

Figure 1 Figure 2 Figure 3

In the first figure the velocity fluctuates wildly for a long time before setting on to 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. We 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 till the two extreme 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 to the lcd of the RCX the process variable and see how long it takes to get to the set point. You can also display the error and see how long it reaches 0. Doing this will also give you some idea how the control sometimes corrects itself but then overshoots the goal.

Task

Make a cruise control system for your robot. It should maintain the same velocity when going up a ramp as it would traveling along a flat surface. If you have additional time you can add an integral controller to the cruise control system which will control the difference in speed between the left and right motors. You can try to make the robot perform turns.

Project 5: PID Control *due 11 Mar 2004, in class*

Specification: This week's project is to build a wallfollowing robot. You will need to build something similar to the bend sensor described in the Martin book (p. 176). The bend sensor should be geared to your rotation sensor so that it can measure the distance the robot is from the wall effectively. 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. For extra credit you can also implment the integral part to your controller. The motors should react depending on whether the bend sensor detects that the robot is going too close or moving away from the wall. Support code for the wallfollower can be found at

`/course/cs148/asgn/pid/wallfollower.c`

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 starting this project:

Bend Sensor. You will need to construct a bend sensor that performs the same functionality as the bend sensor in the Martin book. The rotation sensor should be attached to your bend sensor to measure how far away your robot is from the wall. The closer the robot is to the wall, the more "bent" the bend sensor should get and thus more clicks should occur on the rotation sensor. Thus you will need to gear up your bend sensor such that the rotation sensor will be able to detect small changes to the bend sensor.

Two suggestions for building the bend sensor for PID control. The first way is with one bend sensor. Your set point or goal will be a predefined distance away from the wall. Your error will be the set point minus the current status of the bend sensor. This is pretty simple except for some drawbacks. It will be hard for the robot to detect corners and deal with them effectively. Also it will be difficult to tell when the robot is moving into the wall or away from the wall. The other way for building the bend sensor for PID control will be to use two bend sensors. The process variable or the current state of the sensors will be their difference. Thus the set point will be 0. When the error is zero (error = set point − process variable = 0 − difference between the two bend sensors), then the robot is parallel to the wall. If one of the sensors is bent more than the other, then you can figure out if the robot is heading towards or away from the wall. An example of the bend sensor will be on the website.

Corners You might want to use a bumper to detect when the robot has to turn a corner. For the other type of corner, if you are using two bend sensors, you will know when you are at a corner when you have been travelling parallel along a wall and then suddenly your front bend sensor detects nothing but your rear bend sensor is still detecting the wall.

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, we can calulate this by taking the difference between the current process variable and the last process variable. This will give us how much the process variable changed from our last measurement.

Paper Handin: Your paper hand-in should include a description and drawings of key design elements of your robot such as your bend sensor and the gear ratio between the bend sensor and the rotation sensor. You should explain your PID algorithm including how you dealt with both types of corners. Explain how you went about figuring what proportional and derviative constants to use. Did you observe any notiable difference between your robot using just the proportional constant and using the proportional and the derivative constant? If yes, how did the derivative controller help correct the error. Finally, describe any major failed attempts (bad bend sensor design, bad gear ratio, etc). If your first design was perfect, please state so!

Finally please handin your code **/course/cs148/bin/cs148_handin pid**. One copy of your code will be sufficient for the group

Grading: Both your lab write-up and your actual robot will determine your grade. Your robot will be scored in the following manner:

Follows wall using PID control	25%
Deals with corners effectively	15%
Bend Sensor Design	5%
Overall technical Design	5%
Total	**50%**

Your paper hand-in will be graded as follows

Design explanations	18%
PID algorithm	18%
Descript of determining PID constants	8%
Drawings	4%
Failed attempts	2%
Total	**50%**