Introduction to PID Control

Version 1.0

May 13, 2013

Steven Lao
steven.lao@team3161.ca

Nathan Woodger
nathan.woodger@team3161.ca

www.team3161.ca

Abstract

This paper provides the general idea behind a PID controller and is to be used as a starting point. It is meant to be very brief and easy to read. Some fundamentals of derivatives and integrals are also presented in the Appendix.

PID Control

A PID controller is a simple way of adding intelligence to a system by using feedback from sensors to control the output. The typical feedback layout is depicted in Figure 1. You have an input command which is fed to the controller. The controller decides how much to excite the plant, which produces some output. The output is measured by sensory equipment which is also fed to the controller.

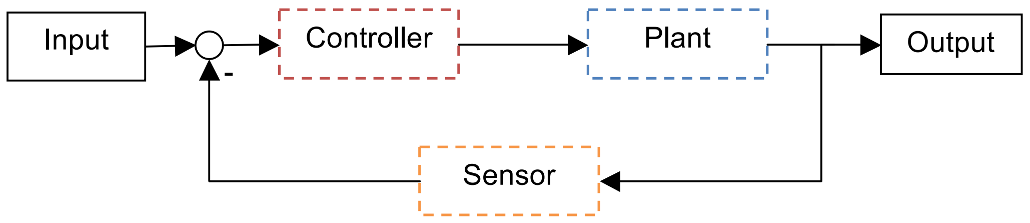

Figure 1: Typical layout of a feedback controller.

Why do we care?

Open loop control cannot handle any external changes that may occur due to sudden disturbances or changes that occur overtime (eg. wear). In other words, it is not a very robust control method.

NOTE: Teleop control can be considered closed loop because the driver acts as the controller to counteract any disturbances that they see.

How do we do it?

The idea behind the PID controller is quite easy to grasp, but it can be tricky to implement. There are three components to the controller: Proportional, Integral and Derivative.

Output = P + I + D

See Appendix A for a brief overview of derivatives and integrals.

Proportional

The proportional is the easiest feedback control method. It uses the amount of error multiplied by a gain to control the output. The greater the error, the greater the output.

P = P_constant * (*target value – current value*)

Derivative

The derivative tells you how fast you are reaching your target. This will limit the overshoot from your target by slowing down if you are attacking your target too fast. Basically, you have to balance the effect of the P-gain with the D-gain. If you are getting very close to your target, the Proportional action will decrease and the Derivative action will make a larger effect. But when you are still very far away from your target, you don't want the D portion to take effect, it's only when you get very close to the target that you start to slow down. Thus, when you have a large error (large P), the P will overpower D.

D = D_constant * (*current error – previous error*)

Here we've assumed that the program is running in discrete cycles with constant times in between, therefore we don't have to do any complex math to determine the real derivative (it gets clumped together with the constant).

NOTE: Watch out for the sign in D. Some people calculate D as *previous error – current error* and then use Output = P + I - D. The effect of D should always counteract the actuator in the system to dampen it (think of it almost as friction).

Integral

The integral is a total of the error (integral = area under the curve). Therefore, the longer it has taken to reach the target the more power we will output. The purpose of the integral action is mainly to get rid of small steady-state errors that PD control cannot achieve. It will also add more power if the system encounters any resistance.

I = I_constant * *sum of errors*

If your target is really far away, this can overpower and crush all the other constants, so you have to be careful with this one.

NOTE: The integral portion is not always necessary, a PD loop is pretty good if you want to keep it simple. In this case, I_constant can be set to zero.

The integral portion is usually the hardest to get right because it difficult to find the right range for your constant. If it's wrong, the system can go unstable very quickly. A few tricks you can add to the integral control to help keep the system stable:

-Reset the sum of errors whenever the sign of the error changes
-Cap the contribution to the sum of errors for each iteration (avoids integral windup)
-Stop adding to the sum of errors when the error is within a certain tolerance of the target

Each of these tricks have different effects and not all of them will be applicable in every case. So the best way to learn is to experiment with them.

Tuning

Arguably, the most difficult aspect of PID is tuning the controller. This is the step where you have to translate some sensor input into a useable value for your actuator. This is done by changing (tuning) the PID constants. Every system will have different constants due to their different characteristics. It is possible to model a system and optimize the gains. But in many cases in industry, it is tuned through trial and error by someone with a lot of intuition and experience with the system.

The three parameters that must be tuned are the P_constant, I_constant and D_constant. A suggested method for tuning the parameters for position tracking is presented below.

Common tuning approach:

A methodical way to tune the PID values is to:

- Start with P-gain small and I-gain, D-gain both zero.
- Raise P-gain until the robot is oscillating consistently around the target.
- Once this is accomplished, start increasing D-gain until the robot stops oscillating.
- Then add I-gain until the robot stops within a desired range of the target.

Driving Straight

Often times there are inefficiencies in the drivetrain that cause a robot to drift toward the left or right due to one side moving faster than the other. A gyro can be used to help correct the drift. If you have a skid steer setup with PID individually controlling the left and right side of the drivetrain you can add a correction factor.

Output_Left = left_PID_out + gyro_correction_out
Output_Right = right_PID_out – gyro_correction_out

To do this, start by creating a PID loop for controlling the angle of the robot. The input would be the gyro reading and the output would be some motor value to send to the motors (one side would be positive and the other side is negative). Tune this PID to control the angular position of the robot. If tuned correctly, the gyro PID should make the robot hold its angular position when an input of zero is commanded, but you should still be able to push the robot back and forth if you push it straight. Now add and tune two PID loops to individually control the left and right side of the drivetrain with feedback from their encoders.

References and More Information:

[1] T. Wescott. (2000, Oct.). “PID without a PhD.” *EE Times*. [Online]. Available: http://igor.chudov.com/manuals/Servo-Tuning/PID-without-a-PhD.pdf

-Very good paper

[2] J. Sluka. (2009). *PID Controller for Lego Mindstorms Robots*. [Online]. Available: http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html

-Easy to understand application of PID in Lego robots

[3] Simbotics. *Textbook For Success*. [Online]. Available: http://www.simbotics.org/resources/workshops

-Good resource and includes source code of C++ implementation

[4] J. Russell. (2008). *PID for velocity control*. [Online]. Available: http://www.chiefdelphi.com/forums/showpost.php?p=690837&postcount=13

-I haven't had much success with this one, but describes a modification for speed control

Appendix A: Introduction to Derivatives and Integrals

NOTE: I won't be going too far in depth with derivatives and integrals. This is merely to give you some exposure and a slight understanding of the math behind what we are doing. We will just approximate the derivate and integral with numerical methods (particularly using finite difference and rectangle rule).

Derivative

The derivative is a measure of the rate of change of a function. We often take derivatives to find the rate of change of displacement of an object to find its velocity (Figure 2 and 3). Let's say f(x) is a function that denotes position, the derivative of f(x) gives us the velocity. The derivative of a function can be denoted as f'(x).

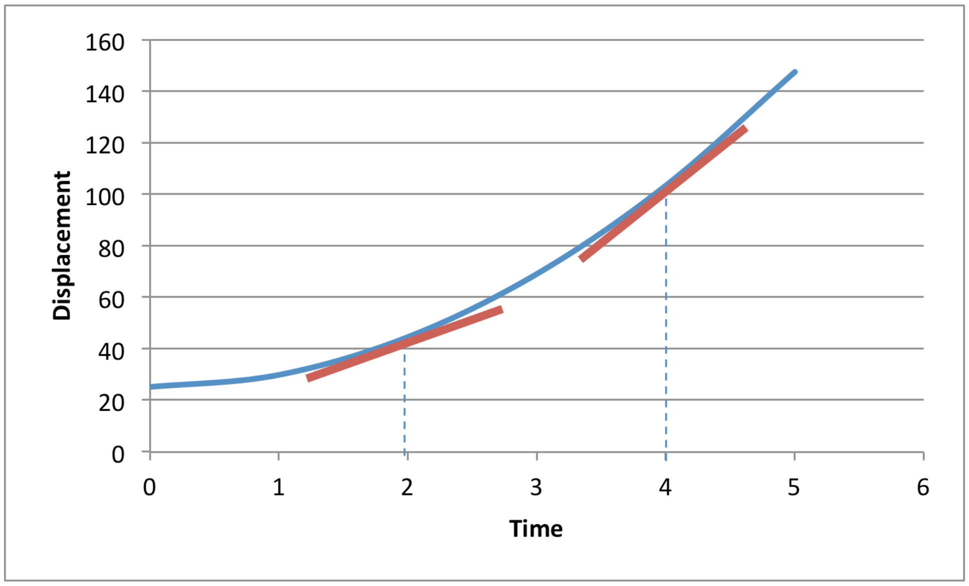

Figure 2: The blue function represents the displacement of an object at a given time. The red lines are the tangent of the function at given times, whose slope represents the instantaneous rate of change.

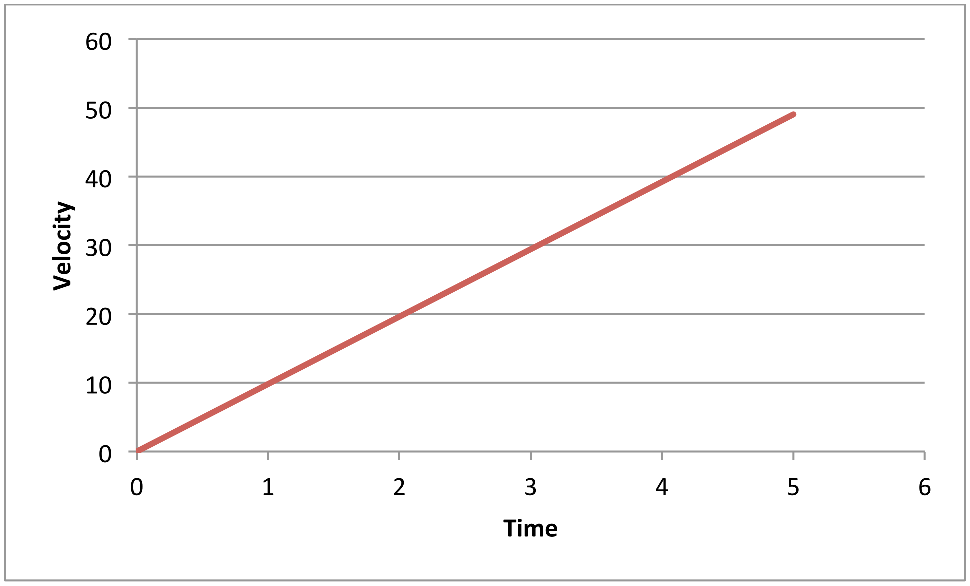

Figure 3: The red function represents the velocity of the object.

Sanity check: To find the slope of the tangent line, we can do rise/run (change in y divided by change in x). The dimension for y is a distance (ie. metres) and the dimension for x is time (ie. s), so when we find the slope the dimension we end up with is distance/time (ie. m/s).

There are two simple ways we can use to approximate the derivative: draw a tangent line or use finite difference approximation. The first method is pretty simple to do by hand on a piece of paper, but not so much in code. So let's examine the finite difference approach.

The basic principle of a derivative is to find the slope between two points – two points that are infinitesimally close together. The equation for a derivative can be found as follows:

$$f'(x) = \lim_{h \to 0} \frac{f(x) - f(x-h)}{h}$$

where f'(x) is the derivative of f(x), and h is difference in x between two points. Since the exact velocity is not necessary, we can relax the limit condition and just use a relatively small difference in x. If we use a constant value, this basically is a finite difference approximation.

Applying this theory to the robot, if f(x) is the displacement and x is the time... The periodic code runs constantly at approximately 50 [Hz], or every 20 [ms]. This is a relatively small difference in x, which we can use. So if we just use the periodic function to find the difference between the current displacement and the previous displacement and divide by 20 [ms], we can find a pretty good approximation of the velocity.

Note: In practice, when we use this in PID control, we will skip dividing by 20 [ms] because we don't care about the exact value of velocity and we will always be multiplying the velocity by some constant (think of the divide by 20 as part of the constant).

Integral

The integral is the direct inverse of the derivative. It is often defined as the area under a curve. One way to do find the area under the curve is to split it up into rectangles as shown in Figure 4.

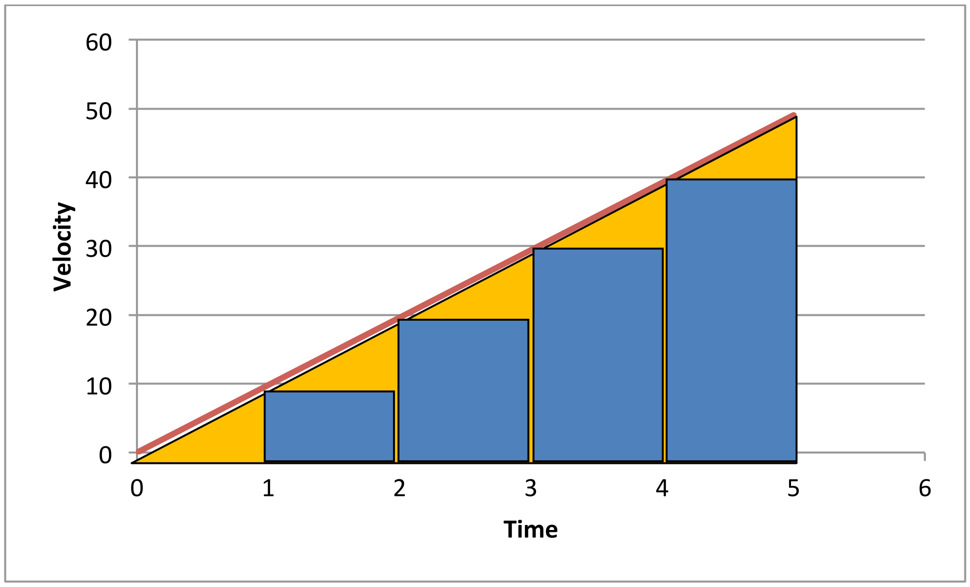

Figure 4: The yellow area represents the area under the curve that we want to find from the function in Figure 2. We approximate the integral of the function by adding up the areas of the large blue rectangles.

This approximation still has large errors as evidenced by the large areas still missing. Like we did for the derivative, we can take smaller divisions to be more accurate (see Figure 5). If we have infinitesimally small slices, we get the area under the curve.

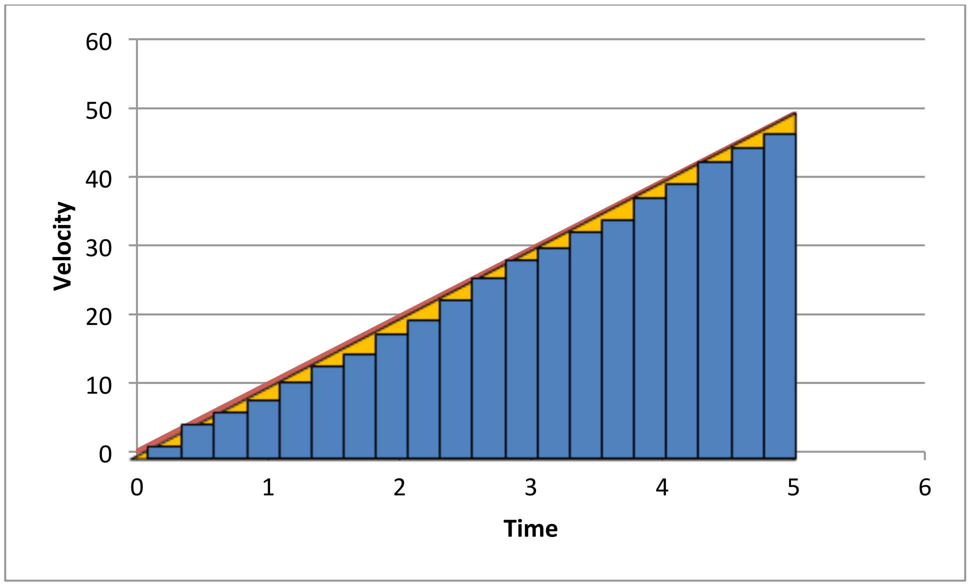

Figure 5: With smaller time slices, we get a more accurate depiction of the area under the curve; evidenced by the smaller area of yellow space.

However, the area under the curve is not the whole picture. The area under the curve only tells you a piece of the picture you need to find the integral. If you take the area under the curve, you find that we displaced 125 [m]. If you look at Figure 2, we didn't take into account the fact that at the starting time (t = 0) the displacement was already at 25 [m]. Therefore, the area under the curve only tells you the *relative* displacement. You need another piece of information that tells you your starting point to complete the integral.

Therefore, the integral can be approximated as:

$$F(x)|_a^b = \sum_{i=0}^{n} \left(f\left(a + \frac{b-a}{n} i\right) \frac{b-a}{n} \right) + F(a)$$

where F(x) is the integral evaluated between a and b, n is the number of divisions, and F(a) is the initial condition.

Note: When we implement the integration in PID, we greatly simplify it and just add the sum of errors. This is done because again we assume the program is running periodically every 20 [ms]. So the (b-a)/n = 20, which is absorbed as part of the constant and then we ignore the initial condition, so a = 0.

NOTE: We did not discuss functions in the 4th quadrant (negative range). In the end, you do the same process, just remember to carry the negative sign.

Putting it all together

You can see a relationship between displacement [m], velocity [m/s] and acceleration [m/s^2]. You take the derivative of displacement of a body to find its velocity. Then you take the derivate of velocity to find the acceleration. To go the other way, you take the integral.