PID Control

PID

Proportional, Integral, Derivative feedback loop

Gain

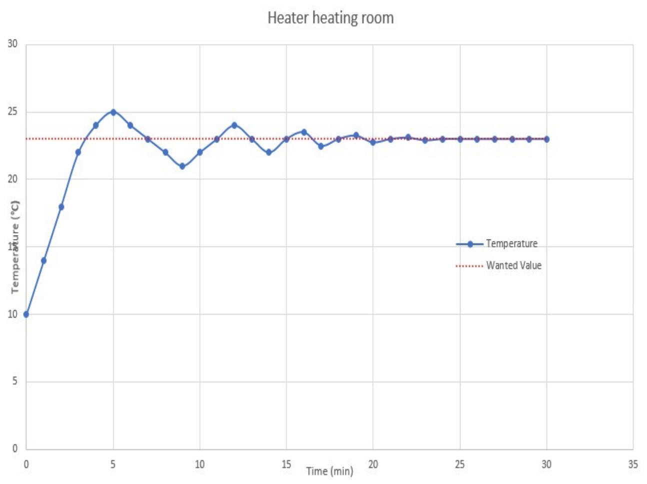
Amount of output given from each of the components of PID.

Feedforward

A known value supplied to the output as a guesstimate so the PID only has to make minor corrections.

PID Control lies at the heart of any advanced robotics motion. Essentially, it is a way of controlling something, i.e. a wheel or an arm, using information gathered by the surroundings, In robotics, data is usually gathered through sensors, like encoders, range sensors, light sensors, etc. Using this data, robots can determine how they should act.

Let's say you have a cold room, like 10 degrees. You want to warm it up to a nice 23 degrees (celsius). Luckily you have a heater. You set it's thermostat to 23, and it starts heating. It heats as fast as it can, and quickly gets to 23 degrees. It immediatly turns off. However, the coils on the heater are still warm, and continue to heat the air for a while after. The room heats up to 25, before the coils cool down, and the room loses heat to the environment. It dips down to 23 degrees, and the heater turns on - but it takes time for the coils to turn on, and during this time the room cools down to 21 degrees. This oscilliation around the set point slowly dies out, over a long period of time.



PID is designed to intelligently approach the target to reach it as quickly as possible. So in this example, the heater would have turned off before it hit 23, say at 21 degrees, such that it natrually warms up to 23.

In robotics, the same concept can be applied. Many teams use PID control to drive during autonomous, using encoders as their sensor, shooting, using cameras as their sensor, or rotating, using gyros as their sensor.

The main equation for PID Control is

$$output = P \times error + I \times \sum error + D \times \frac{\delta error}{\delta t}$$

Proportional

$P \times error$

Proportional control is using a predetermined constant (`kP`)to control how much a mechanism can move. Every time the PID code is run, the error is calculated, and the proportional gain is multiplied to this. In the car analogy, lets say the pressure you were applying was inversely proportional to the distance you were from the stop sign. `P = 1/(error^2)` and `error` is distance from the stop sign. (Note, P is 1/error^2 because you want the output to be 1/error.)

Distance	Output
200	.005
150	.0067
100	.001
50	.02
10	.1
1	1

Using this P value, we apply more pressure the closer we get, causing us to slow down.

Using only Proportional control can be done, and is usually better for slow moving mechanisms, or mechanisms where you don't need com

Integral

$I \times \sum error$

When controlling a motor with just P, you may find that it oscillates. This happens because it has too much power when it gets to the setpoint, and it overshoots. Then when it tries to correct, it overshoots again, and this cycle continues. One way to reduce this is to lower P. This could have some bad side effects though. By reducing P, your motor may not get *all* the way to where you want it to. It may be off by a few degrees or rotations. To overcome this **steady-state** error, an Integral gain is introduced.

If you've taken calculus, you know the integral is the area under a curve or line. It's the same with PID. The Integral gain is the sum of all the past error. This means the gain will increase more and more the longer the motor isn't where it's supposed to be.

Even though this reduces steady state error, it may increase settling time. If you notice it oscillating a little bit before settling, you may need a Derivate gain.

Derivative

$D \times \frac{\delta error}{\delta t}$

Derivative gain works by calculating the change in error. By finding this change, it can predict future system behavior, and reduce settling time. It does this by applying a brake more or less. This can be useful if it is imperative that you don't overshoot. This isn't even used in the industry much, but if you find yourself with long settling times, it may help to introduce a Derivative gain.

Feed-Forward

The most essential part of a good control loop is a well tuned feed-forward. Feed-forward accounts for the known dynamics of the system, whereas feedback accounts for deviations from the known behavior like some friction and minor changes in weight. Feed-forward models can be derived using physics principles like torque and gravity in conjunction with motor characteristics.

Flywheel

The most simple feed-forward model is the "flywheel" model, where a static voltage is always applied to get the motor to spin at a constant speed. For example, if your motor's maximum RPM is 6000 RPM at 12V, then you should apply $\frac{12}{6000} \times Setpoint$ volts to get the motor to spin at `Setpoint` RPM.

Arm affected by Gravity

Another common model is the rotational arm. The model can be derived using the effect of gravity on the arm. The result of this calulation is the voltage theoretically required to keep the arm perfectly static.

$$A \cos \theta$$

In this calculation, θ is the angle of the arm above the horizontal. A has units of volts and can be found by using the motor's stall and free torque in combination with the weight and length of the arm. Derivation and constant determination can be found in this post.

Cascade Elevator

With this model, a static voltage is added to the output to oppose the force of gravity. The amount of voltage can be determined using the weight of the elevator, the motor torque, and spool radius.

Drivetrain

With the robot drivetrain, many nonlinear factors should be considered because of the large mass and high friction. These terms are kF (static voltage for friction) kV (volts per speed) and kA (volts per acceleration). This model can be used to improve following motion profiles as well as improve open loop control. Team 449's paper on FRC drive characterization shows the derivation of these terms as well as an empirical method of determining them.

Using PID on your robot

Now that you know the math behind PID, it's time to implement it with your robot. There are two ways to do this. One is to create the PID calculaitons yourself, the other is to use WPILib's PIDController. Let's talk about both

Let's create an example drive class

Java C++ Python

```
public class Drive(){
    int P, I, D = 1;
    int integral, previous_error, setpoint = 0;
    Gyro gyro;
    DifferentialDrive robotDrive;


    public Drive(Gyro gyro){
        this.gyro = gyro;
    }

    public void setSetpoint(int setpoint)
    {
        this.setpoint = setpoint;
    }

    public void PID(){
        error = setpoint - gyro.getAngle(); // Error = Target - Actual
        this.integral += (error*.02); // Integral is increased by the error*time (which is .02
seconds using normal IterativeRobot)
        derivative = (error - this.previous_error) / .02;
        this.rcw = P*error + I*this.integral + D*derivative;
    }

    public void execute()
    {
        PID();
        robotDrive.arcadeDrive(0, rcw);
    }
}
```

Tuning Methods

Zeigler-Nichols tuning method works by increasing `P` until the system starts oscillating, and then using the period of the oscillation to calculate `I` and `D`.

1. Start by setting `I` and `D` to 0.
2. Increase `P` until the system starts oscillating for a period of `Tu`. You want the oscillation to be large enough that you can time it. This maximum `P` will be referred to as `Ku`.
3. Use the chart below to calculate different `P`, `I`, and `D` values.

Control Types	P	I	D
P	.5*Ku	-	-
PI	.45*Ku	.54*Ku/Tu	-
PID	.6*Ku	1.2*Ku/Tu	3*Ku*Tu/40

Note

The period of oscillation is one full ‘stroke’, there and back. Imagine a grandfather clock with a pendulum, when it is all the way to the right, swings to the left, and hits the right again, that is 1 period.

Which ones to use

Feedforward control is necessary on all but the absolute simplest of systems. It’s incredibly difficult to get a good response without a feedforward calculation.

P control is best used on slow moving parts that aren’t subject to overshooting, or parts of the robot that don’t need complete accuracy. Turning to a certain degree, for example, can be done with just P in some cases (but not all).

The most common control loop is PI. It combines simple P control with the fine tuning feature of an Integral gain. This is teams are most likely to use.

Complete PID may be overkill for an FRC robot, but if you find that PI isn’t working *enough*, feel free to add D gain