Using WPILib's PID Controller Class

You need an understanding of PID Theory to understand this article. If you don't already understand PID, I would recommend looking at the previous PID Control article

This article was written primarily for java. Most of the details are the same between java and C++, however some details may be different. This article is not applicable to python.

So, you now know all about PID and it's control theory magic, and would like to run some PID on your robot! Well, you've come to the right place, because this article is just for that. This article explains how you can use the inbuilt WPILib PID Controller class for all your PID needs.

> Note: If you are using the Talon SRX motor controller, you can use it's in-built PID feature to run PID *on the Talon*, if you are wired up using CAN. The Talon is able to run it's PID loop faster, resulting in better control. The Talon SRX user manual has details on how to set this up.

Advantages of the WPILib PID Controller Class

It is often tempting to roll your own PID code to control your motors, however using the WPILib version offers several advantages, mainly:

Threading

> The PID Controller class utilizes threading so the PID loop runs much faster, giving it much better response and resulting in better control.

Extra functionality

> The PID Controller class provides more than just a basic PID loop. For example, PID Controller provides ways to set maximum and minimum output's, wrap endpoints around, and give tolerances for the setpoint.

Pre-exisiting tested nature

> The PID Controller class has alread been written, tested and debugged. Doing it your self dosen't provide that certainty, and will near definitely take longer.

Disadvantges of the WPILib PID Controller Class

Could be better control-theory wise

While the WPILib PID Controller class is quite good, for advanced teams, its lack of an acceleration term for example, means that it could be improved upon. But, if this is why you don't want to use it, then you probably don't need this article.

Potentially slow(er) on the RIO

As mentioned earlier, the PID Controller class must run on the RoboRIO. Other alternatives, like using the Talon SRX's inbuilt PID controller, will operate faster and so with better control.

Ramping motors up

The PID Controller class can result in very jerky motion which can be bad for geraboxes. However, increasing the D term should smooth out sudden changes. Alternatively, see "Adding Ramping for Motors".

Linear output assumption

The PID Controller, like nearly all versions of PID, assumes a linear output, however in reality motor responses are curves.

Implementing a basic PID Control

PIDController Javadoc PIDController C++ Reference

1. **create a new instance of a PIDController. In the full / largest constructor, the values are:**

Constructur var name	Explanation
double Kp	The P term constant. See the PID Theory article if you don't understar
double Ki	The I term constant. See the PID Theory article if you don't understan
double Kd	The D term constant. See the PID Theory article if you don't understar
double Kf	The F term constant for feedforward control. See the PID Theory artic
PIDSource source	The input device to the PID loop. For example, an encoder or gyro. No
PIDOutput output	The output device for the PID loop. For example, a motor controller. N
double period	How often the PID loop should run. Defaults to 50ms.

2. **Set options you want (see options of PID Control)**

- Mandatory options: setSetpoint, setTolerance(or the %/abs versions)
- Optional ones: setContinuous, setInputRange

3. enable
4. You can set options - such as the PID constants, setpoints, ranges, etc. while the PIDLoop is running - you may want to call reset() if you do though (particularly if you change te setpoint)
5. disable
6. free, if you want to clear up the memory and are done with the PIDController

Options of PID Control

Function/option	Explanation
disable	Sets output to zero and stops running.
enable	Starts running the PID loop.
free	Sets all it's variables to null to free up memory.
reset	
setInputRange	Set's the minimm and maximum values expected from the input. Needed to us
setOutputRange	Set's minimum and maximum output values. Should also constrain the totalErr
setContinuous	Treats the input ranges as the same, continuous point rather than two boundar
setPID	Set's the P,I,D,F constants.
setSetpoint	Set's the target point for the PID loop to reach.
setTolerance	Let's you implemenet your own Tolerance object. PidController.onTarget() will r
setAbsoluteTolerance	Makes PIDController.onTarget() return True when PIDInput is within the Setpoi
setPercentTolerance	Makes PIDController.onTarget() return True when PIDInput is within the Setpoi
setToleranceBuffer	Sets the number of previous error samples to average for tolerances before on

Velocity PID Control

To use PID Controller to maintain a velocity - say for a shooter fly wheel or closed loop driving:

- You should use a feedforward term (Kf)
- Your PIDSource should probably have a PIDSourceType of kRate
- Be careful of what your PIDSource is giving - for example, if you use an encoder, and it gives encoder positions, but you want speed, then you might need to wrap it with your own code that gives the rate of change instead.

Using PID Subsystem

WPILib provides the PID Subsytem class to provide convenience methods to run a PIDController on a subsytem for simple cases. For example, if you had an elevator subsytem that needed to stay at the same height, you could use a PIDSubsystem for that.

To use, rather than extending Subsystem, extend PIDSubsytem.

You will need to define the functions returnPIDInput and usePIDOutput to give to the PIDController, and you will want to in the constructor for your subsytem call:

```
super(name, p, i, d, f, period)
```

You can access the internal PIDController with getPIDController()

Example PIDSubsystem to control the angle of a wrist join (taken from WPI's FRC Control System Screensteps live)

```
      package org.usfirst.frc.team1.robot.subsystems;
      import edu.wpi.first.wpilibj.*;
      import edu.wpi.first.wpilibj.command.PIDSubsystem;
      import org.usfirst.frc.team1.robot.RobotMap;


      public class Wrist extends PIDSubsystem { // This system extends PIDSubsystem

              Victor motor = RobotMap.wristMotor;
              AnalogInput pot = RobotMap.wristPot();

              public Wrist() {
                      super("Wrist", 2.0, 0.0, 0.0);// The constructor passes a name for the
subsystem and the P, I and D constants that are useed when computing the motor output
                      setAbsoluteTolerance(0.05);
                      getPIDController().setContinuous(false); //manipulating the raw
internal PID Controller
              }

              public void initDefaultCommand() {
              }

              protected double returnPIDInput() {
                      return pot.getAverageVoltage(); // returns the sensor value that is
providing the feedback for the system
              }

              protected void usePIDOutput(double output) {
                      motor.pidWrite(output); // this is where the computed output value
fromthe PIDController is applied to the motor
              }
      }
```

Explanation of the various PID WPILib class's

These are all found at edu.wpi.first.wpilibj, except for PIDSubsystem which is at edu.wpi.first.wpilibj.command

PID WPILib Class	Function/role
PIDController	The main PID Class that runs your PID loop and has been referenced many times i
PIDSubsystem	See
PIDInterface	A generic PID interface with generic methods. Extends controller. If you wanted y
PIDOutput	An interface for the function PIDWrite to be implemented by an output device su
PIDSource	An interface to be implemented by input sensors.
PIDSourceType	An enum for the two types of PIDSources - Displacement and Rate.

Adding Ramping for motors

As mentioned earlier, the best way is generally to increase your D term as it will smooth out sudden changes. However, alternative options, if for some reason you could not change your D term:

- Create a wrapper function for PIDWrite that dampens motors. This function would store the previous output to the motor, and if given a new output that was say greater than 0.2 higher, it would only increase it by 0.2, and then increase it more after a brief wait. Note that this will reduce the effictiveness of your control, and will most likely mess up the I term of the PID loop
- Dynamically change the minimum / maximum values of your PID Controller. Say, whever PIDWrite get's called, change the PIDController's maximum and minimum values to be around a certain band. This is basically the first option, but a bit better as it will limit the I term and stop it from going crazy.

Dampener function

```
public class PIDMotor implements PIDOutput
{
        /** The motor that will be set based on the {@link PIDController} results. */
        public PWMSpeedController motor;
        private double previousOutput = 0.0;
        private double rampBand;
        private double output;

        /**
         * Constructor for a PID controlled motor, with a controllable multiplier.
         *
         * @param motor The motor being set.
         * @param rampBand The acceptable range for a motor change in one loop
         */
        public PIDMotor(PWMSpeedController motor, double rampBand) {
                this.motor = motor;
                this.rampBand = rampBand;
        }

        public void pidWrite(double pidInput) {
                if (Math.abs(pidInput - previousOutput) > rampBand) { //If the change is greater
that we want
                        output = pidInput - previousOutput > 0 ? previousOutput + rampBand :
previousOutput - rampBand; //set output to be the previousOutput adjusted to the tolerable band,
while being aware of positive/negative
                }
                else {
                        output = pidInput;
                }
                motor.set(output);
                previousOutput = output;
        }
}
```

Dynamically changing function

```
public class PIDMotor implements PIDOutput
{
        /** The motor that will be set based on the {@link PIDController} results. */
        public PWMSpeedController motor;
        private PIDController controller;
        private double rampBand;

        /**
         * Constructor for a PID controlled motor, with a controllable multiplier.
         *
         * @param motor The motor being set.
         * @param rampBand The acceptable range for a motor change in one loop
         * @param controller The PIDController this was passed as output to
         */
        public PIDMotor(PWMSpeedController motor, double rampBand, PIDController controller) {
                this.motor = motor;
                this.controller = controller;
                this.rampBand = rampBand;
                controller.setOutputRange(0 - rampBand, 0 + rampBand);
        }

        public void pidWrite(double pidInput) {
                motor.set(pidInput);
                controller.setOutputRange(pidInput - rampBand, pidInput + rampBand);
        }
}
```