Udit's Guide to: PID

What is PID?

- It is an acronym for a popular feedback loop that is used to make precise robotic movements. It stands for P: Proportional, I: Integral, D: Derivative.
- It is an advanced feedback loop which interprets sensor values (Encoder, potentiometer, gyro etc) and calculates the appropriate motor power to get the subsystem (drive, arm, lift, manipulator) to a target position.

How is PID useful?

Traditionally, PID was used to help control systems that are used in industrial processes such as an automated conveyor belt, or a robotic welding arm. It can be used in robotics to help automate a subsystem accurately without sacrificing speed. Automation is very important in the VEX Robotics Competition because there is a 15 second autonomous portion of a match, and the additional 60 second programming skills challenge. Both require the robot to move completely autonomously, or in other words, without human control. Typically, time based autonomous movements are the simplest to create, by stating:

```
motor1.move(75);
delay(500);       //move forward for 500 milliseconds at 75 power
motor1.stop();
```

However, the issue with time based autonomous movements is that they are very unpredictable. Unpredictable factors include: changing battery voltage, or changes in environmental conditions such as moving up a slanted surface or colliding with an obstacle. When executing autonomous movements, unpredictability should be reduced, and as a bonus any variances should be corrected on the fly.

Sensors can be used to overcome the shortcomings time based autonomous movements. For example a motor can be programmed to run forward until 1000 counts of a wheel encoder, and then stop:

```
bool atTarget = false;
while(!atTarget){
  motor1.move(75);      //run motor if the target has not been reached
  if(motor1.get_postion() > 1000){
    motor1.move(0);     //stop motor once target is reached
    atTarget = true;    //exit while loop
  }
  delay(25)            //don't hog cpu, allow a small delay
}
```

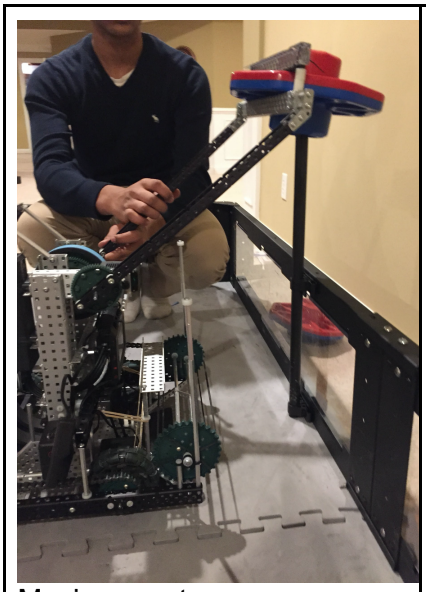
This more accurate than having the robot drive for 1 seconds then stopping, because there is a definite target for the robot to reach. If there is extra weight or friction in the system, a

new load added, or the battery is lower despite taking longer to reach the target, the robot will still continue to the target encoder value.

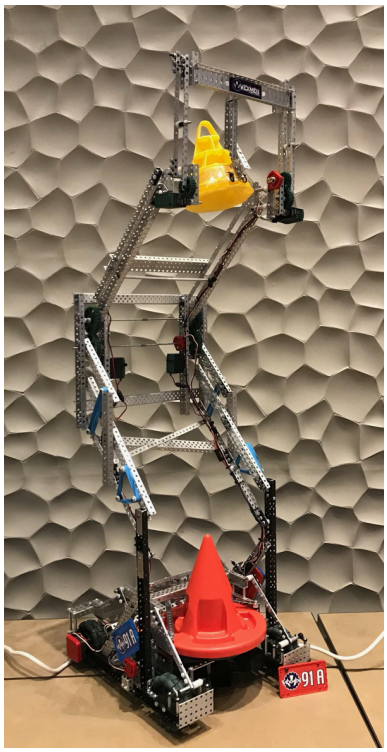
While this loop may force the robot to move a minimum value, there is nothing that prevents it from moving to far. For example, if the robot has a fully charged battery and performs this movement, it may move 1200 encoder counts due to the inertia of the robot moving much faster than normal.

A PID feedback loop solves overshooting and undershooting the target, by performing calculations based on the sensor value, which result in a motor power which would be sent to the motor. Its goal is to bring the robot subsystem to precisely reach its target, without going over or under.

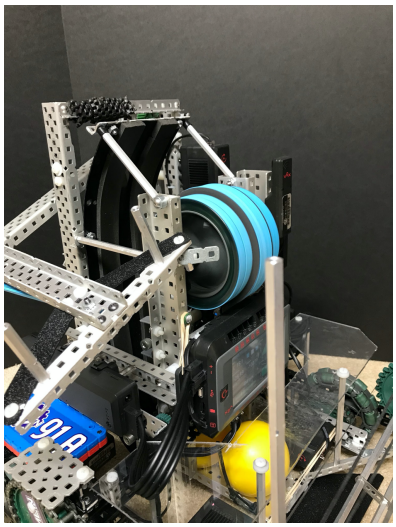
Example Situations:



Moving arm to score a cap from the ground to a fixed post



Picking up 3lb mobile goal and driving, or lifting light cone to various heights



Shooting a ball from a flywheel to hit a target

See it in action!!

How is PID Feedback Calculated?

Until now, the PID loop has been described as some sort of magical feedback loop that solves autonomous movements. It is actually a feedback loop, which means it takes an input and results in a calculated output. The input for a PID loop is a process variable from a variable (essentially the current position of the system) and compares it to a setpoint, which is the target

sensor value that the system is trying to reach. The difference between the setpoint and process variable is known as the error.

```
float error = setPoint - processVariable;
```

A PID feedback loop continuously measures error and elapsed time. Based on these calculations the proportional, integral, and derivative terms are calculated. The output is the sum of the proportional, integral and derivative components, and the output is power that is set to the motors. Below, the calculation for each of the term si shown and their characteristics are described.

Proportional:

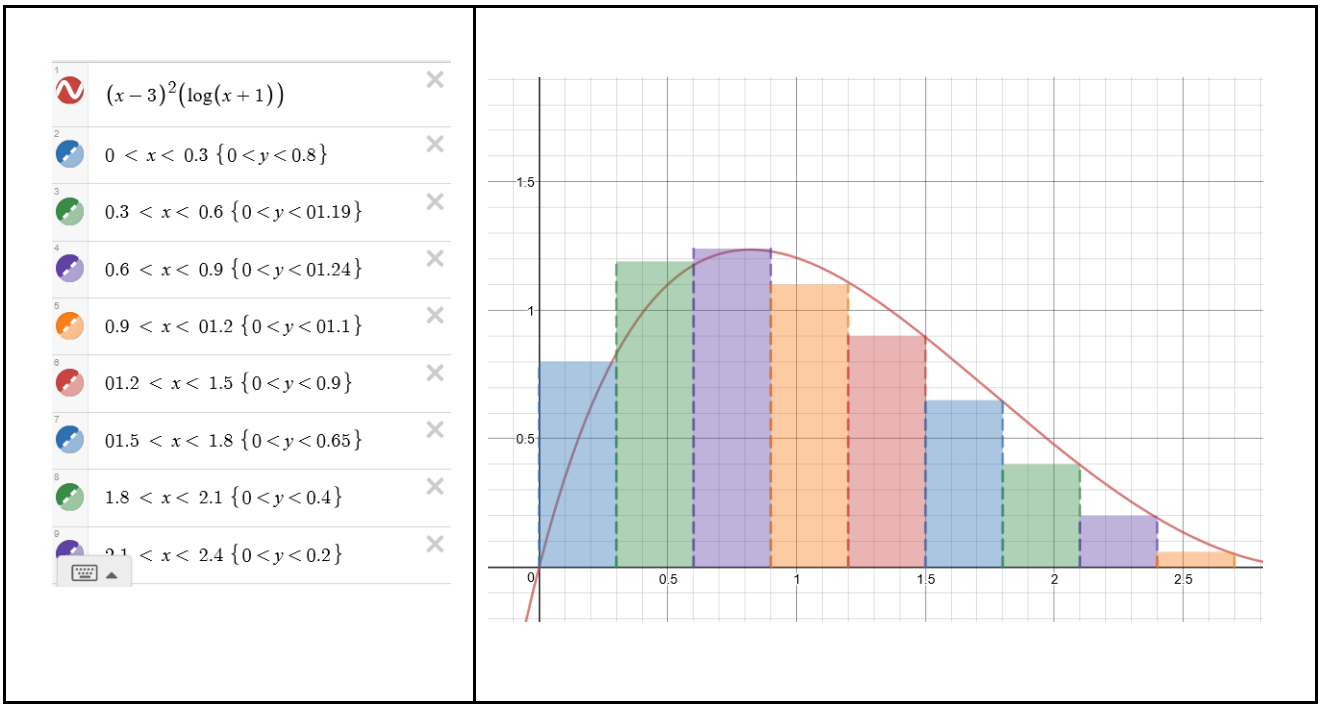
As the name states this term is directly proportional to the error left in the system. As the system approaches the target, the P term decreases. This can be seen as it is calculated by multiplying, the error and a constant called the constant of proportion (kP). As the subsystem gets closer to the target the power being sent to the motors decreases so that the system approaches the target slowly in order to prevent the subsystem from overshooting the target by too much.

```
P_term = error * kP
```

For some instances just calculating the P term and applying it as the power is sufficient for autonomous moments. In other cases, the error between the robot and the target may be too small that once it is multiplied by the kP it will not result in enough power to move anymore. This is known as steady-state error and the robot will undershoot the target. In other cases the kP may be too large and will not slow the subsystem down fast enough to stop accurately at the target, resulting in oscillations around the target.

Integral

In calculus, the integral is the area under the a curve. In this application the calculation for integral is simplified by splitting the area under the curve into small rectangles. For the purposes of the I term, this calculation will suffice. As seen in the following graph:



The area of the rectangles is then calculated by multiplying the error from the current loop and the time that has elapsed since the previous loop. The time that has elapsed since the previous loop will be referred to as the delta time, from here-on forward. Delta time can either be calculated using the processor's timer, or estimated by the length of the delay in each feedback loop. In the graph above, the delta time accounts for the width of the rectangle, while the height of the rectangle is the error of the system. The integral is a sum of all the previous rectangles. Similar to the P term, the I term is scaled to the motor power using another constant which can be called kI.

```
float sigmaI += error * deltaTime;
```

Since the error is added to the total error in every loop, the integral increases to a high number very fast as the robot approaches the target. This term acts oppositely to the P term because it increases as the robot gets close to the target value. The use of the Integral is to help give the motors an extra push when steady-state error arises. The benefits is that it can slowly ramp up the power to the motors if the system slows down unexpectedly. I.e. a bent axle which adds more friction, too much load on the robot, low battery, or old/ broken rubber bands.

Integral is prone to "wind-up" since the integral grows too quickly and could cause the system to overshoot once it breaks free from the steady state error. This could introduce unwanted overshoot. Also, since the integral is summed in every loop, unwanted wind-up can occur especially if the robot needs to cover a large distance

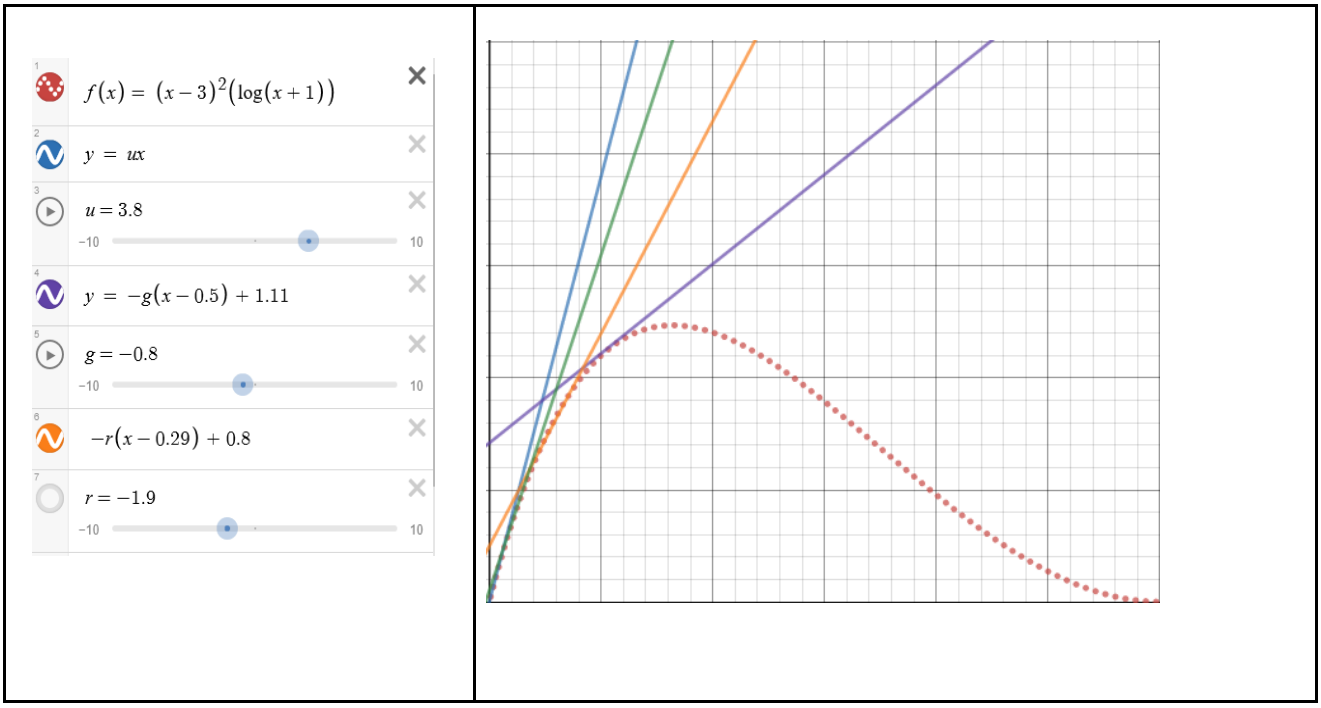
Derivative:

In calculus, the derivative is the slope of a curve over a set domain. In a PID loop, the D term does a similar calculation. For the simplicity of calculations the D term is calculated by the

change in error divided by the delta time. Once again for the purposes of robotics the simplified calculation suffices (Slope formula).

```
float dError = (currentVal - lastVal) / deltaTime;
```

A visualization of the derivative calculations on error versus time graph:



The derivative is also multiplied by a constant, kD, which scales it to be an appropriate motor value. As the robot gets closer to the target, the slope of the motion curve gets steeper, which means the derivative term gets increasingly negative. The purpose of this term is to look at previous errors and detect that the error is decreasing rapidly and start to decrease the power sent to the motors, so that the robot can stop more accurately at the target without overshooting. The benefits are that it can help the robot slow down very quickly once the robot nears the target so that it can stop accurately. The drawback to this is that it can slow the motors down too early, or too much, so the derivative term needs to be calculated only when the robot is near the target.

How to Combine the Terms?

Each term is calculated then added together.The output given to a motor in the system.

```
float output= error * kP + sigmaI * kI - dError * kD;
```

It can be seen that all three terms have a corresponding constant which allows the programmer to adjust their contribution to the final output. Adjusting each of the variable's output can be called "tuning". The ideal values should result in a movement that moves as fast as possible to a

target position, without oscillation, or over/undershoot. Tuning can be done one term at a time by trial and error, or mathematically by examining features of the position versus time graph such as the amplitude and period of the oscillations.

Are all three terms required?

All three terms do not need to be used. You can decide terms based on the strengths and weaknesses of each term.

Quick estimation (if small under/overshoot is allowable) → **P**
Slow, Fine adjustments (might need dampening) → **P+I**
Accurate and fast (prone to steady-state error) → **P+D**
Most precision (requires more time to tune the constants) → **P+ I+ D**

Further Exploration:

This guide just walks through the basics of a PID loop. Here are some various extensions of a PID loop that I have used, or plan to use in the future. (With reference to pg 3)

1. **Selective PID:** considering the drawbacks and advantages to each of the PID terms, a modified PID loop can be written so that the I and D terms are only calculated at select times. Additionally, PID can be
2. **Feed forward + PID:** In some situations a system might need to utilize PID in order to maintain a position. For example a robotic arm has to counter the force of gravity on it and various loads being placed on it. Or a flywheel needs to spin at a constant velocity so that it can accurately shoot balls at a target. In these systems a feedforward component can be used to set a solid base power and PID can be applied for fine adjustment.
3. **Sensor Filtering:** some sensors such as gyros return noisy data, which is unusable for a PID loop. That means the noisy sensor data needs to be filtered before being used in a PID loop. Additionally, the D term may also be noisy, so filtering it can be beneficial.