PID Control System

Joey Marcinowski

February 5, 2021

1 How PID Controllers Work

All PID systems need 3 things, a feedback input, a system output, and some code to manipulate the output. In our case, the feedback input is used with a gyro from the VEX inertial sensor, the output is the voltage to the motors (one side spinning in reverse). While the program is running, I will specify a target heading, which would create an error difference in code, recalculating the motor voltages.

The PID controller is tuned using 3 variables, (Kp, Ki, Kd) each change their corresponding equation to be more in tune, or out of tune. Lastly each of these equations has an input variable which is the error between where it is and the programmed target heading.

1.1 Proportional Control

Proportional control reacts only to how far the robot is away from it's target. When it has a greater error, the robot will spin faster, when it has a lower error, the robot will spin slower. By itself, this is a poor control loop to use because the robot will always end in an oscillation between pointing left, than right. The second issue is that it only works for large errors, like when it's +20 degrees off, When the error is too small (i.e 2 degrees), the output will be too small and the robot won't turn.

$$PID_p = K_p * error$$

1.2 Integral Control

Integral control allows the robot to turn for very small angles. It does this by repeatably summing itself so the output of the equation is always growing, unless the error is 0. On the robot, this looks like the motors will keep speeding up any time the robot isn't turning towards it's target. For smaller angles, this works well, but for larger angles it has the issue of massively overshooting. The way we got around that is we set Ki (the variable to tune this equation) to 0 for any angle greater than 20 degrees, anything else and it'll be 0.005.

$$PID_i = K_i * \int_0^t error * t$$

t = time

In code, it's much simpler to understand

$$PID_i = PID_i + K_i * error$$

1.3 Derivative Control

Derivative control reacts to the speed the robot is turning at, When just using proportional or integral control the robot will continue to overshoot because it's not programmed to stop the turn comfortably. Derivative control by itself will stop any left right accelerations by spinning the wheels opposite to the direction the hit came from. The issue is this control doesn't tell the robot to turn to a specific point, it just stops any unwanted rotation so it must be used with at least proportional control.

$$PID_d = derror * t/dt$$

In code,

$$PID_d = K_d * (error - prevError)/t$$

t = time since the last motor voltage update

1.4 Wheel Voltage

From here, it's very simple to get the wheel voltage of each side.

$$PID = PID_p + PID_i + PID_d$$

$$LeftVoltage = -PID$$

$$RightVoltage = PID$$

The system is capped at 12v on VEX motors

2 PID tuning

2.1 Kp tuning

Each PID variable should be tuned once at a time, which means all other variables should be set to 0. The first variable to tune is Kp, so we set it to 1. If the robot spins out of control, it's because the variable should be negative. If the robot doesn't pass the programmed target point and undershoots, Kp should be raised or lowered away from zero. If the robot begins to shake, it's over tuned and needs to be made closer to zero. Ideally, the robot should steadily oscillate between the left and right hand sides, overshooting enough that the error is big enough for it to correct.

2.2 Kd tuning

The second variable to tune is Kd and the reason is for most VEX robot's Ki control system isn't even needed. Start by setting all the tuning variables to 0 (write down Kp first). Set Kd to 1, again if it spins uncontrollably the variable should be in the negative range. Assuming that it's not spinning, you should be able to try to turn the robot and feel some resistance. If you feel there should be more resistance, make Kd further from zero. If you feel too much resistance and the robot shakes, it should be set closer to zero.

2.3 Ki tuning

If the PD controller is working but the robot isn't making precise enough turns, try setting Ki. We set Ki inside an if statement to not through off the rest of the tuning, and Ki will only be set to a non-zero value when it's inside an x degree range from it's target. Again, if the robot spins uncontrollably it should be the opposite range of numbers. If it shakes, it's over tuned and should be closer to zero, and if it takes too long for the robot to correct for a small error it should be further from zero.

3 Turn Commands

One way of turning with this system is just to set the target heading and wait x seconds for it to finish, however with a simple function we wait for the Ki variable to correct the robot to a reasonable margin before it continues with the rest of the program. For our margin, we wait until the robot is within 0.6 degrees off from it's target.