Proportional Integral Derivative (PID)

A Feasibility Study

By: Pranav Bangarbale

What is PID?

A **Proportional Integral Derivative** is an algorithm commonly used in the industry to improve automation control. Its main purpose is to minimize error with the assistance of feedback loops. One common application is your home's temperature control.

For the purposes of this study, a PID algorithm was implemented to control DC motor rotation so that drift (lateral movement to the right or left) can be minimized on a robot using a pair of such drive motors traveling over specific distances. A simple tank drive program was used as the control for this study.

How does a PID Algorithm Work for Motors?

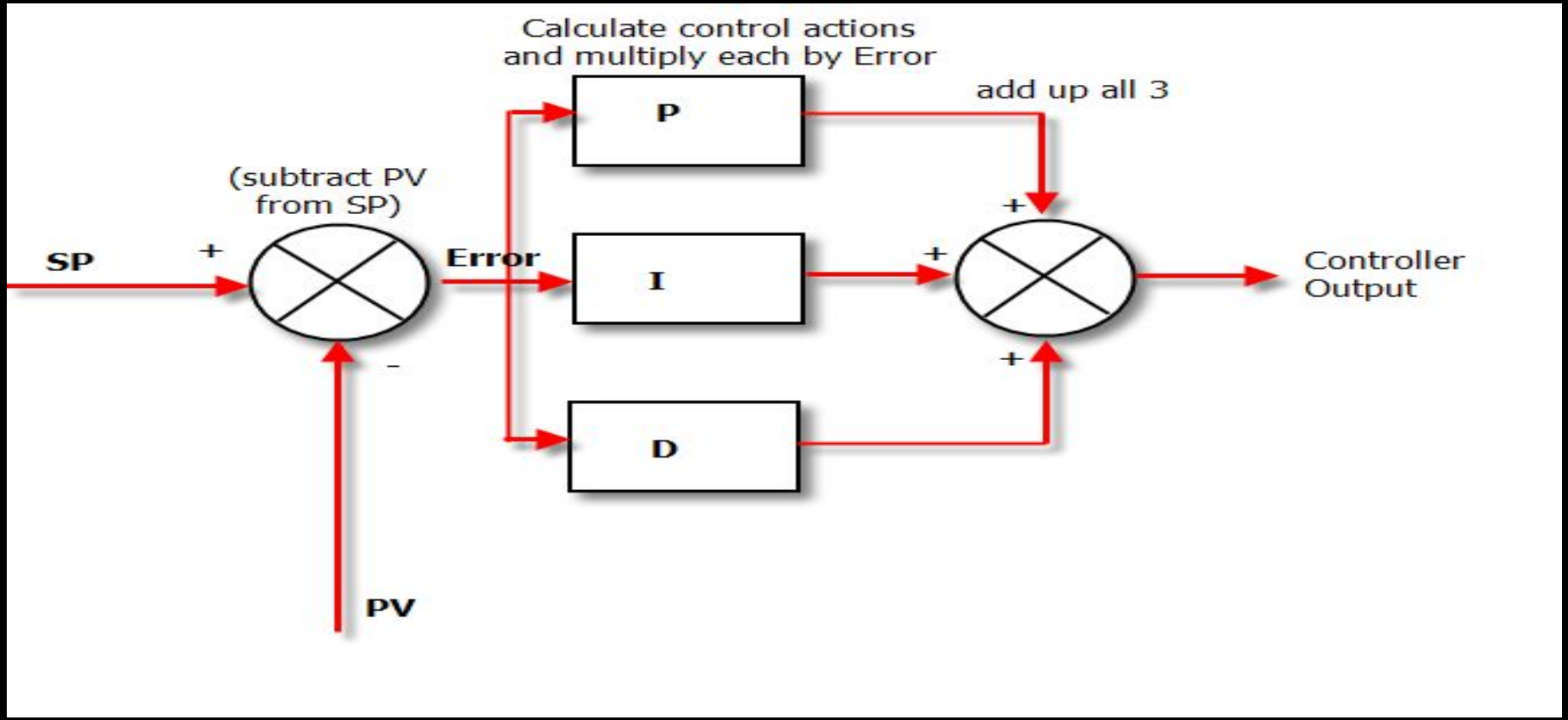
- Constantly calculates and records past and current errors
- The **Error** term is actually the difference between the encoder values across the two motors
- A calculated term based on Error is actually applied to the “input” in the diagram so the input is constantly adjusted to accommodate the feedback.

The Proportional term is multiplied by the error term, to respond to the current error. It defines the error correction applied based on the value of the error.

The Integral term is multiplied by the error term, then added to all of the integral terms before it is used to account for all past errors.

The **Derivative** term is multiplied by the difference between previous error and current error. It is used to predict future error (this becomes very complicated and was not used in this study!)

PID - A Pictorial View



SP = Set Point (Desired target Value/input); P = Proportional Gain, I = Integral Gain, D= Derivative Gain;

Simple Move Algorithm

The Simple Move algorithm sets the motors to the required power independent of the error feedback (see code snippet below). This means that the two motors could rotate at different speeds, causing a drift to the left or right from the target point. Both motors stop rotating when they reach the target encoder value.

```
    rightEncoderSP = rightMotor.getCurrentPosition(); // read the right motor encoder before starting in the loop
    leftEncoderSP = leftMotor.getCurrentPosition();  // read the left motor encoder before starting in the loop

    float totalTicks = rotationsPerInch * (float)distance * ticksPerRotation; //Calculate the total ticks to move.


    while((rightMotor.getCurrentPosition() < (rightEncoderSP + totalTicks)) && (leftMotor.getCurrentPosition() < (leftEncoderSP + totalTicks))) {

        rightMotor.setPower(nPower);
        leftMotor.setPower(nPower);
    }
    rightMotor.setPower(0);
    leftMotor.setPower(0);

}
```

Directly Set Target Power
(NO Error Correction)

Intermediate PID algorithm

This program included two corrections:

- The first correction implemented the “Proportional” term in the PID equation.
- The second correction added the “Integral” term to the PID equation.
- As a result, this program could only respond to the current error and accumulated (past) errors. It could not estimate/correct future errors (no “Derivative” term).

** See code snippet on next slide

Intermediate PID algorithm

```
    error = (int) rightEncoder - (int) leftEncoder; // calculate the error, which is the difference between the motor's encoder readings.

    integral = integral + error;

    correction = (error * KpTimesPower) + (integral * KiTimesPower);

    if (correction > 0)  { // if correction is greater than 0, slow the right motor

        powerRight = nPower - Math.abs(correction); // slow right motor down to correct robot's path

        powerLeft = nPower; // run left motor at nominal speed

        // under no circumstances should the correction be larger than the right motor power.  If it is, set PowerRight to zero rather than letting it go negative.

        if (powerRight < nPower / 2) {

            powerRight = nPower / 2;

        }

    } else { // correction is less than (or equal to) 0, slow the left motor

        powerRight = nPower; // run right motor at nominal speed

        powerLeft = nPower - Math.abs(correction);  // slow left motor down to correct robot's path

        // under no circumstances should the correction be larger than the left motor power.  If it is, set PowerLeft to zero rather than letting it go negative.

        if (powerLeft < nPower / 2) {

            powerLeft = nPower / 2;

        }

    }
    rightMotor.setPower(powerRight); //set the power
    leftMotor.setPower(powerLeft);
}
```

Integral term

Proportional Term

Total Correction

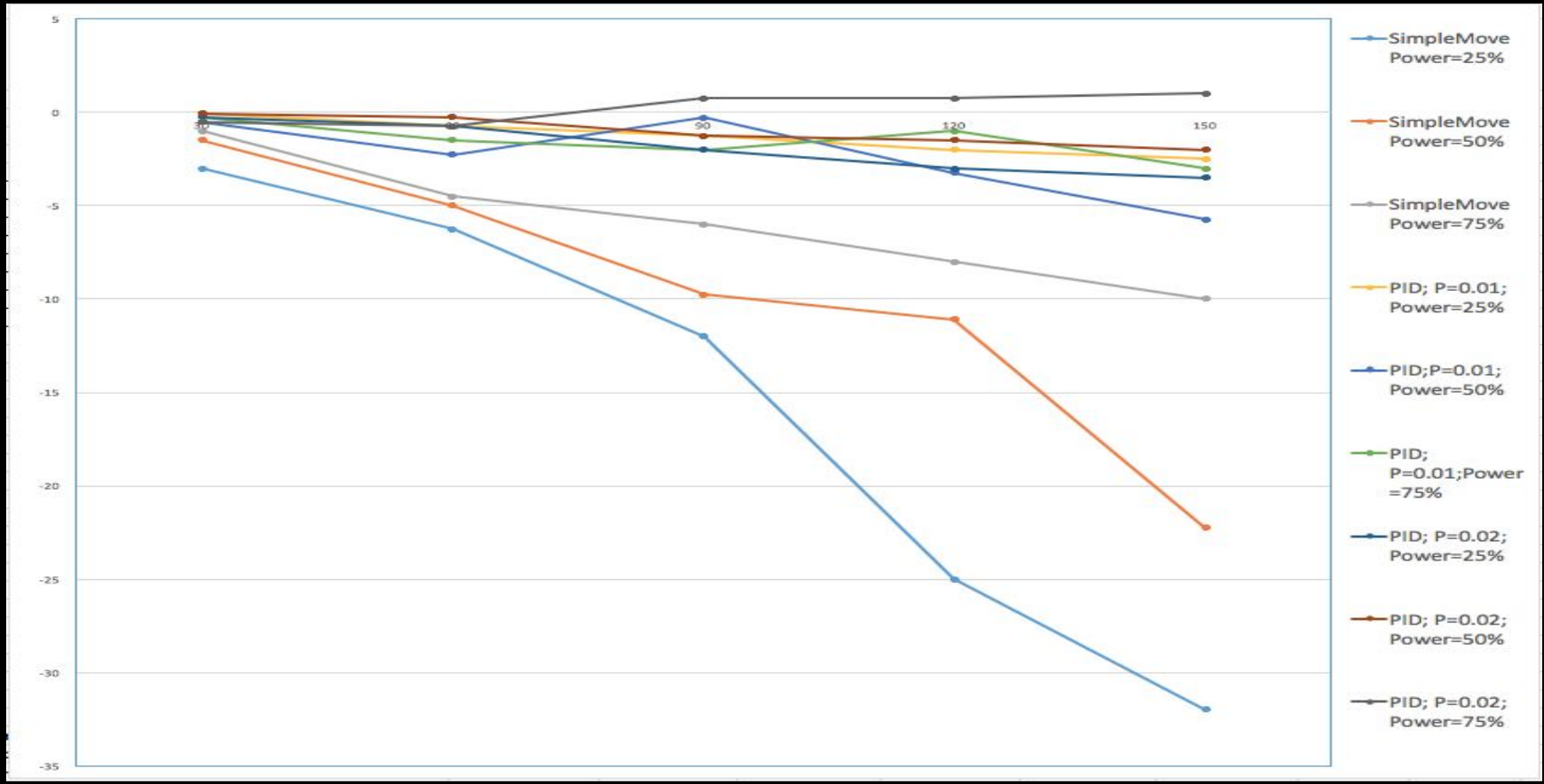
Overcorrection Prevention

Set Target Power w/Corrections

Testing - Results

X-Axis: Distance Traveled (in)

Y-Axis: Drift (in) -ve = left ; +ve = right



Testing - Observations

- When the Simple Move program was executed, the combination of no error correction and motor imbalance on the demo bot caused extreme drift.
- When the PID program was executed, there was minimal drift, only straying to a maximum of 5 inches over a max distance of 150 inches.
- Accuracy of both programs improved with an increase in motor power.
- When the Proportional term (P) was doubled, a minor overcorrection (approx. 1 in) was observed.
- When the Integral term (I) was set to a value <= 0.0001, the robot overcorrected excessively (drift was worse than even Simple Move).

Conclusion

- The PID algorithm outperformed Simple Move by a large margin. (The motor imbalance on the demo bot mostly contributed to this). This settles the debate in favor of using some form a PID algorithm for robot drive motor control.
- The Proportional constant will have to be tuned specifically for every new robot/motor combination for optimal drift correction.
- The P values might need to be tuned when motors are replaced and/or there is a change in the overall balance and distance traveled by the robot.
- The actual drift observed is also dependent on the surface the robot is driving on. For example, the drift values that were observed while testing the demo bot on a flat, hard surface will be different when the same robot is run on an FTC game mat (due to different friction properties).

References

- http://www.csimn.com/CSI_pages/PIDforDummies.html
- https://en.wikipedia.org/wiki/PID_controller
- http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/
- http://www.academia.edu/2202119/Implementation_of_Motor_Speed_Control_using_PID_Control_in_Programmable_Logic_Controller
- Basic PID program in Robot C developed by Jade Traiger