PID Controller for Line Follower

eYSIP-2015

Dhirendra Sagar
Uttam Kumar Gupta
Mentor: Amiraj Dhawan

IIT Bombay

08-07-2015

PID Controller

- Introduction to PID
- Why use PID?
- How to Implement PID?
- PID Implementation
- PID Controller Basics
- PID Constant Factors
- Block Diagram of PID
- PID Formula
- Parameter Comparison
- Waveform Characteristics for PID
- Tuning
- Auto Tuning

Introduction to PID

Introduction:

- PID is an acronym for Proportional Integral Derivative.
- Main task of the PID controller is to minimize the error of whatever we are controlling.
- PID takes input, calculates the deviation from the intended behavior and accordingly adjusts the output so that deviation from the intended behavior is minimized and greater accuracy obtained.
- As the name suggests, PID algorithm consists of three basic coefficients: proportional, integral and derivative which are varied to get optimal response.

Why use PID?

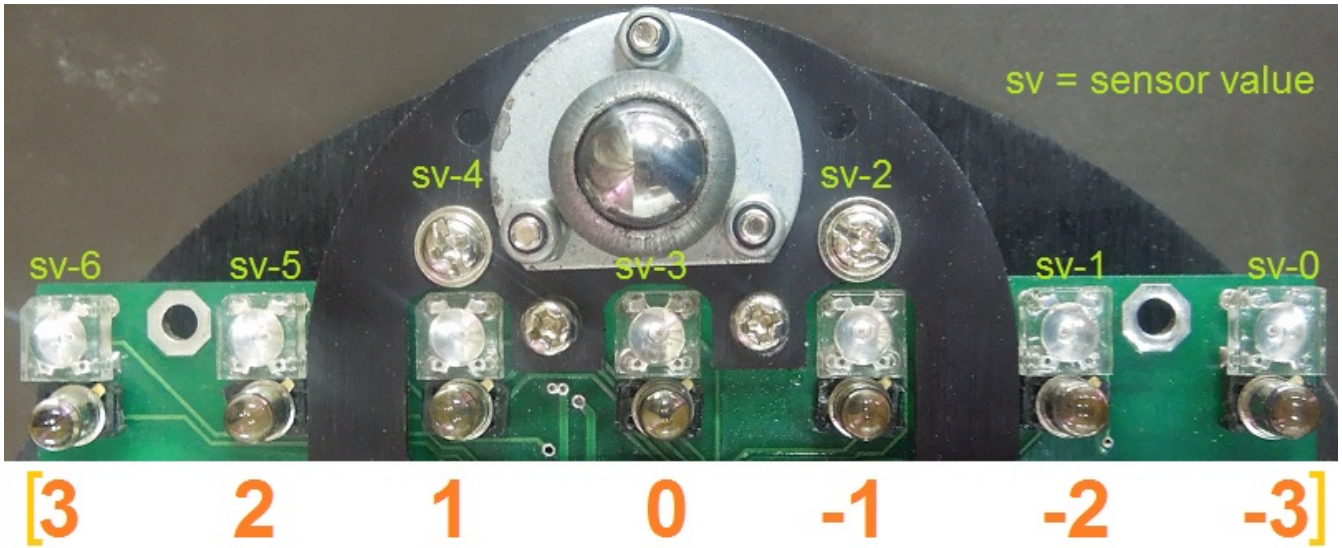
- Line Following seems accurate when carried out at low speed, but as soon as we increase the speed of Robot, it starts to wobble a lot and often gets off the track.
- Hence some kind of control mechanism is required that can enable robot to follow line efficiently at high speeds.
- This is where PID Controller is very useful.
- PID Controller has all necessary dynamics: fast reaction on change of controller input(D Mode), increase in control signal that leads error towards zero(I Mode) and suitable action inside control error area to eliminate oscillation(P Mode).

How to Implement PID ?

- In order to implement the PID line follower one can start with 3 sensor which are so placed on the robot as-
  1. If the centre sensor detects the line it will move forward.
  2. If the left sensor detects the line it will steer right.
  3. If the right sensor detects the line it will steer left.
- This algorithm would make the robot follow line, but the speed of the bot has to be compromised.
- So we can increase the efficiency by increasing the number of sensors, say 5 or 7.
- We used 7 white line sensor strip to follow the white line.

PID Implementation

- Position Variable(PV) ,i.e. deviation from the Setpoint can be obtained as follows:
- We take [-3 to 3] matrix depending upon number of sensor and assign values to each sensor symmetrically.



- Here the sensor values(sv) have to be manually or auto calibrated, so as to provide the symmetric values on both sides of the line to be followed.

PID Implementation

- As shown in figure Final Sensor Value will be obtained after multipling values from matrix with the corresponding sensor value, i.e.
  SV = (-3)*(sv0) + (-2)*(sv1) + (-1)*(sv2) + 0*(sv3) + 1*(sv4) + 2*(sv5) + 3*(sv6)
- For a white line follower all sensors which are on white line (mostly middle 2-3 sensors) will have different sensor values and all sensors which are on black/other line will have different sensor values.
- So variable SV will have different values as the bot moves left , right or forward/backward and will give deviation from SetPoint, i.e. error , which is input to PID Controller.
- For Example- If bot wobble towards left side of the white line, SV value provide negative values to the PID controller, and using those values PID Controller will control the left or right motor speed to align the bot on the white line, same case when bot wobble towards right of White Line.

PID Controller Basics

The basic terminology we use to implement the PID Controller are:

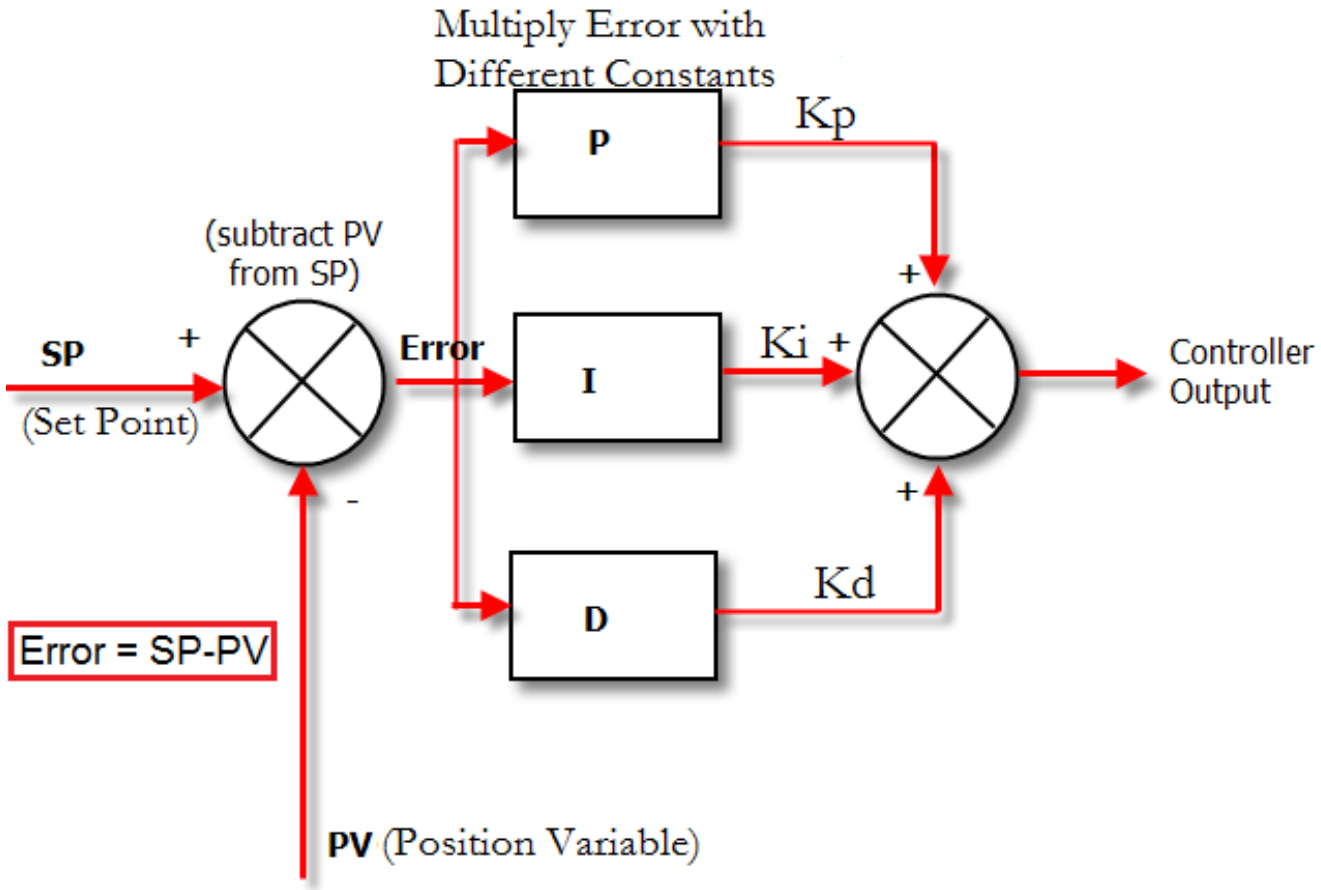
- **Error(e):** The error is the amount by which robot is deviating from its set-point.
- **Proportional(P):** The proportional component depends only on the difference between the set point and the process variable. This difference is referred to as the Error term.
- **Integral(I):** The integral component sums the error term over time.
- **Derivative(D):** The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable.

PID Constant Factors

- Each term (P, I, D) will need to be tweaked in the code. Hence , they are included in the code by multiplying with respective constants.
- Proportional Constant(Kp): It is used to increase or decrease the impact of Proportional.
- Integral Constant(Ki): It is used to increase or decrease the impact of Integral.
- Derivative Constant(Kd): It is used to increase or decrease the impact of Derivative.

Block Diagram of PID

- Output obtained from PID Controller after processing on the error will be added or substracted in motor speed depending on the nature of output(i.e. positive or negative).



PID Formula

- Here we use PID Formula to calculate the Correction ,i.e. the output of PID Conrolleras follows:
  Correction= proportional * Kp + integral * Ki + derivative *Kd where
  1. error(e) = position − setpoint
  2. proportional(p) = proportional + error
  3. integral(i) = integral + proportional
  4. derivative(d) = proportional − last proportional

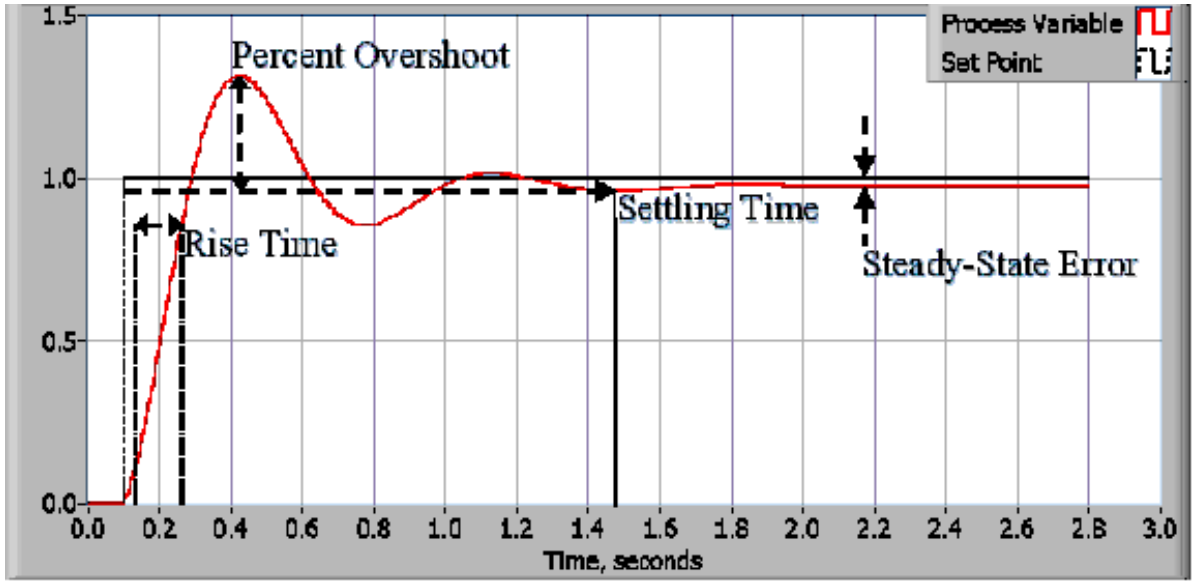
Parameter Comparison

- Summarizing PID Controller:-

term	**constant**	**effect**
Proportional	Kp	It reduces a large part of error based on present error.
Integral	Ki	Cumulative of a small error would further help reduce the final error.
Derivative	Kd	Counteracts the Kp and Ki terms when the output changes quickly.

Waveform Characteristics for PID

- Commonly, the response is quantified by measuring defined waveform characteristics between PID Paremeters and Time:



- **Rise Time** is the amount of time the system takes to go from 10 percent to 90 percent of the steady-state, or final value.
- **Percent Overshoot** is the amount that the process variable overshoots the final value, expressed as a percentage of the final value.

Waveform Characteristics for PID

- **Settling time** is the time required for the process variable to settle to within a certain percentage (commonly 5 percent) of the final value.
- **Steady-State Error** is the final difference between the position variable(PV) and set point(SP).
- Effect of Increasing a Parameter Independently :-

Para-meter	**Rise Time**	**Over shoot**	**Settling Time**	**Steady State Error**
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Minor Change	Decrease	Decrease	No effect

Tuning

- PID implementation would prove to be useless rather more troublesome unless the constant values are tuned depending on the platform the robot is intended to run on.
- PID Tuning is difficult in most cases as there is no proper way to tune , other than experimentally.
- basic guidelines that will help reduce the tuning effort:
  - Start with Kp, Ki and Kd equalling 0 and work with Kp first. Try setting Kp to a value of 1 and observe the robot. The goal is to get the robot to follow the line even if it is very wobbly. If the robot overshoots and loses the line, reduce the Kp value. If the robot cannot navigate a turn or seems sluggish, increase the Kp value.
  - Once the robot is able to somewhat follow the line, assign a value of 1 to Kd (skip Ki for the moment). Try increasing this value until you see lesser amount of wobbling.

Tuning

- Once the robot is fairly stable at following the line, assign a value of 0.5 to 1.0 to Ki. If the Ki value is too high, the robot will jerk left and right quickly. If it is too low, you won't see any noticable difference. Since Integral iscumulative, the Ki value hasa significant impact. You may end up adjusting it by .01 increments.
- Once the robot is following the line with good accuracy, you can increase the speed and see if it still is able to follow the line. Speed affects the PID controller and will require retuning as the speed changes.

Auto Tuning

- For auto tuning you can use simulink so for that purpose either use Matlab but as Matlab is expensive you can try with Scilab.
- In simulink there are various models available for pid tuning.
- For that block diagram of system is created. A transfer function should formed for system. On simulink then pid controller model in Matlab or Scilab is used to get the auto tuning value of the system.

Thank You

THANK YOU !!!