What is PID?

“PID” is an acronym for Proportional Integral Derivative. As the name suggests, these terms describe three basic mathematical functions applied to the error (error = SetVal- SensorVal, where SetVal is the target value and SensorVal is the present input value obtained from the sensor). Main task of the PID controller is to minimize the error of whatever we are controlling. It takes in input, calculates the deviation from the intended behaviour and accordingly adjusts the output so that deviation from the intended behaviour is minimized and greater accuracy obtained.

Why implement PID?

Line following seems to be accurate when carried out at lower speeds. As we start increasing the speed of the robot, it wobbles a lot and is often found getting off track. Hence some kind of control on the robot is required that would enable us to make it follow the line efficiently at higher speeds. This is where PID controller shines.
In order to implement line following one can basically start with just three sensors which are so spaced on the robot that-
If the centre sensor detects the line the robot steers forward
If the left sensor detects the line the robot steers right
If the right sensor detects the line the robot steers left.
This algorithm would make the robot follow the line, however, we would need to compromise with its speed to follow the line efficiently.
We can increase the efficiency of line following by increasing the number of sensors, say 5.
Here the possible combinations represent exact position like-

00100	On the centre of the line
00001	To the left of the line
10000	To the right of the line

There will be other possible combinations such as 00110 and 00011 that can provide us data on how far to the right is the robot from the centre of the line(same follows for left).
Further to implement better line following we need to keep track of how long is the robot not centered on the line and how fast does it change its position from the centre.This is exactly what we can achieve using “PID” control.The data obtained from the array of sensors would then be put into utmost use and line following process would be much more smoother, faster and efficient at greater speeds.
PID is all about improving our control on the robot.
The idea behind PID control is that we set a value that we want maintained, either speed of a motor or reading from a sensor. We then take the present readings as input and compare

them to the setpoint. From this an error value can be calculated, i.e, (error = setpoint-actual reading). This error value is then used to calculate how much to alter the output by to make the actual reading closer to the setpoint.

How to implement PID?

Terminology:
The basic terminology that one would require to understand PID are:
Error- The error is the amount at which a device isn't doing something right. For example, suppose the robot is located at x=5 but it should be at x=7, then the error is 2.
Proportional (P)- The proportional term is directly proportional to the error at present.
Integral (I)- The integral term depends on the cumulative error made over a period of time (t).
Derivative (D)- The derivative term depends rate of change of error.
Constant (factor)- 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.
P-Factor (Kp)- A constant value used to increase or decrease the impact of Proportional
I-Factor (Ki)- A constant value used to increase or decrease the impact of Integral
D-Factor (Kd)- A constant value used to increase or decrease the impact of Derivative
Error measurement: In order to measure the error from the set position, i.e. the centre we can use the weighted values method. Suppose we are using a 5 sensor array to take the position input of the robot. The input obtained can be weighted depending on the possible combinations of input. The weight values assigned would be such that the error in position is defined both exactly and relatively.
The full range of weighted values is shown below. We assign a numerical value to each one.

Binary Value	Weighted Value
00001	4
00011	3
00010	2
00110	1
00100	0
01100	-1
01000	-2
11000	-3
10000	-4
00000	-5 or 5 (depending on the previous value)

The range of possible values for the measured position is-5 to 5. We will measure the position of the robot over the line several times a second and use these value to determine Proportional, Integral and Derivative values.

PID formula:
So what do we do with the error value to calculate how much the output be altered by? We would need to simply add the error value to the output to adjust the robot's motion. And this would work, and is known as proportional control (the P in PID). It is often necessary to scale the error value before adding it to the output by using the constant(Kp).

Proportional:
Difference = (Target Position)- (Measured Position)
Proportional = Kp*(Difference)
This approach would work, but it is found that if we want a quick response time, by using a large constant, or if the error is very large, the output may overshoot from the set value. Hence the change in output may turn out to be unpredictable and oscillating. In order to control this, derivative expression comes to limelight.
Derivative:
Derivative provides us the rate of change of error. This would help us know how quickly does the error change from time to time and accordingly we can set the output.
Rate of Change = ((Difference) – (Previous Difference))/time interval
Derivative= Kd *(Rate of Change)
The time interval can be obtained by using the timer of microcontroller.
The integral improves steady state performance, i.e. when the output is steady how far away is it from the setpoint. By adding together all previous errors it is possible to monitor if there are accumulating errors. For example- if the position is slightly to the right all the time, the error will always be positive so the sum of the errors will get bigger, the inverse is true if position is always to the left. This can be monitored and used to further improve the accuracy of line following.

Integral:
Integral = Integral + Difference
Integral = Ki*(Integral)
Summarizing "PID" control-

Term	Expression	Effect
Proportional	Kp x error	It reduces a large part of the error based on present time error.
Integral	error dt	Reduces the final error in a system. Cumulative of a small error over time would help us further reduce the error.
Derivative	Kd x derror / dt	Counteracts the Kp and Ki terms when the output changes quickly.

Therefore, Control value used to adjust the robot's motion=
(Proportional) + (Integral) + (Derivative)

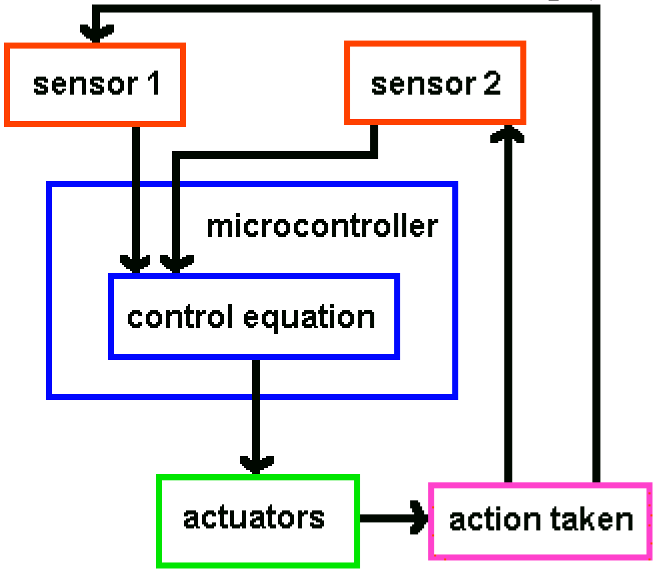
Tuning:
PID implementation would prove to be useless rather more troublesome unless the constant values are tuned depending on the platform the robot is intended to run on. The physical environment in which the robot is being operated vary significantly and cannot be modelled mathematically. It includes ground friction, motor inductance, center of mass, etc. Hence, the constants are just guessed numbers obtained by trial and error. Their best fit value varies from robot to robot and also the circumstance in which it is being run. The aim is to set the constants such that the settling time is minimum and there is no overshoot.
There are some 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.
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 perceivable difference. Since Integral is** cumulative, the Ki value has **a 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.
Pseudo Code:Here is a simple loop that implements the PID control:

```
start:
error = (target_position)- (theoretical_position)
integral = integral + (errordt)
derivative = ((error) - (previous_error))/dt
output = (Kperror) + (Kiintegral) + (Kdderivative)
previous_error = error
wait (dt)
goto start
```

Lastly, PID doesn't guarantee effective results just by simple implementation of a code, it requires constant tweaking based on the circumstances, once correctly tweaked it yields exceptional results. The PID implementation also involves a settling time, hence effective results can be seen only after a certain time from the start of the run of the robot. Also to obtain a fairly accurate output it is not always necessary to implement all the three expressions of PID. If implementing just PI results yields a good result we can skip the derivative part.

A proportional integral derivative controller (PID controller) is a common method of controlling robots. PID theory will help you design a better control equation for your robot.
Shown here is the basic closed-loop (a complete cycle) control diagram:



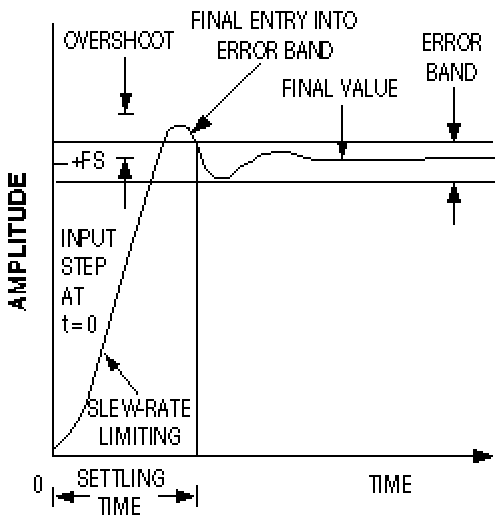
The point of a control system is to get your robot actuators (or anything really) to do what you want without . . . ummmm . . . going out of control. The sensor (usually an **encoder** on the actuator) will determine what is changing, the program you write defines what the final result should be, and the actuator actually makes the change. Another sensor could sense the environment, giving the robot a higher-level sense of where to go.

Terminology

To get you started, here are a few terms you will need to know:

error - The error is the amount at which your device isnt doing something right. For example, if your robot is going 3mph but you want it to go 2mph, the error is 3mph-2mph = 1mph. Or suppose your robot is located at x=5 but you want it at x=7, then the error is 2. A control system cannot do anything if

there is no error - think about it, if your robot is doing what you want, it wouldnt need control!
proportional (P) - The proportional term is typically the error. This is usually the **distance** you want the robot to travel, or perhaps a temperature you want something to be at. The robot is at position A, but wants to be at B, so the P term is A - B.
derivative (D) - The derivative term is the change in error made over a set time period (t). For example, the error was C before and now its D, and t time has passed, then the derivative term is (C-D)/t. Use the timer on your **microcontroller** to determine the time passed (see **timer tutorial**).
integral (I) - The integral term is the accumulative error made over a set period of time (t). For example, your robot continually is on average off by a certain amount all the time, the I term will catch it. Lets say at t1 the error was A, at t2 it was B, and at t3 it was C. The integral term would be A/t1 + B/t2 + C/t3.
tweak constant (gain) - Each term (P, I, D) will need to be tweaked in your code. There are many things about a robot that is very difficult to model mathematically (ground friction, motor inductance, center of mass, ducktape holding your robot together, etc.). So often times it is better to just build the robot, implement a control equation, then tweak the equation until it works properly. A tweak constant is just a guessed number that you multiple each term with. For example, Kd is the derivative constant. Idealy you want the tweak constant high enough that your settling time is minimal but low enough so that there is no overshoot.
P*Kp + I*Ki + D*Kd



What you see in this image is typically what will happen with your PID robot. It will start with some error and the actuator output will change until the error goes away (near the **final value**). The time it takes for this to happen is called the **settling time**. Shorter settling times are almost always better. Often times you might not design the system properly and the system will change so fast that it **overshoots** (bad!), causing some oscillation until the system settles. And there will usually be some **error band**. The error band is dependent on how fine a control your design is capable of - you will have to program your robot to ignore error within the error band or it will probably oscillate. There will always be an error band, no matter how advanced the system.
ignoring acceptable error band example:
```
if error <= .000001 //subjectively determined acceptable
then error = 0; //ignore it
```
The Complete PID Equation
Combining everything from above, here is the complete PID equation:
```
Actuator_Output = Kp*P + Ki*I + Kd*D
```
or in easy to understand terms:
```
Actuator_Output =
tweakA * (distance from goal)
+ tweakB * (change in error)
+ tweakC * (accumulative error)
```
Simplifications
The nice thing about tuning a PID controller is that you don't need to have a

good understanding of formal control theory to do a fairly good job of it. Most control situations will work with just an hour or so max of tuning.
Better yet, rarely will you need the integral term. Thats right, just delete and ignore it! The only time you will need this term is when acceleration plays a big factor with your robot. If your robot is really heavy, or gravity is not on it's side (such as steep hills), then you will need the integral term. But out of all the robots I have ever programmed, only two needed an integral term - and both robots were over 30 lbs with a requirement for extremely high precision (millimeter or less error band). Control without the integral term is commonly referred to as simply PD control.
There are also times when you do not require a derivative term, but usually only when the device mechanical stabalizes itself, works at very low speeds so that overshoot just doesnt happen, or you simply dont require good precision.
Sampling Rate Issues
The sampling rate is the speed at which your control algorithm can update itself. The faster the sampling rate, the higher precision control your robot will have. Slower sampling rates will result in higher settling times and an increased chance of overshoot (bad). To increase sampling rate, you want an even faster update of sensor readings, and minimal delay in your program loop. Its good to have the robot react to a changing environment before it drives off the table, anyway. Humans suffer from the sampling rate issue too (apparently drinking reduces the sampling rate, who would have guessed?).

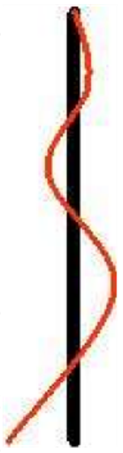
The rule of thumb is that the sample time should be between 1/10th and 1/100th of the desired system settling time. For a typical homemade robot you want a sampling rate of about 20+/second (very reasonable with today's **microcontrollers**).

PID for Line Following

At slower speeds, line following is pretty simple - if the sensors say it is going left, steer right and if going right, steer left. This process has its limitations though, mainly when the speed is increased. This is when a PID controller starts to shine.

PID stands for **Proportional, Integral and Derivative**. A **PID** controller is a mathematics based procedure that processes sensor data and uses it to control the direction (and/ or speed) of a robot to keep it on course. Why does **PID** work better than our simple model described above? Let's talk about how robot acts (or behaves) as it follows a line to see why.

Behaviour of a line follower



Let's say our robot has 3 sensors- Left, Centre and Right. When the Centre sensor sees the line, the robot is programmed to go straight. When the Left sensor sees the line, the robot is programmed to turn right. When the Right sensor sees the line, the robot is programmed to turn left. This will typically cause the robot the wobble back and forth over the line and if going too fast, it may lose control and stop following the line (the red line in the picture to the right) at all.

This method only takes one factor into consideration - is the robot centered over the line. To improve performance, we should also take into consideration 2 more factors - how rapidly is the robot moving from side to side and how long it is not centered over the line. These 3 behaviours are called Proportional, Integral and Derivative in terms of a PID controller.

To discuss PID here is the definition of some terms commonly used:

Target Position - For line following, this is centered over the line. We will represent this as the value 0 (zero).

Measured Position - This is how far left or right robot is from the line. This value will be a negative or positive value to represent the relative position to the line.

Error - The difference between the target position and the measured position the Error.

Proportional - Measures how far your robot is away from the line. The more granular the sensor data is, the more accurately you can measure the robots position over the line.

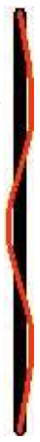
Integral - Measures the accumulated **Error** over time. The Integral value increases while the robot is not centered over the line. The longer the robot is not centered over the line, the higher the Integral value becomes.

Derivative - Measures the rate at which the robot is moving left-to-right or right-to-left. The faster the robot moves side-to-side, the higher the Derivative value is.

P-Factor (K_p) - A constant value used to increase or decrease the impact of Proportional

I-Factor (K_i) - A constant value used to increase or decrease the impact of Integral

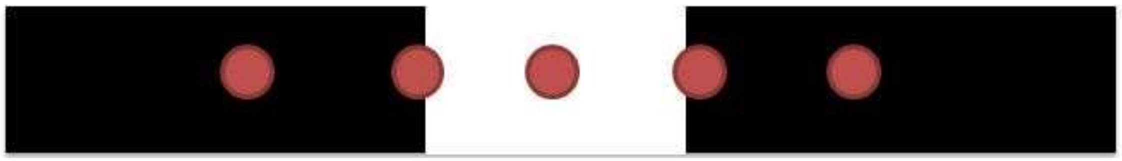
D-Factor (K_d) - A constant value used to increase or decrease the impact of Derivative



By combining Proportional, Integral and Derivative values, we can control the motion of our robot more precisely. The ideal behaviour is represented by the red line in the image to the right. The robot's wavy motion is minimized and the robot stays more centered over the line than it did before.

The overall performance of PID with your robot will depend on the number and precision of the sensors used and the capabilities of the microcontroller you are using.

The Hardware



For this example we will be using 5 sensors for detecting a white line on a black surface. The sensors are used as a digital input of 0 (no line) or 1 (line). The sensors are so placed as shown in the fig above to increase the precision of the sensor array. The sensors values are read in and converted to a binary value to help visualize the line position under the sensors.

00100	Robot is centered over the line
10000	Robot is to the right of the line
00001	Robot is to the left of the line

The full range of weighted values is shown below. We assign a numerical value to each one.

BINARY VALUE	WEIGHTED VALUES
00001	4
00011	3
00010	2
00110	1
00100	0
01100	-1
01000	-2
11000	-3
10000	-4
00000	5 or -5 (depending on previous values)

The range of possible values for the measured position is -5 to 5. We will measure the position of the robot over the line several times a second and use these value to determine Proportional, Integral and Derivative values.

PID Formula

PID is a series of mathematical calculations. We determine Proportional, Integral and Derivative then add them together to come up with a value used to control the robot. K_p, K_i and K_d are used to tune the PID controller.

Proportional
Proportional = K_p*(Difference)

Difference = (Target Position) - (Measured Position)

Integral

Integral = K_i*(Integral)

Since Integral stores the accumulated Difference value, therefore

Integral = Integral + Difference

Derivative

Derivative= K_d *(Rate of Change)

Rate of Change = (Difference) – (Previous Difference)

Derivative is sometimes divided by the interval or time between measurements

Therefore, Control value used to adjust the robot's motion=

(Proportional) + (Integral) + (Derivative)

That represents the math of PID but the real secret of its usefulness is in tuning the PID controller to match the physical characteristics of your robot.

PSEUDO CODE

Here is a simple loop that implements the PID control:

```
 previous_error= (target_position) - (theoretical_position)

 start:
error = (target_position) - (theoretical_position)
integral = integral + (error*dt)
derivative = ((error) - (previous_error))/dt
output = (Kp*error) + (Ki*integral) + (Kd*derivative)
previous_error = error
wait (dt)
gotostart
```

Tuning PID

Once you have PID running in your robot, you will probably notice that it still doesn't follow the line properly. It may even perform worse than it did with just proportional! The reason behind this is you haven't tuned the PID routine yet. PID requires the K_p, K_i and K_d factors to be set to match your robot's characteristics and these values will vary considerably from robot to robot. Unfortunately, there is no easy way to tune PID. It requires manual trial and error until you get the desired behaviour. There are some basic guidelines that will help reduce the tuning effort.

1. Start with K_p, K_i and K_d equalling 0 and work with K_p first. Try setting K_p 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 K_p value. If the robot cannot navigate a turn or seems sluggish, increase the K_p value.
2. Once the robot is able to somewhat follow the line, assign a value of 1 to K_d (skip K_i for the moment). Try increasing this value until you see lesser amount of wobbling.
3. Once the robot is fairly stable at following the line, assign a value of 0.5 to 1.0 to K_i. If the K_i value is too high, the robot will jerk left and right quickly. If it is too low, you won't see any perceivable difference. Since Integral is cumulative, the K_i value has a significant impact. You may end up adjusting it by .01 increments.
4. 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.

Lastly, please keep in mind that you do not need to implement all three controllers (proportional, derivative, and integral) into a single system, if not necessary. For example, if a PI controller gives a good enough response, then you don't need to implement derivative controller to the system. Keep the controller as simple as possible.