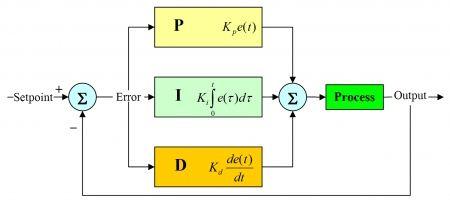
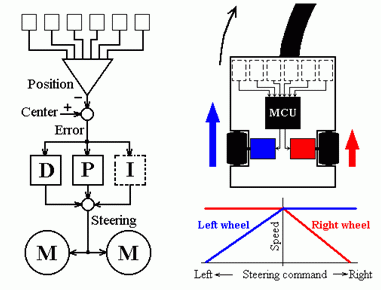
sWhat is PID?



Ok so what is this PID and why is it much talked? Well, PID means proportional–integral–derivative and it is a control feedback method applied in different control systems, its main task is to minimize the error of whatever you are about to control. It is superior quality error control feedback mechanism and the general industry standard. Speaking simple it takes in the inputs calculates the deviation from the intended behaviour and accordingly adjusts the outputs so that the deviation from the desired behaviour is minimized thus getting the highest possible accuracy.  
  
**What does this PID have to do with me?**  
  
Well, there are many applications in the field of robotics like correcting sensor readings, manipulating pwm speed which is the chief objective called motion control. Before using PID control you were simply telling the robots wheels to drive at a certain speed, set by a PWM output hence assuming that the robot wheels would turn at the same speed and the robot would travel in a straight line. This is known as open loop control. This means that you send an output to the motors with no feedback and assume they travel at the speed you set. It is very unlikely that two motors, even two identical motors will turn at the same speed. So some sort of feedback is required to control the speed. This is normally achieved using an encoder. When the speed of the motor is controlled using feedback it is known as closed loop control. Here PID control comes into action! after taking the readings from the encoders it sets the speed such that the bot travels accurately you would have already noticed that the normal geared motors we get (especially in India) don’t have precision they have an error ranging from 3 to 8 percent which can mean a lot when turning especially in line followers even a small error can make a line follower miss the line completely or just confuse the bot!

**The ‘P’ part:-**  
  
The proportional part, for example let us consider that we are making a bot to follow another on a straight line then our interest will be to check continually at what distance we are from the other bot and accordingly set the current speed of the bot .We could do it in two ways just be dumb and say if the distance is greater than the set-point and speed up, then we end up either crashing into the bot of fall behind it based on the value we set to speed up or we can also check the proportional error i.e. the set distance – current distance and then speedup the bot by setting a value to be multiplied by the difference in the distance.  
  
Example for proportional control-  
  
Bot is running at a pwm speed of 180  
Set distance = 2 cm  
Current distance = x  
Deviation = x-2  
Gain = 20 (this is decided by trial and error method)  
If distance < 2  
Pwm speed = pwm speed + (gain x deviation)  
  
in the above example if the deviation is low the increase in speed will be less and if the deviation is more the speed will be increased significantly so the bot will behave better and avoid thing like ramming into other bots or falling behind . The proportional method works pretty well with a high sampling rate and a low settling time which is good thing.  
  
P=E\*Gp  
P is proportional calculated error   
E is the original error  
Gp is the proportional gain constant   
  
**The 'I’ Part:-**  
  
The integral part is the accumulative error made over a set period of time (t); you might think why do we need the accumulative error over time? Let’s take an example, suppose we are driving a car and want to keep it at the centre of the road. At a particular time we go a little to the right then we make the car straight by moving it a little to the left then we get the car to be straight but it might not be at the centre accumulating a minute constant error which after some time when added to the code will get the car to the centre as it grows in number. This could be quite tricky to implement and has a higher settling time also will need a slower sampling rate so as to make it have only small changes and doesn’t accumulate to a fairly large quantity. The integral control improves steady state performance that is when the value is fairly consistent but doesn’t suit ever changing values.  
  
I=Ei\*Gi  
  
Ei Sum of all previous error.  
Gi is the integral gain constant   
  
**The 'D’ Part:-**  
  
The derivative part is the rate of change of error i.e. the difference in the errors as time proceeds For example, the error was C before and now it’s D, and t time has passed, then the derivative term is (C-D)/t. The advantage of the derivative part is that it prevents sudden changes from happening but could also slow down the entire process if not used wisely but along with the integral part it will work wonderfully if tuned correctly. You can use the timer on the microcontroller to measure the time .There exist no D-only controller.  
  
D=E\*Gd  
  
E= (Et1-Et2)/t2-t1 i.e. the difference in value by time difference  
  
Gd is the derivative gain constant   
The whole thing put together should be   
Error = P+I+D  
This error should be added to the value you are controlling  
  
**Sample rate: -**  
  
This is the rate at which the readings are noted i.e. How many times a second are you scanning the sensors per second. A good sample rate will be around 10 to 50 times a second. Strictly speaking it once it’s above 10 it really doesn’t matter much for general robotics but the higher the better and keep in mind that sampling rate matters much when dealing with PID   
  
Tips:-  
• PID doesn’t guarantee off the box results the code requires constant tweaking based on circumstances once tweaked correctly will run exceptionally   
• PID algorithm has some settling time so it will take a few seconds before the bot starts to perform well.  
• Some applications may require using only one or two actions to provide the appropriate system control. This is achieved by setting the other parameters to zero. A PID controller will be called a PI, PD, P or I controller in the absence of the respective control actions.  
• Once you have equations for the PID component functions, the next step is to obtain parameters for the PID equation. Because the selection of parameters depends on the physical characteristics of the system, there is no stock set of values that can be applied to every implementation. Instead, the parameters for the equations must be tuned to the particular platform they are intended to control.  
• The following procedures can be applied when tuning PID values  
i) Analytical:-we can set the values based on theory and circumstantial calculations  
ii) Trial and error :- set random values and keep tweaking the values until desired behaviour is achieved  
• When tuning PID first set the speed to a low value and continue increasing along with retuning the values.  
• When using a PID algorithm a sample rate can be a issue which causes inconsistency just keep playing with the values and try to get the highest sample rate possible keeping the bot follow ideal behaviour  
• When going for competition robotics keep in mind the settling time of the PID algorithm the less the better   
• Though PID works well in controlling errors don’t try to make a 100 rpm and a 300 rpm motor move in a straight line it just won’t happen.  
• If the error can be reduced physically then don’t be lazy DO it!  
  
**PID for line followers**



So, in which aspects of a microcontroller can we use the PID algorithm? Every aspect of it! Form the sensor readings to motor control having PID on everything would be a tough thing indeed but once you get the hang of it your line follower will the best and when having PID running on your bot the more number of sensor you have on your bot the better so that by the time the bot responds at least one of the sensor detects the line   
  
**Sensors interfacing:-**  
  
We should assign an error value for each of the sensor for the PID to be effective   
EX:-  
Consider we have 5 sensors   
The sensors   
00100 – Error value = 0  
01000 – Error value = 2   
10000 – Error value = 5   
00010 – Error value = -2  
00001 – Error value = -5  
  
When setting the error value we notice that by setting the error value to the farthest sensor high we will get aggressive response when that error goes up into the PID control code the value goes too high we should take it into consideration and adjust the value   
  
**Encoders:-**  
  
Before going to motion control we’ll need to know a bit about encoders. Wheel encoders are devices that allow one to measure the precise speed or distance a wheel travels. Depending on the type of encoder used, it may be possible to determine the direction of movement. These are helpful for precise movement, allowing a robot to turn exact angles or move exact distances. Wheel encoders can provide information for odometry and be utilized in localization problems. There are many types of encoders but the main one going with robotics are optical encoders. Encoders can be used to detect things like direction, angle, distance travelled etc for more information refer <http://en.wikipedia.org/wiki/Rotary_encoder>  
  
  
**Motion control:-**  
  
It is better to have encoders on fast line followers to know the actual speed that the motor is running at so as to minimize the possible error which can make the bot totally miss the line. A common control mechanism, for small robots, is to control the velocity of a motor. On a typical robot, differential drive is used for steering. In order to go straight both motor shafts have to be turning at precisely the same rate (and the wheels have to be the same diameter as well). With an encoder one can count how many clicks have gone by in a period and uses that as an indication of velocity. The error between the velocity set point and the actual velocity is used to control the power levels to the motors. Using the PID controller this can be done highly efficiently and accurately.  
  
**Putting Everything Together**  
  
Ok, so now let’s put everything together. Let’s assume we built a line follower robot and are implementing PID for it.  
Specifications (taken in general point of view)  
  
6 Sensors  
10 Bit ADC  
8 Bit PWM  
Gear motors  
Incremental optical encoders  
  
We are applying PID for the sensors and wheel encoders. Lets first consider the sensors generally we use analogue sensors for the line followers here we need to consider the readings of the sensors it is quite common that all the sensors don’t give us a single values to proceed with hence we come in a different approach let us a separate function to read all the sensors and give us a single value so that we can implement the PID algorithm easily let us take the values from all the six sensors and set them to a range of -25 to 25 Then we can set everything correctly the set point should be zero that is when the line lies in the centre then 10 when under an extra sensor right and 25 when to the rightmost sensor similarly -10 when under an extra sensor left and -25 when to the leftmost sensor . Hence the set point should be ‘ZERO’. Now coming to the P, I, D for P and D the E will be same (you can refer to the formula once again) where as for the I term the E is the integral for all the previous errors. all you have to set is the Gain constants for the P,I and D while calculating the gain constants the resolution of PWM on your microcontroller should be taken into consideration in this example I took it to be 8 Bit so the value would range from 0 – 255. Now we should decide the set speed this is also tricky you should get a good response from your bot therefore you should get motors of higher RPM than you require and set the PWM speed to 150 so we can get a change of 100 to nullify the error .After deciding these values now you should go for the gain constants the highest should always be for the Proportional constant then the derivative constant and last the integral constant. The maximum change we can make is 100 so even the error should not exceed 100 and without the error going into the PID loop itself the error is 25 so the maximum gain constant used is 4 (25 x 4 = 100) but we again want to further divide the maximum gain constant among Gd Gp Gi so as I previously told we would want the gain constant for proportional value to be high so we set it to 2.5 then derivative to 1 and integral to 0.5 now we need to keep tuning these values until we get the best behaviour from our line follower it is advised to have these values read in the start so that we don’t need to connect it to a computer every time we need to change the values.  
  
General Tip: - set the proportional value to around 60% to 70% of the change we can make   
  
I have covered most of the aspects but many have been simplified finally here we end the ‘**PID Thingy**’