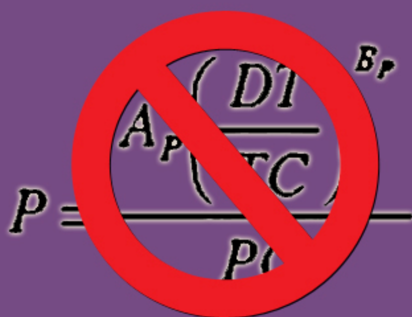
BY FINN PEACOCK

An Idiot's Guide

To the PID Algorithm

HOW THE ALGORITHM WORKS AND WHY IT WORKS.

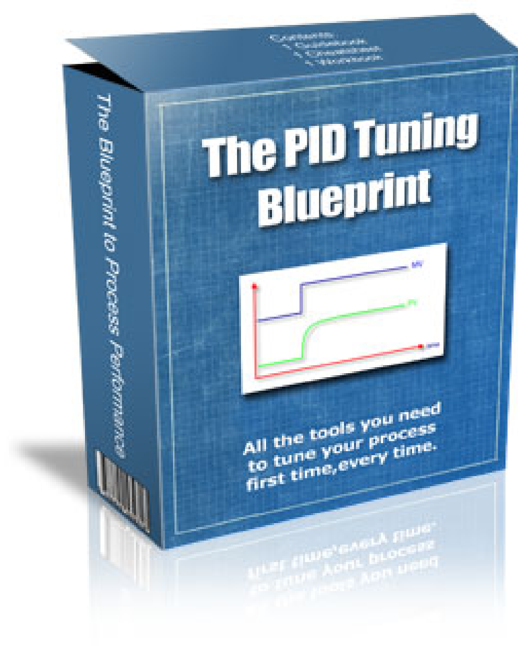


equation free

IN PLAIN ENGLISH!

From the author of the Idiot's Guide to the PID Algorithm:

"Learn How To Tune PID Loops First Time, Every Time"



Proven to tune:

- ✓Temperature Control
- ✓Flow Control
- ✓Pressure Control
- ✓Level Control
- ✓pH Control
- ✓Speed Control
- ✓Servo Control
- ✓plus many more...

Learn more at: www.PIDtuning.net

> "Its power and utility is embedded in its simplicity.
>
> **Your Blueprint launched our process from "expect the worst, but hope for the best" process to one that is reliable and predictable."**
>
> Anthony Keefe
> Michigan, USA

CONTENTS

Calculus with no math 2
Derivatives 2
Integrals.. 6
Feedback Control 11
Understanding the controller 13
Proportional control 16
Integral Action to the rescue 20
Derivative Action - predicting the future 23
Summing it all up 24

© 2008 Finn Peacock – www.PID-Tuning.com

Calculus with no math

When did you last use calculus on the job? My guess is never.

So why do all the control theory textbooks bombard us with equations and expect us to understand how that relates to our real world processes?

In these days of computer aided design and analysis tools – the need to solve a differential equation has been all but eliminated. This means that most engineers have lost touch with the concept of calculus, and how it applies to the real world.

This is a shame because in many engineering disciplines and particularly in process control – the ability to visualise a problem mathematically is what really separates the real pros from the rest of the crowd.

In fact even the humble PID contains those scary sounding calculus terms Integral and Derivative.

This section aims to give you a feel for what these terms really mean. This will give you a real edge when understanding what's going on in your controller.

Derivatives

Go into the control room of a process plant and ask the operator:

"What's the derivative of reactor 4's pressure?"

And the response will typically be:

"Bugger off smart arse!"

However go in and ask:

"What's the rate of change of reactor 4's pressure?"

And the operator will examine the pressure trend and say something like:

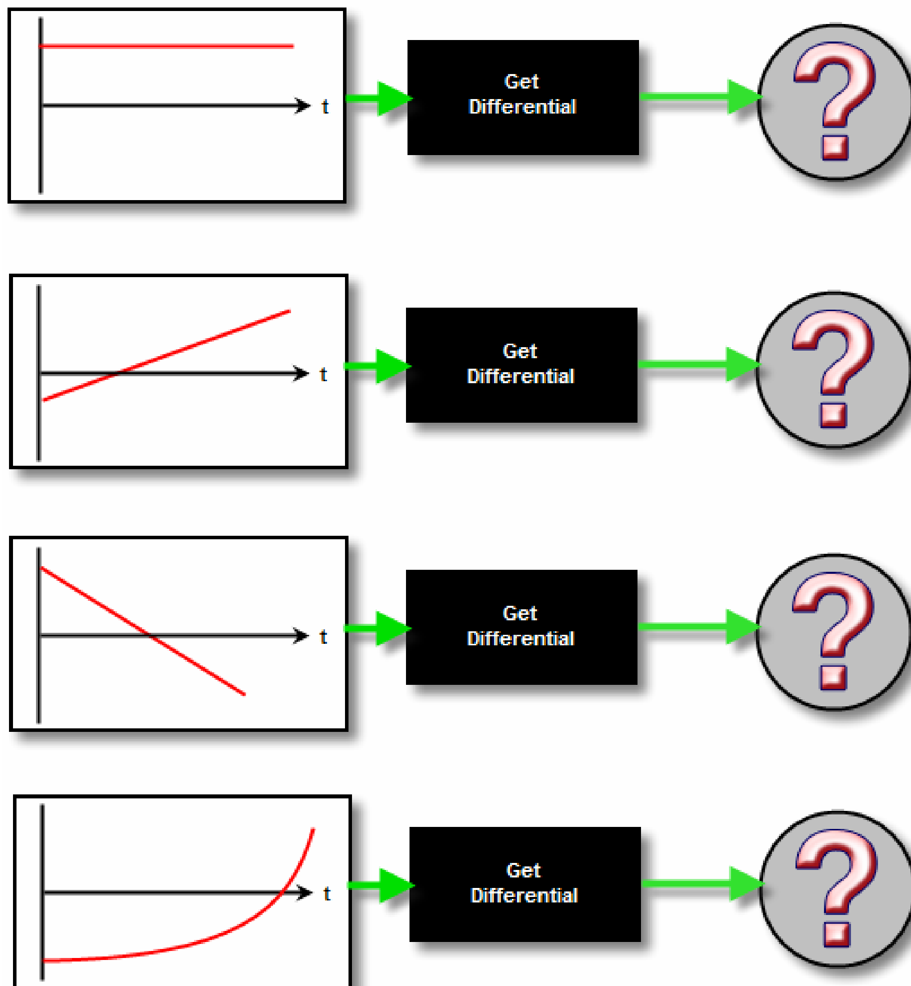
"About 5 PSI every 10 minutes"

He's just performed calculus on the pressure trend! (don't tell him though or he'll want a pay rise)

So derivative is just a mathematical term meaning rate-of-change. That's all there is to it.

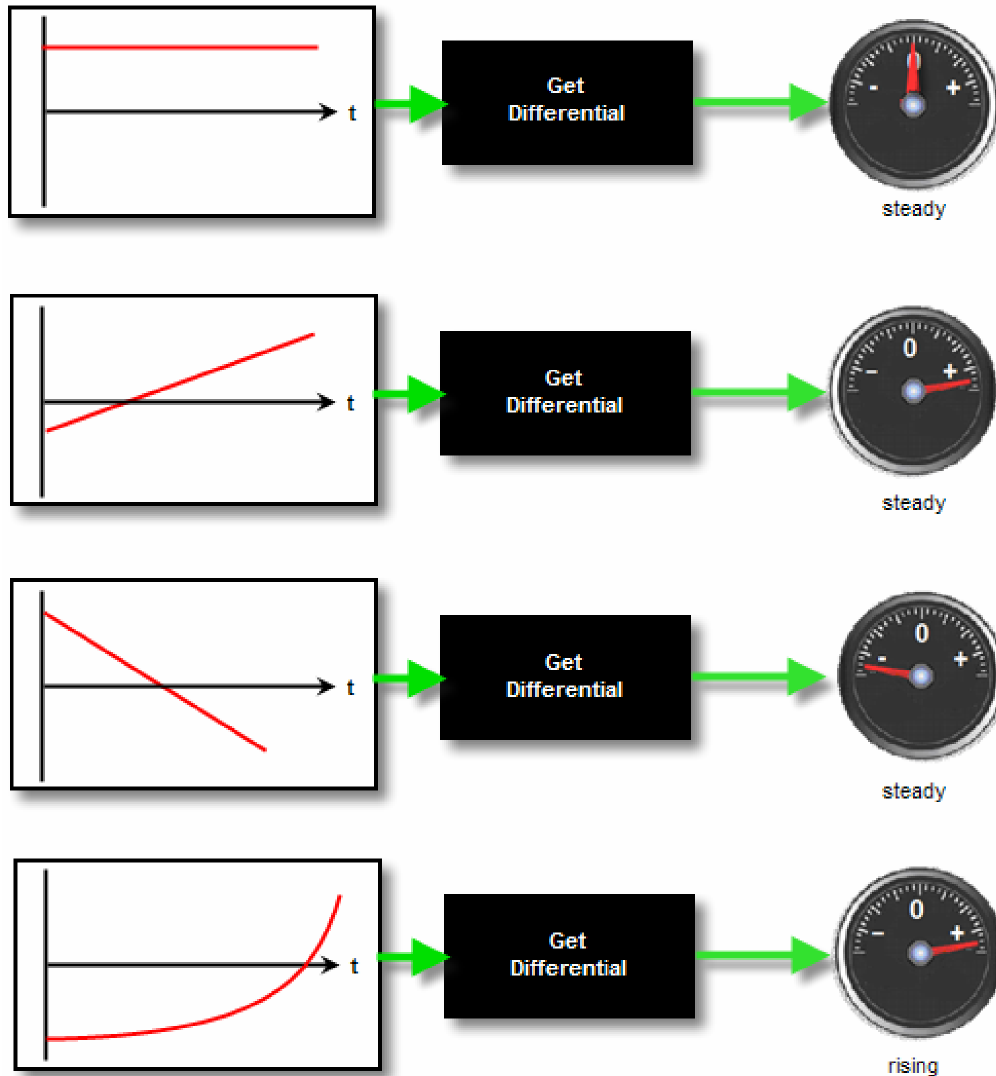
Testing your understanding
Suppose you have a box of electronics that calculates the derivative of its input signal. Its output is connected to an analogue meter which reads zero when vertical, negative to the left and positive to the right.

Look at the diagram below and draw on it where the meter would be pointing for each of the 4 input signals.



PTO for the answer…

© 2008 Finn Peacock – www.PID-Tuning.com



Were you right?

If not – remember that the absolute value of the input signal does not matter, all that matters is whether it is changing through time, and if so in which direction:

- If the input signal is not changing the output will be zero.
- If the input signal is increasing linearly, the output will be positive and stationary.

- If the input signal is decreasing linearly, the output will be negative and stationary.
- If the input signal is getting steeper with time, then the meter will be positive and moving to the right.

And so on.

If you understand these concepts, then you know everything you need to about differential calculus in order to understand the PID algorithm.

Integrals

Is it any wonder that so many people run scared from the concept of integrals and integration, when this is a typical definition?

Integral

From Wikipedia, the free encyclopedia

This article is about the concept of integrals in calculus. For other meanings, see integration and integral (disambiguation).

Integration is a core concept of advanced mathematics, specifically in the fields of calculus and mathematical analysis. Given a function $f(x)$ of a real variable x and an interval $[a,b]$ of the real line, the **integral**

$$\int_a^b f(x)\,dx$$

is equal to the area of a region in the xy-plane bounded by the graph of f, the x-axis, and the vertical lines $x = a$ and $x = b$, with areas below the x-axis being subtracted.

The term "integral" may also refer to the notion of antiderivative, a function F whose derivative is the given function f. In this case it is called an **indefinite integral**, while the integrals discussed in this article are termed **definite integrals**. Some authors maintain a distinction between antiderivatives and indefinite integrals.

The principles of integration were formulated by Isaac Newton and Gottfried Wilhelm Leibniz in the late seventeenth century. Through the fundamental theorem of calculus, that they independently developed, integration is connected with differentiation, and the definite integral of a function can be easily computed once an antiderivative is known. Integrals and derivatives became the basic tools of calculus, with numerous applications in science and engineering.

What the!?!?

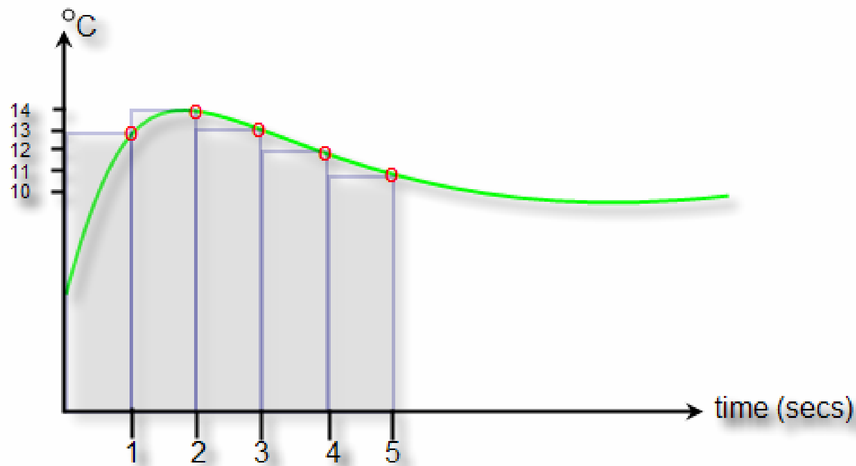
If you understood that you are a smarter person than me.

Here's a plain English definition:

The integral of a signal is the sum of all the instantaneous values that the signal has been, from whenever you started counting until you stop counting.

So if you are to plot your signal on a trend and your signal is sampled every second, and let's say you are measuring temperature. If you were to superimpose the integral of the signal over the first 5 seconds – it would look like this:

© 2008 Finn Peacock – www.PID-Tuning.com



The green line is your temperature, the red circles are where your control system has sampled the temperature and the blue area is the integral of the temperature signal. It is the sum of the 5 temperature values over the time period that you are interested in. In numerical terms it is the sum of the areas of each of the blue rectangles:

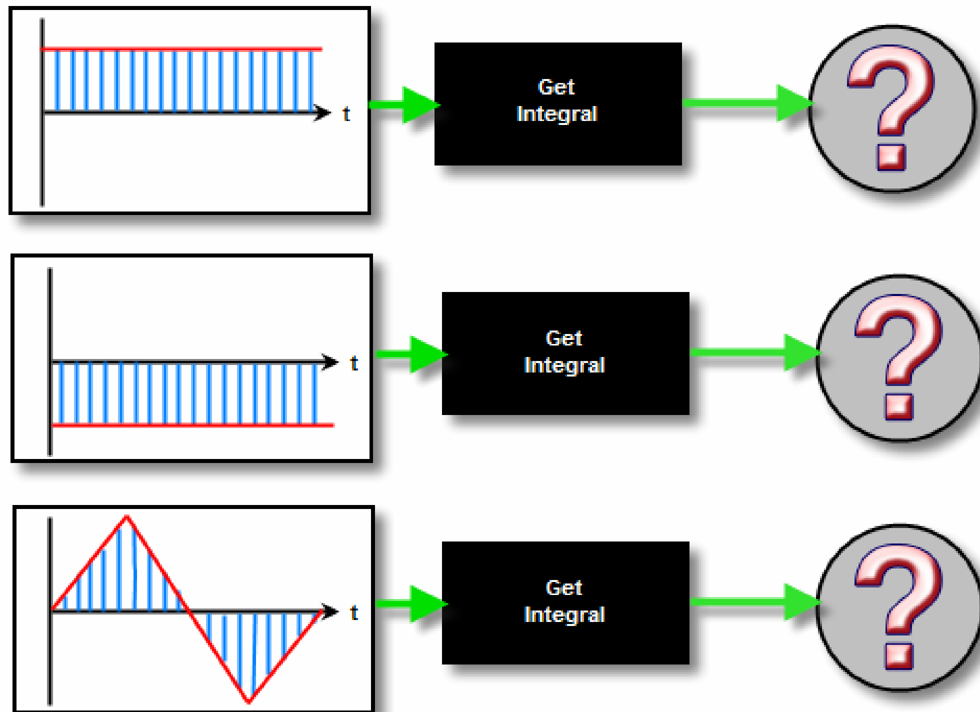
(13 x 1)+(14x1)+(13x1)+(12x1)+(11x1) = 63 °C s

The curious units (degrees Celsius x seconds) are because we have to multiply a temperature by a time – but the units aren't important.

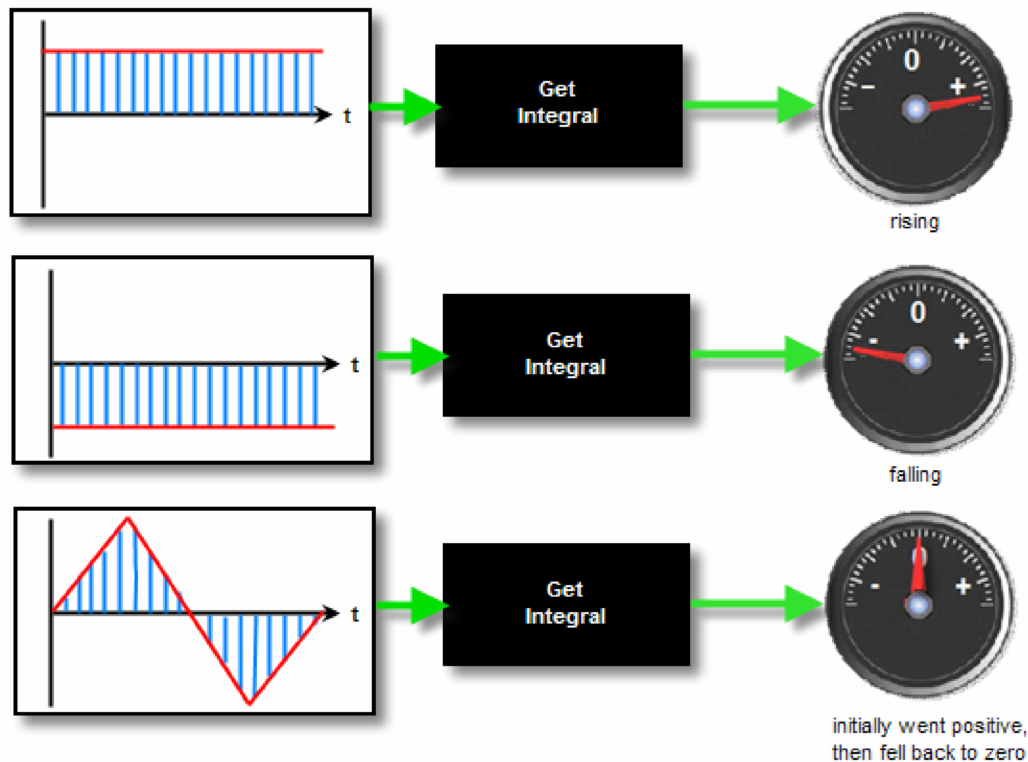
As you can probably remember from school –the integral turns out to be the area under the curve. When we have real world systems, we actually get an approximation to the area under the curve, which as you can see from the diagram gets better, the faster we sample.

So let's go back to our black boxes to check our intuitive understanding. This time we have boxes that take the input signal, calculate the integral, and output the value to a centre-zero meter.

What will the meters read for these 3 inputs?



PTO for the answer…



Were you correct? If not – remember that the meter will only be pointing to zero if the area under the signal is zero, or if the signal has covered exactly the same area above the x-axis as below it(because when it is below the x-axis the area counts as negative).

If the area under the signal is increasing – the meter will get more and more positive.

If the signal is only under the x-axis then the meter will get more and more negative as the signal clocks up more and more negative area.

Also if the signal is horizontal (i.e. constant) then the area is increasing at a uniform rate, so the output signal will do the same.

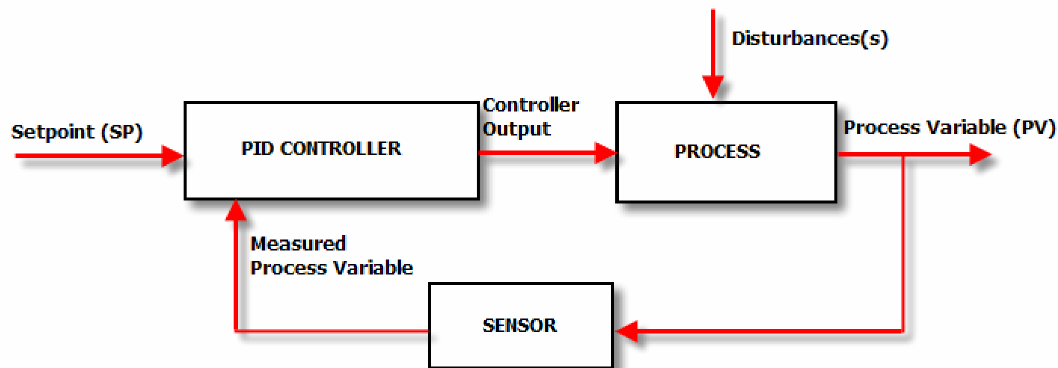
If the meter needle is stationary – all this tells us is that the input signal is at zero for that moment in time. If it is positive and stationary, then that tells us that it is zero at this moment, but in the past it must have been more positive than negative.

If you are happy with these concepts that you have all the understanding you need to understand how the PID algorithm works.

© 2008 Finn Peacock – www.PID-Tuning.com

Feedback Control

Here is the classic block diagram of a process under PID Control.



What's going on this diagram?

The **Setpoint** (SP) is the value that we **want** the process to be.

For example, the temperature control system in our house may have a SP of 22°C. This means that

"we want the heating and cooling process in our house to achieve a steady temperature of as close to 22°C as possible"

The PID controller looks at the setpoint and compares it with the actual value of the **Process Variable (PV)**. Back in our house, the box of electronics that is the PID controller in our Heating and Cooling system looks at the value of the temperature sensor in the room and sees how close it is to 22°C.

If the SP and the PV are the same – then the controller is a very happy little box. It doesn't have to do anything, it will set its output to zero.

However, if there is a disparity between the SP and the PV we have an error and corrective action is needed. In our house this will either be cooling or heating depending on whether the PV is higher or lower than the SP respectively.

Let's imagine the temperature PV in our house is higher than the SP. It is too hot. The air-con is switched on and the temperature drops.

© 2008 Finn Peacock – www.PID-Tuning.com

The sensor picks up the lower temperature, feeds that back to the controller, the controller sees that the "temperature error" is not as great because the PV (temperature) has dropped and the air con is turned down a little.

This process is repeated until the house has cooled down to 22°C and there is no error.

Then a disturbance hits the system and the controller has to kick in again.

In our house the disturbance may be the sun beating down on the roof, raising the temperature of the air inside.

So that's a really, really basic overview of a simple feedback control system. Sounds dead simple eh?

© 2008 Finn Peacock – www.PID-Tuning.com

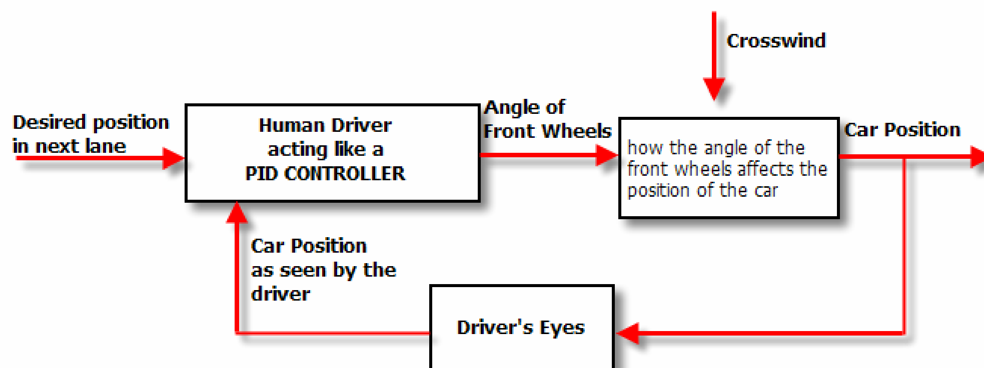
Understanding the controller

Unfortunately, in the real world we need a controller that is a bit more complicated than the one described above, if we want top performance form our loops. To understand why, we will be doing some "thought experiments" where we are the controller.

When we have gone through these thought experiments we will appreciate why a PID algorithm is needed and why/how it works to control the process.

We will be using the analogy of changing lanes on a freeway on a windy day. We are the driver, and therefore the controller of the process of changing the car's position.

Here's the Block Diagram we used before, with the labels changed to represent the car-on-windy-freeway control loop.



Notice how important closing the feedback loop is. If we removed the feedback loop we would be in "open loop control", and would have to control the car's position with our eyes closed!

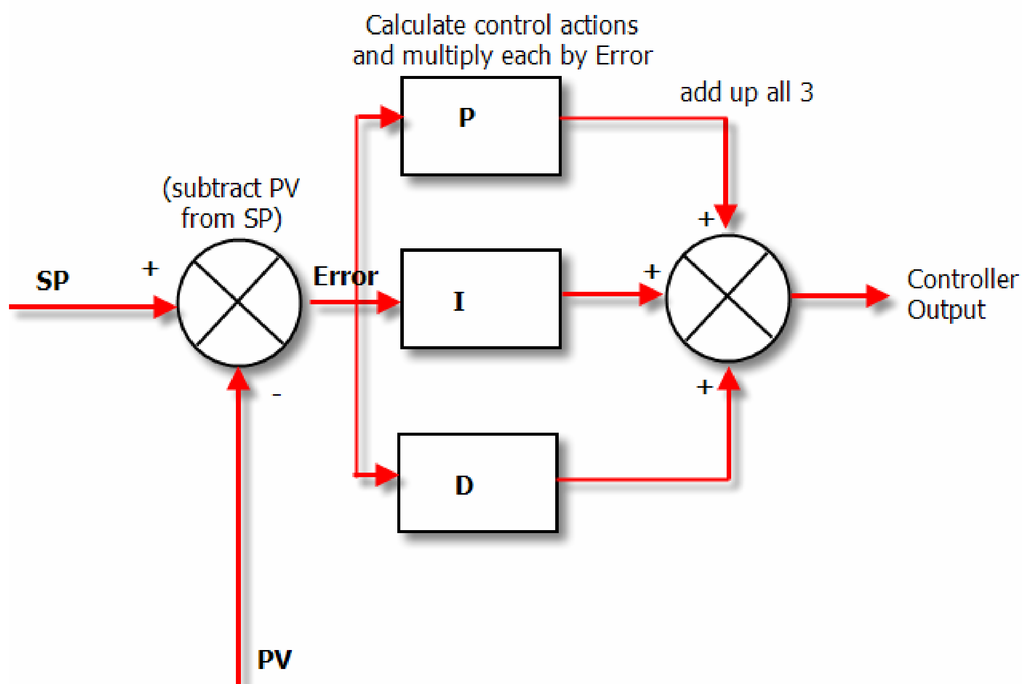
Thankfully we are under "Closed loop control" -using our eyes for position feedback.

As we saw in the house-temperature example the controller takes the both the PV and SP signals, which it then puts through a black box to calculate a controller output. That controller output is sent to an actuator which moves to actually control the process.

We are interested here in what the black box actually does, which is that it applies 1, 2 or 3 calculations to the SP and Measured PV signals. These calculations, called the "Modes of Control" include:

- Proportional (P)
- Integral (I)
- Derivative (D)

Here's a simplified block diagram of what the PID controller does:



It is really very simple in operation. The PV is subtracted from the SP to create the Error. The error is simply multiplied by one, two or all of the calculated P, I and D actions (depending which ones are turned on). Then the resulting "error x control actions" are added together and sent to the controller output.

These 3 modes are used in different combinations:

P – Sometimes used
PI - Most often used
PID – Sometimes used
PD – rare as hen's teeth but can be useful for controlling servomotors.

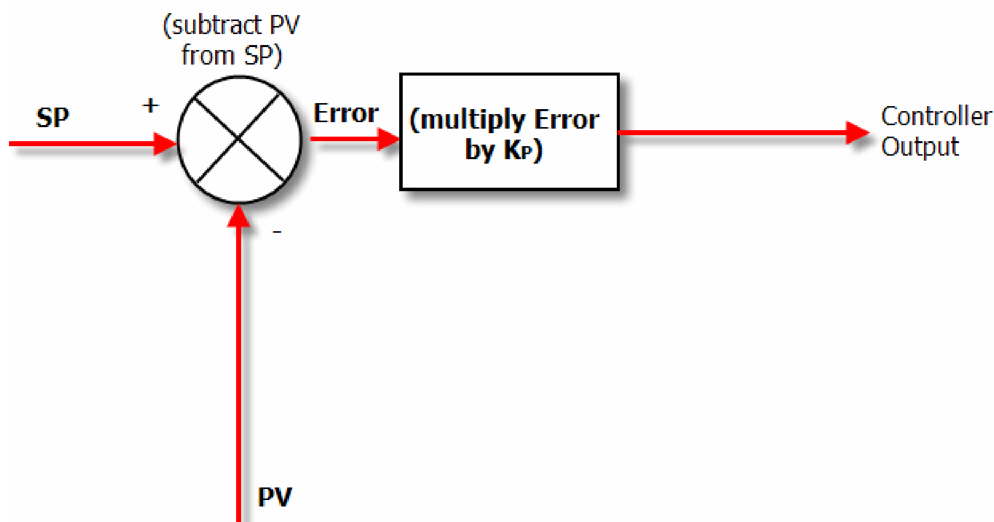
© 2008 Finn Peacock – www.PID-Tuning.com

So let's now get an intuitive feel for how each control action works. In other words let's understand what goes on in the P, I and D boxes in the block diagram above.

To do this we will be using our analogy of driving a car on a motorway, and using the principles of PID control to move the car into another lane.

Proportional control

Here's a diagram of the controller when we have enabled only P control:



In Proportional Only mode, the controller simply multiplies the Error by the Proportional Gain (Kp) to get the controller output.

The Proportional Gain is the setting that we tune to get our desired performance from a "P only" controller.

We have already pointed out that driving a car with your eyes open is analogous to feedback control where you are the controller. To take this analogy further, think about when you change lanes on the motorway. Everything is instinctive of course, but if you analyse your actions the course of events goes something like this:

You have to first choose an initial steering angle (in reality you slowly increase the steering angle - but for the sake of this thought experiment - assume that you quickly turn the steering wheel to an immediate initial angle)

The car immediately starts to move towards the centre of the next lane. Now you don't want to wait until you are in the centre of the next lane to start correcting your steering, so what you intuitively do is reduce the steering angle as you get closer to your target position in the next lane.

i.e. as your "position error" reduces, so does your "control action". Refer to the diagram of the P-Only controller above if this is not intuitive to you: as the error reduces, so must the controller output.

So the safest approach to changing lane successfully is to start with a small angle and slowly reduce it as you get closer to the next lane.

The problem with this is that – if you are in a hurry to get into that lane and you choose a really conservative (i.e. small) angle initially, you will take a long time to get there.

It is the same with a proportional controller – a small proportional gain (Kp in the diagram) is the safest way to get to setpoint – but your controller performance will be slow.

Improving your lane changing performance

OK so you need to get into the outside lane before the approaching Porsche slams into your rear. You wrench the wheel to a really steep angle, and start correcting as you get closer, but , holy crap, the your momentum has taken you over the centre of the next lane and you are rapidly approaching the opposite side of the road. Because you are consistent (just as a PID controller is), you wrench the wheel in the opposite direction just as hard as before, and guess what, now you've over shot back to the other side of the lane. Before you know it you've gone unstable! You are in a vicious cycle of lurching from one side of the lane to the other and back again.

That's why too much P gain in a control loop sends it unstable: the actuator doesn't back off quick enough, you overshoot, and then make the exact same mistake going the other way, and on and on.

So the trick is to find a P gain that is high enough to give you the speed of response that you need, but not so high that you go unstable.

So why don't all control loops use a P only controller that is beautifully tuned not to overshoot the setpoint and leave it at that?

The answer is "disturbances".

Disturbances in a control loop require more that just P

So being a good driver you have found the perfect angle for changing lanes nice and quickly – but without going unstable. You are driving along feeling pretty content with yourself. Every time you change a lane you move the wheels to just the correct initial angle, and as you drift over to the next lane you start reducing your angle the closer you get to your final position. As your 'position error' goes to zero, your steering angle goes to zero too, and you end up in exactly the right position every time. AC/DC is on the car stereo and life is good…

But then you notice a sign warning of extreme crosswinds for the next 10km just as you decide to change lanes again.

You turn your wheel to the right - to your pre calculated initial angle - and find that the car only moves a little to the right before continuing straight on. The cross wind is so strong that as you were moving closer the target lane and reduced your steering angle, the smaller steering angle was being fully counteracted by the strong crosswind, you are still moving in a straight line although you are steering slightly to the right.

The result is that you are offset from your target position by a fixed amount (the position error). And because the rules are that the steering angle must be proportional to the position error, (we are behaving as a P controller remember) you are a bit stuck. You can't change the steering angle because the error isn't changing – and the steering angle only changes when the error changes…

We need some new rules for steering this thing!

What would happen in practice is we would use trial and error under these new cross wind conditions to get the initial angle just right - so that once again we end up in the perfect position. Lets assume that we 'bias' our steering angle in this way to take account of the crosswinds.

But then there is a downhill and we start speeding up, and find that our pre chosen angle over shoots (because the faster we are going the faster we will move across the highway). So we try trial and error again, and just as we get the steering angle bias correct for our new speed, the gradient changes and we start slowing down. What a bummer! It's time to "bias" our angle again.

Let's think about what is happening here – we have a "P only" control loop that was working beautifully, until some disturbances started hitting us. We had to manually bias our initial steering angle each time the disturbance changed.

© 2008 Finn Peacock – www.PID-Tuning.com

This is analogous to a "P only" control loop where we are continually changing the P gain to cope with changing disturbances. Not an optimal situation. We call this changing of the gain "Manual Reset" because we have to manually reset the gain whenever a disturbance changes.

Wouldn't it be nice if we could "automatically reset" the gain?

© 2008 Finn Peacock – www.PID-Tuning.com

Integral Action to the rescue

Guess what… Integral action is also called "Automatic Reset". Hmm guess what it does?

That's right it automatically 'resets' the bias of the gain until the error is zero.

Back to our car with human controller.

So you are driving along with your wheels at a fixed steering angle, but because of the strong crosswind you are moving straight ahead. But you want to be moving sideways to your target position in the next line. What do you do? You do exactly what the Integral term in a PID controller does:

You start to increase your steering angle, and you keep increasing it until you start moving sideways.

This is where integral action is used to overcome the deficiencies of proportional action. Proportional action is simply the "P Gain" multiplied by the error. If the error is holding constant, but you aren't at your target setpoint yet, you will be stuck. This is called an **offset**, and is the problem with P only control.

The addition of Integral action overcomes this deficiency. If there is an error between the SP and the PV; integral action will start to ramp up the controller output until things start moving again.

Mathematically, it "increases the controller output by the Integral of the error". What does this mean? Remember our definition of Integration as:

The integral of a signal is the sum of all the instantaneous values that the signal has been, from whenever you started counting until you stop counting.

Translating this to a control system, it means that the integral action will simply start adding up all the error values, resulting in a ramping signal if the error is non-zero. Of course to make sure that you ramp the error just enough, you need to very carefully tune the "I constant" – but that is what the PID Tuning Blueprint is for.

Pretty straightforward eh?

So now we have pretty good control of our car steering. We have a P term which immediately looks at the instantaneous error between where we are and

© 2008 Finn Peacock – www.PID-Tuning.com

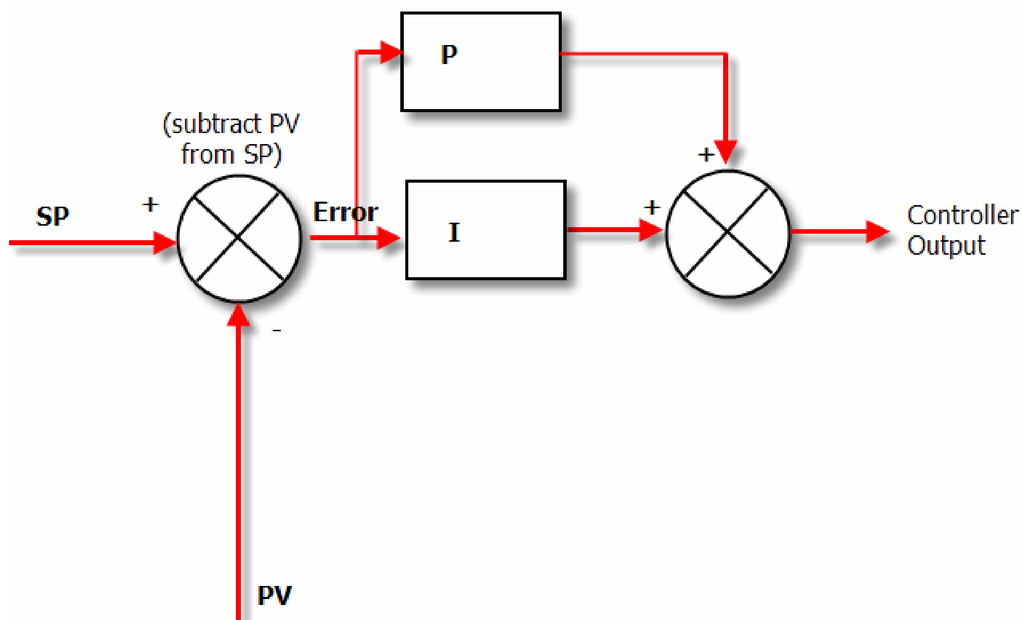
where we want to be and provides us with a good initial angle to turn our wheels to.

As we get closer to our target position, the P action reduces the steering angle as the error decreases.

In parallel with the P action, the "I action" starts to 'bias the steering angle' by continually adding to the steering angle as long as there is a position error. When there are no disturbances, the integral action has the effect of simply making the steering angle a bit steeper, improving our 'controller performance' by reducing the amount of time it takes to get to our destination.

However where our "I action" makes a big difference is where we have a disturbance, such as crosswind. In that situation, our "I action" continues to ramp up our steering angle until we have enough angle to 'break through' the crosswind and continue to our destination.

The Diagram below that shows how the algorithm in a PI controller is calculated.



Adjusting the Integral Action

The way to adjust how much Integral Action you have is by adjusting a term called "minutes per repeat". Not a very intuitive name is it?

So where does this strange name come from? It is a measure of how long it will take for the Integral Action to match the Proportional Action.

In other words, if the output of the proportional box on the diagram above is 20%, the repeat time is the time it will take for the output of the Integral box to get to 20% too.

And the important point to note is that the "bigger" integral action, the quicker it will get this 20% value. That is, it will take fewer minutes to get there, so the "minutes per repeat" value will be smaller.

In other words the smaller the "minutes per repeat" is the bigger the integral action.

To make things a bit more intuitive for hairy arsed engineers like you and me a lot of controllers use an alternative unit of "repeats per minute" which is obviously the inverse of "minutes per repeat".

The nice thing about "repeats per minute" is that the bigger it is - the bigger the resulting Integral action is.

© 2008 Finn Peacock – www.PID-Tuning.com

Derivative Action – predicting the future

OK, so the combination of P and I action seems to cover all the bases and do a pretty good job of controlling our system. That is the reason that PI controllers are the most prevalent. They do the job well enough and keep things simple. Great.

But engineers being engineers are always looking to tweak performance.

Imagine you are back in your car. You are in a rush to get back home before the football starts. So you put your foot down. After a while you need to change lanes again to pass some slow-coach. You move your wheels to the pre-set angle that worked so well earlier. However, because you are going faster, you start to move across the lane quicker. You end up overshooting the target lane and have to turn the wheels in the other direction to get back on track. You may get into the target lane eventually, but the maneuver wasn't as slick as you had come to expect.

Imagine if, immediately after you had turned the wheels, the car had (Knight Rider style) worked out that you were moving across the lanes a bit too fast and that at this rate you would overshoot the target unless the angle was reduced. Imagine if the car had then taken it upon itself to correct your steering angle, reducing it so that you had minimal, if any, overshoot.

Well that's the idea behind Derivative action. As you might have guessed, derivative action uses the rate-of-change of the error to calculate its contribution to the controller output. So let's think through how that actually works in the above example.

When the steering wheel is turned, the position-error of the car starts to reduce rather quickly as you get closer to the target lane. In other words, the rate of change of the error is large and negative. This manifests itself as a negative derivative action, which works to reduce the steering angle which in turn will reduce the overshoot. The trick, of course is to have just enough D to remove the overshoot (and that's where proper tuning comes in – trial and error just doesn't cut it).

So adding derivative action can allow you to have bigger P and I gains and still keep the loop stable, giving you a faster response and better loop performance.

If you think about it, Derivative action improves the controller action because it predicts what is yet to happen by projecting the current rate of change into

© 2008 Finn Peacock – www.PID-Tuning.com

the future. This means that it is not using the current measured value, but a future measured value.

The units used for derivative action describe how far into the future you want to look. i.e. If derivative action is 20 seconds, the derivative term will project the current rate of change 20 seconds into the future.

The big problem with D control is that if you have noise on your signal (which looks like a bunch of spikes with steep sides) this confuses the hell out of the algorithm. It looks at the slope of the noise-spike and thinks:

> "Holy crap! This process is changing quickly, lets pile on the D Action!!!"

And your control output jumps all over the place, messing up your control.

Of course you can try and filter the noise out, but my advice is that, unless PI control is really slow (even after tuning with this blueprint), don't worry about switching D on.

Summing it all up

The Proportional action of a PID controller, simply gives you a controller output that is proportional to the instantaneous error. It will always leave you with a "steady state error" if there are disturbances in your system. In other words if there are changing disturbances and you only have Proportional control, you will never get to your setpoint.

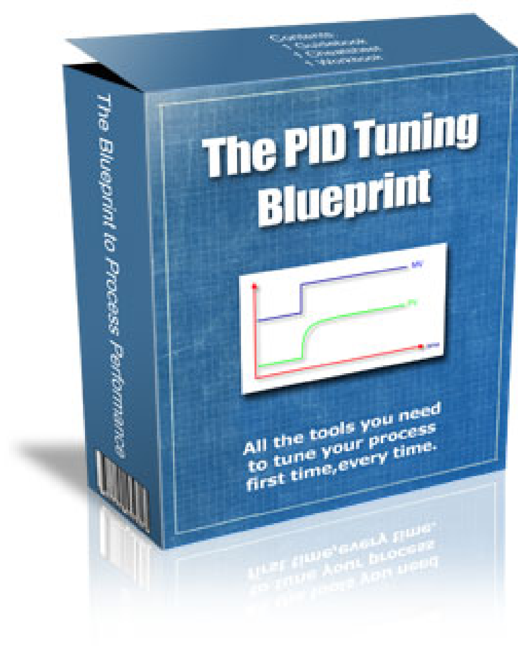
To remove this steady state error we can use Integral action, which keeps adding (integrating) all the values of the error over time to the controller output allowing it to overcome those pesky disturbances.

Derivative action allows us to have bigger P and I actions and still keep the loop stable. It does this by looking at the rate of change of the error and pulling the controller action back if it can see that this rate of change will overshoot the setpoint in the near future. The result (if carefully tuned) is a faster, better performing control loop. However noise on the measurement signal makes Derivative action very difficult to get right.

© 2008 Finn Peacock – www.PID-Tuning.com

From the author of the Idiot’s Guide to the PID Algorithm:

"Learn How To Tune PID Loops First Time, Every Time"



Proven to tune:

- ✓Temperature Control
- ✓Flow Control
- ✓Pressure Control
- ✓Level Control
- ✓pH Control
- ✓Speed Control
- ✓Servo Control
- ✓plus many more...

Learn more at: www.PIDtuning.net

"Its power and utility is embedded in its simplicity.

Your Blueprint launched our process from "expect the worst, but hope for the best" process to one that is reliable and predictable."

Anthony Keefe
Michigan, USA