PID Tuning

APRIL 26, 2017 ~ MATT

PID tuning is a fairly quick and easy process that can make a big difference to your print quality. If you can see while you're printing that your extruder temperature is fluctuating up and down by a few degrees then a quick PID tuning can help to make it rock solid. Temperature fluctuations can even induce visible banding into the prints in extreme cases. The optimal P, I and D values are heavily dependent on the environment that you're printing in, which is why its important to customise these yourself and not just run off the firmware defaults.

Tuning

The process starts by connecting your computer to your printer so that you can send individual G-Code commands. You can do this using a program such as Pronterface, Repetier Host, Simplify 3D, or the Octoprint web interface. Then we run a process which is referred to in Marlin firmware as PID autotuning. This is done with the M303 command.

Use the command `M303 S[temperature] U1` where `[temperature]` is the extruder temperature that you will usually be printing at. If you print with multiple materials at different temperatures that's fine, the temperature value is just to set a target for the extruder to calibrate at. This will cycle through heating up and partly cooling down 5 times and then return the final Kp, Ki, and Kd values when complete.
If your printer uses PID on the heated bed too, this process can also be done for that heater. Simply include `E-1` in the command so that it calibrates the heated bed instead. So the command would look like `M303 E-1 S[temperature] U1`. Usually the process would be to then enter these values in the printer settings, but this is already done for us with `U1` in the initial command. Now all we need to do is save it with `M500` and its all done!… Except for Original Prusa users.

If you're doing this on a printer running Prusa firmware, the `U` flag of the M303 command hasn't yet been integrated. I've added a request for it here, but in the mean time that means you'll need to enter the values back into the printer settings. Upon completion, the PID process should have printed out something like this:

```
Kp: 17.54
Ki: 1.18
Kd: 65.39
PID Autotune finished! Put the last Kp, Ki and Kd constants from above into Configuration.h
```

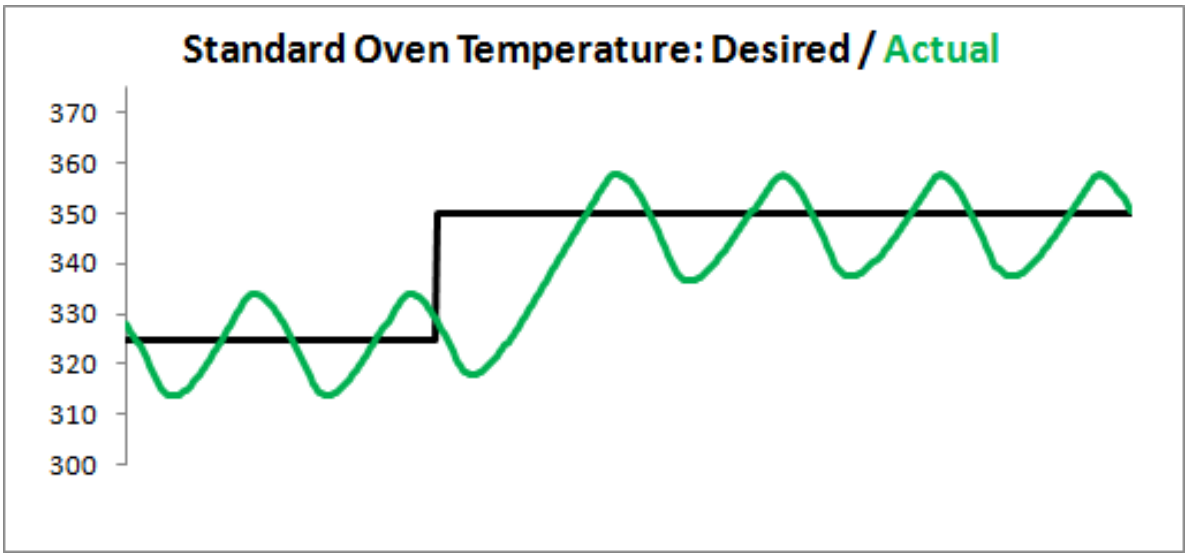
We won't be entering these into Configuration.h, as there's a much easier way to set them. Instead we can store them in EEPROM, which is the printer's persistent memory. Do this using the command `M301 P##.## I#.## D##.##`, where the hashes after `P` are replaced with your Kp value, the hashes after `I` are replaced with your Ki value, and the hashes after `D` are replaced with your Kd value. You can see here that I've taken the Kp, Ki, and Kd values from the M303 output and put them into the M301 command – `M301 P17.54 I1.18 D65.39`. The same can be done if you're calibrating your bed instead by using `M304` instead of `M301`. Once you've sent that, save it with `M500` and your PID tuning is complete. You should now experience much more steady hot end temperatures. If you want to learn more about how PID works, it's all explained below.

Theory

PID stands for "Proportional, Integral, Derivative," and is a control loop feedback mechanism. The system continuously calculates an error value between the desired point and the current measurement. It then applies a correction that is proportional to the error value with the aim to achieve the desired point. In our case, this is applies to the temperature of the hot end. The P value accounts for the present error value – so if the error is large, the output will also be large. The I accounts for the past error values, and it sort of acts as the learning system for

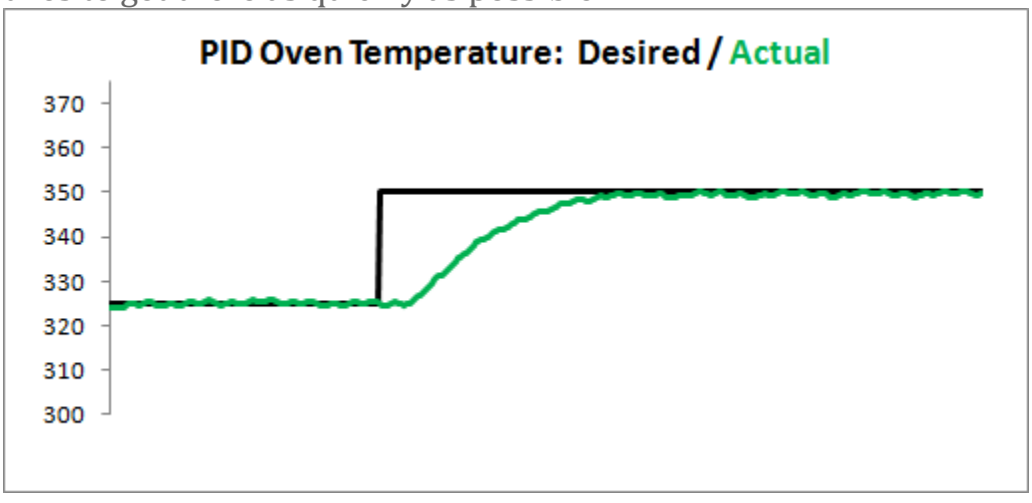
the equation. For example if the output is not sufficient, the error will accumulate over time and the system will respond by applying a stronger output. The D value accounts for future trends of the error value, based on its current rate of change. All of these values are considered to produce an output at that specific point in time that will most efficiently reach the target value.

The alternative configuration to this is a system called **bang-bang**. Rather than producing a proportional output, bang-bang is either on or off. PID is almost always used for the hot end, but bang-bang is sometimes used for the heated bed. This is because the beds usually have a larger thermal mass, and take longer to heat up or cool down. As a result of this, fine temperature control isn't required as the temperatures won't fluctuate massively anyway. Bang-bang works by checking the temperature every 5 seconds to see if the heater needs to be on or off based on whether the current temperature is higher or lower than the target. It's also slightly more complicated than this though, due to a concept called **hysteresis**. Hysteresis in this context can be thought of as the buffer in the temperature so that the heater isn't constantly switching on and off too frequently. The value set in firmware as BED_HYSTERESIS determines how big this buffer is, in degrees. For example if it was set to 2, then the heater would only switch on if the temperature reading was below target – 2° and the heater would only switch off if the temperature reading was above target + 2°. This is useful for relay switches that aren't able to use **pulse width modulation**, which is the way that variable power levels from PID are set on the heaters. The graphs included below were taken from ospid.com. They refer to oven temperatures, but the concept is exactly the same as our hotends.



This graph depicts a bang-bang temperature control system in use. You can see the large fluctuations in temperature that occur here. This would be the case if bang-bang was used on a hotend due to the low thermal mass, but on a heated bed it would be much more stable.

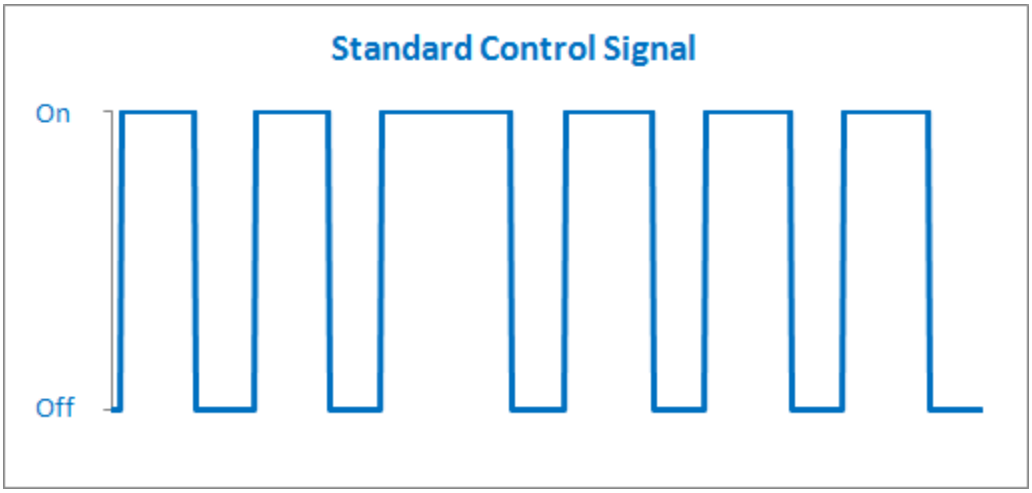
PID implementation in Marlin on the other hand works by first keeping your heater on 100% power until it gets to within 10 degrees of the target temperature (determined by `PID_FUNCTIONAL_RANGE` in firmware), and then the PID controlling takes over, moderating the power output as it gets closer and closer to the target temperature. This is designed to reduce overshoot (when it goes too far above) when reaching the target temperature, but at the same time tries to get there as quickly as possible.



In this PID controlled system, you can see how the temperature steadily rises with

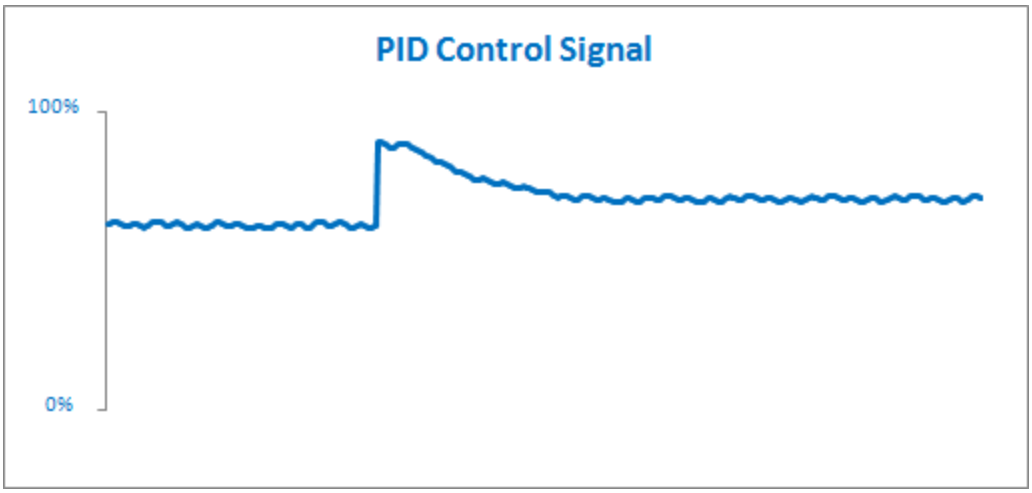
only minor fluctuations all the way through. It also shows how overshoot is counteracted by gently approaching the target temperature.

In these next graphs, you'll be able to see a depiction of the power output to the heater, instead of just the resulting temperature change. You can see in the bang-bang system (labelled as 'standard' in the graph) that it is constantly switching on and off. This is what causes the large fluctuations in the temperature graph that you saw above.



Bang-bang controlled system power output

In the PID tuned output however, you can see how the heater is given 100% power until it gets close to the target temperature, and then begins to gradually settle back down to a constant power output.



PID controlled system power output

Hopefully this guide has given you a good insight into PID tuning and the two most common temperature control systems! There are plenty of resources online if you'd like to learn more. Reading through the temperature related sections of the Configuration.h and Configuration_adv.h files of Marlin firmware also helped my understanding a lot. Here are some further references if you'd like to do some more reading.