Deep Dive

Introduction

The purpose of this page is to provide the reader with detailed information about the inner workings of the BetaFlight firmware. This information has been collected from sources such as:

- The BetaFlight RC Groups Forum (credit will be given where possible).
- YouTube Channels that have provided relevant content.

Grab a snack and make yourself comfortable !

Contents

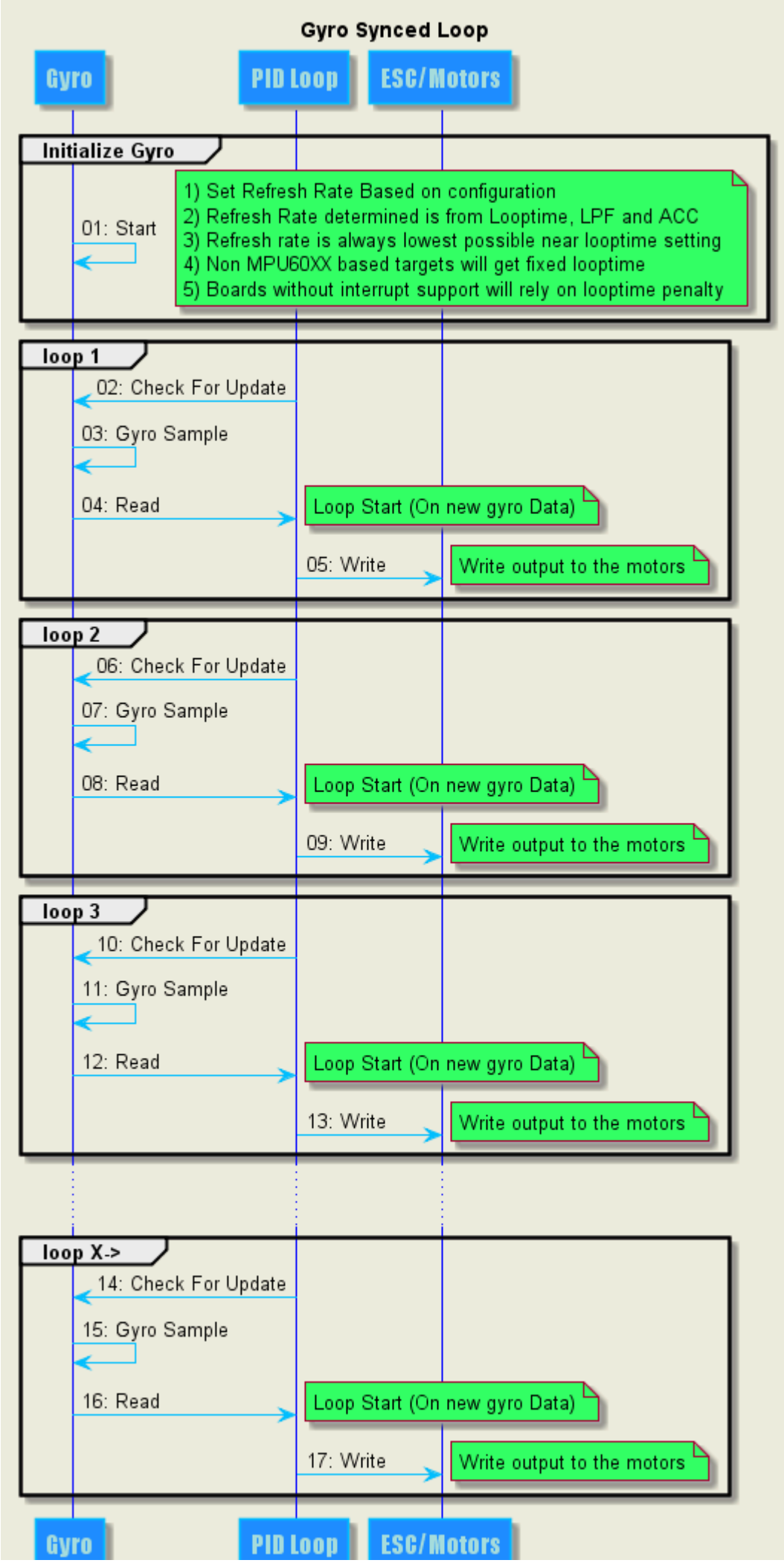
How Opensource Software Development Works

This video covers how multiple versions of the same software in the hobby exist (CleanFlight/BetaFlight/RaceFlight etc) and how developers exchange ideas and promote code between the projects.

http://www.youtube.com/watch?v=kZvoei1dzNQ

Gyro based loop implementation

Gyro update is leading the loop. The loop will start after interrupt is triggered for new gyro sample. The PID controller will always be doing the calculation of the most fresh gyro value. The sampling gyro rate of 1khz will be used and that will automatically run looptimes of 1000us or 500us depending of configuration and target capabilities. This also makes the looptime setting unnecessary. There is no need for this parameter as our gyro decides when loop will run. There is no drift between gyro and control loop and your PID tune will be consistent. No aliasing should be experienced. This also helps filters to do better job in giving clean gyro traces.


Gyro Synced Loop
Gyro
PID Loop
ESC/Motors
Initialize Gyro
01: Start
1) Set Refresh Rate Based on configuration
2) Refresh Rate determined is from Looptime, LPF and ACC
3) Refresh rate is always lowest possible near looptime setting
4) Non MPU60XX based targets will get fixed looptime
5) Boards without interrupt support will rely on looptime penalty
loop 1
02: Check For Update
03: Gyro Sample
04: Read
Loop Start (On new gyro Data)
05: Write
Write output to the motors
loop 2
06: Check For Update
07: Gyro Sample
08: Read
Loop Start (On new gyro Data)
09: Write
Write output to the motors
loop 3
10: Check For Update
11: Gyro Sample
12: Read
Loop Start (On new gyro Data)
13: Write
Write output to the motors
loop X->
14: Check For Update
15: Gyro Sample
16: Read
Loop Start (On new gyro Data)
17: Write
Write output to the motors
Gyro
PID Loop
ESC/Motors

The delta_from_gyro setting and all about the PID Controller D values

Boris B wrote:

On = delta from measurement / gyro

Off = delta from error

It is really hard to explain without reading a lot about pid controllers and how they really work.

But here's my attempt for a quick and "easy" to understand explanation:

Before explaining I will first tell you what determines the Dterm part of PIDsum. It is the derivative or, in other words, rate of change. It is basically the speed of something.

Lets say you're driving at a constant speed of 50km/h, then your speed derivative is 0. But when you push the pedal and keep accelerating and start speeding up with 5km/h per second then the derivative of your speed is 5. When you start slowing down, your speed derivative goes negative.

Delta from error: This is the classical Derivative approach. It reflects the speed of error. Error is derived from stick input. Once you move the stick you introduce an error and PID controller has to correct the error. The faster the error grows the bigger derivative gets. It helps PID loop to reach target faster. It is accelerating it more what creates the dampening effect.

Delta from measurement/gyro: This is mathematically simplified formula for Derivative from error assuming that error is 0. When you leave out error out of the formula as it is 0 anyway that there is just -(measurement delta) left. And this is the speed of change from gyro. The faster the gyro grows in positive direction the higher Dterm part will be in opposite direction. This has a direct dampening effect to P.

The main difference is that Delta from error kind of accelerates your stick input, but also creates derivative kick because it follows the stick which is slower refresh rate and delta from gyro doesn't accelerate your stick input at all and doesn't create derivative kick.

ctzsnooze wrote:

Actually, Dterm in gyro mode is inversely proportional to the rate of change of the gyro signal. It always opposes changes to the gyro signal. Frame inertia delays the response of the gyros to P changes, but since Dterm always opposed gyro changes, it typically is opposite in sign to Pterm (definitely opposite in sign to Dterm.

When Pterm increases due to positive stick inputs, inertia of the quad initially results in a big Pterm error signal. Initially, the quad isn't moving, due to inertial mass. With no movement, Dterm is zero, while Pterm is quite large. Very quickly the motors spool up and start to rotate the quad, reducing Pterm as the rotation rate reaches target (i.e. as error falls). Dterm slows down the onset of the rotation, smoothing out both the start of the rotation and the end of the roll. But Pterm is always stronger, so the quad eventually gets the desired rotation rate. Because the rate of rotation is now constant, both Pterm and Dterm head to zero. Momentum may cause the frame to keep rotating a bit past the point at which it was supposed to stop. P weakly opposes this since typically the actual magnitude of the offset due to momentum errors is small, but Dterm is very sensitive to the high velocity of Pterm overshoots and damps them away much more quickly than Pterm can.

In the absence of stick movement, Pterm opposes an external force on the frame. The magnitude of the Pterm response is always in proportion to the error between the intended rotation rate and the measured rotation rate (e.g. unwanted rotation from flying into a gust of wind).

When there is a positive stick movement, the positive rcCommand will result in a sudden negative error because the frame won't be rotating as fast as the new value for rcCommand requests - at least immediately it won't. That causes a small spike up in Pterm that can be seen with every positive rcCommand step. This is what it means to say that P term is error based . As a result, PTerm changes in the same direction as rcCommand with stick inputs. Once the frame starts to rotate at the desired rotation rate, the Pterm error falls, and reaches zero when it is rotating at the desired speed.

When Dterm is gyro based, it is proportional to the rate of change of the gyro signal, and opposite in sign. A non-error based Dterm doesn't care about stick inputs. It just opposes the rate of change of gyro - smoothing out sudden changes in rate, however they arise. While mathematically a derivative, it acts as a damping mechanism for oscillations, like the shock absorber in a car's suspension.

While a gyro based Dterm will be delayed a little behind Pterm due to frame inertia, it will typically be of opposite sign to Pterm during stick inputs.

ctzsnooze wrote:

Dterm can be calculated on rate of change of gyro signal alone, or the error from rcCommand to gyros.

The former approach was first used in Lux, whereas multiwii and Rewrite Dterm were historically error based.

Until recently, BetaFlight used error based Dterm calculations, but as of 2.3.4 Boris has gone back to the original Lux approach where Dterm is derived only from rate of change of the gyro signal.

Dterm fundamentally opposes changes in gyros, and opposes fast changes more than slow changes. If you hand hold your quad in acro mode and, without stick input, suddenly tilt it a tiny bit, but very, very quickly, the immediate opposition to that sudden small change is coming almost entirely from Dterm.

The two methods behave exactly the same in the absence of stick inputs, but differ markedly when a control input results in a sudden 'error' between current roll rate and the new intended roll rate.

When D is based on error, a strong 'dterm kick' happens when rcCommand suddenly changes the error value. The kick is usually in phase with the desired outcome, and speeds up the response to stick inputs, because the kick typically overrides the usual Dterm response of opposing movement. However the kick effect causes 'spikes' on the Dterm trace, smaller with Sbus but big on PPM. Dealing with those spikes was the main reason why rcSmoothing was implemented - it removes the spikes, but retains the 'kick' effect.

During a sudden stick input, the quad will respond a bit quicker if Dterm is error seeking because D will not oppose the movement resulting from your stick inputs. Whether this is noticeable is unclear, but theoretically it should be a bit more responsive.

There is a possible downside, however, in that wobble after sudden stick inputs (e.g. at the end of a roll) may be less of a problem when Dterm is not error seeking. Additionally if the 'kicks' aren't smoothed they can cause noise in motors during stick inputs, reducing efficiency, and since RC smoothing to smooth them out may cause delays or otherwise be problematic, I think Boris is keen to revisit which approach is best.

It might be useful, purely for testing purposes, if we could select the Dterm calculation mode via a switch, so we could change the functionality in flight to really see if it makes much of a difference. Or change it via a CLI parameter. Otherwise its a bit difficult to really know which approach is best, you have to reflash with a custom version and then it's not so easy to compare one to the other.

But yeah, with big stick inputs, the new Dterm approach will be smoother on BlackBox during the input, and will always be out of phase with gyros so may delay responsiveness a little to Pterm during stick inputs. It could be that a small increase in P will override the loss of Dterm kick. If so it is tempting to no longer use the error seeking Dterm calculation.

Additional information

Effect of D term on P/D controller: http://www.youtube.com/watch?v=xMygUvegC80

General explanation of D term: http://en.wikipedia.org/wiki/Derivative

Filtering, Aliasing and Gyro Sync explained

The following videos have been produced with BetaFlight in mind, and provide a great resource for in-depth learning of these complex subjects.

Filtering Basics: http://www.youtube.com/watch?v=CpW8_fOJ7_M

Nuhertz Spectra for analyzing copter gyro data: http://www.youtube.com/watch?v=fZm9N-WFkQk

Analyzing FFT graphs for multirotor tuning: http://www.youtube.com/watch?v=nxHK-V7GCYY

Aliasing and Gyro Sync Explained: http://www.youtube.com/watch?v=-lmoKal_e4s

New Biquad filter in BetaFlight V2.3.x: http://www.youtube.com/watch?v=Q2tSWU1MsVk

Rates rc rate translations into deg sec Tables

Boris posted Tables in:

Effect of rate combined with static rc rate of 100. (rc rate doesn't affect yaw). Full Stick input http://www.rcgroups.com/forums/showpost.php?p=34028934&postcount=18514

Effect of rc rate with static rate of 0. Full stick input http://www.rcgroups.com/forums/showpost.php?p=34028937&postcount=18515

and to make it complete. Only effect of yaw rate http://www.rcgroups.com/forums/showpost.php?p=34028986&postcount=18516

Motor update

I am also coming from the world of VOIP what is completely different than control laws, but there jitter is even much more of an issue in realtime audio/video processing. There are several ways to deal with it. Another option would be dropping motor output signals when they come to soon....before the old signal finished. That would be the only other option.

Another example: The new way runs cycletime of 125us and PID loop of 375us The motor update happens at begin of the new cycletime and not the begin of new PID looptime!

Start GYRO/PID loop------>+125us GYRO/MOTOR----->+125us GYRO----> +125 PID/GYRO------->+125 GYRO/MOTOR (375us)

Having faster motor rate than loop rate is complete nonsense as the value will still be overwritten and cause jittering. That's why i am reworking the tasking at the moment. The ideal motor handling would be to only write motors when there are new mixer calculations available and with the time interval between the motor updates never lower than the desired period. Longer motor updates are even not bad just as long as next motor update doesn't fall into previous one.

Lets say the pid controller / mixer requests full motor power on oneshot125 which is about 250us PWM interval.....if we were updating motors at 2k interval than there is in total 250us free interval where it doesn't matter when you update it. so allowed jitter period is a bit less than 250us.

On 4k speed the motor update period is totally not allowed to jitter to make full throttle possible.

It doesn't matter if the motor gets updated too late but should never be updated too soon.

With low looptimes like now we are reaching the point where motor updates happen near the end of cycletime. So there is no such thing as delay there. It doesn't matter if you do something at the end or beginning. Imagine like running circles you can't tell what is begin or end. The order doesn't matter anymore.

It's near the end of the cycletime where there is the most jitter. The time variations there are between 0 and 100us. Especially on low looptimes with scheduled tasking mechanism to get the maximum efficiency the end of loop cycle is really jittery. So when moving the motor update to the beginning of the loop you always have the constant timing. The variable delay you would have when writing the motors at the end of the loop now becomes a constant delay.

Jitter is quite crucial on oneshot signals as period of 60us is almost a half oneshot signal.