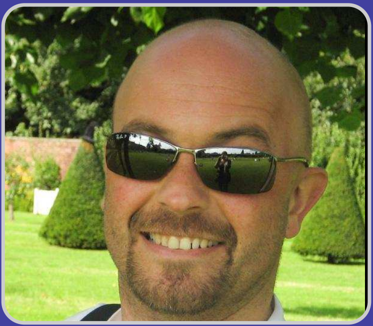
Quadcopter pre-flight checks



Andy Baker

Guest Writer

SKILL LEVEL : ADVANCED

[Ed: This is an advanced skill level article. It is not a step-by-step guide for beginners. Quadcopters are dangerous. They can cause damage and serious injury. Be responsible, educate yourself and take precautions to protect everything and everyone in the vicinity.]

In Issue 19 I presented a high-level overview of how to build your own quadcopter using a Raspberry Pi as the flight-controller. However, there were lots of details missing in that article about tuning and testing the settings for your quadcopter. This article fills in those blanks.

Some prerequisites...

These tests cannot be powered from a wire - you need a fully charged battery attached to your quadcopter. Unlike the rest of the world where duct tape is "the force", with quadcopters it seems to be velcro! Use it generously to make sure everything is securely attached.

Connect to your quadcopter

Your quadcopter needs Wi-Fi. You need another machine (Raspberry Pi, Windows or Linux) that can open a secure shell and log into your quadcopter Raspberry Pi. I use an iPad with WebSSH or rlogin from one of my other Raspberry Pi's depending on where the testing is happening.

Get the latest code

This project is still live and the code changes regularly. Make sure you have the latest software, which is available at https://github.com/PiStuffing/Quadcopter. Copy the software into your home directory, e.g. `/home/pi`.

The command-line

The quadcopter Python code `qc.py` has various command-line parameters which control its testing and flights. Here's the complete list. We'll go through these in detail throughout the article. To run your quadcopter, at the command prompt where the code lives, type:

```
sudo python ./qc.py
```

The following is a list of the available parameters:

`-f` – fly the quad
`-v` – run the video
`-g` – calibrate gravity
`-h XXXX` – set the PWM pulse width for hover
`--tc #` – run test case number #
`--hvp XXXX` – horizontal velocity PID P gain
`--hvi XXXX` – horizontal velocity PID I gain
`--hvd XXXX` – horizontal velocity PID D gain
`--vvp XXXX` – vertical velocity PID P gain
`--vvi XXXX` – vertical velocity PID I gain
`--vvd XXXX` – vertical velocity PID D gain
`--aap XXXX` – absolute angle PID P gain
`--aai XXXX` – absolute angle PID I gain
`--aad XXXX` – absolute angle PID D gain
`--arp XXXX` – angular rate PID P gain
`--ari XXXX` – angular rate PID I gain
`--ard XXXX` – angular rate PID D gain

Countdown beeps

My quadcopter has a beeper – it counts down from 5 beeps to 1 beep before the blades spin up. I put it there to add time before take-off allowing me to abort the flight for any reason. Some of the tests we are going to do require you to start up your quadcopter and then hold it or move it in some way. You will do this during the countdown beep sequence. Therefore you MUST have a beeper, or some equivalent indicator, that take-off is imminent. The beeper can be replaced by an LED as long as you can see it clearly.

Binding the code to the hardware

I'm assuming here that you already have a fully built breadboard matching the circuit diagram / breadboard layouts provided in Issue 19.

The code refers to the propellers by their location (front-left, front-right, back-left and back-right – looking from the top) and the direction the propellers rotate (clockwise or anticlockwise). The first step of pre-flight setup is to ensure that the hardware agrees with the software.

With your battery disconnected, fit the blades to the motors to match the code: front-left and back-right blades spin anti-clockwise; the other pair clockwise. Look carefully at your set of four blades. It should be clearly visible which are designed to move clockwise and which anti-clockwise – fit them to match the code's expectations.

Plug the three motor wires into the ESC (Electronic Speed Controller) and the ESC PWM leads into the breadboard. The ESC cables are generally brown, red and orange (resistor colour coding for 1, 2, 3) or black, red and white / grey for ground, 5V and signal respectively. Connect the orange / 3 / white / grey signal wire to the following breadboard pins:

Back-left ESC plugs into BCM pin 22 / GPIO pin 15
Front-left ESC plugs into BCM pin 17 / GPIO pin 11
Front-right ESC plugs into BCM pin 18 / GPIO pin 12
Back-right ESC plugs into BCM pin 23 / GPIO pin 16

The red / 2 wire connects to 5V and the remaining brown / 1 / black wire connects to ground.

Now we need to make sure the connection to the ESCs are also in agreement with the direction each blade should rotate. For that we need to use some test code that starts up each of the motors in turn at a slow speed starting from front-left, then front-right, then back-left and ending with back-right.

Search the code for "TESTCASE 1:" for the code change used by this test.

Run the test from the command prompt:

```
sudo python ./qc.py --tc 1 -h 200
```

200 is an ESC PWM setting that should be high enough that all motors start to turn, but low enough that they spin slowly.

As each blade spins up, you should check that each spins in the correct direction. The easiest way is simply to feel the airflow from each blade as it spins – it should be blowing downward! If it feels wrong, swap any two of the three ESC wires to the motor and try again.

As mentioned, the code steps through the motors in order front-left, front-right, back-left and back right – if they don't run in that order for you, check that your ESCs are connected to the correct GPIO pins.

Hover speed

The next step is to determine an approximate hover speed. That means finding the PWM pulse width in microseconds that is required to sustain a hover. If you don't set it on the command line with the `-h` parameter then the code defaults the value to 590, which is the value that works for me. But how do you find this magic number for your quadcopter?

It involves another set of code changes – look for "TESTCASE 2:" in the code. This sets all the motors to spin for 10s, and then stop – how fast they spin is set by the command line parameter. Start up your quad with:

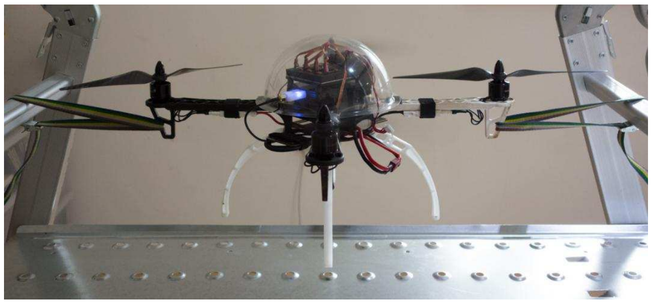
```
sudo python ./qc.py --tc 2 -h 100
```

While it's beeping pick it up and and hold it above your head, making sure the propeller blades can't hit anything (especially you!) when they start spinning. DO NOT LET GO, there's no control software running and the quad will fly around like a screaming banshee trying to cut everything in sight with its blades, especially you!

As the blades spin, focus on how the quadcopter feels. If the blades made no difference (or don't spin at all) increment the -h parameter by 100 and run the test again. As you start to feel the blades having effect, and your quadcopter is feeling lighter, be a lot more cautious. Increment -h by 25 or less until you get to a stage where your quadcopter feels weightless. At this point note the `-h` number. Find the line of code saying `cli_hover_speed = 590` and replace 590 with your magic number.

Inner PID balancing

For this test, you will need two raised platforms to balance your quadcopter between – chairs, stools, sturdy cardboard boxes will all do nicely – they just need to be the same height. They only need to be tall enough to lift your quadcopter's legs off the floor. Your quadcopter centre of gravity needs to be just below the support point – if it's above, your quadcopter will be too unstable to do PID tuning. I had to hang my quadcopter between the rungs of a step-ladder in my latest tests to achieve this. Look at http://blog.pistuffing.co.uk/?p=1314 to see how I did this previously with stools.



The code changes in "TESTCASE 3:" disable the front-right and back-left motors. Place these arms on the two platforms such that your quadcopter dangles in free space between the platforms as shown in the photo. The other two motors will spin at your newly found `cli_hover_speed`. Your quad may not balance between the platforms unless gently supported by you. Enter:

```
sudo python ./qc.py --tc 3 --arp 50
--ari 0 --ard 0
```

Between the 5 beeps and 4 beeps, your quad needs to sit on the floor as gyro calibration is underway. At 4 beeps, get your quad onto the platforms with the front-right and back-left legs balancing in between. Gently hold the nearest leg. When the blades spin up, carefully let go – what happens?

• If the arm you were supporting drops with your hand as you move it down, your angular rotation rate PID P gain (`--arp`) is not doing its job and needs to be increased significantly – try doubling it.

• If the arm seesaws increasingly meaning you need to hold the leg firmly, your PID is way too high and needs significantly lowering – try reducing by 50%

• If the arm stays where it was, wobbling slightly, but the wobble neither shrinks nor grows, you're done – lets call this K_u – the ultimate gain – in my recent retesting it came out to be `--arp 150`.

You are aiming to reach the point where your quadcopter starts seesawing – this is much easier to approach from below rather than trying to reduce the frantic wobble if you start with the P gain too high!

The next stage is to start playing with the I and D gain values. I did this months ago with days of testing, first adding I gain and reducing P gain as a result, and finally adding D gain. However, I've learnt recently from a comment on my blog that the P, I and D gains are related. Check out http://en.wikipedia.org/wiki/Ziegler-Nichols_method.

Using this method you can find K_u then K_p. The values for K_i and K_d can be estimated mathematically. To do so, you need to count how many wobbles happened during the test. For me that was 54 in a 15s test flight, so the oscillation period $T_u = 15 / 54 = 0.278$s. With these values, I tried Z-N PIDs – each worked well, but each with a minor flaw. In the end, I picked the best from each resulting in $K_p = 65$, $K_i = 360$ and $K_d = 5.2$.

I tried these values and they worked perfectly as the picture on the left shows. The left and right props are stationary, the front and back props are spinning. No blur in the photograph equals no wobbles in the hover.

With your equivalent values in place, rerun TESTCASE 3. When the test is running you should be able to let go and tap the quadcopter and both feel it resisting due to the differential gain, and returning to where it started due to its integral gain.

If you're in that position, replace the default values for `cli_arp_gain`, `cli_ari_gain` and `cli_ard_gain` in `CheckCLI()` with your own values. If the PID values don't work it is time for free-form testing:

• Oscillation is due to a too high P gain
• Lack of return to horizontal is due to a too low I gain
• The jitters are due to a too high D gain.

Enjoy the tinkering until you find your perfect tuning.

Now it's time to tune the other PIDs outdoors, but before that happens, sensor calibration is needed.

Gravity Calibration

Taking off from a sloping surface, even if it's very subtle will ultimately lead to sideways drift. That is because if the quadcopter takes off from a slope, it stays leaning throughout the flight. Leaning means some of the power that would be applied as lift is now redirected as horizontal acceleration... or that would be the case if only the gyroscope outputs were used to calculate the tilt angles. Luckily the accelerometer is also used and the result combined produces accurate values over time.

The accelerometer reads the force of gravity distribution across its X, Y and Z axes. When the quad is horizontal, its output in the X, Y and Z axes should be 0, 0, 1... or it will be after you calibrate it.

Find yourself a flat strong platform on which your quadcopter can stand. I made a custom one from some perspex with bolts at the corners for easy fine tuning but any wood / metal / glass platform propped up with cardboard / paper / anything thin will do just fine.

Place your platform on the floor. With a spirit level in both X and Y axes, get the platform as horizontal as you can by propping up the corners. Now sit the quadcopter on the platform. Leave the battery disconnected and instead boot it up from the the micro USB plug / wall wart. Then enter:

```
sudo python ./qc.py -g
```

Wait a few seconds – job done. Check the output by typing:

```
cat qcgravity.cfg
```

You should see the corrective values required by the accelerometer so it reads 0, 0, 1, when it is horizontal.

Gyro Calibration

This happens automatically for every flight and you don't need to do anything. However it is worth mentioning it in passing for completeness.

In the same way the accelerometer gives innaccurate readings without tuning, so too does the gyro. The net result of this is the quadcopter believes it is rotating on its axes even though it is not. Gyro calibration happens as the first thing when the quadcopter code starts up while it is sitting still on the ground.

The remaining PIDs

Your quadcopter is now safe for outdoor test flights. The move outdoors reduces the number of valuable things that the quadcopter can hit during these tests!

Luckily the remaining PIDs are based upon mathematical guesstimation – and if you get it wrong nothing really bad happens. It does depend somewhat on the 'gut-feel' for propeller speeds you have acquired while doing the earlier testing.

For outside testing the `--tc` parameter is not used. Instead `-f` is used to signify a preprogrammed flight pattern: a 5 second takeoff at 0.35 m/s, a 5 second hover at the resultant height of 1.75m, followed by a 5 second descent back to the ground.

Vertical velocity PID

First tuning is for the vertical velocity PID; for take-off, hover and landing. Imagine a speed of ascent of 1 m/s – that is very fast and going to need a lot of power. I guestimated (based on the experience from TESTCASE 2) that an input of 1 m/s puts out a PWM increment of 150µs to the vertical velocity PID P gain. I added another 50 as the vertical velocity PID I gain. These values just worked, so I have left them as the default values in `CheckCLI()`.

Place your quadcopter on the horizontal platform and type:

```
sudo python ./qc.py -f --vvp 150 --vvi 50
--hvp 0.0
```

It should follow the predefined flight plan as described. Do not be tempted to run this test indoors – the expected results I describe may not be what you see, and you don't want your quadcopter hitting the ceiling!

If it all starts going horribly wrong, pressing <CTRL>+C will move on to the next step in the pre-programmed flight pattern. Hammering it hard in a blind panic will bring the quadcopter crashing to the ground – been there, done that, too many times.

If this testcase works for you then update `cli_vvp_gain`, `cli_vvi_gain` and `cli_vvd_gain` in `CheckCLI()` to use these values.

Absolute angle PID

Next the absolute angle PID – again I used the same gut feel plus guesswork algorithm. If we want to change the quadcopter from say hover to 2° and we want this to happen quickly, but not manicly, then the absolute angle PID needs a P gain of about 5 to convert from the 2° error to a 10°/s rotation speed. Again this value worked fine so I left it at that. Check yours by taking off from the ground (the horizontal take-off platform is no longer needed) by entering:

```
sudo python ./qc.py -f --aap 5 --aai 0
--aad 0 --hvp 0
```

You should see the same flight pattern, even if the ground is uneven or sloping. Again, if this works for you, update the default values for `cli_aap_gain`, `cli_aai_gain` and `cli_aad_gain` in `CheckCLI()`.

Horizontal velocity PID

Finally, the horizontal velocity. This PID takes a target of the desired speed, puts out a target acceleration which is then converted to a desired tilt angle by dividing the output by gravity g (9.81m/s^2) and taking the inverse tangent. So if we wanted to accelerate at 1g horizontally, then the quadcopter would tilt at 45°. So we start with a target to move at 1 m/s. Let's say we want this to produce 5 m/s^2 acceleration ($\approx 0.5g$) => tilt angle of $\arctan(0.5g / 1g) \approx 26°$ which feels about right. So the P gain here is 1 m/s to 0.1g = 0.1. Seems like a good gut-feel starting point.

You may notice this time I haven't said "and it worked fine for me" because I am still testing it. This PID is there to prevent drift in windy conditions – the quadcopter should hover above the take-off point regardless of wind. But wet winter weather, combined with crashes in testing, are preventing me from completing this test. Stick with ––hvp `0.0` unless you are feeling confident to take on these final testing tweaks yourself.

Flight Control

Currently the quadcopter is autonomous – its flights are hard-coded rather than under the control of the user via a radio control (that's another article, once I've built and tested it). That means if you want to change a flight pattern, you need to change the code. Search for the `# Interpreter:` tag.

Currently, this code is set up to climb for 5 seconds at 0.35 m/s, hover for 5 seconds and then descend for 5 seconds at 0.35 m/s. By all means tinker with the timings and vertical velocity targets and see what effect they have.

Flying in the middle of nowhere

Throughout this article, there's an assumption that the quadcopter and the SSH client have Wi-Fi connectivity. But if you're out flying in a field then that will probably not be the case. You need to set up your quadcopter as a wireless access point (WAP). I won't explain the details of this here but just make you aware of the requirement. Check out my blog entry at http://blog.pistuffing.co.uk/?p=594, plus the others linked from there and many more like it on the Raspberry Pi forums, for details on setting up a WAP.

Q&A

If you've deviated from my design you may have questions about why your quadcopter is not working for you. Here are a few I've second guessed to guide you.

Q: How do I use my UBEC to power my Raspberry Pi?

A: My ESCs don't have an UBEC built in, so I added a DC-DC converter (switching regulator with input / output isolation) to produce 5V from the LiPo battery. That's what your UBEC does too. I suggest looking in the Raspberry Pi Forum or joining http://diydrones.com and use their forum.

Q: How do I protect against the battery going flat?

A: A really good battery can still only power a flight for under 10 minutes. Start all flights and testing with a full battery and keep in mind how long your testing has been going on. I do need to add power management to my quadcopter, but that's not very interesting so very low priority for me. Do some research, prebuilt options exist.

Q: My quadcopter does XXXX during outdoor testing. Why?

A: There are so many reasons. Luckily with each flight comes a set of diagnostic data saved as a .csv file. This is a generic spreadsheet format which Microsoft Excel and LibreOffice can understand, allowing you to diagnose the problem. Have a look in my blog http://blog.pistuffing.co.uk/?tag=statistics for the various graphs that can be plotted from the raw data.