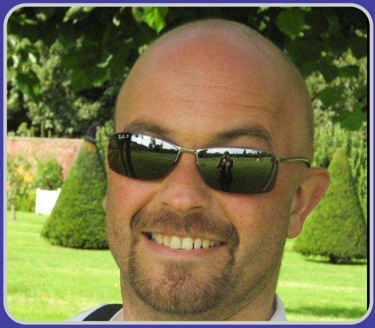


Andy Baker

Guest Writer

Building an airborne autobot

SKILL LEVEL : ADVANCED

A quadcopter is a flying machine with four propellers controlled either autonomously (programmed with a fixed flight routine) or via a remote control.

This first article (hopefully of a series) covers a brief overview of how they work, how to build one controlled by a Raspberry Pi, information about where to get all the bits you need and how to bolt them all together physically, electronically and in software. The result should be a quadcopter which can take-off, hover and land autonomously (and with care!)

Future articles will cover more details on testing and tuning this basic quad including code enhancements to allow lateral movement, a Raspberry Pi remote control, and perhaps future developments covering GPS tracking.

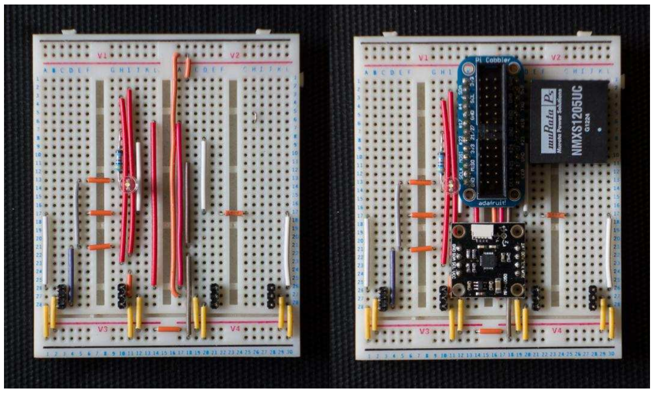
Parts of a quadcopter

First a quick breakdown of all the parts that make up a quadcopter.

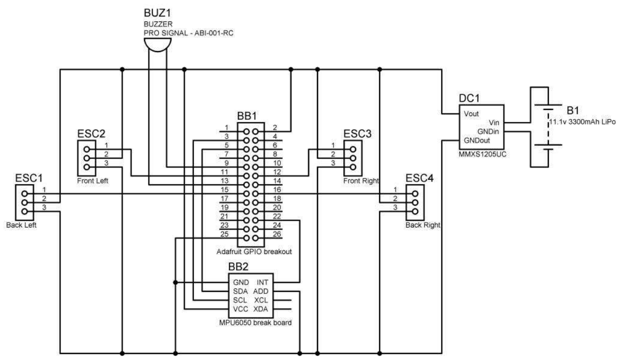
There are four propeller blades. Two of the four are designed to rotate clock-wise; the other two anti-clockwise. Blades which are designed to move the same way are placed diagonally opposite on the frame. Organising the blades like this helps stop the quadcopter spinning in the air. By applying different power to each propeller, and hence different amounts of lift to corners of the quadcopter, it is possible to not only get a quadcopter to take-off, hover and land but also by tilting it, move horizontally and turn corners.

Each propeller has its own DC brushless motor. These motors can be wired to rotate clockwise or anti-clockwise to match the propeller connected to them. The motor has coils in three groups around the body (called the stator) and groups of magnets attached to the propellor shaft (called the rotor). To move the blades, power is applied to one group of the coils and the rotor magnets are attracted to that coil, moving round. If that coil is then turned off and the next one powered up, the rotor moves around to the next coil. Repeating this around the three coils in sequence results in the motor rototating; the faster you swap between the three powered coils the faster the motor rotates. This makes the motor suitable for ‘digital’ control – the direction and speed of movement of the propeller blade exactly matches the sequence and rate power pulses are applied to the coils. These motors take a lot of power to spin the propeller blades fast enough to force enough air down to make the quadcopter take-off – far more power than a Raspberry Pi can provide – so an Electronic Speed Controller (ESC) bridges that gap. It translates between a Pulse Width Modulation (PWM) control signal from the Raspberry Pi and converts it to three high-current signals, one for each coil of the motors. They are the small white objects velcro'd under the arms of the quadcopter.

Next there are sensors attached to the breadboard on the shelf below the Raspberry Pi; these provide information to the Raspberry Pi about rocking and rolling in three dimensions from a gyroscope, plus information about acceleration forward, backwards, left, right, up and down. The sensors connect to the Raspberry Pi GPIO I^{2}C pins.



In the circuit diagram you can see I am considering adding a beeper, so I can hear what the quadcopter thinks it's doing.



The power for everything comes from a single lithium polymer (LiPo) battery which provides 11.1V up to a peak current of 100A, with the full-charge of 3300 mAh thus supplying 3.3A for an hour or 100A for two minutes or anywhere in between. This is a powerful and dangerous beast, yet it only weighs 250 grams. It requires a special charger – if not used, a LiPo battery can easily become a LiPo bomb – beware. There is a regulator on the breadboard to take power from the battery and produce the 5V for the Raspberry Pi and also provide a degree of protection from the vast power surges the motors draw from the battery.

That just leaves the beating heart of the quadcopter itself; the Raspberry Pi. Using Python code it reads the sensors, compares them to a desired action (for example take-off, hover, land) set either in code or from a remote control, converts the difference between what the quad is doing (from the sensors) and what it should be doing (from the code or remote control) and changes the power to each of the motors individually so that the desired action and sensor outputs match.

Creating your quadcopter

First and foremost, flying machines and boats are female and they have names; mine is called Phoebe ("Fee-Bee"). Choose a name for yours and treat her well, and the chances are she'll reciprocate!
Phoebe's body, arms, legs, blades, motors, ESCs and batteries are from kits. Total cost is about £250 – together with a Raspberry Pi, and other accoutrements, the total cost is perhaps £300 – £350. Not cheap for something which certainly at the start has a preference to crash rather than fly!

A complete bill of materials (BOM) is available at http://blog.pistuffing.co.uk/?p=1143

I've actually upgraded my motors to higher power, lighter weight varieties but this is absolutely not necessary – the equipment provided by the kits is excellent. Upgrading components for weight reduction / power efficiency and strength is definitely an afterthought once the basics are working.

The Raspberry Pi is a model A, chosen for lower weight and lower power consumption; these are factors reflected through other pieces of the design. I have removed the audio and video outputs and use a micro-SD card adapter from the guys at Pimoroni – all in the name of weight saving.

The Raspberry Pi case is a variant of the Pimoroni PiBow model B case with a couple of levels removed and some holes sealed for reduced weight and increased strength (protection from crashes!). I've posted the design for these at http://blog.pistuffing.co.uk/wp-content/uploads/2013/10/Pibow002-AB5.pdf. Phenoptix do a great job of laser cutting 3mm acrylic at a very reasonable price.

Talking to Phoebe

Whether Phoebe is autonomous or remote controlled someone needs to talk to her to tell her what to do. To that end, Phoebe runs a wireless access point (WAP) so another computer can join her private network and either SSH in or provide remote control commands. You can see how I did this at http://blog.pistuffing.co.uk/?p=594.

For initial testing the WAP function isn't necessary, any household wireless network will do, but as your quadcopter comes to life you will want to be doing your testing away from animals, children and other valuables you don't want damaged (like yourself). Having a WAP means you can take the testing out into the garden or local park or field.

Presenting Phoebe's Python code

The final step is obviously the most important; once you have a physical quadcopter with associated blades, motors, ESCs, power and circuitry, we use Python code to glue all the pieces together. I'm not going to go into this blow by blow here as the code is available at https://github.com/PiStuffing/Quadcopter and it should be self-documenting. There are more lines of explanatory comments than there are lines of code actually doing something constructive!

The I^{2}C class provides a programming interface to read and write data from the sensors. Built on that, the MPU6050 class configures the sensors and then provides API access to reading the data and converting the values from the sensor into meaningful values humans would understand (like degrees/sec for rotation or metres/sec^2 for acceleration).

The QUADBLADE class handles the PWM for each blade handling initialization and setting the PWM data to control the propeller blade spin speeds. The PID class is the jigsaw glue and the core of the development and testing. It is the understanding of this which makes configuring a quadcopter both exciting and scary! It is worth an article in its own right – for now there is a brief overview of what they do and how at the end.

There are utility functions for processing the startup command line parameters, signal handling (the panic button Ctrl-C) and some shutdown code.

Last, but not least, there is the big "while keep_looping:" loop which checks on what it should be doing (take-off, hover, land, etc), reads the sensors, runs the PIDs, updates the PWMs and returns to the start one hundred times a second!

PID

The PID (Proportional, Integral, Differential) class is a relatively small, simple piece of code used to achieve quite a complex task. It is fed a "target" value and an "input" value. The difference between these is the "error". The PID processes this "error" and produces an "output" which aims to shrink the difference between the "target" and "input" to zero. It does this repeatedly, constantly updating the "output", yet without any idea of what "input", "output" or "target" actually mean in its real world context as the core of a quadcopter: weight, gravity, wind strength, RC commands, location, momentum, speed and all the other factors which are critical to quadcopters.

In the context of a quadcopter, "target" is a flight command (hover, take-off, land, move forwards), "input" is sensor data and "output" is the PWM pulse size for the motors.

Phoebe has 4 PIDs running currently – pitch, roll, yaw and vertical speed – these are the bare minimum needed for an orderly takeoff, hover and landing.

The PID's magic is that it does not contain any complex calculations connecting power, weight, blade spin rates, gravity, wind-speed, imbalanced frame, poor center of gravity or the many other factors that perturb the perfect flight modelled by a pure mathematical equation. Instead it does this by repeated, rapid re-estimation of what the current best guess “output” must be based only on the “target” and the “input”.

The “P” of PID stands for proportional – each time the PID is called its “output” is just some factor times the “error” – in a quadcopter context, this corrects immediate problems and is the direct approach to keeping the absolute “error” to zero.

The “I” of PID stands for integral – each time the PID is called the “error” is added to a grand total of errors to produce an output with the intent that over time, the total “error” remains at zero – in a quadcopter context, this aims to produce long term stability by dealing with problems like imbalance in the physical frame, motor and blade power plus wind.

The “D” of PID stands for differential – each time the PID is called the difference in error since last time is used to generate the output – if the “error” is worse than last time, the PID “D” output is higher. This aims to produce a predictive approach to error correction.

The results of all three are added together to give an overall output and then, depending on the purpose of the PID, applied to each of the blades appropriately.

It sounds like magic... and to some extent it is! Every PID has three configurable gain factors configured for it, one each for “P”, “I” and “D”. So in my case I have twelve different gain factors. These are magic numbers, which if too small do nothing, if too large cause chaos and if applied wrongly cause catastrophe. My next article will cover this in much more detail, both how they work and how to tune the gains. In the meantime, use the bill of materials on page 5 and get on with building your own quadcopter. The PID gains in the code I’ve supplied should be a reasonable starting point for yours.

Flying Phoebe

At the moment it is simple but dangerous! Put Phoebe on the ground, place a flat surface across her propeller tips, put a spirit level on that surface and make sure she’s on absolute horizontal by putting padding under her feet – this is absolutely critical if you don’t want her to drift in flight – we’ll fix this in another article with some more PIDs.

Connect the LiPo battery. The ESCs will start beeping loudly – ignore them.

Wait until the Wifi dongle starts to flash – that means Phoebe’s WAP is working.

Connect via SSH / rlogin from a client such as another Raspberry Pi, iPad etc. which you have joined to Phoebe’s network.

Use the `cd` command to change to the directory where you placed Phoebe’s code. Then enter:

```
sudo python ./phoebe.py -c
sudo python ./phoebe.py -c -t 550 -v
```

`-c` calibrates the sensors to the flat surface she’s on
`-t 550` sets up the blades to just under take-off speed
`-v` runs the video camera while she’s in flight.

There are other options. You can find them by entering:

```
sudo python ./phoebe.py
```

Enjoy, but be careful.