A Practical Guide to Building a Pentacopter

Tao Du
taodu@csail.mit.edu

July 17, 2018

Contents

1 Introduction

This document tells you (almost) everything you need to know to build a multicopter from scratch. Specifically, this tutorial will go over all the steps in the process of building and flying a pentacopter (Figure 1). The final flight test video can be found here on YouTube. The idea of pentacopter was originated from this paper but the old pentacopter was not ideal for demonstration due to its bulky size and heavy weight. So I decided to remake this pentacopter and it now serves as a long-term demo in our lab.

You don't need to know anything about copters ahead, but I assume you are comfortable with learning some simple physics (e.g., Newton's law) and writing a few lines of C++ code. You will also need to have access to a machine shop and be happy with soldering, laser cutting, or 3D printing. The

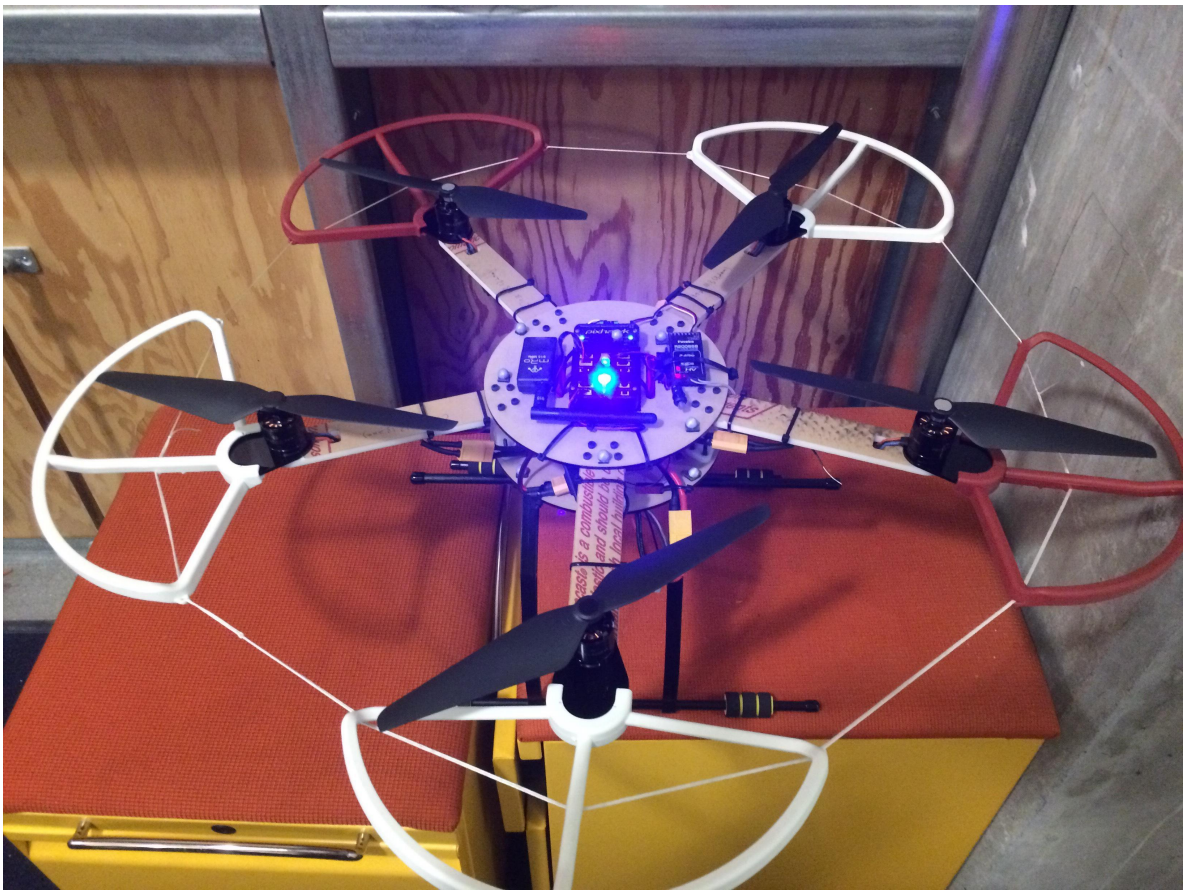

Figure 1: The final pentacopter demo.

last two are necessary only if you want to replicate my pentacopter exactly, but hey, they are very useful skills, and you are very likely to need them in your future customized build.

Another thing I should mention ahead is that building your own copter is not a cheap hobby. Building a single pentacopter cost our lab about $600, and we did not even include the cost of 3D printing and laser cutting. If your budget is tight, this tutorial is less useful for you, and you should probably start with a micro quadcopter (there are plenty of build logs online and they generally cost less than $100).

2 Design

In this project, we plan to build a pentacopter that weighs 1.5kg to 2kg. The size of the pentacopter we have in mind is roughly 55cm by 55cm. I chose these values based on my prior experiences and some successful commercial designs (e.g., DJI F450 and F550). Figure 2 shows all the necessary electronic devices and their connections. We will briefly explain each of them shortly.

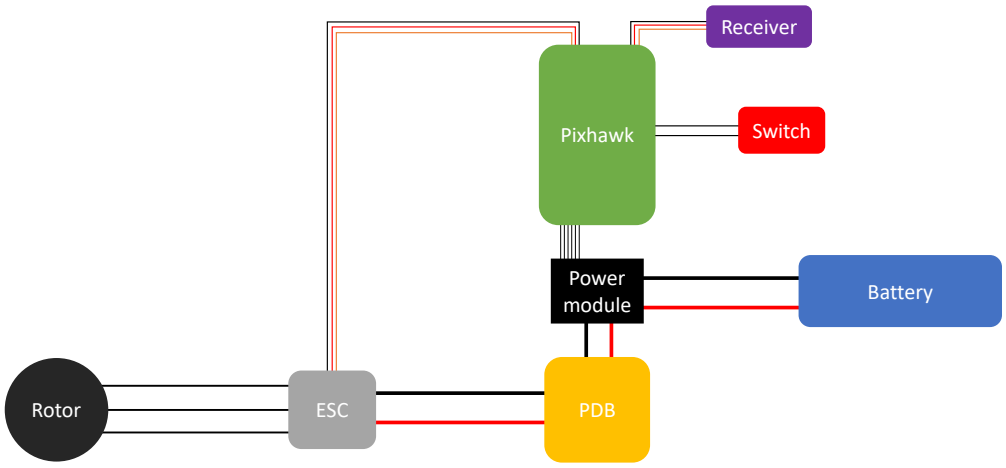

Figure 2: The diagram shows how each electronic component is connected to each other. For simplicity only one rotor is displayed.

Battery A battery is the only power source to the whole copter. For the size and weight of our copter, the most common choice is 3S to 4S LiPo, 2000mAh to 4000mAh LiPo batteries. They typically weigh 100g to 300g and support 5 to 15-minute flight.

Power module Your battery provides power to both the rotors and the flight controller, but they require different voltages. Typically, a rotor needs a voltage of 12V to 16V, and the flight controller needs only 5V. The power module is designed for this purpose. It provides power to the Pixhawk flight controller via a 6-pin cable and outputs 12V to 16V (depending on the number of cells in your LiPo battery) to the power distribution board (PDB) via a pair of red and black wires.

Power distribution board PDB is just a neat way to organize all of your ESC wires. It is nothing more than a simple board that connects all of your ground connectors together and connects all positive connectors to one another. A PDB usually has one pair of ground and positive connectors for input voltage, and 4 to 6 pairs of output ground and positive connectors, one for each ESC that powers one rotor.

ESC An ESC, or electronic speed controller, is a device that controls the motion of your rotor (and the corresponding propeller, of course). It first takes as input a PWM signal from the Pixhawk via a 3-pin cable, then it determines how much current is needed to be drained from the power source via the red and black wires to the PDB. The final signals are sent to the rotor via 3 wires, whose order controls the spinning direction of your rotor. The rotor and its propeller then spin, and therefore

generate thrust to lift your copter.

Receiver This device talks to your RC transmitter. During the flight, you use your transmitter to send control signals, typically in 4 to 8 channels. The receiver receives these commands and forwards them to Pixhawk so that it can make the correct decisions on the next move.

Switch This is just a safety switch for Pixhawk. If you do not press it before your flight it won't allow you to arm your rotors.

Pixhawk This is the brain of your copter. During the flight, Pixhawk executes a fast loop (400Hz in ArduPilot) to control the position and attitude of your copter. Roughly speaking, at each iteration, it does the following jobs in order:

- Read raw sensor data (gyroscope, accelerometer, barometer, etc) to sense the environment;
- Fuse all the sensor data and determine the copter status (e.g., location, attitude, linear velocity, angular velocity);
- Parse the input control signals from your receiver;
- Run the control algorithm (e.g., multi-layered PID) and determine how fast each rotor should run to achieve the desired goal;
- Send signals to ESCs and spin the rotor to generate desired thrust.

In our project, we use Pixhawk 1 (hardware) plus ArduCopter 3.5.5 (software) to control the pentacopter. Pixhawk 1, which used to cost us $200, is now officially discontinued but there are hardware and software compatible replacements which cost roughly $100 at this point. Of course, you can choose to buy the latest release (Pixhawk 4 at the time of writing, which costs aroud $200) and most of the content in this document is still valid. Alternatively, if you are concerned with the price, there are other cheaper flight controllers online. They will make your copter fly eventually but their quality may not be as good as Pixhawk.

2.1 Choose each component

Now you have a rough idea what are needed to build your own copter. The next step is to compile a shopping list and place some orders. Below I will briefly explain my choices for each item and how I make the decision.

Rotors, ESCs, and propellers These are called the propulsion system of your copter. A rule of thumb in choosing the propulsion system is that you want the hovering thrust is 40% to 50% of the maximum thrust. Recall that our copter weighs no more than 2kg and we have 5 rotors, so we should be looking for something whose maximum thrust is roughly 800g to 1000g. A few good options I found online are T-MOTOR Air Gear 450 and DJI E310, and I chose the former because E310 seems discontinued on DJI's official website.

Let's take a closer look at the Air Gear 450 specifications:

- Weight (65g rotor, 21g ESC, and 17g propeller): the total weight of five of them is $(65 + 21 + 17) \times 5 = 515$g. This is around a third of our total weight budget.

- Maximum thrust: 1293g according to their chart. However, notice that their measurement is taken under 16V, which is likely to be a bit higher than the battery voltage in your real flight test. This is because a) your battery voltage will gradually decrease as you drain power from it, and b) part of its voltage is wasted on the internal resistance, this is especially obvious when the current is large. Moreover, as the air density decreases when the altitude increases, the maximum thrust in the air won't be as large as you measure on the ground. As a result, although 1293g may sound like an overkill for our application, it really isn't.
- Maximum current: 16A according to the chart. This means the peak current through your PDB is $16 \times 5 = 80$A. You should keep this in mind when choosing the AWG value of your wires to solder.
- Battery: A 3S-4S LiPo battery is recommended. Recall that the nominal voltage is 11.1V for a 3S battery and 14.8V for a 4S one. Since their reported maximum thrust is measured under 16V, a 4S battery provides a closer voltage and therefore is a better option. In fact, I measured the performance of Air Gear 450 once I got them and the maximum thrust I got is 1256g at 90% throttle (PWM 1900) under 15.73V provided by a 4S battery.

Battery As I described earlier, I chose a 4S LiPo battery. To determine the capacity, we need to do some math: when the copter is hovering (let's say 50% throttle), each rotor drains 3.5A from the battery so 17.5A in total. If we want to fly it for 10 minutes, that requires $17.5/6$Ah $= 2916$mAh. You don't want to completely drain the battery so let's divide it by 80%: $2916/0.8 = 3645$mAh. I ended up using a Turnigy 4S 3000mAh battery because that provides a good trade-off between the flight time and the battery weight.

PDB Power distribution boards are usually very cheap and light. I only checked two things when I chose the PDB: a) how many ESCs it can support. Most PDBs support only 4 ESCs because they are designed for quadcopters, but I need at least 5; b) what the maximum peak current it can bear. In our case, I need at least 80A, as described earlier. I ended up choosing HobbyKing Matek PDB-XT60.

Power module This is another cheap and light component. Anything you found on Amazon that is compatible with Pixhawk should work. The one I chose is FPVDrone power module v1.0.

Switch You should get this when you buy your Pixhawk.

Receiver Be prepared to see another hefty price tag here. The receiver itself is not very expensive but you need to buy a RC transmitter, which could cost you as much as $200. When you choose the transmitter, keep in mind that you need at least 5 channels (4 channels for continuous control signals like roll, pitch, yaw, and throttle, and at least one discrete channel for switching flight modes). An 8-channel transmitter would be ideal. In our lab we use Futaba T8J 8-channel 2.4GHz transmitter and R2008SB 2.4GHz receiver, which cost us $250. If you don't want to invest that much, OscarLiang has compiled a list of more affordable options for you.

Pixhawk As I explained earlier, I used Pixhawk v1 because it was the latest release when our lab bought them a few years ago. At the time of writing, it probably makes more sense to buy the latest Pixhawk v4 if you are starting a project from scratch. Both cost you roughly $200.

Landing gear This is optional but I highly recommend it. If it is your first time to build your own copter, then I assure you your first flight will end up with a crash. It is definitely worth investing a high quality landing gear to minimize your loss. I looked into three options on Amazon and I will list all of them for you to choose from:

- Option 1: We had some in our lab for our DJI F450. They are cheap, light-weighted, and provide minimum protection to your copter. However, they are not very durable according to my tests. If you are a new pilot, you will probably break a few of them in your first flight.
- Option 2: I used this for my pentacopter demo. It is much more resilient to collisions but is also significantly heavier (but worth it!).
- Option 3: I was planning to buy this one before because it looks extremely durable. However, its over 300-gram weight may be a bit too much for my pentacopter. I would use it if I were to build a much heavier copter (say 4kg to 5kg) in the future.

Propeller guard This is also optional but highly recommended. The propellers are the most fragile part and are always easy to break whenever collisions happen. I used this one from Amazon but there are tons of options there. Do remember to check it is compatible with the size of your propellers.

Optional peripherals If you want, you can add a buzzer, a telemetry, a GPS, an airspeed sensor, or even a sonar to your Pixhawk. You copter can still fly without any of them though.

Name	Quantity	Total price ($)	Total weight (g)
T-Motor Air Gear 450	2	260	465
Turnigy 4S 3000mAh Battery	1	30	288
Matek PDB	1	8	10
Power module	1	11	24
Pixhawk v1	1	200	41
Futaba transmitter and receiver	1	280	55
Landing gear	1	21	212
Propeller guard	1	25	150
Total	-	835	1245

Table 1: List of items

Table 1 is a shopping list for your reference if you want to replicate my pentacopter demo described in this tutorial. Each Air Gear 450 combo consists of 4 rotors + ESCs + propellers, so I bought 2 and got 3 spare ones for replacement, and the total weight (465g) was for 5 rotors + ESCs + propellers. This also means your cost could be a little lower if you can find a retailer that sells them separately. I actually bought 2 batteries, 2 PDBs, and 3 power modules just in case I need to replace any of them. The transmitter was shared in our lab so what I actually bought was just an extra receiver, which cost me only $50.

2.2 Design the frame

Now you have placed the order and all your items will arrive in one week. It is time for you to learn how to use CAD software to design your own frame. You have options like SolidWorks, AutoCAD,

Onshape, Blender, etc. It usually does not matter which CAD software you use, just pick one and spend a few days getting yourself familiar with its basic features. For this project, I used Onshape because it has a friendly learning curve and the fact that it is a web-based application does not require you to download and install a 10GB program on your computer. I spent one night learning Onshape Fundamentals 1 to 4 online, and these already covered all I need to design a usable copter frame (Figure 3). The final design plan can be accessed here if you have an Onshape account.

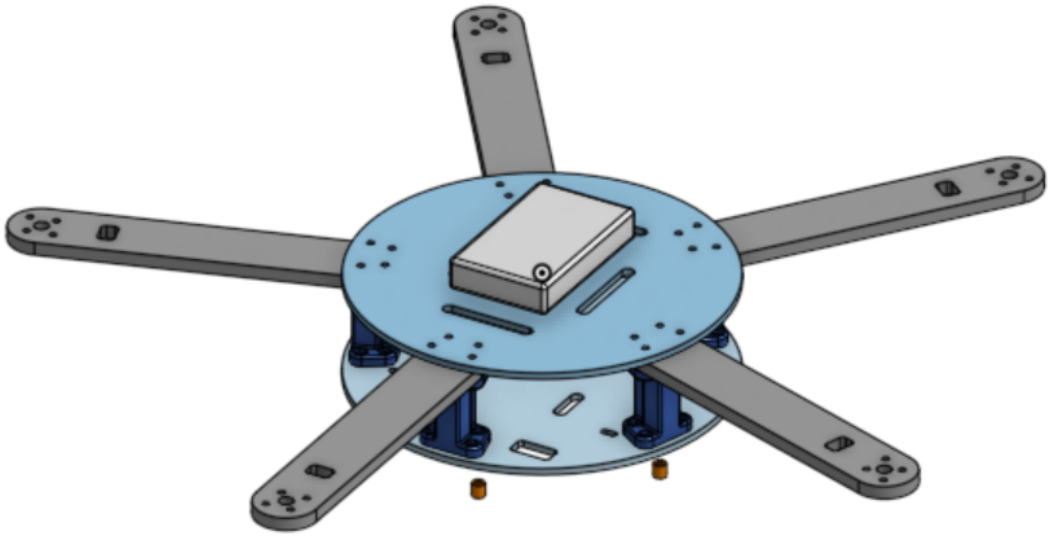

Figure 3: The screenshot of our pentacopter design in Onshape.

Once the design was finished, I used Onshape to estimate its weight: 80g for the top board, 170g for all the arms, 90g for the bottom board, and 80g for all the five supports between the top and bottom boards. Putting them together, the weight of our pentacopter becomes 1665g. We also need to add another 100g to 150g for the additional connectors, cables, screws, strip ties, VELCRO tapes, etc. Based on all the data so far, we estimate the final weight should be between 1750g to 1850g, which is well within the 1.5kg to 2kg range we set earlier.

3 Simulation

It is always a good idea to verify our design in simulation before you do a real flight test. Our lab has developed our own copter simulator for research purposes (Figure 4). One of the benefits of our software is that we support customized copter frames that are rarely seen in standard copter simulators. All you need to do is express your customized copter design in an XML file and our simulator will present an interactive interface for you to do a virtual flight test. You can read our user manual if you are interested in knowing more.

Of course, there are a lot more simulators for you to choose from if you build a standard quadcopter

or hexacopter. The SITL simulator in ArduCopter seems to be a very good one, although I haven't tried it personally.

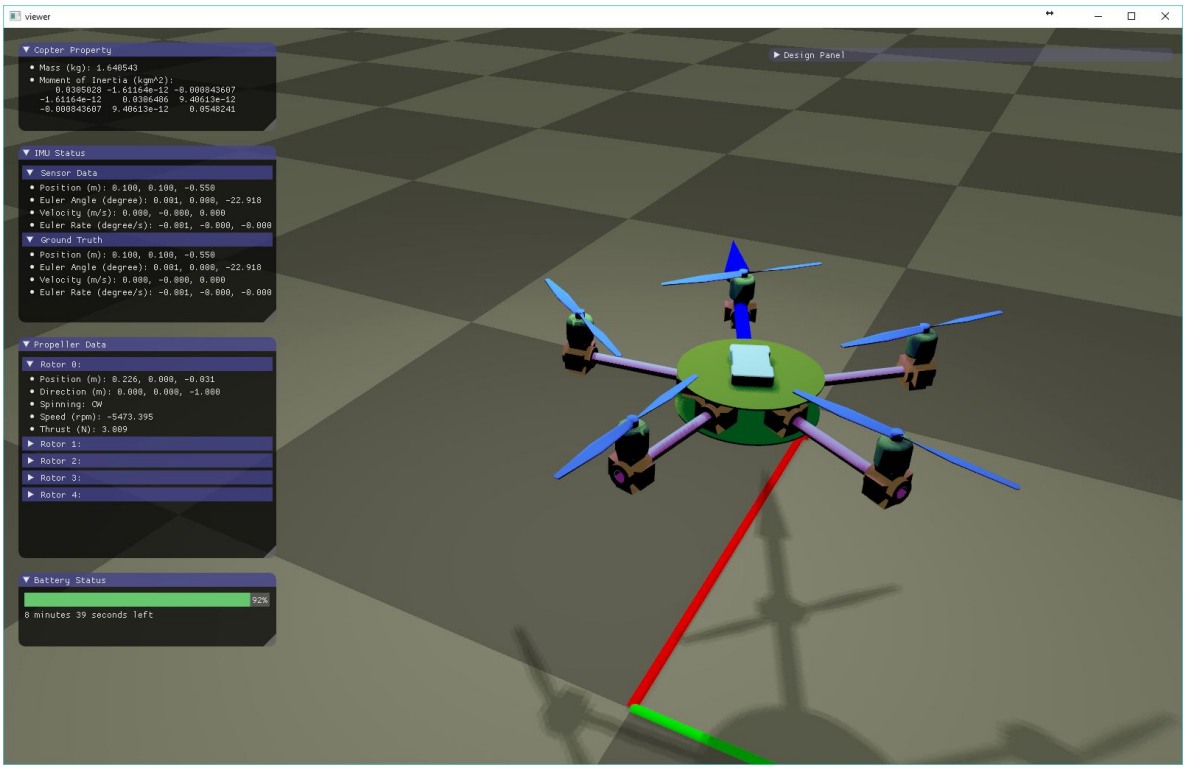

Figure 4: Simulating the flight in software.

4 Control

We have talked about the controller briefly at the beginning. So far a controller looks like a black box that magically computes the correct output thrust signals. It's time to gain a better understanding of this black box now.

4.1 The physics behind a copter

The first thing to understand is the mechanics behind a copter. To simplify it I will first explain how a quadcopter works, and then extend it to our pentacopter.

How to fly a quadcopter A quadcopter has two clockwise and two counterclockwise rotors (Figure 5). Let T_1, T_2, T_3, T_4 be the thrusts generated by each rotor. Apart from the thrust itself, each T_i generates a torque that attempts to roll the copter over along certain axis. When hovering, the quadcopter requires each T_i be a quarter of its gravity so that both the net force and the net torque

are completely balanced. If you want to lift it up or down, you just increase or decrease all T_i by the same amount.

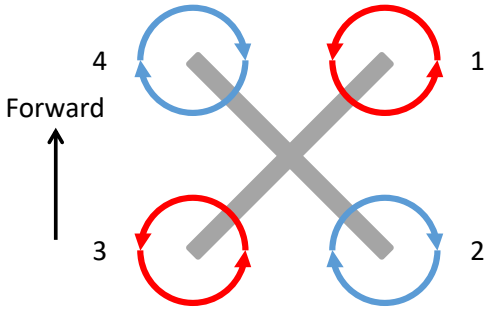

Figure 5: A top-down view of a quadcopter. The spinning direction of each rotor is indicated by color.

To fly the quadcopter forward, we need to increase T_2 and T_3 and decrease T_1 and T_4 by the same amount of force (Figure 6(a)). In this way, the copter leans itself forward but does not rolls over to its left or right. Similarly, to move the quadcopter laterally, we should increase T_3 and T_4 and decrease T_1 and T_2.

Finally, to change the heading of a quadcopter, you need to know that each T_i also generates a spinning torque Q_i that is proportional to T_i. Intuitively, this just describes the fact that if a clockwise rotor spins, the body of the copter will spin in the opposite direction. So how can we change the direction of a quadcopter without flipping it over? The answer is to increase thrusts from the clockwise rotors and decrease thrusts from the counterclockwise rotors by the same amount (Figure 6(b)). In this way, the net torque that attempts to flip the copter is zero and the quadcopter can spin stably.

If you still need more information, WIRED recently made an excellent video that explains how a quadcopter flies.

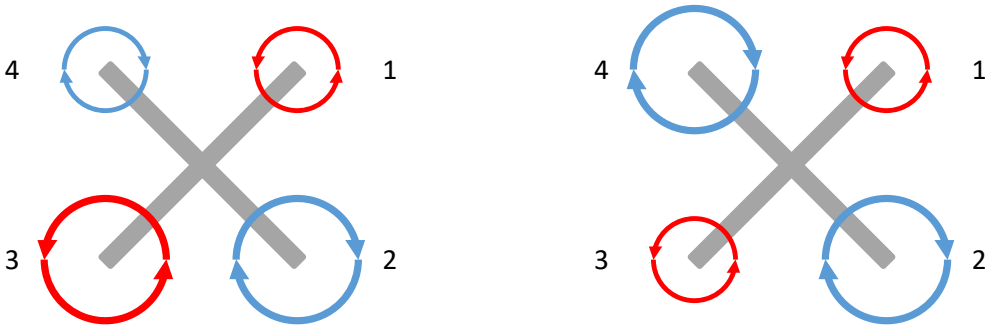

Figure 6: Change of thrusts when a quadcopter moves laterally or spins. Bigger circles represent larger thrusts.

Math behind a quadcopter Next, we will write down some math to explain the same thing again but in a more quantitative way. Imagine you are a pilot sitting in a quadcopter facing forward, let's establish a local frame rigidly attached to the body of your quadcopter in the following way: we will use the forward direction as the x axis, the side direction on your right as the y axis, and the z axis goes straight down. We further assume your local frame follows the right-handed rule and the origin is at the center of mass. Let's also use North-East-Down as our inertial world frame, i.e., a frame that never changes during the flight. At each time t, let $\mathbf{p}(t)$ be the location of the center of mass of your copter in the world frame, and let $\mathbf{R}(t)$ be the rotational matrix that represents the transformation from your body frame to the world frame. These two time variant functions fully determine the motion of your copter.

So how does $\mathbf{p}$ change with time? If we define $\mathbf{z} = (0, 0, 1)$, then $-T_1\mathbf{z}$ is the thrust induced by rotor 1 defined in the body frame. According to Newton's law:

$$m\ddot{\mathbf{p}} = m\mathbf{g} - \mathbf{R}\sum_{i=1}^{4} T_i\mathbf{z} \tag{1}$$

To understand the rotation, we need to define the torque induced by each rotor. Let L be the arm length, i.e., the distance from a rotor to the center of mass. We can now write down the location of each rotor in the local frame:

$$\begin{aligned} \mathbf{p}_1 = &\quad (L\cos 45^\circ, L\sin 45^\circ, 0) \\ \mathbf{p}_2 = &\quad (L\cos 135^\circ, L\sin 135^\circ, 0) \\ \mathbf{p}_3 = &\quad (L\cos 225^\circ, L\sin 225^\circ, 0) \\ \mathbf{p}_4 = &\quad (L\cos 315^\circ, L\sin 315^\circ, 0) \end{aligned} \tag{2}$$

Recall that each rotor also generates a spinning torque that is proportional to its thrust, the net torque from rotor i is

$$\tau_i = -\mathbf{p}_i \times T_i\mathbf{z} - \lambda_i T_i\mathbf{z} \tag{3}$$

where $\lambda_i = \pm\lambda$. λ is a constant determined by your rotor and propeller combination, and the sign depends on their spinning direction. After applying Euler's equation, we conclude:

$$\mathbf{J}\dot{\omega} + \omega \times \mathbf{J}\omega = \sum_{i=1}^{4} \tau_i = -\sum_{i}^{4} T_i(\mathbf{p}_i \times \mathbf{z} + \lambda_i\mathbf{z}) \tag{4}$$

where $\mathbf{J}$ is the moment of inertia, which does not change over time. ω is the angular velocity that determines how fast $\mathbf{R}$ changes over time:

$$\dot{\mathbf{R}} = \omega \times \mathbf{R} \tag{5}$$

The derivation of Euler's equation can be found in any physics textbook so I won't include it here.

Now if you look at the summation in Equation 2 and Equation 4 carefully, you will notice that they are both linear to the input thrusts. In fact, these two summations represent the net force and the net torque from all of your rotors, and they can be rewritten as a matrix-vector multiplication between a constant matrix $\mathbf{M}$ and a thrust vector $\mathbf{T} = (T_1, T_2, T_3, T_4)^\top$. To give you a concrete idea, here is the

$\mathbf{M}$ for the quadcopter example above:

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 \\ -L/\sqrt{2} & -L/\sqrt{2} & L/\sqrt{2} & L/\sqrt{2} \\ L/\sqrt{2} & -L/\sqrt{2} & -L/\sqrt{2} & L/\sqrt{2} \\ \lambda & -\lambda & \lambda & -\lambda \end{bmatrix} \tag{6}$$

where each entry shows the contribution of each rotor to the 6 degrees of freedom of your copter. I can further rescale each row to simplify this matrix even more aggressively. This equals a change of unit in some degrees of freedom so it does not affect its correctness.

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{7}$$

Now the quadcopter mechanics should be extremely straightforward to you: starting with a base case where $\mathbf{T} = (mg/4, mg/4, mg/4, mg/4)^\top$, you can quickly verify $\mathbf{MT} = (0, 0, -mg, 0, 0, 0)^\top$. This means the net thrust cancels out the gravity and the net torque is zero so it won't spin in any direction. In order to change roll without affecting other degrees of freedom, we are looking for $\delta\mathbf{T}$ such that $\mathbf{M}\delta\mathbf{T} = (0, 0, 0, \delta, 0, 0)^\top$, and it is easy to verify $\delta\mathbf{T} = (-\delta/4, -\delta/4, \delta/4, \delta/4)^\top$ is a valid solution. This tells you that in order to only tilting the copter to its right, you will need to decrease T_1 and T_2 and increase T_3 and T_4 by the same amount. The rules for changing pitch or yaw can be inferred similarly and you will find they are consistent with the solutions I described before.

Extensions to a pentacopter At this point, you probably already realize that the frame type of a copter only changes the $\mathbf{M}$ matrix. Once you recalculate $\mathbf{M}$, Equation 2 and Equation 4 still apply. To control its rotation along the x axis, for example, you solve $\mathbf{M}\delta\mathbf{T} = (0, 0, 0, \delta, 0, 0)^\top$ in the same way as before except that your matrix and vector dimensions change. This is all the tricks behind the pentacopter model in this document. If you are curious to learn more, I made more aggressive changes in our paper by replacing $\mathbf{z}$ with $\mathbf{z}_i$ for each rotor i to describe the motion of a pentacopter with tilted rotors. It complicated $\mathbf{M}$ even more but the basic idea was still the same.

4.2 Closed-loop controllers

So far I have described an open-loop style "controller" that determines the output thrusts based on target poses without using sensor data. For example, if I want my copter to lift up by 10 meters, I should increase all T_i by some δT. Unfortunately, this controller won't tell you when to stop adding your δT so that it won't fly past the desired altitude. On the contrary, a closed-loop controller makes heavy use of sensors and adjust its output signals accordingly so that you can closely track the desired target. The most commonly used closed-loop controllers are proportional-integral-derivative (PID) controllers. I will briefly explain how PID works in a copter, and show you the actual code to control the pentacopter in the next section.

An altitude PID controller I will use the quadcopter example before to describe how its altitude can be controlled by a PID controller. Let's say the quadcopter is hovering at 0 meter with $T_i = mg/4$ now. Along with Pixhawk there is a barometer which tells its current altitude. Let's further assume the barometer is perfect so that it always returns the perfect altitude without any noises. Our goal is to lift the quadcopter up by 10 meters and we already understand each T_i should be increased by the same amount of thrust δT. The first idea is to introduce a proportional controller, which means the control output is proportional to the difference between the goal and the sensor data:

$$\delta T(t) = k_p(h_0 - h(t))) \tag{8}$$

where $\delta T(t)$ is the value of δT at time t, k_p is a constant, h_0 is our target height (10 meters in this example) and $h(t)$ is the height from the onboard barometer. This means as your copter approaches the desired height, δT decreases to zero. I implemented this P controller and here is what I got (Figure 7):

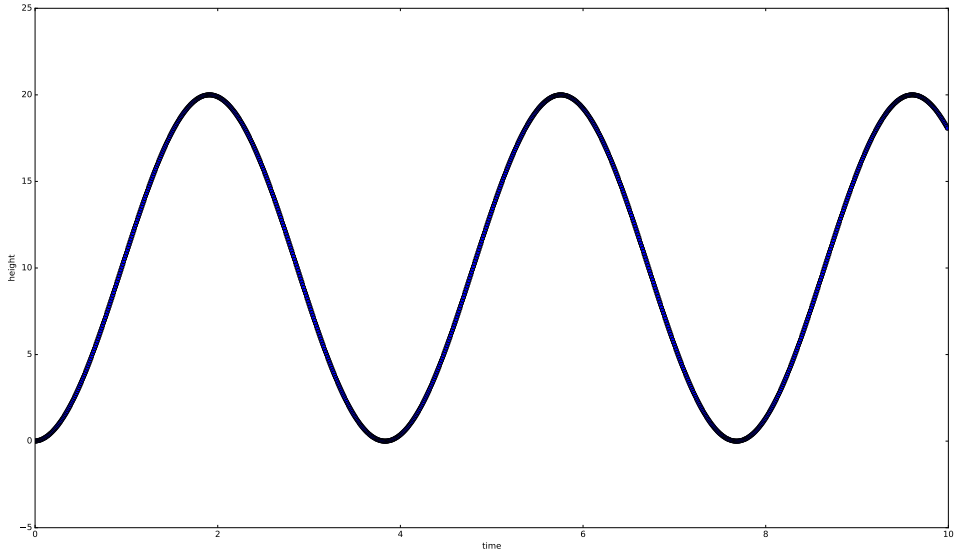

Figure 7: Altitude of the quadcopter running a P controller.

As you can see, the P controller manages to bound the altitude so that your copter won't lift up endlessly. It is not ideal, however, as the height oscillates a lot. This is common when you use a P controller only. In this example, this is because when the copter approaches the target height, its climb rate is still nonzero so it overshoots the target. The P controller then produces a negative δT to drag the copter back. If you have perfect sensor data, this loop will continue forever.

To resolve this issue, here comes the second idea in PID: the derivative controller. Since we also want the climb rate to be zero at the target height, it is natural to apply the P controller to control the climb rate:

$$\delta T(t) = k_p(h_0 - h(t)) + k_d\frac{\delta(h_0 - h(t))}{\delta t} = k_p(h_0 - h(t)) - k_d\dot{h}(t) \tag{9}$$

Not surprisingly, the new controller is called a PD controller. The new term on the right-hand side penalizes the climb rate when it is nonzero. Figure 8 shows the performance of our new PD controller. You can see the oscillation is damped after a few cycles and the height converges to 10 meters in the end.

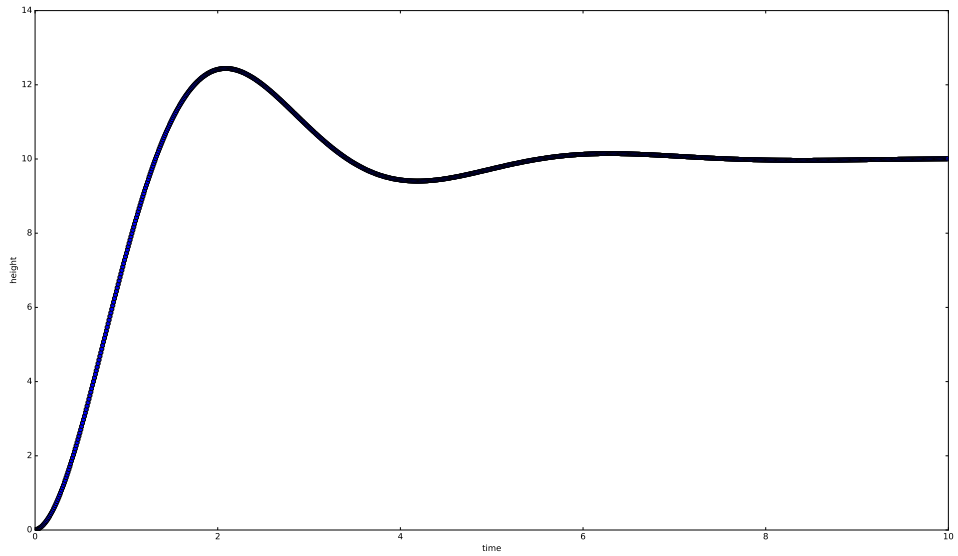

Figure 8: Altitude of the quadcopter running a PD controller.

PD controllers seem great except for one problem: they cannot counter steady-state errors. To illustrate this, recall that the final thrust sent to each rotor is $mg/4 + \delta T$. Now imagine I make a mistake in measuring the weight of the copter: instead of the correct mass m my measurement is $m' = 0.8m$. Now when the copter hovers it has to satisfy:

$$mg/4 + \delta T = m'g/4 \tag{10}$$

Since $m' \neq m$, this means $\delta T \neq 0$ when it is hovering. Comparing it to the definition of δT, we immediately see $h_0 - h = \delta T/k_p \neq 0$. This means the cover is hovering at a different altitude than desired. This is called the steady-state error. Due to the nature of the PD controller, it has no incentive to modify its control output so the steady-state error remains nonzero forever.

To fix the steady-state error, we need to introduce the last idea in PID: the integral controller. The idea is very straightforward: you need a time-variant control signal to break the balance in the steady state, and an integral over the error signal becomes a natural choice:

$$\delta T(t) = k_p(h_0 - h(t)) + k_d\frac{\delta(h_0 - h(t))}{\delta t} + k_i\int_0^t (h_0 - h(t))dt \tag{11}$$

The integral term guarantees to suppress any steady-state error to zero: if this is not the case, you will have a integral signal that grows over time and your control signal becomes time-variant, so you cannot stay in a steady state anymore.

Figure 9 shows the performance of our new PID controller. The previous PD controller was applied before 10 seconds. As can be seen from the plot, the copter stabilizes itself at approximately 8.5 meters, causing a steady-state error. At 10 seconds, I switched to a PID controller, and it quickly responded to the error and increased its height to the desired target (10 meters). The Python source code that generates all the figures in this section can be accessed here.

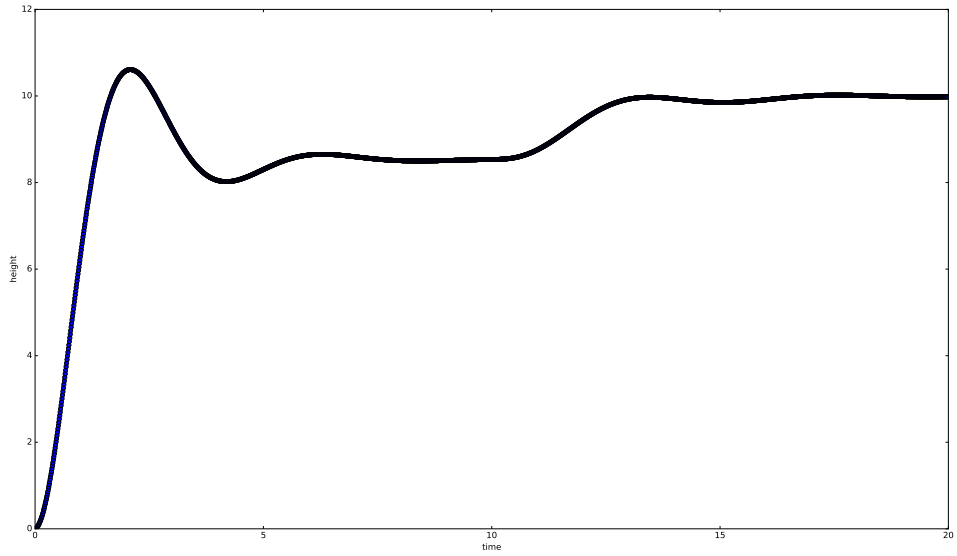

Figure 9: Altitude of the quadcopter running a PID controller.

PID in practice The altitude PID controller is a simplified version of those in real quadcopters. In fact, the dynamics of the vertical motion is so simple that an analytical solution can be derived. In practice, PID controllers are widely used due to its simplicity: even if you don't know the exact dynamics, you can still use a PID controller to attach basically any inputs and outputs. As a result, PID controllers have been developed not only for altitude control in quadcopters, but also for controlling climb rates, roll speed, pitch angle, horizontal offsets, or basically everything.

The harder part, however, is tweaking the PID gains. It generally requires some domain knowledge and much experience. There are both theoretical papers and practical methods for tuning PID controllers. If you are curious to know more, you might want to read this useful guide from ArduPilot.

4.3 Write a pentacopter controller

We will wrap this section up by providing the C++ code we use to control the pentacopter. Our code is based on ArduCopter 3.5.5, which develops a multi-layer PID controller to stabilize the altitude and attitude of a copter with standard frames. To extend its support to a pentacopter, the modification needed is surprisingly little thanks to their high-quality code base.

Modify the control matrix Recall that the only difference between the dynamics of a quadcopter and a pentacopter is the control matrix $\mathbf{M}$. This matrix defines the contribution of each rotor to each degree of freedom. The PID controller is a module built on top of that and does not need any changes. ArduCopter does not explicitly build the $\mathbf{M}$ matrix but it does store the contribution of each rotor to each rotational axis in `AP_MotorsMatrix`:

```
class AP_MotorsMatrix : public AP_MotorsMulticopter {
    ...
    // each motors contribution to roll
    float               _roll_factor[AP_MOTORS_MAX_NUM_MOTORS];
    // each motors contribution to pitch
    float               _pitch_factor[AP_MOTORS_MAX_NUM_MOTORS];
    // each motors contribution to yaw (normally 1 or -1)
    float               _yaw_factor[AP_MOTORS_MAX_NUM_MOTORS];
    // each motors contribution to throttle (1 for most frame types)
    float               _throttle_factor[AP_MOTORS_MAX_NUM_MOTORS];
    ...
}
```

Listing 1: Storing the contribution of each motor

Here I added a new `_throttle_factor` array to support our pentacopter because the 5 rotors have to generate different amount of thrust to balance both the gravity and the torque. These arrays are initialized in the `add_motor_raw` function:

```
void AP_MotorsMatrix::add_motor_raw(int8_t motor_num, float roll_fac, float pitch_fac,
     float yaw_fac, float thr_fac, uint8_t testing_order)
{
    // ensure valid motor number is provided
    if( motor_num >= 0 && motor_num < AP_MOTORS_MAX_NUM_MOTORS ) {

        // increment number of motors if this motor is being newly motor_enabled
        if( !motor_enabled[motor_num] ) {
            motor_enabled[motor_num] = true;
        }

        // set roll, pitch, thottle factors and opposite motor (for stability patch)
        _roll_factor[motor_num] = roll_fac;
        _pitch_factor[motor_num] = pitch_fac;
        _yaw_factor[motor_num] = yaw_fac;
        _throttle_factor[motor_num] = thr_fac;

        // set order that motor appears in test
        _test_order[motor_num] = testing_order;

        // call parent class method
        add_motor_num(motor_num);
    }
}
```

Listing 2: Adding a new motor

Here the `roll_fac` and `pitch_fac` are computed from the rotor position. If you take a look at the `add_motor` function in the same file, you will see how these two values are computed:

```
void AP_MotorsMatrix::add_motor(int8_t motor_num, float roll_factor_in_degrees, float
    pitch_factor_in_degrees, float yaw_factor, float throttle_factor, uint8_t
    testing_order)
{
```

```
    add_motor_raw(
        motor_num,
        cosf(radians(roll_factor_in_degrees + 90)),
        cosf(radians(pitch_factor_in_degrees)),
        yaw_factor,
        throttle_factor,
        testing_order);
}
```

Listing 3: Adding a new motor

Finally, this `add_motor` function is called inside `setup_motors` in the same file. Not surprisingly, you will see a long `switch` statement that sets up the motor configuration for all types of frames. We only need to add a new `case` statement for our pentacopter:

```
void AP_MotorsMatrix::setup_motors(motor_frame_class frame_class, motor_frame_type
    frame_type)
{
    ...
    bool success = false;
    switch (frame_class) {
        case MOTOR_FRAME_PENTA: {
            // for now we ignore the frame type.
            // rcout 1: right midfielder (CW)
            // rcout 2: right back. (CCW)
            // rcout 3: center forward. (CW)
            // rcout 4: left back. (CW)
            // rcout 5: left midfielder. (CCW)
            // if you are a soccer fan you will immediate understand this:
            /*                CF (rotor 3)
                               |
                               |
                               |
                LMF <------Pixhawk------> RMF (rotor 1)
                             / \
                            /   \
                           /     \
                  (rotor 4) LB     RB (rotor 2)
            */
            add_motor(AP_MOTORS_MOT_1, 72, -1, 0.90450844f, 1);
            add_motor(AP_MOTORS_MOT_2, 144, 1, 1.25000007f, 2);
            add_motor(AP_MOTORS_MOT_3, 0, -1, 0.90450844f, 3);
            add_motor(AP_MOTORS_MOT_4, -144, -1, 0.69098298f, 4);
            add_motor(AP_MOTORS_MOT_5, -72, 1, 1.25000007f, 5);
            success = true;
            break;
        }
        ...
```

Listing 4: Initializing the motor matrix in `AP_MotorsMatrix`

The code snippet above shows how I defined and added all of the 5 rotors in my pentacopter by calling `add_motor`. The second argument is the angle of each rotor with respect to the forward direction. The third argument is the yaw contribution. The fourth argument is the desired, normalized thrust when the pentacopter is hovering, which is computed from solving $\mathbf{MT} = (0, 0, -5, 0, 0, 0)^\top$.

In the meantime, we need to add another macro `MOTOR_FRAME_PENTA` in the `AP_Motors` header file:

```
    enum motor_frame_class {
        MOTOR_FRAME_UNDEFINED = 0,
```

```
        MOTOR_FRAME_QUAD = 1,
        MOTOR_FRAME_HEXA = 2,
        MOTOR_FRAME_OCTA = 3,
        ...
        MOTOR_FRAME_HELI_DUAL = 11,
        // Tao Du
        // taodu@csail.mit.edu
        // Jul 4, 2018
        MOTOR_FRAME_PENTA = 12,
    };
```

Listing 5: Adding a motor frame class

That's all the places you need to change your code. The completed version can be accessed in this branch. You can follow the instructions in `README` and upload this customized firmware to your Pixhawk board. After you load the firmware, you will also need to manually change the `FRAME_CLASS` parameter to 12 because that is the value of our `MOTOR_FRAME_PENTA` macro.

5 Fabrication

Our fabrication plan is available online. The `Top board`, `Arm`, `Bottom board`, and `Prop guard` were manufactured using the laser cutter in our lab, and the `Support` was printed using the Objet 3D printer in the machine shop (Figure 10). You also need about 70 M3 screws to assemble the frame and mount the landing gear. For the electronic parts, you need to do some soldering to connect ESCs to the power distribution board. There are tons of tutorials on laser cutting/3D printing/soldering online so I won't talk too much about them. However, I did learn some expensive lessons from my previous build and I hope these will be helpful for you:

- Place your Pixhawk and your battery as far away as possible. The power cable from the battery has the largest current in the whole system, and the magnetic field from your battery will interfere the sensors in Pixhawk. That is why I put my Pixhawk on top of the top board and mount the battery under the bottom board.
- Do not solder ESC wires directly to the PDB. Use connectors instead, and make sure you are consistent with your connector choice. Soldering wires directly to the PDB might save you some weight and space, but you will regret it if you need to replace them after a severe crash.
- Use strip ties to mount all of your cables, especially those near the propellers. I once forgot to mount the charging wires of the battery and it became loose in the flight and got cut by the spinning propeller. I was very lucky because that could have caused a fire on the lawn.
- Try to make every component removable because you might need to replace or repair any of them in the future. For example, do not glue your Pixhawk or battery on the board, but use a hoop-and-loop tape instead.
- Choose thicker wires to solder if you are uncertain. Keep in mind that the peak current in the power cable of your PDB could be 5 times larger than the peak current in each ESC. So do the math before you solder them. You don't want to see your flying copter on fire (although it is very unlikely).

Moreover, ArduCopter has this extremely useful webpage of assembly instructions. It is worthwhile to read them carefully.

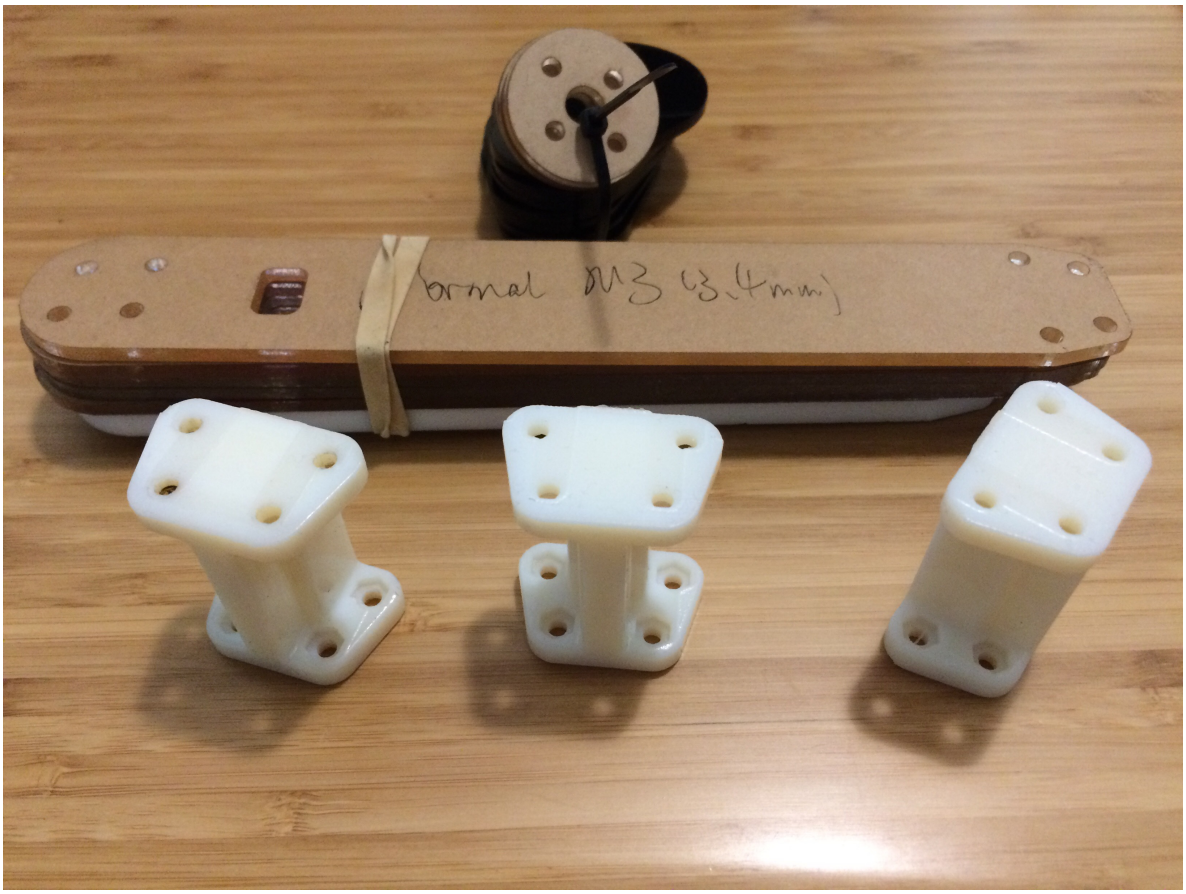

Figure 10: A few spare components made by our laser cutter and 3D printer.

6 Flight Test

Given that we have already updated the controller in ArduCopter, flying this pentacopter is as straightforward as flying a quadcopter. You need to bring your laptop and install a ground station (I used Mission Planner in this project) and find a big field to do your first flight test. Testing it indoor is also possible but would be more challenging.

6.1 Upload the firmware

You can find how to build and upload my customized ArduCopter firmware on this webpage. There are a few customized parameters (e.g., the frame class) you need to specify. The safest way is to override all the parameters in Pixhawk using the `demo.param`, then do all the mandatory setup in Mission Planner to configure your IMU, power module, compass, transmitter, etc. You can find more instructions from this webpage on ArduCopter.

6.2 Test your copter

At this point, you have everything you need to fly or crash your pentacopter. Here are some useful resources that you should probably read before your first flight:

Safety goes first Read this page carefully. Keep in mind that a copter is not a toy, and it can cause serious injuries to people:

- Wear goggles or sunglasses if necessary.
- Do not touch the spinning propeller.
- Keep your copter away from people.
- Don't drain your battery.

Do not skip the pre-arm check Never skip the pre-arm check. Your homemade copter is not as high quality as industrial products and any component could go wrong. Apart from those checks in the link, I would also check the spinning direction of each rotor and each propeller is correct, and each ESC is plugged into the correct RCOUT port in Pixhawk. They cannot be inspected in Mission Planner, but if they went wrong, your copter would flip immediately followed by a catastrophic crash.

I will use my flight test video and photo (Figure 11) to conclude this tutorial. Building your own copter is not an easy task but it's definitely worth it. Wish you good luck. Have fun, and happy flying!

Figure 11: A photo taken from my flight test. You can see we have put lots of efforts into building a safe flight test environment.

7 Acknowledgment

I would like to thank Mike Foshey for teaching me how to use the soldering iron and the laser cutter, and for bringing his dog Buttercup to the lab. She is the cutest dog in the world and provides so much emotional support to everyone in our lab.