

Make: Drones



```
#define FASTLOOPTARGET 10000 // length of inner control loop  
  
unsigned long fastLoopLength;  
unsigned long fastLoopStart;  
  
void loop() {  
  
    // the fast loop for flight control  
    unsigned long fastLoopEnd = micros();  
    fastLoopLength = fastLoopEnd - fastLoopStart;  
    if ((fastLoopLength) > FASTLOOPTARGET)  
    {  
        FastLoopStart = fastLoopEnd;  
        readIMU(); // read raw gyro/accel/mag  
        calcIMU(); // calculate current yaw/pitch/roll  
        doMode(); // calculate yaw/pitch/roll commands  
        doPID(); // smooth yaw/pitch/roll commands  
        doCDS(); // converts cads to motor speeds  
        writeMotors(); // write to pwm  
        mspRead(); // check for new telemetry msgs  
        FastLoopTiming = micros() - fastLoopEnd;  
    }  
}
```

Teach an Arduino to Fly
by David McGriffy

Make: Drones

If you're passionate about drones, get ready to take your skills and understanding to new heights. Drones make a great platform for learning about the fascinating, useful, and even beautiful world of control systems. With this guide, you will learn how to harness the breakthroughs that make possible the remarkable flying abilities of multirotor copters.

The book covers three classes of drones. You'll start by building a small drone using the Hubsan frame and brushed motors. As you move up to a medium-sized drone, you'll use an Arduino Teensy as the core of a flight-control system and add modules for power, sensors, and radios. Finally, you'll build an autonomous drone using the S500 frame kit, brushless motors, and an ArduCopter flight control system that makes a great aerial photography platform.

Inside Make: Drones, you'll find:

- » Hands-on DIY projects for building your own drones using existing frames
- » An in-depth examination of flight-control systems and programming
- » Tips for upgrading and improving off-the-shelf systems
- » A lucid explanation of multirotor aerodynamics
- » Custom circuit board design with modular components
- » Instructions for installing RC radio, telemetry, and GPS systems
- » Fine-tuning and troubleshooting for flight stabilization
- » And much more!

If you're ready to get serious about drones and drone control systems,
this is the book for you. Teach your Arduino how to fly today!

US \$29.99 CAN \$34.99

ISBN: 978-1-6804-5171-9



5 2 9 9 9
A standard linear barcode representing the ISBN 978-1-6804-5171-9.

9 781680 451719

Make:
makezine.com

Make: Drones

Teach an Arduino to Fly

David McGriffy



Make: Drones

by David McGriffy

Copyright © 2017 David McGriffy. All rights reserved.

Printed in the United States of America.

Published by Maker Media, Inc., 1160 Battery Street East, Suite 125, San Francisco, CA 94111.

Maker Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safaribooksonline.com>). For more information, contact O'Reilly Media's institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editor: Roger Stewart

Production Editor: Nicholas Adams

Copyeditor: Charles Roumeliotis

Proofreader: Rachel Head

Indexer: Ellen Troutman-Zaig

Interior Designer: David Futato

Cover Designer: Julie Cohen

Illustrator: Rebecca Demarest

October 2016: First Edition

Revision History for the First Edition

2016-10-05: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781680451719> for release details.

Make:, Maker Shed, and Maker Faire are registered trademarks of Maker Media, Inc. The Maker Media logo is a trademark of Maker Media, Inc. *Make: Drones* and related trade dress are trademarks of Maker Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Maker Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

978-1-680-45171-9

[LSI]

Table of Contents

Preface	ix
1. What Makes a Drone Possible	1
Motors	2
Batteries	4
Control Systems	5
2. Popular Drone Control Systems	9
MultiWii	9
Dronecode	11
APM/ArduPilot	11
PX4/Pixhawk	13
DJI/Naza	14
KK2	14
CC3D/OpenPilot	15
Naze32/Baseflight/Cleanflight	16
Choosing a Flight Control System	17
3. Drone Activities	19
Aerial Photography	19
FPV Racing	20

Education	21
Flying	22
4. Improve the Hubsan X4	23
Goals and Test Methods	24
Battery Upgrade	26
Prop Upgrade	27
5. Build the X4Wii	31
Parts	32
Build	35
Flashing New Code	35
Wiring	37
Assembly	40
Setup and Tuning	42
RC Rates	42
PIDs	43
Modes	45
6. The Visible Drone	49
A Teensy CPU	50
Radios	51
IMU/AHRS	52
Power	53
Circuit Board with Motor Controllers	54
Frame/Motors/Props	55
Weight Budget	56
7. Basic I/O Code	57
Setting Up the IDE	57
Main Loop	59
RC Receiver	60
Bluetooth	63

Motors	66
8. Multirotor Aerodynamics	69
Lift and Thrust	69
Pitch and Roll	71
Yaw	71
Translational Lift	73
Vortex Ring State	74
9. IMU/AHRS	75
Gyros	76
Accelerometers	77
IMU	78
Magnetometers	80
10. Mode and Mix	83
Mode	83
Angle Mode	84
Rate Mode	85
Mix	85
11. PIDs	87
The Algorithm	87
P—Proportional	87
I—Integral	88
D—Derivative	88
Implementation	90
Tuning	90
Theory	90
Practice	94
12. Circuit Board and Motor Controllers	97
Design Tools and Files	98
Component Placement	98

Power Input and Distribution	99
Motor Controller	100
Schematic and Layout	102
Construction	104
13. Construction and Tuning	107
Install the Controller and Connect the LEDs	107
Connect the Motors	108
Install the Radios	109
Complete the Frame	110
Ground Test	112
Flight Test and Future Work	114
14. A Bigger Frame	117
The S500 Kit	118
Follow the Instructions	119
15. Install the Power System	123
Mount the ESCs	123
Wire the Power Distribution Board	124
Mount the Motors	126
Determine Direction	127
16. Brushless Motors and Their ESCs	131
Brushless Motors	131
ESCs	134
BECs	135
17. Install the Flight Controller and Radios	137
Pixhawk Lite and Shock Mount	138
RC Radios	140
Telemetry Radio	143
Safety Switch	144
Testing	144

18. GPS	147
History	148
Theory	148
HDOP	150
Installation	151
19. Magnetometer	155
The Earth's Magnetic Field	155
Declination and Deviation	156
Magnetoresistance	158
Use in Drones	159
20. Flying the S500	161
Install the Props	162
Connect the Battery and Voltage Monitor	164
Radio and Accelerometer Calibration	166
Flight Modes	168
Initial Tuning	168
Final Checklist and First Flight	169
21. Optical Flow, Sonar, and Lidar	171
Sonar	172
Lidar	173
Optical Flow Sensors	173
Visual Odometry	175
22. Vibration	177
Causes of Vibration	178
Effects of Vibration	178
Damping and Isolation	179
Prop Balancing	180
23. Failure Modes and Fault Tolerance	183
Failsafes	183

Redundancy	185
24. Interfaces	187
Computer Interfaces	187
RC Radio Standards	190
DSM/DSM2/DSMX	190
FrSky	191
Servo/ESC Control	191
Telemetry	192
GPS	193
25. The Future	195
Specialization	195
Regulation	196
Ease of Use	196
Safety	196
Human Flight	197
Index	199

Preface

Success

The successful people I have known all seem to have gotten there by some personal mix of talent, luck, and work. I hope that the mix turns out right in this book and that I succeed in inspiring you to make something. It will probably be a drone, but if it's a great new garage door opener, I'll count that as a success as well.

I have put my unique talents into this book, as you will into your project. I was born with a soldering iron in my hand, or almost, thanks to my father's ham radio hobby. Also, largely due to his efforts, I started finding things to program when I was still a kid. I happen to love mathematics, which has served me well as a coder, of course. And there are several pilots on my father's side of the family. But there are other threads in my life that led to drones.

From my mother's side I get music. My love of music and my love of gear came together in recording technology. This led to computer audio and digital signal processing. And now we get back around to drones. Many of the very same algorithms I have coded for audio processing get used in drones to process sensor data and control inputs.

Add in some professional experience doing consulting in the control systems business, and perhaps you can see why I love building drones. It all comes together—building electronics, flying, radios, programming, signal processing. But there was also some luck involved in the creation of *Make:Drones*.

My old friend Tim Deagan was working on his book, *Make: Fire*, and I shot some drone video of his fire projects. Then Roger, who would become our mutual editor, said, "Drones are hot, send me a book proposal on drones," so I did.

Then there is the work. And I don't just mean mine. Roger Stewart makes all those things happen that turn a bunch of files into a book on a shelf. Ron Parsons has saved me from many an error as technical editor. He also let me play on his Processor Technology Sol per-

sonal computer when I was a kid, giving me one of my very first programming experiences. And without the constant support of my wife, Marva, and kids, Stella and Sean, the production and marketing teams would never have had anything to work on.

The real success, of course, is not just a book on a shelf. I hope that this book inspires you to make something that pulls the threads of your life together. I hope that this book helps you see that even something as complex as a drone control system can be broken down into fundamental, understandable parts. Over and over we see that the same technologies that make a drone fly make many of our other modern gadgets possible. I hope that they will make your project fly, figuratively and perhaps literally.

Small, Medium, Large

You can divide this book into three sections, as drones nicely divide themselves into small, medium, and large based mostly on their motors. Small drones use brushed motors directly connected to small props. Medium-sized drones use the same 6- to 8-millimeter motors, but gear them down in order to use larger, more efficient props. Large drones use brushless motors and electronic speed controllers. Each of these classes can actually vary greatly in size, but the general trend holds true.

In Chapters 4–5 we work with small drones. First we take some measurements and try to improve the performance of an existing small drone, the Hubsan X4C. Then we build a new small drone using a Hubsan frame and an open source flight controller. It's called the X4Wii since it uses an X4 frame and MultiWii flight control code.

Chapters 6–13 cover the construction of a medium-sized drone. Once again we start with the frame from an existing drone, the Syma X5. We use an Arduino Teensy 3.2 as the core of our new flight control system, adding modules for power, sensors, and radios. A custom circuit board ties it all together. Finally, so that people can see this new custom controller, we put a clear lid on it and call this project the Visible Drone.

In Chapters 14–20 we build a large drone, based on the S500 frame kit. For flight control, this one uses the powerful Pixhawk Lite controller and ArduCopter flight control software. Combined with a high-performance GPS unit, this system can fly completely autonomous missions—and it has the power to carry a useful payload while doing it. This one will make a great aerial photography platform.

Having talked about what makes current drones possible in the first few chapters, we will look at the future of drones in the last few. Researchers and hobbyists alike continually push the limits of drones with new sensors and ever more sophisticated software. Be a part of this future and make a drone!

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.



This element signifies a general note, tip, or suggestion.



This element indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at <http://www.makedronesbook.com>.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from Make: books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Make: Drones* by David McGriffy (Maker Media). Copyright 2017 David McGriffy, 978-1-680-45171-9."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at bookpermissions@makermedia.com.

Safari® Books Online

Safari Books Online offers a range of [plans and pricing](#) for [enterprise](#), [government](#), [education](#), and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds [more](#). For more information about Safari Books Online, please visit us [online](#).

How to Contact Us

Please address comments and questions concerning this book to the publisher:

Maker Media, Inc.
1160 Battery Street East, Suite 125
San Francisco, CA 94111
877-306-6253 (in the United States or Canada)
707-639-1355 (international or local)

Maker Media unites, inspires, informs, and entertains a growing community of resourceful people who undertake amazing projects in their backyards, basements, and garages. Maker Media celebrates your right to tweak, hack, and bend any technology to your will. The Maker Media audience continues to be a growing culture and community that believes in bettering ourselves, our environment, our educational system—our entire world. This is much more than an audience, it's a worldwide movement that Maker Media is leading. We call it the Maker Movement.

For more information about Maker Media, visit us online:

Make: and Makezine.com: <http://makezine.com>
Maker Faire: <http://makerfaire.com>
Maker Shed: <http://makershed.com>

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at <http://bit.ly/make-drones>.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

What Makes a Drone Possible

1

The quadcopter glides gently through the sky, using every subtle variation in the atmosphere to its advantage, performing that delicate dance with the air that we call flight.

Doesn't quite sound right, does it?

Let's face it, a quadcopter doesn't so much dance with the air as arm wrestle. Usually the quadcopter wins. Some days the sky wins.

No, the beauty of a drone is not in the graceful curve it cuts through the sky or the subtle shape of its wings. It has no wings, of course. In fact, among the best things about our modern multirotors are all the parts they *don't* have. No wings. No control surfaces and no servos to move them. No complex collective pitch hubs. No stall speed, timed turns, or holding patterns. We have stripped flying down to its bare essentials. We have four moving parts that push us upward, and the rest is in the control system ([Figure 1-1](#)).



Figure 1-1 A micro drone in a mint tin

And now we are getting closer to where the beauty lies.

As with the most interesting people and the best books, the *real* beauty of the multirotor drone is on the inside. An ArduCopter-based drone has four main inputs from two joysticks and four outputs to its motors. To compute those four outputs from those four inputs, it uses nearly three hundred parameters. At first it may appear that this is so complex as to be magic, but it's not. It's engineering. And precisely because of all the parts it doesn't have, a drone turns out to be a wonderful platform to learn about engineering principles.

The algorithms used in drone control systems are basic, important, and beautiful. You'll discover that learning how to use them will be fun and interesting. But these methods have been around for decades. So whatever the aesthetics, they are not the breakthrough that made modern quadcopters possible. And make no mistake: there *have* been breakthroughs.

I have flown radio control aircraft off and on for many years and have watched them with great interest since I was a kid. There have long been RC helicopters, but they were expensive and very hard to fly. Then there were toy helicopters, but to watch them fly is to know that they *belong* in the toy aisle. The military built cruise missiles, but they cost millions. Then one day a friend brought a quadcopter into the office. It would fit in your hand and it flew like an aircraft, not just a toy. And the punch line? It cost \$20! Now that's what I call a breakthrough.

So I bought one, then another, then a bigger one. Then I built one from scratch and started to modify the flight control code. After many crashes and much thought, I am now ready to say what has changed: motors, batteries, and control systems.

Drones Versus Quadcopters

You will see several terms for the flying things we will build here. Some take "drone" to mean only a military system or vehicle that can fly autonomous missions. Since it is by far the shortest, easiest word for it, I intend to use the word "drone" for just about anything that can fly without a human on board. The military actually calls them unmanned aerial vehicles (UAVs) and the FAA calls them unmanned aerial systems (UASs), but I use enough acronyms in my life without using one where a one-syllable word

like "drone" will do. The term "multirotor" leaves out fixed-wing aircraft and traditional helicopters, but describes everything we will build here. The term "quadcopter" describes an even more limited class—only those with four, not six or eight, props—but still covers all of our projects. We will use these words to differentiate our drones from fixed-wing or six-rotor vehicles when needed, but also sometimes just for variety.

Motors

That first little quadcopter I saw at the office, like most micro drones, used tiny electric motors. They are smaller but look essentially the same as the small DC motors I bought 10 to a bag as a kid, but those were nowhere close to powerful enough to lift themselves. It

turns out that there is an essential difference between the quadcopter motors and the motors I was familiar with. Like many motors these days, they use rare earth magnets, but that's not the essential difference. These "coreless" motors ([Figure 1-2](#)) are optimized for aviation.

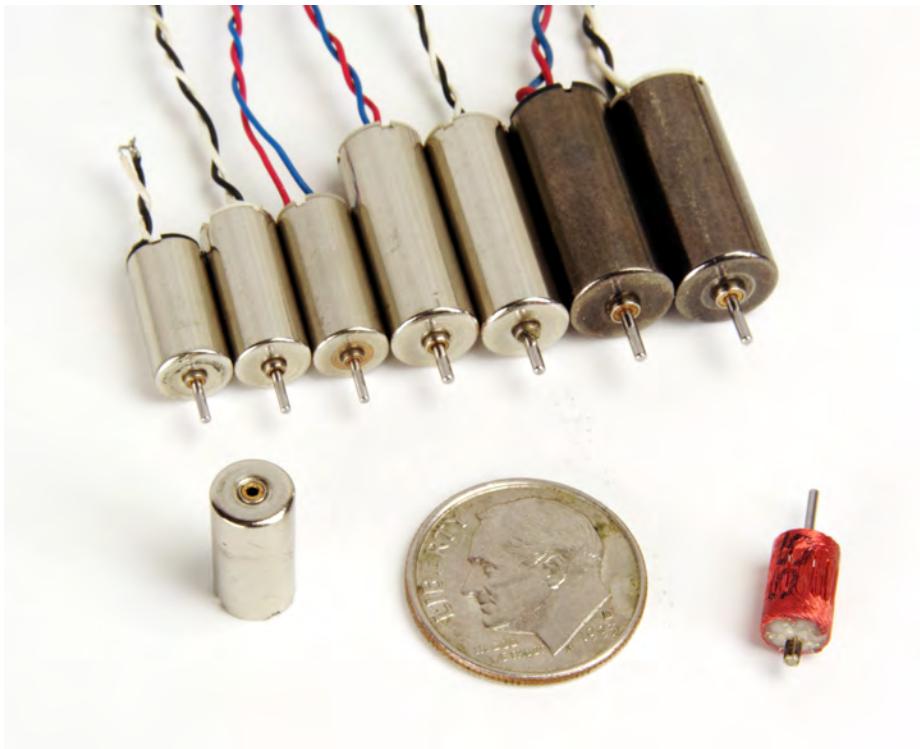


Figure 1-2 6 mm, 7 mm, and 8.5 mm coreless motors

The windings in small motors, like nearly all electric magnets, are generally made around a core of some ferrous material like iron. This makes the magnets more efficient—electrically efficient, that is. It turns out, however, that weight is more important than energy efficiency in aviation, so someone thought to take out that heavy core, and our tiny, flight-ready motors were born.

These coreless motors, used in smaller drones, are often called "brushed" to differentiate them from the sort of motor used in larger multirotors, which have no brushes. In most motors, brushes riding on a series of contacts control which windings get energized in what order. In a brushless motor, first developed for floppy disk drives (see [Figure 1-3](#)), electronics control the windings instead of contacts, making a simpler, longer-lasting motor.



Figure 1-3 Modern drones and old floppy drives use similar brushless motors

Like the drones they fly in, brushless motors gain mechanical simplicity at the expense of control system complexity. Each of the four motors in a brushless quad has to have its own little processor, always watching the back current on the unused coils to determine position, and adjusting the current to the other coils to achieve the speed commanded by the central flight controller.

Batteries

I suppose it should come as no surprise that the power plant is one of the important technologies in a drone. It has always been so in aviation. From that first piston engine that Orville made according to Wilbur's design, to Whittle's first jet engine, to the rocket that Chuck Yeager flew, advances in power have driven advances in aviation. And each new power plant comes with its new fuel source.

For our modern electric aircraft—and this means mostly drones, though there are a few experimental human-carrying electric aircraft—the fuel source is a battery. The most common battery technology used by drones today is the LiPo, or lithium polymer battery, as seen in [Figure 1-4](#).



Figure 1-4 A selection of 1S and 3S LiPos

LiPos come in all sizes and shapes, from batteries the size of a thick postage stamp to batteries the size of a suitcase that can literally run your house or your car. Their capacities and capabilities vary accordingly. They can be rather finicky things compared to say, nickel-metal hydride (NiMH) batteries. LiPos don't like to be left fully charged for long but don't want to be stored completely empty either. Treat them wrong and it is pretty easy to ruin them, or worse. They have a bit of a reputation for catching on fire, but with a few reasonable precautions LiPos will allow us to do things no other batteries will.

Control Systems

In between the batteries and the motors is the control system. Even the tiniest micro drone has several subsystems, including radio receivers, gyros, accelerometers, and, of course, a processor to tie it all together. Since each of these components is just a little black chip on a circuit board that looks much like those out of any modern consumer electronics gadget, people may be less aware of the advances that have made modern drone control systems possible.

Before looking at what's new, however, it's worth pointing out that some parts of these little control systems date back years or even decades. The processors used are fairly modern, but the microcontrollers of several years ago could have done this job as well. On the software side, the algorithms used, including the Kalman filter and PID controller, have been used by control systems engineers since long before the microprocessor was invented.

Having acknowledged the great work of engineers past, however, some aspects of drone control systems do represent new breakthroughs. The general shrinking of electronic components in both physical size and power consumption makes smaller drones possible. Modern design and manufacturing techniques allow these tiny chips to be integrated into task-specific circuit boards that can be economically made in small numbers. A flight controller, containing all the subsystems mentioned above and more, can be put on a single board that you could lose in a shirt pocket.

At least one of the chips on that little board represents a breakthrough that is more specific to aviation: the gyro. Gyroscopes have been used to stabilize aircraft since Lawrence Sperry demonstrated his gyroscopic stabilizer apparatus on a Curtis C-2 biplane to a French audience in June 1914. As mentioned earlier, radio-controlled helicopters were notoriously hard to fly, so someone eventually made a gyroscopic stabilizer for them. The first models used physical spinning disks, just like the children's toys, and thus were rather large and heavy for all but the biggest model helicopters. The price of a single axis stabilizer would buy a complete midsized quadcopter today.

Now we can get three gyros and three accelerometers in a single chip ([Figure 1-5](#)) and they only cost a few bucks, even in small, maker-friendly quantities. And this brings us to one more breakthrough and pleasant surprise. It's nothing new for a consumer gadget like a drone to be made more cheaply in China. What makes this new world of micro aviation so exciting for makers is that Chinese manufacturing has resulted in a range of drone parts that we can buy in single units, mix and match, and make into our own designs.



Figure 1-5 A gyroscope module and its inspiration

For all these innovations, we are yet at the beginning of the drone revolution. As of this moment, the FAA has just announced new rules for commercial drone use; and many types of activities, like package delivery, are not allowed at all. The drones themselves are only just starting to add safety features like collision avoidance and recovery parachutes that will make them safe and reliable enough to share the airspace with each other and with other aircraft. The first international drone racing championship was just held, and everywhere drones are being used in classrooms to excite kids and teach engineering concepts.

And teaching engineering concepts brings us back to what we will do in this book. We will take the tools we are already familiar with—an Arduino, some motors, and a battery—and we will teach them how to fly. In the process, we will learn some basic engineering that can be used in making anything from robots to coffee pots. Then, when the construction is complete and the tuning is done, when you have put in the hours and learned to control your drone, you will be rewarded with one of humanity's oldest dreams: flight, an experience made deeper because your understanding made it possible.

Popular Drone Control Systems



Of the three breakthroughs that made drones possible—motors, batteries, and control systems—I find control systems the most interesting. Perhaps I am biased by a lifetime of writing code, but I am also drawn by the human elements. Of course, real people with stories to tell developed our motors and batteries too, but those stories are trapped inside companies in China. The flight controllers we will look at here are mostly open source, so the story of their development gets played out in public.

In this chapter we will look at several drone control system projects. Each of these projects has developed software that runs on drones for stability and control. Many projects also include ground control station software to monitor and tune drones, or even open source hardware specifications. We'll look at the history of each and sort out the reasons one might choose one system over another.

MultiWii

Yes, the two i's at the end of the name are a reference to the Nintendo video game system. When solid-state gyros were first becoming commercially available, one of the earliest products to use them was the Wii controller. Untold numbers of Wii controllers were taken apart because they were the most reliable place to get the chips. In time, a whole control system grew up using gyros taken from Wii controllers that were connected to Arduinos to control multirotors. This project became known as MultiWii.

The original Wii controller only had accelerometers, no gyroscopes, which wasn't enough to control flight. Using either the Motion Plus extension or the Nunchuck, one could get three axes of gyroscope. Some of the earliest MultiWii models didn't use accelerometers and thus had no absolute reference to the horizon. But stabilizing the roll, pitch, and yaw rates provided just enough control to fly. Today, the MultiWii codebase supports all the common gyro/accelerometer chips, including the currently very popular InvenSense

MPU-6050 in the MicroMWC pictured in [Figure 2-1](#), which we will use later when we build the X4Wii.



Figure 2-1 A MultiWii-based flight controller with its namesake

The original MultiWii multirotors used an Arduino Pro Mini, but the codebase now supports all the common versions of Arduino. One of the advantages of using MultiWii is the number of integrated flight controllers you have to choose from. They all start with some type of Arduino processor and a gyro/accelerometer chip. They may include a radio control receiver, brushed motor controllers, or a compass or barometer. Some will have plugs for easy attachment of external devices such as radios or GPS devices. Which one is right for your project depends on whether you are using brushed or brushless motors, whether you need GPS, and so on.

MultiWii is mature as drone control systems go, and because of this it has given birth to several offshoot projects. These include, most significantly, Baseflight and Cleanflight, which we will cover later in this chapter. This maturity means that there are also a variety of support programs available. MultiWiiConf, the main configuration program, is available for all the major desktop and mobile platforms. Because several of the offshoot projects have maintained compatibility with the MultiWii Serial Protocol (MSP) that it uses to communicate, MultiWiiConf and many of the MultiWii support programs can be used with these other control systems as well.

While MultiWii supports GPS stabilization and there are experimental versions for all the latest sensors, it does have its limits. As the “multi” in the name implies, MultiWii is for multirotors, not for helicopters and certainly not for fixed-wing aircraft. Nor can it run complex,

autonomous missions. (The latter is the strength of the next drone control system we will look at.)

The First MultiWii Tricopter

The first aircraft to use what would become MultiWii was a tricopter built by Alexandre Dubus in 2010 ([Figure 2-2](#)). It had only gyroscopes (no accelerometers), so it had no absolute horizon reference. Nevertheless, reports are that it flew pretty well.



Figure 2-2 The first MultiWii tricopter had no accelerometers

Dronecode

Dronecode is a Linux Foundation Collaborative Projects initiative that combines open source hardware and software projects with APIs and communications protocols to create a rich and wide-ranging development platform for unmanned aerial vehicles (UAVs). On the software side, there are two main flight control projects: APM/ArduPilot and PX4. Each has a related hardware project with which it shares a name, though there is some cross-compatibility. There are also related projects for things like ground stations and simulators.

We will divide our discussion of the Dronecode world into two sections, centered around APM/ArduPilot and PX4/Pixhawk. Let's take the older APM/ArduPilot system first.

APM/ArduPilot

You will see the terms APM, ArduPilot, ArduCopter, and APM:Copter. The most confusing term is APM, because it refers both to the APM 1 & 2 hardware and the APM flight code project. The latter is also known as ArduPilot, and it is a general-purpose autopilot that has variants for different types of vehicles now known as APM:Plane, APM:Rover, and APM:Copter. These were previously known as ArduPlane and so forth. The addition of a GPS, as shown in [Figure 2-3](#), makes this system capable of autonomous flight.



Figure 2-3 An ArduPilot flight controller with power monitor and GPS

The APM:Copter flight control system originated from a collaboration between Chris Anderson and Jordi Muñoz. Chris had been building a plane controlled by a Lego Mindstorms controller (see “[Lego Mindstorms Autopilot](#)”) and Jordi was working on a traditional helicopter. Together they built a plane that won the first SparkFun autonomous vehicle competition in 2009. This plane used thermopiles, an array of six photocells, to stay level by keeping track of the sun. Chris and Jordi went on to found 3D Robotics and the controller from that plane went on to become ArduPilot and then APM.



3D Robotics

It is important to remember that 3D Robotics is synonymous with neither APM nor PX4. 3DR is perhaps the largest, but not the only, retailer of the hardware. While it is a key sponsor of the various Dronecode projects and sells products based on them, 3DR also extends these systems so that its products have many unique capabilities as well.

To add a bit of naming confusion, the first hardware that ran ArduCopter was called ArduPilot, a name now used only for software. Other earlier versions of the hardware for running ArduCopter were called APM 1 through APM 2.6. Only the 2.x boards are still supported as of this writing. These older flight controllers use 8-bit microcontrollers and can only run versions of ArduCopter up to 3.2.x. To run the newer versions at the speed required to keep up with aircraft requires more power. This is where the cross-

compatibility between different Dronecode projects comes into play. To run APM:Copter software version 3.3 or later, you need one of the more powerful platforms. Several options are available, including one based on the Raspberry Pi 2, but the most common choice is PX4 hardware.

Lego Mindstorms Autopilot

Chris Anderson built this autopilot, based on a Lego Mindstorms NXT controller ([Figure 2-4](#)). He was working on building an autonomous aerial vehicle for under one thousand dollars. One measure of how far we have come is the fact we can build one for closer to one hundred dollars today. This first design put the normal R/C servo on top of the NXT servo so that moving either one moved the rudder. That way the pilot could control the plane normally or engage the autopilot and let the autopilot execute a one hundred eighty degree turn then fly level until the pilot takes control again.



Figure 2-4 Chris Anderson's Lego Mindstorms Autopilot includes both a Lego and tradition R/C servo

PX4/Pixhawk

The Pixhawk and PX4 flight controllers have 32-bit processors with floating-point units, improved sensors, and much more connectivity than the older APM hardware. They run the NuttX operating system and feature a middleware layer, also called PX4, that gives the flight control application a consistent interface regardless of the hardware being used. The flight control application, PX4 Autopilot, flies the aircraft. The PX4 hardware and middleware also support the APM/ArduPilot stack.

The PX4/Pixhawk software stack and hardware specifications were developed with support from the ETH Zurich research institute in Switzerland, as well as many other labs and individuals. Advanced controllers, I/O boards, and sensors are available (including, notably, the PX4Flow optical flow sensor that allows aircraft to remain fixed over a spot on the ground without using GPS). The PX4 system also supports research projects doing advanced vision work, using either internal or external cameras to provide position information to the autopilot. Due to its advanced autonomous flight abilities, PX4 is an obvious choice for projects involving the simultaneous control of multiple aircraft.

To sum it up visually, the world of APM/PX4 looks something like the diagram in [Figure 2-5](#).

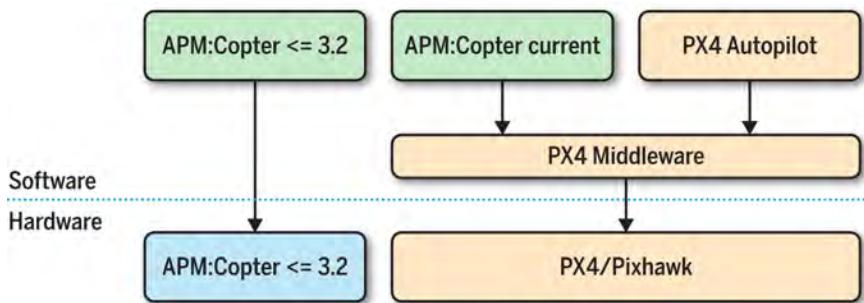


Figure 2-5 Diagram of the APM/PX4 system

DJI/Naza

If imitation shows some kind of respect, then DJI is surely the most respected drone maker in the world. Every all-white, fully enclosed, X-shaped drone with red and silver bands on its arms is a form of homage to DJI's Phantom line of quadcopters. The control systems that run these and DJI's larger commercial multirotors are versions of its proprietary flight control system, Naza. DJI specializes in aerial photography and, with the Phantom series, is making drone photography accessible to an ever-larger audience.

DJI's expanding audience includes many people who just want photographic results and who are not so much interested in spending hours learning the art of flying. While they may be missing out on the beauty of aviation, they have a job to do. The real-world applications drones are used for, from ranching to real estate to search and rescue operations, deserve hardware that is as reliable and easy to use as possible, and DJI is leading the pack.

The technology that DJI uses to make its products easy to use includes receivers combining GPS and Glonass satellites and optical flow sensors. In its control systems, DJI pays careful attention to tuning the flight control parameters. Because DJI sells complete integrated systems, it can do this precisely for the particular hardware being used. Its batteries and charging systems are more foolproof, and more expensive, than most. DJI builds its own cameras and gimbals, and thus these systems are highly integrated as well.

The Naza control systems used in these integrated packages are also available separately for use in custom drones where reliability and smooth operation are critical. They offer advanced failover systems and modes specially tuned for getting photographic results. Some may complain that Naza-based drones aren't as fun to fly, but judged by their own goals they are hard to beat.

KK2

KK2 flight controllers started with the KKMulticopter board designed by Rolf Bakke. The first features you notice on almost all KK2 boards are a screen and four buttons, as seen in [Figure 2-6](#). This enables the principal advantage of the system, which is that it can be com-

pletely configured and tuned without the use of a PC or even a smartphone. Recent versions update the system with the latest gyros and even a backlit screen. And they are cheap. With a screen, buttons, and a processor for under \$20, it's a wonder that these boards aren't more often repurposed for projects other than building quadcopters.

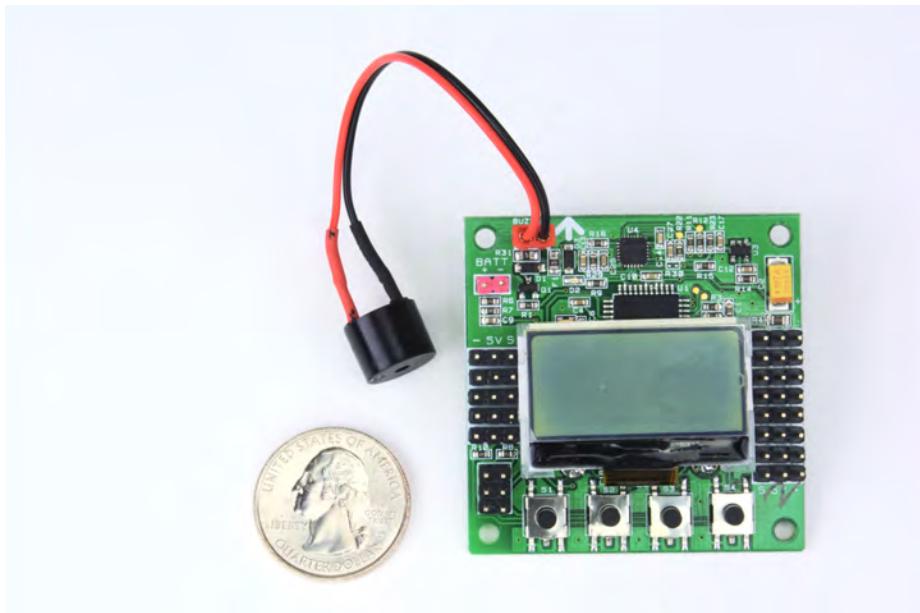


Figure 2-6 KK2 controllers traditionally include a screen

The KKs still use 8-bit CPUs. Given, however, that they are not trying to do operations such as GPS waypoint navigation or collision avoidance, they work quite well. In fact, solid flight performance is the other thing KK controllers are known for. In the early years—they go back to 2012—KKs did a better job than most systems using the limited gyros that were then available. Today, with the same InvenSense MPU-6050 six-axis gyro/accelerometer chip that everyone seems to be using, the flight performance of the KKs is quite good. What really sets these boards apart, though, is that little screen (and the bigger ones it lets you do without).

CC3D/OpenPilot

The OpenPilot project espouses similar goals to DJI: providing a stable, easy-to-use platform for aerial photography or robotics. But unlike the DJI systems, OpenPilot is open source and nonprofit; keeping costs down is another of the project's goals. In addition to multicopters, the system supports helicopters and fixed-wing aircraft. Like the APM systems, OpenPilot supports flight control software, ground stations, and a couple of hardware options.

The OpenPilot flight control software includes a complete flight-plan scripting language, supports wireless telemetry, and has all the features you would expect of a modern drone control system. The latest versions require the latest hardware, of course, which is called Revolution or Revolution Nano. But it's the *old* hardware, called CC3D for ControlCopter 3D (see [Figure 2-7](#)), that really makes this whole project significant.

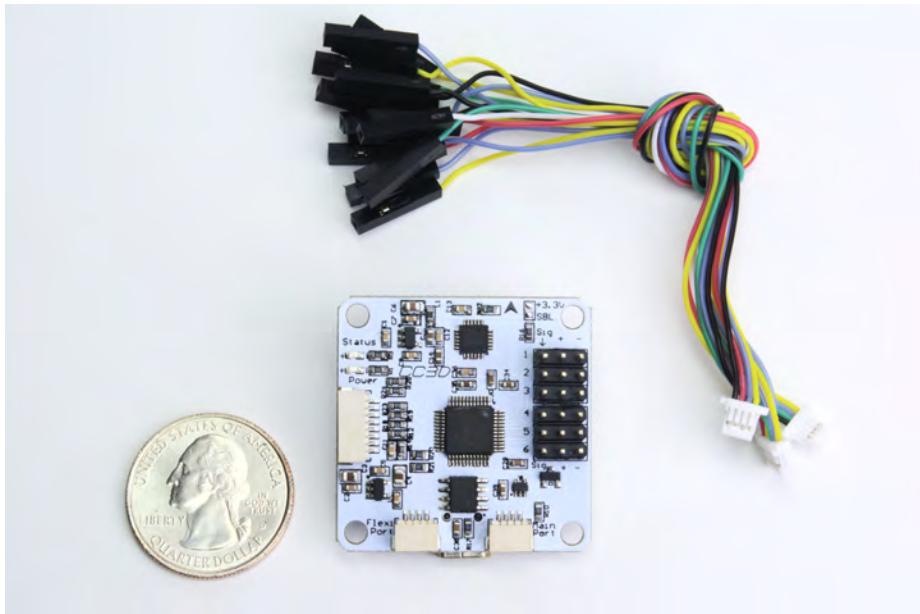


Figure 2-7 This CC3D board can run several flight control systems

The CC3D board is one of the most cloned and, thus, most widely available and inexpensive flight controllers. The KK2 controller is also inexpensive and widely available, but CC3D gives you a choice of several flight control systems to run on it. It supports not only OpenPilot but also the Baseflight and Cleanflight systems, which we will cover next. Not bad for a controller that can be bought in a case for less than the cost of most Arduino boards!

Naze32/Baseflight/Cleanflight

So, we have come full circle. We started with MultiWii and now we are back to a group of MultiWii forks. Naze32 (not to be confused with the DJI system, Naza) brings MultiWii into the 32-bit world with its AfroFlight line of hardware. Baseflight, the software that goes with Naze32, was forked from MultiWii to take advantage of the 32-bit processors that were becoming available. Because of its MultiWii heritage, Baseflight can still use MultiWiiConf as a base station, but BaseflightConf base station software is available as well. Here the story turns a little ugly, though.

Another developer came along and thought he could do better yet. He forked the Baseflight code, cleaned it up, and called it Cleanflight. And there is a CleanflightConf base station program. The original developer of Baseflight was not happy about this new fork. You might want to read the history of the disagreement and make your choice about which to support.

Based purely on flying qualities and other features, the two are reasonably close. Each flies well enough to have its own set of supporters, especially among the drone racing crowd. So what differentiates them? Baseflight is a bit more mature, with less frequent updates and less support for new features. Cleanflight gets updated more often and will certainly be the first to support things like GPS. Perhaps most significantly, Cleanflight will run on a CC3D board, making it a very economical option, and there are now a couple of Cleanflight boards with integrated brushed motor controllers for making micro drones.

Choosing a Flight Control System

We've looked at a few of the most popular drone control systems, but there are certainly more. Tau Labs is a fork of OpenPilot that deserves a mention. Paparazzi is an open source flight controller project that includes the Lisa/S autopilot, the smallest flight controller that includes GPS and thus can perform completely autonomous flights.

So how do you choose? In many cases, there are several options that will do a fine job. After making sure your basic requirements are met, you can usually choose based on price or because of a feature that strikes your fancy. [Figure 2-8](#) is a Venn diagram showing the domains in which each system tends to be used most often.

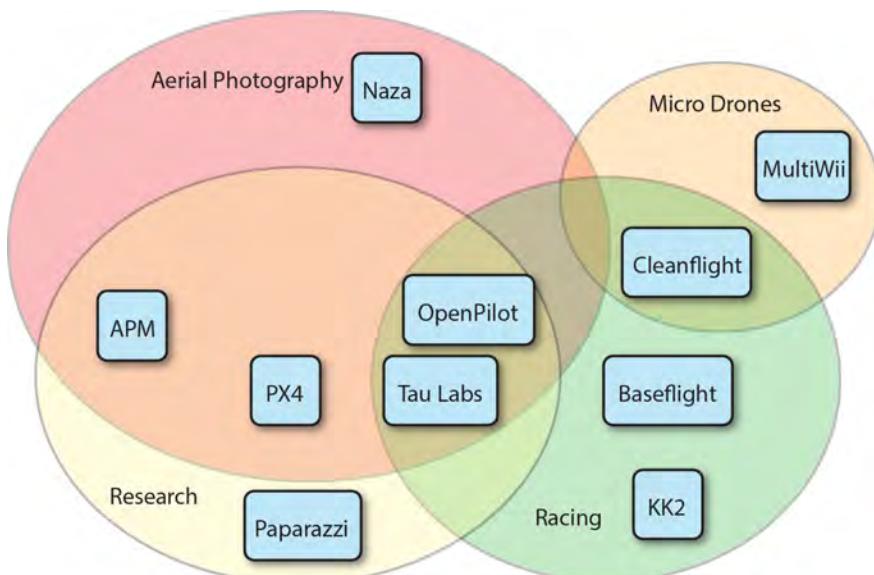


Figure 2-8 Common uses of different drone control systems

Choosing a Flight Control System

We will get more experience with making these decisions as we choose controllers for our hands-on construction projects in the coming chapters.

Drone Activities

3

While the main focus of this book will be building drones, let's stop for a moment to consider what we can do with them once they are built. Different drones for different jobs, of course. The smallest toy drones might be mostly for fun, but they can be great for training too. Most larger drones carry a camera, but aerial photography can be used for anything from real estate to search and rescue.

While we may customize each drone for a particular activity—bigger motors and props for a heavy camera, small and low-latency video gear for an FPV racer—the design principles and even much of the equipment gets shared between these subgroups. Once you've built the projects covered in this book, you will be well prepared to dig into any one of these activities.

Aerial Photography

It seems like every drone in the store has a camera on it now. Most of these are good for little more than aerial selfies, but you can also see what your home or flying field looks like from the air. Step up to better gear and you have the basics for aerial real estate photography, which has been one of the first really viable businesses you can run with a drone.

Aerial cinematography—taking shots for movies from the air—has been around for a long time. Before the development of modern multirotors, Hollywood used RC helicopters for the job, but these were large, expensive, and notoriously difficult to fly. Today filmmakers use hex- and octocopters, still large but much easier to fly thanks to modern control systems. The combination of high-performance medium-sized or large drones like the final project in this book with action cameras like the GoPro means that even home movies can use aerial photography now ([Figure 3-1](#)).



Figure 3-1 The widely available Cheerson CX-20 can carry a GoPro and a gimbal

Not everything shot with a camera is art, of course—by which I mean things like mapping, not just bad photography. Drones can easily be programmed to follow grid patterns and record the GPS positions of each frame as they go. Specialized software then stitches the images together into a complete map of the covered area. Complete turnkey solutions for this have been around for a while, and now they are adding features like infrared cameras and lidar for more accurate height data.

Like their ground-based robot cousins, drones get sent into places where humans can't or shouldn't go just to look around. Search and rescue teams use drones to search for lost people. Disaster recovery teams might use drones to survey the area after a tornado. Ranchers can use drones to track down lost cattle. Most of these jobs have been done with traditional helicopters or ultralight aircraft in the past, but drones can do much of the same work at drastically lower cost and risk to humans.

FPV Racing

The very definition of first-person view or FPV racing includes a camera, yet this activity doesn't have much to do with aerial photography. To begin with, the drones rarely get that far off the ground. The race courses usually include multiple barriers or gates that the pilots must pass through to complete a circuit. The pilots themselves just sit on the side-lines somewhere, stare into their FPV goggles, and move the sticks on their transmitters.

The pilots control their drones entirely through the video downlink viewed in an isolating headset, so that one looks out from the point of view of the drone (that is, in first person).

The cameras used for this must be pretty small, and good low-light response helps a lot. FPV racers won't use a gimbal to steady the picture because they need to know which way the drone is pointing. Actually, it's not uncommon for a racer to carry an action cam in addition to the main FPV camera so the pilot can watch a recording of the flight later.

The FPV racing community has settled on the 250 class as the size for the top racers. This means that they measure 250 millimeters from motor to motor. They usually take the form of an "H" rather than an "X" like the drones we will build here. This design favors the mostly forward flight that they are built for. [Figure 3-2](#) shows a typical 250 class FPV racer.



Figure 3-2 Benjamin Tovar's 250 class FPV racer

Education

Kids love drones, and I believe that drones have a lot to teach them—and the rest of us as well. Kids can be hooked in by the chance to fly a drone and then go any number of ways, depending on their interests or the setting. A science class might learn about the physics of flight, or might use the drone as a platform to plan experiments, gather data, and analyze results, just as we will soon when testing different props and batteries.

Older kids (no age limit here!) can build complete drones following the instructions here, or build their own using the same skills. A complete project includes planning and design, as well as learning to predict performance before you build it (useful skills in many fields). Students can learn about electronics construction on a fun platform, or, for those more comfortable with a keyboard than a soldering iron, learn to configure all the flight parameters and analyze flight logs using ground control station software.

Flying

Of course, drones are for flying. But I believe the public overlooks this aspect of why we fly drones too often. Each of us may do one or more of the activities we have just talked about with our drones. But I find that the people who stick with the hobby, the ones who can't wait to get to the field, are the people who just love flying.

Many drone pilots hold traditional pilot's licenses. Quite a few have flown other types of RC aircraft in the past, or perhaps still do. Most of us will gladly take a ride in just about anything that flies or stand and watch for hours as luckier pilots than us get to fly fun-looking aircraft for us at airshows.

They say flying is one of our species' oldest dreams. I have certainly dreamt of it, and taken part whenever I get the chance, since I was a kid. Flying generates science and poetry. Sometimes, when the latest project crashes more than it flies or for any number of reasons I get stuck in front of a screen for too long, I try to remember to take a micro quad out in the front yard and fly for a while. Perhaps some kid will walk by and get hooked.

Improve the Hubsan X4

4

The Hubsan X4 series includes some of the most popular micro-sized quadcopters available ([Figure 4-1](#)). They come in a variety of versions ranging from plain to those with live video feeds, but they are all about the same size, and they feel very similar to fly. Spare parts—including props, batteries, motors, frames, and controllers—are inexpensive and widely available.



Figure 4-1 The Hubsan X4 107C will be our test platform for this chapter

This is all great for learning to fly and for everyday stick time, but as makers we always want a little more than the manufacturer specs give us. In this chapter, therefore, we will see if we can improve the performance of the Hubsan X4 107C with different props and batteries. Along the way, we will learn some test techniques and gather data that can help us later with other designs.

Goals and Test Methods

To begin, we need to define what we are going to improve and how to test for it.

One of the most talked-about specifications for any drone is flight time. In the real world, flight time depends on factors such as the kind of flying you're doing, the wind, how beat up your props are, and so forth. We're going to try to control for most of these factors so we can focus on just two of the things that affect performance and are easily changed: batteries and propellers.

To control for type of flying, let's first restate our goal more precisely as "hover time" instead of just flight time. Since hovering smoothly in place is not that easy, we will control for flying style and ability by building a test stand that limits the quadcopter's motion to up and down and thus reduces our required control inputs to throttle only.

This test stand is very simple, and it's easy to build. It lets the quad hang off the edge of a table to allow free downward airflow. But before describing its construction, it's worth pointing out a bit about how engineering works. Not only do we learn design patterns and algorithms from the works of others, but we get test data too. If you are flying an X4 107C on one of the batteries tested here, then there is no reason for you to retest. If you find a great new battery, or want to do similar tests with a new drone design of your own, then here's the parts list and a picture of what it looks like before it's built ([Figure 4-2](#)):

- 10" of 1x4, cut into two 4" pieces and one 2" piece
- 18" of 1/4" square dowel
- One 3" #4 machine screw, two #4 nuts, and several #4 washers
- Two Velcro straps
- A bit of wood glue and/or 1 1/2" wood screws
- A 5/32" drill bit



Figure 4-2 The parts for this test stand are available at any hardware store

The three pieces of 1x4 go together in a "U" formation, held together with glue and/or screws. Glue is stronger, but screws are quicker. Be sure to pre-drill if you use screws since 1x4s tend to split. Drill a 5/32" hole in the exact middle of the 1/4" square dowel. Drill another 5/32" hole in the middle of the two sides of the "U" and bolt the dowel into it with the 3" screw. It's best to use washers on each side of the dowel and on the outside of the 1x4s, though it's rather tricky to hold them in place while pushing the screw through. The first Velcro strap is for balance. With the drone *not* on the end, move the strap around to achieve the best balance possible. Then use the second strap to secure the drone. The only critical points in this design are that the dowel move freely and be relatively balanced. [Figure 4-3](#) shows what it looks like ready for a test run.



Figure 4-3 This fixed test stand limits motion to up and down for more reliable tests

Battery Upgrade

The easiest part to upgrade on most quadcopters is the battery. We remove it after every flight, so why not just put a better one back in? But what will give us better hover time? The lighter battery with less to lift, or the bigger battery with more capacity? Let's test several batteries and see.

First we have to find some candidate batteries. The Hubsan X4 107C comes with a 380 mAh, 25C, 1S LiPo. The 380 mAh is a measure of the total energy that the battery can deliver between the fully charged and fully discharged states. More would give more hover time, except that the battery would also get heavier.

The 25C is a measure of how fast the battery can delivery its energy. A 1C battery can empty itself in one hour. A 25C battery can theoretically empty itself in $60/25 = 2.4$ minutes. One can also multiply this 25 by the 380 mAh capacity, giving 9,500 mA or 9.5 amps of current. The quadcopter won't draw this much current all the time, or perhaps ever, but it requires batteries that can deliver this much when needed.

The 1S LiPo means that it is a single lithium polymer cell. There are lots of smaller 1S LiPos, but if they have the same 25C rating, then they probably won't be able to deliver enough current. For testing, suppose we've found a 260 mAh battery with a 35–70C rating. When the C rating is a range, the low number is the sustained output and the high one is the peak. The low end of this range only gives 9.1 amps, not quite what the original can do, but

the 70C peak will certainly be enough. On the larger end, there are many 500 mAh batteries that fit in the 107C, and a normal 25C rating times this larger capacity will be plenty.



Warning—Don't Use Underrated LiPos

For example, the stock 240 mAh, 25C battery delivered with the smaller Hubsan X4 107L is not enough for the 107C we are using here. Using a LiPo that is not rated to deliver enough current will result in the battery overheating and “puffing” up or even catching fire. A tiny amount of expansion after a flight is normal, but if a battery stays puffed up it should not be used again.

The test procedure will be to strap the quadcopter down, plug in the battery, turn on the transmitter and let it bind to the receiver, then start the timer and give it just enough throttle to keep the quad level. This will be a little less at the start; climb slowly then quicker until the lights blink a low battery warning. The measurement will be the time between throttle up and low battery. **Table 4-1** lists the results for the batteries described.

Table 4-1 Hover time for different batteries

Description	Capacity	Weight	Hover time
Smaller 35-70C	260 mAh	7.7g	4:58
Stock	380 mAh	10.6g	6:12
Larger 25C	500 mAh	13g	7:22

The larger battery has almost twice the capacity of the smaller one, but gives less than 50% more hover time. Overall, the larger the battery, the longer the hover time. This relationship will be limited, though, as too small a battery will not have the power to lift the drone and the drone will not have the power to lift too large a battery.

Does this mean that we should always choose the largest battery that we can? Not at all. Hover time is only one parameter, and it comes at the cost of performance. The drone does not climb as fast or maneuver as well with the heavier battery. We might choose the smaller battery just for the fun of the higher performance. Or perhaps we need to add a few grams to the drone elsewhere and want to use a smaller battery to hold the all-up weight constant, even if this means sacrificing a little flight time.

Prop Upgrade

Propeller design requires a difficult blend of art and science. One might think that a century of design work would have given us the perfect prop, but even today we see very different designs for the same purpose. One might be extra wide (i.e., have a longer chord),

in toward the hub and another almost no chord at all in the middle. At first glance a big prop with a wide chord and an aggressive pitch angle might look like it will give more thrust, but most gas engines and electric motors give more power as they spin faster, and that big prop will slow them down. This is why in a plane with a variable pitch prop, maximum horsepower is achieved with the flattest pitch.

Thankfully, we don't have to design propellers from scratch. We will limit ourselves to testing a few of the props available to fit our quads. Since the only requirements are fitting the common 1 mm shaft and not being so big they hit each other, we can easily find several to choose from ([Figure 4-4](#)).



Figure 4-4 A selection of small propellers that will fit a 1 mm shaft

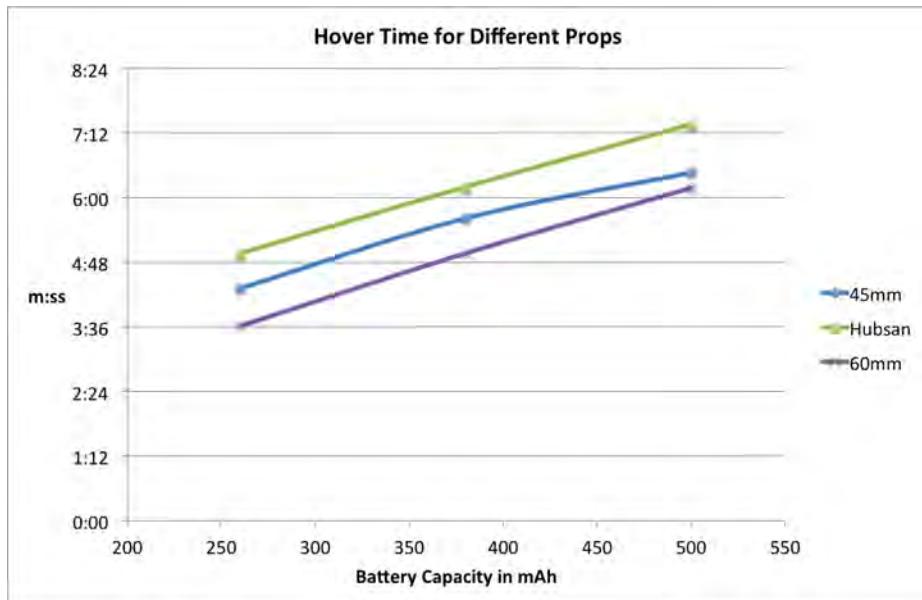
Our test procedure will be exactly the same as that described earlier, testing each set of props with a small, stock, and large battery ([Table 4-2](#)). In fact, our test runs from the previous section will serve as the first series for this experiment. Note that with the exception of the stock Hubsan props, the hubs of these propellers needed to be drilled out to fit. A #62 drill bit is ideal, but they are hard to find; the shaft ripped out of an old micro motor, wrapped in tape to fit the drill chuck, can be used as a poor substitute. Either way, be very careful to get the hole straight.

Table 4-2 Hover time for different props

Description	260 mAh	380 mAh	500 mAh
45 mm Hobby King	4:19	5:37	6:28
55 mm Hubsan	4:58	6:12	7:22
60 mm Graupner	3:37	4:59	6:11

In an interesting result, the props on the lower left in [Figure 4-4](#), the Graupner 55 mm model, didn't generate enough lift for even the lightest battery. The low battery warning didn't go off for well over four minutes, but for most of that time even full throttle was not enough for the drone to lift itself.

In the end, nothing beats the stock prop. This is not really an unexpected result, as tests of other props for micro drones have similarly failed to find a better one. Of course, each of these props may be the ideal prop for some other application. [Figure 4-5](#) shows what the relationship between battery capacity and hover time looks like for each of the props tested.

**Figure 4-5** A plot of the linear relationship between battery capacity and hover time

And what have we learned from this? Not that one combination of prop and battery is best. For props, add price and availability to performance and the original Hubsans are hard to beat. For batteries, if you value long hover time, the 500 mAh batteries have real benefit at an acceptable loss of performance. But it may turn out more interesting to us to

know that there is a lighter battery that can still handle the load. This at least allows us the choice of sacrificing flight time for payload.

Propeller Failure

The first fatal accident in a powered airplane occurred when a propeller of a new design failed while Orville Wright was performing the first flight carrying a passenger as part of demonstrating the Wright Flyer to the US Army ([Figure 4-6](#)). That passenger—Lt. Selfridge, the army's most experienced aviator in those days before the airplane—had worked with Glen Curtis and Alexander Graham Bell in the Aerial Experiment Association that Bell led. Orville might have saved the plane but he was also using a new stick control for the first time, which might have delayed his responses. Orville recovered from his injuries but would suffer from sciatica for the rest of his life.



Figure 4-6 The crash that killed Lieutenant Thomas Selfridge was caused by propeller failure

As a final word on performance, it's worth noting why we haven't looked at changing the other main component in this drone's power system: the motors. There are more powerful ones available that will fit the X4 (0820 motors, which are 8.5 mm in diameter and 20 mm long). Perhaps the best reason not to try such motors is that they draw significantly more current, and we don't really know what the X4's controller is rated for. Also, the increased thrust they would certainly give would come at the expense of flight time, and as we have just seen, we can achieve much the same at lower expense and lower total weight by just using a smaller but powerful battery.

Next, we'll move to the other component besides motors that we haven't changed: the flight controller.

Build the X4Wii

5

In the previous chapter, we ran a series of tests on the Hubsan X4 107C to find out how changing props and batteries affects performance. Now we are going to try changing that most interesting of components, the flight controller. In fact, we are going to build a new drone from scratch, using the frame, motors, and props from an X4, but putting in an open source flight controller. We'll use the MicroMWC mentioned in [Chapter 2](#) (see [Figure 5-1](#)).

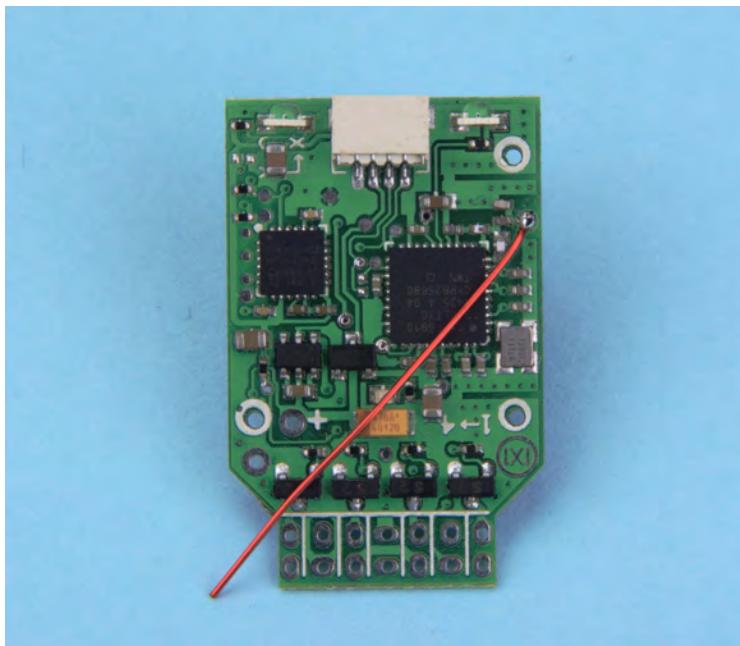


Figure 5-1 This MicroMWC flight controller will give our X4Wii its character

Parts

You will be surprised how the personality of the drone changes when flying with different software. If you have flown an original X4, you may not like the new feel as much at first. But now you can tune the system to your needs, whether that be fast for racing or smooth for video. And you can include accessories that are not available with the stock X4.

Where the original Hubsan X4 107D has a video camera and transmitter, we will substitute a Bluetooth module. With it, you can connect a GUI on your computer or smartphone to the X4Wii without stopping to disconnect the battery and connect a USB cable each time you want to change something. This makes the X4Wii a great platform for learning things like how to do proportional–integral–derivative (PID) tuning. When you have tuned the controller just the way you like it, you can then switch to manual mode and learn acrobatic flying!

Parts

As mentioned, the X4Wii will use the frame, motors, and props from a Hubsan X4 107D. Rather than buying each of these items separately, it is much cheaper to get what is called a crash kit ([Figure 5-2](#)). The kit includes a frame, props, motors, and batteries.



Figure 5-2 A “crash kit” contains many of the parts we need

No electronics come with the crash kit. Instead of stock electronics, we will use the items pictured in [Figure 5-3](#), including, from left to right, an FTDI programmer, the MicroMWC flight controller, and an HB-02 serial Bluetooth module. I suggest getting the FTDI adapter packaged specifically for the MicroMWC because it comes with the two cables also shown.

One of the cables fits the MicroMWC, but it just so happens that the other cable, meant for a MiniMWC, fits the Bluetooth module. So, we can combine these two cables and build one that connects the MicroMWC to the Bluetooth module. While not required, I'd actually suggest buying two of these adapters: one for the two cables and another to keep handy for later reflashing of the MicroMWC.

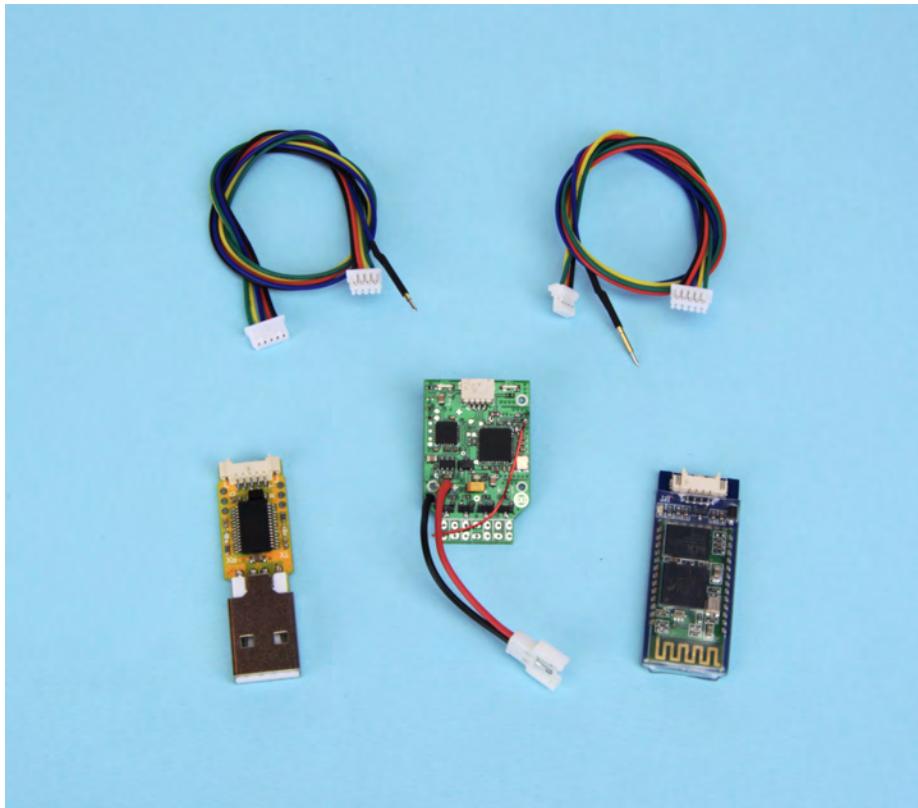


Figure 5-3 A MicroMWC flight controller with accessories

The built-in receiver on the MicroMWC board understands the DSM2 protocol. Since the original Hubsan transmitters use a proprietary protocol, we will need a new transmitter. The Blade MLP4DSM, shown in [Figure 5-4](#), does everything we need and can be bought for under \$20 used. Any DSM2 transmitter with at least four channels should work, though having at least one additional channel lets you change flight modes easily. The MLP4DSM handles this by having a push of the right stick toggle the AUX1 channel and a push of the left stick momentarily change the AUX2 channel, meaning it is actually a six-channel transmitter.



Figure 5-4 The MLP4DSM uses stick pushes for its fifth and sixth channels

We will also need a few other parts, like a battery connector, heat shrink tubing, a few little bits of wire, and some even tinier screws. [Table 5-1](#) is the complete list.

Table 5-1 X4Wii parts list

Part	Quantity
Hubsan X4 107D "crash kit"	1
MicroMWC flight controller	1
FTDI adapter for MicroMWC	1 + 1 optional
Bluetooth module (HB-02)	1
Male Micro Losi connector	1
Heat shrink tubing, 1/16"	6"
Wire (~36 AWG)	6"
Screws (1.6 mm x 4 mm)	3
DSM transmitter (MLP4DSM)	1

Build

The first thing we need to do is flash new code to the MicroMWC. When I built the first X4Wii, it kept flipping over. It thought up was down and left was right. It's an easy fix for a drone, but one that requires a code update.

Flashing New Code

Benedikt, founder of micro-motor-warehouse.com, has set up a version of MultiWii 2.3 for the MicroMWC controller, which we will use as our starting point. To tell the controller which way is up, you must uncomment the line quoted here in the *config.h* file. Remove the // at the start of the line and add the minus signs as shown:

```
#define FORCE_ACC_ORIENTATION(X, Y, Z) {imu.accADC[ROLL] = -Y; imu.accADC[PITCH] = -X;  
imu.accADC[YAW] = Z;}
```



Download the Arduino Project

The complete Arduino project with all the changes described in this chapter can be downloaded from the book's companion website.

We must also change a couple of settings to accommodate the transmitter. The Blade MLP4DSM outputs a range of about 1,150 to 1,830. This range represents the width of a pulse in microseconds in some systems but is just an arbitrary scale here. This matters to us because the control system compares these RC values to preset minimum and maximum values to determine when to do things such as arming the motors. Many transmitters will output a wider range, say 1,000–2,000, or allow you to set the range, but we must make the software accommodate the hardware. Specifically, these values must be changed in the file *MultiWii.h*:

```
#define MINCHECK 1200  
  
#define MAXCHECK 1800
```

Once you have the modified code ready in the Arduino integrated development environment (IDE), you can upload it to the MicroMWC. To set up your environment for this board, select "Board: Arduino Pro or Pro Mini" and "Processor: ATmega328 (5v, 16 MHz)." The port setting will vary depending on your computer, but it will include "usbserial." [Figure 5-5](#) shows how these settings look in OS X.

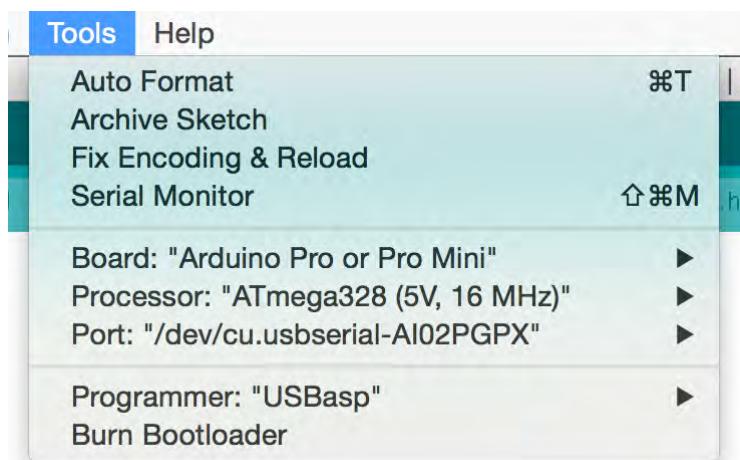


Figure 5-5 The settings for connecting the MicroMWC to the Arduino IDE

Next, connect the MicroMWC to your computer using the FTDI adapter. Only one of the supplied cables fits and it only fits one way, so this part is easy. Getting the MicroMWC into boot mode, by contrast, is tricky.

To get the MicroMWC to reset properly so the bootloader will accept new code, you have to touch the extra pin hanging from the FTDI adapter to just the right little spot on the circuit board at just the right time. [Figure 5-6](#) shows the spot that you have to hit. Hold the pin in place, then start the upload in the IDE. You can remove the pin once the download has started, as indicated by rapidly flashing lights on the FTDI adapter.

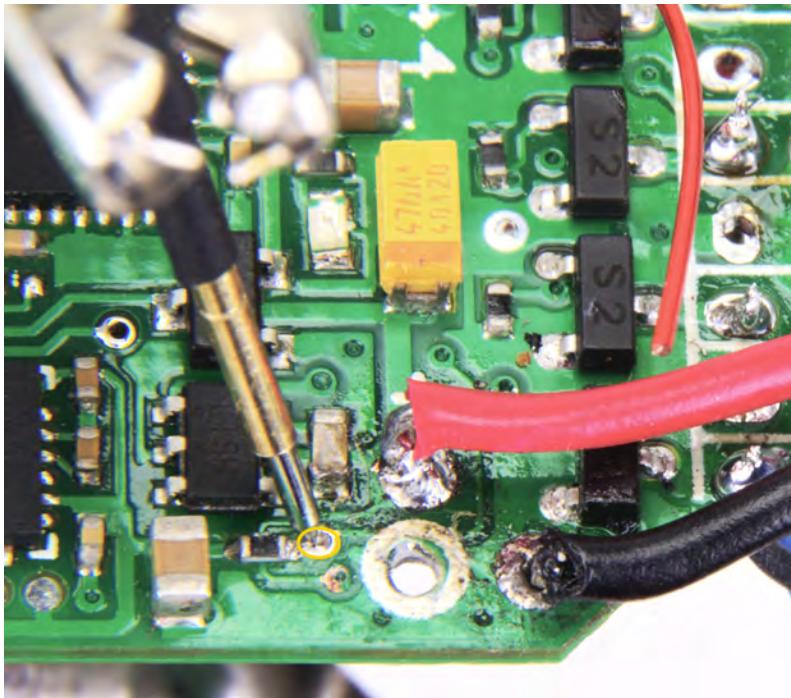


Figure 5-6 You must hold the pin against this small spot to start the flashing process

Wiring

I usually start building each new project by wiring the power. In this case all we have to do is solder on the battery connector. The connector needed is a male Micro Losi, which is the same as on a stock Hubsan X4, so the same batteries will fit. You can buy these connectors from several places, but I actually salvaged one off of a cheap battery charger. The connection on the board for the positive terminal is well marked and the negative one is near it, as seen in [Figure 5-7](#). This connector should go on the same side of the board as the red radio antenna wire.

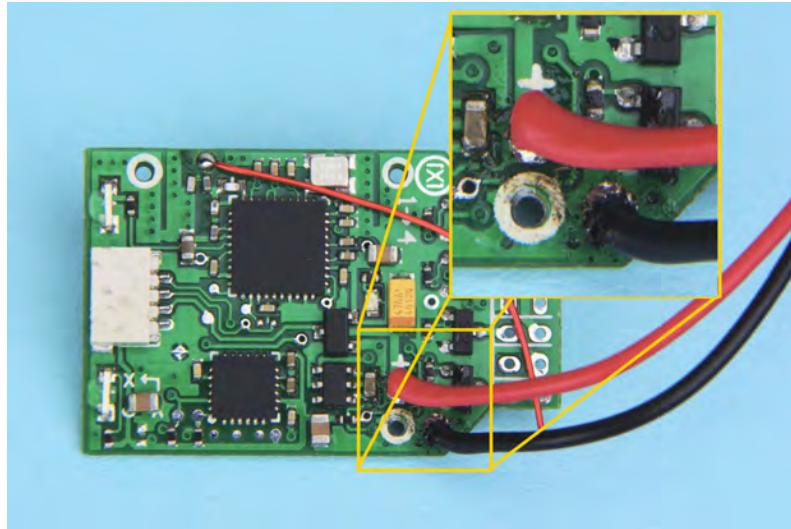


Figure 5-7 The battery connector soldered onto the controller

Our new quad will really start coming together as we put the motors in the frame and solder them to the controller. Be careful to get the motors in the right corners. The front left, as seen from the top of the X4, should be red and blue. Then they alternate going around, as can be seen in [Figure 5-8](#). The wires on the front two motors do not reach far enough, so this is where we use that little extra bit of wire. I salvaged some wire that matches from old motors. The motor wires should come out of the opposite side of the board from the battery cable. Complete the motor installation by putting some heat shrink on and soldering the front motors to their extensions as shown.

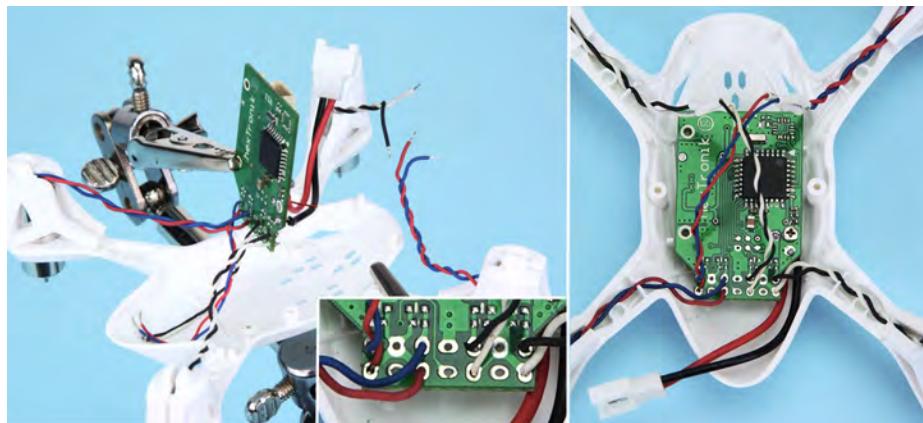


Figure 5-8 The motor wires should be soldered before the MicroMWC is screwed in

Finally, we will build a cable to connect the Bluetooth module. As mentioned, we will use the two cables that come with the FTDI adapter. Take the smaller end of each one and connect the like colors as shown in [Figure 5-9](#). Note that the colors of the supplied cables may change, in which case, of course, it is the order and not the color that really matters.

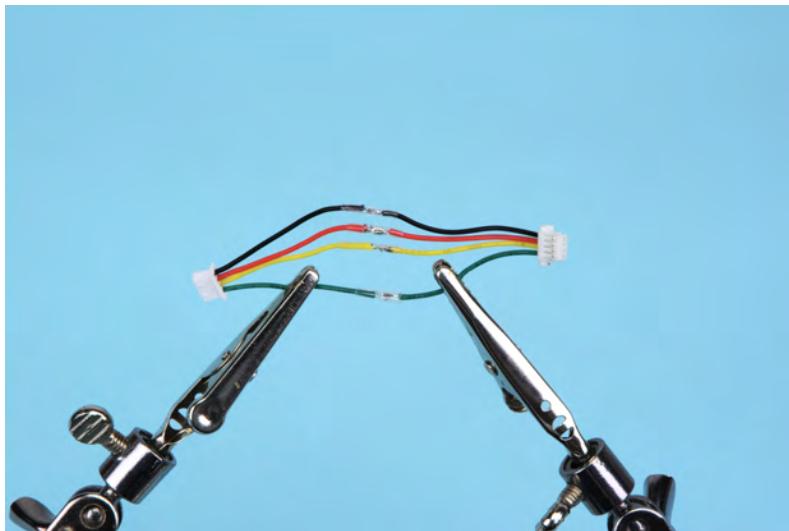


Figure 5-9 Building the cable to connect the MicroMWC to the Bluetooth module

Also note that, as with all serial connections of this sort, RX goes to TX and TX goes to RX. The result should look something like [Figure 5-10](#) when plugged in.

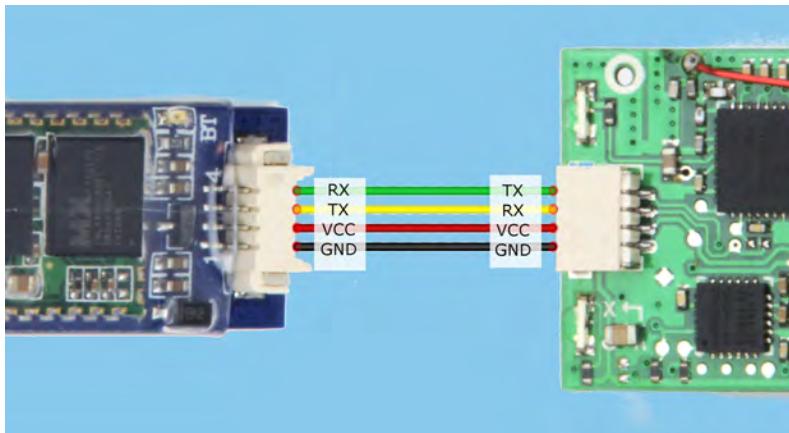


Figure 5-10 Wiring diagram for the Bluetooth cable

With all the wires in place, we can fit it all together.

Assembly

Now we can assemble everything to see how it flies! Plug the cable you just built into the MicroMWC and the Bluetooth module. Use one screw to hold the board into the top of the frame. The only hole that lines up between the Hubsan frame and the MicroMWC is at the lower right. It fits very tightly into the case, so one screw should be plenty. Next, place the middle piece of the frame on top, being careful that the motor wires don't get pinched. The Bluetooth module fits nicely into the slot designed for the video transmitter. A little wire routing, and the result should look like [Figure 5-11](#).

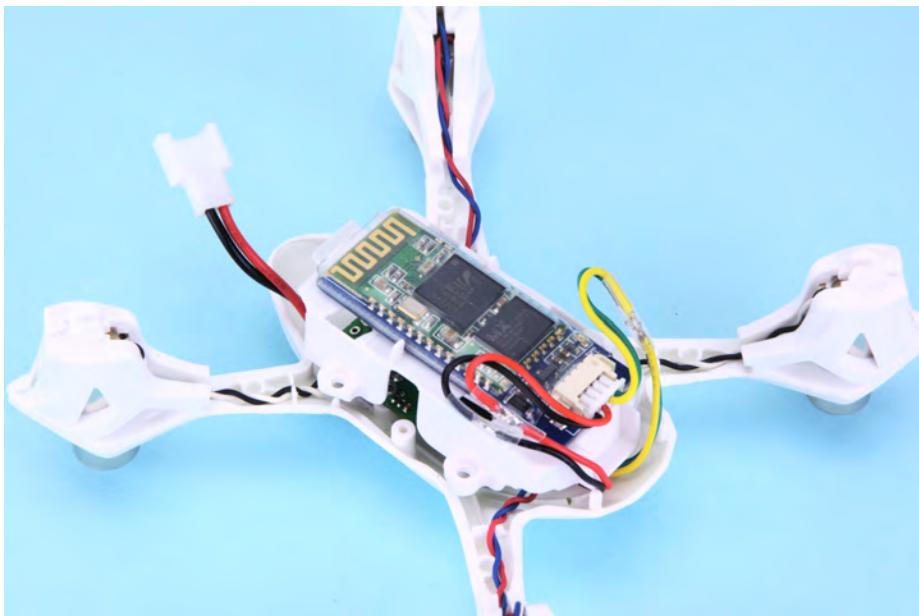


Figure 5-11 The Bluetooth module fits where the video transmitter would go

Finally, fit the bottom of the case on, again being careful not to pinch any wires. The four motor arms snap together. As you know if you have flown a stock X4 before, these motor arms come unsnapped all the time in crashes. This seems to be a way to absorb energy and keep anything from actually breaking, since they always snap right back together again. Two screws hold the middle of the case together.

To fly, we must put on some props. The MultiWii starts with clockwise at the left front and alternates as shown in [Figure 5-12](#). It helps as a visual cue to put contrasting colors in the front and the back. Press the props on as far as they will go so that crashes will push the prop against the top of the motor, rather than pressing the motor's shaft down into it.



Figure 5-12 Yellow turns clockwise, green turns counterclockwise

You are ready to fly! The default parameters in the MultiWii work reasonably well for the X4Wii. The DSM radios require a different power-on sequence than the Hubsan's, though, and since MultiWii also runs much larger, more dangerous drones, we have to intentionally arm the motors before we can fly. This checklist should get you airborne:

1. Turn on the transmitter.
2. Place the battery into the X4Wii and plug it in.
3. Place the X4Wii on solid, fairly level ground until the green blinking light stops.
4. Hold the left stick on the transmitter down and to the right until the green light comes on solid.
5. Throttle up very slightly and then back down; take note that all props turn.

(The green light mentioned shines through the case and can be very hard to see in bright sunlight. If you have trouble with the arming sequence, try it in the shade or indoors.)

The X4Wii responds a bit sluggishly to stick input compared to the stock X4, at least with the default parameters. But we will tune that up in the next section.

Setup and Tuning

Our X4Wii flies all right with the default settings, but we chose the MicroMWC and its open source MultiWii firmware so that we could improve and customize the way the X4Wii flies. We will take this in three steps. First we will increase the RC rates a bit—this is a feature most expensive transmitters have that now we can do in software! Then we will experiment with tuning the PIDs, though this is a complex subject we will tackle in greater detail in a later chapter. Finally, we will set up acrobatic mode on our X4Wii for a real flying challenge.

RC Rates

Radio control (RC) rates are like a gain knob for the control sticks. Turn up the rate, and it's like you are moving the stick further. Turn the rate down, and it makes your stick movements smaller. Advanced transmitters allow you to adjust this, but we can get this feature using our simple transmitter by adjusting parameters in MultiWii. The default roll and pitch response—that is, how fast it will rotate front to back or side to side—feels a bit slow to me, so let's increase these RC rates a little.

The “RATE” shown in [Figure 5-13](#) changes the RC rate for pitch and roll. Setting this higher makes the drone respond quicker as you move the sticks. To keep this from making the controls too twitchy and hard to manage, the “EXPO” setting makes the stick response lower around the center. The settings of 1.1 and 0.1 for “RATE” and “EXPO” are not extreme, so feel free to try higher values.

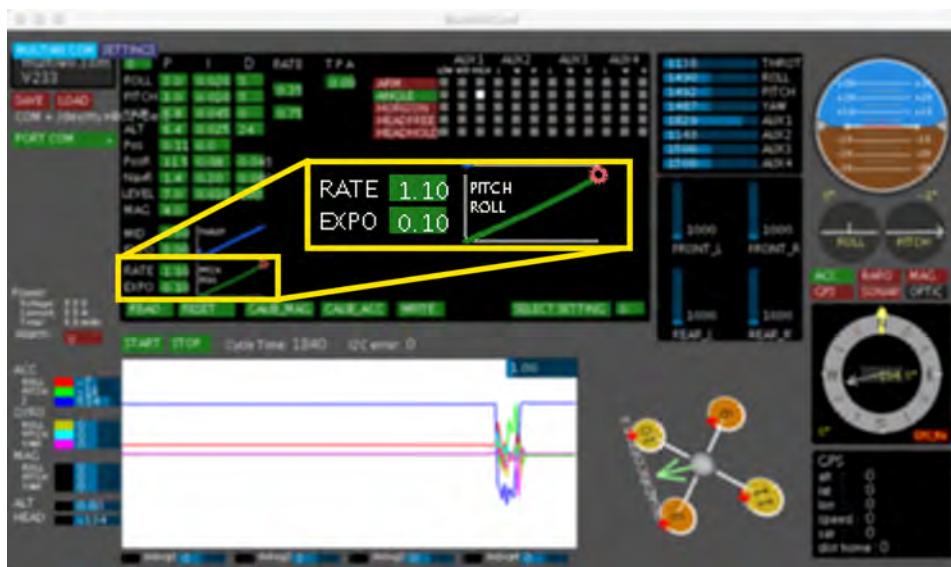


Figure 5-13 Setting RC rates with MultiWiiConf

Desktop Versus Smartphone

You can use the desktop program MultiWiiConf or various smartphone programs like MultiWii Configurator or Naze32 Configurator to adjust settings on your X4Wii (Figure 5-14).

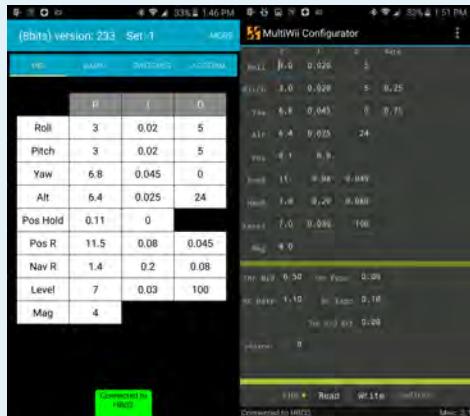


Figure 5-14 Viewing PID settings in two smartphone apps

The process for connecting different computers or phone applications varies, but in every case, when asked what to connect to, look for “HB-02.” I find the phone app is easier for changing settings, but the desktop app is easier to read since more settings fit on one screen.

PIDs

Setting RC rates can help, but this just makes the drone do the same thing with a little less stick movement. To really make our new drone fly better, we will need to tune the PIDs (a topic we’ll discuss in more detail in [Chapter 11](#)). The PIDs in MultiWii include an extra “RATE” parameter in addition to the “P” (proportional), “I” (integral), and “D” (derivative) parameters, and this new parameter happens to do just what we want: it increases the rate when using high stick deflection. Try setting the “PID Yaw Rate” and the “PID Roll/Pitch Rate” to 0.25 and 0.75, respectively (see [Figure 5-15](#)).

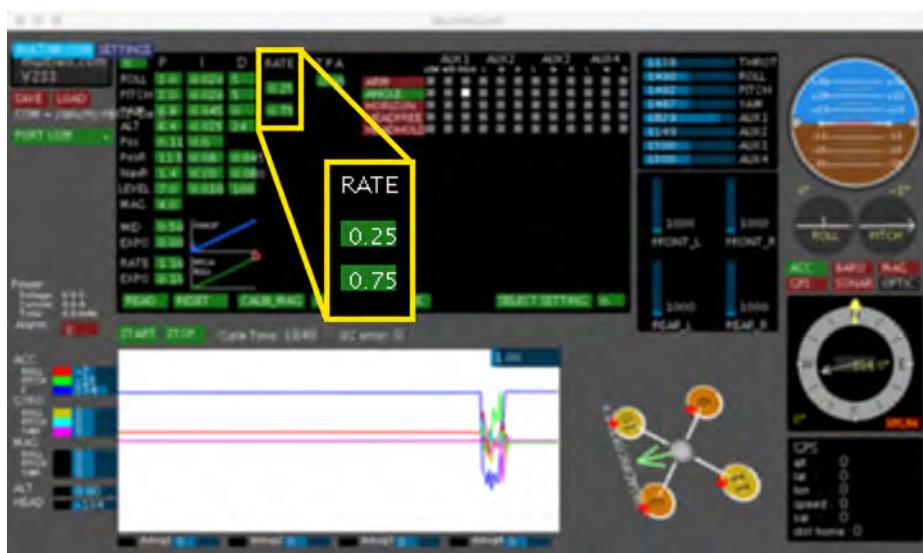


Figure 5-15 Using MultiWiiConf to adjust the extra RATE parameter

I use full yaw all the time, so this setting really helps. Roll and pitch need more adjustment in the midrange, so we will adjust the actual “P,” “I,” and “D” settings next.

The PID controllers, embodied by a few lines of code in the firmware, either hold something constant or smooth out any required control changes so that they are as fast as possible without causing problems such as overshoot or oscillation. The cruise control system in a car uses something like a PID controller, for example. In drones we use them to control roll, pitch, and yaw at a minimum. As we add other systems, like a barometer for altitude hold, PIDs get added to manage these variables too.

The “P” setting controls how strongly the controller tries to correct an error between the requested and actual values. The value in question could be roll angle, roll rate, or even latitude. Using roll angle as an example, the desired and actual angles might be different because the wind blew your drone around, changing the actual angle, or because you moved the stick, changing the desired angle. The “P” value in this case would control how much roll force gets applied to reduce the error.

If you set “P” too low, the system doesn’t have enough control and the drone will sort of wobble around. If you set “P” too high, the control system goes into oscillations. This may at first look similar to the wobbling around of a low “P.” To be sure which you are seeing, adjust through the middle, more stable zone first. Most tuning guides say to increase “P” until oscillations begin, then back off a little. But adjusting “I” and “D” affects how “P” works, so you may have to make several passes before you get the response you are after (see [Figure 5-16](#)).

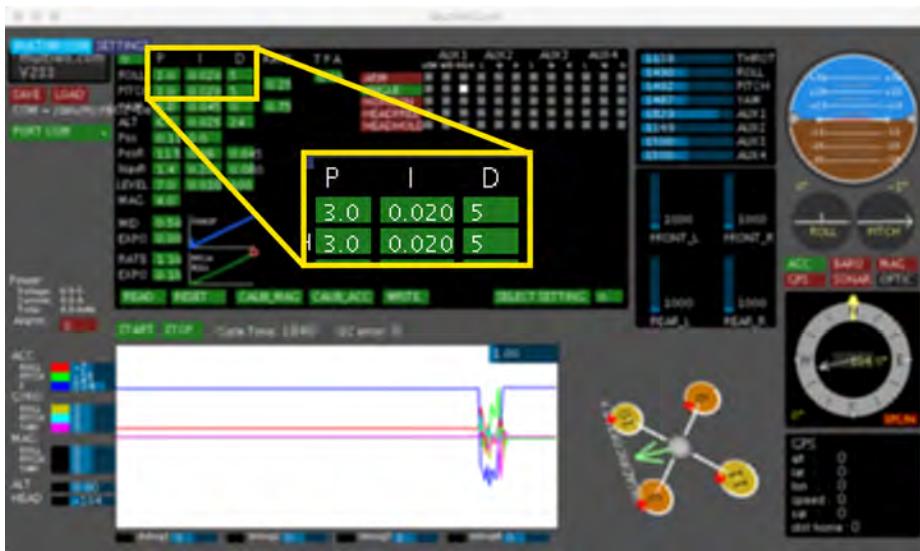


Figure 5-16 “P,” “I,” and “D” settings for a smooth, but more responsive feel

You should now have “P,” “I,” and “D” set to your liking—but this is only angle mode. Adding other modes may require more tweaking, and that’s what we’ll talk about next.

Modes

For extra credit, let’s try setting the X4Wii to what MultiWii calls “manual” mode. Most would call this “rate” mode or full acrobatic mode. It means that the control system uses the gyros to stabilize roll, pitch, and yaw, but it does not use the accelerometers to give any reference to the horizon. In the normal angle mode, the angle of the stick is proportional to the pitch or roll angle of the drone. In rate mode, the angle of the stick determines the roll rate or pitch rate. If you hold the stick all the way to the left, the drone should roll completely around. Of course, acrobatics are not that easy!

There is a trick to setting MultiWii to “manual”—it doesn’t show up in the list of modes! You get manual mode when no other mode is selected. You select modes when flying by changing AUX channels on the MLP4DSM, and you do *that* by pushing in the right stick. This changes the AUX1 value from high to low. Modes get selected when the box under the appropriate value, “L” or “H,” is highlighted (Figure 5-17). You highlight a box in MultiWiiConf by clicking on the boxes until you have the settings you want and then pressing “Save.” In general, I find the cell phone apps easier for tweaking settings, which is why we added the Bluetooth connection.

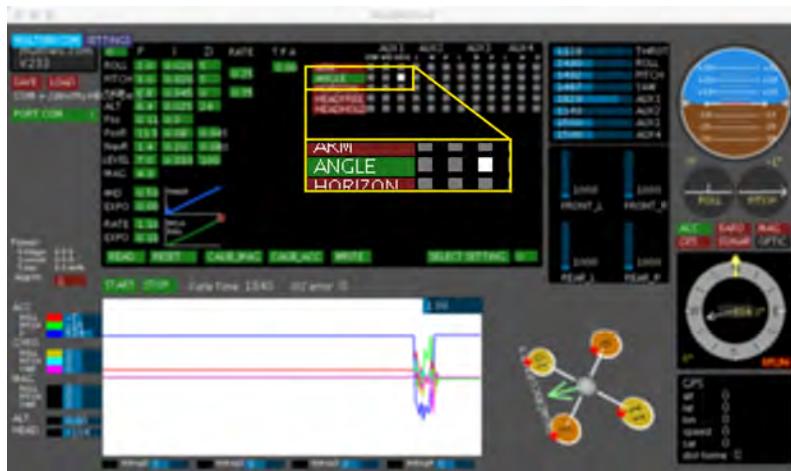


Figure 5-17 With only the "H" box selected, a low value on AUX1 will give manual mode

This completes the construction and initial tuning of the X4Wii (Figure 5-18). We started with a stock frame and motors, but this little drone can do many things that its stock cousins can't. We can change the RC rates and adjust the PIDs to fine-tune the X4Wii's flying to our individual tastes. Thanks to the Bluetooth module, we can easily connect to ground control station software on a computer or smartphone. We can also start learning how to fly in rate mode. In short, the X4Wii has many of the features of larger, hobby-grade drones in a small, safe, and cheap to maintain package.



Figure 5-18 The completed X4Wii

Now that we've seen how much difference the flight controller makes, it's time to dig in and see how one works. As makers, we learn how things work by building them, right? So, we shall build a flight controller from separate modules and then write the code that makes it fly.

The Visible Drone



By building the X4Wii, we learned what a difference a flight controller can make. This made me all the more curious to find out what's going on in there, so in fine maker style, we will build our own! Instead of a single, integrated flight controller, we will build a modular one, combining a standard Arduino-compatible CPU with an accelerometer/gyro module, radios, and a custom motor controller to create our hardware platform.

By upgrading key components from what is integrated in the MicroMWC, we will gain significant functionality—such as altitude hold and heading hold—and still have future expandability. Later, the modular nature of the system will allow us to upgrade to a better barometric pressure sensor if one becomes available, or use an even faster processor. It's like moving up from a boom box to a component stereo.

As with any modern gadget, our flight controller will need software. While we could port the MultiWii software we used with the X4Wii to our new hardware platform, that wouldn't teach us anything new about how flight control software works. So, we will write our own! While it takes many person-years of effort to create a really complete flight control system, by leveraging existing code and keeping our initial scope modest, we can quickly create a flight control system that not only works, but makes it easy to see how it works.

Given the highly experimental nature of our controller, it seems best to once again use the frame, motors, and props from a stock drone. Keeping the design as small as possible, but allowing for a heavier controller, we will move to geared motors instead of direct drive. Geared motors can lift more payload for a given amount of power because they use a larger prop. However, the larger prop and the gear train make the system less responsive. There is always a trade-off in aviation.

In the spirit of making it clear how this drone works, we're not going to hide our custom controller away inside the body. A window cut into the top of the case will allow us to show off our work and perhaps explain to the next generation of drone makers that this part is the radio, this part is the gyro, and so forth. So, with credit to the clear plastic model of a human with removable organs, we will call this design *the Visible Drone*.

A Teensy CPU

The processor in the Visible Drone will be upgraded significantly from the one in the X4Wii. It's not that the 8-bit CPU on the MicroMWC was having trouble keeping up. It can consistently run loop times around 2 milliseconds, which means 500 updates per second. Still, the reasons for wanting a more powerful CPU are twofold. Having power to spare will allow for future expansion, but more importantly, we want to move up to a processor with a hardware floating-point unit.

To understand what a floating-point number is, see the sidebar on the subject. While an 8-bit Arduino can do floating-point arithmetic, it must do so in software, and this gets too slow on older CPUs. Hardware floating point *is* fast enough to use in a flight control loop, which means that we can avoid difficult code optimizations that also make the code far less readable. It will allow us, in fact, to take an academic code example and run it as is. Floating point makes the code easier to write and easier to read.

Floating-Point Numbers

Computers use floating-point numbers like we use scientific notation, only in binary instead of base 10. Both consist of a main part, or mantissa, and an exponent. The exponent says how far left or right to move the decimal point, thus the name "floating point." For example, with a mantissa of 1.000 and exponent of 5, we get 1×2^5 , or 32.

With an exponent of -5, we get 1×2^{-5} , which is 1/32 or 0.03125. This allows the representation of very large and very small numbers without loss of precision.

In another move to keep the coding easy, we will choose a processor that can be programmed using the Arduino integrated development environment. Currently, the Teensy 3.2 from PJRC ([Figure 6-1](#)) looks like the best combination of small size, the latest 32-bit CPU with hardware floating point, and plenty of fast ports. As we shall see when we start programming it, it takes a couple of tweaks to the Arduino IDE to support this board, but with those done it feels very familiar.

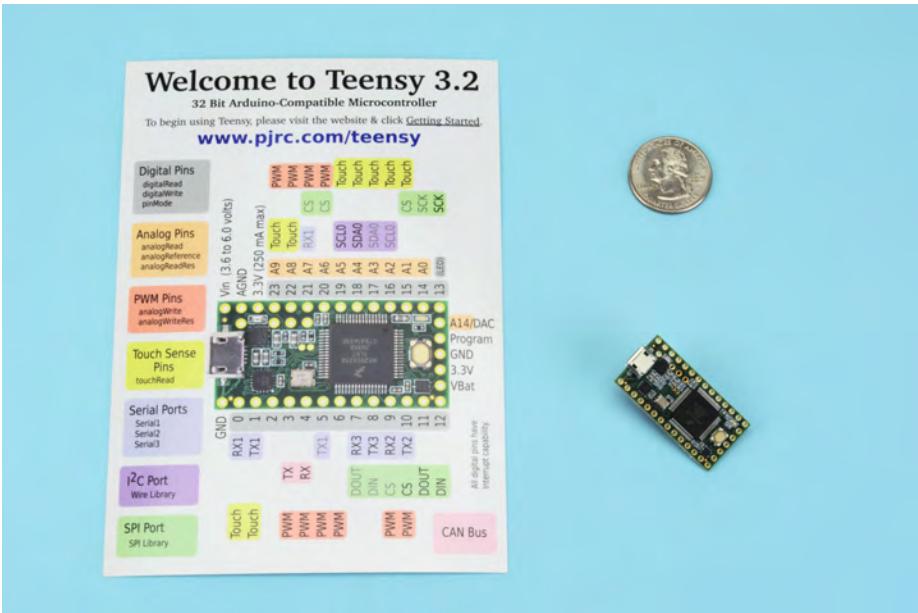


Figure 6-1 The Teensy 3.2 comes with a lovely pinout card

Radios

We can use the same radio control (RC) transmitter with the Visible Drone that we used with the X4Wii since a single transmitter can be used with multiple aircraft, but we will need a new RC receiver. It *must* support the same DSM2 standard as the MLP4DSM transmitter, but that leaves many choices from several companies.

Historically, RC receivers simply talked directly to servos. When gyro stabilizers first appeared, they just plugged inline between the two, using these same servo connections. The trouble is that with lots of channels, each one having three wires (two for power and one for signal), this adds up to lots of wires. Today we have several standards available that put all the channels on a single connection.

We will talk about various RC radio standards in [Chapter 24](#), but interestingly, the one we will use wasn't originally developed as an external interface at all. Spektrum, a maker of RC radios and owner of the DSM2 standard, started making little secondary remote receivers that could be placed in multiple locations on a model for better reception. It turns out that the simple three-wire serial interface they use to talk to the main receiver is also easy to read directly with an Arduino, which is just what we will do.

Several companies make Spektrum-compatible satellite receivers. ("Satellite" refers to the fact that they are remote, not that they receive signals from orbit.) For this design, we will use the Lemon RX DSM2 satellite. This tiny board communicates nicely with our transmitter, weighs less than a gram out of its case, and costs well under \$10.

Like with the X4Wii, we will also add a Bluetooth radio to the Visible Drone for real-time telemetry and ease of parameter changes. We will use an HC-06 module instead of the HB-02 we used before, but they differ only in the type of physical connection—plug versus pins. More significantly, since we are writing this control system ourselves, we can feed it any data that we want to see and we can use the same channel to send commands back to the drone. [Figure 6-2](#) shows the HC-06 module with the Lemon RX DSM2 receiver.



Figure 6-2 The Lemon RX receiver and HC-06 Bluetooth module

IMU/AHRS

IMU stands for *inertial measurement unit* and AHRS stands for *attitude and heading reference system*. The latter term covers a broader range of systems that include noninertial sensors like magnetometers and barometers. In the academic literature on the subject—and there are decades of research out there—the latter are often referred to as MARG systems, for the magnetic, angular rate, and gravity sensors they include.

Almost all micro quads today claim six-axis stabilization. A more technical description is that they measure rotation rates around three axes and acceleration rates in three directions, making what is known as a six degrees of freedom or 6DOF system. We're going to improve on that!

Today, you can get a 10DOF sensor module that fits in the same space and talks to the same interface, though it does cost all of \$10 instead of \$3. The GY-88 module ([Figure 6-3](#)), available from several sources, includes an accelerometer/gyro chip like other small flight

controllers, but adds a magnetometer and a barometer. That's 3 new degrees of freedom to describe the direction of the Earth's magnetic field and 1 more for altitude on top of the 6 we already had, giving 10.

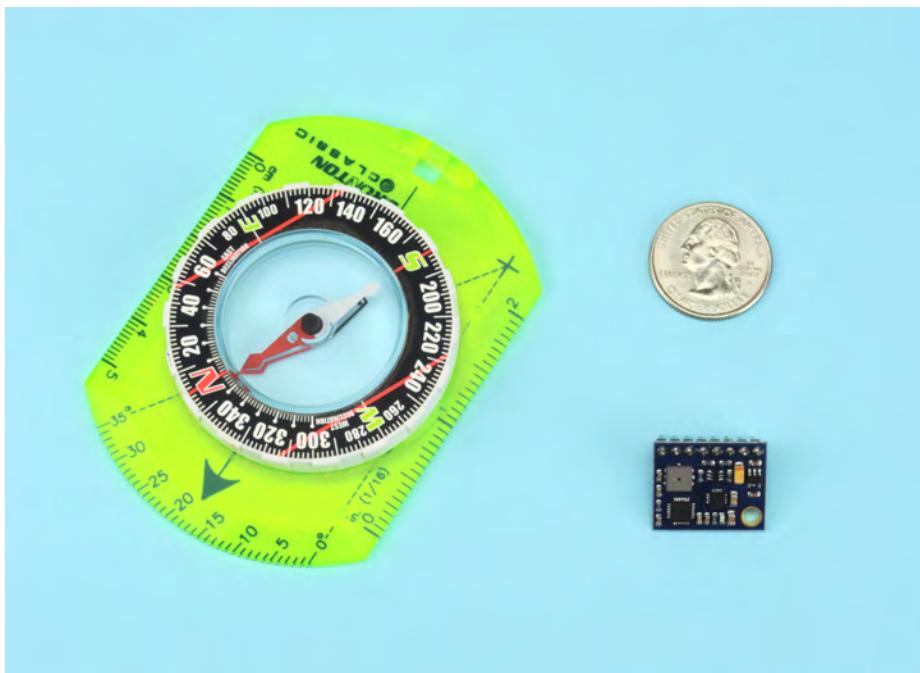


Figure 6-3 The GY-88 module includes a compass

Power

For this design we will stick with 1S LiPos, meaning single-cell lithium polymer batteries. As drones get larger, they tend to use higher-voltage LiPos of 3S or more. The higher voltage means that less current and therefore smaller, lighter wires are needed to deliver the same amount of power. For the small, coreless motors in this design, we can get enough power through small wires using only a 1S source. Making the motor wires any lighter would just make them hard to work with, so there is little to be gained by using a more complex and expensive battery.

Modern electronics like to have clean, consistent power, but the voltage coming from batteries varies with load and their state of charge. Our 1S LiPos will run from 4.2V down to 3.7V or so during a normal flight. Also, many of our modules require a 5V supply. All of this can be solved with a DC-DC voltage step-up module like the one shown in [Figure 6-4](#). Connect a 1S LiPo on one side and get up to 2 amps of clean 5V power on the other.

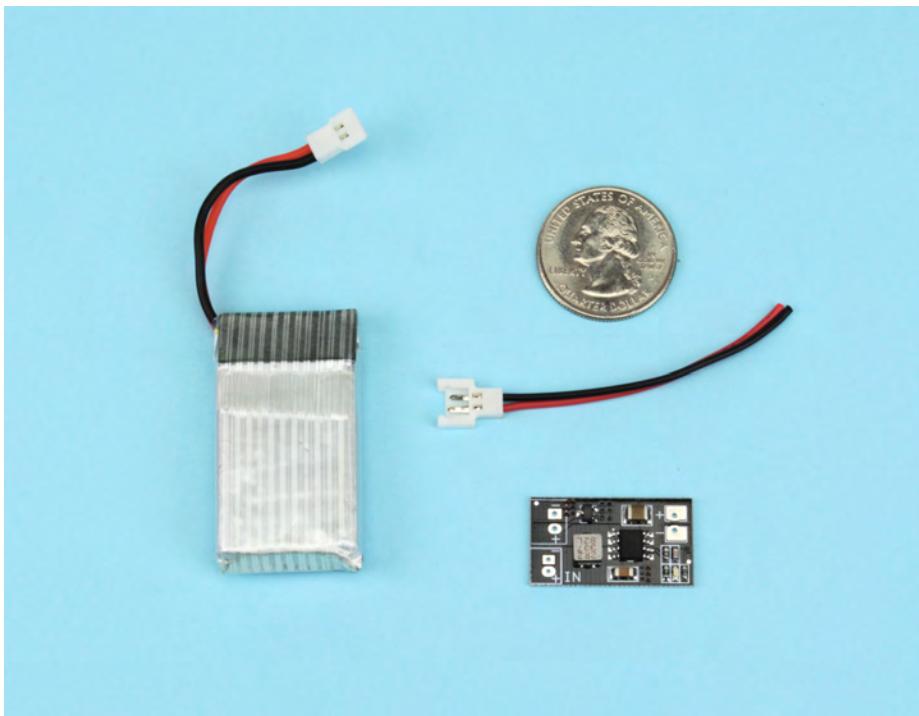


Figure 6-4 A 1S LiPo and this DC-DC convertor give clean 5V power

Circuit Board with Motor Controllers

In the small quads we have looked at so far, everything is integrated into a single circuit board, including the motor controllers. This makes sense as the circuits are simple and these small quads need to be as light as possible. In larger quads using brushless motors, a complex controller with its own processor is needed for each motor, and the current requirements vary widely, so these electronic speed controllers (ESCs) tend to be separate units. The Visible Drone design exists in an odd gap in between. We are using simpler, brushed motors, but building a modular control system.

Most of the available brushed motor controllers are for land vehicles and thus are too big and heavy. Also, they have additional circuitry for going forward and in reverse, which we do not need. Well, we were going to need some kind of circuit board to connect everything together anyway, and it only takes a couple of parts per motor, so we're going to build our own motor controllers!

The scale of our controller project makes it well suited for testing on a breadboard, but we will need something a little more robust, not to mention smaller and lighter, for flying. Thankfully, one can get custom circuit boards fabricated quickly and cheaply now and the tools to design them are free and easy to use, so we will use a custom PC board for our motor controllers and the connections between all the other modules.

Frame/Motors/Props

Everything we have talked about so far could be the brains of a car, a robot, or anything that uses motors to move and needs to know its position. Of course, that's part of the point. We are using drones as a vehicle to learn general control systems engineering concepts. All of these components, from the RC radios to the motor controllers, can be used in projects far removed from building drones. But this time we are building a drone, so let's choose the frame, motors, and props.

In between the little direct-drive micro drones with brushed motors like the X4Wii and the large, hobby-grade multirotors that use brushless motors (also direct-drive), there exists a world of mostly toy drones with brushed motors running larger props through gears. This allows them to run slower and more efficiently, which in turn means they can carry more payload—so, let's build a geared drone this time.

I like to stay as small as possible for most drone projects because I can fly the resulting product in a small space and the hazards of crashing it into things are mostly to my drone and not to the things I hit (like the neighbor's car). While you can get geared drones as small as the Hubsan X4s, we will need a little more space for all the components in our control system. The Syma X5 ([Figure 6-5](#)) has a nice roomy case, and spare parts are widely available. I got the bind and fly (BNF) version without a transmitter. There are also crash kits, but again they don't come with screws, so the BNF version worked out better this time.



Figure 6-5 The Visible Drone project starts with a stock Syma X5

As mentioned, I think our new modular controller will be really cool looking, so I'm going to chop a hole in the top of that case and put in a clear window—but we will cover that in [Chapter 13](#).

Weight Budget

We will need a few other parts like pins, plugs, and wires, and we won't really know what the circuit board weighs until much later the project, when the design is complete and back from the fab. Nonetheless, it is our duty as aircraft designers to do a weight budget and check the results against our available thrust. [Table 6-1](#) shows our best available estimates.

Table 6-1 Weight budget for the Visible Drone

Item	Grams
Teensy 3.2	2.7
Lemon RX DSM2 satellite receiver	1
HC-06 Bluetooth module	3.5
GY-88 10DOF sensor module	2
19 0.1" pins & 4 motor plugs	1
Circuit board	5
Matek DC-DC 1S to 5V step-up convertor	1.8
Frame/motors/props from Syma X5C	64
680 mAh 1S LiPo	16.2
Total	97.2

For comparison, a stock Syma X5C flies at an all-up weight (AUW) of 85.6 grams, making our modular controller design almost 12 grams heavier. This compares to a measured thrust of 41.7g per motor, for a total of 166.8 grams of thrust. That's a thrust to weight ratio of only 1.7, a little low. We like to see a ratio of 2 or higher. But that AOW number for the stock X5 is stripped. Most of them fly with a camera, bigger landing gear, and prop guards, easily making them as heavy as ours. The Visible Drone, like the stock X5, will be a gentle flyer.

Basic I/O Code

7

In this chapter we will create a basic framework for our new flight controller code and learn to communicate with some of our key peripherals, including the radios and motors. We will use the framework we build here as the platform for the next few chapters, where we will add the IMU and mode functions.

The figures in this section show the circuit being built up on a breadboard. If you plan to use exactly the same circuit as shown in the following chapters in your own project, then you can skip directly to building all of this on the printed circuit board that we will lay out later.

Time to write some code!

Setting Up the IDE

As mentioned in the previous chapter, the Teensy 3.2 processor can be programmed using the Arduino IDE. This can be downloaded from www.arduino.org/software#ide, where there are also instructions on how to install it. Using this IDE with the Teensy 3.2 requires that you install *Teensyduino*, a package from PJRC (the designers of the Teensy), that includes the *Teensyloader* program and some Teensy-specific libraries. This package is available at www.pjrc.com/teensy/teensyduino.html. First install the Arduino IDE, then this package. With the included Teensyloader program running, you can upload code to a Teensy just like any Arduino. Open the Arduino IDE, connect the Teensy by USB, and set up the IDE as shown in [Figure 7-1](#).

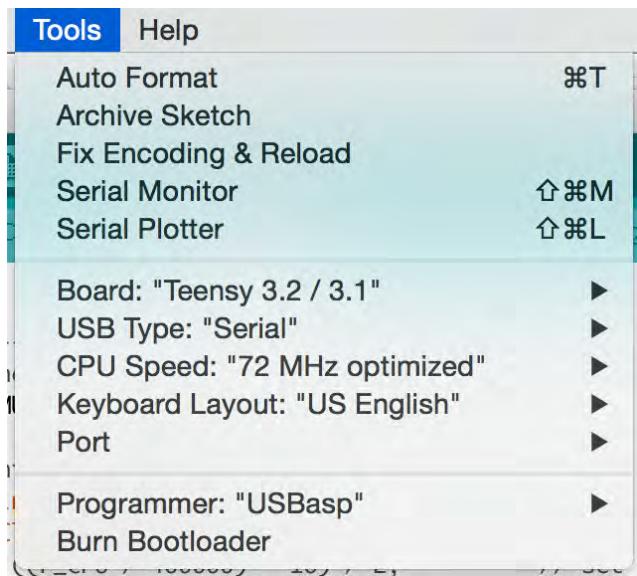


Figure 7-1 Arduino IDE settings for the Teensy 3.2

To verify that everything is working, it is traditional to run the [Blink](#) example program. Open this sketch with File → Examples → Basics → Blink on the Arduino IDE menu, then press the upload button. After the program is compiled, you should be able to see the download progress on the little Teensyloader screen (Figure 7-2).

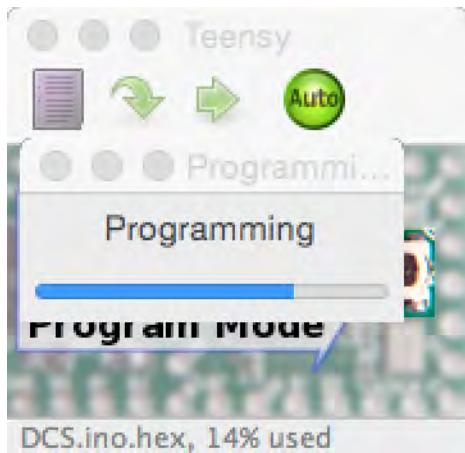


Figure 7-2 Teensyloader uploads code to the Teensy 3.2

If this sketch uploads and runs correctly, you will see an amber LED on the Teensy blink, one second on, one second off. Working? Great!

Main Loop

Most flight control systems will have at least two main loops, called *inner* and *outer* or *fast* and *slow*. There may also be background tasks and interrupt-driven tasks and more complex task scheduling systems, but we're trying to keep it simple, so we will stick to a fast and a slow loop for now. We will be indirectly using interrupt processing, but we will do it through established libraries.



The code for this project can be downloaded from GitHub at github.com/dmcgriffy/DroneControlSystem or directly from www.makedrones-book.com/projects/VizDrone.

The fast loop runs the essential flight stuff. This includes reading the accelerometers, gyros, and control inputs; computing our position and any needed corrections; and finally updating the motor speeds. All of this can be done in two or three milliseconds even on a slow 8-bit processor. We will set our inner loop target time to two milliseconds. The inner loop code should be able to execute quite a bit faster than this, but we will run it at exactly two-millisecond intervals for consistency. Finally, after the flight control is done, we read any pending telemetry messages. Reading on the fast loop helps keep the input queue from overflowing, but we only write on the slow loop. Here is what the inner loop looks like:

```
#define FASTLOOPTARGET (2000) // length of inner control loop in microseconds

unsigned long fastLoopLength;
unsigned long fastLoopStart = 0;

void loop() {

    // the fast loop for flight control
    unsigned long fastLoopEnd = micros();
    fastLoopLength = fastLoopEnd - fastLoopStart;
    if ((fastLoopLength) > FASTLOOPTARGET)
    {
        fastLoopStart = fastLoopEnd;
        readIMU();           // read raw gyro/accel/mag
        calcIMU();          // calculate current yaw/pitch/roll
        doMode();            // calculate yaw/pitch/roll commands
        doPID();             // smooth yaw/pitch/roll commands
        doMix();              // convert cmds to motor speeds
        writeMotors();        // write to pwm
        mspRead();            // check for new telemetry msgs
        fastLoopTiming = micros() - fastLoopEnd;
    }
}
```

The first line defines the loop length. Two thousand microseconds is our two-millisecond target. Ignoring the declarations, at the start of each pass through the main program loop we check to see if enough time has passed. If so, we reset the clock and run through the list

of fast loop calls. Finally, the time it took to process the fast loop is recorded. We hope that this stays significantly lower than our target loop time.

The slow loop has much the same structure. The tasks in our slow loop will all be communications for now, in particular the sending of all our telemetry data. We can trade off the speed of this loop against the amount of data to be sent in each pass. The baud rate of the Bluetooth connection (115,200) limits the total amount of data that can be sent. With an update rate of 10 per second we can still send plenty of data in each pass. Here is the slow loop code:

```
#define SLOWLOOPTARGET (100) // length of outer control loop in milliseconds

unsigned long slowLoopLength;
unsigned long slowLoopStart = 0;

void loop() {

    ...

    // the slow loop for communications
    unsigned long slowLoopEnd = millis();
    slowLoopLength = slowLoopEnd - slowLoopStart;
    if ((slowLoopLength) > SLOWLOOPTARGET)
    {
        slowLoopStart = slowLoopEnd;
        long slowLoopTimingStart = micros();
        mspWrite();
        slowLoopTiming = micros() - slowLoopTimingStart;
    }
}
```

Right now, we only have one task in the slow loop: the sending of telemetry messages. Doing the sending here guarantees that we will send messages at most 10 times per second.

RC Receiver

Now let's put the radio into radio control by connecting the Lemon RX receiver. First, we must bind the receiver to our transmitter. In order to put the receiver in bind mode, a sequence of pulses must be applied to the power pins. Since the receiver needs only 3.3V and doesn't draw much power, it is possible to actually power the unit off of a data pin on a 5V Arduino. This would make it easy to pulse the power to the receiver, except that we're not using a 5V Arduino. The Teensy uses 5V power, but only 3.3V so-called logic levels.

After considering more and more complex circuits to allow the Teensy to control the power to the Lemon RX receiver, I finally realized that adding all this circuitry, which we will never need while flying, was the wrong approach. Instead, I recommend that you just get any old 5V logic level Arduino and perform this simple process. I used an old Arduino Uno that I had lying around, and they are available retail from places like Radio Shack and Frys.

Temporarily solder some pins on the end of the receiver cable so that they can be plugged directly into the headers on the Arduino. Now connect the red (or leftmost, looking down at the receiver), wire to pin D5 on the Arduino, the black (or middle) wire to pin D4, and the yellow (or rightmost) wire to pin D0. You can see how this looks in [Figure 7-3](#).

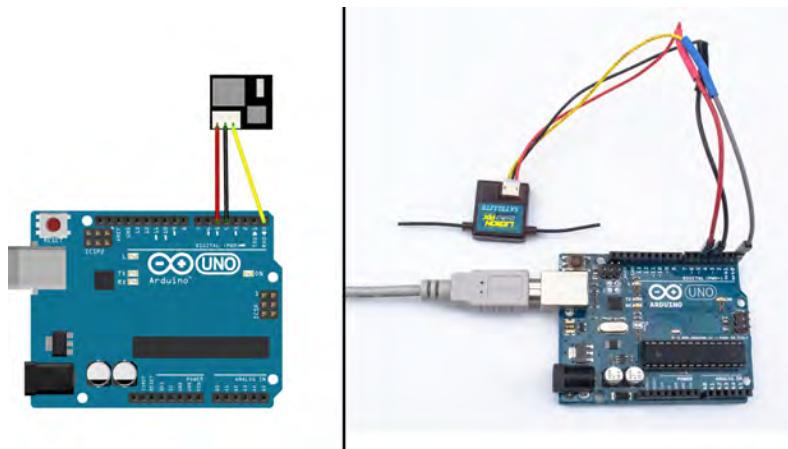


Figure 7-3 Fritzing and actual images of using an Arduino Uno to bind the Lemon RX receiver

You can download the code for the binding process at <http://www.makedronesbook.com/projects/VizDrone>. Remember to change the port settings in the Arduino IDE to match the board you are using, and then set them back for programming the Teensy later. Download and run [spekbind](#), and the receiver should momentarily start a fast blink. Once the receiver is ready, hold down the left stick on the transmitter and turn it on. The transmitter should beep and the receiver blink pattern should change.

With the receiver bound to the transmitter, let's wire it up to the Teensy 3.2 and update our code to read from it. This receiver takes 3.3V, as mentioned, but this is no problem now since the Teensy 3.2 outputs up to 250 mAh of 3.3V power, regulated down from its 5V input. This means we can just wire the receiver straight to the Teensy as shown in [Figure 7-4](#).

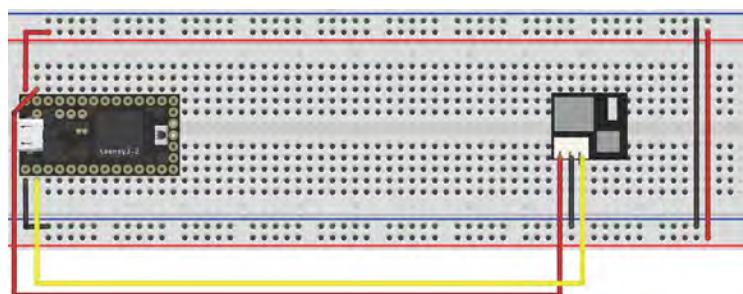


Figure 7-4 The Lemon RX receiver gets its power from the Teensy 3.2 CPU

We only receive data on this connection, since there are no settings to send back to the receiver. The data stream consists of frames at 11-millisecond intervals, each containing eight 16-bit values, a header, and seven channel values. For radios with more than seven channels, two frames are needed to transmit all the values. Strangely, the channels seem to be in random order, so we must pay attention to the channel value in bits 11–14. A picture will make it clearer, so look at [Figure 7-5](#).

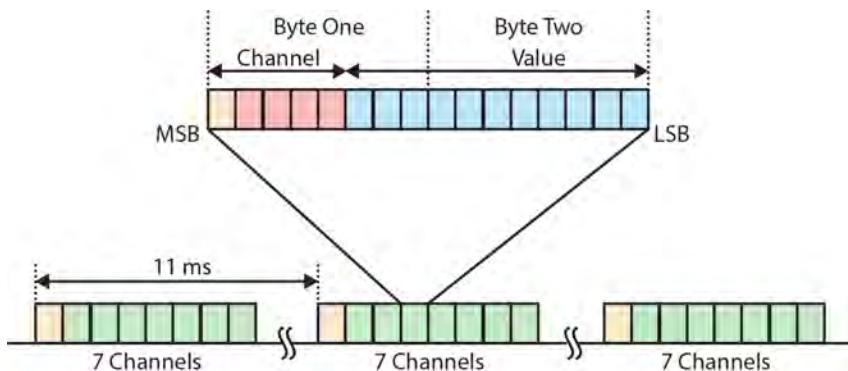


Figure 7-5 The DSM radio format includes seven channels in each frame

We will use the standard Arduino [Serial](#) library to read from the receiver. We will, however, have to use a serial event interrupt instead of just letting the library buffer for us since we need to know the timing of the arriving bytes in order to determine when a frame starts. The interrupt handler buffers the new bytes and parses the frame if it is complete. The code looks like this:

```
// RX interrupt
void serialEvent1() {
    checkForNewFrame();

    // put data in buffer
    while ((Serial1.available() && (rxBufPos < RX_BUFFER_SIZE))
        rxBuf[rxBufPos++] = (char)Serial1.read();

    // parse frame if done
    if (rxBufPos == SPEK_FRAME_SIZE)
        parseRXData();
}
```

Parsing the raw receiver data into real channel values inevitably involves some of that messy integer math we are trying to avoid. But we have a 4-bit channel number and a 10-bit value packed into 2 bytes, so the code will end up looking something like the following no matter how you do it:

```
// parse raw serial data into channels
void parseRXData() {
    // convert to channel data in the 1000-2000 range
    #define SPEK_FRAME_SIZE 16          // 1024 mode
```

```

#define SPEK_CHAN_SHIFT 2      // Assumes 10 bit frames, that is 1024 mode.
#define SPEK_CHAN_MASK 0x03    // Assumes 10 bit frames, that is 1024 mode.
for (int b = 2; b < SPEK_FRAME_SIZE; b += 2)
{
    uint8_t bh = rxBuf[b];
    uint8_t bl = rxBuf[b+1];
    uint8_t spekChannel = 0x0F & (bh >> SPEK_CHAN_SHIFT);
    if (spekChannel < RC_CHANS)
        rcValue[spekChannel] = 988 + (((uint16_t)(bh &
SPEK_CHAN_MASK) << 8) + bl);
}

// constrain outputs and implement deadband
#define DEADBAND (10)
#define DEADTOP (MIDRC + DEADBAND)
#define DEADBOT (MIDRC - DEADBAND)
for (int i=0; i<4; i++)
{
    rcValue[i] = constrain(rcValue[i], MINRC, MAXRC);
    if ((rcValue[i] > DEADTOP) && (rcValue[i] < DEADTOP))
        rcValue[i] = MIDRC;
}
}

```

Note that this routine also implements a deadband on the input values. This means that when you hold the sticks near enough to the middle of the ranges, the values will be converted to exactly the midpoint. This helps eliminate small drift in roll, pitch, and yaw.

Bluetooth

The HC-06 Bluetooth module requires four wires for transmit, receive, and power. It uses the same 5V power as the Teensy. [Figure 7-6](#) shows both the Bluetooth and RC radios wired up on a breadboard.

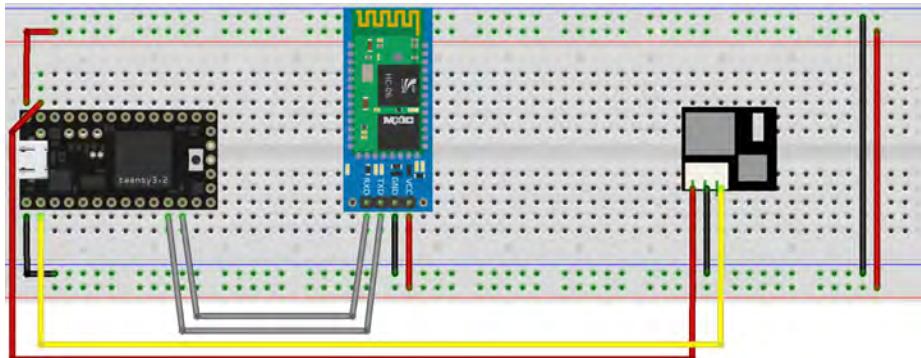


Figure 7-6 The Bluetooth and RC radios both use the 2.4 GHz radio band

We will use the MultiWii serial protocol (MSP) for telemetry, which means that we will be able to use existing ground control station software like MultiWiiConf. This protocol is designed to be simple and efficient. The ground station requests updates and sends commands. The drone sends the requested messages and updates its settings as requested. To request an update, the ground station just sends the update message with a zero data size. Each message starts with "\$M" and either "<" or ">", depending on whether the message is to or from the drone. For example, to request the current roll, pitch, and yaw, collectively called *attitude*, the ground station would send:

```
"$M<, 108, 0, <crc>
```

The drone might then respond with:

```
"$M>, 108, 6, <roll>, <pitch>, <yaw>, <crc>
```

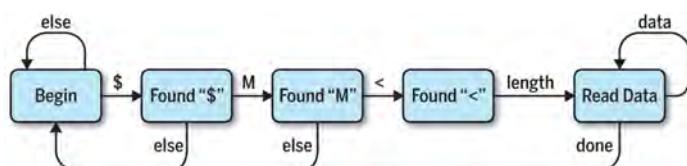
The number 108 stands for the MSP_ATTITUDE command. The <crc> at the end of each message stands for *cyclic redundancy check*, which in this case means an exclusive-or of each byte in the message. It is an 8-bit value. The <roll>, <pitch>, and <yaw> values are all 16-bit.

To send a message, we have routines that send the preamble, write data to the buffer, then calculate the CRC and send it. See the online source code for details, but as an example, here is the code to send the message we have been talking about:

```
void writeMSP_ATTITUDE() {
    mspWriteStart(MSP_ATTITUDE);
    mspWriteWord(IMURoll*10);
    mspWriteWord(-IMUPitch*10);
    mspWriteWord(-IMUHead);
    mspWriteEnd();
}
```

Note that *10 on roll and pitch. It's that pesky integer math again. You can multiply by 10 and still not overflow the range of a 16-bit integer, so this allows one decimal place of precision. This apparently isn't needed for yaw.

To parse the incoming MSP message, we will use a state machine. This is just a fancy way of saying that we will wait for a "\$", then wait for an "M", and so on. In the case of an error—for example, if we don't get an "M" after a "\$"—we go back to the start. We call the condition *Found "M"* a *state*. Once we have read the "\$M<" preamble, we read the length byte, then loop in the same state while reading the data bytes. [Figure 7-7](#) shows this in a state diagram.



[Figure 7-7](#) This state diagram shows how to parse an MSP message

Having a diagram like this to work from makes it easier to write the code. The resulting code looks like this:

```

enum {MSPSTATE_BEGIN, MSPSTATE_DOLLAR, MSPSTATE_M, MSPSTATE_LT, MSPSTATE_DATA}
mspState = MSPSTATE_BEGIN;
int mspBytesRemain = 0;
void mspRead() {
    // read available messages
    while (Serial.available())
    {
        char ch = 0x7f & Serial.read();
        switch (mspState) {
            case (MSPSTATE_BEGIN) : mspState = (ch == '$') ? MSPSTATE_DOLLAR : MSPSTATE_BEGIN; break;
            case (MSPSTATE_DOLLAR) : mspState = (ch == 'M') ? MSPSTATE_M : MSPSTATE_BEGIN; break;
            case (MSPSTATE_M) : mspState = (ch == '<') ? MSPSTATE_LT : MSPSTATE_BEGIN; break;
            case (MSPSTATE_LT) :
                mspState = MSPSTATE_DATA;
                mspBytesRemain = ch + 2; // data+cmd+checksum
                if (mspBytesRemain == 0)
                {
                    evaluateMSPCmd();
                    mspBufPosIn = 0;
                    mspState = MSPSTATE_BEGIN;
                }
                break;
            case (MSPSTATE_DATA) :
                mspInBuf[mspBufPosIn++] = ch;
                if (--mspBytesRemain == 0)
                {
                    evaluateMSPCmd();
                    mspBufPosIn = 0;
                    mspState = MSPSTATE_BEGIN;
                }
                break;
        }
    }
}

```

Near the end of this routine, the real work gets done in the call to `evaluateMSPCmd()`. When we get a request for an update, which is most of what MultiWiiConf sends, we just set a flag noting the request. The update actually gets sent later in the slow loop to make sure we don't send more than 10 per second and take time away from flight control. Actual commands like changes to PID settings or even RC stick positions get handled as they are received.

Like with the RC receiver code, we will use the standard Arduino `Serial` library to implement our telemetry. In this case we can let the library do the queuing since we don't need to know the exact time when a byte comes in. We just need to read the queue often enough to keep it from overflowing—hence the read in the fast loop. The only complexity is that

we actually need two input queues, since we can get messages from either USB or Bluetooth. You can see the details in the file [MSP.cpp](#) in the source code.

Motors

After the complexity of implementing telemetry, controlling the motors is easy. We drive the motors with pulse width modulation (PWM) from a digital output on the Teensy. The actual motor driver circuit we will cover in a later chapter. Right now, let's look at the software required.

Pulse width modulation conveys information by the length of each pulse. Our motor responds mostly to the duty cycle, or ratio of on time to off time, but it turns out the rate or frequency at which those pulses occur still represents an interesting compromise. Too low and we would see the motor pulse, but we don't need to go that low. At moderate frequencies, in the few hundred to few thousand cycles per second range, the pulses become audible as a high-pitched whine. But we can't just keep going higher.

Our motor driver circuit, like most, uses a field effect transistor (FET) to turn on and off the large flow of current based on the input from the Arduino. It turns out that in our application, the FET uses power mostly when switching, so the faster we switch, the more power it must dissipate as heat. Motor PWM for small, brushed motors like the ones we are using here often runs at 8 kHz. While high pitched, most people can hear a sound in this range. We will start there, but consider going higher if the sound turns out to be a problem.

The code for this is simpler than the description. We use the standard Arduino analog `Write()` calls:

```
uint16_t motor[4];
uint8_t PWM_PIN[4] = {20,21, 22, 23};

void writeMotors() {
    for (int i=0; i<4; i++)
        analogWrite(PWM_PIN[i], (motor[i]-1000)/4);
}

void initMotors() {
    for (int i=0; i<4; i++) {
        motor[i] = 1000;
        pinMode(PWM_PIN[i],OUTPUT);
        analogWriteFrequency(PWM_PIN[i], 8000);
    }
    writeMotors();
}
```

The values in the variable `motor[4]` range from 1,000 to 2,000. This represents the range of standard servo PWM in microseconds, but they are just arbitrary numbers for us. The 1,000 gets subtracted off and divided by 4 to give a number in the range of 0 to 256 for the actual PWM register.

If we were controlling a naturally stable, fixed-wing aircraft, we could just connect the RC radio to the motors or servos and be done. In fact, we wouldn't need a processor at all, as evidenced by all the RC planes that flew for decades with no stabilization or automatic control.

However, our modern quadcopter designs do need stabilization, and in order to stabilize a quadcopter we must understand what makes it fly. Thus, our next stop will be a brief summary of multirotor aerodynamics.

Multirotor Aerodynamics



Quadcopters basically have no aerodynamics, so this will be a pretty short chapter, right? Well, it's true that some of the usual factors, like drag, are insignificant for drones, but drones do have a few unique properties. And the laws of physics apply to all, so at least some of the basics must be the same, starting with this: it takes lift to make something go up.

Lift and Thrust

The most basic equation in flight says that lift, the upward force on the aircraft, must equal weight, the downward force due to gravity. Certainly this still applies to multicopters, but where fixed-wing aircraft use thrust to create forward motion and then use wings to convert this forward motion into lift, we skip all that. We just point the props up, so lift equals total thrust, as shown in [Figure 8-1](#).

Of course, this only applies when level. If the drone gets tilted, either by the wind or because of control inputs, then only a portion of the total thrust gets converted to lift. The rest becomes horizontal thrust and accelerates the drone forward, backward, or sideways. [Figure 8-2](#) shows this relationship.

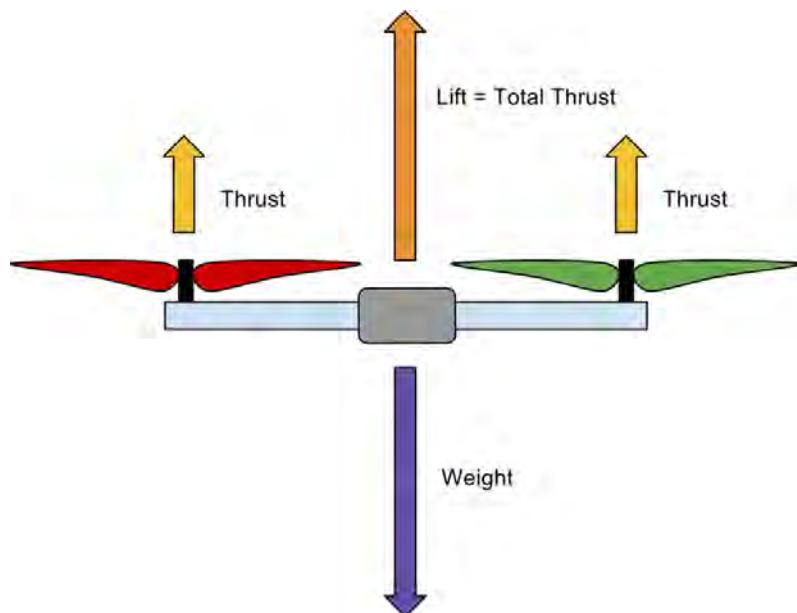


Figure 8-1 When hovering level, weight = lift = total thrust

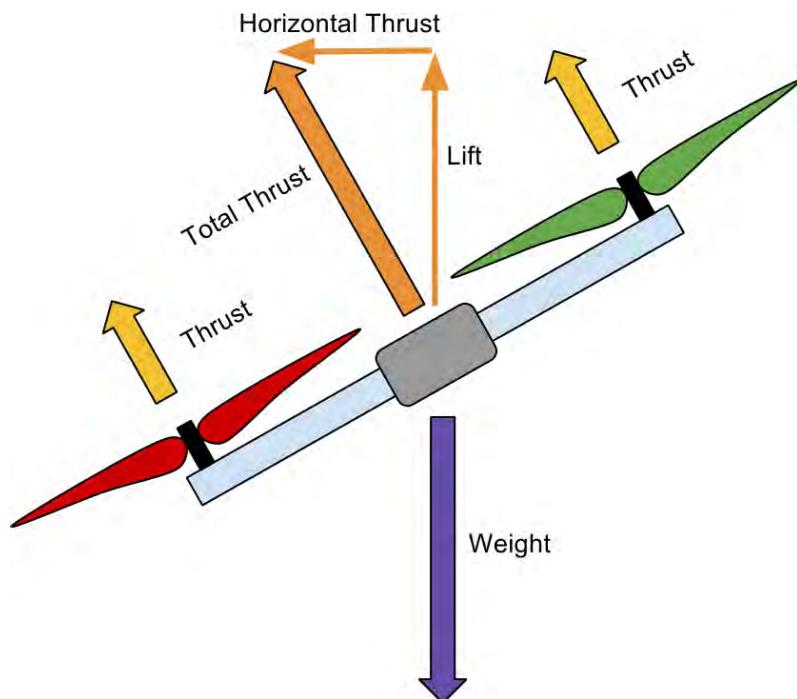
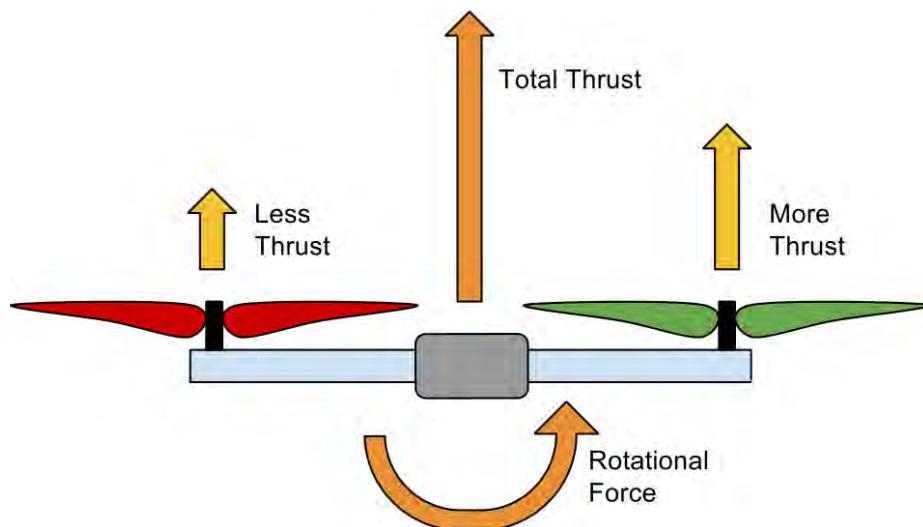


Figure 8-2 Only part of the total thrust becomes lift when the drone is tilted

Thus, if we are to maintain our hover—that is, if we want to keep lift equal to weight—we must increase the total thrust when the drone tilts over. We will accommodate this in our control system with a correction factor applied to the throttle value.

Pitch and Roll

Pitch means tilting forward and backward. *Roll* means tilting side to side. As noted, when the drone is tilted, we get some amount of horizontal thrust, and, of course, drones use this to maneuver. When we want to tilt in a given direction we just decrease the thrust on the motors in that direction and increase the thrust on the motors in the opposite direction. [Figure 8-3](#) shows how this so-called differential thrust creates a rotational force.



[Figure 8-3](#) Differential thrust creates a rotational force

If we keep applying differential thrust to our drone, it will roll faster and faster, eventually doing flips and more. This may be what the aerobatics pilot wants, but it's not what we expect when flying in angle mode. In this, our usual mode, the desired angle of the drone is proportional to the angle of the stick. When you move the stick to a new angle, the flight control system applies differential thrust just long enough to rotate the drone to the desired angle. It will then use small amount of differential thrust to hold it there until you move the stick again.

Yaw

Yaw is rotation around the vertical axis, the way a car turns. While most people understand that you can speed up the motors on one side and make the drone tilt, it is less obvious what makes it yaw. Instead of differential thrust, we use differential torque. It's one of those ideas that is clever in its very simplicity.

As each motor turns its prop and the prop pushes against the air, the motor must in turn push back against something. That something is the frame of the quadcopter. This torque would make our drones spin around if all the motors turned in the same direction. By having half of the motors spin clockwise and half counterclockwise, the torque cancels out, resulting in zero rotational force.

Then, when we want to make our drone yaw, we just make the half of our motors spinning in the opposite direction spin a little faster. To keep the total thrust the same, we slow down the other motors a corresponding amount. The difference between the push back of the motors in each direction, the differential torque, creates the desired rotational force on our drone. [Figure 8-4](#) illustrates how this works.

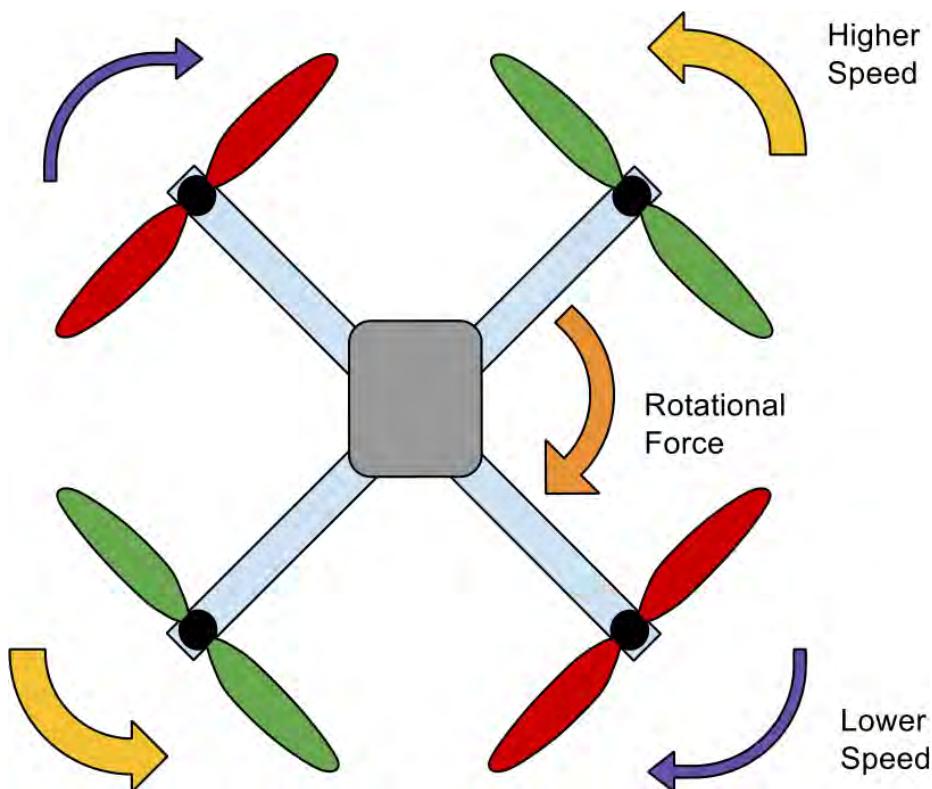


Figure 8-4 The differential torque between the high- and low-speed motors results in rotational force

While gravity gives us a nice absolute reference for pitch and roll angles, we also need to measure the Earth's magnetic field in order to get an absolute reference for yaw. We need to move from a 6DOF sensor to our 10DOF sensor. Just stabilizing yaw, with no absolute reference, can be enough for just flying around. But for true autonomous flight, we need to know which way is north.

Translational Lift

So far we have looked at lift, roll/pitch, and yaw. The aerodynamics describe how the drone uses the air to do what we command with our stick movements. The next two items, translational lift and vortex ring state, describe the sometimes surprising ways that the air reacts to the drone.

While helicopters and multirotors can hover, it turns out that it helps to keep moving. Multirotors, like the helicopters they share their strange aerodynamics with, create lift more efficiently when moving horizontally through the air. To understand why this would be so, first consider hovering in place.

As seen in the first part of [Figure 8-5](#), when in a stationary hover, the air going through the rotor is moving straight downward. Our aircraft has to work harder to maintain altitude when the air it's in is moving downward. It's like hanging on a rope that is slowly being lowered down. You have to climb just to stay in the same place.

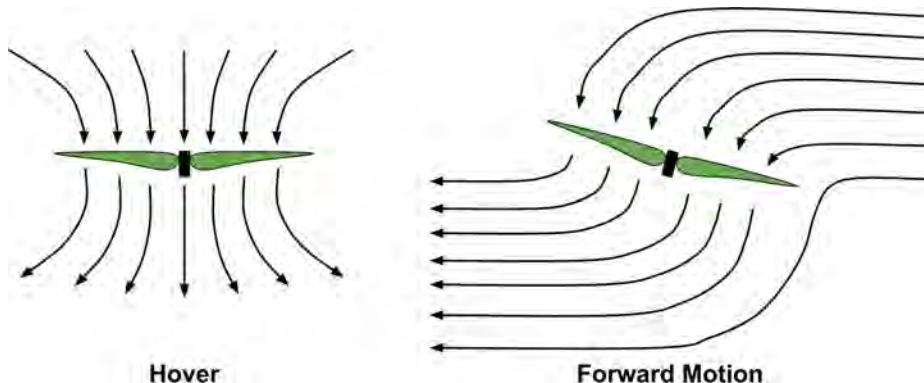


Figure 8-5 Translational lift occurs when the rotor pushes on air that is not already traveling downward

In the second part of [Figure 8-5](#), we start moving forward. The air entering the rotor comes from in front rather than above, and thus it is not already moving downward. Once you are moving fast enough, you are always in clean air, not the turbulent, downward-moving air you just pushed on a minute ago. To stretch the rope analogy, as soon as you pull on the rope and it starts to move downward, instead of climbing, you just grab the next rope over that isn't moving yet.

We call this increased lift with horizontal motion [*translational lift*](#). It occurs whenever the air moves horizontally across the rotor, whether you are moving forward or flying into the wind to maintain a fixed position. This comes up for us as drone pilots when we turn from flying with the wind to flying into it, or the other way around. This changes our effective air speed and in turn the amount of throttle required to maintain a constant altitude.

Vortex Ring State

To help understand this next effect, visualize a smoke ring. Smoke rings occur when a column of smoke gets pushed relative to the air around it. The interface between the moving and nonmoving air creates turbulence, leading to swirls or vortices. When these all line up just right, you get a smoke ring, technically called a *vortex ring*. When a rotating prop pushes a column of air, a similar thing happens and we get a vortex ring around the tip of each prop. As the air rotates around the prop tip, some of it comes around and pushes back down on the prop, canceling some of the lift from that portion of its length. Under normal circumstances, these vortex rings don't affect our performance too much, as shown in the first half of [Figure 8-6](#).

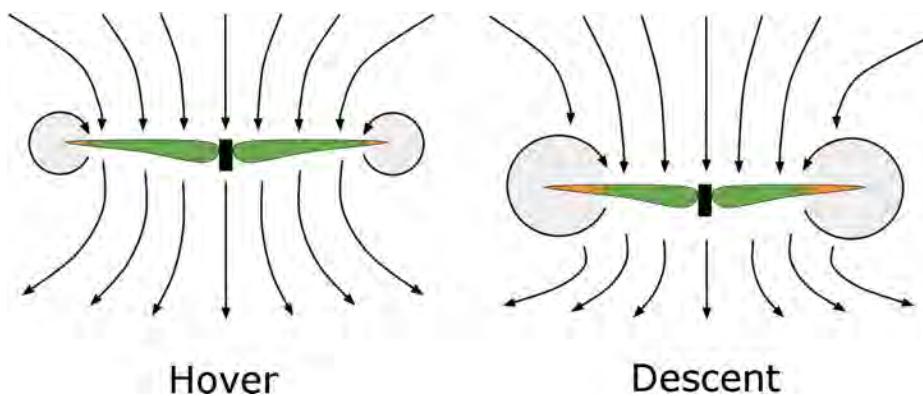


Figure 8-6 Prop tip vortices get much larger when descending

However, when descending rapidly, the additional upward air movement increases the effective angle of attack of the prop blades and thus increases the size of the vortices, as seen in the second half of [Figure 8-6](#). As the vortices become larger, they reduce lift and our rate of descent increases. Add power to stop the descent and the vortices get bigger yet, creating a vicious cycle that helicopter pilots know as "settling with power."

As multirotor pilots, we experience the vortex rotor state (VRS) as an inability to power out of a descent coupled with increased turbulence. Full power may not be enough to slow your descent and prevent a crash, making you think your battery must be low. Then, when taking off again (assuming the crash wasn't bad!), full power makes your drone head for the sky. Thankfully, there's a simple answer.

Just don't go straight down! Whenever descending rapidly, be careful to also maintain some forward speed. Of course, in a drone backward or sideways will do just as well. By moving horizontally, we continually fly out of the vortex rings we are creating instead of going down through them.

IMU/AHRS



Now that we understand the basic aerodynamic forces acting on our drones, we can create a control system that uses these forces to make our drone do what we want. But in order to control our position we must know our position, and for this we start with an *inertial measurement unit* (IMU). Inertia describes the tendency of objects at rest to stay at rest and objects in motion to stay in motion. In other words, inertia embodies Newton's first law of motion.

Inertia explains why you get thrown back into your seat when the car you are in accelerates or why you get thrown to the side when the car turns. Your body tried to stay in one place when the car moved or tried to keep moving in a straight line when the car turned. By way of ingenious design, engineers have built tiny structures into integrated circuits that measure these forces.

When these chips measure straight-line acceleration, we call them *accelerometers*. While accelerometers will detect the acceleration of the drone due to its own thrust, we primarily use accelerometers in drones to detect the constant force of gravity. When trying to keep a drone up, it's important to know which way is down.

When our integrated circuits measure rotation, we call them *gyroscopes*. Large spinning-disk gyroscopes were used for this job in the past, and the name has stuck even though today's chips operate on a different principle, as we will see. In drones, we use gyroscopes to stabilize our rotation. They will not tell us which way we are facing, but they will let us keep facing the same way.

By combining accelerometers to determine the direction of gravity and gyros to determine our rotation rates we can keep ourselves level with the horizon and keep ourselves from spinning around, but we can't tell which way we're going. In other words, we have an absolute reference for roll and pitch but not yaw. Enter the *magnetometer*.

Our magnetometers work like little compasses in three dimensions. By adding this data to our inertial sensors, we can hold an absolute heading. With a new sensor comes a new acronym. We now call the system an *attitude and heading reference system* (AHRS), since it can give us a complete and absolute reading of our direction.

Sometimes we need to know not just exactly where we are heading, but exactly where we are—for example, for complete autonomous waypoint navigation. To accomplish this we will have to add more sensors, like an altimeter and GPS, but those are subjects for later chapters.

Gyros

Engineers have developed a number of interesting devices for stabilizing aircraft over the years. Orville Wright, in one his few developments independent from Wilbur, built a wing leveler that used vanes to measure air currents, but it never worked very well. Lawrence Sperry built the first really functional autopilot around the same time, using spinning gyroscopes for attitude reference. Gyroscopes were the Sperry family business, Lawrence's father having invented the gyrocompass and gyro stabilizers for ships.

A little more recently, in 1963, scientists developed the laser interferometry or ring laser gyroscope. This operates by sending a laser beam in two directions around a path and noting the resulting interference patterns. As you rotate the system, the light's path is longer in one direction and shorter in the other simply because you have moved the point at which they meet. The different path lengths change the interference patterns (known as the *Sagnac effect*), and this can be measured to determine the rate of rotation. Very cool technology, but it has remained too expensive to see much use in model aviation.

The integrated circuit gyros we put into our modern drones are based on yet another technology: micro electrical mechanical systems (MEMS). Sometimes called vibrating structure gyroscopes, the IEEE has standardized the name of these components as *Coriolis vibratory gyroscopes*, since they depend on the same Coriolis force that makes storms turn one way in the Northern Hemisphere and another in the Southern Hemisphere. Gaspard-Gustave de Coriolis described this effect in 1835 after noting that long-range cannon shots tended to land to the right of their target in the North and to the left in the South. To see how this works, try the following thought experiment.

Imagine a tiny ball bouncing back and forth from the front to the back of your drone. Now, just as the ball has bounced off the back and is heading toward the front, imagine the drone turns right. If the little ball isn't connected to the drone it will keep flying in the same direction and thus land to the left of where it otherwise would have on the drone. In [Figure 9-1](#) the green bar represents the deflection of the ball's path, relative to the drone, due to rotation.

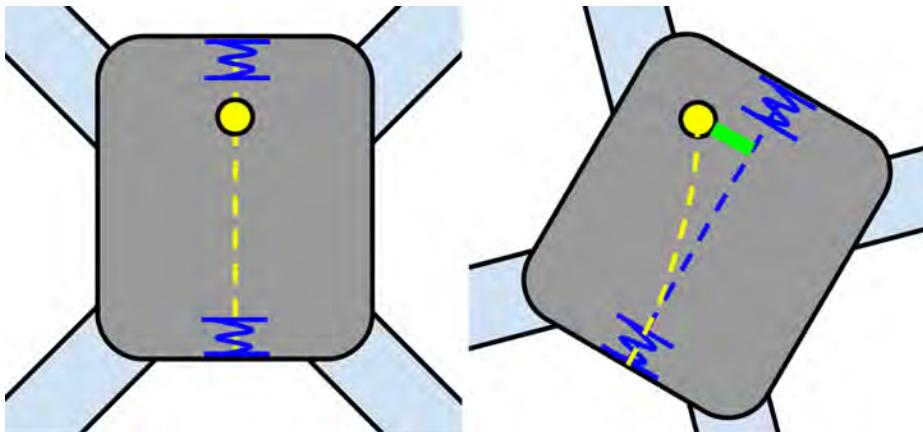


Figure 9-1 The ball ends up in the same place (it's the drone that moved!)

In much this same way, tiny vibrating structures inside the chips we put in our drones get pushed to the left or right as they get rotated. By measuring this deflection, the chip can then tell us how fast it is rotating. Several such structures are then used to measure rotation in different directions.

Accelerometers

Unlike the wide variety of gyroscope designs operating on different principles, all accelerometers work basically the same way. They measure the force exerted by a weight as the frame it is held in gets accelerated. Usually, the weight is on the end of a spring and we measure the deflection of the spring. Sometimes piezoelectric sensors—crystals that produce a voltage when pressed on—are used to measure the force instead.

The integrated circuit accelerometers in our drones use MEMS technology like the gyros. In this case the structures consist of a small mass held in place by springy, flexible arms. Small fingers on the side of the mass move relative to fixed fingers in the frame, which creates a variable capacitor that can be measured to determine how far the mass has deflected the spring arms. Each of these structures can measure acceleration in only one or two directions, so several such structures must be used to get a three-dimensional measure of acceleration.

Measuring acceleration can, of course, tell us how fast we are going. If we accelerate at 5 miles per hour (mph) per second for 3 seconds, then we are going 15 mph. If we then stop accelerating and travel at 15 mph for 1 hour, we travel at 15 miles, so we can use acceleration to find distance as well, though this takes extremely accurate measurements. If we are actually going a little more or less than 15 mph, the error gets compounded every hour we drive.

Some drones use acceleration to determine velocity and thus position, but this is usually only done in advanced drones and as an addition to GPS. In this case we need only use this

so-called inertial navigation between GPS updates, which are generally from 1 to 10 times per second, so errors don't have time to accumulate too much. Nonetheless, such a system puts a premium on vibration reduction in the control system since vibrations can add up to look like straight-line motion to an accelerometer.

More importantly for us, in addition to their being affected by the motion of our drones, gravity pulls on the little weights in the accelerometers, thus telling us which way is down. It is this information that we will make use of as we combine accelerometer and gyro data to create a complete inertial measurement unit. In [Figure 9-2](#) you can see how the deflection of the horizontal and vertical springs changes as the drone tilts forward.

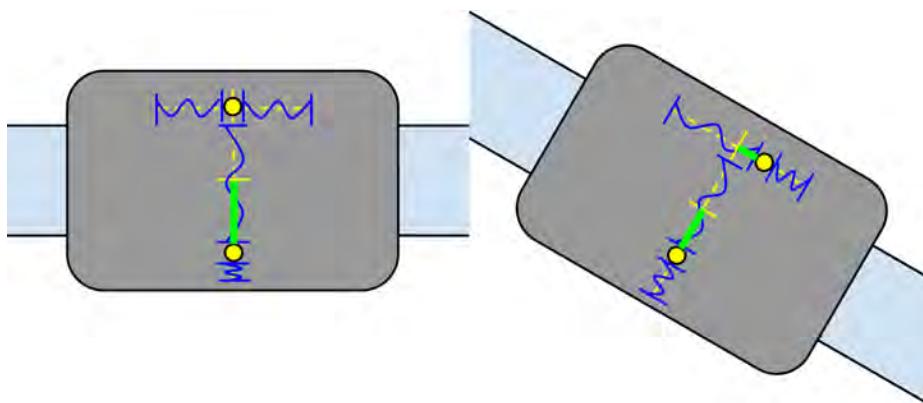


Figure 9-2 By comparing the deflection of the two balls on springs, we can determine the direction of gravity

Of course, real gyro/accelerometer chips like the MPU-6050 we chose have much more complicated structures than these examples. The springs are really flexible arms and there are lots of little fingers sticking out of the structures, which form variable capacitors by which to measure their movement. These tiny changes are detected, digitized, and made available to us as a few values on the inter-integrated circuit (I2C) bus. It's the job of our IMU code to read this data and make sense of it.

IMU

The IMU reads the raw data from the sensor chips, combines it, and converts the results to values we can use like roll, pitch, and yaw. We will use the Arduino [Wire](#) library to communicate with the MPU-6050 chip. Our first messages to this chip must be initialization codes. These set parameters for how to scale and filter the results before giving them to us. While these settings are critical, and may need to change if the fast loop timing changes, the code is highly specific to the application and unlikely to get reused, so let's move on to reading the sensors.

Once during each fast loop, we call the routines `readIMU()` and `calcIMU()`. The first one reads the raw values and the second calculates roll, pitch, and yaw from these values. We

keep these functions separate so that other functions, in particular calibration routines, can read the raw values and do other things with them. To read a set of values from the I2C bus, we make a request and then read bytes until the response is complete. In our case we will read seven 16-bit values, for a total of 14 bytes. The request looks like this:

```
//-----
// read gyro and accel
void requestIMU()
{
    Wire.beginTransmission(MPU);
    Wire.write(0x3B);           // starting with register 0x3B (ACCEL_XOUT_H)
    Wire.endTransmission(true);
    Wire.requestFrom(MPU,14,true); // request a total of 14 registers
}
```

When reading the response, we also check for any bus errors. When the message is complete, we have to do some of that ugly integer math to combine the bytes into 16-bit words. Here's the result:

```
//-----
// read available IMU bytes and process if full message
uint8_t imuBufPos = 0;
uint8_t imuBuf[14];
void readIMU()
{
    // read the whole IMU set
    requestIMU();
    while (imuBufPos < 14) // block until enough data read
    {
        if (Wire.available() > 0)
        {
            imuBuf[imuBufPos++]=Wire.read();
            if (Wire.getError() != 0)
                i2cErrorCnt++;
            debugVals[2] = i2cErrorCnt;
        }
    }
    if (imuBufPos == 14)
    {
        Acc[0]=imuBuf[0]<<8|imuBuf[1];    // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
        Acc[1]=imuBuf[2]<<8|imuBuf[3];    // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
        Acc[2]=imuBuf[4]<<8|imuBuf[5];    // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
        Tmp=imuBuf[6]<<8|imuBuf[7];      // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
        Gyro[0]=imuBuf[8]<<8|imuBuf[9];   // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
        Gyro[1]=imuBuf[10]<<8|imuBuf[11]; // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
        Gyro[2]=imuBuf[12]<<8|imuBuf[13]; // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)

        //Acc[2] = -Acc[2];
        imuBufPos = 0;
    }
}
```

Now for the fun part, the calculations! We can choose from several thoroughly researched algorithms, including the Kalman filter (which we will revisit later), the complementary filter, and the Mahony/Madgwick algorithm. The Madgwick implementation of Mahony's method has the advantage of being available as simple source code with clear rights. It also has a history of working successfully in other model aircraft projects. Finally, since it is implemented with quaternions, the Madgwick optimized code is extremely efficient.

What's a quaternion, you ask? Invented in 1843 by Rowan Hamilton, who was so pleased with the thought that he carved the formula into the bridge he was crossing in Dublin when he came up with it, quaternions are used in computer graphics and navigation, among other things, to represent rotations. They make some computations easier and avoid *gimbal lock*. Gimbal lock occurs in real gyroscopes when they reach some physical limit of motion. Something similar occurs in our code when we reach some limit of the mathematical representation of rotation. It's like standing at the North Pole and asking which way is south.

Stripped of the language of higher math, the process goes something like this:

1. Read the raw gyro and accelerometer data.
2. Calculate our rate of change based on current direction and gyro data.
3. Determine the difference between this gyro-based estimate and an estimate from the accelerometers.
4. Correct the gyro-based rate of change by nudging it toward the accelerometer-based direction according to a tuning parameter.
5. Update our direction estimate based on this corrected rate of change and the amount of time that has passed.
6. Convert the quaternions to roll, pitch, and yaw.

Have a look at the actual codebase to see the details, or just [read about the Madgwick implementation online](#). The code in our IMU has been changed to use only our calibration data and to output the so-called Tait–Bryan angles of roll, pitch, and yaw.

So far, we have described a 6DOF system, with three degrees of freedom each for the gyros and accelerometers. Now, let's add three more for the magnetometers.

Magnetometers

Magnetometers measure the direction of the magnetic field passing through them. Usually the Earth's field dominates this measurement, but interference from magnetic fields generated by the drone's own power system can be a problem. We get this information in the form of a vector, much like the accelerometer data, and the magnetometer data gets used in the attitude calculations in much the same way as the accelerometer data.

Mathematically speaking, this is quite significant. While a 6DOF system gives absolute roll and pitch relative to the Earth's surface, it can only give yaw relative to where it started, and that is limited by gyro drift. When we move up to a 9DOF system, we can now calculate an absolute, three-dimensional rotation relative to the Earth. Our equations give a single solution.

Practically, going from six to nine degrees of freedom does not buy us much new functionality by itself. We could implement a so-called *headless mode*, where the sticks on the transmitter correspond to absolute directions like north and south, instead of forward and backward relative to the drone. While this is interesting, the magnetometer becomes essential when we add GPS and try to do autonomous navigation.

With the addition of the magnetometer our IMU gets a new name. Such systems are commonly known as *attitude and heading reference systems* (AHRSs), or, in academic literature, *magnetic, angular rate, and gravity* (MARG) systems.

Next, we'll bring our IMU results together with the RC commands.

Mode and Mix

10

In a car, you turn the wheel and hold it to make the car turn at a constant rate. In a fixed-wing plane, you turn the control wheel for a few seconds until the plane has rolled far enough, then return the wheel to nearly zero to hold the roll position. The roll angle determines the rate of turn, and the control wheel determines this indirectly by controlling the rate of roll. In the language of drones, we might call the car “angle mode” and the plane “rate mode.” Interestingly, drone pilots will use both of these modes and several others, sometimes switching between several in a single flight.

The current mode determines what the sticks on the transmitter mean. Combine this with our attitude, and the mode code outputs commands to the mix code like “pitch forward strongly” or “yaw left slightly.” The mix code takes these commands and determines how much throttle to send to each individual motor. In other types of aircraft, this mixing might also control some servos or possibly require complex sequences of movements (“bank right, then hold bank angle”). By sticking to multirotors, we eliminate a lot of complexity and thus make it possible to create something functional while we learn.

Mode

I’ve said that the mode determines the meaning of the sticks on the RC transmitter. But before describing the modes we will use in detail, a word about another use of the word *mode*. RC transmitters come in mode 1 or mode 2 (or some are switchable), which in this case means that the right stick or the left stick controls the throttle. One can reorder the channels in software, of course, but the throttle stick usually doesn’t have a spring, so this makes a real difference. Almost all transmitters used with drones use mode 2, which means throttle on the left. Now that we know which stick controls the throttle, we can dig in and figure out what the other sticks do.

The different control systems have different names for the various modes, even though the functionality remains very similar. What we will call *angle mode* corresponds to *stabilize*

mode in ArduCopter. In MultiWii, the closest thing to the *rate mode* we're about to implement is what's called *manual mode*, though the two are not exactly the same, as we shall see. As we add sensors, we must add modes to use them. We could add *altitude hold mode* to use the barometer data, and if we add GPS we will need modes for that too. Right now, let's look at the two basic modes, which depend only on inertial sensors—angle mode and rate mode.

Angle Mode

Almost all toy-class drones run in angle mode. In fact, if a toy drone has two different modes, they are probably just two versions of angle mode with different tuning parameters. Beginners find angle mode intuitive, as the angle of the stick corresponds directly to the angle of the drone. It also has the advantage that when you let go of the sticks, the drone goes back to level and at least stops accelerating, even if it does not stop moving.

When flying in angle mode, the angle of the drone with respect to the ground is proportional to the angle of the control stick. As we saw when tuning the X4Wii, several settings can modify this relationship even before we really do the mode calculation. The RC rate determines the ratio between the angle of the stick and the angle of the drone. The drone might move 20 degrees when the stick moves 10 degrees or only 5, which would be an RC rate of 2 or 0.5, respectively. The RC exponential setting takes this straight-line ratio and flattens it out near 0 so that we can make fine control adjustments more easily.

To calculate the final desired angle for roll and for pitch, we first convert the RC value to a float in the range -1 to 1, then multiply this by the RC rate and a parameter called `maxAngle`, which determines the actual drone angle when the RC value, as modified by the rate, reaches 1. To find the command value—the force by which we wish to rotate the drone—we take the difference between the desired angle and the actual angle. Divide this difference by 90 degrees and we have a measure between -1 and 1, which we call `axisCmd` and which a control systems engineer would call the error because the control system is trying to drive this difference to 0. Here's the code for pitch and roll:

```
// pitch/roll: axis cmd = imu angle - rc angle
axisCmd[1] = -((float)IMURoll - maxAngle * rcRate[1] * (float)((int16_t)rcValue[1]
    - (int16_t)MIDRC) / halfRange) / 90.0f;
axisCmd[2] = -((float)IMUPitch - maxAngle * rcRate[2] * (float)((int16_t)rcValue[2]
    - (int16_t)MIDRC) / halfRange) / 90.0f;
```

We interpret the yaw stick in rate mode even when the system overall is in angle mode. It would be awkward to have to hold the stick to the left or right the whole time we wanted to face in that direction. Instead, the stick controls the rate of change of yaw. A small deflection might mean "turn 5 degrees per second," where in true angle mode it would mean "face 5 degrees to the right." Similarly, when we center the stick, the drone keeps pointing where it's pointing. It doesn't snap back to north.

To control the yaw rate, we must first compute it by taking the difference between the current heading and the previous heading. The function `circleDiff()` does this correctly, even when the two directions are either side of north. The next line smooths the yaw rate

to eliminate noise and vibration. The general form of $X = (1 - \alpha)X + (\alpha)previousX$ can be used anywhere a signal needs to be cleaned up. When α is near 0, the signal passes nearly unchanged. When it is near 1, the output changes very slowly, responding only to the average of the input.

Once we have computed the yaw rate, the code for yaw looks much like the code for roll and pitch. We just take the difference between the actual and desired values. The code ends up like this:

```
const float yawRateSmooth = 0.01f;

// compute and smooth the yaw rate
float yawRate = deltat_inv * circleDiff(IMUHead, prevHead);
yawRate = (yawRateSmooth * yawRate) + ((1.0f - yawRateSmooth) * prevYawRate);
prevYawRate = yawRate;

// yaw: axis cmd = imu rate - rc rate
axisCmd[3] = (yawRate - 90.0f * rcRate[3] * (float)((int16_t)rcValue[3]
    - (int16_t)MIDRC) / halfRange) / 90.0f;
prevHead = IMUHead;
```

In the next section, we'll cover full rate mode, where pitch and roll are handled the same as yaw has been here.

Rate Mode

What we will call *rate mode* many people know as *acro* or *acrobatic mode*. MultiWii calls this *manual mode*, though that seems less descriptive since we still stabilize the drone in this mode.

Rate mode does not require any absolute reference to the ground. The drone will just keep rolling left as long as you hold the stick to the left, no matter which way is up. As such, we do not need accelerometers to implement rate mode, and historically, when gyros and accelerometers were in separate chips and harder to get, many pilots flew exclusively in rate mode with nothing but gyros.

In our control system, however, we have accelerometers. In some cases you might want to avoid using them to avoid the noise the sensors introduce, but it doesn't seem to be a problem here, so we will simply calculate roll and pitch rates from the IMU's existing roll and pitch angle outputs, just as we did with yaw above. In fact, the code is so similar, we won't take up space repeating it here.

Mix

When I'm flying out in public and someone stops to ask, "How does that work?" more often than not what they want to know is something about how the mixing works. It's a natural enough question. You can see the control sticks on the transmitter and clearly the motors are the things being controlled, but how does one lead to the other?

We covered most of the answer back in [Chapter 8](#), really. For example, we saw how turning the motors faster on one side and slower on the other can make the drone roll. The code for this is pretty simple too. First we set the output for all motors based on the throttle (`axisCmdPID[0]`), then we modify each individual motor based on the appropriate axis (roll in the following example), scaled by the current throttle:

```
// set throttle
for (int i=0; i<4; i++)
    motorCmd[i] = (int16_t)axisCmdPID[0] / throttleCorrection;

// apply roll
motorCmd[0] += (int16_t)(axisCmdPID[0] * (-axisCmdPID[1]));
motorCmd[1] += (int16_t)(axisCmdPID[0] * (-axisCmdPID[1]));
motorCmd[2] += (int16_t)(axisCmdPID[0] * ( axisCmdPID[1]));
motorCmd[3] += (int16_t)(axisCmdPID[0] * ( axisCmdPID[1]));
```

The pitch and yaw sections are similar, except for which motor speeds go up and which go down.

One fine point and one huge subject are hiding in that example. As we saw in [Figure 8-2](#), when a drone is tipped to the side, only part of the thrust becomes lift. Dividing by `throttleCorrection`, a measure of this reduction calculated for us by the IMU, adds a little thrust to make up for this. It's a fine point, but it makes our drone fly a little easier.

Now for the big subject: perhaps you noticed that while our mode code used the variable `axisCmd` as its output, the input here is actually called `axisCmdPID`. The PID here is an algorithm that effectively does a little smoothing in between the mode and mix sections. This topic deserves a chapter of its own.

Hidden between mode and mix we find the proportional–integral–derivative (PID) controller. On the one hand, it's just a little smoothing filter applied to the axis commands before sending them to the motors. On the other hand, as a drone maker and pilot, you may well spend more time dealing with PIDs than any other component of the control system—not changing the algorithm, since that comes in but a few standard flavors, but *tuning* the PID's parameters. And few things will make as much difference in how your drones fly!

The Algorithm

As with gyroscopic stabilization, at least some of the credit for the PID algorithm goes to the Sperry family. In this case it was Elmer Sperry, father of Lawrence, who built the first autopilot for airplanes; he came up with the idea while working on an autopilot for Navy ships. He noticed that the helmsmen would compensate for not just the current course error, but the immediate history of that error and any sudden changes in the error. He was implementing this with pneumatic systems, but the principle remains the same to this day.

P—Proportional

When the person at the helm moves the rudder further as the ship gets further off course—that is, if the rudder position is proportional to the course error—then that person at the helm is acting as the “P,” or proportional, part of the algorithm. Similarly, if you press the accelerator pedal of your car harder in proportion to how much your speed is below the target, you are implementing the “P” part of a PID controller.

The “P” term controls the overall force the control system will use to achieve the desired output. Not enough “P,” and your drone won't have the power to respond to your control inputs. The controls will feel mushy. Too much “P,” and your drone will overreact and possibly oscillate wildly.

I—Integral

So now you go up a hill in your car and slow down a bit, making you press the pedal harder. Perhaps this is enough to hold your new, slower speed, but there is a persistent error of a couple of miles per hour. When you give it a bit more gas to fix this ongoing error, you are now adding the “I,” or integral, part of the algorithm.

A mathematician might describe an integral as the area under a curve. In [Figure 11-1](#), the blue line represents the error and the red line the area under this curve, or the sum of past errors. As long as the error stays *positive* (the green areas), then the larger the area gets, the higher the red line goes. When the blue line goes below the axis we say the resulting area is *negative* and subtracts from the total. When we add up all the areas we get the sum of all past error.

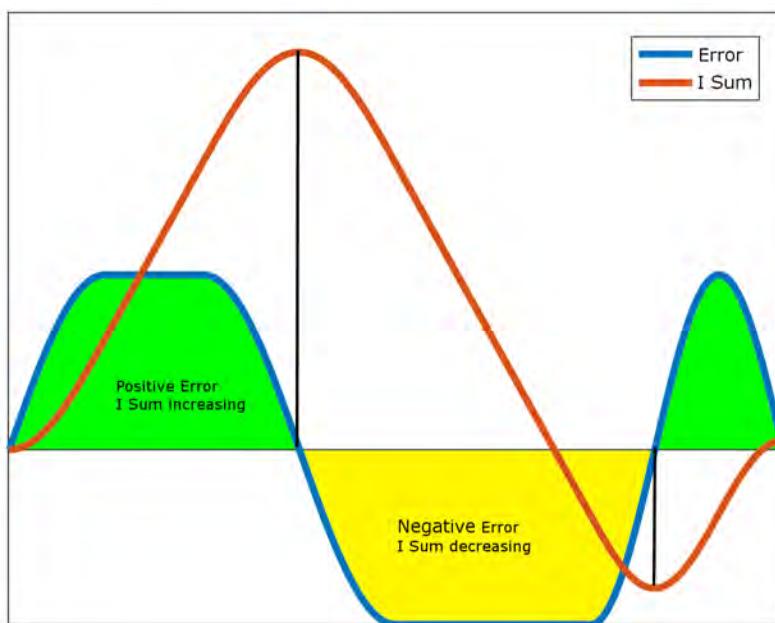


Figure 11-1 The red line is the sum of all past values of the blue line

The “I” term corrects for long-term error to make sure that the output finally reaches the desired value. This also causes the output to reach the desired value faster. Increasing “I” makes your drone feel snappier, responding quicker to control inputs, but too much can cause overshoot and oscillation, much like too much “P.”

D—Derivative

If the integral term is about the long-term error, the derivative or “D” term is about the instantaneous response. In the moment when you first hit that hill and your car begins to slow down, you might just give a bit of extra gas to keep it from slowing down too much

while you find the right pedal position to maintain your speed. The “D” term responds to the change in the error, not the current size of the error.

A mathematician would probably describe the derivative of a curve as the slope at a point. If we plot the speed of your car versus time, then as you start up the hill, the curve will slope downward. Similarly, if you don’t take your foot off the gas quickly enough when the road levels out, your speed might suddenly increase and the plot of your speed will slope upward. In [Figure 11-2](#), the red line is above zero when the blue line is rising and below zero when it falls.

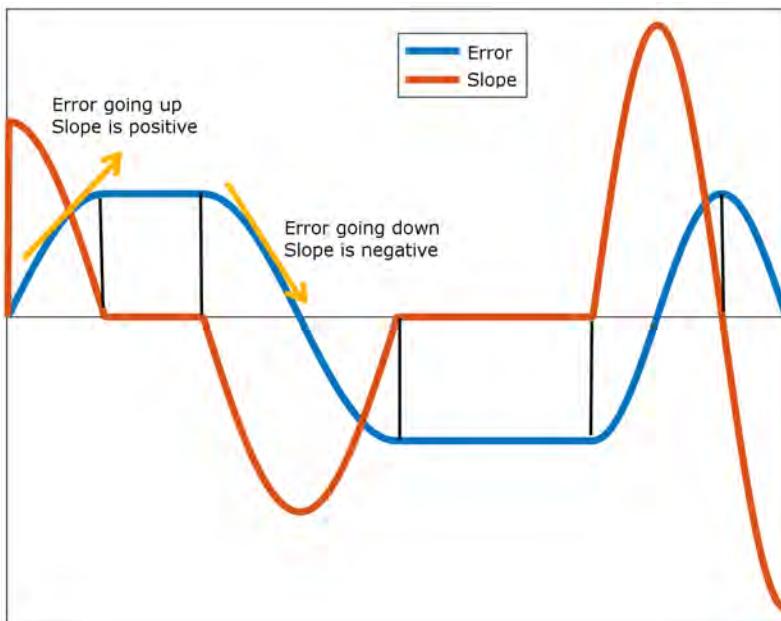


Figure 11-2 The “D” term depends on the slope of the error

The “D” term makes the immediate response faster and can help prevent overshoot and damp out oscillations.

PID or Pee-Eye-Dee?

How do you pronounce “PID”? Does it rhyme with “squid,” or do you pronounce each letter in the acronym? I’ve met professional control systems engineers who say it both ways, and I

always use the short version inside my head. But pronouncing the letters, like “Pee-Eye-Dee,” seems to be the more common usage, among both engineers and drone pilots.

Implementation

For an algorithm that has so many uses and that gets so much academic attention, implementing it is pretty easy. There are a couple of fine points between theory and practice even here, though.

In theory, the integral or sum of all past errors will just keep getting larger and larger if it is not corrected completely. In practice, it rarely works out well to let a variable increase without limit, so actual PID implementations usually include a limit on possible runaway integrals, called *integral windup*. We see this in the following code, where `constrain()` is used in the calculation of `pid_isum`:

```
// compute PID for each axis
for (int i=0; i<MYPIDITEMS; i++)
{
    e = axisCmd[i];
    pid_isum[i] = constrain(pid_isum[i] + (e * deltat), -1, 1); // constrain to stop windup
    pid_ddt[i] = (e - prevE[i]) / deltat;
    axisCndPID[i] = (pid_p[i] * e) + (pid_i[i] * pid_isum[i]) + (pid_d[i] * pid_ddt[i]);
    prevE[i] = e;
}
```

Tuning

Tuning a PID can be as simple as turning a knob on your transmitter until your drone flies better, or as complex as a full functional analysis of the system followed by an algorithmic calculation of the tuning parameters. We will use an iterative process of changing a value (perhaps with your smartphone) and doing short flight tests until a satisfactory result is obtained.

Theory

The following process assumes a starting point with "P" set to some low value, 0.5 or less, and "I" and "D" set to 0. At each step, change the parameter in question using the Bluetooth connection to your computer or smartphone, then take a short flight to test the results. A few seconds is usually enough.

1. Increase "P" until oscillations begin. Reduce to half this amount.
2. Increase "I" if needed to make control response quicker. If oscillations occur, back off some and leave it.
3. Increase "D" if needed to make response to disturbances quicker.
4. As "I" and "D" are increased, "P" may need to be decreased some.

This will all make more sense with an example. Let's say the setpoint represents the desired roll angle of our drone, shown in Figures 11-3 through 11-5 by a blue line. It starts out at the center, moves quickly to one extreme, then moves more slowly to the other extreme

and suddenly back. The green lines show the output of the PID controller and the red lines show the resulting roll angle of our hypothetical drone.

In Figure 11-3, we start with “P” set to 0.5, which is a little low. The resulting angle does not move as far as the setpoint. As we increase “P” to 1, the result gets closer to the setpoint, and the little oscillations are starting to get very noticeable. By the time we have “P” set to 2, the oscillations are too much, so according to our procedure, we back off and set it to 1.

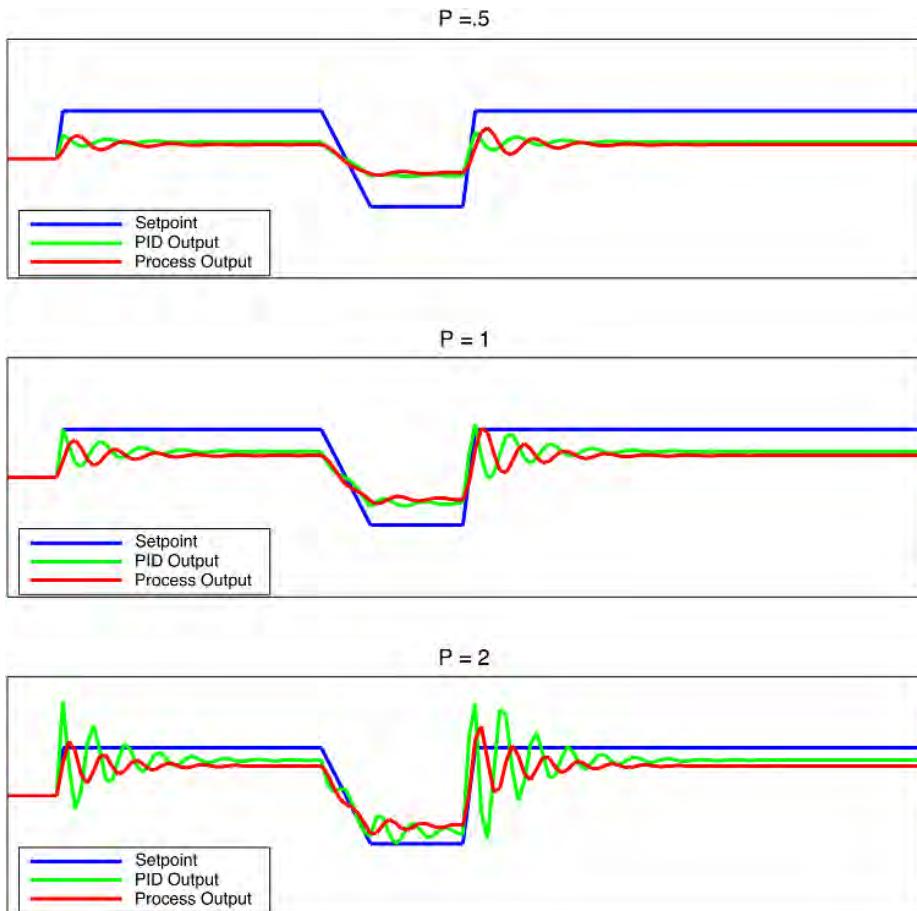


Figure 11-3 Even with “P” turned up until the oscillations get bad, the output still does not reach the setpoint

A drone with too little “P” just won’t have the maneuvering power to respond to your control inputs or disturbances from the wind. The word “wobble” comes to mind. With too much “P”, we get oscillations. This might seem a lot like the low-“P” wobble, but it will usually be faster. The drone drives itself back and forth instead of just flopping around.

In Figure 11-4, we add a new, cyan-colored line to show how the integral term adds up. In the top graph, we leave “I” at 0. You can see that the integral just keeps getting bigger or

smaller as long as we hold position, since we are doing nothing to correct the error. In the middle graph, we set “I” to 0.5, which just corrects out the error, but also adds to the oscillations. With “I” set to 1, the oscillations are increasing, not decreasing, so we must back off a bit, according to our process. We will go forward with “I” set 0.8.

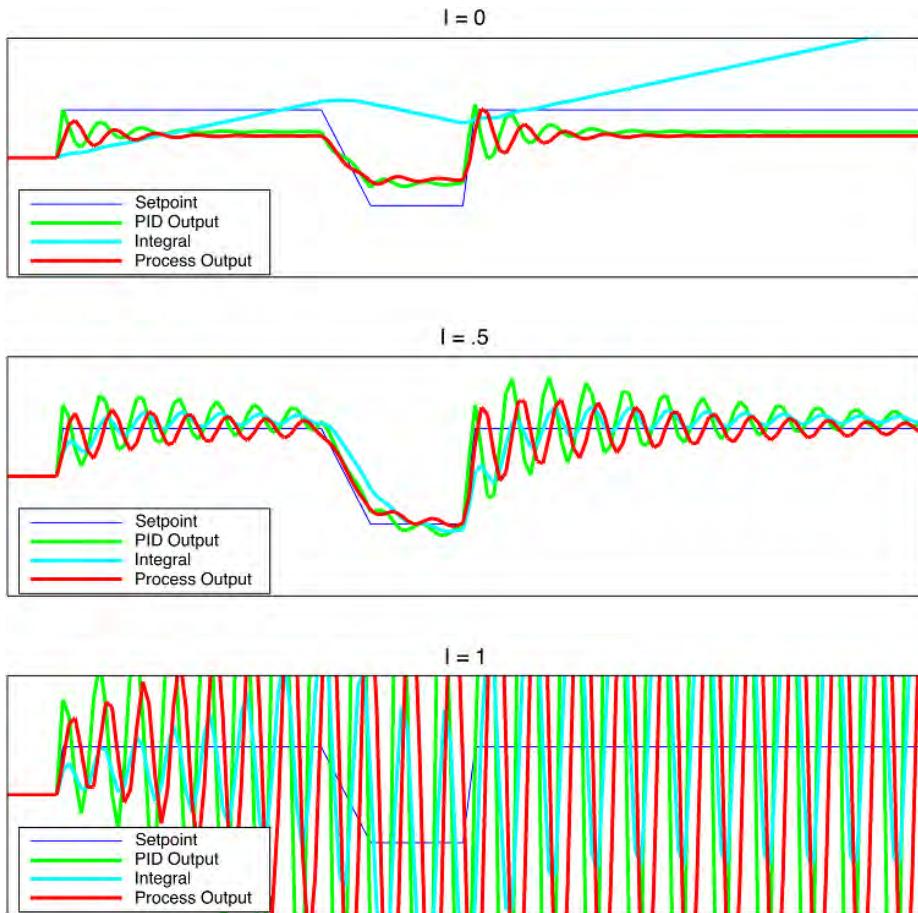


Figure 11-4 Increasing I brings the output up to the setpoint but adds more oscillation

Setting “I” to 0 works fine sometimes, so it’s hard to say what too little “I” looks like, except that a little more makes everything a little snappier. Too much “I” creates obvious hard, fast oscillations, much the same as with “P.”

At this point, the oscillations are too high to tolerate, but just watch what the “D” term can do! After trying several values, a “D” of around 1.4 seems to most perfectly damp out the oscillations without slowing down the response or creating oscillations of its own. In the third graph we can see that pushing “D” up to 3 creates unacceptable ringing in the output.

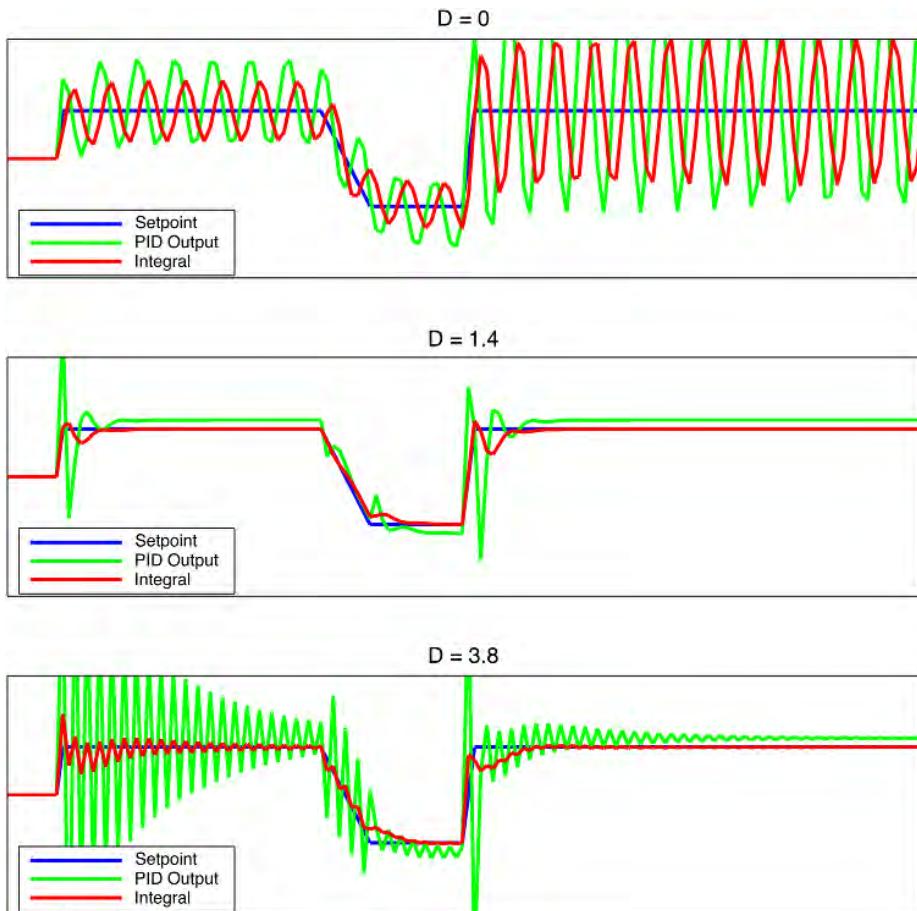


Figure 11-5 Increasing “D” can damp out the oscillations from “P” and “I,” or add more of its own

Too little “D” does not make a signature of its own; it simply allows the “P” and “I,” oscillations to continue.

Set right, “D” damps these oscillations out. Set “D” too high and we get oscillations again, but often faster those caused by “P” or “I.” They may, in fact, be too fast to see directly as many drone control systems run PID loops 500 times a second, but the drone will become unflyable nonetheless.

The final result looks quite flyable. At each fast setpoint change, the PID controller gives an immediate strong response, followed by a strong response in the other direction to stop the motion as the setpoint is reached. There is a single wobble after that as the system locks on to the new position, and then it stays perfectly flat.

Practice

Getting your PIDs tuned right can be frustrating, especially when more than one parameter needs changing and the resulting oscillations keep crashing your drone. But most people need never find themselves in this situation. In fact, in the most common case—trying to make a working drone fly a little better—turning PIDs can be both easy and satisfying.

If your drone flies well now but you would like it to be either smoother or quicker in its control response, then the PID adjustments should be easy. Actually, I would always start by adjusting the RC parameters, but assuming you have done that, you can almost always just move “P,” “I,” and “D” up and down in equal proportion. If the current settings are 1, 0.1, and 0.01, then adjusting up by 10% gives 1.1, 0.11, and 0.011. Move up or down in small increments and you should be able to tell that it’s getting worse before things get so bad as to cause crashes.

If you’ve just built a brand new drone, however, or made a major modification like changing motors or props, then you might have to make more drastic changes. If possible, find someone else who has built a drone using similar components and start with their values. I give working values for all the projects in this book, for example. If you can’t manage that, the default values in the popular flight control systems are often better than starting with “D” and “I” at 0. Some flight control systems, like ArduCopter, allow you to tune parameters in flight using your RC radio, which can make the process much faster; some can even perform an autotune and get to workable settings by themselves.

If you have a completely new codebase or otherwise are starting from scratch, then some kind of test rig will allow you to debug the code without crashing and breaking something. [Figure 11-6](#) shows the test rig I used to debug the PID algorithm in the Visible Drone’s code.

The PID algorithm is one of the most useful and important algorithms in control systems engineering. For me, understanding PIDs really starts to unlock the mystery of how things work. There really is more going on than meets the eye, but it is a process that we can understand and use to make our machines work better. Your drone will respond better. Your robot arm will move more smoothly. Your coffee pot will hold a more precise temperature. Not bad for less than 10 lines of code!

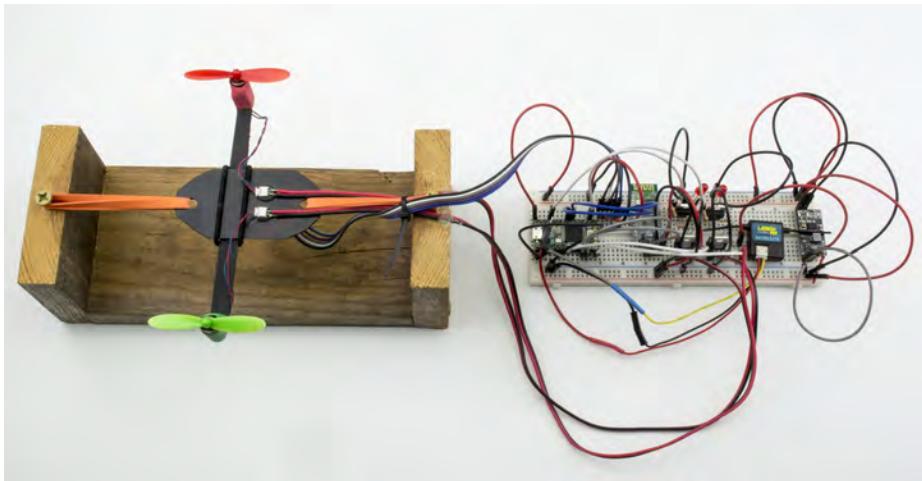


Figure 11-6 This rig allows a new PID implementation to be tested without crashes

Circuit Board and Motor Controllers

12

If small, smart flight controllers count as one of the breakthroughs that make drones possible, then surely the ease with which one can design small circuit boards and have them fabricated must be a part of the story. An individual maker can use one of a selection of open source tools to design a custom circuit board and have it fabricated in small quantities in a reasonable time for a reasonable price.

While these tools and services, combined with a little surface mount soldering, make it possible to design and build a controller every bit as integrated as the tiny toy drones, we will not target quite that level of integration. Circuit board design for things like blinking lights or even motor controllers is pretty simple. Adding things like the radio frequency (RF) components of the receivers and the high-frequency digital signals of the processor can make the component layout much more critical and thus require a great deal of research into chipset data sheets. Fun as that reading might be, we will compromise and build our controller out of modules.

Our circuit board will provide power to the different modules and connect each to the Teensy CPU's various ports. I could not find a widely available brushed motor controller module to use, so we will have to include that on our board at the component level. That turns out to be only two components per motor, so building that part will be easy.

At each step in the design, we must compromise between the ability to construct the results with only the soldering skills required of any other Arduino project and keeping the size and weight low enough that the drone will actually fly.

Design Tools and Files

I have used Fritzing to design the circuit board for this project. Fritzing, available from fritzing.org, wins for being the most commonly used design tool and for having a direct connection to a circuit board fab with reasonable single-piece rates. The design software is free, so I recommend that you download it and open the final design file for this project, available at www.makedronesbook.com/projects/VizDrone, even if you don't intend to modify the design or order one. It will be easier to see what's going on when you can zoom in and out or turn the layers on and off.

Component Placement

Our board design should maximize ease of construction, while keeping size and weight down. Measuring the interior space available in the Syma X5 body, the biggest board we can fit in there will be about 55 mm square. The mounting screws go in three of the four corners of a 25 mm square.

The Teensy goes in the middle, so that we can get to the USB port. The surface mount parts in the motor controllers go on the corners to make soldering them as easy as possible. To save space, the gyro board will go up above the Teensy on a header and the RC and Bluetooth radios will be kept off the main board. This has the advantage of allowing us to reposition the radios if there is any interference trouble.

Next, we add the power module at the back, behind the Teensy. It fits on the connectors labeled J1 and J2. Along the edges go the sockets for the motors and solder pads for the LEDs. The theory here is that motors get changed fairly often, so using sockets prevents tearing up the circuit board with repeated soldering and desoldering. Finally, we have headers for the various ports and the component placement is complete, as shown in [Figure 12-1](#).

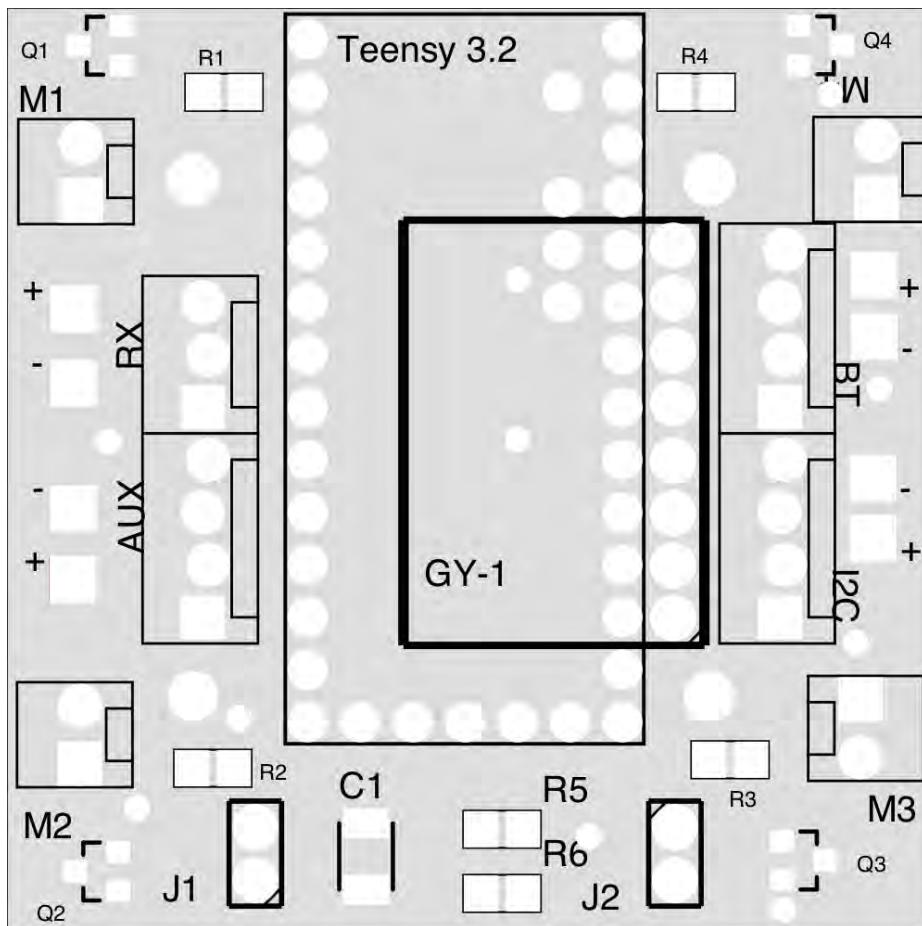


Figure 12-1 The surface mount components are on the edges to make soldering easier

Power Input and Distribution

The power system starts with a 1S LiPo battery, which will output between 4.2 and 3.7 volts as it discharges. The motors run directly off this voltage with the positive side connected directly and the negative controlled by the processor. The circuit board traces for these connections need to be as wide as possible because they will carry fairly large currents. Each motor can draw close to 2 amps, for a total of 6 to 8 amps at full throttle.

Everything but the motors gets powered through the Maytek voltage booster. This module takes the varying voltage from the battery and gives us a steady 5V. The 5V distribution is shown in red in [Figure 12-2](#). The RC receiver only takes 3.3V, which the Teensy conveniently provides, and is shown in blue. The unregulated positive voltage off the battery feeds the motors directly and is shown in purple. Ground distribution is black, of course. The ground plane has been removed for clarity.

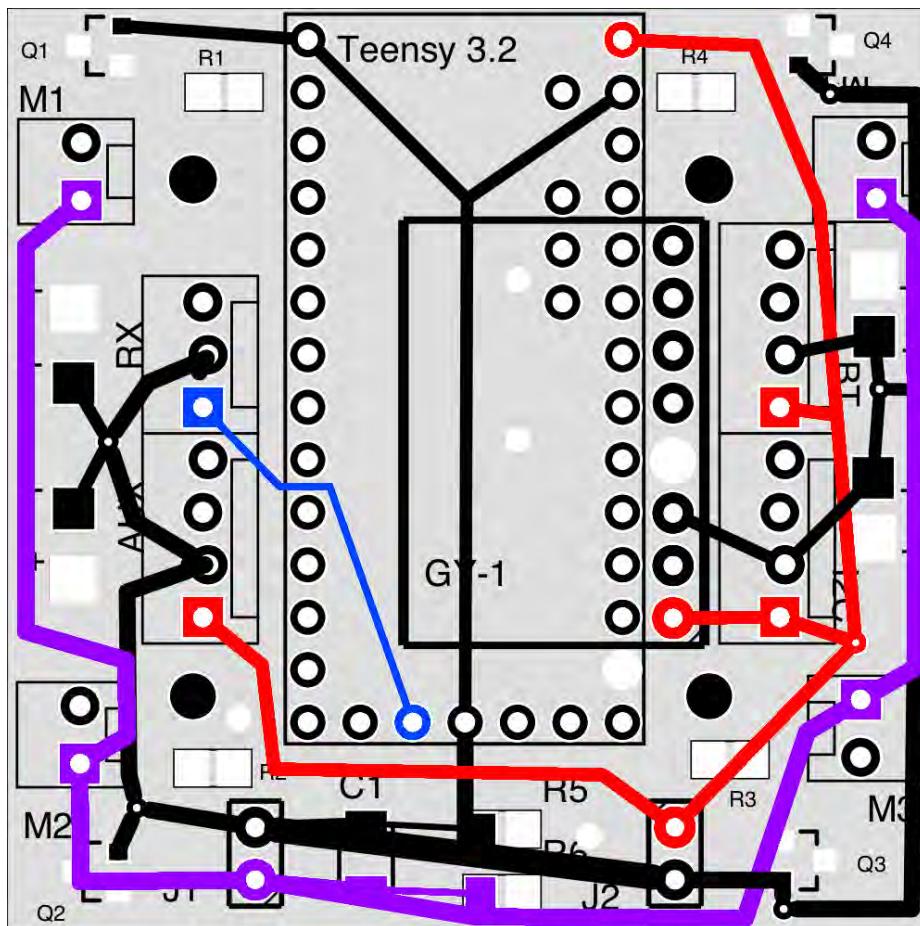


Figure 12-2 Power distribution for the Visible Drone (red is 5V, blue is 3.3V, and purple is battery voltage)

Also included as part of the power system is a large capacitor, C1, which protects the input of the power conditioner module from spikes as the motors power up. And so that the processor can monitor the power system, we include a pair of 10K resistors, R5 and R6, as a voltage divider and connect the result to one of the Teensy's analog input pins. When two resistors are placed in series like this, the midpoint will be at half the voltage that is put on the ends. We use this to bring the 4.2V LiPo cell voltage down to a range that the Teensy can measure.

Motor Controller

We will use the simplest possible motor controller circuit for the Visible Drone. Most small motor controllers use metal oxide semiconductor field effect transistors, called MOSFETs or just FETs, but some require more complex circuits. Ours will use a single FET and a single resistor, as we will see—but first, some explanation of what is not included.

To go forward and in reverse requires at least four FETs per motor instead of the single FET we use, but forward is enough for this application. Some designs would use a pull-down resistor on the input to make sure the motor doesn't come on when the processor boots up in a strange state. This hasn't proven to be a problem with the fast-booting and reliable Arduinos, though, and perhaps more to the point, the consequences of accidentally running a motor in a quad this size are not too serious.

Finally, this motor controller design does not include flyback or protection diodes. When driving an inductive load—that is, one that uses coils like motors and solenoids—the same mechanism that we drive with current will often produce current when pushed back the other way. Solenoids can easily produce three times their original drive voltage, in reverse, when released. This current needs somewhere to go! A diode placed across the load the right way will take this flow instead of the FET, which may not be able to take this higher, negative voltage.

In our application, the reverse voltage should never get higher than what we drive the motors with, since they are unlikely to spin faster by themselves than when we are driving them. Also, the FET we have chosen has a body protection diode. Putting this current through that diode doesn't keep it out of the FET overall, but it seems to be enough for this application. Few brushed motor drivers for small drones include protection diodes.

Choosing a FET

If you search for "FET" at Digi-Key or Mouser Electronics, you will get a bewildering list of results with over a dozen attributes each. For our application only a few of these specs matter. First, we can filter by package. We want an SOT-23 surface mount. Next, we have to be able to control it, so it must work with 3.3V logic levels. This really means that the FET must turn on with anything over 1.8V, so we need a FET with a $V_{gs} \leq 1.8$. Our FET must also be able to handle the operating voltage of our motors. This is an

easy challenge at only 3.7V; nonetheless, our V_{dss} , or voltage from drain to source, must be higher than that. Finally, the tough one: current handling. If the FET has much internal resistance when on (R_{ds}), it will dissipate too much power and get too hot. A low R_{ds} will give a high I_d , or drain current. Our motors can only draw about 2 amps each, but it's good to have some overhead in this area, so we'll choose a FET with more like 6 or 8 amps, since they are still available and inexpensive.

The one component other than the FET that we do include is a current limiting resistor. When first turning on, the gate input of a FET acts like a capacitor charging up and it can sink a great deal of current. Trying to sink too much current out of an Arduino pin could cause actual damage, or (much more likely) could cause a voltage sag inside the processor that causes a freeze or reboot. Neither will make for a good flight, so we add a small 100-ohm resistor between the Arduino and the FET.

Schematic and Layout

We need to add connectors for the various ports and solder pads for the LEDs, but these connectors all go directly to Arduino pins. They must be pins capable of pulse width modulation (PWM), but otherwise this is simple wiring from one header pin to another as far as circuit board design goes. As mentioned, keeping the RF and high-frequency digital signal local to each module keeps the routing rules simple. The result is shown as a schematic in [Figure 12-3](#).

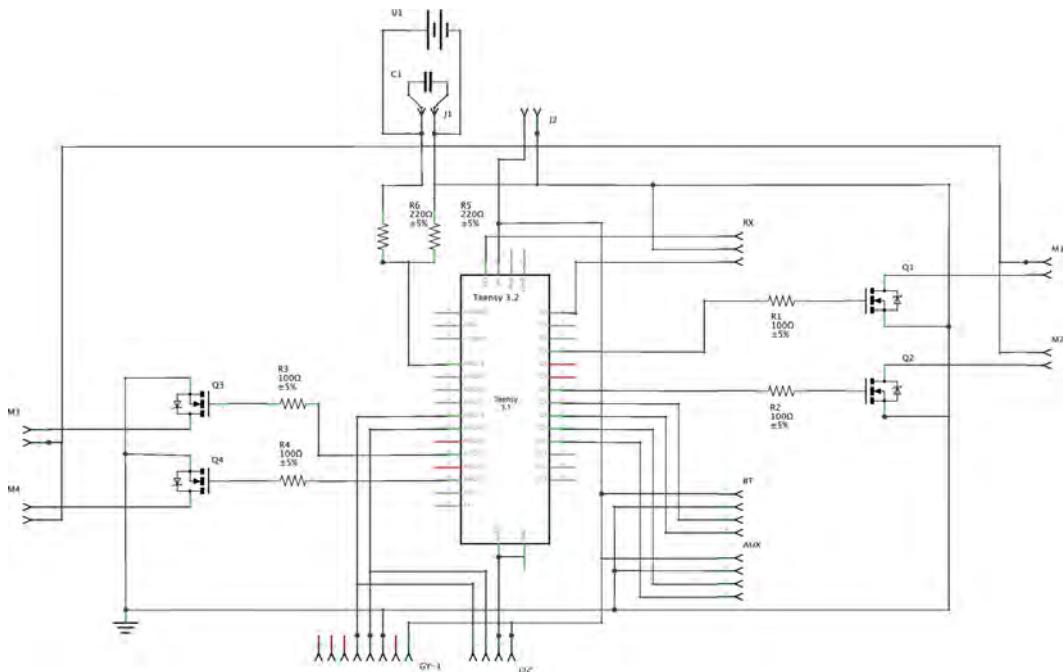


Figure 12-3 The complete schematic for the Visible Drone controller

[Figure 12-4](#) shows the final circuit board layout. The colors are yellow for the top layer data, green for the bottom layer data, red for the +5V regulated supply, purple for the raw battery power, black for ground, and finally blue for the single 3.3V power supply line from the Teensy to the RC radio.

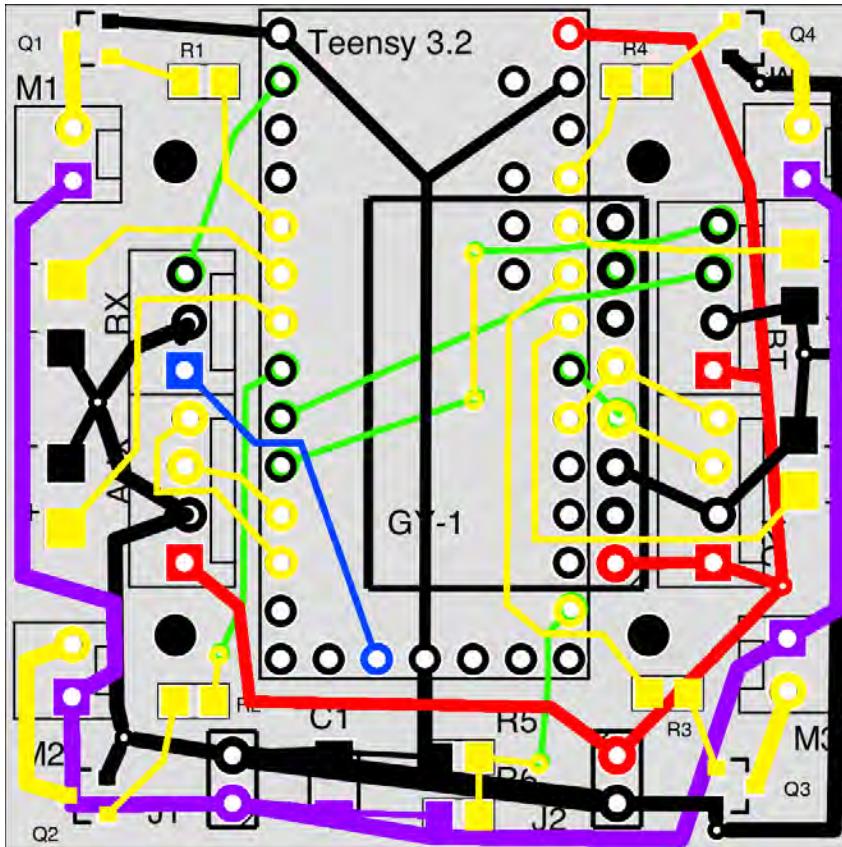


Figure 12-4 The complete Visible Drone controller circuit board layout

Figure 12-5 shows what the board looks like, fresh from the factory.

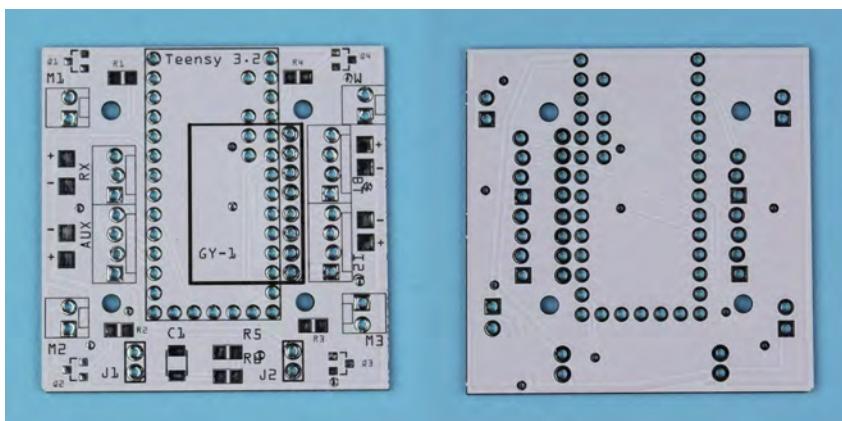


Figure 12-5 The front and back of the circuit board as it comes from the factory

Construction

Construction will be easiest if you solder in the surface mount components before adding the motor sockets and headers. A 100-ohm resistor and a FET go in each corner, as shown in [Figure 12-6](#), and the 100-microfarad capacitor and the two 100K-ohm resistors go in the space that will be under the voltage booster module.

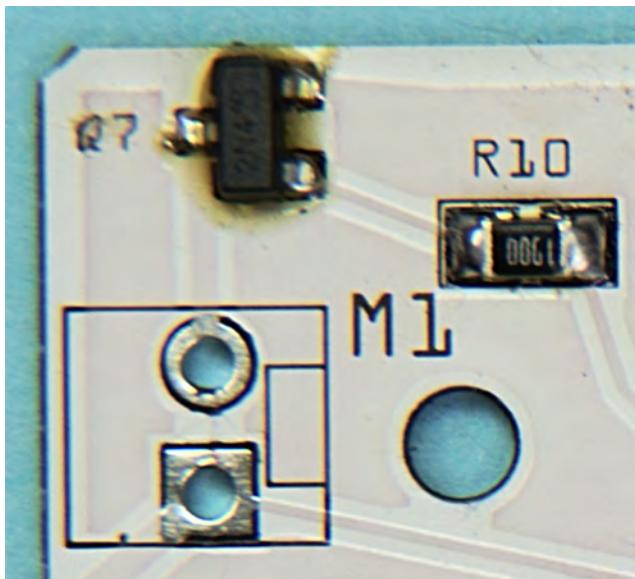


Figure 12-6 The discoloration around the FET is leftover flux, not burned board

SMC Soldering—Hot Air Versus Iron

Drones have forced me to learn how to solder surface mount components (SMCs). When the FETs blew on my favorite controller, I got out my smallest-tipped soldering iron, some copper braid, and some ChipQuick and I managed to replace them. While doing surface mount soldering with such gear is certainly possible, I highly

recommend investing in a hot air soldering station. Not only does it make the SMC soldering easy, but it has many other uses, like shrinking heatshrink tubing or flowing hot melt glue into little crevices. For less than a hundred bucks, it's a tool worth having.

With all the SMC components in place, let's build the power system. First solder the battery cable to the two pins nearest the word *in*. The wires should come out the bottom of the module, opposite where the components are. And of course, red is positive and black is negative. Next, solder pairs of straight pins into the remaining holes. The result should look like [Figure 12-7](#). When it is complete, you can solder it to the board. Clip the pins off on the bottom once you are done.

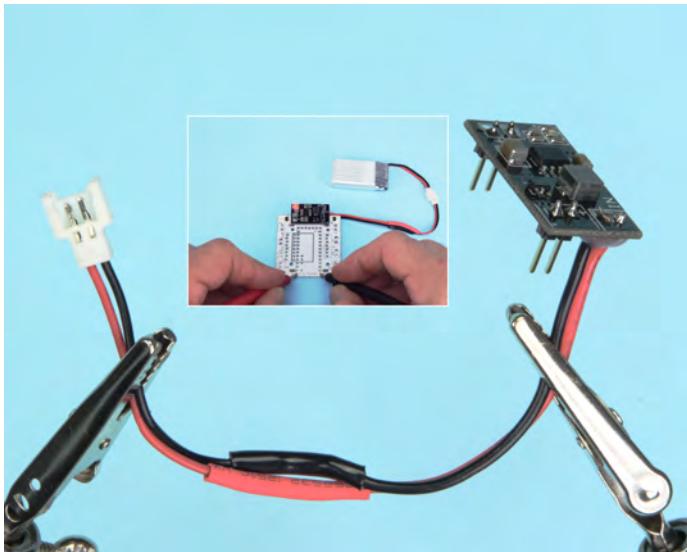


Figure 12-7 The Maytek voltage booster with battery lead installed (the inset shows how to test the power before adding any more components)

Even the best manufacturers make mistakes, so I recommend checking the power and ground to each module before adding it to the board. The inset in [Figure 12-7](#) shows how to test the Teensy's power. You can use [Figure 12-4](#) as a guide to the power distribution. Assuming it all checks out, you're ready to add the processor and gyro module.

Solder straight pins into all of the GY-88's holes and those on the outside of the Teensy 3.2. We don't need any of the inside pins or the pads on the bottom. They should look like the ones in [Figure 12-8](#). The female pins with the larger plastic case will be used to hold the gyro module up above the CPU.

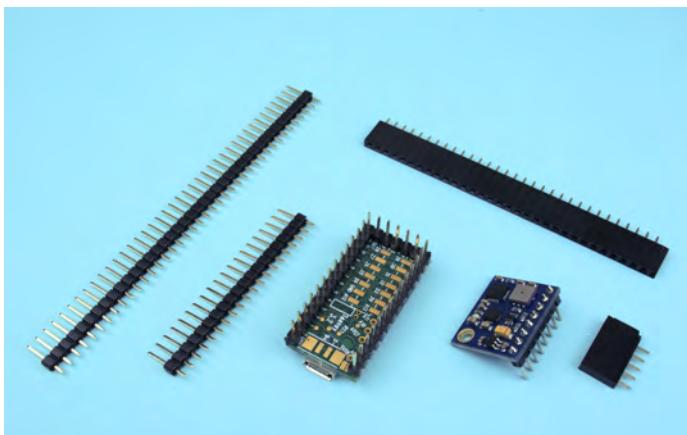


Figure 12-8 The Teensy and GY-88 modules with pins soldered on

We will now work from the inside out, putting the rest of the parts on the circuit board. The Arduino is in the middle, so we solder it next, followed by a row of seven female to male header pins for the gyro board. These will let the GY-88 sit above the Teensy. Now, solder straight pins in all the holes for the various ports with the long side on the top. Finally, put the micro JST sockets in for the motors, being careful which way they go. One of them faces in the opposite direction, as shown in [Figure 12-9](#).

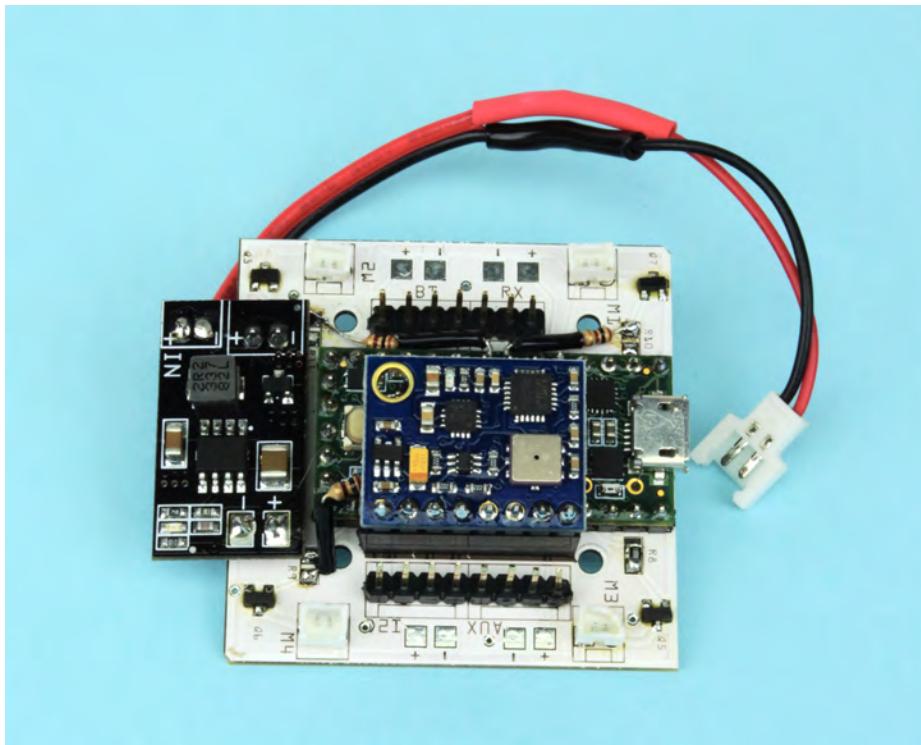


Figure 12-9 Note the one motor socket in the lower right that faces in the opposite direction

You might have noticed as you soldered the sockets in for the motors that they don't fit in the holes. OK, this was originally a mistake, but I have left it this way so that if you would rather just solder the motors to the board instead of using plugs and sockets, the pads are big enough for easy soldering. Given that they don't fit very well, I do recommend reinforcing with a little hot melt glue.

This completes the construction of the controller for the Visible Drone. It weighs in at a few grams more than the original controller, but it has much more power, more sensors, and more connectivity.

Construction and Tuning

13

We've been through several chapters now getting the pieces ready. We've been putting together flight control software, starting from the aerodynamics. We have built a controller to run it on, from power distribution to motor controllers. Now it's time to put it all together into a drone.

Install the Controller and Connect the LEDs

Getting inside the Syma X5 case means removing a lot of little screws, but they do give it strength when it's all put together. Once inside, unscrew the existing controller and cut all the LED and motor wires. Now screw in our new controller with the USB port facing the back where the battery goes in. Next, solder the existing LED wires to the new LED pads on each side of our circuit board. Red and yellow are positive. Black and white are negative. The result should look like [Figure 13-1](#).

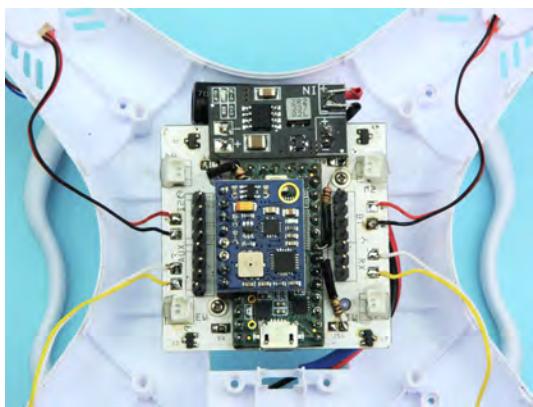


Figure 13-1 The existing LED wires connect directly to the new controller

Connect the Motors

The design as shown calls for putting plugs on the motors and sockets on the main circuit board. Given the likelihood of changing motors over time, this saves possible damage to the circuit board each time you unsolder and resolder something. On the other hand, the motors in this geared design are much better protected and should last much longer than the direct-drive motors we used in the X4Wii. You could skip the plugs and sockets and just solder the motors to the board. Assuming you do put sockets on the board, though, let's crimp some plugs on the motors.

It's pretty small work, and the crimp tool (mine's an AMP 29004-1) isn't as cheap as you might like, but I still recommend using plugs on your motors. The crimp-on ends come in strips that you can easily break apart by bending them back and forth a bit with long-nose pliers. Then put a single end into the crimp tool as shown in the first half of [Figure 13-2](#), being careful not to put any pressure on the crimp tool before you're ready. I find that crimping makes the pin too wide to fit in the plastic holder easily, but a quick squeeze with the long-nose pliers gets it back into shape. Using the little tab to hold it, stick each pin in one side of the plug body, as seen in the second half of [Figure 13-2](#). Be sure to get the polarity right! White and red go on the left as shown.

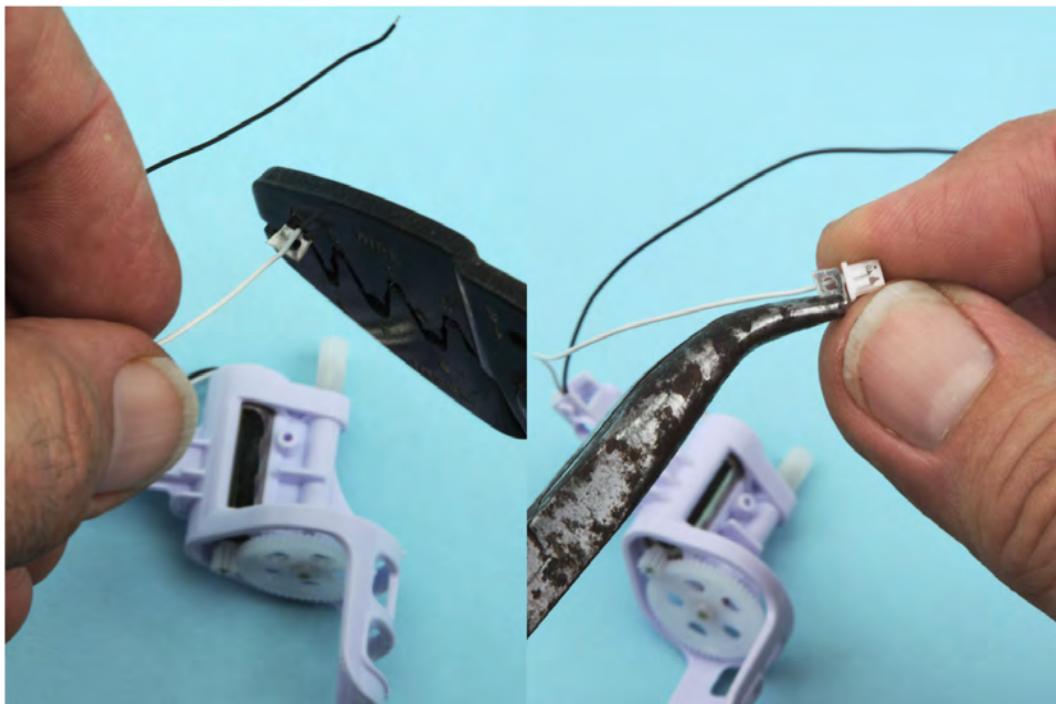


Figure 13-2 First crimp the ends on the motor wires, then stick them into the plug body

With the plugs on the ends of the motor wires, install the drive units in the frame and plug them in. Each motor goes to the nearest corner, of course, and assuming you put the sockets on the board correctly, one of them will face in the opposite direction. The result looks like [Figure 13-3](#).

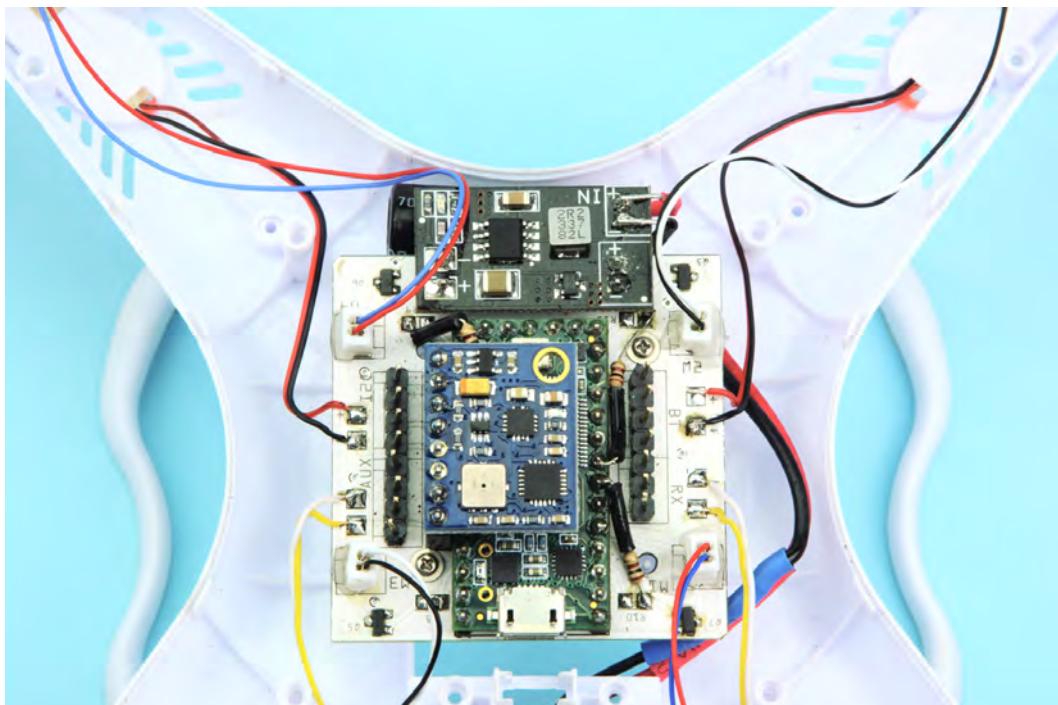


Figure 13-3 Be careful to get the polarity correct as you wire the motors!

Install the Radios

The physical installation of the radios consists of stuffing them down into the spaces under each side of the main circuit board. The HB-02 Bluetooth module plugs in using short, female header wires. Make sure the Vcc wire goes toward the front of the drone, where the voltage booster module is.

We will have to make a cable for the RC receiver. Cut the cable that comes with the Lemon RX receiver and solder it to half of a standard servo extension. Since servo connectors are symmetrical, it doesn't matter which way you solder the cable; just be sure when you plug it in to get the direction right. The power, which starts on the left side of the receiver as you look into the socket, should end up toward the front of the drone, the same as the Bluetooth module. [Figure 13-4](#) shows how it looks inside with everything installed.

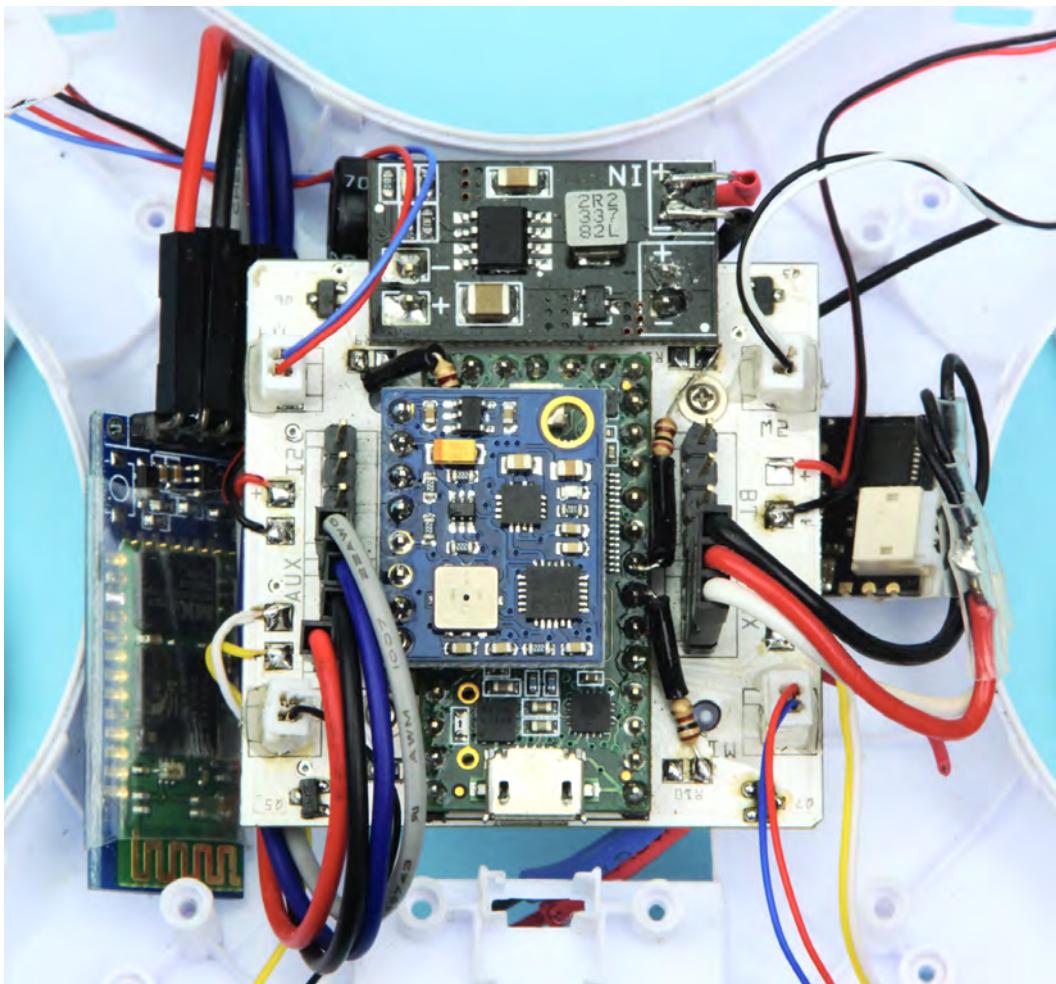


Figure 13-4 With both radios installed, it's starting to get a little crowded

Complete the Frame

We're going to want to get to the USB port on the Teensy, so we will have to make the hole that once held the power switch big enough to get a USB cable through. The thin plastic of the Syma X5 shell cuts easily with an X-Acto knife. Carefully cut just inside the two screw posts up to just above the words **ON** and **OFF**, as shown in [Figure 13-5](#).

It would be a shame to cover up all that cool-looking hardware, so next we'll cut a hole in the top and install a window. Try to cut right at the edge of the dome as shown in [Figure 13-6](#).



Figure 13-5 The existing hole must be widened to allow a USB cable through



Figure 13-6 Remove the dome so we can see our new controller

Ground Test

The dome that we cut out of the original top will be our guide for cutting a window. Any thin, clear plastic will do. The plastic lid used here once covered a buttermilk pie. Simply cut around the dome, leaving about a quarter inch extra. Now, hold the new window over the hole in the shell top and cut slits along the ridges of the arms, from the outside to the edge of the hole. This creates tabs that fit into the body in between the arms. Add a little decoration, and some colored props, and we have [Figure 13-7](#).



[Figure 13-7](#) The Visible Drone

Put all those little screws back in, and construction is complete. Are we ready to fly? Maybe. Perhaps we should do some testing.

Ground Test

Connect the Visible Drone to your computer via USB. You've already checked the power supplies to all the modules, so the likelihood of smoke at this point is low. Lights should immediately glow on the power module and the gyro module. To really make some lights glow, try the [VisDroneBlink](#) sketch, available at www.makedronesbook.com/projects/VizDrone. If this program works correctly, the LED pods on the arms should start blinking in sequence. No problems? Let's load the flight code!

We have to use USB to load new code, of course. For telemetry data, we could connect via Bluetooth, but that would require a battery, which in turn could then run the motors. The way our power system works, the motors will never run when on USB power, which makes our first tests safer.



Don't Use USB and a Battery Together!

You should never plug in a USB cable and a battery at the same time. This is true for most drones. Small differences between the voltages in the two supplies could lead to large current flows where they were not designed to go.

The latest flight code for the Visible Drone is available at www.makedronesbook.com/projects/vizdrone. Load this sketch into the Arduino IDE and upload it to your drone. If all is well, a few seconds after the processor reboots, the LEDs will flash around a few times. Next, let's see if all the sensors are working.

In MultiWiiConf, connect to the Visible Drone's USB port and press start. Several fields on the screen should turn green. The version will display 23. And if you pick up your drone and move it around you should see all the flight indicators track your movements. The graphs should show the data as well, as we can see in [Figure 13-8](#). If this all works, then let's try the RC radio.



Figure 13-8 The flight info and graphs should respond to the Visible Drone

Turn on the Blade MLP4DSM. After several seconds the light on the Lemon RX receiver in the drone should come on. Note that sometimes if the transmitter or receiver loses its connection and reconnects, the receiver light will blink. Other than telling you that there was a loss of connection, this doesn't affect anything. Now when you move the sticks on the transmitter, you should see the RC values changes on the MultiWiiConf screen, as shown in [Figure 13-9](#).



Figure 13-9 The RC values should change when you move the transmitter sticks

Flight Test and Future Work

The flight code we loaded comes with a reasonable set of default parameters, so you shouldn't have to change anything to fly. You may wish to change the RC rate or expo settings, as these are more a matter of taste than of finding some optimum value. The procedure for this is the same as what we did with the X4Wii in [Chapter 5](#).

The process for powering up and taking off goes like this:

1. Put a battery in the compartment and plug it in.
2. Place the drone on a perfectly level surface and wait for the flashes.
3. Arm the drone by pressing in on the right stick.
4. Test the motors with a short blip on the throttle.

That's it. Go fly! The Visible Drone as constructed here is pretty heavy for its power system. This means that it will only climb slowly and can have trouble pulling out of rapid descents. This drone, like the Syma X5 it is based on, seems to suffer more than small, direct-drive

drones from translational lift changes and seems especially prone to vortex ring state trouble when coming down. Both of these factors lead to this drone flying best when in constant forward motion.

There is so much we haven't covered—there are many fascinating things in the code that there was not space to mention, and you may have noticed that we haven't actually used the compass or barometer yet.

The Visible drone and its drone control system serve as a platform—a platform for future development and a platform for learning. But we've made it halfway through this book without spinning a brushless motor. We will remedy this situation next.

A Bigger Frame

14

It's time to build The Big One—the S500 (Figure 14-1). We're not going to set any records for size, but this drone will literally be in a different class. It will weigh well over the 250-gram limit for requiring registration with the FAA in the US. In fact, the frame alone will weigh more than that. Despite, or perhaps because of this heavier frame, one of the main goals of this drone design will be to carry a useful payload. Like most people, I will probably load a camera and gimbal on it first, but the possibilities are endless.



Figure 14-1 The completed S500 drone

The other design goals for the S500 include using brushless motors with electronic speed controllers and using one of the most powerful of the open source drone control systems—ArduCopter. This firmware will let us integrate the latest sensors quickly and tap into a world of more powerful ground control software. This control system with all of its peripheral parts will weigh more than the last complete drone we built, but it will be controlling motors that more than make up for the added weight.

Unlike the little brushed motors of our previous drones, which come in three or four sizes, brushless motors come in dozens of sizes, all with different power ratings, propeller mountings, etc. To make sure you have a combination of frame, motor, and prop that will work, look for examples of successful builds by others or kits based on the parts you are interested in. The DJI Flamewheel 450 and 550 quadcopters set the standard in this size range. The S500 frame we are starting with is right in that range at 480 millimeters, but by using off-the-shelf components and an open source controller, we will be able to add many advanced features.

The S500 Kit

The S500 drone frame kit can be ordered from several sources, including [Hobby King](#). It comes in two versions, one with power distribution integrated into the frame and the one we will use, which comes with a separate power distribution board. [Figure 14-2](#) shows what comes in the kit.

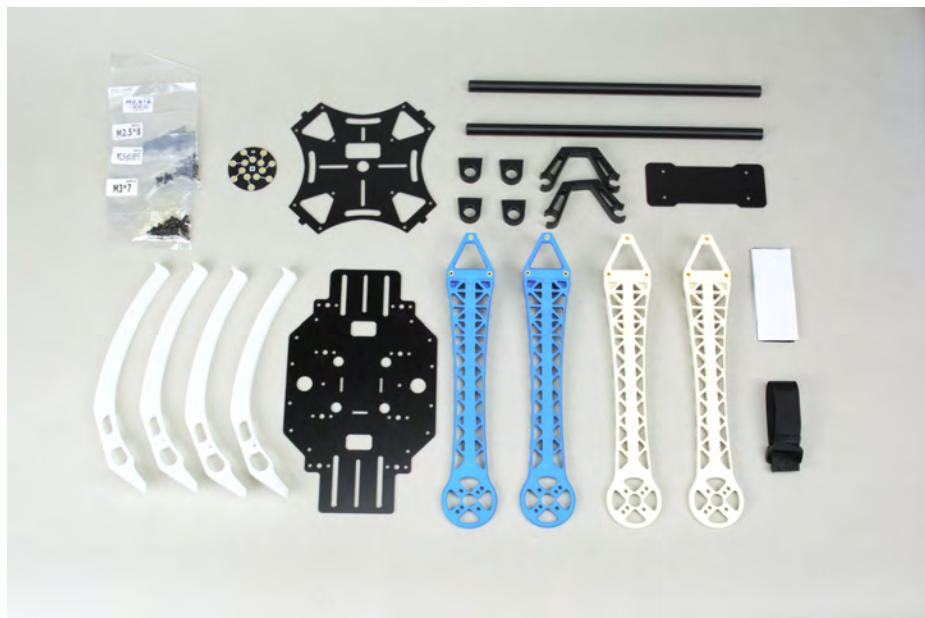


Figure 14-2 Contents of the S500 frame kit

Be careful to get one that includes the carbon fiber rods in the arms. Some of the cheap knockoffs don't include this, and they are not nearly as strong. You can tell by looking for little slots in the back of the arms that show the rods, as seen in [Figure 14-3](#).

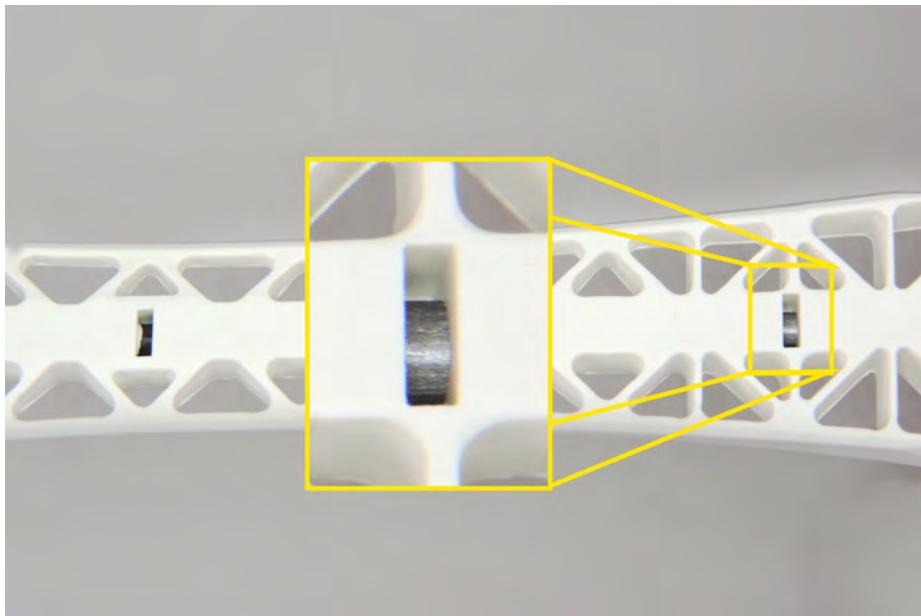


Figure 14-3 The slots in the bottom of the arm allow you to see the carbon fiber rod inside

Follow the Instructions

Like the famous IKEA instructions, the single page that comes with the S500 frame doesn't even try to use words, so there is no chance of bad translation. There is no sense in repeating everything here, but I will point out some exceptions to how they show to build it.

To begin with, don't do step 1. The directions show to mount the motors first, but we're going to wait until later, after we have everything else mounted on the arms. Step 2 covers building the undercarriage. There are two rods that mount underneath the frame on which a platform for strapping the battery can slide. The set screws that hold the rods in place are rather hard to screw in, but there are convenient holes in the frame's bottom plate for setting them.

The plastic parts that ride on the rails can be snapped into place by putting them in a little sideways and twisting. You can't do this once they are both secured to the battery plate, so snap them in before using the bevel-headed screws to secure this plate. The resulting undercarriage can be seen in [Figure 14-4](#).

Follow the Instructions

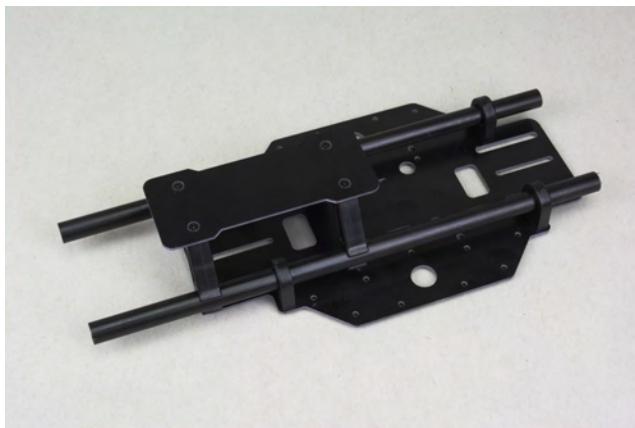


Figure 14-4 The rails allow the small plate to be repositioned easily

The set screws for the rods are accessible, so they can be repositioned. The mounts for the rods can even be moved, as there are extra sets of holes drilled in the bottom plate. The battery should be positioned in the middle if it is the only thing under there. We can move the battery back and perhaps even move the rods out of the way later, when it's time to hang a gimbal on the front.

The rest of the instructions can be followed as shown, until the very end. Since we are going to be going back into the middle of this frame to mount various control system components, just put one screw in the top of each arm to hold them in place. With this done it looks like [Figure 14-5](#) and you can start to tell that this is going to be a strong frame.

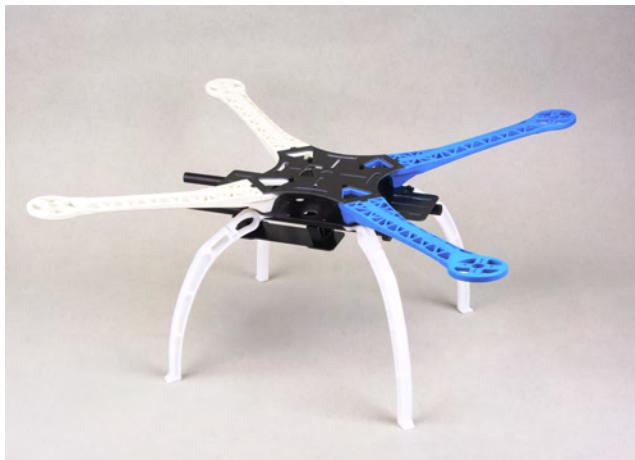


Figure 14-5 The S500 frame complete

Since it is put together with screws, too heavy for our smaller drones, the S500 will be easy to maintain. Spare parts are readily available and not too expensive. Let's add some motors, and it will start looking more like the powerful drone it will be.

Install the Power System

15

Let's start strapping hardware to that big frame. As usual, the first thing to add is power. One reason for this is simply that power wiring is much thicker and harder to route, so we do it first and route the smaller stuff around it. Also, once we know where the power wiring goes we can arrange certain other things, like the magnetometer, away from this wiring since the high currents can cause magnetic interference.

Our power distribution system must take power from the battery and get it to the four electronic speed controllers (ESCs). The rest of the control system runs on a regulated 5V supply instead of battery voltage, so that will get wired separately.

We will also mount the motors, to complete the power system, though we will only wire up the one whose ESC we wired power to. This way we can safely test one complete power to ESC to motor path without powering up the others.

In the following chapter we will explain enough about how brushless motors and ESCs work so that all of the testing and tuning that follows will make sense.

Mount the ESCs

ESCs are generally mounted on the arms of multirotors. They pair with the motors on the ends of the arms, so this makes basic sense. It also keeps the changing currents of the motor outputs away from the magnetometer and radios, where it will cause less interference, and should help with cooling.

In this case, I have chosen to mount the ESCs on the bottom of the arms. This takes them out of the direct prop wash, so the cooling effects might not be as great. Large, 1 1/2" clear heat shrink tubing holds the ESCs to the arms, as seen in [Figure 15-1](#). They come wrapped in one layer of heat shrink, so this shouldn't compromise cooling any further and it provides a very secure mount. The motors we have chosen should only draw 10 to 14 amps

compared to the 20-amp rating of the ESCs, so we should be fine (and if not, we can always upgrade to 30A ESCs).

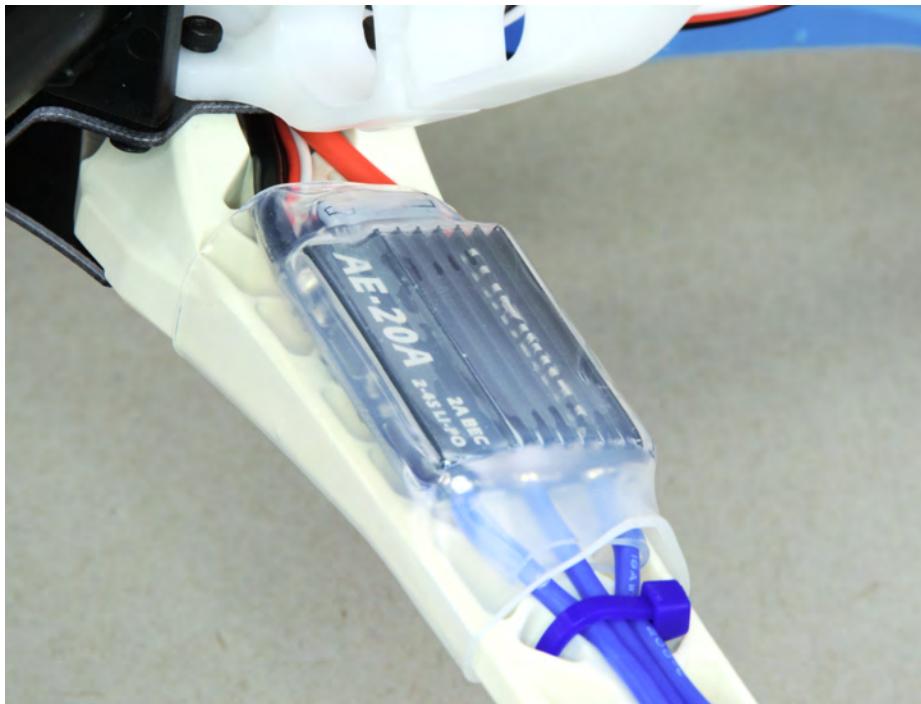


Figure 15-1 Clear heat shrink tubing holds the ESC in place

What we lose in cooling by mounting under the arms, we gain in wire routing simplicity. The motor wires are wire-tied to the frame to isolate their movement from the solder joints. The power and signal wires from each ESC fit neatly through the tunnels in the arms into the space between the upper and lower plates of the frame. And this is just where we will do the power distribution wiring.

Wire the Power Distribution Board

As mentioned, the S500 frame kit comes in two versions: one has the power distribution built into the lower frame plate and the other, which we are using here, comes with a separate power distribution board, shown in [Figure 15-2](#). This board has two sets of connected solder pads, one for the positive leads and one for the negative. Power comes in from a battery connector and goes out to each ESC.

Obviously, be sure to connect all the positive or red wires to the positive pads, which are the inner circle, and the negative or black wires to the negative pads on the outer circle. Any ESC can be wired to any set of pads. With the first ESC connected, it should look like [Figure 15-3](#).

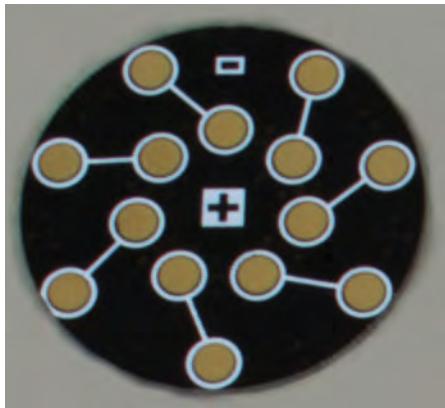


Figure 15-2 The power distribution board that came with the S500 frame kit

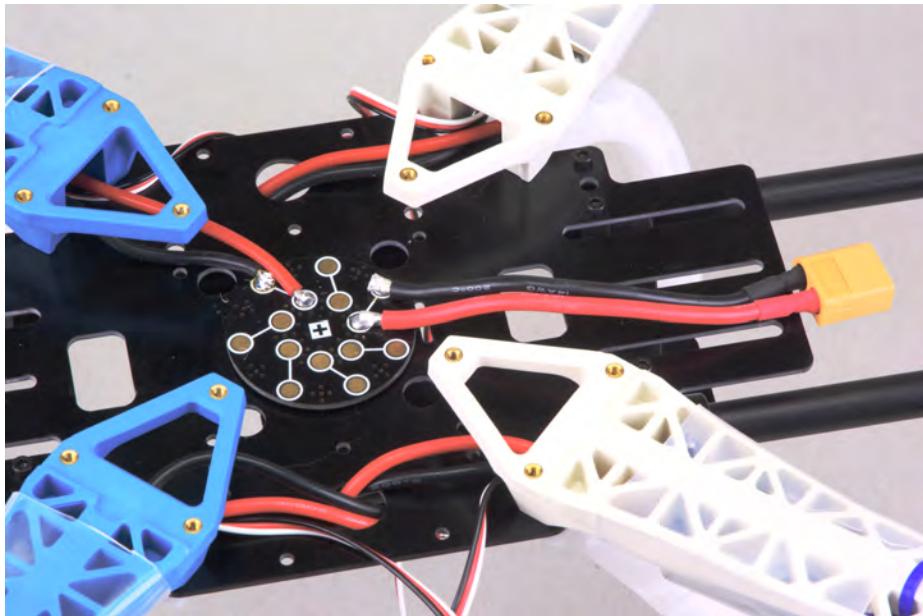


Figure 15-3 We wire only one ESC first, so we can test which direction the motor spins

Why only wire one, with everything else tucked out of the way? Well apparently, the easiest way to make a brushless motor turn the correct direction is to wire it up, make it spin, and swap any two wires if it goes the wrong way. The next chapter should help explain why this works.

Mount the Motors

The motors mount directly to the frame, using screws provided with the S500 frame kit. Each motor sits on top of the circle at the end of an arm. Then, since we have chosen to mount the ESCs under the arms, the motor wires can be routed down through the frame and then immediately wire-tied down. Be sure to screw the motors down pretty tight as they can be subject to a lot of vibration. [Figure 15-4](#) shows how this looks from underneath.



Figure 15-4 The motor wires are routed along the underside of the arm

Don't Trim the Motor Wires!

The wires that come out of the motor are the same magnetic wire that the coils are made of internally. This means that each strand is individually covered in enamel. The high melting point of this enamel makes it ideal for high-performance motors, but it is also very hard to

remove for soldering. Therefore, I highly recommend that you do all the necessary trimming to the output wires of the ESC, which are normal copper, and use the pre-tinned ends that the motor comes with.

Figure 15-5 shows how the motor sits in the middle of the frame end. It turns out that this provides pretty good, but not perfect, protection for the motor in a crash. You can go ahead and solder the one motor to the ESC we wired power to earlier. Put some heat shrink tubing on before you solder, but don't shrink it down yet and leave the other motors undone until you make sure this one is spinning in the right direction. For the first one, just trim the ESC wires and connect them in order. As the next chapter will show, all you have to do to reverse the direction is swap any two of the three wires.



Figure 15-5 The larger end of the frame provides some protection for the motor

Determine Direction

We have now put together enough of the power system to test and see which way the motors spin. You will need a charged 3S LiPo. Capacity doesn't matter right now. First, with the one ESC wired to the one motor, check all the connections once more for proper polarity. Make sure that when the battery plug is lined up with the socket, the red wires go together and the black wires go together. I have seen examples of batteries using the same connector with different polarity for different drones. Look good? Plug in the battery and see what happens.

We're mostly looking for no smoke here. There should be no loud pops. The ESC that is getting power will probably play you a little song, but it might also make no noise,

depending on the mode it is in. ESCs are known for playing different little tunes as a form of feedback on what they are doing. I find attempts to annotate these melodies in ASCII-art in the middle of badly translated English most entertaining. Assuming everything looks OK, or perhaps especially if it doesn't, unplug the battery.

Check Those Cables!

I have received a number of cables, both signal and power wiring, that came from the factory wrong. I have burned out regulators (which are actually easy to replace with a hot air reflow station), and I got a nice pop out of a PX4 power module that had its battery connector put on the wrong way. In [Figure 15-6](#), note that if you plugged in the battery sitting next to it, red would go to black and black to red.

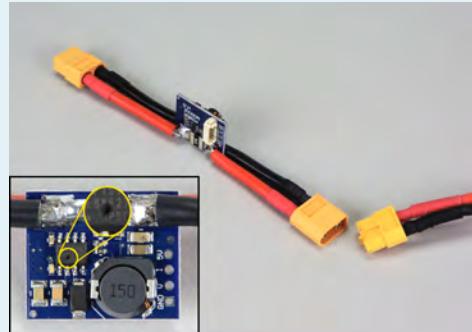


Figure 15-6 This bad cable popped a hole in the chip when plugged in

Next, we are going to add the RC receiver so that we can make a motor spin. We just need to plug the ESC's signal cable into the throttle channel on the receiver, which is channel 3. The ESC will provide power to the receiver. In [Figure 15-7](#) you can see that for channel 3 you plug into the fourth slot, since the first one is the combined output that we will use later to connect to the flight controller.

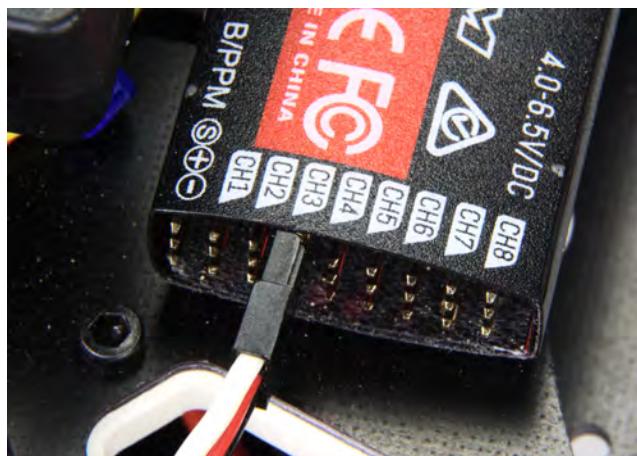


Figure 15-7 A single ESC plugged directly into channel 3, which is the throttle

For this test, make sure there are *no props* installed. You might want to put a bit of tape on the motor's axle if you are having trouble seeing which way it is spinning. Next, turn on your transmitter and pull the throttle all the way down. Now plug in the battery. The red light on the receiver should come on solid, meaning that it has established a link with the transmitter. If this looks good, slowly move the throttle up until the connected motor starts to spin. Check [Figure 15-8](#) to see which way each prop should spin (clockwise or counter-clockwise).

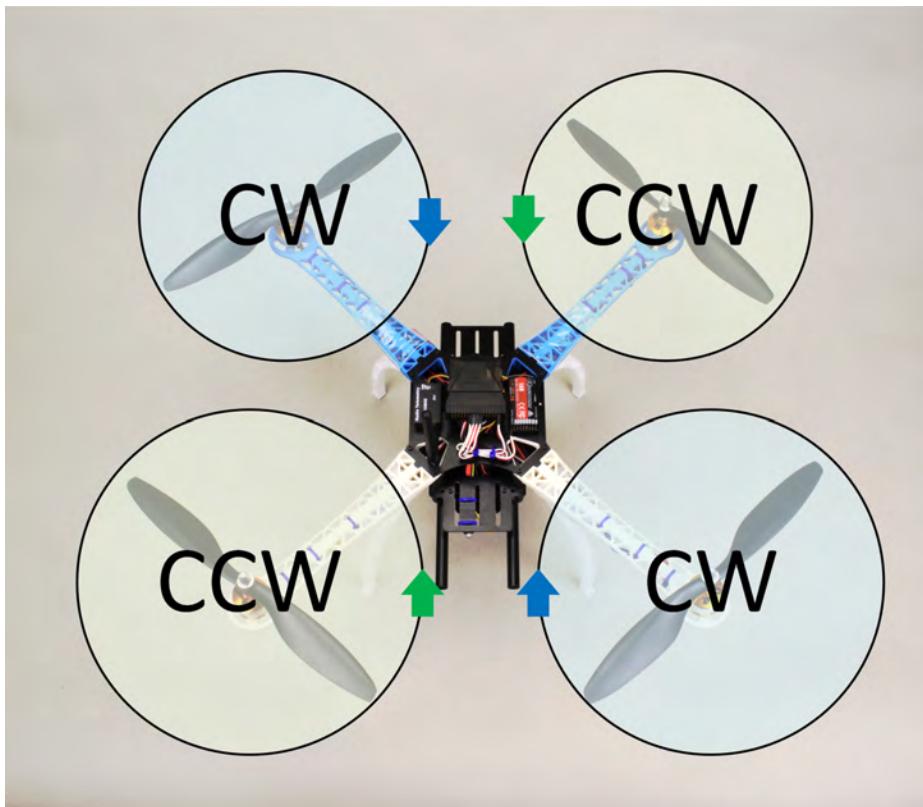


Figure 15-8 Swap ESC wires as needed to get the correct prop rotation

If the one motor you wired spins in the correct direction, then go ahead and shrink the heat shrink. If not, just unsolder it and swap any two of the wires between the ESC and the motor. Whichever way makes the first motor spin in the right direction, we can do the same on the opposite arm. The motors on the other two arms should spin in the other direction, so swap the wires or don't as appropriate. [Figure 15-9](#) shows examples of swapped and un-swapped motor wiring.

Determine Direction

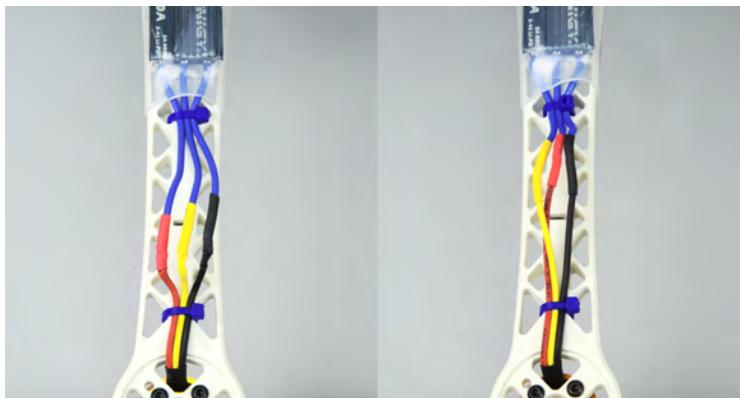


Figure 15-9 Swapping any two wires makes the motor turn the other way

You can also complete the wiring of the power distribution board at this time. When it's done, the result should look like [Figure 15-10](#). We may use some of those empty slots later to power video gear.

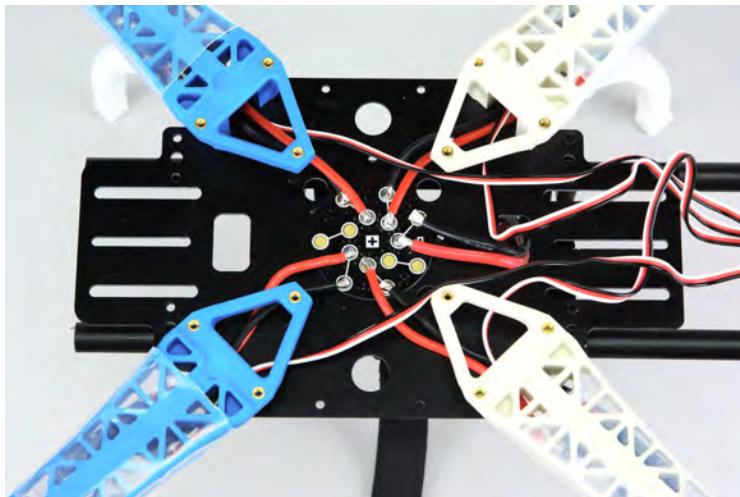


Figure 15-10 The power distribution board with all of the ESCs wired

Now that we have mounted and wired all the motors and their ESCs, in the next chapter we will talk about how these components work. It's a fascinating story.

Brushless Motors and Their ESCs

18

Aside from its size, what really sets this bigger drone apart from the other projects we've done are the brushless motors and the electronic speed controllers (ESCs) that it takes to run them. Technically, the single FET, fed by a single resistor, that we used to drive each motor in the Visible Drone counts as an electronic speed controller, but our brushless motors need much more.

Brushes, as contained in the small coreless motors we used previously, fail very quickly at the high speeds these motors turn. Eliminating them adds a great deal to the long-term reliability of brushless motors, but it means we need a more complex speed controller.

ESCs take the current from the battery and switch it to the different coils in the motor at just the speed commanded by the control system. They are really amazing pieces of kit, as we shall see in a minute. But first, a little about the history of a variety of brushless motors.

Brushless Motors

Many people had their first introduction to brushless motors back in the days of floppy disks, though they probably didn't know it. These motors were not fast or powerful, but they were very reliable, which was important given the constant use that floppies once received. This reliability stems directly from the lack of brushes for which they are named.

In a brushed motor, the coils move, so to get the current to the coils, springs hold stationary brushes against rings on the spinning axle. These brushes are usually small blocks or strips of metal. In a direct current (DC) motor, the rings are split into sections that go to the different coils, but the splits cause faster wear and can spark. In most alternating current (AC) motors, unsplit rings are used, which don't spark but still wear out.

In a brushless motor, we reverse the usual structure and have a moving permanent magnet going around a set of stationary coils. [Figure 16-1](#) shows the type of motor we are using opened up so you can see the coils and magnets. The coils can be connected directly

to the electronics with no rings or brushes or anything. The electronics must then turn on each coil in turn so that the magnets get pulled around at the desired rate. Of course, the electronics must know the position of the motor in order to do this correctly.

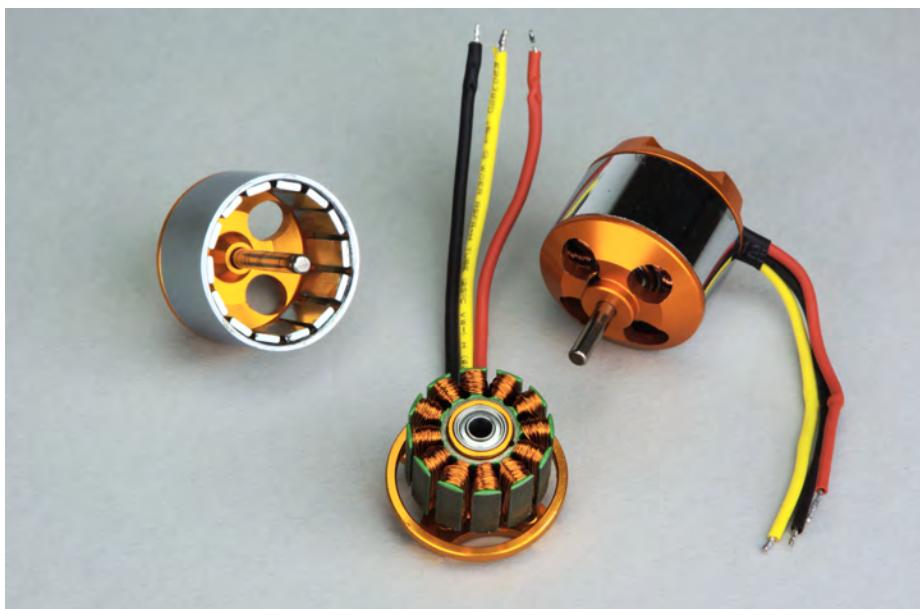


Figure 16-1 This motor has 12 coils and 14 magnets

In large, industrial AC motors, the position is usually determined with some sort of optical encoder. Gimbals have IMUs similar to our drones to determine their position. But both of these systems require extra components and wiring. Because they generally run fast and in the same direction, the brushless motors we use to drive propellers can use a simpler and very clever solution, which we will discuss in a minute when looking deeper into ESCs. First, a bit more about how the drive works.

The brushless motors we use in RC aircraft power systems, and those in brushless gimbals, and, in fact, the large majority of AC motors in the world, are three phase. In motors, this means that however many coils they have, they will be wired in groups of three. These groups of coils will then be energized in sequence so that as you move around the motor you pass coils from group A, then B, then C, then A again and so on. Remember, these coils are stationary. Around them move a series of permanent magnets with alternating poles facing in, first north, then south, then north again.

At the point in the sequence pictured in [Figure 16-2](#), the first A coil is drawing magnet X to it with a north magnetic field; coil B has just pulled magnet Y in with a south field, and since magnet Z has passed it, coil C will now switch off for a moment, then switch to a south field. As the sequence continues, magnet X will get drawn in turn to A, B, and C, then the next A.

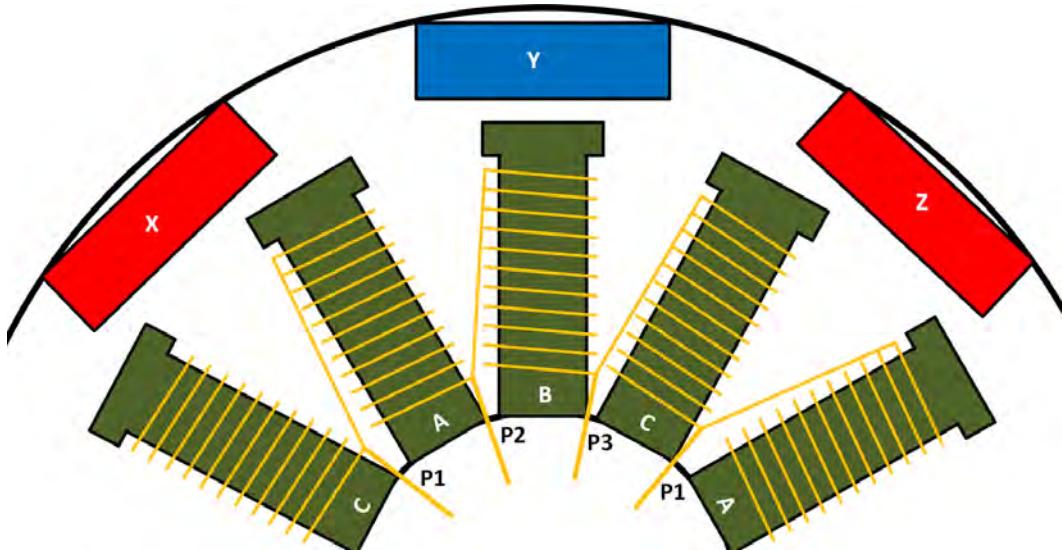


Figure 16-2 Brushless motors usually have three sets of coils, here labeled A,B, and C

While you might expect that we need six wires, two per coil group, to control this motor, the coils are actually wired in series so that we only need a total of three wires, labeled P1, P2, and P3.

Brushless motors may have as few as 4 or as many as 24 or more coils. The number of magnets, or poles, must be different and is usually slightly greater. This ensures that the coils and poles don't start out perfectly lined up such that the motor can't tell which way to go. Low-power, low-speed motors can get away with fewer coils. Brushless gimbal motors usually have lots of coils to optimize positional accuracy over speed or complexity.

I chose EMax MT2216 II KV810 motors for this drone—or rather, that's what I have on mine now. (I originally used some generic 2212 13T/1000KV motors that didn't work out.) What do all these numbers mean? Except for the last one, not much to us. The first, four-digit number in a brushless motor description gets broken up into two two-digit numbers, here 22 and 16. In our case these are the width and height of the stator, or internal, coils. In other cases, these numbers represent the size of the outer case. The last number, KV810, relates the no-load speed of the motor to the input voltage.

Some motors also get labeled with the number of turns of wire around each coil, like 13T for 13 turns. The KV rating of a motor depends largely on the number of turns used, so these numbers are directly (OK, inversely for the math geeks out there) related. In general, the higher the number of turns, the lower the KV. A lower KV rating means the motor will be better at swinging a large prop and carrying high loads. A higher KV rating means the motor will turn faster for a given voltage—good for small props and fast quads. The RC car community tends to use the number of turns rating much more than the RC aircraft world.

ESCs

An electronic speed controller, or ESC, takes power from the battery and sends it to the coils of the attached brushless motor in such a way as to make it run at the speed commanded at its input. Each one contains several FETs to control the power and some sensors to determine the motor's position, as well as an embedded controller to tie it all together. Yes, each ESC has its own small processor, and in some cases you can actually update the code, though we won't need to.

As mentioned in the previous section, the brushless motors and ESCs we use for RC applications are three phase. This means that the coils of the motor, however many it has altogether, are wired in three groups (consider [Figure 16-3](#)).

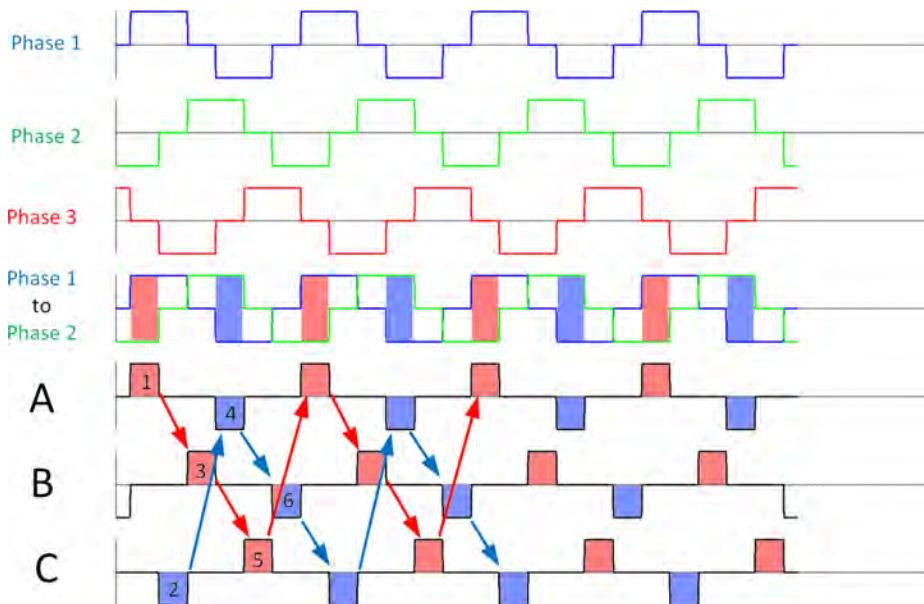


Figure 16-3 These graphs show how the three-phase output of an ESC energizes each coil group in sequence

The three outputs of the ESC each follow the same pattern but start at different times. You can see how they are distributed in the top half of [Figure 16-3](#), where the blue, green, and red lines represent the P1, P2, and P3 outputs. The bottom half isolates only the first two outputs so that the difference between them can be more easily seen. When P1 is higher, the difference is shaded red and when P2 is higher, it is shaded blue. In the line labeled A, this difference is redrawn centered around the origin.

We label this line A because it shows the voltage delivered to the A coil group. Where this is positive or red, the coil generates a north magnetic field and where it is negative or blue, it generates a south magnetic field. Lines B and C show the differences between P2 and P3 and P1 and P3, respectively, which will be the voltages in the B and C coil groups.

If the signals in [Figure 16-3](#) were used to drive the motor shown in [Figure 16-2](#), we would be just at the beginning of the timeline with coil group A just starting to pull magnet X toward it. As we follow the sequence of red lines, at position 1, magnet X will pass coil A. At position 3 it will pass coil B. After passing coil C at position 5, the sequence then begins again as our same magnet X will pass the next coil in group A. The blue lines similarly trace the route of the Y magnet.

BECs

Our ESCs also provide 5-volt power for the flight control system via a battery eliminator circuit (BEC). Instead of a receiver powering a servo over the middle wire of the servo cable, we run it backward. The ESC sends power back along this wire to the flight controller.

Any one of our ESCs can produce up to a couple of amps, which is plenty of power for our whole control system. But they all do this, and you should not just connect several sources of power together. Any small differences among the output voltages will result in the ESCs wasting current fighting amongst themselves and possibly overheating.

The traditional solution is to remove the middle (red) wire from the connectors on all but one of the ESC signal cables. [Figure 16-4](#) shows how to slip a knife under the clip that holds the pin in. Pull the pins from three of the four ESCs and cover them with heat shrink for safety as shown in the inset photo. We'll tie them back out of the way as we complete the control system wiring.

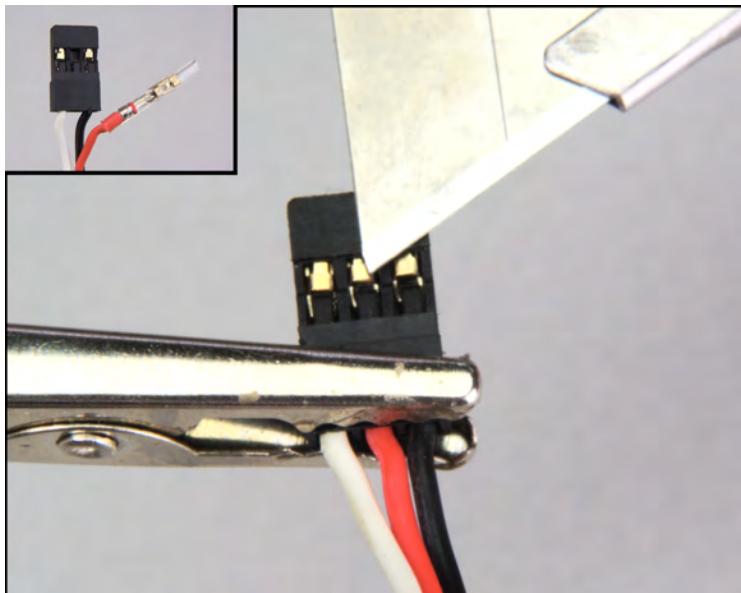


Figure 16-4 Pulling the red pin from three of the four ESCs prevents power supply trouble

Now that we have powered our motors and ESCs and understand them, we can wire them into the control system. As complex as they are, they turn out to be simple, dependable things to use in practice.

Install the Flight Controller and Radios

17

We've put in the muscle. Now it's time to add the brains (and, I suppose, the eyes, ears, and mouth, to stretch the analogy a bit). More seriously, we are going to mount a Pixhawk Lite flight controller, a GPS receiver, the RC receiver, and the telemetry radio, then wire them all up. Oh, and a safety button—another new feature very appropriate to a larger, more powerful drone. [Figure 17-1](#) shows these components wired together, but not installed.



Figure 17-1 The central Pixhawk Lite controller with three radios and a power module

I got all of these components, except for the RC radio, in a single kit, but they can also be bought separately. We'll discuss each part in more detail as we install them. When we are done, this will complete the first round of hardware installation on the big drone. We still have a couple of chapters to go before we properly understand all this new hardware, but once we do this, it will be a very capable drone. Among other things, this drone will be able to fly itself. Using GPS and a compass, it will be capable of following a series of waypoints in fully autonomous flight!

Pixhawk Lite and Shock Mount

To make sure we have power for future expansion, we will put a 32-bit Pixhawk Lite flight controller in this drone. The Lite version of this controller eliminates a few connectors that few people use in favor of being smaller, lighter, and cheaper.

When last we left our drone, we had installed only the bottom plate, with the ESC power and battery connector attached. Push the battery connector down through the large hole in the back of the bottom plate. Plug a six-pin signal cable into the power module, then plug the power module into the end of the battery connector. Push the signal cable up through the hole where the battery connector comes down. Arrange everything as neatly as you can and wire-tie the parts in place so they won't move when flying around, or especially when disconnecting a battery. The results should look something like [Figure 17-2](#).

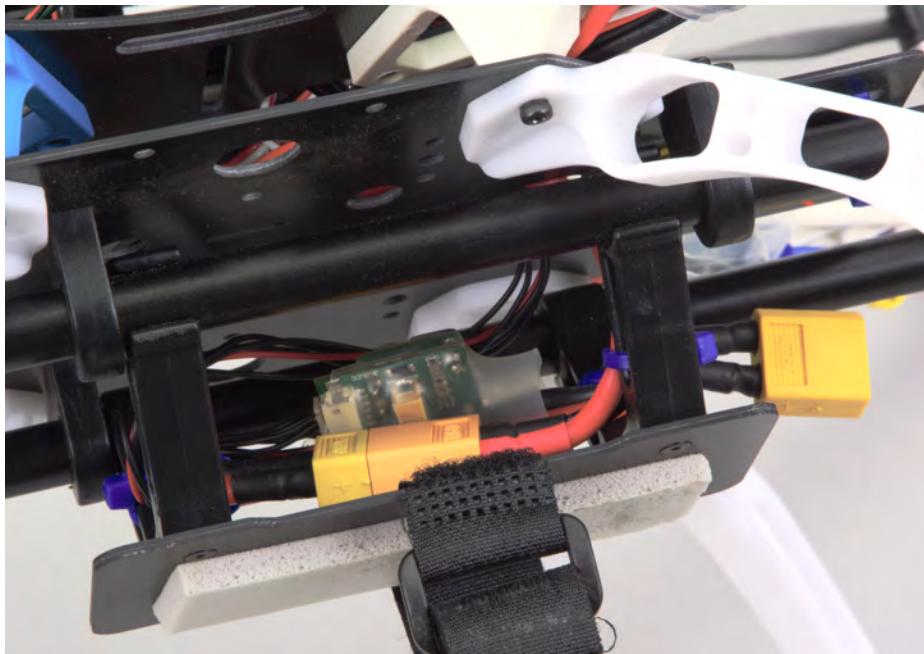


Figure 17-2 A close-up showing how the power module fits below the bottom plate

One thing included in many flight controller kits, though not the one I got, is a shock mount. This mount provides some level of vibration isolation between the controller and the motors. As we will cover in detail in [Chapter 22](#), more powerful motors can create more vibration and more sophisticated flight controllers can be more sensitive to vibration, so shock mounts become important as drones get larger.

We're ready to mount the top plate, so bring all the ESC signal cables and the signal cable from the power module up through the big hole in the middle of the top plate and screw it into place. Now, using double-stick foam tape, place the shock mount back against the hole where the ESC cables come up. You can see the correct placement in [Figure 17-3](#).

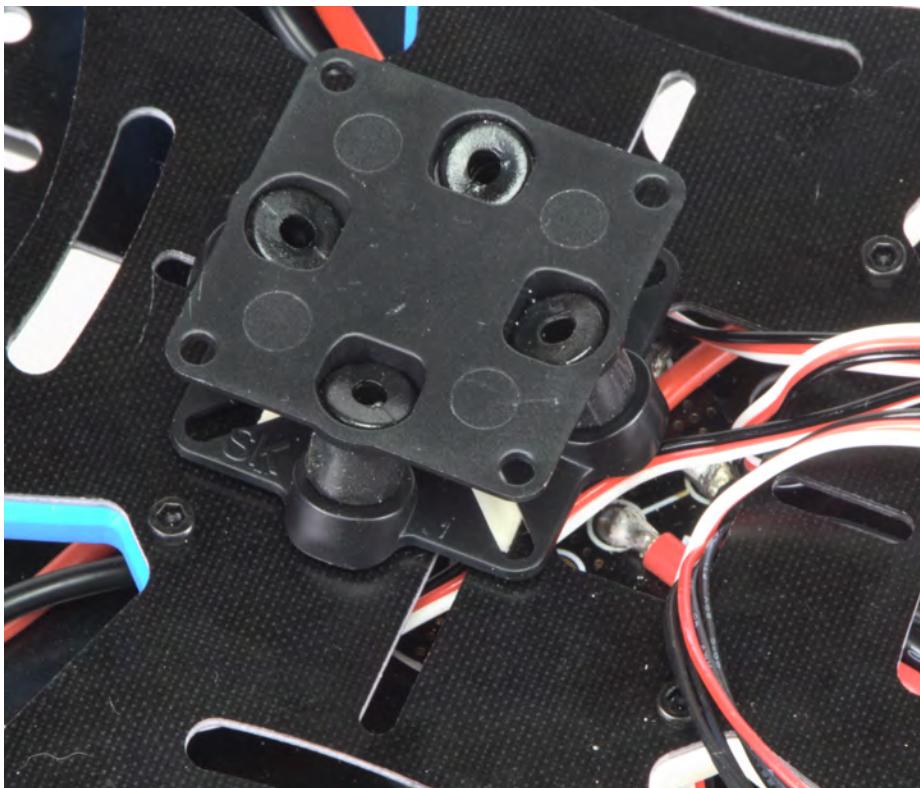


Figure 17-3 The shock mount should be placed up against the large hole in the top plate

Using more double-stick foam tape, place the controller on top of the shock mount with the pins facing to the rear. Be careful to align the controller as well as possible since this is where the drone gets its sense of up and forward, though the firmware can compensate for small errors. Now route the cable from the power module up through the center hole in the top plate and plug it into the port on the left side of the flight controller, as seen in [Figure 17-4](#).



Figure 17-4 The power module plugs into the left side of the Pixhawk Lite

RC Radios

I chose the Quanum i8 ([Figure 17-5](#)) radio control transmitter and receiver for this drone. Almost any system with five or more channels will work, but I like this set because it has some features designed specifically with drones in mind. Like with nearly all RC aircraft transmitters, the first four channels contain roll, pitch, throttle, and yaw, though not necessarily in that order.

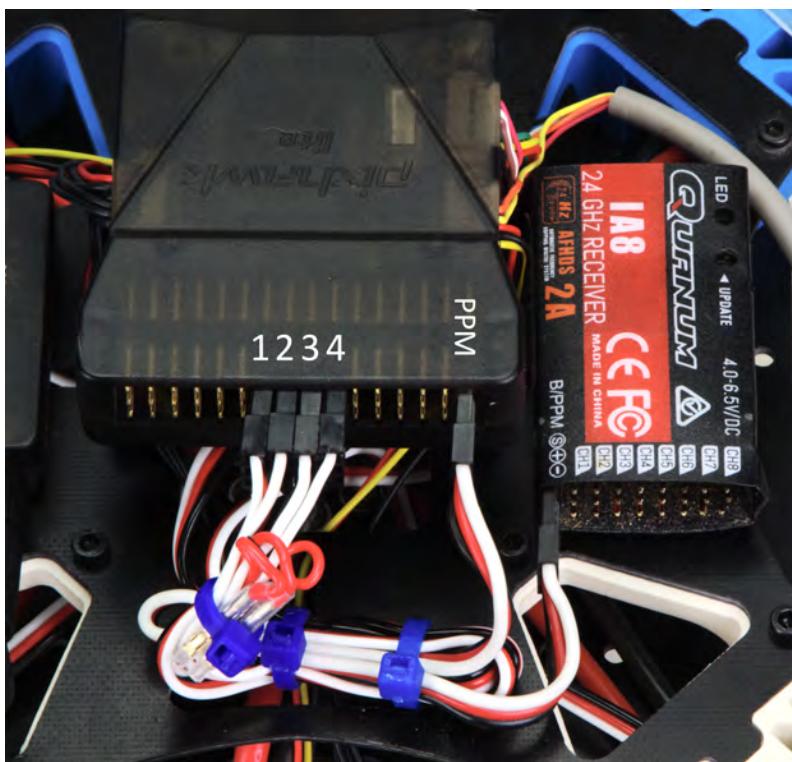


Figure 17-5 The mode lights at the top center of this Quanum i8 transmitter suit drones perfectly

Many flight control systems use channel 5 for mode control, and the Quanum i8 transmitter handles this perfectly. There is a rotary knob with LEDs that show the positions. It even comes with stickers to label the different modes. Regrettably there is no PosHold sticker, since position and hold mode works very well on this quadcopter.

The Quanum i8 receiver has eight of the standard three-pin pulse width modulation (PWM) outputs that can directly drive servos or ESCs. In another feature that supports drones well, the receiver also sends all eight channels out on a single output using combined pulse position modulation (CPPM, or just PPM). This allows us to connect the receiver to the flight controller with a single servo cable, which also serves to provide power to the receiver.

Place the receiver on the top plate, to the right of the controller, using double-stick foam tape. The pins should face to the rear, the same as the controller. Run a three-pin servo cable from the leftmost port of the receiver to the rightmost port on the controller as seen in [Figure 17-6](#). You can also plug the ESC signal cables into the controller at this point, but be careful: the outputs on the controller start with channel 1 in the middle, not on one end as you might expect. [Figure 17-7](#) shows how the motors are numbered.



[Figure 17-6](#) The Pixhawk Lite outputs don't start where you might expect them to

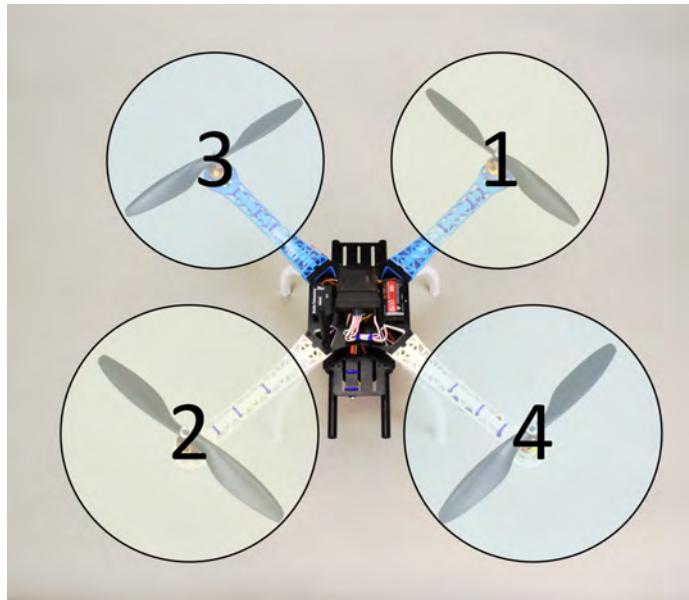


Figure 17-7 Motor numbering for the Pixhawk flight controller

On the other end of the RC receiver, two antennas come out. With two antennas, one will hopefully continue getting a good signal when the other gets interference. I routed these down through the slot in the front of the top plate and then replaced the heat shrink on the forward ESCs to include them. This keeps them out of the way, as seen in [Figure 17-8](#).

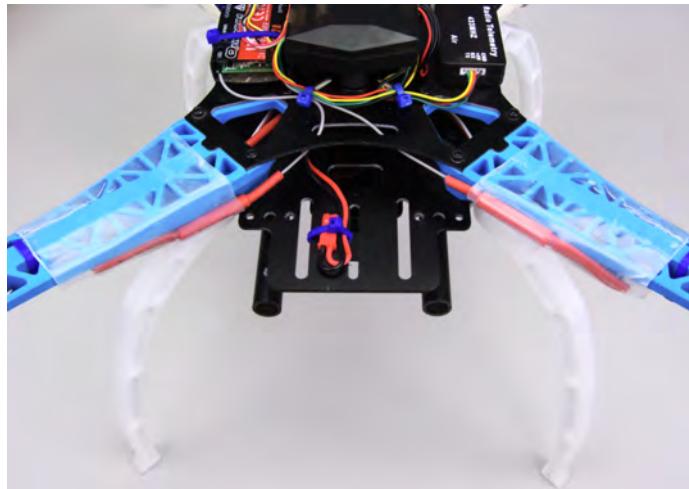


Figure 17-8 Putting the RC antennas in the heat shrink keeps them safe

Telemetry Radio

For telemetry, we will use a 3DR 433 MHz radio set. These are widely available, fairly cheap, and easy to interface to an APM- or PX4-based flight controller. Note that use of the 433 MHz band in the US requires a ham radio license.

We can actually do much more than just telemetry (i.e., real-time monitoring of flight information) with this radio. Basically, this link can do anything a USB link can do, though it's not as fast. This includes setting turning parameters, creating autonomous flight plans, and even sending commands like "start the programmed mission," or "return to land."

Turn On All Your Radios

Even though for some flights you may think you don't need telemetry or video or whatever, turn on all the radios before you fly. I have had lost radio connection with my drone and used a smartphone to land it via telemetry. I have even used radio signal strength as a tool to find a crashed and lost drone. Your phone or computer

might have a log of GPS positions right up to a crash if you're logging telemetry data. You might save your drone by flying by video link if your view gets obstructed during a line-of-sight flight. You never know which link will save you, so turn them all on!

The telemetry radio comes with several cables. The one attached to it includes a split to send the telemetry data to the on-screen display (OSD) that will add information to the video downlink. We don't need this now, but we might later. One end of each cable will fit the telemetry radio. Find one where the other end fits the telemetry port on the side of the flight controller. Now check the cable wiring very carefully to verify that it looks like [Figure 17-9](#).

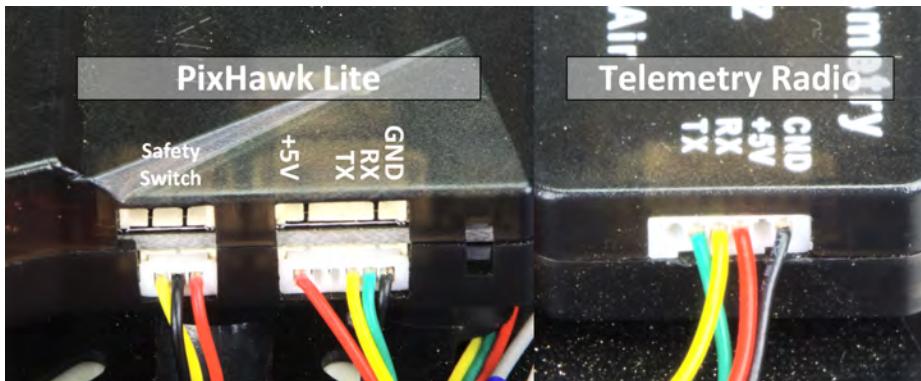


Figure 17-9 Verify the cable wiring before powering up the system

Safety Switch

To complete this chapter, we will install the safety switch. When preparing for flight, you must press and hold this switch for several seconds before the motors can be armed. This ensures that no single action on the RC transmitter or via telemetry can accidentally arm them. As drones get larger and more powerful, it becomes increasingly important to guard against spinning props. In addition to the safety switch, we must always take the time to remove the props when not required for testing and make sure the area is clear before allowing a prop to spin.

Wiring the safety switch just means wire-tying the switch itself to the bottom plate as shown in [Figure 17-10](#), then routing the cable up through both plates and plugging it into its port, as seen in [Figure 17-9](#).

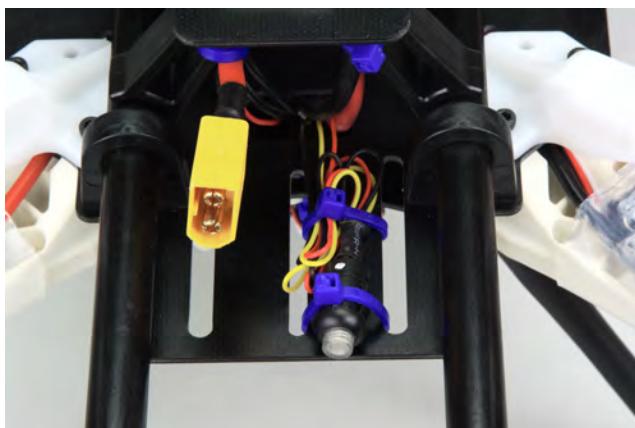


Figure 17-10 Secure the safety switch to the bottom plate with wire ties

Testing

Let's stop and test what we've just installed before going any further. First, plug the controller into your computer via USB. You should get blinking lights on both the telemetry radio and the RC radio. (And no smoke!) If you don't see a good response, immediately unplug the system. You might save a component that has its power wired backward if you're quick!

Next try powering the system from a battery. Unplug the USB and plug in a charged, 3S lithium polymer (LiPo) battery pack and you should get the same response you got previously, though you might also get some beeps from the ESCs since they are getting power now too. If the USB test worked and this one didn't, check the cable between the power module and the controller.

Leaving the quadcopter powered up, plug the telemetry ground unit into a USB port on your computer. The light inside should come on and within a few seconds this light and

the one in the telemetry air unit should both stay on constantly, meaning that they have established a link. You can now start up your favorite MAVLink ground station software and connect to your drone.

Personally, I don't have an overall favorite. I use a combination of Mission Planner, QGroundControl, APM Planner (Mac), and finally Tower on my smartphone. QGroundControl has the slickest interface. It connects automatically in most cases, which really helps, and it has great calibration screens for both accelerometer and compass. And it runs on both Mac and Windows.

MissionPlanner only runs on Windows, but it has a built-in log viewer that I'm used to and puts all the PID tuning parameters on one very useful screen. I also tend to use Mission-Planner for, well, planning missions (that is, creating autonomous flight plans). QGroundControl also does a good job of this so my choice is mostly habit.

I use APM Planner, which is Mac only, just for its *compassmot*. It produces a nice graph while it's running that lets you see how good the data is. I use the Tower application when running on a smart phone. Using a small USB adapter, it will connect to the telemetry dongle. I always keep this turned on when flying, though my phone often sits in my pocket, just giving audio feedback on modes, battery state, and altitude.

At the moment, the point is only that the radio calibration screen in Mission Planner makes a nice radio test. I recommend looking at the connection drop-down before and after connecting the telemetry receiver so you can see which port got added. Select that port and a baud rate of 57,600 and it should connect up. Once it finishes reading all the parameters you can switch to the Radio Calibration screen in the Initial Setup category and the graphics should respond to the controls on your RC transmitter (see [Figure 17-11](#)).



Figure 17-11 Mission Planner's Radio Calibration screen is a good test of the RC radio

This completes the installation of everything we are going to wire up on the top plate, so go around the system and wire-tie everything down. I found that the unused mounting holes on the corners of the shock mount made good tie points. Shake your drone around a little. Does anything move too much? If so, reroute or use more wire ties. Their weight won't matter this time.

Two radios installed so far, and one more to go. In the next chapter we will install a GPS receiver and I'll explain a bit about how it works.

GPS

18

The Global Positioning System (GPS) allows our drones to find their three-dimensional position by listening to the signals from several satellites. The receivers we have today are small, cheap, fast, and accurate. The first GPS receiver I used was too heavy for one person to lift. Thankfully, the receivers and the satellites have continued to evolve. [Figure 18-1](#) shows one of the latest generation of satellites.



Figure 18-1 A GPS series IIIa satellite, courtesy NASA

GPS found its way into aircraft and survey crews years ago, but it took the cell phone revolution to make the receivers as small and cheap as the one we will use. The GPS receiver we will use for our big drone, combined with a three-axis magnetometer that we will discuss in the next chapter, fits in a little round puck about two inches across.

History

The Global Positioning System was conceived in the US in 1973, and the first satellite was launched in 1978. By the mid-1980s, when I was working on prototype GPS systems for the US Air Force and Navy, there were still only a few working satellites. The receivers we used weighed over 50 pounds and cost \$150,000 each. And each one had an external cesium clock, adding a few thousand to the total.

Then, in 1983, the Russians shot down Korean Airlines flight 007 when it strayed accidentally into their airspace. In response, Ronald Reagan announced that the full accuracy of the GPS system would be made available to civilians. Previously, the high-accuracy part had been encrypted and only a lower-accuracy subsystem was available to the public.

Today the Russians run their own satellite positioning system, called GLONASS (GLobal NAVigation Satellite System). And the latest models of GPS receivers for drones can use GLONASS and the American GPS satellites at the same time. This means that there are more satellites available, which helps with the reliability and accuracy of the results, as we shall see in the next section.

Theory

Satellite navigation depends on the simple idea that time equals distance. If we know the time a radio signal takes to travel from a satellite to us, and we know it goes at the speed of light, then we can compute the distance to the satellite. So, GPS satellites keep broadcasting messages like “I sent this message at exactly 03:22:17.64836125 GMT.” Then we just have to know the exact time when we receive such a message. Knowing the current time very accurately is part of the trick.

By noting exactly when we receive each message by our internal clock and taking the difference between this and the start time contained in the message, we can compute an estimated time of flight. Convert these times to distances and include the known positions of each satellite, and we can begin to see the geometry problem.

To make it really easy to see, let’s just consider two dimensions and three satellites. In the *Theory* frame of [Figure 18-2](#), the circle around each satellite represents our computed distance to it. In other words, if we only knew about one satellite, then the circle around it would represent all the places we might be. Add a second satellite, and the two circles (usually) intersect in two points. Add a third, and there is only one point that we can be at —our position!

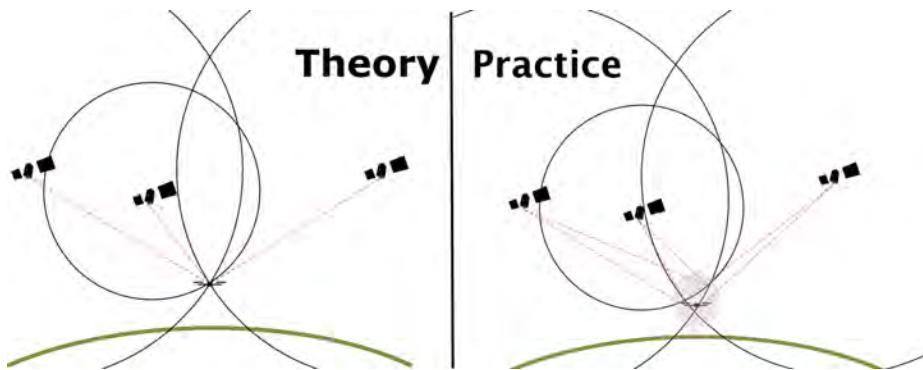


Figure 18-2 To make GPS work in practice we use Kalman filters

Since we put three variables into our equations, we should be able to determine three output quantities, but in our flat world we have only two. But nothing is lost here! The solution to these equations also tells us how far off our local time is from the shared time of the satellites. This extremely accurate source of timing information gets used in many applications that don't need position at all, like keeping the clocks on cell phone towers synchronized.

Of course, the real world does not really look like a problem in a geometry textbook. The circles don't actually line up, as shown in the *Practice* frame of Figure 18-2. This means we get to use yet more interesting math! Instead of just looking at one set of measurements, we take the average of lots of measurements over time. And not just a simple average, either—a Kalman filter or other equally interesting data-smoothing technique.

No GPS Without Einstein

If you think relativity is only for stars and cosmic rays, well, think again. Einstein's theory of relativity predicts that clocks on a fast-moving satellite will run slower than clocks on Earth—which they

do. If this slowing of the clocks on the satellites were not taken into account, positions calculated by GPS could drift as much as 10 kilometers per day.

The Kalman filter was developed by Rudolf E. Kálmán, among others, around 1960 and can be found in applications from aerospace to robotics to seismology. Like most algorithms that have a name, shelves full of papers have been written on the subject, but here's a simple outline of how it works.

First, we need a model that can predict the next value. In one simple case, if we know our altitude and our vertical speed, we can compute our next altitude. Then we take some measurements: perhaps a barometer reading and the sum of our accelerometer values. Now we take a weighted average of the altitudes from our three sources (the prediction and two measurements). The trick is in the weights, of course. We change the weights for

each altitude value in order to make the total error—the difference between our final output and each value—as small as possible.

To get the most accurate positions, variations in the ionosphere must be taken into account. Radio signals travel at the speed of light, as Einstein tells us, but the speed of light varies a little depending on what the light is passing through. The ionosphere, a section of the upper atmosphere between 75 and 1,000 kilometers above the Earth, slows down the signals from the GPS satellites just a bit, and it varies in thickness.

In order to compensate for the varying thickness of the ionosphere, GPS satellites transmit on two different frequency bands (really, up to five now). Since the amount that the ionosphere slows down the signals also depends on the frequency of the signal, a receiver can measure the difference in the time of arrival of the same message in each band to determine how much the ionosphere affects signals from a given satellite.

Sadly, most of the GPS receivers we use in drones, like those in cell phones and many consumer devices, only receive on one of the two primary GPS bands and so can't take advantage of this correction. Because of this, we need to pay attention to space weather. Yes, space has weather!

Things like solar storms can stir up the ionosphere and make it thicker. A dual-channel receiver can correct for this, but our single-channel receivers cannot. There will be days when you are tracking lots of satellites and the geometry (or HDOP, discussed in the next section), looks good, but your positions still seem awful. This could be due to a space storm, and there are websites that will let you predict and track their effect on GPS signal quality.

HDOP

HDOP stands for *horizontal dilution of precision*. This measures the degree to which a bad arrangement of satellites, or constellation, degrades the accuracy of your position. With a theoretical accuracy of 1 meter but an HDOP of 5, you could be 5 meters away from where you think you are.

In the first frame of [Figure 18-3](#), a small error in the distance to one satellite results in an error of similar size in the calculated position. In the second frame, the two satellites are close together in the sky. In this situation, a small error in the distance to one satellite results in an error of several times that size in the calculated position. You can see that the red arrow representing the position error is now about four times as long as the blue arrow representing the original satellite position error.

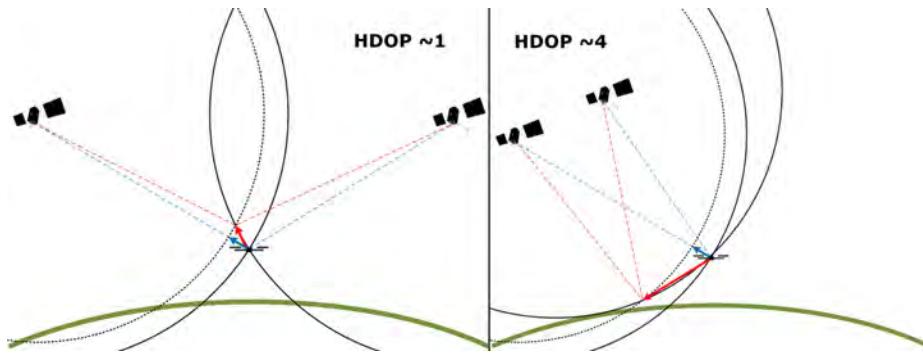


Figure 18-3 A high HDOP means that position errors get enlarged

Practically, we like to fly with an HDOP of 2 or less. You can set a minimum HDOP and number of satellites required in the flight control system. If these conditions aren't met, then you can't arm the motors in a mode that requires GPS. However, beware: you can still switch into a GPS-dependent mode, even when the HDOP is high, when you are already flying.

Installation

With all the complexities in the theory of GPS operation, the complexities of its installation may seem a bit trivial—but complexities they are, nonetheless. The GPS unit I got was wired for a PX4 or APM flight controller, and the Pixhawk Lite has unique combined GPS/telemetry ports that require some special wiring. We will return to this if we decide to add an on-screen display (OSD) unit that will need the second telemetry port. For now, we just have to wire the GPS correctly.

Wiring generally goes better when the thing we are wiring sits still, so let's install the GPS mast before doing the wiring. You will need four #4-32 × 3/8" bolts with nuts. Bolt the base in place as shown in [Figure 18-4](#), lock the mast into the base and the top on the mast with the set screws, then stick the GPS puck on top with the provided double-stick foam tape. Now we are ready for the wiring.

My GPS actually came with an extra empty plug for rewiring, but if you're careful you can pull the pins out and reinsert them without damaging the plug. [Figure 18-5](#) shows how to slide a knife blade down into the retaining clip and release the pins.

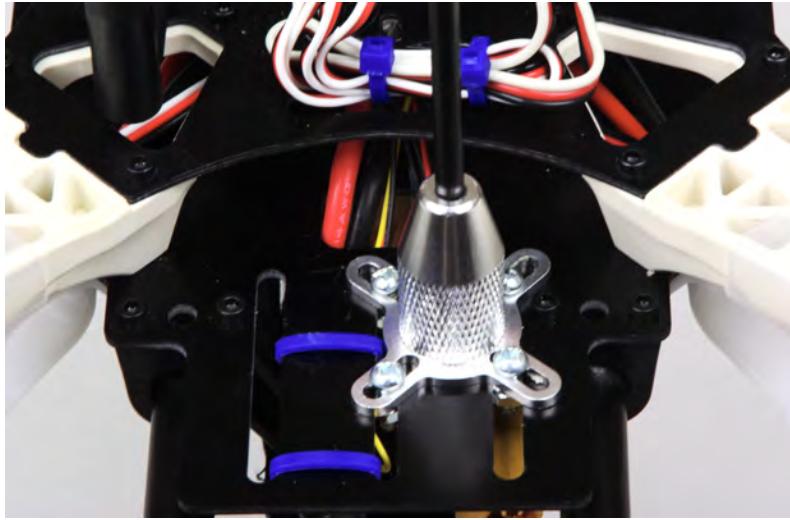


Figure 18-4 The GPS mast bolts solidly to the bottom plate of the frame

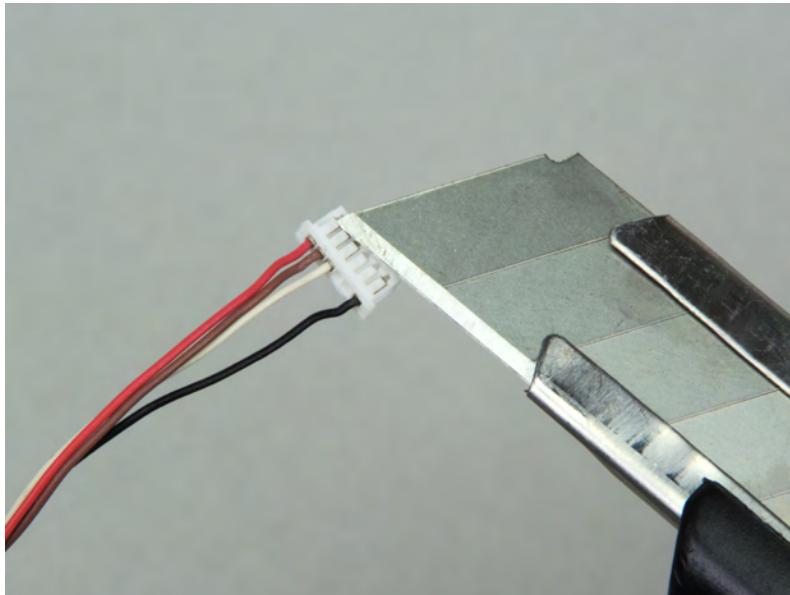


Figure 18-5 With care, you can release the pins without damaging the plug

Red almost always means positive—in this case, 5 volts—so looking at the pinout of the Pixhawk in [Figure 18-6](#) you can see that the red and black wires must be swapped. I guessed at which wires were transmit (TX) and receive (RX), but I guessed wrong. When I powered up my flight controller, I got blinking lights on the GPS, but the controller didn’t recognize it. You can check by connecting to your ground station software. If the GPS

works, you will see some status like “GPS: No Fix.” If you don’t get any kind of GPS status, try swapping the TX and RX lines.

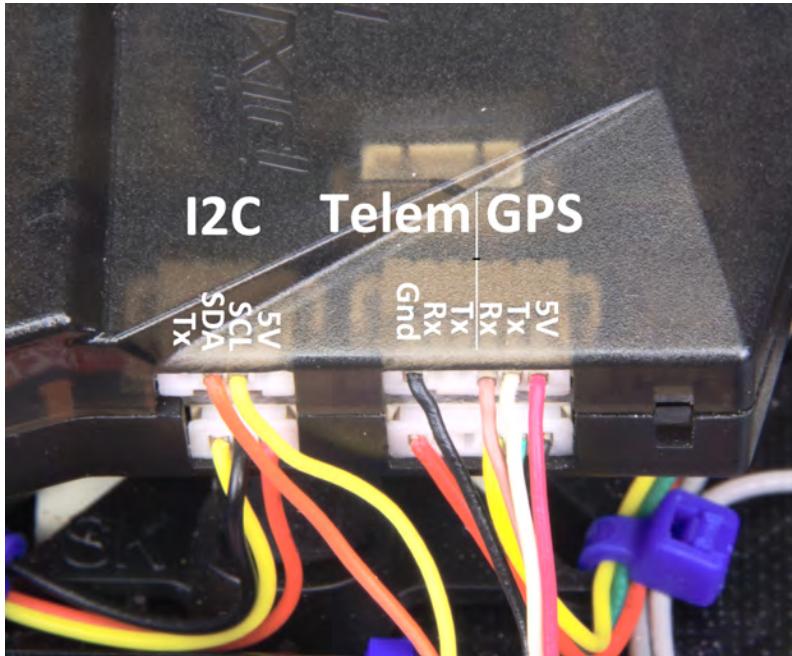


Figure 18-6 The Pixhawk Lite uses shared GPS and telemetry ports

Now you should be able to see the GPS status in your ground station software. To get a fix, the drone will have to be outdoors or at least near a window. If you power the drone from a battery, you can set it outdoors, connect via radio telemetry, and watch on the computer as your GPS acquires satellites and the HDOP improves.

In the next chapter, we will discuss magnetometers, but guess what? You just finished installing one. GPS units built for use with PX4 systems also contain a three-axis magnetometer.

Magnetometer

19

Back in [Chapter 9](#), we talked about adding a magnetometer to the mix to prevent drift in the yaw axis. When you start doing navigation, knowing your absolute direction becomes more important. You may know where you are and where you want to be, but if you don't know which way you are pointing, you can't get there.

Humans have been using magnetic sensing devices for centuries, and animals perhaps for millennia. We've come a long way from floating bits of iron, both in our apparatus and in our understanding. Humans have learned, often by crashing ships onto rocks, about deviation and variation, and about the change in variation over time. The magnetic variation of my local airport has changed over half a degree during my lifetime, as noted on the aeronautical charts.

Practically, in the drone world, the hardware has become pretty simple. A single chip with a convenient I2C interface gives us a three-axis measurement of the local magnetic field. Operationally, compasses remain, as they have always been, a bit of trouble. They must be calibrated (in our case by doing the "magnetometer dance"—just wait!), and sometimes we must compensate for changes in the fields due to the current of the motors.

We have already installed a magnetometer along with the GPS unit, so this chapter will be a discussion of how it works and how best to use it.

The Earth's Magnetic Field

We've all seen the basic picture of how the magnetic field lines flow through the Earth ([Figure 19-1](#)). And if you have played much with magnets, you know that they like to line up a certain way, north to south and south to north. So, if we let a magnet move freely without any other magnets nearby, it will line up with the huge but weak magnet that is the Earth. Humans have long known how to float a lodestone (a naturally magnetic rock), or later iron pins, so that they would point toward magnetic north.

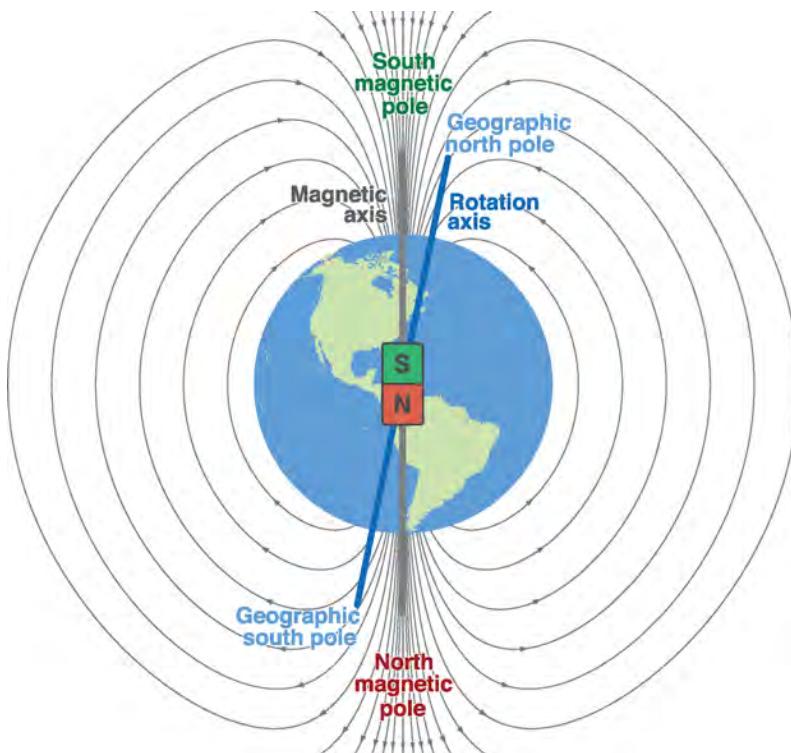


Figure 19-1 The Earth's magnetic field is not quite aligned with the geographic north pole

Notice the term *magnetic* pole. The first thing you need to know in order to make use of a compass for navigation is that the magnetic north pole is currently several hundred kilometers from the geographic north pole, the axis that the Earth rotates around. And the magnetic north pole moves over 50 kilometers per year, lately heading northward out of Canada. Geologists tell us that the Earth's magnetic field has flipped north for south several times in its history, though we don't expect this to affect navigation any time soon.

We, or perhaps our computers on our behalf, must pay attention to the difference between magnetic and geographic north when trying to fly a magnetic course and track ourselves on a geographically oriented map. To get this right requires an understanding of not just the global but also the local differences between maps and compass readings.

Declination and Deviation

Declination, more commonly called *variation*, measures the local difference between magnetic and true north. This includes both the difference because of the different positions of the poles and the local differences caused by things like differences in the thickness of the Earth's crust or local deposits of iron ore. Humans have learned about declination the hard way, often paying with the lives of mariners.

Christopher Columbus noted that the difference between magnetic and true north changed considerably as he crossed the Atlantic in the fifteenth century. By the seventeenth century, it was understood that this difference also changes over time, and that there will be purely local differences. But it wasn't until the nineteenth century that accurate and useful maps of the resulting compass variation were available.

Deviation, as opposed to declination or variation, describes our attempt to measure the Earth's magnetic field, not the field itself. It has long been known, for example, that any iron in a ship can throw a compass off—a real problem once the hulls began to be made of it! In the 1800s John Gray and then Lord Kelvin came up with systems to correct for this effect. Lord Kelvin's system places two large balls of iron, one on either side of the ship's binnacle or compass stand, which can be moved to compensate for the error. Traditionally painted red and green ([Figure 19-2](#)), these are commonly known as Kelvin's balls.



Figure 19-2 The red and green balls on this ship's binnacle correct for the presence of iron in the ship

In modern aircraft, we still carry a basic magnetic compass, and it will also include a set of small magnets placed around it to correct for the presence of iron in the structure of the plane. But this correction is never perfect, so aircraft compasses also come with a so-called *compass card*. This small card, placed next to the compass, lists the remaining error at different angles so the pilot can correct that last degree or two. When you're traveling at 500 miles an hour, a small angle can become a big distance in a hurry!

Magnetoresistance

Of course, our drones don't carry an iron needle floating in oil. They carry solid-state devices using a property known as *magnetoresistance*. Scientists and engineers have discovered many different mechanisms by which the electrical resistance of a material can be made to vary with the magnetic field around it. As they found ever larger effects they had to make up ever sillier names, like "giant magnetoresistance," "colossal magnetoresistance," and even "extraordinary magnetoresistance." The magnetometer chips in our drones generally use *anisotropic magnetoresistance* (AMR), which means that the change in resistance varies with the direction of the magnetic field—a useful property when you are trying to find direction.

Each of the named types of magnetoresistance may, in fact, be produced by a combination of multiple underlying physical mechanisms. Without going into all the details, one outcome of the way AMR works also has a fun name that you may hear used to describe these systems: barber-pole sensors. AMR sensors start with a thin strip of permalloy, a nickel-iron alloy. But by itself this will only tell you, in effect, how close you are to pointing to north, not whether you are a bit east or west. By placing a series of aluminum strips across the permalloy at 45-degree angles, the response to the left and right become different, and the resulting chip looks something like a barber pole, as seen in [Figure 19-3](#).

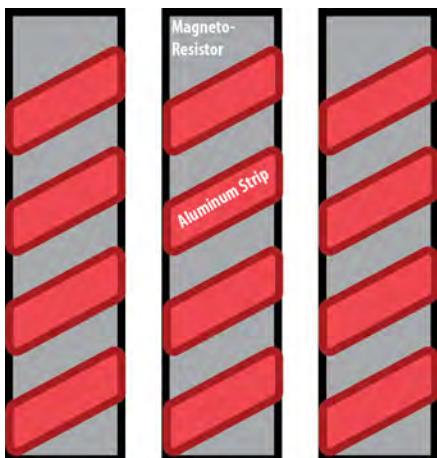


Figure 19-3 The aluminum strips help determine left from right and make the structure look like a barber pole

Use in Drones

Putting a modern magnetometer into our drone control system is easy. Choose a chip like the HMC5883L three-axis magnetometer that shares the puck with our GPS receiver. Now connect power and the I2C interface, and you're done. But building compasses was always the easy part. Learning to use them is the problem.

All of the things we've just talked about show up in our flight software and calibration processes. When you're setting up a drone using ground control station software, the system can read your position from the GPS receiver and, using the Internet, find your local magnetic declination. Alternatively, you can enter the value by hand.

Next comes what we call compass calibration. Just as they still swing whole ships and planes around to determine the deviation of their compasses, we get to swing our drone around in what is known as "the compass dance." You start the process by choosing the compass calibration option in your ground control station software, then follow the instructions as you point the drone in different directions and twirl around. [Figure 19-4](#) shows a complete calibration using QGroundControl.

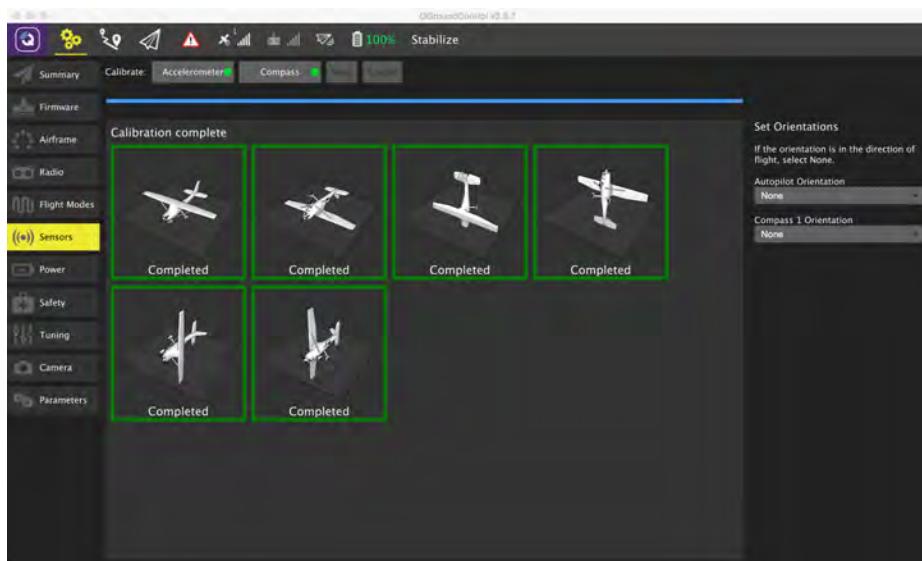


Figure 19-4 This compass calibration screen guides you as you do the compass dance

This process compensates for the presence of lots of strong permanent magnets in the motors—one type of deviation in our compass reading. But we also have electric magnets in those motors and high currents running through our power system, which can create magnetic fields that change during flight. Our large drone design, like many, places the compass and GPS receiver on a mast, several inches above the power system and other radios. This minimizes the interference with the compass, in our case to acceptable levels

(as we shall see). To test this and compensate for it if necessary, we can perform the [compassmot](#) test.

To perform a [compassmot](#), you must first remove all your props (we'll talk about choosing and installing props in the next chapter) and put them back on so that they will push down instead of up. Either swap pairs or move them each around by one arm. Then, in a safe location with no other people too close, start the [compassmot](#) calibration, which will instruct you to move the throttle up and down. The system will monitor the current being used and the resulting magnetic interference and compute a correction factor for later use. [Figure 19-5](#) shows the results I got: less than 10 percent interference. This low result is not surprising since we have a compass up on a mast.

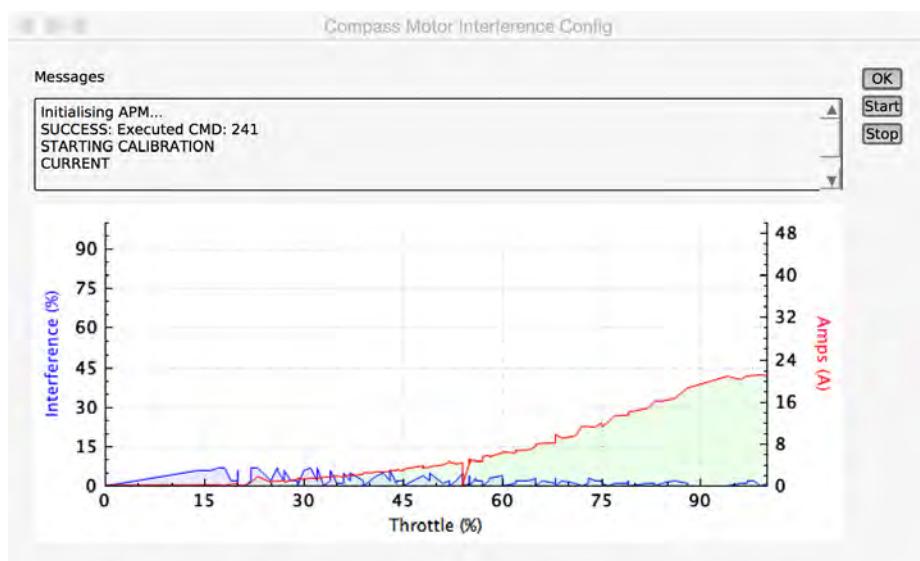


Figure 19-5 These compassmot results show insignificant interference

Given that I have demonstrated the very low level of interference with this design, I would actually not recommend running this test on this drone unless you have reason to suspect compass problems. First of all, it is dangerous to run these props up to full speed anywhere near people, including yourself. There is also the hassle of and room for error when switching the props around. And finally, it could happen that a bad test result gives a calibration that flies significantly worse than not having done this at all. I've done the test, so you don't have to. This is engineering at work.

We have hardly touched on the fascinating history of the compass, but hopefully you have learned enough to know why you must do that funny dance when setting up your drones. And, of course, you should repeat it if you change the hardware, especially when adding things with more motors, like a gimbal, or if you change locations by more than a few miles. But enough theory for a while. We're almost ready to fly!

Flying the S500

20

We have put all the pieces in place. We've done some testing. Hopefully, you have even gained some understanding of how it all works. But there comes a time in the development of any aircraft when you just have to take off and see how it goes.

Next, we will install the props and talk for a moment about prop choice. At this point the drone will look something like [Figure 20-1](#). Then we will strap on a battery and backup power meter.



Figure 20-1 The completed large drone stands ready to fly

The systems will demand some calibration before being willing to fly. We did the magnetometer in the previous chapter, but we must still do the radios and accelerometers. You can choose which flight modes you want for which switch positions, and then, of course, there's tuning.

If you have built something close to the design shown, then you can simply load the tuning parameters I use from www.makedronesbook.com/projects/S500. If you have made any important changes, like frame size or motor specs, then you might be better off starting with the default settings and tuning it yourself.

Install the Props

For initial flight testing, we'll use the nylon 1045 props that commonly come with motors in this size range. You can get them from multiple sources for a reasonable price, and they work quite well. These props don't come perfectly balanced, however, so I strongly advise following the prop balancing procedure outlined in [Chapter 22](#).



Don't Use Carbon Fiber Props for Testing

Carbon fiber props are lighter, thinner, stronger, more efficient, and, perhaps most important, really cool looking! So why not use them on your shiny new drone? They also spin a bit faster and are sharp as razors. Until you have confidence in your drone and your ability to fly it, make the world a little safer and use plastic props.

The EMax motors we're using can accommodate traditional props with a hole all the way through, or DJI-style self-tightening props that screw on directly. The traditional props we're using, shown in [Figure 20-2](#), require that you choose the right adapter ring from the provided set. Find the smallest one that fits over the motor shaft. Now, go balance them!

When installing props on the motors for flight, the adapter ring side goes down. [Figure 20-3](#) shows the proper direction, clockwise or counterclockwise, for each prop. If in doubt which way a prop goes, just spin it around a bit and visualize which way pushes the air down. The different color prop nuts match the different directions, black for clockwise and silver for counterclockwise.

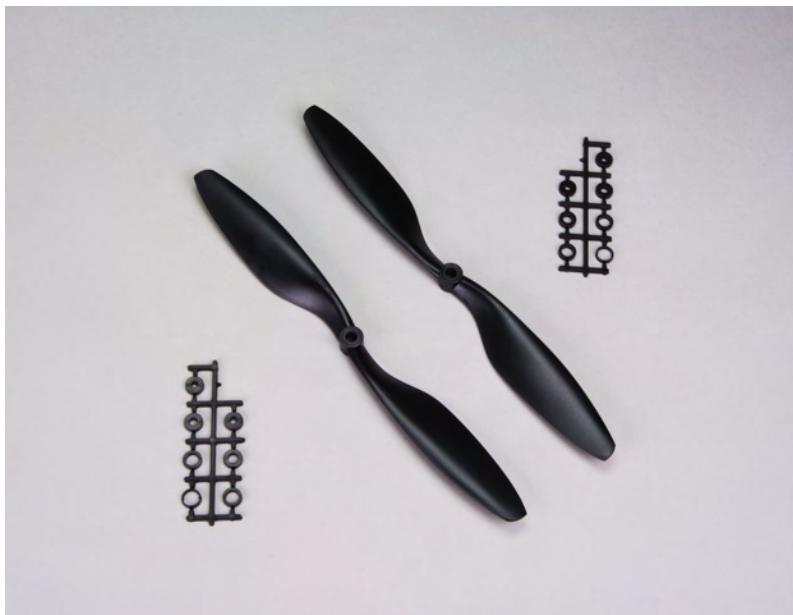


Figure 20-2 These generic 1045 plastic props require adapters to fit on motor shafts

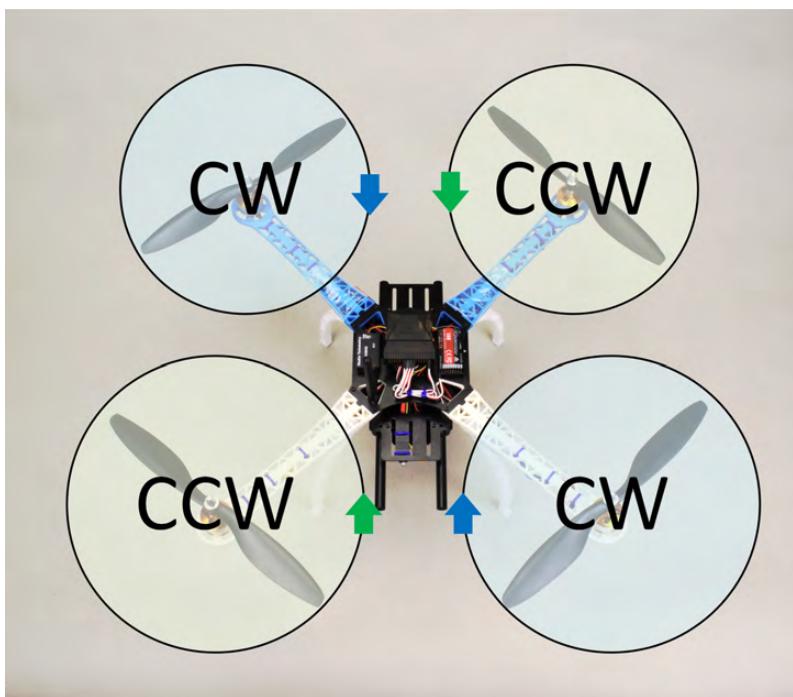


Figure 20-3 Be sure your props spin in the right direction before flight

You want the prop nuts on pretty tight, of course. It's obviously bad if one comes off in flight! But the thread direction for each motor causes the nuts to tighten rather than loosen with the force of the motor. And if you put too much force on the prop hub, it can crack. I tighten mine using no more than an inch or so of wrench length. If you pushed hard at the end of a six-inch wrench, you would crack them easily.

Connect the Battery and Voltage Monitor

Depending on what other payload you strap to it, this drone could carry a very large battery. I've done my flight testing with 3S, 2,700 mAh batteries ([Figure 20-4](#)), mostly because I already had several. This will be our first time to step up from 1S or single-cell batteries to 3S batteries that have three cells in series.



Figure 20-4 These 2,700 mAh, 25C battery packs worked well for flight testing

According to a chart from the manufacturer, the EMax 2216 motors with 1045 props and 3S batteries can draw nearly 10 amps. Rounding up to 10, times 4 motors, gives us 40 amps total current. If you divide this by the 2.7 Ah rating of our battery, you get just less than 15C. My packs are rated for 25C, so I have been fine. If you want to try anything smaller, do the math and make sure your battery can provide at least 40 amps, hopefully with a bit of margin. Anything with a larger mAh or C rating will be fine as long as you don't run it down too far—which is why you'll want a voltage monitor.

For something less than five bucks, you can buy a small digital voltage meter ([Figure 20-5](#)) with an alarm that plugs directly into the balance plug—the small white one—from the battery. This way it can monitor each cell individually and warn if any one goes below a set limit.

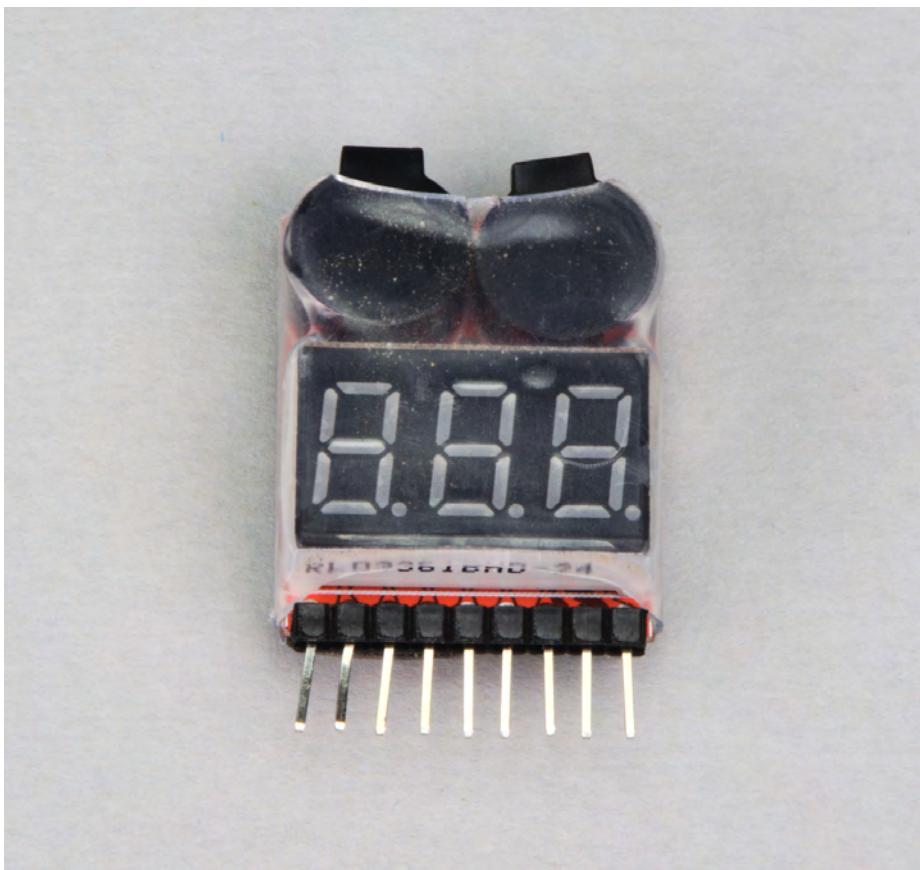


Figure 20-5 A cheap voltage monitor could save your drone

Attach the battery monitor to the battery with the black or ground wire on the left side (check your unit to make sure this is correct, though; it should have some kind of instructions on the back side). When properly attached, the LEDs will display first the total voltage, then the voltage of each cell. If any cell goes below the limit an amazingly loud alarm sounds—loud enough that you can hear it over a powerful drone at a distance.

The limit can be changed using a small button on the top of the unit. I set mine at 3.7 volts. This gives me enough flight time and leaves the battery at right about the storage voltage of 3.75V (the voltage dips a bit under load, so the final resting voltage will be a bit higher).

I installed my voltage monitor by putting a wire tie around one of the mounts that hold the battery to the tubes. Having a bit of play left makes it easier to get the balance plug connected to the unit. You can see the results in [Figure 20-6](#).

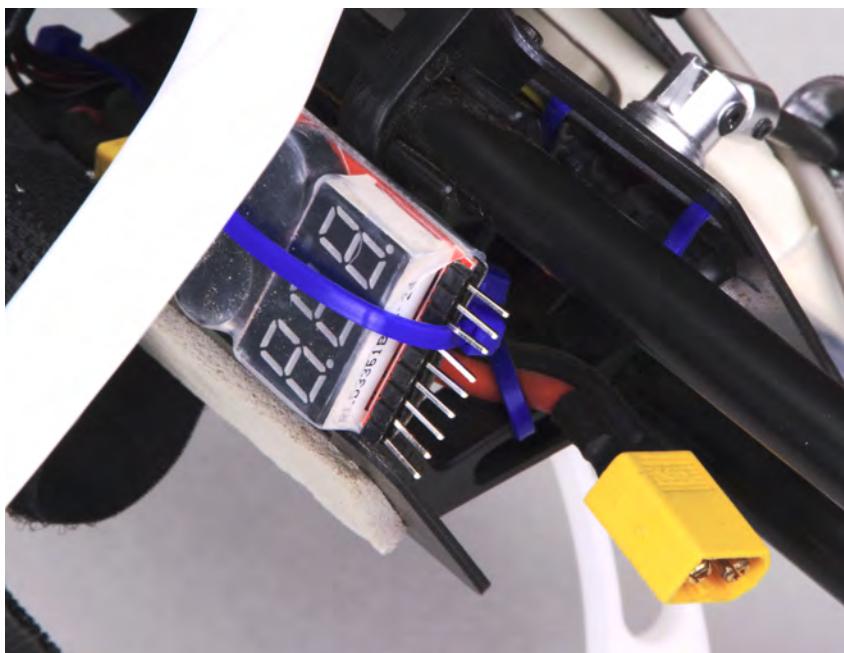


Figure 20-6 The voltage monitor must be installed close to the battery connector

The power adapter we installed earlier reports voltage and current to the flight controller, and we can also use telemetry to connect to a smartphone to give us low voltage warnings. It's a pretty cool system, but there are lots of ways it can go wrong. If you always plug in that backup voltage monitor, your \$5 investment may well save a \$50 battery!

Radio and Accelerometer Calibration

The APM flight stack won't let you arm the motors until you have finished certain setup tasks. We did the compass calibration previously, while discussing how magnetometers work. You must also calibrate the radio and accelerometers.

In the days before it was practical to put computers in our model aircraft, advanced RC pilots would use complex transmitters that could save settings for multiple models. Today, the computer in the model can be set to respond correctly to a standard transmitter. All it needs to know are the minimum and maximum values that your transmitter puts out on each channel, since these vary a bit from one transmitter to another. Knowing exactly what "all the way down" translates to matters for actions like arming. I like QGroundControl's calibration screens the best, so [Figure 20-7](#) shows that radio calibration screen.

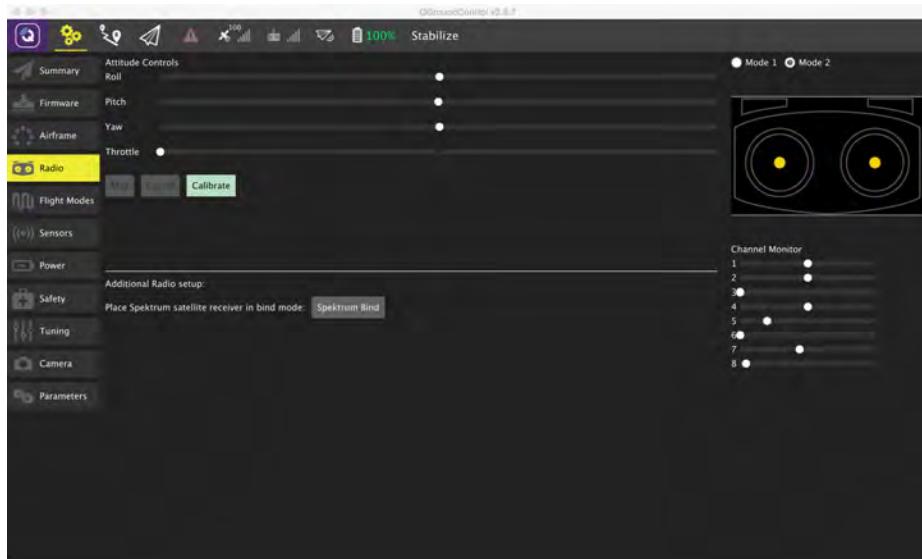


Figure 20-7 Radio calibration tells the flight controller about your transmitter

If calibrating the compass is a dance, then perhaps calibrating the accelerometers is the rehearsal. We go through the same basic positions, but instead of twirling around, we just hold still in each one. **Figure 20-8** shows this calibration screen in QGroundControl.

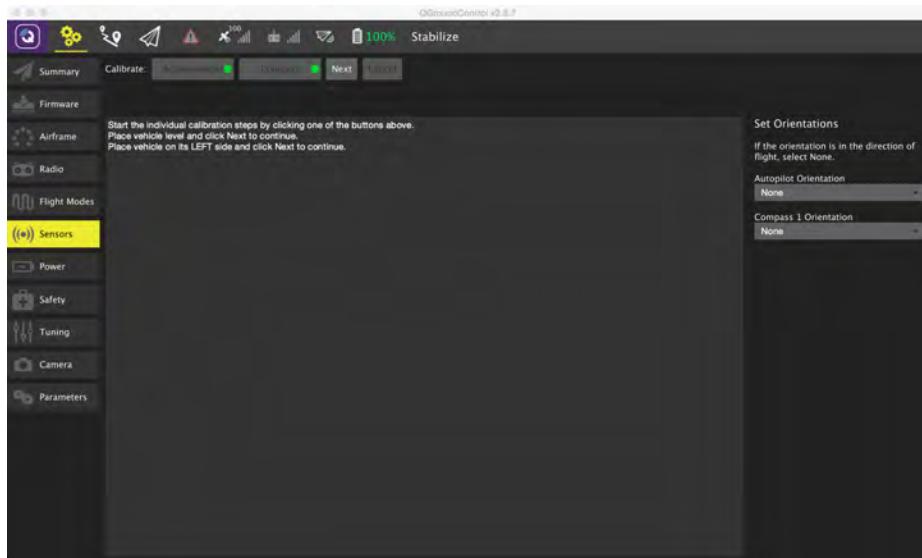


Figure 20-8 Accelerometer calibration lets the controller know which way is down

Flight Modes

Strictly speaking, setting flight modes isn't required. By default in APM all switch positions translate to stabilize mode, which is a fine place to start. But this drone can do so much more! In order to access other modes, we have to map the positions of the channel 5 rotary switch on the transmitter to different modes. The Quanum i8 transmitter outputs values that map perfectly to the default ranges in APM, so we just have to choose which modes to use.

I've mapped the six positions on my transmitter to stabilize, altitude hold, loiter, position hold, auto, and return to land, as you can see in [Figure 20-9](#). This makes a useful sequence since I generally take off in stabilize, which requires the least number of sensors, then go to altitude hold, which adds the barometer; only if that seems be working well do I use any of the last three modes, which all require GPS.

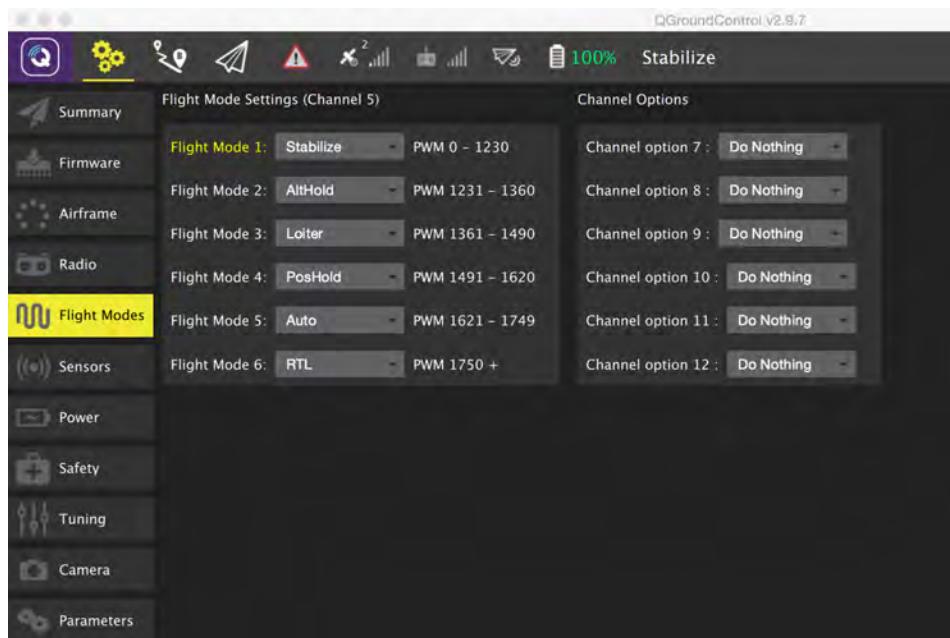


Figure 20-9 The default PWM ranges matched our transmitter, making setup easy

Initial Tuning

I'll be honest. I figured that this was a pretty typical setup, and I've tuned enough drones to at least my own satisfaction by now that this one should go easy. Nope. The first time I hit the throttle, it flipped on its back with amazing speed, shattering one prop. OK, I had two motors plugged in backward. But the tuning still took many flights. I believe that one factor was doing testing with a small battery and no payload—that is to say, lighter than I will

probably fly most of the time. This meant I was dialing back from twitchy instead of turning it up from sluggish. Not conceptually harder, but it kept my heart rate up and I broke a few props.

The APM flight system used here, version 3.3.3, does respond very nicely to being well tuned. For this work I prefer the extended tuning screen in Mission Planner ground control station. There must be a dozen different P or PID controllers that can be adjusted. But it turns out that you usually only have to adjust the core PID controllers for the roll and pitch rates, and the rest will all fall into place. For those key values, I am currently flying at 0.11, 0.00, 0.004. Since this quad is symmetrical, roll and pitch are the same, as seen in Figure 20-10.



Figure 20-10 The extended tuning screen in Mission Planner is well organized

Final Checklist and First Flight

Consider carefully where to do your first flight testing with any drone this size. Unlike our previous projects, this one can dent cars, break windows, or hurt people if it gets out of control. A large, unobstructed place with no other people around would be best, but few of us have access to an ideal location, so do the best you can. Working in a small space with nothing but brick walls to hit might be bad for your drone but safer for the world. When in doubt, err on the side of safety.

This Drone Requires FAA Registration

This drone weighs far more than 250 grams, so you must register with the FAA before flying it. It won't take you long or cost you much, so go do it. And then you have to follow through and really put that number on your drone before flight. The process starts at www.faa.gov/uas/registration/.

Next, consider the weather conditions for your first flights. A dead calm would be best. Try to notice what time of day tends to have the least wind in your area. You might also check the space weather at a site like www.uavforecast.com to make sure nothing funny happens to your GPS signals.

With time and place chosen, prepare the drone:

1. Check for any unplugged cables or loose parts.
2. Check the props for damage or missing balance tape.
3. Turn on your transmitter and make sure the throttle is all the way down, with the mode set to stabilize.
4. Strap on a battery, connect it to the voltage monitor, and check that you have enough power to fly, around 12.5 volts total.
5. Check the polarity and plug the battery into the main XT60 connector.
6. (Optional) Connect to the telemetry via your smartphone or computer and check the status.
7. Wait for a 3D GPS lock and an HDOP of less than 2 (ideally close to 1).
8. Check that the area is clear and arm the motors by holding the left stick down and to the right.
9. Slowly raise the throttle. The motors should all start.

Raise the throttle more, and you will fly! If the drone flies well in stabilize mode, then stay there. I find this one much easier to hover in altitude hold, but staying in stabilize requires the least number of sensors to be working right. This is where piloting skill really pays off. Time spent flying the smaller drones will make you more confident with this one.

Optical Flow, Sonar, and Lidar

21

Technology moves ever forward. The \$150,000 GPS receivers of the 1980s have become the \$15 pucks that we navigate our drones by. And while they can't do everything those military-grade receivers did, they work much better on the whole.

Similarly, sensors (and the processing power necessary to use them) that were once the exclusive domain of top-secret military projects can now be integrated into our drones. The hardware vendors trade open source designs to keep costs down, and the open source software communities work constantly to support the latest gadgets.

In this chapter, we'll look at several new technologies that are finding their way into drone projects today—optical flow sensors ([Figure 21-1](#)) and the related technologies of sonar and lidar. These last two measure distance; usually for us that means altitude, which we must know in order to use an optical flow sensor.

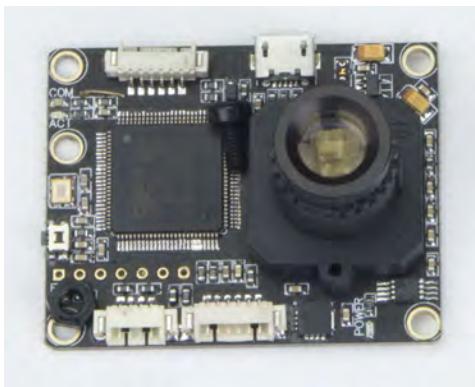


Figure 21-1 A PX4Flow optical flow sensor

Sonar

When we hear the word *sonar*, it conjures up images of scared men listening intently to pings in their submarines. Or perhaps the less martial of you think of bats using ultrasonic pings to track and catch insects. Starting with the Polaroid SX-70 camera back in the 1970s, which used sonar to focus, sonar has become a very cheap and common form of range finding in a wide variety of applications today.

Hobbyists use sonar for collision avoidance on robots or as motion detectors for interactive displays or a million other things. The HC-SR04 ultrasonic pinger module is so common, easy to use, and cheap that practically any project can include one. They cost just a few bucks and only take four wires and a few lines of code to use. [Figure 21-2](#) shows one of these modules.

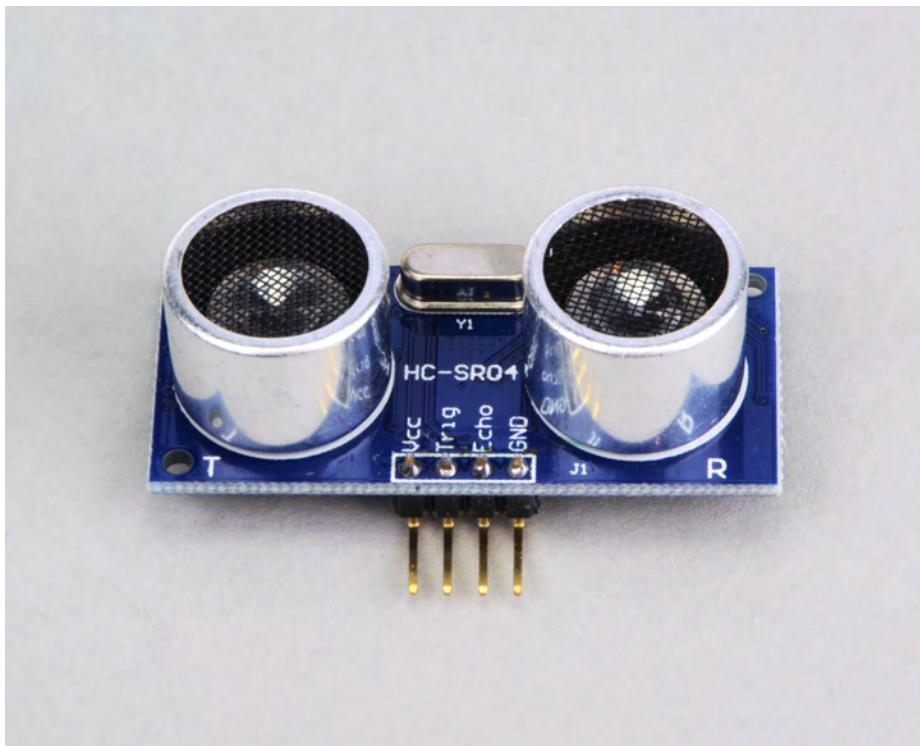


Figure 21-2 This cheap ultrasonic pinger can be used to determine altitude

People have successfully used HC-SR04 modules to control the altitude of drones. Their simplicity suits use with older, 8-bit APM systems. But they also have their limits. These are not the most accurate range finders available. They are subject to interference from background noise. And most importantly for us, they can only work up to a couple of meters at most.

For better range and accuracy, the Dronecode team recommend moving up to a somewhat more expensive sonar module like the Maxbotix HRLV-EZ4, which is included with the optical flow sensor we'll discuss shortly. Maxbotix sells a whole line of better ultrasonic range finders, and this one has the narrowest beam for the longest range. According to the manufacturer, this sensor measures down to the millimeter at ranges of up to five meters.

Lidar

The name *lidar* comes from a combination of *light* and *radar*. The technology uses light, generally lasers, to measure the distance to an object by measuring the time it takes for a light pulse to travel to that object and be reflected back. Light moves much faster than sound, of course, so a lidar system must be able to measure very, very small amounts of time accurately; thus, these systems still cost much more than sonar.

On the plus side, lidar can be more accurate than sonar and is much less prone to interference from the environment or other systems on the drone. Several manufacturers now make lidar systems specifically for drones, and the APM flight stack supports the Lidar-Lite series of modules. At this point, the open source projects like APM only use lidar for altitude measurement. In the future, it may also have a role to play in collision avoidance.

Optical Flow Sensors

Whether we know it or not, most of us use a form of optical flow sensor every day in the form of an optical mouse. Those of you who remember mice with rolling balls tracking movement know how much better today's optical technology can be. These work by comparing consecutive images from a low-resolution camera to see which direction the image has moved. The same idea—in fact, the same hardware—can be used to keep a drone over a fixed spot on the ground.

When the original Parrot AR drone came out to much fanfare, one of the technologies that made it stand out from the crowd was optical flow sensing. This gave it a position stability much tighter than is possible with GPS, indoors or out.

Today, in particular in the world of APM and PX4, we see two common optical flow systems being used. The cheaper option uses a DNS3080 sensor, literally built to be used in mice. You can connect this unit via its SPI bus to older APM 2.x 8-bit control systems. Combined with a super-cheap HC-SR04 sonar module, you can try out optical flow technology for under \$50—but it takes a lot of hacking, and you can't expect top-dollar results from bottom-dollar gear.

The better option in the Dronecode world, called PX4Flow, uses a purpose-built controller board with a better camera. As mentioned, drone makers generally pair this unit with the better Maxbotix ultrasonic sensors, which can connect directly to the controller. This results in a single, complete unit that connects easily via I2C, as seen in [Figure 21-3](#).

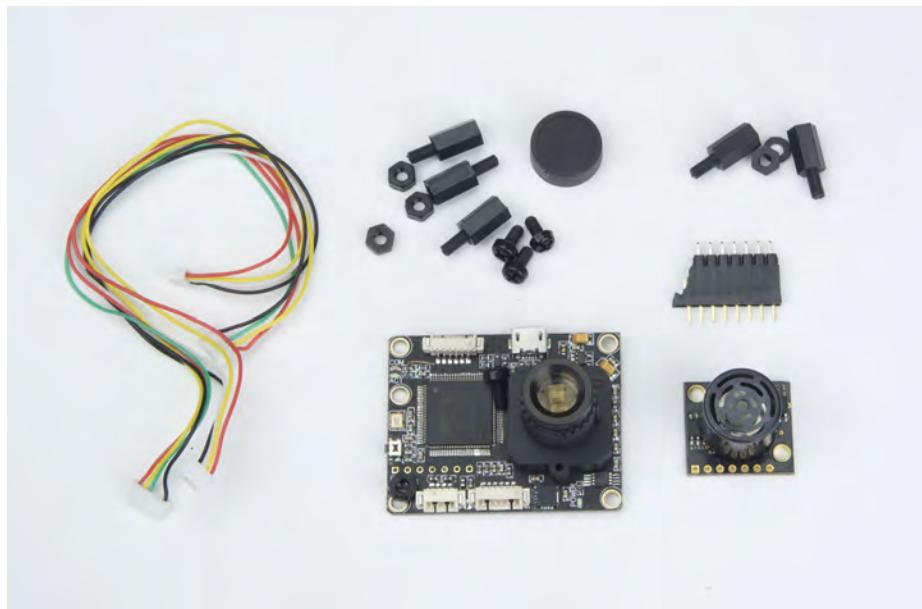


Figure 21-3 The PX4Flow often comes with an ultrasonic pinger for altitude

These still require some setup, though in this case the setup is kinda fun. You can connect directly to the optical flow controller board via USB and calibrate using special software, which includes the ability to directly view the camera image it is seeing ([Figure 21-4](#)).

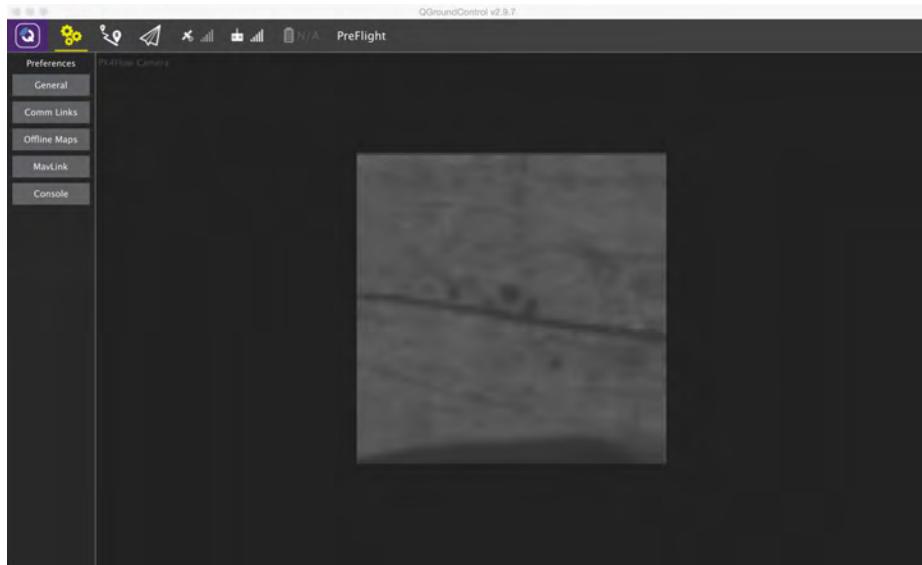


Figure 21-4 The PX4Flow config screen lets you see the camera image

Visual Odometry

Visual odometry: isn't that a fancy term? It just means figuring out where you are by looking around. We do this every day, but it's only starting to find its way into our robots and drones. Academics have been working on this technology for years now, especially as it applies to the world of augmented reality. The computer must be able to map the world around us in order to project things onto it.

Putting the processing power on board a drone to do this kind of interpretation and mapping of video images has been prohibitive until recently. Academic projects have used remote processing to test their theories. DJI actually built an experimental device called Guidance, which used a series of camera and ultrasonic range finders to attempt collision avoidance.

Right now, this is not something you can implement cheaply and easily: you can't just buy a part from China, plug in a standard bus, and enable it with a checkbox in your ground control software. But who knows, by the time you're reading this, visual odometry might be in every self-respecting drone out there.

When installed and set up correctly, some combination of range finding and optical flow sensors makes a very practical addition to today's advanced drone. This will allow you to fly with position stabilization indoors where there is no GPS, or to do precision automatic takeoffs and landings. Tomorrow's systems may allow your drone to find its place on a map by sight or fly near other drones without running into them. Watch this space for exciting developments in the immediate future!

Vibration

22

If you read drone forums for long, you will inevitably run across a discussion of vibration. I have had several hard crashes I believe were caused by vibration, so I have a very personal interest in the subject.

Vibration matters to anyone who flies a multirotor aircraft. We all use inertial stabilization systems, gyros and accelerometers, and these are very sensitive to vibration. Aerial photography depends on having a stable platform from which to shoot. And in the end, bad vibrations can simply shake a drone apart.

Figure 22-1 shows the log from a test when my drone was suffering from some devastating vibrations. It shows forces in the range of three to five times the force of gravity (that is, 3–5G). A reasonable reading here would be less than one tenth of that. A good aerial photography platform will often read less than 0.1G.

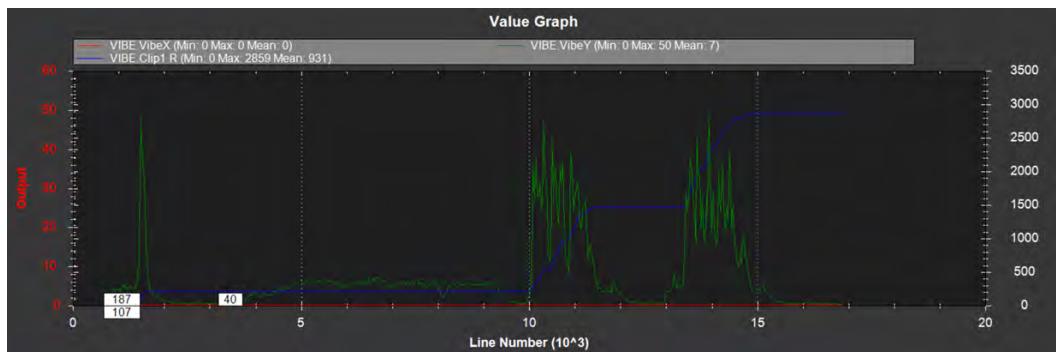


Figure 22-1 This is what excessive vibration looks like in a log

Causes of Vibration

It's ironic that vibration should be such a problem on a vehicle with so few moving parts, but those moving parts spin very fast—up to 7,000 RPM for the 1045 props and 2216 motors of the big drone we just built. If a prop or motor or even a prop nut is even a little out of balance, it will create vibrations. Since we run our motors over their full possible range of RPM during normal flight, if there is a bad frequency in there that lines up with the resonance of the frame, we will eventually hit it. [Figure 22-2](#) shows a frame from a video of this happening with this drone, with a different motor.



Figure 22-2 When vibrations resonate with the frame, they can get large

You will be glad to know that with decently balanced props, the EMax 2216 motors we will fly this drone with have never exhibited this kind of vibration. It should make a fine aerial photography platform.

Effects of Vibration

Vibration causes many troubles. In the drone forums, people talk the most about jello in their video. I'll get back to what this means in a minute, but I believe the reason this gets mentioned so often is simply that shooting video from a drone happens to be a really good way to see how bad the vibrations are at different times in the flight. You can look at logs, of course, but jello in the video is what people notice when everything else is fine.

I suppose at a certain point vibrations might simply rattle parts off, but I haven't had this happen. Certainly, screws will come loose at a faster rate as the vibrations get worse. This

could cause parts to fall off eventually, but as long as you check on the important screws before flying, this won't happen to you.

The insidious problem that you can't see until it's too late happens when the vibrations start affecting accelerometers and gyros. It might just cause a little drift. Since inertial navigation gets used to supplement the barometer in altitude hold mode, vibration can make your drone wander up and down in this mode. Finally, advanced multisensor systems like the one we're using here constantly check their sensors against each other and will stop using one that doesn't seem to be working. This will almost certainly happen at a critical moment, just when you could use all the help you can get from your sensors!

Back to jello in the video now. When your drone vibrates back and forth, and the camera isn't well enough protected from those vibrations, you will see these small movements reflected in the video. But action cameras don't take a whole frame at once. They scan down the frame at some rate. If your drone vibrates back and forth three times for each frame the camera takes, then vertical lines will appear to have three waves in them as you look down the frame.

Damping and Isolation

Technically, what everyone calls "vibration damping" in drones—the kinds of shock mounts we put our controllers on or that most gimbals hang from—should really be called "vibration isolation." Damping occurs when the energy of the vibration gets converted to some other form, usually heat caused by friction, before it gets to the item being protected. Automobile shock absorbers do this when they use a fluid to convert the car's motion into heat. Isolation occurs when the energy passes through the item without affecting it much. Springs, whether steel or silicone like the damping balls often used in drones, temporarily absorb the energy so the protected item doesn't have to.

The best systems use a combination of damping and isolation. Cars have shocks and springs, or perhaps struts, which are nothing but a well-balanced combination of the two. We don't see actual damping systems in drones much, so let's see how we can use isolation to reduce the trouble that vibration causes.

First, let's be more specific about our goal. We want to stop the high-frequency, regular motion that comes from the rotating props and motors. But we must let through the actual motion of the drone without significant change or delay. Mathematically, these are the low-frequency movements. Drone makers use silicone balls, stretched o-rings, and pads made of special vibration isolation material. In each case, the material acts as a kind of spring. To make it work well, we must adjust the stiffness of that spring to the situation.

The frequencies of the movements we want to stop and pass affect the ideal spring stiffness, but we can't change this. We can however, change the other important factor—the weight of the thing being sprung. The heavier the object being isolated is, the easier it is to isolate it from the small, high-frequency movements that we don't want. We can make use of this fact by including more weight with the control system. I have considered using a

larger shock mount on this drone and putting the RC and telemetry radios on with the flight controller, though the vibration levels are fine already. Some commercial drones, like the Parrot Bebop, place the battery on the isolated side with the control system and camera, adding a great deal to the sprung weight.

Prop Balancing

If you are having vibration problems, anything from mild jello to crashes, you need to balance your props. Even if you see no sign of vibration whatsoever, balancing your props will make your motors run a bit smoother and let the bearings last longer. It's easy. It doesn't take expensive tools. It does take a specialized tool, a prop balancer, but a basic one like that shown in [Figure 22-3](#) costs less than \$10. Better ones will allow balancing self-tightening props that don't have a hole all the way through.



Figure 22-3 A prop balancer, scissors, and electrical tape are all you need to balance a prop

With the proper hub adapter in place, gently screw the prop into the balancer. Cut a piece of electrical tape, one centimeter square for a badly balanced prop or less if the prop already seems close. Stick the tape on the leading edge and check the balance. Move it back and forth until the blades seem as even as possible. Now it's time to balance the prop

hub. Yes, it's kind of a fine point and not everyone does it, but you've got it on the balancer, so why not?

Once the blades are perfectly balanced, push one side down. If the prop has a tendency to return to level no matter which blade you push, the lower side of the hub is heavier. If, on the other hand, when you push either blade down, that side stays down, then the top side of the hub is heavier. You can lighten the heavier side by sanding a bit with a medium-grit sandpaper. It is probably best to take the prop off the balancer so you don't sand it and unbalance your tool. I find that one side of the hub always has noticeably more flashing and ends up heavier, so I will often sand that side down a bit before even starting.

If you get good motors and balance your props well, you can enjoy smooth flying and smooth video. And even if your drone flies fine with props straight out of the delivery box, do your bearings and your gyros a favor and balance them up anyway. Your drone will feel more like the precision machine that it is.

Failure Modes and Fault Tolerance

23

If you start thinking about all the ways a failure can bring your drone down, it can get a little depressing. Any one prop coming loose, any one of the dozen solder joints running the motors coming loose, any one of the 24 FETs in the ESCs failing..., and that's just the power system. And everything goes back to a single battery connection. We call each one of these items a single point of failure, and we can do better.

Engineers will sometimes talk about having "no single point of failure," but we almost never really reach this level of reliability, even in just the control system. The trouble comes in the transition. If a component fails, its backup must not only be present but be up and running and in exactly the same state so it can take over seamlessly.

If we can't keep running, perhaps we can at least fail safely. In fact, failsafes are now common in drone control systems, and drone makers have learned to duplicate certain key components and either switch over or manage without when one fails.

Failsafes

No doubt, pilots of RC aircraft have been dealing with loss of their main radio connection ever since the first radio-guided balloon took flight in the nineteenth century. The old-timers have endless stories of the "one that flew away." Today, we can (usually) do better.

If your drone has a good GPS lock when you take off, should you happen to lose radio contact the flight control software can be set to automatically return and either hover or land at the takeoff site, first climbing to a safe height to avoid obstacles. [Figure 23-1](#) shows the failsafe setup screen in QGroundControl, where all the safety-related settings are nicely gathered in one place.

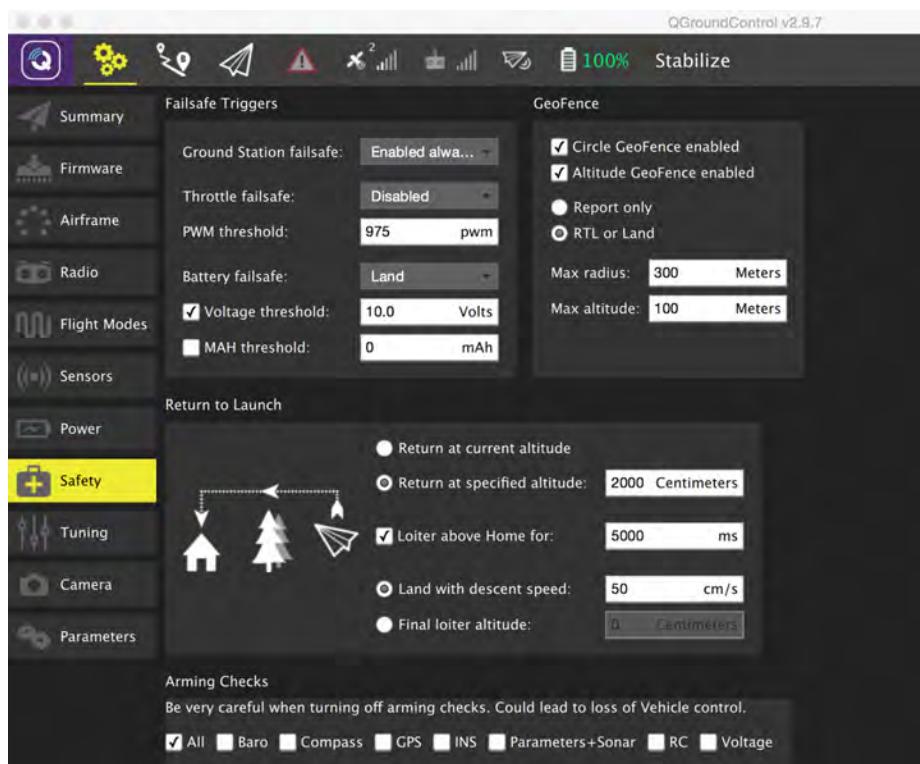


Figure 23-1 QGroundControl puts all safety settings in one place

The other common failsafe of this sort that current control systems usually implement triggers when your battery gets low. Even the smallest drones will blink their lights as a failsafe warning when they are running low on juice. More complex drone control systems will often have a two-step response: first a warning, like lights blinking or warning buzzers, followed by an automatic landing if the battery voltage drops below a certain limit. I highly recommend that you review all the failsafe settings for any drone you fly. It's very confusing to see your drone start to climb when you tell it to land if you don't know that it has been set to a safety height of 30 meters.

Seeing a drone that you lost control of coming home to land can be quite a relief, but not every failure leaves a drone able to perform this maneuver. In the case of a motor failure or power loss, you can't really keep flying, so some heavier commercial drones carry recovery parachutes that can be triggered when it all goes wrong. Such recovery chutes are required of commercial operators in parts of Europe where drones may be operated in close proximity to people.

Redundancy

Pilots of traditional multiengine aircraft train at length on how to continue flying in the face of an engine failure. While it is possible for a quadcopter with an engine out to keep flying, it takes specialized programming. Once we move up to six or eight motors, it becomes more reasonable to expect the control system to be able to cope with the loss of one. If the drone has enough excess power and the failure doesn't result in some kind of terrible imbalance and vibration, then a well-tuned system might cope well enough to allow a controlled landing, at least. This is probably on the minds of professional aerial photographers as they mount their expensive cameras, but then those cameras are pretty heavy. I wonder if they have the excess power to really pull it off.

The control system of even a moderate-sized quadcopter may use a couple of other types of redundancy to improve reliability and performance. Since everything depends on the power supply, it's no surprise that power problems cause many failures. Highly redundant power systems might use two batteries with each running to different sets of motors, but even with one battery, controllers like the Pixhawk and APM can receive power from both a power module and a battery eliminator circuit in one of the ESCs and switch completely to the good source when one source fails.

As we add more and more complex sensors, like GPS receivers, we must start thinking about what happens when they fail. The Pixhawk flight controller actually has two accelerometers and two gyros. It only has one compass, but it is expected that you will add an external one as well. There are two GPS ports too. The navigation filters in the ArduPilot flight stack were converted to use extended Kalman filters (EKF), which make the handling of sensors coming and going work better, and much logic was added to constantly compare the different sensors.

As drones become more complex and integrate ever more technology, failsafes and redundancies will be essential tools in their design. As the designers of the first vacuum tube-based computers knew all too well, one fairly reliable part makes a fairly reliable system. Use 10,000 of them, and you have to start thinking about detecting and handling failures.

Throughout this book we have been connecting one processor to another using some kind of interface with another acronym for a name. These standards allow us to buy a GPS unit from one company and a flight controller from another and believe that they will be able to talk. But the number of them, and the constant stream of new ones, can be a bit daunting. This chapter lists many of the interfaces commonly used on drones, with some discussion of what each can be used for.

Computer Interfaces

Much of the wiring on a complex drone connects different computers together: for example, the flight controller to an ESC or to the telemetry radio. All of these connections use some kind of serial interface. This just means that the bits are sent one after another on a single wire instead of in parallel on several wires.

In some cases, no higher standard is named. It's plain old serial (no acronym). The telemetry receiver connects to the flight controller using four wires: power (often called Vcc), ground (GND), transmit (TX), and receive (RX). For two computers to talk this way each one's TX must be connected to the other's RX; then one usually supplies power to the other, as shown in [Figure 24-1](#).



Figure 24-1 In this basic serial connection, each side's transmit connects to the other side's receive

As with any electrical connection, you must be careful of the voltage. The power supply is probably five volts (5V), but it could be 3.3V or even 12V. The so-called logic level on the TX and RX lines could be 3.3V or 5V, or perhaps even 3.3V, but 5V compatible. This last phrase means that a device will put out a 3.3V logic level but if you feed it a higher 5V output it won't fry anything. Otherwise, the voltages should match everywhere.

As we found out when choosing a FET, a system that uses so-called 3.3V logic does not actually output 3.3V for high and 0V for low. If you put a volt meter on a high signal in a 3.3V system, it will probably read something just under 2V. Similarly, a low signal won't be quite 0V. As long as you connect systems of the same family together, either 3.3V or 5V, then you don't need to think about the exact voltages used. It should only come up in detail when you do strange things like use a 5V data pin to power a 3.3V receiver so you can pulse it to put it in bind mode.

Physically, simple serial interfaces can use almost any connector. Basic Arduino projects often use no connectors at all, relying on simple pins and headers or wires in breadboards. Drones will often use smaller connectors like the DF13 or the Micro JSTs shown in [Figure 24-2](#). These connectors come in many sizes, with two to a dozen or more pins and with pin spacings of either 1 mm, 1.25 mm, 1.5 mm, or 2.5 mm. The extra pins might be unconnected, or they might be used for a second port on the same connector, as on the Pixhawk Lite flight controller.

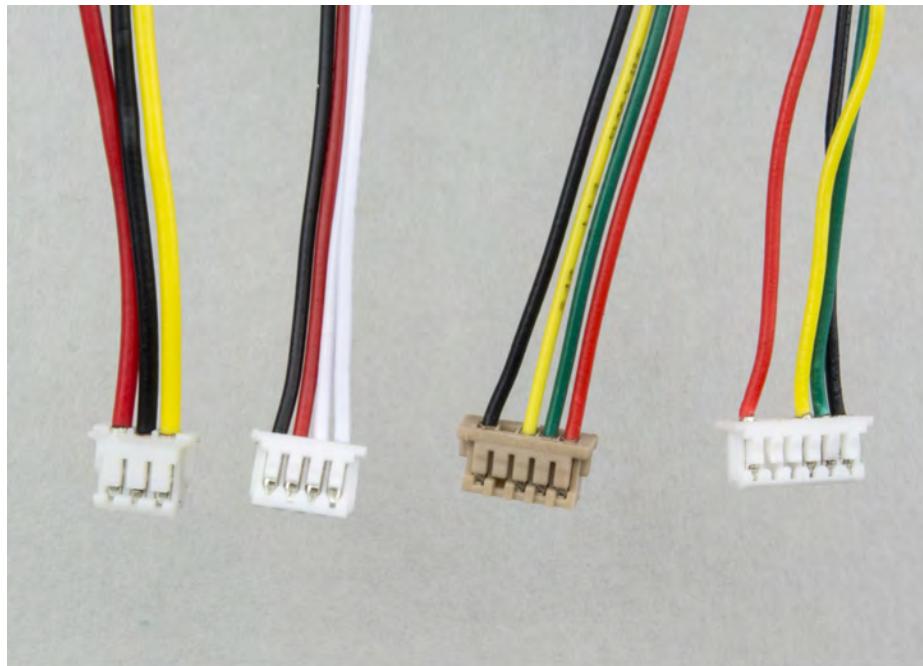


Figure 24-2 Micro JST connectors with 1 mm spacing and from three to six pins

That's the physical connection. Logically, the sending computer holds the data line high until it is ready to start sending a message. Each message begins with a low start bit to let the other computer know a byte is coming. The sender then puts out eight bits, one after another, followed by a high bit to show that the message has ended. The order runs from most significant to least significant. This format, called 8N1 to signify eight data bits, no parity or check bit, and one stop bit, seems to dominate basic serial communication, though other formats are certainly possible. [Figure 24-3](#) shows the value 0xB2 being sent this way.

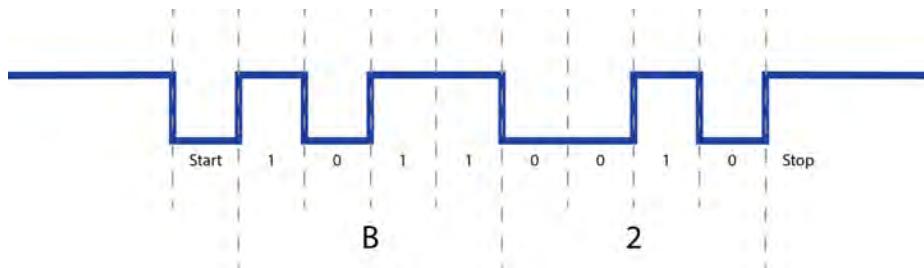


Figure 24-3 The hex value B2 as sent down a serial line

We call the speed at which these bits get sent the baud rate. The unit baud (Bd) honors Émile Baudout, who invented the 5-bit code used in early paper teletype machines. The unit measures the rate at which symbols, in this case ones and zeros, are sent. In the case of a simple serial line this is the same as the bit rate. Common baud rates in drones are 57,600 or 115,200 bits per second. Some connections, like I2C, might be much faster.

Now we've sent a single number. Other standards define what the numbers mean. A more complex message might start with an address, so that multiple devices can share a connection but send messages to only one of them, then maybe a message type indicator defining the rest of the content. For each interface we use—inter-integrated circuit (I2C), serial peripheral interface (SPI), controller area network (CAN), and many more—committees meet and long, technical documents get written. This is all very important to certain engineers, actually.

Thankfully, you don't need to read those documents to connect an I2C compass to your flight controller, but here are a few things that might help when using I2C. Notice that the basic serial wiring diagram ([Figure 24-1](#)) includes only two devices. We call I2C a *bus* because it allows multiple devices to connect to the same port. We can, for example, add an optical flow sensor on the same port where the compass connects. Also, knowing that two of I2C's four wires are for data and two for power explains why the compass connection from an integrated compass/GPS only has two wires. The compass gets its power from the GPS's connection.

One more thing about I2C, since it is such a common interface on drones: interference from a power system or other busses can cause communication errors with I2C. It's not the most fault-tolerant way two processors can talk. There is a reason that there are logs for I2C

errors. If you have trouble with an I2C bus, try just twisting the wires a few times. Other measures, like using shielded cables, work too, but a few twists can do wonders to equalize the interference in each data line so it can be canceled out.

Given the trouble with I2C, drone makers have been looking for better alternatives. And what better place to look than cars, a notoriously harsh environment in which to run computers? The automotive industry came up with a very robust bus called the controller area network, which works well for connecting multiple devices to a control system in tough conditions. Already, some ESCs use a CAN bus and the Storm32 gimbal has been redesigned to use it as well.

Finally, as far as computer-to-computer connections go, I've watched as other industries turned grudgingly to Ethernet instead of other protocols designed specifically for control systems. Like so many things, the mass market has made very fast, reliable networking gear available. Ethernet's randomized retry method of determining who gets to use the bus does not suit control systems engineers' desire for predictability. But Ethernet runs so much faster for so much less money than most of the alternatives that I won't be surprised to find it in my drones soon.

RC Radio Standards

Radio control systems have come a long way since the time when doing even single-channel electronics was hard, so systems like the Galloping Ghost used mechanical parts instead. On the radio frequency side, where once you had to have a flag on your transmitter's antenna to show which channel you were using, now many people can share the 2.4 GHz band without interference because of frequency hopping and spread spectrum technologies. And once again, mass market gear (WiFi this time) using these same frequencies helps make the parts common and therefore cheap.

Still, the world of RC radio has its multiplying standards, like DSM/DSM2/DSMX, FrSky, or even AFHDS 2A. The systems named by these acronyms each use the 2.4 GHz radio band a little differently. The systems vary in their cost, ability to reject interference, range, and number of channels, and the variation among the different models using the same standard might be just as great. You should research the particular model you are considering, not just the standard.

DSM/DSM2/DSMX

While as a rule your transmitter and receiver must use the same standard, we have actually seen one of the exceptions in our earlier projects. The DSM/DSM2/DSMX series of standards, owned by Spektrum, each build upon the last. We used a DSMX transmitter and a DSM2 receiver. The original DSM used a single frequency, DSM2 could switch between two frequencies to avoid interference, and DSMX uses multiple frequencies in a more complex way. But any DSM2 or DSMX transmitter should work with any DSM2 or DSMX receiver, so our use was fine.

Some variety of DSM seems to be the most popular radio standard for the smallest, completely integrated flight controllers, like the MicroMWC we used in the X4Wii or the similarly sized but more powerful Pico32 controller. If you want to go modular, like we did in the Visible Drone, the receivers are still cheap, and we used the same transmitter for both of those projects. The ability to use just the satellite receiver makes very small projects possible as well.

FrSky

FrSky builds a line of advanced transmitters and receivers. The transmitters support advanced mixing and mode controls similar to many of the RC settings available on flight controllers. FrSky radios take advantage of the fact that a 2.4 GHz radio link is actually two-way in order to send telemetry data back from the aircraft to the transmitter. The company also sells a line of sensors, including airspeed, GPS, and current sensors, that plug directly into the receiver for display on the transmitter's screens. While most of these features might be more useful in fixed-wing aircraft, many serious drone pilots use FrSky for its quality and performance.

Servo/ESC Control

While you might see a few different connectors used, almost all servos used in RC models adhere to the same electrical and logical standard. They use 5V logic levels and send data using pulse width modulation (PWM). There will generally be three wires, for power, ground, and signal. The signal consists of a series of pulses, each from 1,000–2,000 microseconds long. This is why you often see the RC control positions represented as a number in the range 1,000–2,000. The length of the pulse tells the servo what position it should attempt to get to or stay at. With each pulse, the position can change. Most electronic speed controllers have adopted this same PWM standard for their control.

Pulse position modulation (PPM) uses a string of timed pulses like PWM, but every pulse will be the same width. The timing, or position, of each pulse determines the value that it represents. Combined pulse position modulation (CPPM) builds on this idea, sending groups of eight values in sequence with a longer sync gap in between groups so that you can tell where the group starts. Many modern RC receivers will have separate PWM ports for each channel, but also a single CPPM port that allows the receiver to be connected to a flight controller with a single cable.

Graphics often convey concepts like PWM and PPM better than words, so here's an example. The black line on the top in [Figure 24-4](#) shows the original to be encoded. The green line shows how that signal might be encoded using PWM. The blue line on the bottom shows how that same signal might be encoded using PPM.

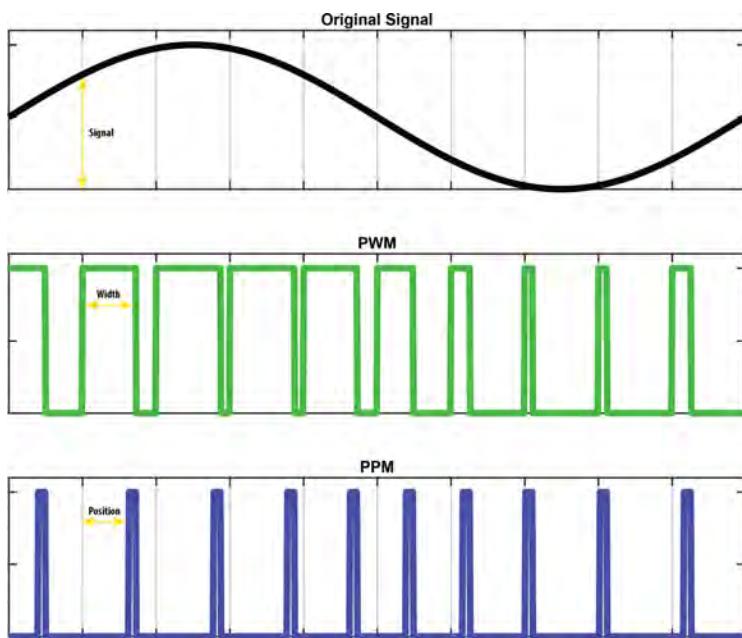


Figure 24-4 PWM and PPM encode the same information in different ways

Telemetry

Telemetry data tells us things like the drone's current attitude or position. It can also contain status or configuration information like modes or PID settings. Telemetry goes both ways in most cases, meaning we can change things on the drone as well. This might let us update configuration like those PID settings, or take control of the drone, change modes, and tell it where to go. This makes telemetry a useful backup in case you have trouble with your main RC radio.

Telemetry data gets communicated within the drone control system via serial links and between the drone and the ground over various types of radio links. The complex and varied ways that data gets represented on the radio fascinate me, but you don't usually need to know about these as long as your transmitter and receiver match. What you do need to be aware of are the definitions of the different telemetry standards.

The two most common telemetry standards each have their own worlds of flight control and ground control software that support them. The MultiWii/Baseflight/Cleanflight branch of the flight control software tree all use the MultiWii Serial Protocol (MSP). The Visible Drone, earlier in this book, also implemented some MSP message types. The ArduCopter/PX4 flight control systems use what they call MAVLink to talk to their ground stations.

To show what these messages look like, let's consider an example. All MSP messages start with the characters "\$M". Then, for messages coming from the drone the next character will be ">", and it will be "<" for messages going back up. Next we have a count of the data

payload, not including all the preamble and command bytes, and then the command. Finally, there are the actual binary values. The attitude message, called `MSP_ATTITUDE`, is command number 108. Thus, the complete message looks like this:

\$M	Preamble
>	Direction indicator
6	Payload byte count
108	Command
<i>16-bit integer</i>	(Roll in degrees)*10
<i>16-bit integer</i>	(Pitch in degrees)*10
<i>16-bit integer</i>	(Yaw in degrees)*10

GPS

In addition to whichever telemetry protocol is in use, we often see one other serial communication standard in our drones—the National Marine Electronics Association’s NMEA 0183 standard for GPS data. Like MAVLink and MSP, NMEA 0183 defines a series of message types, in this case for things like fix quality and position. The standard also includes things about the physical connection, baud rate, etc., but the GPS units we see in drones don’t use that part of the standard. They usually communicate using 3.3V logic levels at 57,600 baud, like nearly everything else. Note that things like changing the baud rate are not a part of the standard and thus may vary from one brand of GPS to the next.

With GPS units as cheap and easy to use as they are today, you can include one in any project where knowing your position, or even just the correct time, would help. A quick example will serve to demonstrate this simple protocol. The message shown here gives a current position. Note that unlike with MSP, numeric values are given in text, not binary, and the data fields are separated by commas:

\$	Preamble
GP	Message from GPS
GGA	Time and position message
172814.0	UTC time
3018.9370	Latitude as <i>ddmm.mmmm</i>
N	North or South latitude
09743.0739	Longitude as <i>dddmm.mmmm</i>
W	East or West longitude
1	Fix quality, 1=3D fix

16	Number of satellites in use
0.9	HDOP
151.0	Height above sea level
M	Height is in meters
2.0	Height above the ground
M	Height is in meters
<empty>	Used for differential GPS
<empty>	Used for differential GPS
*	End of message indicator
1234	Checksum

When we put this together in a complete message it looks like this:

\$GPGGA,172814.0,3018.9370,N,09743.0739,W,1,16,0.9,151.0,M,2.0,M,,,*1234

While that might not immediately say Austin, TX, to you, it's pretty easy for a computer to interpret.

An understanding of the various standards and protocols in use on drones will serve you well, whether you want to write a fancy app to control your drone remotely or just want to understand why we have to swap the transmit and receive wires sometimes. Perhaps you are building some new type of aircraft that the world has never seen! If you make it speak MSP or MAVLink, you can take advantage of existing ground station software and concentrate on your new design. Maybe understanding how these protocols work will inspire you to invent your own protocol. If it's good enough, maybe they will name it after you, like Samuel Morse and his Morse code or Émile Baudot with both Baudot code and baud rate.

The Future

25

As futuristic as a current drone performing a completely autonomous flight might seem, most people agree that we are only at the beginning. There may yet be new breakthroughs that change the game, breakthroughs like in the motors, batteries, and control systems I described in [Chapter 1](#) as making drones possible. Battery technology seems like one place where radical changes might happen. But we will stay on safer ground, or perhaps in safer skies, here by only asking where current trends might take us.

Specialization

Already an aerial photography drone looks different from a racing drone. If and when we have delivery drones, those will surely be a different breed again. Just as today we have several well established categories of cars (economy car, pickup, sports car, off-road vehicle, etc.), I would expect there to be several established categories of drones.

Just as today few people could tell you how many horsepower the engine in their car can produce, many categories of drone will simply do a job with performance measured by one or two obvious numbers, like flight time. Then, as with the car hot-rodgers, there will be certain groups who know every last thing about their drones, who are constantly trying to find the best combination of components and programming.

I would, however, expect one aspect of drone flying to move toward standardization rather than specialization: the controls by which we fly them. Just as cars have the pedals in the same places and traditional aircraft have standard instrument layouts, I hope that the different manufacturers and control system communities will adopt some common human interface standards. I think the names and precise definitions of several standard flight modes could be common. Safety would be increased if safety features like the safety switch in our big drone got used the same way in every flight stack.

Regulation

Governments have taken an interest in drones as well, and new regulations on drones get passed all the time. It's a pretty good guess that this trend will continue.

Ease of Use

The trend toward ease of use in drones has taken place mostly in the mid- to large-sized drones so far, because these have the payload capacity to carry the new technology. As that technology becomes more common and more highly integrated, it should make its way into smaller and smaller drones.

An easier-to-fly drone translates to a safer drone. If the controls are too complex, then mistakes are more likely. There are commercial jobs that drones can do, but if someone has to go practice hundreds of hours to learn to fly them, it's not gonna happen.

The educational possibilities for drones will be opened up as they become easier to fly, too. Just think what kids could do with a drone that could be sent on autonomous missions with an interface as fun as Lego Mindstorms! The lesson, like the job, is more likely to succeed when the reward is more immediate.

On the other hand, I hope there will always be a place for piloting skill as well. Drone racing is fast becoming a popular spectator sport. The pundits already debate whether race car drivers are athletes in the same sense as runners. I wonder if a guy sitting in a lawn chair with goggles on twiddling his thumbs on some sticks will count? In his head he's going a hundred miles an hour!

Safety

As we saw in [Chapter 23](#), drones have been getting steadily safer, with features like sensor redundancy and failsafe systems. I certainly hope this trend continues with recovery parachutes and fully redundant power systems and controllers, and that we handle failures ever more gracefully.

As the number of drones increases year by year, we will be forced to keep making our drones safer if we want to be allowed to keep flying them. If drones are to be allowed to fly over people, just as traditional aircraft fly over our cities every day, then the risk of a drone crashing must be much lower than most drones in the air today can claim.

I believe we will learn to build a class of drone that can be safely flown in and around people. People fly their drones close to people all the time now, regardless, so this will be good for everyone, pilots and bystanders alike.

Human Flight

Every so often, a new story shows up where someone or some company has strapped enough drone technology together to be able to carry a human. Sadly, I have to say that most of them look like a bad idea. I have already commented on the number of single points of failure in today's drones. This can be reduced by redundancy, but actually designing control systems where human life is at risk starts much further back than that.

When designing for human safety, you start with the management processes. If you don't know what you are doing, how will you know that you are doing it right? Redundancy and fault tolerance in the narrow sense that we discussed earlier certainly play a part. Specialized analysis techniques like failure mode and effects analysis (FMEA) or testing techniques like hardware-in-loop (HIL) testing allow the systems to be validated before they are actually built. These things will come to drones. Already HIL testing and simulation can be done using certain ground control station software.

Driverless cars that carry humans share the road with us now. Our small, remotely or autonomously piloted drones will need to share the sky with larger, traditional aircraft as our numbers grow. We may not be there yet, but I can imagine the day when some convergence of drone and conventional aircraft technology lets me take off in my own multirotor. I can't wait!

Index

Symbols

10DOF (degrees of freedom) sensors, 52
250 class FPV racers, 21
380 mAh, 25C, 1S LiPo battery, 26
3D Robotics, 12
3DR 433 MHz radio sets, 143
3S, 2700 mAh batteries, 164
6DOF (degrees of freedom) sensors, 52, 80
9 DOF (degrees of freedom) sensors, 81

A

accelerometers, 6, 75, 77
calibrating for S500 drone flight test, 167
in MultiWii Manual mode, 45
Rate mode and, 85
vibrations affecting, 179
acro or acrobatic mode, 85
(see also Rate mode)
activities with drones, 19-22
aerial photography, 19
education, 21
flying, 22
FPV racing, 20
aerial photography, 19
aerodynamics, multirotor, 69-74
lift and thrust, 69
pitch and roll, 71

translational lift, 73
vortex ring state, 74
yaw, 71
AFHDS 2A standard, 190
Ah rating of batteries, 164
AHRS (attitude and heading reference system), 52, 76
(see also IMU/AHRS)
alternating current (AC) motors, 131
altitude
controlling using HC-SR04 ultrasonic pinger module, 172
measurement using lidar, 173
altitude hold mode, 168
effects of vibration in, 179
analogWrite() function, 66
Anderson, Chris, 12
Angle mode, 45, 71, 84
anisotropic magnetoresistance (AMR), 158
antennas, Quantum i8 RC receiver, 142
APM Planner (Mac), 145
compassmot test, 160
APM/ArduPilot, 11
APM:Copter flight control system, 12
running software version 3.3 or later, 13
ArduCopter, 118
drones based on, 2
ArduCopter/PX4 flight control systems, 192
Arduino boards

in MultiWii control systems, 10
using Arduino Uno to bind Lemon RX receiver, 60
Arduino IDE
modifying code for X4Wii build, 35
setting connecting to MicroMWC controller, 35
setting up for use with Teensy 3.2 processor, 57
Arduino project for X4Wii, downloading, 35
ATTITUDE command (MSP), 64
auto mode, 168
axisCmd, 84
axisCmdPID, 86

B

Bakke, Rolf, 14
barber pole sensors, 158
Baseflight, 16
batteries, 4
and hover time for different props, 29
connecting 3S, 2700 mAh battery for S500 drone, 164
for Visible Drone project, 53
getting low, warnings about, 184
hover time for different batteries, 27
in redundant power systems, 185
soldering battery connector onto MicroMWC controller, 37
upgrading for Hubsan X4 107C, 26
USB power and, 113
battery eliminator circuits (BECs), 135, 185
baud rate, 189
Blade MLP4DSM transmitter, 33
Blink example program, running, 58
Bluetooth
connection baud rate, slow loop and, 60
HB-02 serial Bluetooth module, 32
building cable to connect to MicroMWC, 39
HC-06 Bluetooth module, 63
radio for Visible Drone project, 52
BNF (bind and fly) version, Syma X5, 55
brushed motors, 3, 131

(see also coreless motors)
brushless motors, 3, 118, 131-136
and their ESCs, 134
BECs, 135
EMAX MT2216 II KV810 motors, 133
overview, 131

C

C rating for batteries, 164
cables
checking for defects, 128
connecting MicroMWC to Bluetooth module, 39
calcIMU() function, 78
cameras on drones, 19
CAN (Controller Area Network), 189
capacitors, 100
carbon fiber props, 162
CC3D/OpenPilot, 15
channels, parsing raw serial data into, 62
checklist, final, before first flight, 170
cinematography, aerial, 19
circleDiff() function, 84
circuit board and motor controllers, 54, 97-106
component placement, 98
construction, 104
design tools and files, 98
motor controller, 100
power input and distribution, 99
schematic and layout, 102
Cleanflight, 17
coils in brushless motors, 131
number of, 133
width and height of, 133
combined pulse position modulation (CPPM or PPM), 141, 191
compass calibration, 159
compass dance, 159
compassmot, 160
computer interfaces, 187
connectors in serial interfaces, 188
constructing and tuning a drone, 107-115
completing the frame, 110
connecting motors, 108

flight testing and future work, 114
ground test, 112
installing radios, 109
installing the controller and connecting LEDs, 107
control systems, 5, 9-18
adding to sprung weight, 179
building custom controller for Visible Drone, 49
circuit board with motor controllers, 54
Teensy 3.2 CPU, 50
CC3D/OpenPilot, 15
choosing a flight control system, 17
DJI/Naza, 14
Dronecode, 11
in brushless motors, 4
installing a controller, 107
installing flight controller and radios, 137-146
 PixHawk Lite and shock mount, 138
 RC radios, 140
 safety switch, 144
 telemetry radio, 143
 testing, 144
KK2, 14
MultiWii, 9
Naze32/Baseflight/Cleanflight, 16
possible future standardization in, 195
use of redundancy to improve reliability and performance, 185
writing basic I/O code for, 57-67
 main loop, 59
 RC receiver, 60
 setting up the IDE, 57
ControlCopter 3D (see CC3D/OpenPilot)
Controller Area Network (CAN), 189
coreless motors, 3
 brushed motors and, 3
Coriolis vibratory gyroscopes, 76
CPUs
 in drone control systems, 5
 more powerful CPU for Visible Drone, 50
crash kits, 32
crimping tools, 108
cyclic redundancy checks (crc), 64

D

damping (vibration), 179
DC-DC converters, 53
declination (magnetic), 156, 159
derivative (in PID algorithm), 88
 tuning, 92
design file (VizDrone), 98
desktop program, MultiWiiConf, 43
deviation, 157, 159
differential thrust, 71
differential torque, 71
diodes, flyback or protection, 101
direct current (DC) motors, 131
DJI Flamewheel 450 and 550 quadcopters, 118
DJI/Naza, 14
DNS3080 optical sensor, 173
drone activities (see activities with drones)
Dronecode, 11
 APM/ArduPilot, 11
 PX4/Pixhawk flight controllers, 13
drones versus quadcopters, 2
DSM/DSM2/DSMX standards, 190
DSM2 protocol, 33
 power-on sequence for DSM radios on X4Wii, 41
 seven channels in each frame, 62

E

Earth, magnetic field, 155
ease of use in drones, 196
education, applications for drones, 21
Einstein, Albert, 149
electric aircraft, 4
electronic speed controllers (ESCs), 54, 123, 131
 completing wiring to power distribution board, 130
 for brushless motors, 134
 BECs, 135
 mounting, 123
 plugging signal cable into throttle channel on RC receiver, 128
 PWM standard for control, 191
 soldering motors to, in S500 drone, 127

swapping wiring to get correct prop rotation, 129
testing, 127
wiring to power distribution board, 124
EMax 2216 motors with 1045 props and 3S batteries, 164
EMAX MT2216 II KV810 motors, 133
Ethernet connections, control systems and, 190
evaluateMSPCmd() function, 65
extended Kalman filters (EKF), 185

F

FAA, registration requirements for drones, 117, 170
failure mode and effects analysis (FMEA), 197
failure modes and fault tolerance, 183-185
failsafes, 183
redundancy, 185
fast and slow loops, 59
field effect transistors (FETs), 66, 100
choosing, 101
in brushless motor ESCs, 134
Flamewheel 450 and 550 quadcopters, 118
flight control systems (see control systems)
floating point numbers, 50
flying, 22
flight test for Visible Drone, 114
S500 drone, 161-170
connecting battery and voltage monitor, 164
final checklist and first flight, 169
flight modes, 168
initial tuning, 168
installing props, 162
radio and accelerometer calibration, 166
FMEA (failure mode and effects analysis), 197
FPV racing, 20
frames
building a drone with a large frame, 117
choosing for Visible Drone project, 55
completing in drone construction, 110
Fritzing, 98
FrSky radios, 191

FTDI adapter, connecting MicroMWC to your computer, 36
FTDI programmer, 32
future of drones, 195-197
ease of use, 196
human flight, 197
regulation, 196
safety, 196
specialization, 195

G

gimbal lock, 80
gimbals, 132
brushless gimbal motors, 133
Global Positioning System (see GPS)
GLONASS (GLobal NAVigation Satellite System), 148
GoPro cameras, 19
government regulation of drones, 196
GPS, 147-153
HDOP (horizontal dilution of precision), 150
history of, 148
in APM/ArduPilot, 11
installation, 151
NMEA 0183 standard for data, 193
theory, 148
use of GLONASS and GPS satellites by receivers, 148
gravity
determining direction of, 78
sensors in MARG systems, 52
GY-88 sensor module, 52
gyro/accelerometer chips, 15, 78
gyroscopes, 75, 76
advances in, 6
in MultiWii Manual mode, 45
Rate mode and, 85
vibrations affecting, 179

H

ham radio license, requirement for use of 433MHz band in US, 143
hardware-in-loop testing (HIL), 197

HB-02 serial Bluetooth module, 32
building cable to connect module to MicroMWC controller, 39

HC-06 Bluetooth module, 52
writing I/O code for, 63

HC-SR04 ultrasonic pinger module, 172

HDOP (horizontal dilution of precision), 150

headless mode, 81

heat shrink tubing, 123

HMC5883L three-axis magnetometer, 159

hot air soldering stations, 104

hovering, 24
hover time for different batteries, 27
hover time for different propellers, 28
translational lift and, 73

Hubsan X4, improving, 23-30
battery upgrade, 26
goals and test methods, 24
Hubsan X4 107C, 24
propeller upgrade, 27

human flight, 197

I

I/O (input/output) code, 57-67
for HC-06 Bluetooth module, 63
for motors, 66
main loop, 59
RC receiver, 60
setting up Arduino IDE, 57

I2C (inter-integrated circuit) interfaces, 189
communication errors with, 189
reading values from I2C bus, 79

IMU (inertial measurement unit), 52

IMU/AHRS, 75-81
accelerometers, 77
gyroscopes, 76
IMU, 78
magnetometers, 80

inner and outer loops, 59

integral (in PID algorithm), 88
integral windup, 90
tuning, 91

interfaces, 187-194
computer, 187

GPS, 193

RC radio standards, 190
servo/ESC control, 191
telemetry, 192

interrupt handler (serial events), 62

Invensense MPU-6050 six-axis gyro/accelerometer chip, 15

isolation (vibration), 179

K

Kalman filters, 5
extended, 185
using to make GPS work in practice, 149

Kelvin, Lord William Thomson, 157

KK2 flight controllers, 14

KKMulticopter boards, 14

Kv rating of motors, 133

L

laser interferometry gyroscopes, 76

LEDs
connecting to new controller, 107
testing on completed Visible Drone, 112

Lemon RX DSM2 receiver, 51

lidar, 173

Lidar-Lite series of modules, 173

lift, 69
translational, 73

LiPo (lithium polymer) batteries, 4
1S LiPos for Visible Drone project, 53
380 mAh, 25C, 1S LiPo with Hubsan X4 107C, 26
warning about underrated LiPos, 27

logic levels (voltage), 188

loiter mode, 168

M

magnetic field (Earth), 155

magnetic versus geographic north, 156

magnetic, angular rate, and gravity (MARG) systems, 52, 81

- magnetometer sensors, 72
magnetometers, 75, 80, 155-160
 declination and deviation, 156
 magnetoresistance, 158
 use in drones, 159
magnets in brushless motors, 131
 number of, 133
Mahony/Madgwick algorithm, 80
main loops, flight control systems, 59
Manual mode (MultiWii), 45, 84, 85
mapping, using drones, 20
MAVLink, 145, 192
maxAngle parameter, 84
Maxbotix HRLV-EZ4 sonar module, 173
metal oxide semiconductor field effect
 transistors (MOSFETs), 100
micro electrical mechanical systems (MEMS), 76
 in integrated circuit accelerometers, 77
MicroMWC flight controller, 31
 connecting to your computer using FTDI
 adapter, 36
 settings for connection to Arduino IDE, 35
 soldering battery connector onto, 37
 with accessories, 33
Mission Planner, 145
 tuning screen, 169
mix, 83, 85
MLP4DSM transmitter, 33
modes, 83-85
 Angle, 84
 flight modes for S500 drone, 168
 names for, in different control systems, 83
 Quantum i8 RC transmitter, 141
 Rate, 85
 RC transmitter, 83
 setting for X4Wii, 45
motor controllers, 54, 100
motors, 2
 brushless, 118
 brushless motors and their ESCs, 131-136
 choosing for Visible Drone project, 55
 connecting in drone construction, 108
 creating vibration, 178
 for S500 drone, determining spin direction,
 127
making S500 drone motors spin in desired
 direction, 129
mounting on S500 drone, 126
numbering for PixHawk Lite flight controller,
 141
soldering motor wires onto MicroMWC
 controller, 38
upgrading for Hubsan X4 107C, 30
writing code to control, 66
MSP (see MultiWii Serial Protocol)
MT2216 II KV810 motors, 133
multirotor aerodynamics (see aerodynamics,
 multirotor)
multirotors, 2
 MultiWii controllers, 10
MultiWii Configurator, 43
MultiWii control systems, 9
 Naze32/Baseflight/Cleanflight forks, 16
MultiWii Serial Protocol (MSP), 10, 64, 192
 ATTITUDE command, 64
 parsing incoming MSP messages, 64
 sending a message, 64
MultiWii/Baseflight/Cleanflight flight control
 software, 192
MultiwiiConf, 10, 43
 tuning modes, 45
 using to ground test Visible Drone, 113
Muñoz, Jordi, 12

N

- National Marine Electronics Association's NMEA
 0183 standard, 193
Naza flight control system, 14
Naze32 Configurator, 43
Naze32/Baseflight/Cleanflight, 16
NMEA 0183 standard, 193
NuttX operating system, 13
nylon 1045 props, 162

O

- OpenPilot project, 15
optical flow sensors, 171, 173

P

Paparazzi flight controller project, 17
parseRXData() function, 62
Phantom line of quadcopters (DJI), 14
photography, aerial, using drones, 14, 15, 19
PID controllers, 5
 adjusting, 169
PIDs (proportional–integral–derivatives), 32,
 87-94
 axisCmdPID, 86
 implementations of the algorithm, 90
 PID algorithm, 87
 derivative (D), 88
 integral (I), 88
 proportional (P), 87
 tuning, 90
 for X4Wii drone, 43
 in practice, 94
 in theory, 90-93
piezo-electric sensors, 77
PixHawk Lite flight controller, 137
 and shock mount, 138
 GPS installation, 151
polarity, checking, 127
position hold mode, 168
power
 constructing power system on circuit boards,
 104
 hazards of using USB and batteries together,
 112
 input and distribution for circuit board and
 motor controller, 99
 installing power system for S500 drone,
 123-130, 138
 completing power distribution board
 wiring, 130
 mounting the motors, 126
 wiring power distribution board, 124
 redundancy in power supply, 185
propellers
 balancing, 180
 creating vibration, 178
 hover time for different propellers, 28
 installing for S500 drone, 162

putting on X4Wii drone, 40
swapping ESC wiring to get correct rotation,
 129
 upgrading for Hubsan X4 107C, 27
proportional (in PID algorithm), 87
 tuning, 91
proportional–integral–derivative (PID) (see PIDs)
pulse position modulation (PPM), 141, 191
pulse width modulation (PWM), 66, 102
 for small, brushed motors, 66
Quantum i8 RC receiver, 141
 standard for servo/ESC control, 191
PX4 flight control systems, 192
PX4/Pixhawk flight controllers, 13
 diagram of APM/PX4 control systems, 13
PX4Flow optical flow sensor, 173

Q

QGroundControl, 145
accelerometer calibration screen, 167
compass calibration, 159
failsafe setup screen, 183
radio calibration screen, 166
quadcopters, 1
 drones versus, 2
Quantum i8 RC transmitter and receiver, 140
quaternions, 80

R

radios
 adding RC receiver to S500 drone, 128
 calibrating for S500 drone flight test, 166
 for Visible Drone project, 51
 I/O code for RC receiver, 60
 ground testing in Visible Drone, 113
 installing for PixHawk flight controller, 137
 RC radios, 140
 telemetry radio, 143
 installing on circuit board, 109
RC radio standards, 190
 DSM/DSM2/DSMX, 190
 FrSky, 191
RC radios for Visible Drone project, 51

RC transmitter modes, 83
testing installation of, 145
Rate mode, 85
RC (radio control) rates, 84
 changing RC values for X4Wii, 35
 tuning for X4Wii, 42
readIMU() function, 78
redundancies to handle failures, 185
regulation of drones, 196
resistors, 100, 101
return to land mode, 168
ring laser gyroscopes, 76
roll, defined, 71
roll, pitch, and yaw, 75
 calculating pitch and roll in Angle mode, 84
 getting values with MSP_ATTITUDE
 command, 64
 in MultiWii Manual mode, 45
PID settings for, X4Wii, 43
pitch and roll, 71
roll and pitch settings in MultiWii, 42
smoothing and computing the yaw rate, 85
Tait-Bryan angles of, 80
yaw, 71

S

S500 drone
 flying, 161-170
 frame, 117-121
 completed frame, 120
 following instructions, 119
 S500 drone frame kit, 118
installing the power system, 123-130
 determining direction of motor spin, 127
 mounting the ESCs, 123
 mounting the motors, 126
 wiring power distribution board, 124
safety switch, installing, 144
safety, trends in, 196
Sagnac effect, 76
satellite receivers, 51
sensors
 AMR, 158
 failure of, 185

in MARG systems, 52
multisensor systems, effects of vibration on, 179
new technologies, 171-175
 lidar, 173
 optical flow sensors, 173
 sonar, 172
 visual odometry, 175
serial interfaces, 187
 connectors, 188
Serial library (Arduino), 62
serial peripheral interface (SPI), 189
servos, standard PWM, 66, 191
shock mount, installing for PixHawk Lite
 controller, 139
slow loop, 60
smartphone programs, 43, 145
soldering surface mount components, 98
 hot air versus soldering irons, 104
solenoids, 101
sonar, 172
space weather, checking, 170
specialization in drones, 195
Spektrum, RC radios, 51
SPI (Serial Peripheral Interface), 189
stabilize mode, 84, 168
 (see also Angle mode)
states (in MSP), 64
Syma X5, 55

T

Tait-Bryan angles of roll, pitch, and yaw, 80
Tau Labs, 17
Teensy 3.2 processor, 50
 setting up Arduino IDE for use with, 57
Teensyduino package, 57
Teensyloader program, 57
telemetry data, 192
telemetry radios
 for Visible Drone project, 52
 installing 3DR 433 MHz radio set, 143
testing
 flight test of Visible Drone, 114
 ground testing Visible Drone, 112

hardware-in-loop testing (HIL), 197
installation of control system and radios, 144
test rig for PID implementations, 94
test stand for Hubsan X4 107C, 24
three-phase motors, 132
throttle, setting, 86
thrust, 69
pitch and roll and, 71
torque, differential, 71
translational lift, 73
tuning drones, 87
(see also PIDs)
initial tuning for S500 drone flight, 168
tuning parameters for S500 drone, 162

U

ultrasonic pinger module (HC-SR04), 172
USB, using to load new code, 113

V

vibrating structure gyroscopes (see Coriolis
vibratory gyroscopes)
vibration, 177-181
balancing propellers to mitigate, 180
causes of, 178
damping and isolation, 179
effects of, 178
resonating with the frame, 178
VisDroneBlink sketch, 112
Visible Drone, building, 49-56
circuit board with motor controllers, 54
frame, motors, and propellers, 55
IMU/AHRS, 52
power, 53
radios, 51
schematic and layout for controller, 102
test rig for PID implementations, 94
upgrading the CPU, 50

weight budget, 56
visual odometry, 175
voltage, 123
(see also batteries; power)
in serial connections, 188
voltage monitors, 165
vortex ring state, 74
vortex rotor state (VRS), 74

W

weather conditions for first flights, 170
weight budget (Visible Drone project), 56
Wii controllers, 9
Wire library (Arduino), 78
wiring
for S500 drone motors, not trimming, 126
safety switch, 144
telemetry radio, 143
writeMSP_ATTITUDE() function, 64

X

X4Wii, building, 31-47
assembly, 40
flashing new code, 35
parts, 32
setup and tuning, 42
modes, 45
PIDs, 43
RC (radio control) rates, 42
wiring, 37

Y

yaw, 71
(see also roll, pitch, and yaw)
PID settings for, X4Wii, 43

About the Author

David McGriffy has been a programmer since the earliest days of the personal computer. He has piloted both radio-controlled and human-carrying aircraft, including gliders of both sorts. He worked on GPS systems before they had even finished launching all the satellites. He's written embedded systems code for top computer companies and for fun. He has traveled the world to do failure-mode analyses on the control systems of offshore oil rigs and is an inventor on a patent for a vibration analysis system. He is an expert on website performance and has been published by IEEE on the subject. David runs an audio software company, VVAudio, which produces plugins for surround sound and virtual reality. He has a Ph.D. in Physics from the University of Texas at Austin.

Colophon

The cover image is from David McGriffy. The cover and body font is Myriad Pro; the heading font is Benton Sans; and the code font is Ubuntu Mono.