Intro: RC Quadrotor Helicopter
This project is a RC quadrotor helicopter (quadrocopter, quadcopter, quadricopter, etc). It's a RC helicopter that uses 4 rotors.

Watch it in 720p

You need some pre-requisite skills:

- How to use Arduino, enough to get started
- Soldering, wiring, basic electronic skills
- Basic hand tool operation

A quadrotor helicopter flies with 4 spinning propellers on a + shaped frame.

When one rotor spins faster than the rotor on the opposing side, the faster side will have more lift, and thus the helicopter will tilt. When the helicopter is tilted, the air is being blown slightly sideways instead of directly down, and the helicopter will move.

The propellers also need to be in counter-rotating pairs, two spin clockwise and two spin counter-clockwise. This way, the helicopter does not spin on the vertical axis since the rotational energy is cancelled out. But when the pair that’s spinning in one direction is faster than the other pair, the helicopter will spin on the vertical axis. This is how the helicopter controls its direction.

We will be building a flight controller circuit that contains an accelerometer and gyroscope sensor so that a microcontroller can detect undesired changes in the helicopter’s angle, and adjust each rotor’s speed accordingly to counter the variation. This microcontroller will do this hundreds of times per second, keeping the helicopter stable in the air.

The flight controller is a completely open source circuit. The circuit schematic and PCB files are provided. The flight controller is completely Arduino compatible. The source code is a modified version of AeroQuad (open source Arduino based quadrocopter control code). The flight characteristics can be adjusted using AeroQuad's configurator utility.

Attached is a diagram that shows you the direction of spin for each motor, remember this diagram! You won’t be able to fly if your setup does not follow this diagram.

This microcontroller will also take input from a RC radio receiver, so you can control the helicopter from the ground using a RC radio transmitter.

This helicopter will use four brushless motors. Each motor will be controlled by an ESC (Electronic Speed Controller). The ESCs will be controlled by the microcontroller.

A lithium polymer battery will power the entire contraption.

Summary of Downloads:

- Over 100 pictures in all the steps
- Step 9 contains flight controller circuit and PCB files
- Step 10 contains bootloader and core for microcontroller
- Step 12, 13, 14 contains demo Arduino sketches
- Step 26 contains the flight control software
Image Notes
1. flight controller
Step 1: Parts

RC radio transmitter and receiver
You need 4 channels minimum but I am begging you to get one with 6 channels. Also get one that uses 2.4 GHz technology if you can. Turnigy has a 9 channel model that is actually very inexpensive, and it runs on an AVR microcontroller that you can put custom firmware on. Personally, I have a old $25 radio that uses 75 MHz FM but I’ve converted it into a 2.4 GHz radio using a conversion kit.

Four brushless outrunner motors are needed. I used hexTronics 20-22L (this number represents the diameter and coil winding configuration of the motor, there’s also a kv value that relates speed and power, higher kv = more powerful faster motor (thanks to the correction from slick8086, I was incorrect to relate power to Kv, it's the RPM that's faster if Kv is higher), 800 to 1200 is acceptable motors. They come with a ton of parts (the bullet connectors, heatshrink, propeller adapter, the screws you need, mounting plate, spare shaft and spare C-clip, all included). Get extras just in case one gets damaged in a crash.

Four brushless motor electronic speed controllers (ESC) are needed. One that is rated for 18 amps is enough. I have heard good things about the Turnigy Plush ESCs because they support high update frequencies (more frequent adjustments means more stable flight). I got the HobbyKing brand clones of the Turnigy Plush ESCs because they are cheaper.

Some ESCs are “card programmable”, meaning you can change their configuration using a cheap ($6) programming card, which is really convenient. Buy the programming card that is compatible with the ESCs you’ve chosen. I got the Turnigy ESC programming card because they are compatible with my ESCs.

You'll obviously need a battery. You will use a 3S1P lithium polymer battery that is rated at least 20C (this is the discharge rating). 3S means 3 cells in series, 1P means one set in parallel. This will give you a combined 11.1 volts. I suggest a 2500 mAH capacity battery (or more). A general rule of thumb is doubling the capacity of the battery means 50% increase in flight time (due to the extra weight).


Make sure you pay attention to the type of connector that comes attached to your battery. You're going to need to get the matching connector. I've personally built my entire helicopter using bullet connectors. 4.0mm bullet connectors for the battery and 3.5mm bullet connectors for everything else (the motors come with 3.5mm bullet connectors). (You can get other connectors such as XT-60 connectors , just watch your polarities, also note that all my pictures show bullet connectors)

You need a lot of heat-shrink tubing to act as insulation when using the 3.5mm bullet connectors. Get different colours so you can tell which wire is which.

You need 12 gauge stranded core wire . It must be 12 gauge or thicker to handle the current. It must be stranded core so that it is flexible. Get different colours so you know which wire is which polarity. The best wire is fine stranded copper wire with silicon insulation, but this is expensive.

Get a good battery charger , it must be able to balance and charge multi-cell lithium polymer batteries. I have one of these Turnigy chargers that have many settings, a LCD, and cooling fan, very nice. I also use a laptop power brick to power the charger since an ordinary wall adapter won't be able to handle the current required.

Get a battery monitor so you know when your battery is low. A lithium battery will become permanently damaged if you drain it below a certain threshold. Having a monitor will prevent you from damaging your battery. I have one of these that reports the status of each cell.

The quadcopter's frame I use is this one from HobbyKing . For $15, you get every part you need, plus every screw and nut you need. In comparison, a stick of aluminum would cost me $10 from Home Depot, which is not economical in comparison. I suggest you buy several frames so you have plenty of spare "arms" if you crash (plus plenty of spare screws and nuts).

The propellers must be in counter-rotating pairs (a "pusher" and a "puller"). I use 10x4.7 APC slow-fly propellers . 10 indicates diameter in inches and 4.7 indicates pitch. Larger diameter means more lift but requires more powerful motors. 10 inches is about right for the frame size I am using.

You'll need a entire flight controller circuit (meaning another entire bill of material), I will talk about this in detail later. Meanwhile, you'll need a USB-to-serial cable ( FTDI cable ) and an AVR programmer.

Plenty of servo cables are required to connect stuff. A minimum of 6 female-to-female cables is required for the 6 channels from the RC radio receiver to the flight controller.

Velcro tape and double sided velcro strapping will be useful as a lightweight way of mounting things and strapping in the battery.

Get a bubble/spirite level (like this one ) to help with sensor calibration.

Make sure you get plenty of extras, stock pile on wire, cables, connectors, heat-shrink tubing, electrical tape, glue, screws, etc.

Step 2: Bullet Connector Soldering

You'll be doing a lot of soldering, especially with these rather large connectors. I'm writing this section because of the special connectors we'll be using requires some special soldering techniques. There's a lot of current going through these wires, and they have to withstand crashes and you yanking on them. So you should do this properly.

You need a good hot soldering iron for this, since the connectors are rather large and act as a heatsink.

First, take a piece of wood, and drill some holes into it to hold your connector while you solder. This is your soldering jig.

Strip your wire, the length of exposed wire should be as long as the cavity in the connector's end. If you are soldering the 4mm connectors with the polarized housing, make sure the housing goes on first before you solder (see pictures).

Place the connector into the hole in the wood. Then stick your soldering iron's tip into the small hole on the side of the connector. Fill the cavity with solder (fill the cavity about 60%). While the solder is hot, put the wire into the cavity, continue heating for a few seconds until you are sure the wire has reached thermal equilibrium with the solder, and then remove the iron. Hold the wire still until the solder cools and solidifies.

Wait until the entire connector is cool enough to touch. Clean the solder joint with rubbing alcohol.

If you are using the plastic housing, pull on the wire into the housing until the connector clicks into the housing. If you are not using the housing, then use heat-shrink tubing to insulate the connector. See pictures.

Also relevant links (not created by me):

- [http://www.youtube.com/watch?v=B9yY9Kk4bEA](http://www.youtube.com/watch?v=B9yY9Kk4bEA)
- [http://www.youtube.com/watch?v=MnldHRjre8](http://www.youtube.com/watch?v=MnldHRjre8)

Image Notes
1. A wooden jig with holes to hold the connectors during soldering.
Step 3: Preparing the Motors

If you got the same motors that I did (hexTronics 20-22L), then you'll notice that the shaft sticks out the bottom of the motor. I cut this off with a cutoff-disk attached to a rotary tool. This is because I don't want the motor to be damaged from a minor bump with the ground.

When you are cutting off the shaft, make sure you first wrap the entire motor with tape first to prevent metal shavings and debris from entering the motor. Dust and debris can damage the windings and bearings. Also when you are done, thoroughly clean the motor by spraying it with compressed air.

Assemble the motor, attach the large propeller adapter, and the bottom mounting plate, using the appropriate screws that are provided.

The motors comes with 3.5mm bullet connectors and heat-shrink. Solder the male connectors to the motor. The female connectors are meant for the ESC (we'll do that later).
Step 4: Understanding Brushless Motors

It is important for you to understand how brushless motors and ESCs work, so that you don't destroy your ESCs.


To reverse the direction of motor spin, DO NOT REVERSE THE POWER INPUT TO THE ESCs.

You'll note that there are three wires that goes between the motor and the ESC. These motors are three phase motors, meaning there are three coils inside. The coils are energized in sequence to make the motors spin. So the ESC's job is to energize the coils in sequence, but it needs to time each energization correctly so the motor can actually accelerate to the right speed. The ESC has a microcontroller inside that turns on or off the coils using FETs and also determines timing by measuring the feedback in the coils caused by the movement of the magnets.

Now you should understand that to reverse the direction of spin of the motor, you can swap any two of the three wires between the motor and ESC. To visualize this, if the original sequence was:

```
123 123 123
```
and we swap 2 and 3, we get:

```
132 132 132
```

and thus the direction of spin is reversed.

Also, the ESCs contains a 5V voltage regulator (most likely a cheap 7805), and a low battery cut-off feature. Our flight controller will use this 5V power supply.

The cut-off feature means when your lithium polymer battery is drained, it will stop spinning the motor to protect the battery. This is because permanent damage can occur to your battery if it is drained below a threshold. But when the motor stops spinning, the voltage regulator still provides the 5V. This is so that servos on an airplane can still function using that 5V supply, and the RC airplane can still glide to a safe landing even without an engine.

However, in our helicopter situation, if one motor dies, the helicopter will spin out of control in the air. It will become a safety hazard to everybody around it. Thus, we set the cut-off voltage to "low" so we maximize flight time. We rely on our battery monitor instead, so we avoid over-draining the battery.

our flight software to use them since the code is already available. However, for the rest of this instructable, I'm still using PWM controlled ESCs.

![Diagram of delta and wye configurations]

**Step 5: Preparing the ESCs**

Solder the female 3.5mm bullet connectors to the ESC's output wires. Also solder connectors to the power input wires of the ESC. Make sure you keep track of what polarity of your connectors are so you don't connect positive to negative.

Also, please load the following settings to all 4 ESCs using the programming card:

- Brake: Off
- Battery Type: Li-xx
- Cut Off Type: Soft-Cut
- Cut Off Voltage: Low
- Start Mode: Normal
- Timing Mode: High
- Li-Po Cells: 0 (meaning auto-detect)
- Governor Mode: Off

Also see these links (not by me):

- [http://www.youtube.com/watch?v=4Kg7MzGwvdw](http://www.youtube.com/watch?v=4Kg7MzGwvdw)
Step 6: Preparing the Battery Monitor
This step is optional but I recommend it so you never plug in your battery monitor backwards.

Get some 4 pin JST connectors to replace the existing connector on your battery monitor.

I also used a permanent marker to indicate the connection polarity.

Place some velcro on the battery monitor so you can mount it to the helicopter easily. This is optional, I usually just leave it dangling off the battery.

Watch the monitor all the time. As soon as one of the lights go red, then land as soon as possible. Draining any one of the three cells below the threshold will cause permanent damage to that particular cell.

The flight controller also has battery monitoring capabilities, but it does not monitor all 3 cells like this one does. Other battery monitors have a buzzer which is better than lights, but they also do not monitor individual cells.
Step 7: Making the Power Distribution

This is sometimes called a "power spider".

You need to split the power from the battery into the four ESCs. Solder some of your 12 gauge wire into a spider-like shape to distribute the power. You need two of these "spiders", one for positive, one for negative. I suggest you colour code the positive as red and the negative as black (or blue).

One pair of the wires should be significantly longer, this is the wire that connects to the battery. I have cut this wire and placed a pair of 3.5mm bullet connectors there so I can swap out the battery connector if I decide to buy a battery with a different connector.

Make sure you insulate these two sets from each other. I placed the center solder joint into a bottle cap and filled it with hot glue.

Solder 3.5mm bullet connectors to the split-off ends so you can connect to your ESCs, make sure you don’t screw up the polarity.

On my setup, I added a 5th wire pair just in case I need it later (maybe to power a camera). I suggest you do the same, but note that the male bullet connector will be exposed, so make some "insulated dummy connectors" (the yellow things in my pictures) with spare connectors and heat-shrink to put over the exposed connector.

There is no power switch in this setup. A switch that has a decent current rating would be too heavy. To turn off the helicopter, just disconnect the battery, which is a good habit anyways.
Image Notes
1. Plenty of heat-shrink tubing is the key to making a safe connection
**Step 8: Flight Controller Overview**
The flight controller is a system that uses a microcontroller to control the quadcopter. It needs to take inputs from the user via a RC radio receiver, input from a accelerometer and gyroscopic sensor. Its outputs are to the four ESCs.

The sensors being used are the BMA180 3-axis accelerometer, and the ITG-3200 3-axis gyro. Both of these are available in a breakout-board format from SparkFun so you don't have to solder the chips. Note that both of these are digital sensors, this means they both have internal analog-to-digital converters (ADC) that perform better than the integrated ADCs of most microcontrollers. Both of these sensors use the I2C bus, which means only two wires are required to interface both sensors. The I2C bus can run at 400 KHz.

The RC radio receiver will have 6 or more channel signal pins, each signal will be connected to one pin on the microcontroller. These signals will be pulse width modulated signals, so the microcontroller will measure the width of each pulse to take input from the user.

The ESCs each expect a pulse width modulated signal. One pin on the microcontroller will be connected to one ESC. The microcontroller will output pulses to each ESC to control their individual speeds.

Although an ordinary Arduino would have enough performance as the microcontroller, I decided to use an ATmega644P in my design so I have more memory and more pins to experiment with. My design still uses 16 MHz, uses the Arduino bootloader, and it is compatible with the Arduino IDE.

The ATmega runs at 5V because 5V is required at 16 MHz, this 5V power supply will be provided by the ESCs since they all have a built-in 5V voltage regulator.

The sensors run on 3.3V and thus our circuit will contain a 3.3V low-dropout voltage regulator that converts the 5V to 3.3V so we can safely interface with the sensors.
Step 9: Flight Controller In-Depth
I have previously designed this circuit: Ro-4-Copter
It's hosted on Google Code, I am the sole author of all the wiki pages, the circuit, PCB, and bill-of-materials. The code is heavily modified versions of AeroQuad and MultiWiiCopter with all original copyright notices, but also including modifications by me.

You may want to replicate it, all the design files (I've included the Gerber files too) and bill-of-materials are provided. I am also in the process of designing a "retail" version, which may be sold once I test the design and finalize it.

See wiki article about how to assemble my flight controller design, it's written by me, and the part-list is provided here.

To make the PCB, generate the Gerber files from the Eagle CAD PCB files, there are plenty of guides on the internet showing you how to do this. Here's the SparkFun tutorial. I usually send the Gerber files to Seeed Studio's Fusion service (10 PCBs for $28, takes 5 days to make + shipping time).

Below is a short version of the assembly guide, for more detail, see the full guide (linked above).

Note that all resistors are in 0805 packaging, although 0603 will also fit perfectly fine. They should be 1/16 watt or greater. Note that all capacitors, unless otherwise specified, are ceramic chip capacitors in 0805 packaging (or 0603 since they will also fit perfectly fine). They should be rated 16V or greater.

This assembly guide also contains some notes about the various areas on the circuit.

Microcontroller
The ATmega644PA obviously must be installed, solder in a 40 pin DIP socket so you are able to replace the IC if you need to.

The ATmega644PA requires a 16 MHz resonator.

You must solder C1 and C2, which are 0.1 uF decoupling capacitors for the ATmega644PA. Also solder C9 which is a 0.1 uF decoupling capacitor for the analog reference.

On the bottom of the board, there is a footprint labelled "AREF-SEL", which allows you to choose the analog reference voltage by soldering over the pads labelled 5V or 3V. You may also use a 0 ohm resistor here, or an ferrite bead. Note that AVCC is always connected to 5V.

Microcontroller Supporting Areas
The reset switch must be installed, R13 10K ohm is the pull-up resistor for the reset switch. This resistor is optional because the ATmega644PA has a built-in pull-up resistor on the reset pin. You should install this if you do not trust the built-in resistor.

A push button labelled "BOOT" can be installed only if you are using a non-Arduino-standard bootloader, I personally hate the way "time sensitive" bootloaders work. This button is active low and connected to PB6 on the ATmega. Be sure to enable internal pull-up resistors for this button.

The ISP header (labelled AVRISP) should be installed if you need to use an AVR programmer (perhaps to load the bootloader for the first time, etc). The long line at the side indicates the side that the "red wire" (also pin 1) of the cable should be, and the short line at the bottom indicates where the notch of the connector should be.

LED Indicators
LED1, LED2, LED3 are 3 mm LEDs, choose your own colours. They can be mounted vertically or horizontally (pointing out). Please watch the polarity indicated by the hole shape and silkscreen. R3, R4, and R5, are all 1K ohm resistors to limit current to these LEDs.

Power Supply
Solder a LM1117 3V3 regulator in SOT-233-3 packaging where indicated on the PCB. Also solder the accompanying C3, which is 1 uF.

LED-PWR is a 3 mm LED that indicates that there is power present on the 3V3 supply bus. You may solder this vertically or sideways (pointing out). R12 is the 1K ohm current limiting resistor for this LED.

Two rows of 3 pin male headers should be installed at "3V3-TAP", these headers give you access to the 3V3 supply bus. Two rows of 3 pin male headers should be installed at "5V-TAP", these headers give you access to the "SYS" 5V supply bus.

If you are powering this circuit from a battery, you must install a LM1117 5V regulator in SOT-233-3 packaging where indicated on the PCB. This may be omitted if you are using the ESCs as the power supply (be sure to set SJ1 correctly), or under other circumstances that you are under.

The 1 uF capacitor C4 should be always soldered regardless of whether or not you are using a 5V regulator.

The battery power input is used if you need to connect a battery. There are four ways to do this:

- Using a barrel jack
- Solder wire directly onto PCB (you can enlarge some holes to act as strain relief)
- using a screw terminal
- Using a 3 pin header

A beefy diode should be installed at D2 to protect the circuit from a battery connected backwards (so omit this diode if you believe in your own competence, you must bridge the pads of the diode footprint with solder if you omit this diode). This must be a diode that can handle 20V of reverse voltage, and handle 1.5A of forward current. The footprint is SMB or similar.

C5 and C6 are aluminium electrolytic 5.3mm x 5.3mm capacitors that must be installed, solder them in the correct orientation according to polarity.

RC Input
PORTC's pins PC7 to PC2 are connected to 6x3 headers, solder in male headers here. These are dedicated to taking RC pulse signals from your RC receiver. The voltage for the center pin can be connected to 5V or 3V using SJ3, and remember that there is a "5V-TAP" and "3V-TAP" if your receiver needs power from a 7th cable.

Important: If you select V-RC with SJ15 (to use the same voltage that the RC inputs are using) and you have selected 3V3 using SJ3, then you must not use SJ17 to select anything. Doing so will cause the 5V supply to be joined with the 3V3 supply, which can cause damage to 3V3 devices.

Sensors
Solder R1 and R2, which are 4K7 pull-up resistors for the I2C bus. Be sure that you disable internal pull-up resistors in the software (note: already done for you in my modified software). This allows you to use 3V3 I2C sensors without using a level shifter.

Install a BMA180 accelerometer breakout board and a ITG-3200 gyro breakout board, both from SparkFun.
You may also choose to install a BMP085 barometric sensor breakout board, and a HMC5842 compass breakout board. These are optional and you can configure the software to use them for altitude-hold and compass-heading-hold.

**Outputs to Motors and Servos**

Four sets of 3-pin male headers should be soldered near "ESC" labels, these are connections for standard RC pulse signal ESCs. SJ1 can be used to connect the voltage supplied by the ESCs to the main system supply bus. (note, for this instructable, please connect SJ1)

**Serial Ports**

There are a few possible serial port devices. The ATmega644PA has two serial ports, USART0 and USART1, they are labelled SER0 and SER1 on the PCB.

Use horizontal mount male headers for the FTDI cable connection. This connection is always powered by 5V, RX and TX are at 5V level. Use SJ9 and SJ10 to decide whether to connect this to SER0 or SER1. (note, for this instructable, select SER0)

### Part List

<table>
<thead>
<tr>
<th>Part</th>
<th>Other Attributes</th>
<th>Identifiers</th>
<th>Quantity Required</th>
<th>Example Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATmega644PA</td>
<td>40 pin plastic DIP</td>
<td>IC1</td>
<td>1</td>
<td>Digikey: ATMEGA644PA-PU-ND</td>
</tr>
<tr>
<td>40 pin DIP socket</td>
<td></td>
<td>IC1</td>
<td>1</td>
<td>Digikey: A411-ND</td>
</tr>
<tr>
<td>4K7 ohm resistor</td>
<td>0805 or 0603 packaging, 1/16 W minimum</td>
<td>R1, R2, R9, R10</td>
<td>4</td>
<td>Digikey: P4.7KACT-ND</td>
</tr>
<tr>
<td>10K ohm resistor</td>
<td>0805 or 0603 packaging, 1/16 W minimum</td>
<td>R8, R11, R13</td>
<td>3</td>
<td>Digikey: P10KACT-ND</td>
</tr>
<tr>
<td>15K ohm resistor</td>
<td>0805 or 0603 packaging, 1/16 W minimum</td>
<td>R6</td>
<td>1</td>
<td>Digikey: P15KACT-ND</td>
</tr>
<tr>
<td>7K5 ohm resistor</td>
<td>0805 or 0603 packaging, 1/16 W minimum</td>
<td>R7</td>
<td>1</td>
<td>Digikey: P7.5KACT-ND</td>
</tr>
<tr>
<td>1K ohm resistor</td>
<td>0805 or 0603 packaging, 1/16 W minimum</td>
<td>R3-5, R12</td>
<td>4</td>
<td>Digikey: P1.0KACT-ND</td>
</tr>
<tr>
<td>1 uF capacitor</td>
<td>ceramic, 0805 or 0603 packaging, 16V rating minimum</td>
<td>C3, C4</td>
<td>2</td>
<td>Digikey: 311-1358-1-ND</td>
</tr>
<tr>
<td>0.1 uF capacitor</td>
<td>ceramic, 0805 or 0603 packaging, 16V rating minimum</td>
<td>C1, C2, C7, C8, C9</td>
<td>5</td>
<td>Digikey: 311-1361-1-ND</td>
</tr>
<tr>
<td>33 uF capacitor</td>
<td>electrolytic, 5.3 mm x 5.3 mm SMD aluminum can, 16V rating minimum</td>
<td>C5, C6</td>
<td>2</td>
<td>Digikey: PCE3886CT-ND</td>
</tr>
<tr>
<td>1N4148 diode</td>
<td>MELF packaging</td>
<td>D1</td>
<td>1</td>
<td>Digikey: LL4148DICT-ND</td>
</tr>
<tr>
<td>Rectifier Diode</td>
<td>SMB packaging, 20V reverse voltage minimum, high forward current rating</td>
<td>D2</td>
<td>1</td>
<td>Digikey: SSB44-E3/2TGICT-ND</td>
</tr>
<tr>
<td>16 MHz resonator</td>
<td></td>
<td>Y1 (16 MHz on PCB)</td>
<td>1</td>
<td>SparkFun: COM-09420</td>
</tr>
<tr>
<td>Male headers</td>
<td>0.1 inch pitch</td>
<td>ESC, I2C, I2C-GROUP, BARO, ACCEL, IMU, GYRO, MAG, BATTERY, BATT-MON, ADC1-7, RC1-6, 5V-TAP, 3V3-TAP, SERIAL, FTDI, maybe BLUETOOTH</td>
<td>Buy long strips and break them off</td>
<td></td>
</tr>
<tr>
<td>2x3 male headers</td>
<td>0.1 inch pitch, no shroud allowed</td>
<td>AVRISP</td>
<td>1</td>
<td>Use the male headers you buy</td>
</tr>
<tr>
<td>TXB0101 bidirectional level shifter</td>
<td>SOT-23-6 package</td>
<td>IC4, IC5</td>
<td>2</td>
<td>Digikey: 296-21664-1-ND</td>
</tr>
<tr>
<td>Tactile momentary push button switch SPST</td>
<td>Right angle, 7.50mm x 7.10mm</td>
<td>RESET, BOOT</td>
<td>2</td>
<td>Digikey: P12232SCT-ND</td>
</tr>
<tr>
<td>5V regulator</td>
<td>SOT223, SOT-223-3 package</td>
<td>5V-REG</td>
<td>1</td>
<td>LM1117MPX-5.0</td>
</tr>
<tr>
<td>3V3 regulator</td>
<td>SOT223 package</td>
<td>3V3-REG</td>
<td>1</td>
<td>LM1117MPX-3.3</td>
</tr>
<tr>
<td>LED</td>
<td>3 mm, any colour</td>
<td>LED1-4</td>
<td>4</td>
<td>SparkFun: COM-00533</td>
</tr>
<tr>
<td>Barrel jack</td>
<td>Battery</td>
<td></td>
<td>1</td>
<td>SparkFun: PRT-00119</td>
</tr>
</tbody>
</table>

If you just want something that flies and don’t care about the details, then just build the circuit and upload the code. If you want to know how I designed this circuit, continue reading this step.

The first step is to draw out the circuit.

I started with the microcontroller. I knew it needed a reset button, a clock source (16 MHz ceramic resonator), a 6-pin header so I can program it with an AVR programmer, a serial port connection for the Arduino bootloader, and also 5V of power.

The sensors are connected to the I2C pins of the microcontroller. I2C busses require 4.7 kilo-ohm pull-up resistors to 3.3V to work. I can’t use the ATmega’s internal pull-up resistors because those go to 5V, which will damage the sensors.

The 3.3V voltage regulator is connected to the 5V power supply. Some capacitors are added in various spots to decouple and filter.

Connections to the RC radio receiver and the ESCs are simply male header pins connected to the microcontroller. The three pins are always "signal, +5V, ground".

This gave me a basic circuit. After, I altered it to add features such as indicator LEDs, solder-jumpers for configuration, xBee socket, etc.

Laying out the PCB is also simple.

I start out with a square outline. Then I added slots to each corner of the square so I can mount it to any frame. Although I highly suggest you design the mounting hole patterns to suit whatever frame you get.

Next, I put down a ground plane on both sides.

The sensors need to be placed in the correct orientation, as you can see, they are at a 45 degree angle.

The microcontroller is sort of the center piece. Header pins are placed where they are needed. The ESC pins are placed in each corner of the square to make them easy to identify later. The RC radio receiver input pins are placed in order to make them easy to connect. Any external connections such as the FTDI serial port are placed on the edge to make it easy to connect/disconnect. LEDs are also placed on the edge to make them more visible.

Remember that:

- Power rails might require a thicker trace on the PCB
- Avoid sharp corners
- Place capacitors physically close to the component they are meant for
- Label as much as possible
- Just because your PCB manufacture says they can do 6 mil trace thickness or clearance, doesn’t mean you should try
- 0805 sized surface mount components have pads that are as big as through hole components, so to save space and weight, I suggest you use surface mount components. For the big ATmega644P, I used a through hole version because I wanted to use a chip socket so it’s replacable.

When you think the design is ready, print out a 1:1 scale copy on paper to make sure that all the text will be readable, and all the component footprints are correct.

Keep tweaking the design until you are ready for manufacturing the PCB. Export the Gerber files and send them to the manufacture. I usually use Seeed Studio who will make ten copies for $28 + shipping within 5 days + shipping.

Solder your circuit together and it should be good-to-go.
Step 10: Preparing the Microcontroller

If you don't have the Arduino IDE, download it and install it. I am using version 0022.

The following steps are explained in detail in this wiki article: http://code.google.com/p/ro-4-copter/wiki/GeneralSoftwareSetup
This page is just a really short summary of what is already in the above wiki article!

The Arduino bootloader should be programmed into the microcontroller first. This step requires the use of an AVR programmer. However, after the bootloader is flashed into the microcontroller, you will only need an USB-to-serial cable (such as an FTDI cable) to upload code from the Arduino IDE.

The bootloader (I've attached it) is modified slightly since I'm using the ATmega644P instead of the usual Arduino's ATmega328P. The files are provided. The fuse-bits need to be written to the correct values as well.

AVRDUDE is the utility that we use to flash the bootloader .hex file into the microcontroller. The AVRDUDE commands you'll need is

```
"avrdude -c programmer_name -p atmega644p -U flash:w:bootloader_filename.hex -U lfuse:w:0xFF:m -U hfuse:w:0xD8:m"
```

Pay attention to the programmer_name and the bootloader_filename! The fuses are to get the ATmega to use the 16 MHz resonator, set the bootloader size, enable the bootloader, disable JTAG and enable SPI downloading.

Also copy the bootloader folder "Ro4Copter_Boot" (remember to rename it without the date version) as .../arduino-0022/hardware/arduino/bootloaders/Ro4Copter_Boot.

Also, download the Arduino "core" I've provided here (called Ro4Copter_Core, remember to rename it without the date version), and place it into the "core" directory within Arduino so you get .../arduino-0022/hardware/arduino/cores/Ro4Copter_Core. Also modify the boards.txt file to include an entry to use this core. This will allow the Arduino IDE to compile for the ATmega644P instead of the original Arduino's ATmega328P. Be sure to select the correct board entry inside the Arduino IDE menus. For more details on this step, check out the wiki page I linked above.

The board.txt entry:

```
Ro4Copter.name=Ro4Copter
Ro4Copter.upload.protocol=stk500
Ro4Copter.upload.maximum_size=57344
Ro4Copter.upload.speed=57600
Ro4Copter.bootloader.low_fuses=0xFF
Ro4Copter.bootloader.high_fuses=0xD8
Ro4Copter.bootloader.extended_fuses=0xFF
Ro4Copter.bootloader.path=Ro4Copter_Boot
Ro4Copter.bootloader.file=Ro4Copter_Boot_arduino.hex
Ro4Copter.bootloader.unlock_bits=0x3F
Ro4Copter.bootloader.lock_bits=0x0F
Ro4Copter.build.mcu=atmega644p
Ro4Copter.build.f_cpu=16000000L
Ro4Copter.build.core=Ro4Copter_Core
```

Here's a screenshot to show you where all these folders are:

The sketch you need to compile is provided in step 26. Compile it and upload it to the ATmega644P using the bootloader later.

The next few steps will help you understand what exactly the code is doing. Skip them if you don't care.
Step 11: Explanation of Source Code

As with all Arduino sketches, there is a setup routine and a main loop. The setup initializes everything, and the main loop performs the input, calculation, and output.

The PWM signals from the RC radio receiver is taken in using "pin change interrupts". When the voltage on the input pin changes, the interrupt routine is called, and a time stamp is taken. The pulse width is calculated from the differences between the time stamps.

The PWM signal output to the ESCs is generated using several timers with "compare match outputs", which means the pulse width is set, and the timer will automatically pulse the pin for the set period of time.

The sensors communicate using the I2C bus, which is a synchronous data bus that has excellent support for multiple devices communicating using only 2 wires.

The main loop is divided into tasks, each of which is executed at a different frequency. This gives each task a priority. The code that keeps the quadcopter stable has the most priority. Stuff like communicating with the computer has less priority. It is possible for you to add GPS navigation code if it is given lower priority.

The later steps will have demonstrations that specifically demonstrate each technique and technology individually.
Step 12: Arduino Demo: PWM Input

Here is a Arduino sketch that shows you exactly how pin-change-interrupts and a timer is used to read in pulse widths from several signal wires. Whenever a change in pin state is detected, the interrupt vector records a timestamp using the timer, and the difference between timestamps is the pulse width.

Also I've attached a logic analyzer screenshot to show you what the signals from my RC radio receivers look like. Each one of those pulses have a pulse width between 1000 and 2000 microseconds, the period is about 20 milliseconds.

For more information, research:

- PWM
- Servo Signals
- ISRs in AVR (Interrupt Service Routine)
- PCINT, Pin Change Interrupt
- 16 bit timers in AVR

Here is the code:

```c++
void setup()
{
  DDRC = 0; // pins as input

  // enable PCINT 18 to 23
  PCICR |= (1 << PCIE2);
  PCMSK2 = 0xFC;
  Serial.begin(115200);
}

typedef struct {
  byte edge;
  unsigned long riseTime;
  unsigned long fallTime;
  unsigned int lastGoodWidth;
} tPinTimingData;
volatile static tPinTimingData pinData[6 + 1];
volatile static uint8_t PCintLast;

ISR(PCINT2_vect)
{
  uint8_t bit;
  uint8_t curr;
  uint8_t mask;
  uint32_t currentTime;
  uint32_t time;

  // get the pin states for the indicated port.
  curr = PINC &0xFC;
  mask = curr ^ PCintLast;
  PCintLast = curr;

  currentTime = micros();
  
  // save the time the event occurred.
  time = currentTime - pinData[bit].riseTime;
  if (time > pinData[bit].fallTime - pinData[bit].riseTime) {
    pinData[bit].riseTime = time;
    pinData[bit].edge = (mask & PCINT assertions);
    pinData[bit].lastGoodWidth = 0;
  }

  if (pinData[bit].edge & PCINT assertions) {
    pinData[bit].fallTime = time;
    pinData[bit].lastGoodWidth = time - pinData[bit].fallTime;
  }
}
```

for (uint8_t i=0; i < 6; i++) {
    bit = 0x04 << i;
    if (bit & mask) {
        // for each pin changed, record time of change
        if (bit & PCintLast) {
            time = currentTime - pinData[i].fallTime;
            pinData[i].riseTime = currentTime;
            if ((time >= 10000) & (time <= 26000))
                pinData[i].edge = 1;
            else
                pinData[i].edge = 0; // invalid rising edge detected
        } else {
            // time = currentTime - pinData[i].riseTime;
            pinData[i].fallTime = currentTime;
            if ((time >= 800) & (time <= 2200) & (pinData[i].edge == 1)) {
                pinData[i].lastGoodWidth = time;
                pinData[i].edge = 0;
            }
        }
    }
}

void loop() {
    Serial.println();
    for (byte i = 0; i < 6; i++) {
        Serial.print("C");
        Serial.print((int)i + 1);
        Serial.print("C");
        Serial.print((int)1);
        Serial.print("*");
        Serial.print(pinData[i].lastGoodWidth);
        Serial.print("*");
    }
    delay(500);
}
Step 13: Arduino Demo: PWM Output

Attached is an Arduino sketch that shows you how our code generates PWM signals so each ESC knows how fast to spin the motor.

The timer compare output feature of the timer is used to generate PWM signals. When the timer reaches 0, it turns on a pin, when it reaches a certain number (that we specify), it turns off the pin. This generates a square wave with a variable duty cycle.

Although normal servo signals usually have a period of about 20 ms (meaning a frequency of about 50 Hz), the output from our microcontroller have a shorter period (thus a higher frequency, about 250 to 300 Hz). The Turnigy Plush ESCs (and the HobbyKing brand clones) have been reported to be able to handle the higher frequency, and thus more adjustments in motor speed can be made per second, therfore the quadrocopter will become more stable.

I've also attached screenshots from my logic analyzer to show you what the 4 signals look like.

Here is the code:

```c
#define PWM_FREQUENCY 300 // in Hz
#define PWM_PRESCALER 8
#define PWM_COUNTER_PERIOD (F_CPU/PWM_PRESCALER/PWM_FREQUENCY)

void setup()
{
  // pins as output
  DDRD |= (1 << 4) | (1 << 5) | (1 << 6) | (1 << 7);

  // default to 1000 microsecond pulse width
  OCR1A = 1000 * 2 ;
  OCR1B = 1000 * 2 ;
  OCR2A = 1000 / 16 ;
  OCR2B = 1000 / 16 ;

  // the setup is:
  // Clear OCnA/OCnB on compare match, set OCnA/OCnB at BOTTOM (non-inverting mode)

  // setup timer 1
  TCCR1A = (1<<WGM11)|(1<<COM1A1)|(1<<COM1B1);
  TCCR1B = (1<<WGM13)|(1<<WGM12)|(1<<CS11);
  ICR1 = PWM_COUNTER_PERIOD;

  // setup timer 2
  TCCR2A = (1<<WGM20)|(1<<WGM21)|(1<<COM2A1)|(1<<COM2B1);
  TCCR2B = (1<<CS22)|(1<<CS21);

  // the period is fixed for timer 2, it's about 244 Hz

  // note that timer1 is a 16-bit timer and timer2 is a 8-bit timer
}

void loop()
{
  int pw;
  for (pw = 1000; pw <= 2000; pw += 20)
  {
    OCR1A = pw * 2 ;
    OCR1B = pw * 2 ;
    OCR2A = pw / 16 ;
    OCR2B = pw / 16 ;
    delay(10);
  }
  for (pw = 2000; pw >= 1000; pw -= 20)
  {
    OCR1A = pw * 2 ;
    OCR1B = pw * 2 ;
  }

```
Step 14: Arduino Demo: Sensor Reading
This step contains an Arduino sketch that will take readings from the two sensors (BMA180 accelerometer and ITG-3200 gyro).

These two sensors are connected to your microcontroller using a I2C bus. This type of bus is designed to allow multiple devices to communicate together using only two wires.

I've also attached the logic analyzer waveforms of the I2C bus signals. (both screenshot and Saleae Logic 1.1.8 session file)

For more information:
- I2C Tutorial - Embedded Lab
- BMA180 Datasheet
- ITG-3200 Datasheet

The flight controller software will initialize the accelerometer by first resetting it, then setting up a 10 Hz low-pass filter and setting the read range to +/- 2 G. Read its datasheet and see my demo code for more details about how to do this. There are other features such as interrupts, tap detection, etc, that we are not using.

The flight controller software will initialize the gyro sensor by first resetting it, then setup its 10 Hz low-pass filter, and telling it to use its own internal oscillator. Read its datasheet and see my demo code for more details about how to do this. This sensor allows you to use an external oscillator and also has digital temperature outputs, but we don't use those features.

Here is the code:

```c
#include <Wire.h>

void setup()
{
    Serial.begin(115200);
    Wire.begin();

    Serial.println("Demo started, initializing sensors");
    AccelerometerInit();
    GyroInit();

    Serial.println("Sensors have been initialized");
}
```

void AccelerometerInit()
{
  Wire.beginTransmission(0x40); // address of the accelerometer
  // reset the accelerometer
  Wire.send(0x10);
  Wire.send(0xB6);
  Wire.endTransmission();
  delay(10);

  Wire.beginTransmission(0x40); // address of the accelerometer
  // low pass filter, range settings
  Wire.send(0x0D);
  Wire.send(0x10);
  Wire.endTransmission();

  Wire.beginTransmission(0x40); // address of the accelerometer
  Wire.send(0x20); // read from here
  Wire.endTransmission();

  Wire.requestFrom(0x40, 1);
  byte data = Wire.receive();

  Wire.beginTransmission(0x40); // address of the accelerometer
  Wire.send(0x20);
  Wire.send(data &0x0F); // low pass filter to 10 Hz
  Wire.endTransmission();

  Wire.beginTransmission(0x40); // address of the accelerometer
  Wire.send(0x35); // read from here
  Wire.endTransmission();

  Wire.requestFrom(0x40, 1);
  data = Wire.receive();

  Wire.beginTransmission(0x40); // address of the accelerometer
  Wire.send(0x35);
  Wire.send((data &0xF1) | 0x04); // range +/- 2g
  Wire.endTransmission();
}

void AccelerometerRead()
{
  Wire.beginTransmission(0x40); // address of the accelerometer
  Wire.send(0x02); // set read pointer to data
  Wire.endTransmission();

  Wire.requestFrom(0x40, 6);
  // read in the 3 axis data, each one is 16 bits
  // print the data to terminal
  Serial.print("Accelerometer: X = ");
  short data = Wire.receive();
  data += Wire.receive() << 8;
  Serial.print(data);
  Serial.print(" , Y = ");
  data = Wire.receive();
  data += Wire.receive() << 8;
  Serial.print(data);
  Serial.print(" , Z = ");
  data = Wire.receive();
  data += Wire.receive() << 8;
  Serial.print(data);
  Serial.println();
}

void GyroInit()
{
  Wire.beginTransmission(0x69); // address of the gyro
  // reset the gyro
  Wire.send(0x3E);
  Wire.send(0x80);
  Wire.endTransmission();

  Wire.beginTransmission(0x69); // address of the gyro
  // low pass filter 10 Hz
  Wire.send(0x16);
  Wire.send(0x1D);
  Wire.endTransmission();

  Wire.beginTransmission(0x69); // address of the gyro
  // use internal oscillator
  Wire.send(0x3E);
  Wire.send(0x01);
  Wire.endTransmission();
}

void GyroRead()
{
  Wire.beginTransmission(0x69); // address of the gyro
  Wire.send(0x1D); // set read pointer
  Wire.endTransmission();
}
Wire.endTransmission();

Wire.requestFrom(0x69, 6);

// read in 3 axis of data, 16 bits each, print to terminal
short data = Wire.receive() << 8;
data += Wire.receive();
Serial.print("Gyro: X = ");
Serial.print(data);
Serial.print(" , Y = ");
data = Wire.receive() << 8;
data += Wire.receive();
Serial.print(data);
Serial.print(" , Z = ");
data = Wire.receive() << 8;
data += Wire.receive();
Serial.print(data);
Serial.println();
}

void loop()
{
AccelerometerRead();
GyroRead();
delay(500);  // slow down output
}
**Step 15: Control Principle**

The code allows you to fly the quadcopter in 2 modes:

One is where the accelerometer is ignored, and you control the helicopter's angular velocity with your joystick. This is referred to as "aerobatic" mode. In this mode, the code's job is to spin the helicopter when you want it to, and stop it from spinning when you don't. The advantage of this mode is that you can perform stunts like flips and barrel rolls.

The other mode is where the accelerometer is used with the gyro sensor to calculate the quadcopter's current angle with respect to the ground. You control the angle of the quadcopter instead. So in this mode, if you let go of the joystick, the quadcopter should stay at a horizontal level. If you move the joystick to a non-center location, the quadcopter will spin to one angle, then stay at that angle (and move since it's blowing air at an angle instead of straight down). This mode is referred to as "stable" mode or "attitude" mode. This mode requires an accelerometer to work, and will not allow you to perform flips and rolls (since your joystick does not do 360 degree rotations).

To calculate the quadcopter's current angle, the angular velocity is read using the gyro sensor. Angular velocity multiplied by time equals the amount of rotation during that time. There's a lot of error in this process due to drift and noise in the gyro sensor. To correct this error, the accelerometer is used to measure gravity, which is mixed into the calculation. For more details on this, please see:

- The Balance Filter
- The Kalman Filter
- AHRS, Attitude and heading reference system

The flight controller uses this information along with a PID controller to control the motors. PID means "Proportional, Integral, Derivative". See these links:

- PID Controller
- [http://www.engin.umich.edu/group/ctm/PID/PID.html](http://www.engin.umich.edu/group/ctm/PID/PID.html)

To understand the "P" term, imagine you are trying to park a car. If you are very far away from the parking spot, you drive fast towards the spot. As you get closer, you should slow down. If you drive over the line, you need to reverse (and thus your velocity is negative). Speed = distance to spot * "P" in this case.

To understand the "I" term, imagine you are trying to park your car but your car is tied to a rope that won't let you reach the spot. You keep trying harder and harder until the rope snaps. Engine RPM = I term * integral of distance to spot over this period of time.

To understand the "D" term, say you moved 10 meters towards your spot, since you know you made progress, you should slow down a bit. Speed = difference between position * D term.

Now imagine instead of a parking spot and car, you use this PID controller to get your quadcopter from one angle to another angle by adjusting the difference in the rotor's spinning speed. By tuning the values of the PID terms, we can get the flight controller to make big adjustments and small adjustments accordingly, keeping the quadcopter stable in flight.

![PID Controller Diagram](image)

**Step 16: Quadcopter Assembly**

Since your circuit is now done, you can assemble the entire helicopter. I can't let you tune the helicopter without it being able to fly first.

Remember to unplug the battery whenever you make electrical changes (connecting and disconnecting things).

The entire frame takes several steps to assemble, there are way too many pictures for one "step" so I've divided them. I'll be going into a lot of details because the kit does not come with instructions.

Also the pictures may not follow chronological order, this is because I may have decided to re-order some steps.
Step 17: Reinforce Arms
The frame kit comes with the arms pre-assembled. Although the joints are glued, I found them too unreliable and prone to cracking. Also some joints have gaps which are too big.

Get some Gorilla Glue, the general purpose kind. Apply it to the joints. When this glue cures, it expands to three times its original volume, it will actually fill in the cracks for you.

By doing this, the amount of vibration in the frame is lowered, giving your sensors much better readings.

Step 18: Insert ESCs in Arms
Stuff the ESCs into each arm. Do his before assembling anything else because I found it impossible to get the ESCs to fit if there are anything in the way.

I made the heat-sink side of the ESCs face upwards in hopes of improving the cooling due to the airflow from the propellers.

Make sure all the thick wires point down. The thin servo cable should still point up. If you got the same parts I have, all the wires should be the perfect length required.
Step 19: Top Plate

Make sure you have a way of mounting your circuit board. You can design the circuit board to fit onto the holes already provided on the top plate, or drill holes into the plate yourself. My circuit uses slots to fit any square mounting pattern. I also have another design that I have prepared my top plate for.

If you drill into the top plate, you might want to drill the same holes into the bottom plate so you can use stand-offs with those holes.

Screw in all the stand-offs (I have extras since I purchased two frame kits).

Each arm piece slides onto the top plate. There should be three red arms and one that is grey. The grey one is the “front”. The plate should have one side with triangles that indicate the forward direction. So make sure you mount the grey “front” arm in the forward direction.

There should be a hole for each arm that aligns with a hole on the plate. Place a screw and nut here. You can cut the nylon screw to an appropriate length to make this step easier and look nicer.
**Step 20: Landing Skid**

It is impossible to attach the landing skid if the bottom plate is already attached to the top plate. Take the black landing legs and place them into the thin slots in the top plate. These slots should be next to the left and right arms. Align the hole and then use some screws and nuts to secure them.

There should be some thin black cross-members for the legs in your kit. Some people glue these in place but I used electrical tape.

The actual skid piece goes on the bottom of the legs, it has slots so it stays in place but they still tend to fall off very easily. I've used electrical tape to secure them.
Step 21: Bottom Plate
I made two aluminum strips which will mount onto the bottom plate. The attached pictures show you the dimensions of these stripes.

Some double-sided velcro straps are glued to one of the aluminum strips with epoxy glue.

The aluminum strips are screwed to the bottom plate, using a nut as a spacer. These pieces will make it easy to strap in the battery later.

Place the power distribution “spider” between the top and bottom plate before you join the two plates. Arrange the wires in a neat fashion first.

Join the two plates using the stand-offs, nuts, and screws.
**Step 22: Mount the Circuit Board**

Before the top dome can be placed, the circuit should be mounted first. Do so using the short stand-offs and screws. You might want to use washers with your screws. The screws might also need to be cut to a length such that it can fit into the stand-offs.

The circuit board should have a direction indicator, make sure it is pointed in the right direction.

![Circuit Board Image](http://www.instructables.com/id/RC-Quadrotor-Helicopter/)

**Step 23: Top Dome**

There are 8 dome support pieces, 4 cross-members, and 2 top pieces in the frame kit.

The support pieces attaches to the sides of each arm. Use the screws and nuts to secure them. The screws will need to be cut as the width of the arm is too narrow for two full length screws.

Some of these are grey to indicate those are the pieces which should face forward. Attach those to the front arm.

The cross-members goes between two support pieces. The two top plates will sandwich all the support pieces. Use screws and nuts to clamp down on both top plates. Notice that it is very easy to over-tighten these particular screws due to the gap between the two plates. You will risk cracking the plates if you over-tighten.

I've also attached some velcro tape to my RC radio receiver and the dome to make it easy to mount and dismount.

![Top Dome Image](http://www.instructables.com/id/RC-Quadrotor-Helicopter/)
Step 24: Mounting Motors

Each brushless motor can be mounted to an arm. Use 1/2" long #4-40 machine screws (please use steel) along with nuts, flat washers, and spring-locking washers. Make sure these screws are very tight.

When you are ready, mount the propellers. Follow the rotor spin direction diagram that I’ve provided.

The motor should have arrived with all the hardware you need in order to mount the propellers. There is a cone shaped washer, make sure that piece goes between the nut and propeller, and that the cone is facing the propeller. The cone shape is what keeps the propeller dead center on the shaft (if it’s not centered, it will vibrate from the imbalanced centripetal forces).

Step 25: Final Wiring
Plug the motors into the ESCs. Do not care about the order of the three wires for now. We'll a test later, if they spin the wrong way, we'll fix it then.
Plug the ESCs into the power distribution "spider", watch your polarities.
Plug the servo cables from each ESC into the flight controller circuit. The appropriate location for each ESC should be indicated on the PCB.
Channel 1 to 6 on the RC radio receiver must go into the respective RC signal input of the flight controller. Make sure that the RC radio receiver is powered by 5V from the flight controller somehow.
Please be careful, do not plug in anything backwards. The PCB should be clearly labelled with the direction in which the servo wires should be plug into.
If you are using my circuit design, please follow the appropriate wiki article for even more details.
Step 26: Software Setup

In the "preparing the microcontroller" step, you should already have the bootloader installed onto the microcontroller and Arduino IDE completely ready to compile for the ATmega644P.

Connect your FTDI cable (or similar USB-to-serial cable) to the circuit board.

Attached here is a modified version of AeroQuad V2.4.1 flight software. I have modified it to work with my circuit design. Download it, open it with Arduino IDE, compile it, and upload it to the circuit board.

Also please download and install the AeroQuad Configurator (version 2.7.1 is what I'm using).

Below is just a small summary of what you need to do with the AeroQuad Configurator. Please follow the official instructions for AeroQuad Configurator for more detailed instructions.

Use the configurator to talk with the circuit, just make sure it works first. Then run through the steps to initialize the EEPROM. Run the sensor calibration with the help of a spirit/bubble level. Make sure that the sensors respond correctly. Make sure the RC radio receiver PWM signals are being read correctly.

Calibrate the ESCs by following the instructions in the configurator (basically power off the ESCs, click an OK button on the screen, power on the ESCs, click another OK button). You can then make sure the motors spin in the correct direction (do this without the propellers for safety reasons)

Step 27: RC Controls

Your RC radio transmitter has two joysticks for the first 4 channels. I have attached an image to show you my radio and which joystick direction corresponds to which channel (the arrows indicate direction of increasing pulse width). Your radio might be different, but they should be labelled.

Usually one joystick controls throttle and yaw, and the other stick controls roll and pitch.

Depending on whether you are left or right handed, you may wish to control throttle & yaw using the left or right stick. To make this change, simply swap the signal wires between the flight controller circuit and the RC radio receiver.

If a direction is reversed, the RC radio transmitter should have a built in reversing setting. If it doesn't, you must have the worst radio in the world and should probably buy a new radio, or else, find the source code that takes the signal inputs and flip the values. (or you can disassemble your radio and re-wire the joystick's potentiometers).

Also the joysticks are usually spring loaded so that they return to center when you let go. You probably do not want this for the throttle. Please see your RC radio transmitter's manual to see how the spring can be disabled. On my RC radio transmitter, I simply take it apart and remove the spring (don’t lose it!).

The flight software (AeroQuad) have an ARMED and DISARMED state, in the disarmed state, the motors are disabled for safety. Move the throttle down and then to the left to disarm. Move the throttle down and then to the right to arm. Memorize this so you can disable the motors in an emergency. On my circuit board design and modified version of AeroQuad, the LED3 indicates that the motors are armed.

The switch on channel 5 switches the software between the "acrobatic" mode and the "stable/attitude" mode in the software.

Image Notes
1. Remove This Spring

Image Notes
1. No Spring = No Return-To-Center
**Step 28: Tuning and Calibration**

Charge and strap in your battery. You must learn to ARM and DISARM your helicopter first, get good at this since you must disarm your helicopter as soon as something goes wrong. For the AeroQuad software, to ARM and DISARM, lower the throttle joystick to the bottom, and the move it left for DISARM and right for ARM. With my circuit and modified software, LED3 being lit means the helicopter is ARMED.

This should be the first time when you have your motors running. Make sure you always have a good solid hold on the frame such that you will not be harmed by the spinning propellers. Also wear eye protection! Tuning is best done with two people, one person to hold the RC radio transmitter, and the other person to hold the helicopter.

First, read this: Pre-Flight Checkout List

Turn on the helicopter with the propellers installed. Run the configurator to calibrate all the ESCs (follow the provided official instructions). Make sure they spin according to the rotor spin direction diagram. If a motor needs to be reversed, then recall what I said about swapping the wires between the ESC and the motor.

There are several parameters that you can tune.

The transmit factor is there to adjust the sensitivity of your RC radio transmitter. This value depends on your personal piloting skills and preference. The smoothing factors are the constants for low-pass-filtering the RC PWM signals, leave them at the default values unless your transmitter is very noisy.

Perform a sanity test by using default values. Raising the throttle should move all 4 motors faster. Trying to yaw will cause two motors which spin in the same direction will spin faster than the other two. Trying to pitch or roll will cause opposing motors to spin at different speeds. You should be able to feel the forces with your hand that is holding the helicopter. Also use your hand to introduce forces, and the helicopter should respond by applying counteracting forces appropriately. If something doesn't make sense, a value is probably reversed/negative in the software, use the configurator to identify it and correct it in the Arduino sketch.

You can also adjust the PID constants to suit your airframe. Start off with the default values. The P term is adjusted first, try and rotate the helicopter by hand while it is flying, raise the P value until the helicopter resists your hand very well, but lower it if the helicopter oscillates. Do this for all three axis.

AeroQuad's documentation do not suggest you adjust the I term for acrobatic mode. It also says "A negative D value is used to help the AeroQuad change faster to a level position after forward flight. It is possible to leave D = 0 and still see attitude flight. Using a negative D value is only needed based on user preference."

My summarized version about PID tuning is really short, for more detail, please see:


Also see the official AeroQuad Configurator documentation for more details about tuning stuff.

---

Image Notes

1. FTDI USB-to-Serial Cable

---

Step 29: Flying
Now that everything is tuned, you should be ready to fly! Since you already tuned the helicopter, you should already be familiar with how to control the helicopter. Charge up and strap in your battery and go to your airspace.

Make sure you stand far away from the helicopter. Wear eye protection. The propellers are really dangerous. Do not attempt to “hand launch” the helicopter, just in case it wants to slice your neck as soon as you let go.

Try to fly outdoors during no wind conditions. If you want to fly indoors, then find a large room because the downwash air from the helicopter would cause too much disturbance in a small room.

Buy plenty of spare propellers, which break the fastest. Also, the frame from HobbyKing isn't very durable since it's made from plywood, but since each frame kit is $15 and comes with 4 arms, you should buy extras since arms break fast.

Watch your battery monitor while flying. When any three of the cells in your battery pack becomes drained below the safe threshold, you must try to land immediately, or else you will damage your battery.

Related Reading:
- Beginning Flight Tips - AeroQuad Wiki
**Step 30: Repair, Re-tune, Recharge**

If you crashed while flying, you might have broken some of the quadcopter's arms. There's no easy way to fix this, so just disassemble the frame and replace the arm with a spare.

After crashing, check if the battery is damaged. A damaged lithium polymer battery can explode! Immediately after the crash, disconnect the battery immediately if it has not exploded already, and inspect it before judging whether or not you can keep using it. If it's dented, bulging, bent, punctured, or cut, then dispose of it properly.

Broken propellers are easy to replace, but make sure that the shaft or motor isn't damaged. Remove the propeller first, then run the motor at a low speed to see if it's still functioning and not wobbly. If it's wobbly, then you should replace the motor, do this while not flying, or else you risk having the motor fail in mid-air which is more dangerous.

If you don't like the flight characteristics, just reconnect your flight controller back to your computer and re-tune the parameters until you are satisfied.

If you've noticed that your quadcopter doesn't stay stable no matter how hard you tune the configuration parameters, then maybe the airframe is vibrating too much for the sensors to be effective. Vibration is caused by imbalance of mass around the spinning motors. When the mass is balanced, the net centripetal force around the motor is 0. When there's an imbalance, then the centripetal force will have a magnitude greater than 0 and the angle of that force will spin. This becomes oscillation, which is vibration. So to eliminate vibration, we need to apply a force equal in magnitude but opposite in direction. Get two zip-ties per motor, and tie two zip-ties around each motor. While the propellers are installed, move the zip-ties around until the force they introduce cancels out the vibration.

Recharging the battery should sound straightforward. My charger is pretty nice and full of features, but it didn't come with a manual (I found one online later). There are two main power wires (positive and negative) that connect from the battery's main wires (the thick ones) to the charger (two banana plug jacks), connect those in the correct orientation. Then plug the JST balance connector from the battery to the balancer jack of the charger. One characteristic of lithium polymer batteries is that each cell must be monitored and balanced with other cells during charging, hence why your battery has 2 different connectors.

Place the battery into a Li-Po bag (it's a fire-proof bag you can buy that's designed for charging lithium polymer batteries), or put them into a cooking pot like me. On the charger, select the lithium polymer charging routine, set the number of cells to 3, and start the charge cycle. The process should be automatic but you should not leave the battery un-attended during charging.

If the charger can charge at 6 amps, then get a power supply that can handle 6 amps (I use a laptop power adapter). Most ordinary wall adapters cannot handle that much current, do not make that mistake.

Please charge at 1C or less, this means if the battery is 5000 mAH, then you can charge at 5 amps, but if it's 2200 mAH, then you can charge at 2.2 amps. Maximum Current divided by Battery Capacity equals C rating. For more info, read this guide on lithium batteries.
This is the battery pack for my RC radio transmitter
various custom adapters I made
My balancing charger
My main battery, 3S1P, 20C, 5000 mAh

Plenty of spare battery packs so I can keep flying
Step 31: Final Thoughts

You can also experiment with flight software based on MultiWiiCopter, which is very similar to AeroQuad. MultiWiiCopter supports more airframe configurations such as tri-rotor, hex-rotor, and octo-rotor. Its graphical user interface is also different (based on Processing.org, which is Java, I like it better than the LabView based AeroQuad Configurator).

To take flight videos, I spent $5 on a keychain video camera like this one. They are cheap and lightweight. Do not use velcro to mount the camera since the vibrations will ruin the video, use tape instead.

The frame kit comes with a camera attachment, you can attach a small servo to it, but I didn't use it. Also, apparently this helicopter landing skid is a good match for the frame.

My instructable focuses on + configuration flying, but you can easily convert your quadcopter to fly in X configuration. It requires some reconfiguration and recompiling of the source code. The circuit board needs to be mounted at 45 degrees. The landing skid needs to be re-mounted at 45 degrees (the frame kits have slots and mount pieces already for X configuration mounting).

You probably have seen two of my flight controller circuit designs. My designs usually allow you to add in magnetic compass sensors and barometric air pressure sensors. These sensors can be used to let your software get heading and altitude data, which can aid in keeping the quadcopter facing one direction and stay at one altitude. GPS can also be added. The combination of these extra sensors can help with autopilot capabilities. AeroQuad code and MultiWiiCopter code both have some support for these sensors.

Thanks to:
- AeroQuad
- MultiWiiCopter
- Arduino
- Wikipedia
- WPClipart
- Bird Clipart

More info:
- Quadrocopter and Tricopter Info Mega Link Index
- RCGroups - Multi Rotor Helis