ArduPilot 2.x manual

Last updated 9/19/09

Table of Contents

Introduction to ArduPilot

ArduPilot is an inexpensive, simple to use open source autopilot platform created by Chris Anderson and Jordi Munoz of DIY Drones. The hardware consists of the core autopilot board, sold by Sparkfun, and various sensors and accessories to add to its functionality. The software is also open source and is available in several versions, including navigation-only, navigation+stabilization and versions that support additional sensors and expansion options. This version of the manual is just for the full navigation and stabilization version of the code, which is version 2.1 and above.

Features:

- Programmable 3D waypoints; return-to-launch mode, in-flight reset ability, fully programmable actions at waypoint and totally expandable board.
- Integrates the stablization and navigation functions, eliminating the need for a thrid-party Co-Pilot.
- Controls elevator, ailerons/rudder and throttle.
- "Fly-by-wire" mode stabilizes the aircraft in RC mode, duplicating the function of the FMA Co-Pilot.
- Stores home waypoints and sensor calibration in EEPROM, so they are retrained even in the case of a system restart.
- Currently optimized for the three-channel EasyStar. Versions have also been tested on Funjet and four-channel Superstar.
- Can use any thermopile XY sensors (default settings are for the FMA sensor, but Paparazzi, AttoPilot and custom sensors can also be used).
- Designed for the EM406 GPS. Other GPS modules, including 3.3v 5Hz modules such as the uBlox5, are supported via the optional "shield" expansion board and the 2.2 and above code.
- Uses "chained PID loops" to combine the stabilization and navigation functions seamlessly.

ArduPilot 2.1

- Last version of the software to support the original Atmega168-based ArduPilot board (now discontinued)
- Supports XY Sensor mounted in a diagonal postion, facing forward or back.
- Z sensor highly recommended but not absolutely required.
- Controls throttle if airspeed sensor is attached via the expansion board.
- Uses a desktop setup utility for waypoints and autopilot settings.
- Supports the Ground Station for real-time telemetry

ArduPilot 2.3 and above

- Requires the newer ATMega328-based ArduPilot board.
- Z sensor required.
- Requires the expansion board ("shield") with airspeed sensor
- Includes all the features of 2.1, plus:
- Supports the EM406 and uBlox5 GPS modules in efficient binary mode.

- Incorporates a NMEA parser to support a wide range of other GPS modules at up to 5hz (especially useful for people who already are using GPS for on-screen display and other functions).
- Supports updated Ground Station with battery voltage display.
- All user-modifiable settings are in a separate file; enables sharing of configuration files for different airframes.

What you need to make a complete autopilot

- ArduPilot board
- Shield expansion board kit
- GPS module (uBlox5 recommended; EM406 also supported in native mode. All other GPS modules supported in NMEA mode)
- XY and Z sensors
- FTDI cable for programming
- [Optional] Two Xbee modules for real-time wireless telemetry, with an adapter in the air and a different one on the ground/laptop side

What open source hardware means

ArduPilot is open source software *and* hardware. That means that the hardware schematics, PCB files and parts lists are all freely available on the website, published under a Creative Commons license that allows free use and modifications as long as the resulting product retains the DIY Drones credit. The software, both for the autopilot and the desktop utilities, are also open source, published under an Apache 2.0 license that allows free use and modification as long as the resulting product is also open source and DIY Drones attribution is retained.

We released ArduPilot as open source technology because we wanted to create a community, not just a product. Is there some feature of ArduPilot that you'd like that we haven't enabled. Do it yourself and share it with others! Want to use in for a submarine, not a plane? Modify away!

Because the hardware and software are open source, you can inspect them, understanding how they work, learning from them, and ensuring that they'll do what you want. And because many people are doing the same, the hardware and software are more likely to improve quickly and in the direction the users want. Open Source creates development communities, and communities can innovate faster than individuals or often even companies.

In practice, what we are doing is giving away all the ArduPilot intellectual property. If you want to build it yourself, you don't need to pay us a penny. If you want us (either DIY Drones or our partners, such as Sparkfun) to build the board for you, we'll do for a fair price and guarantee the product. What you are paying for is labor and components, not for trade secrets or our development work.

Recommended uses for autopilots and amateur UAVs

What are amateur UAVs for? A lot of things! Here are the most frequently-heard reasons to build one:

- Fun!
- Learning about aerial robotics
- Learning about "physical computing" (the combination of hardware and software engineering used in embedded systems)
- Aerial photography and mapping
- Contests
- Scientific sensing
- Automated navigation for "first-person view" video flying
- "Return to home" safety measure so RC aircraft don't fly away when they lose radio contact
- Stabilization of RC flight, to help new pilots and improve the flight performance of unstable planes

UAV safety and regulations

Safety must be your most important priority. That means checklists on the ground before launch to ensure that the aircraft will behave as expected, responsible mission planning and flying only within accepted range and locations. Formal laws on the integration of UAVs into the national airspace (NAS) are currently in development, but at the moment UAVs are governed by several FAA guidelines and regulations. We recommend you read our full FAQ on regulation, but the basics are as follows:

You can only fly autonomous aircraft in the NAS without Certificate of Authorization (COA) if you are doing so as an amateur. If you are flying for money or are a company, you are considered commerical and must apply for a COA. This is neither easy or fast to get; indeed you probably will not be granted one unless your aircraft goes through a certification process. ArduPilot is intended for amateur use and has not been certified for professional use.

Flying as an amatuer, you must adhere to RC aircraft rules. They include the following restrictions:

- You must keep the flight below 400ft above ground level
- You must keep the aircraft within unassisted visual line of sight
- You must have a pilot in control at all times, which is to say that the pilot must be able to take over at any time
- You must stay away from built-up areas and any airports.

Failure to follow these rules is not only dangerous but stupid. It will only take one amateur UAV recklessly sent in the path of a manned aircraft to have us all banned. You will not only go to jail, but you'll ruin this hobby for all of us. Don't do it.

In addition, here are some other tips for safe UAV flying:

- Never take off with the autopilot enabled. Only enable it a safe altitude
- Stay "three mistakes" above ground. Autopilot glitches can happen. You need time to switch into manual mode and recover. Never fly autonomously below 100 feet
- If the aircraft looks like it has missed a waypoint and is flying away, switch to manual and bring it home immediately. Better to run the course again than to lose the UAV forever.
- Always test the UAV on the ground before launch. Ensure that the control surfaces move the right way when you roll and pitch the aircraft (it won't be perfect because of IR noise near your body and the ground, but it should be directionally correct)

The RC Aerial Photography Association has a some excellent guidelines here.

Preparing the hardware

Here is how to complete your new ArduPilot board:

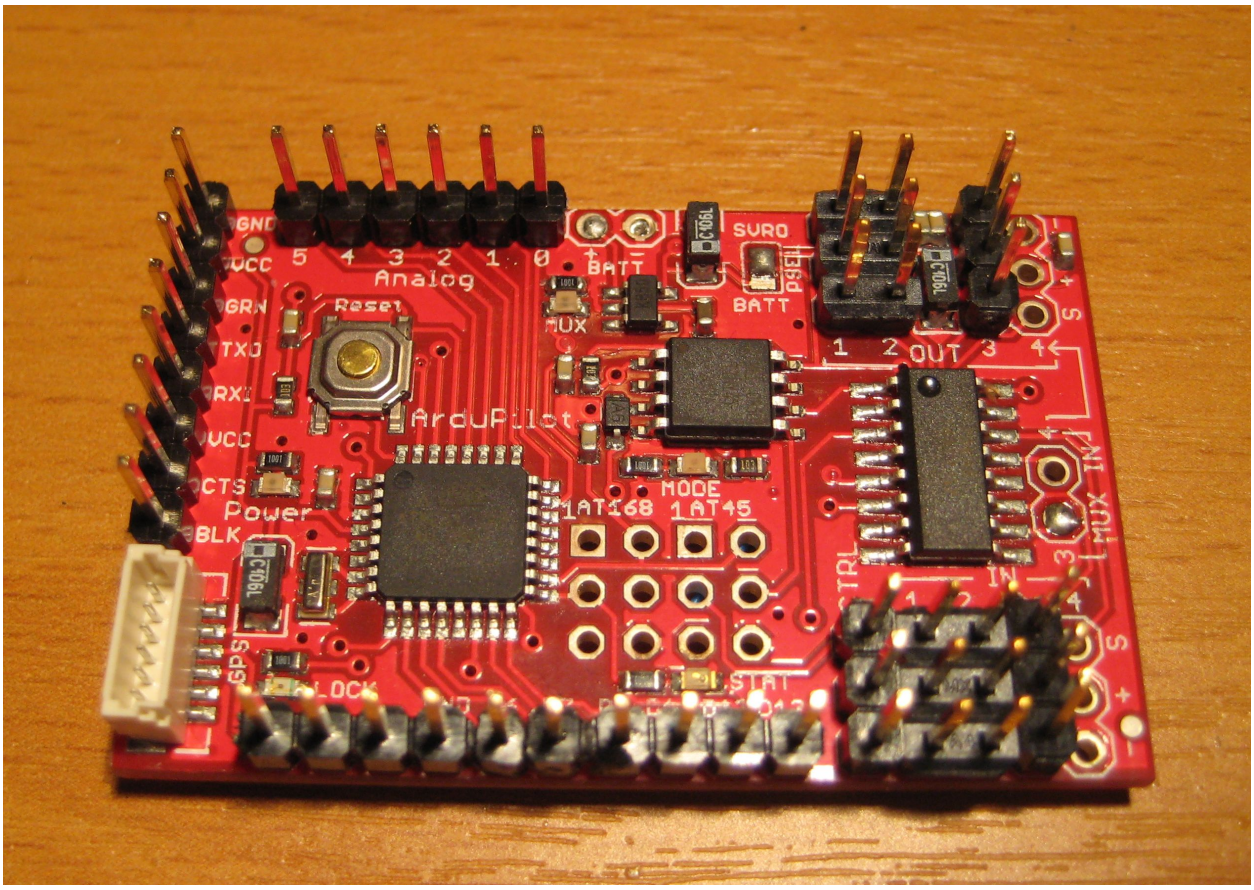
You should have received the basic ArduPilot board with all surface-mount components already soldered and the essential firmware already loaded on the chips. All you have to complete it is to solder some connectors to the board and load the autopilot software.

You will also need an FTDI cable if you do not already have one to load the software.

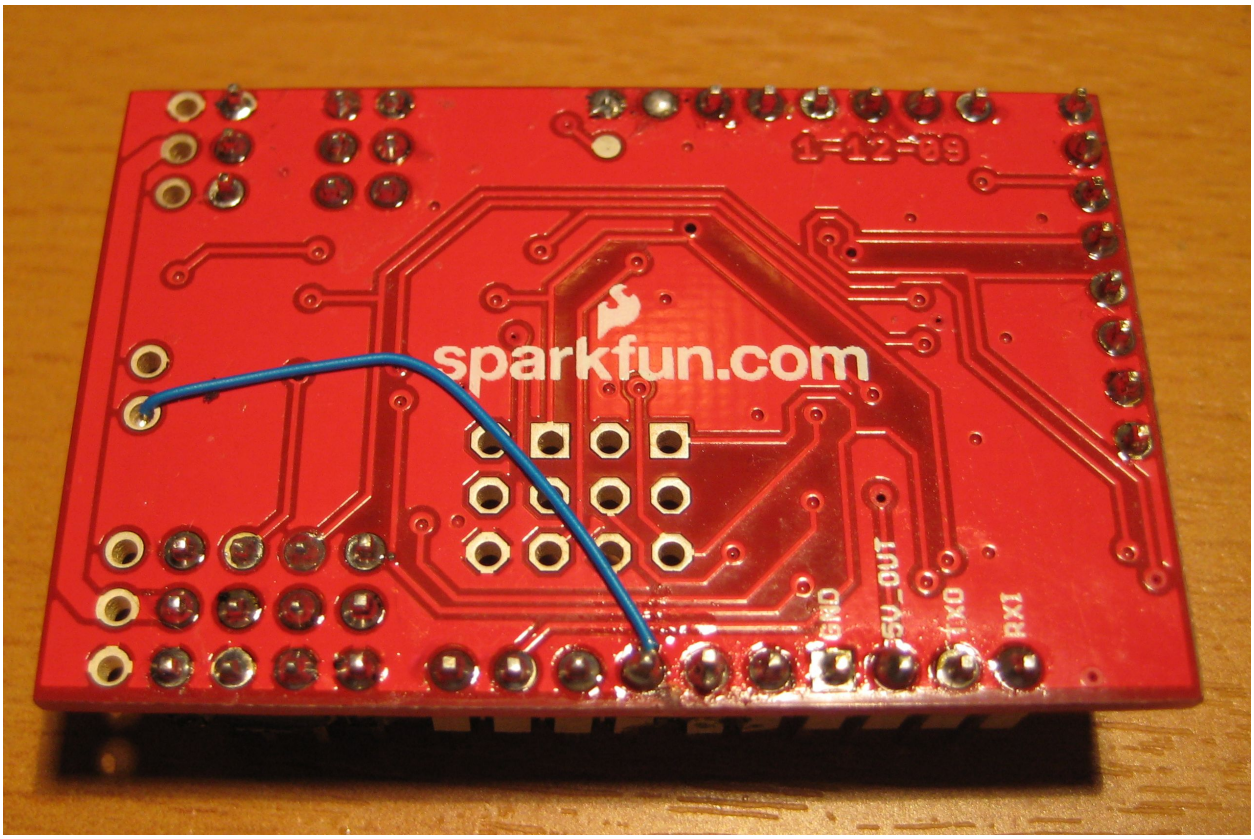
We also strongly recommend that for aircraft use you purchase the ArduPilot Shield expansion board kit with the airspeed sensor. The following instructions will assume that you are using the shield and have all the components that come with the kit. If you choose not to use the expansion board, the fucntionality of the autopilot will be impaired and we will be unable to provide technical support.

Soldering on the connectors

The first thing to do is solder on the connectors. From the strip of breakaway connectors, use a pair of pliers or a snipper to cut off seven 3-pin segments, a ten-pin segment, and eight-pin segment and a six-pin segment. Solder them in the pins as shown below:



Now, on the bottom of the board, solder a short strip of wire-wrap wire between the pins shown (this is to connect the throttle):



[Note: the board comes from the factory set up to get power from your RC system. If you want to power the board from a separate power source, you can do so--the board has a built-in power regulator that can accomodate power sources from 5v-15v. Instructions to make that change are below]

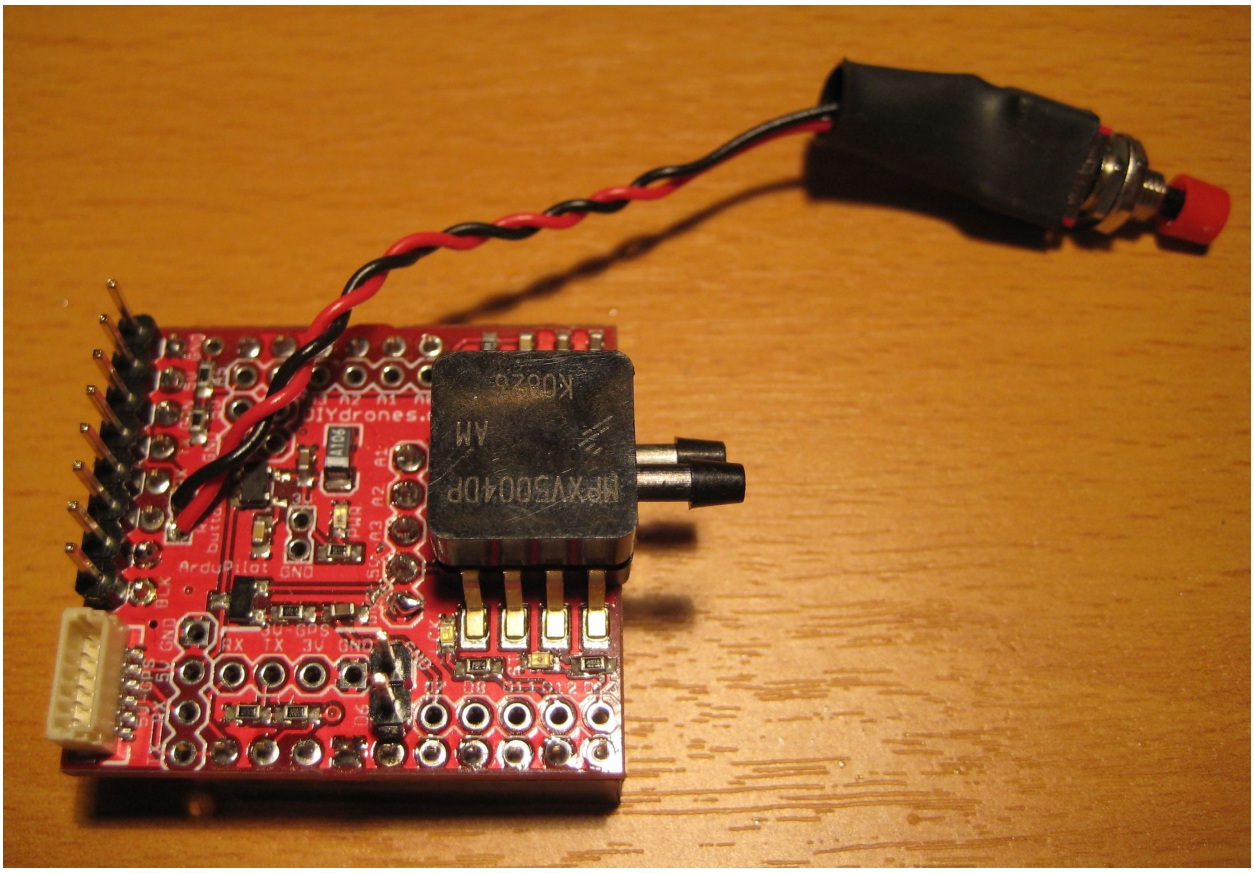
Now set up your shield board. It will have come with three female connector strips, male breakaway connector strips, an IR sensor connector, a cable for the IR sensors and a reset button and wires. Detailed assembly instructions are here, but the basics are the following:

Start by soldering the three female connector strips on the bottom side of the shield. To ensure that they will mate properly with Ardupilot, plug the female connectors into the matching male connectors on ArduPilot before pushing their pins through the matching holes in the Shield. Once they're in place, you can solder them where their pins emerge from the holes on the upper side of the Shield.

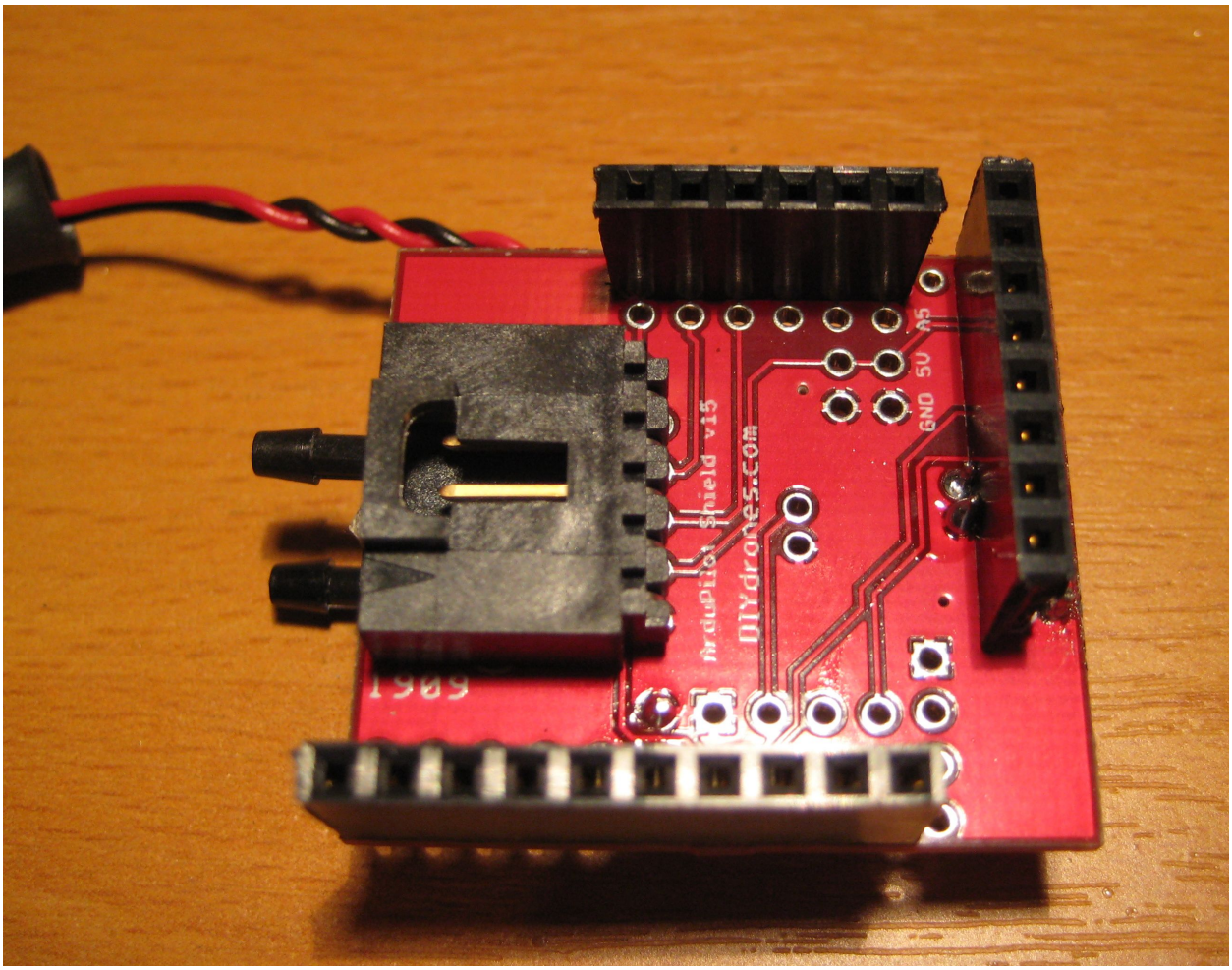
Now add a 8-pin male breakway connector strip next to the GPS connector on the Shield, as shown at left below. This mirrors the FTDI row on ArduPilot and will be used for programming and connecting the Xbee module for wireless telemetry.

Solder in the sensor connector (on the bottom, as shown in the second photo below) and the reset switch (on the top)

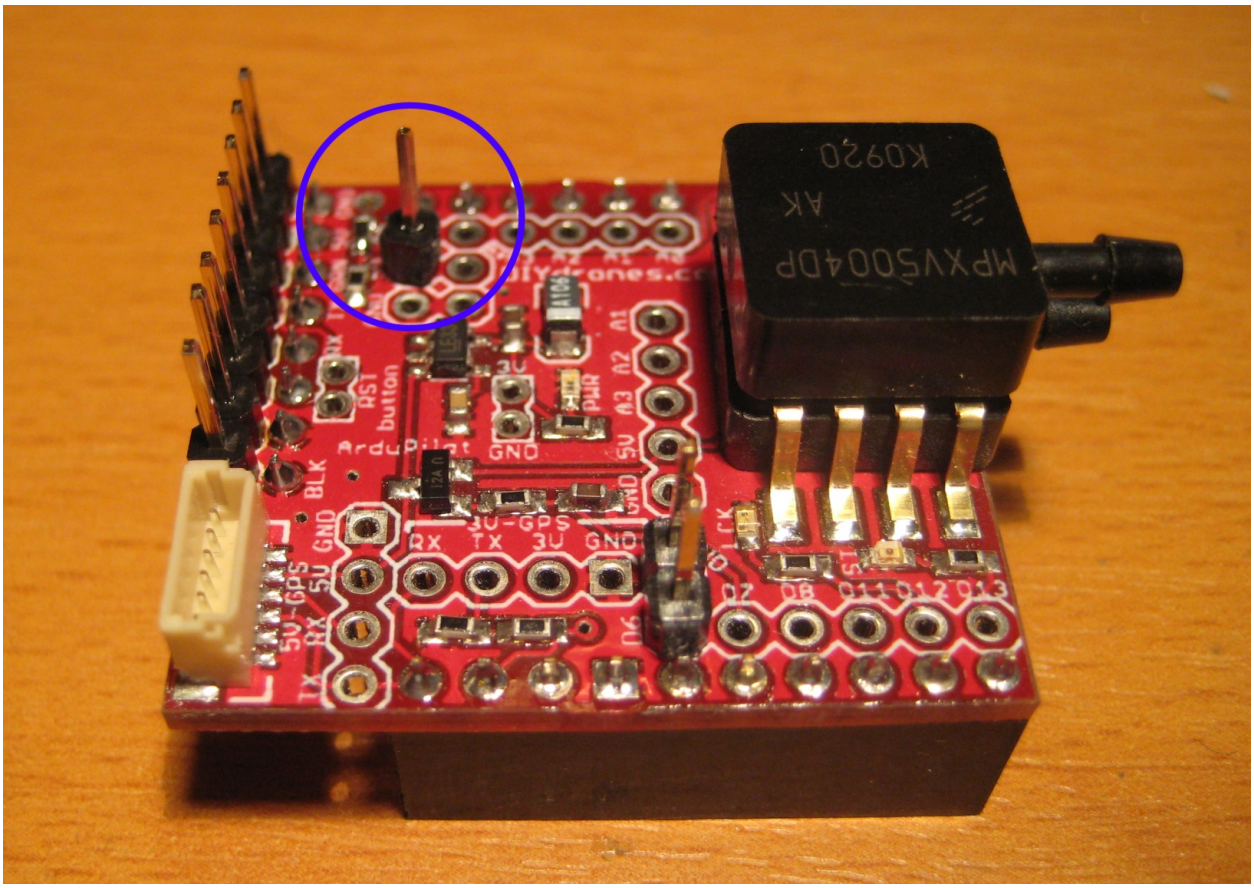
Top:



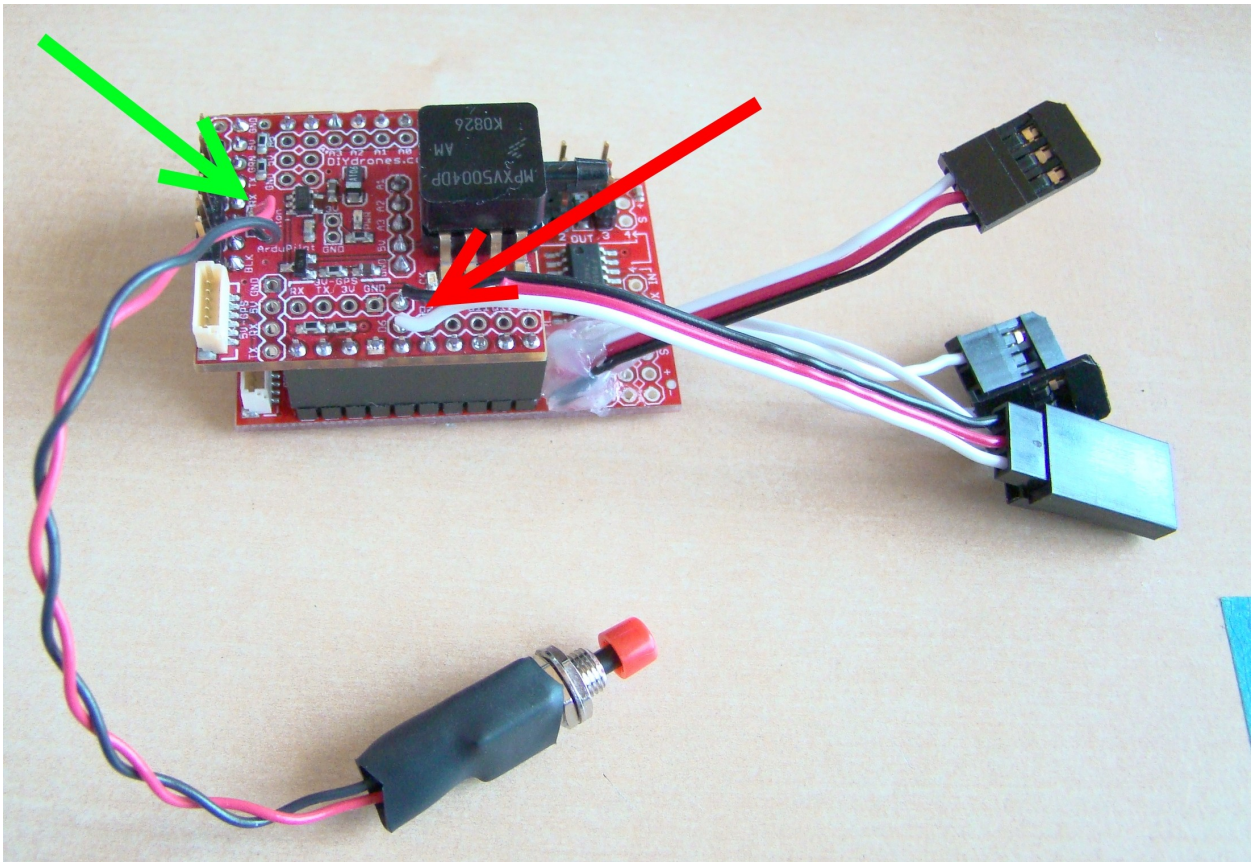
Bottom:



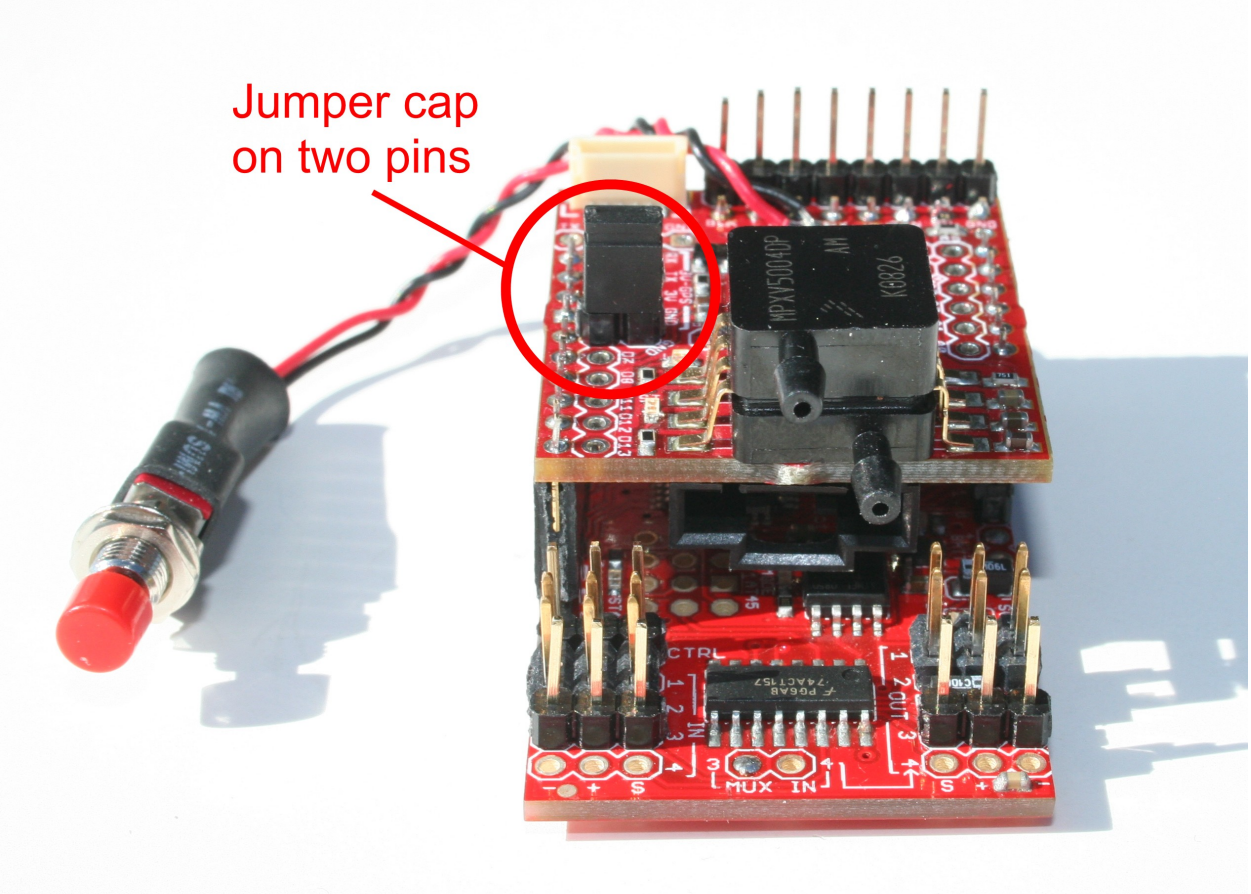
Add one pin (shown below circled in blue) to provide power for your wireless telemetry.



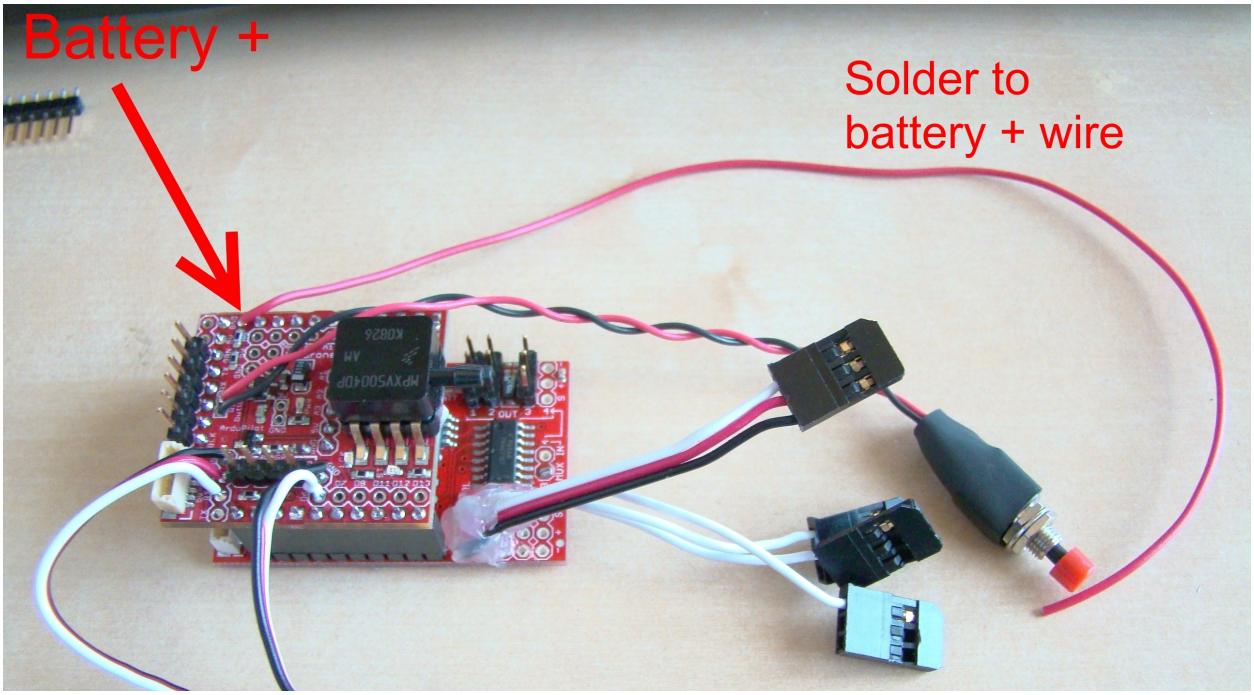
Option 1: If you want to have access to field setup jumper from outside your aircraft, cut one of the male-to-female servo cables in half, strip the wires (you can cut off the red wire entirely, since you won't be using it) and solder the black and white wires to the Shield D6 and GND holes as shown below with the red arrow (the green arrow points to the reset switch holes):



Option 2: Alternatively, you can just put a two-pin header in those holes, and use a jumper cap to bridge them as shown below:



If you want the Ground Station to record the remaining charge of your battery, you'll need to solder a wire from the shield to the battery's positive terminal. The picture shows how to attach the optional voltage measurement wire:



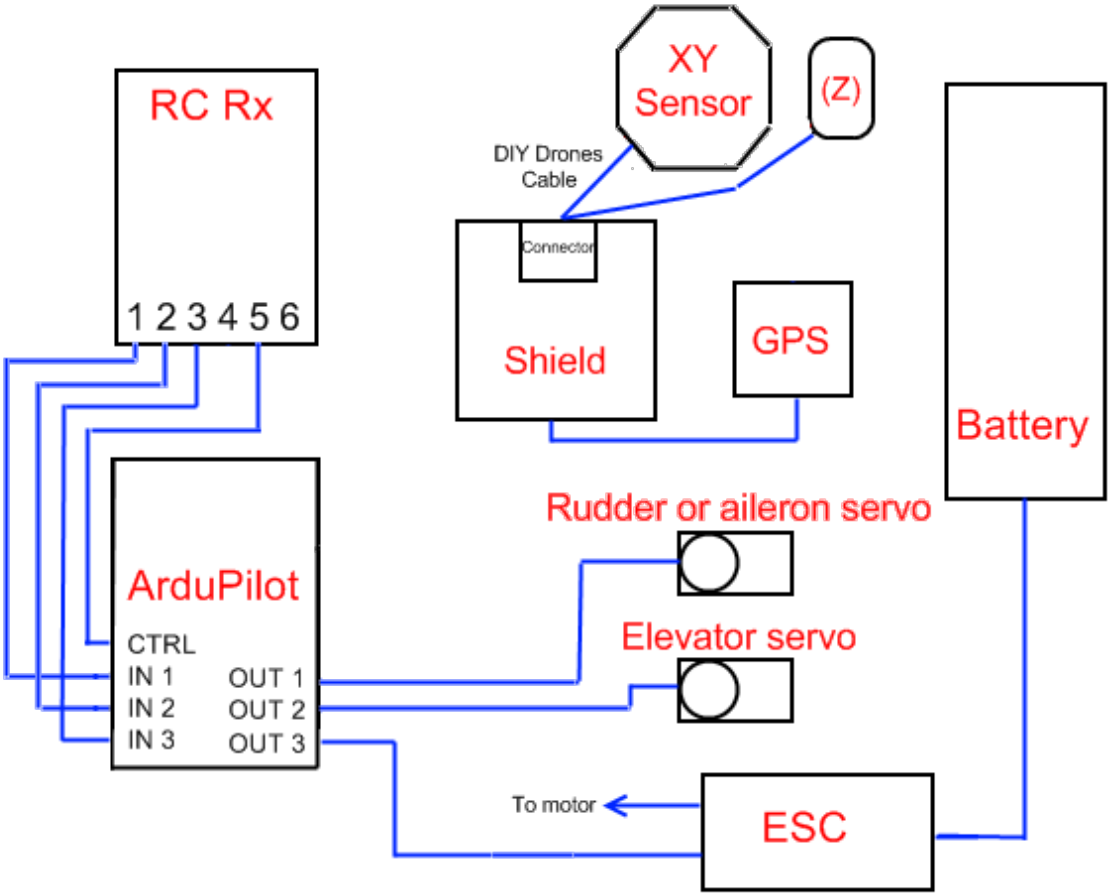
Now it is time to connect all the RC gear and sensors. We recommend the DIY Drones XYZ sensors. (You can also use two FMA XY sensors [they've discontinued the Z sensors]; just don't use one of the wires on the XY sensors you're using as a Z sensor, so you're just using one pair of thermopiles.)

Attach the ArduPilot inputs to your RC receiver with female-to-female cables (alternatively, you can cut those cables in half and solder them in the ArduPilot holes, making permanent "pigtails", as shown in some of the pictures above). Both the cables in from your RC receiver and the cables out to your servos should have the black wire (ground) on the pin closest to the edge of the board. Input 1 should go to your receiver's Channel 1 (aileron), Input 2 should go to Channel 2 (elevator) and Input 3 should go to Channel 3 (throttle). CTRL should go to the channel you'll use to enable the autopilot, usually 5 or 6. If you have a three-position toggle on your transmitter, use that channel.

The servos go to the Output side of ArduPilot. If you have a three-channel plane like an EasyStar, connect the rudder servo to Out 1; if your plane uses ailerons, connect those to Out 1. The elevator servo should be connected to Out 2, as shown below. The ESC/throttle is connect to Out 3. (Out 4 is not currently used)

Using the sensor cables that came with your shield kit, connect the four-wire one to the XY sensor and the three-wire one to the Z sensor. Plug the other side into the connector on the underside of the shield board.

Here is a diagram that shows the configuration (optional voltage measurement wire not shown):



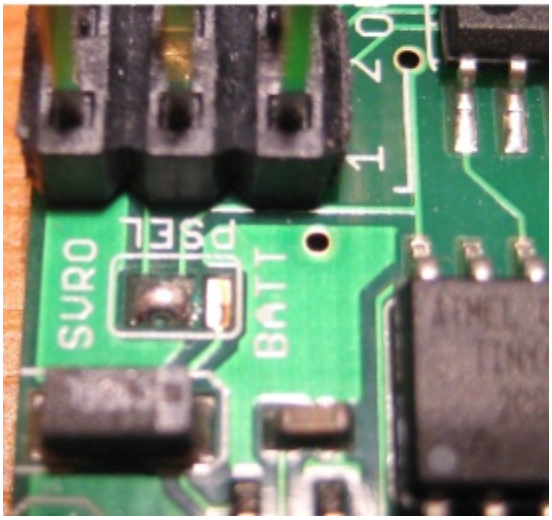
See Appendix 1 for a table showing all correct LED displays in each ArduPilot mode.

[Optional] Changing the power solder jumper
Although ArduPilot is designed to be powered by your ESC/RC receiver, it also has a built-in power regulator, so you can power it from a separate battery ranging from 5-15v if you want. To do this, you have to switch the solder jumper from the default RC power to battery power. This means unsoldering the solder jumper that the board ships with and resoldering it on the other side. These pictures show the difference (apologies from my crappy soldering on the second one):

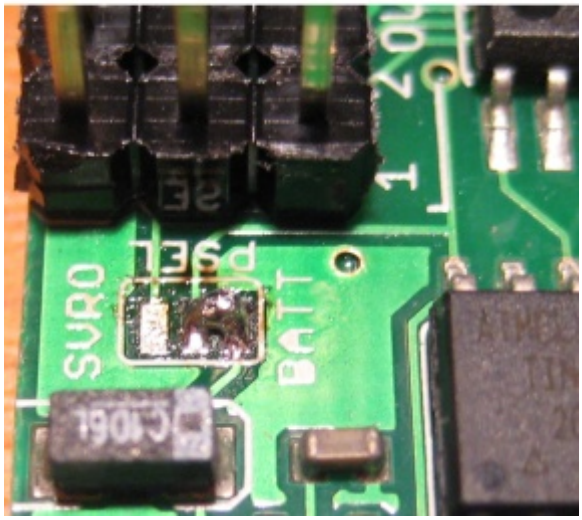
If you decide to make this change you must be aware of the following: Unlike the standard setting, if you use BATT input it will only power the ArduPilot board; it will not power your RC receiver or servos. They must have their own power supply, such as a receiver battery pack or ESC for an electric plane. This option is designed for people who fly gas planes (no ESC) and want the RC system on a separate power circuit so the autopilot can not drain that battery. Note that if you use this option, you must power your RC system (Rx and servos) separately, with another battery; the ArduPilot power regulator will not power the servos.

If however, you want to power both the RC system and the ArduPilot board from the same source, leave the board in the default setting and connect the battery straight to the RC receiver; the receiver will then power the ArduPilot board.

Normal (ESC) power setting



Battery power setting

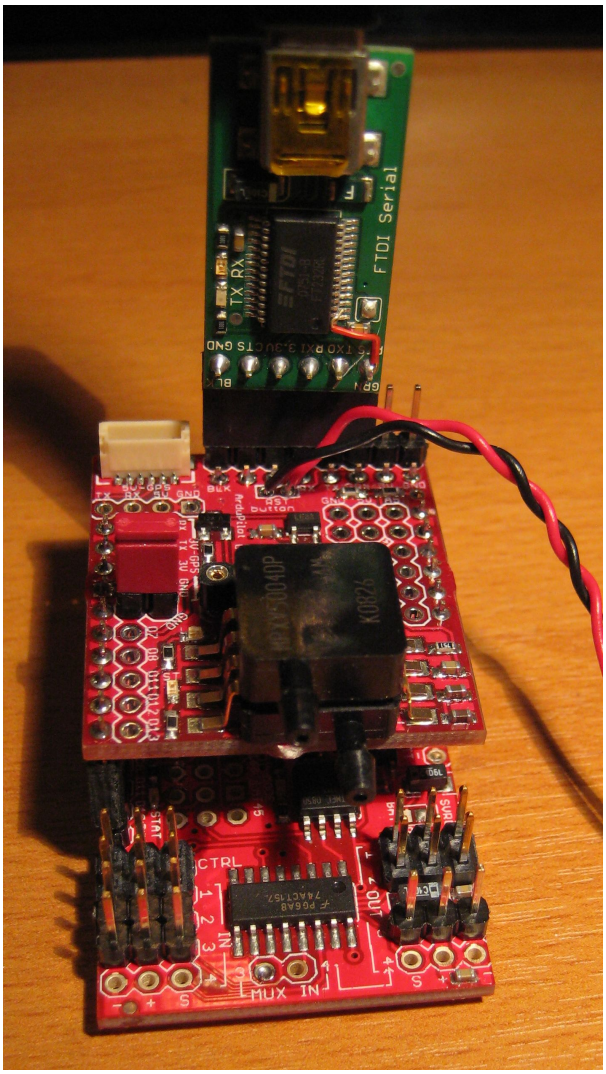


Loading the code

Now it is time to load the software. Download and install the latest version of Arduino, if you do not already have it. If you've never used Arduino before, please use these setup instructions to become familiar with Arduino basics. Once you've done so, you're ready to load the ArduPilot code.

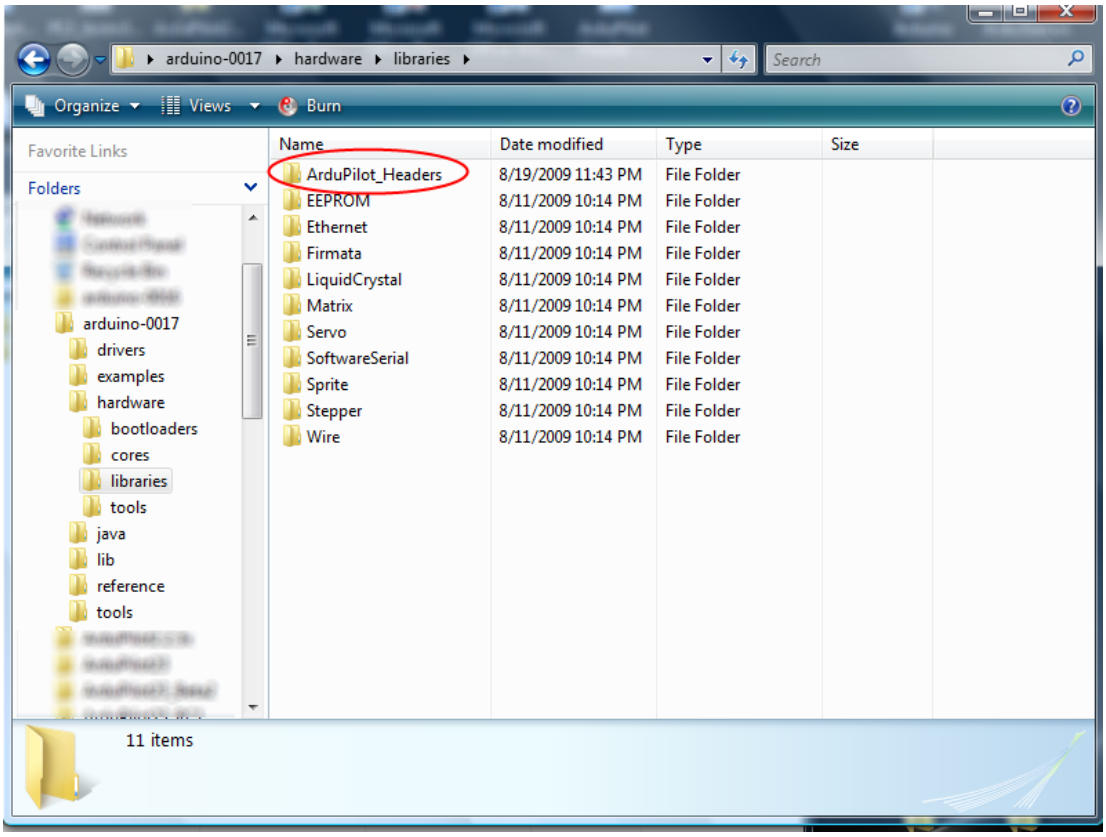
First, power on the board by plugging your ESC into a battery or using some other 5v power source (do not attempt to just power the board with the FTDI cable. We did not connect the power pins on the FTDI port to the processor to avoid power conflict when the board is powered by the Rx and you are using the FTDI as a serial monitor). The red power LED should go on. Now plug your FTDI cable into the board (with the black wire or, in the case of the Sparkfun board, the GND pin on the side marked "BLK") and plug it into your computer's USB port.

Here is what the Sparkfun FTDI board looks like when it is plugged in properly to the shield (if you're using the DIY Drones FTDI cable instead, ensure the black wires is on the pin closest to the GPS connector):



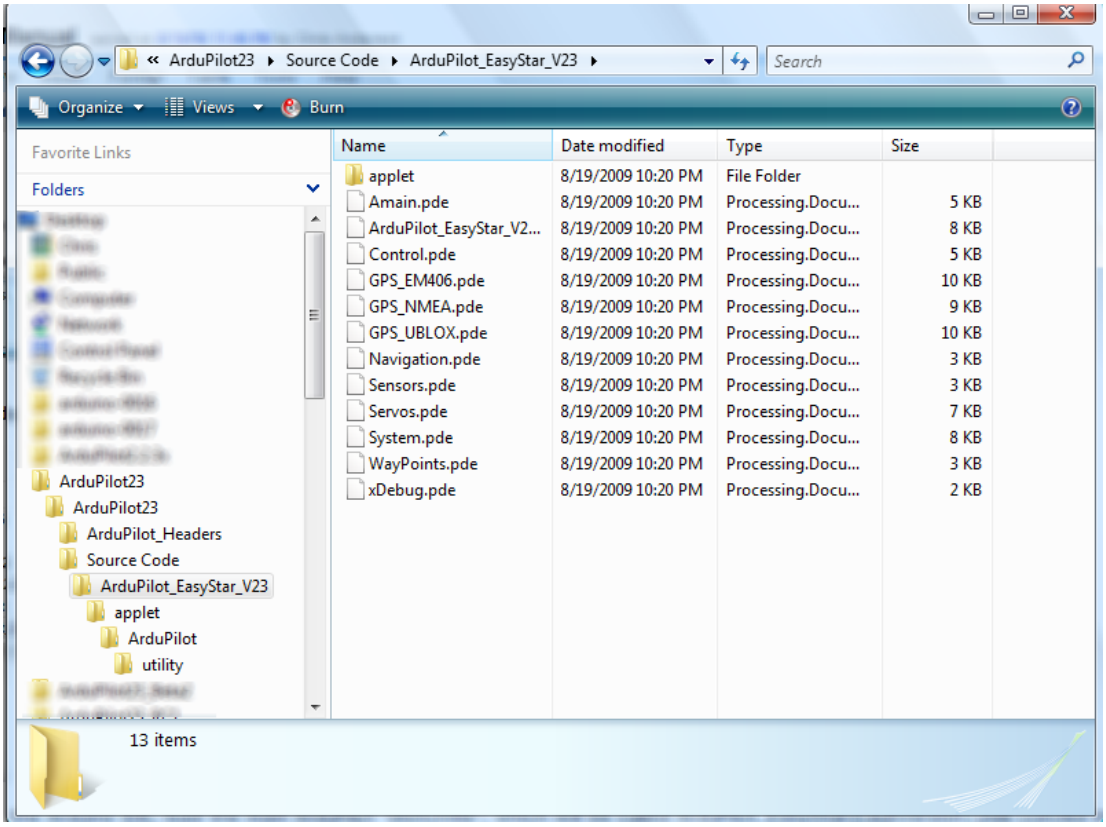
Download the latest ArduPilot code, which can always be found in the download section at right on the ArduPilot code page. If you have the current ArduPilot board (the red one, with the ATMega328 processor), you should download the latest code (currently 2.3). If you have the earlier green board with the ATMega168 processor, which has less memory, you will use 2.1. Version 2.1 is the last version that will support the green board, so if you want to advance with the project you will want to upgrade your board to the new version at some point.

Uncompress the code to a folder on your desktop. You will find two folders inside: one, the ArduPilot code, which will be called Source Code, and the other, ArduPilot_Headers, which contains an EasyStar.h file (and possibily other .h files). Move the ArduPilot_Headers folder to the \arduino-0017\hardware\libraries folder on your PC, as shown below (you can also put it in the \arduino-0017\hardware\tools\avr\avr\include folder if you're getting compile errors; if you do so, change the #include line in the first tab of the source code to "#include <ArduPilot_Headers\easystar.h>"). [Note, if you're using a Mac, the process is slightly different. Right click on the Arduino app

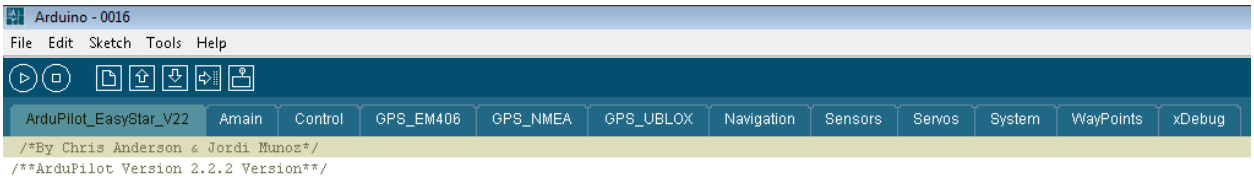


As you use ArduPilot on different aircraft, you may want to have an [airframe].h file for each one. Just ensure that the you edit the source code to call the right header file when you're uploading the code. So, for example, this is what I put in the first tab for my Superstar (replacing the easystar.h line): "#include <ArduPilot_Headers\superstar.h>".

Meanwhile, your code folder should look like this:



Now, with the the Arduino IDE, load the main ArduPilot "sketchfile", which will be called ArduPilot_EasyStar2[subversion].pde (circled in red above), which will load the rest of files in tabs (there are about 12 tabs in all). The tab bar will look something like this:



Then in the Arduino software in the "Tools" menu make sure you have selected the right serial port (the FTDI cable will create a new one, which is probably port 5 or higher). Also ensure that the board selected is "Arduino Diecemila or Arduino Duemilanove w/ATmega168" if you have one of the older 168-based boards, or "Arduino Duemilanove w/ATmega328" if you have the newer 328-based boards.

At this point unplug your GPS module if it was plugged in. The Arduino code will not load if the GPS module is attached, because they share the same serial port. Once you have uploaded the code, you can plug your GPS module back in. Please remember this in the future as you are uploading code: you should unplug the GPS first. [Note, with some GPS modules you can leave them plugged in while you're uploading code if you're using the shield. For us that works fine with the uBlox and EM406 modules, but it may not work for you. Test it and see; if it does work, it's a good idea not to unplug your GPS module every time, to save stress on the cable and connectors.]

Now press the "Upload to I/O board" icon (the little arrow point to the right). Nothing should happen for about 30 seconds as the code is compiling, then the ArduPilot LEDs aside from power will go out while the board is reset and the code downloaded. (If you are using the Sparkfun 3.3v FTDI board, you will see a red LED on that board flash a few times and stay on for about ten seconds as the code is being downloaded). In less than a minute, at the bottom of the Arduino IDE the software should report that the sketch was successfully uploaded by reporting "Done uploading". If not, and if you get a bunch of error messages in red, check out the Arduino debugging tips here.

Now you can disconnect the FTDI cable and reconnnect the GPS module. Your autopilot is ready to be integrated into an aircraft.

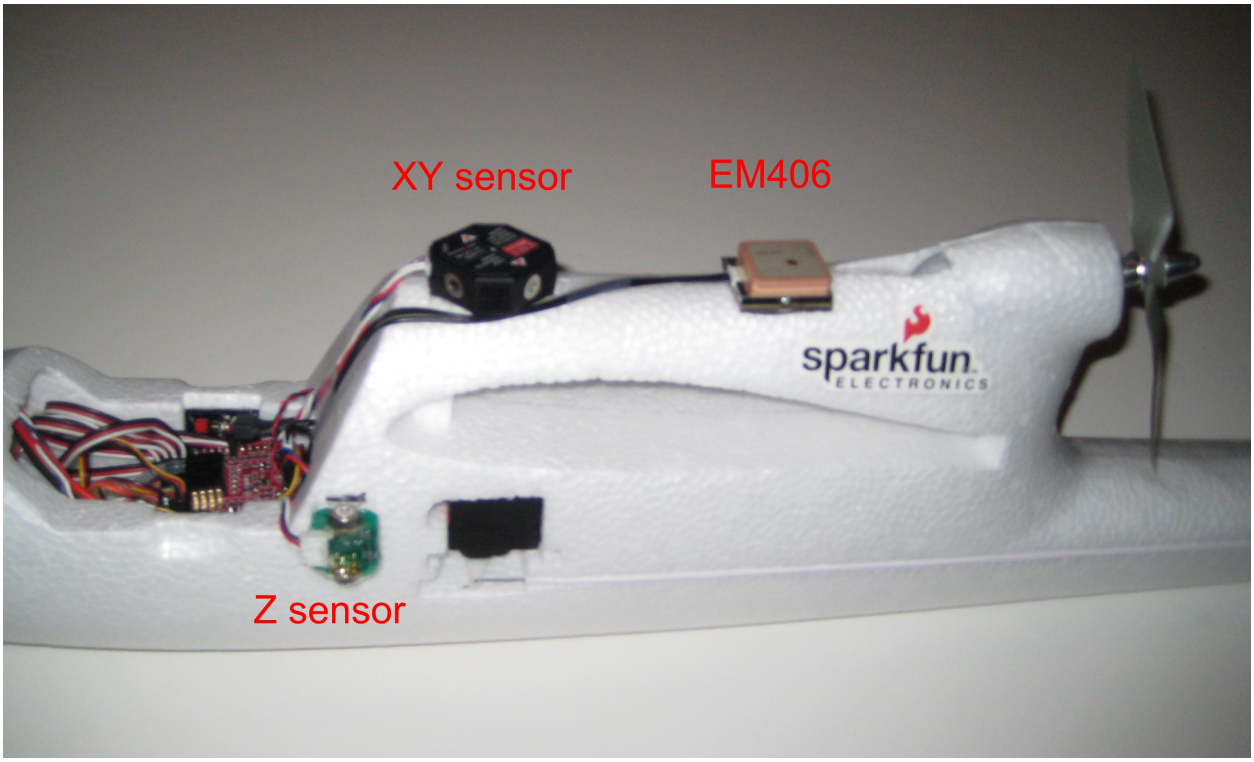
Aircraft integration

ArduPilot is optimized for the Multiplex EasyStar, a powered foam glider that is one of the most popular aircraft in the world, thanks to being cheap, easy to fly, easy to make and nearly indestructible (Lord knows we've tested that enough!). It also has loads of interior room for its size (thanks to a pusher motor and servos on the outside of the body) and good carrying capacity. It is in short, an ideal entry UAV platform. You can buy it at most hobby shops or online at retailers such as Tower Hobbies. We recommend that you upgrade the motor to a brushless and use LiPo batteries, which can be done with the Multiplex upgrade kit and an appropriate LiPo battery and charger (we like this battery and this charger/power supply combination, which will handle all your charging needs as you grow with the hobby).

You can use ArduPilot on other aircraft, of course, and it has been used on platforms such as the Multiplex Easy Glider and FunJet. But each aircraft needs to be tuned differently for an autopilot, and doing so is a bit of a trial-and-error process involving the settings in Appendix 2. Over time, we will release [airframe].h files for different aircraft, which will make that process easier. But for now we recommend you start with the EasyStar. Here's how to install the various parts:

The sensors:

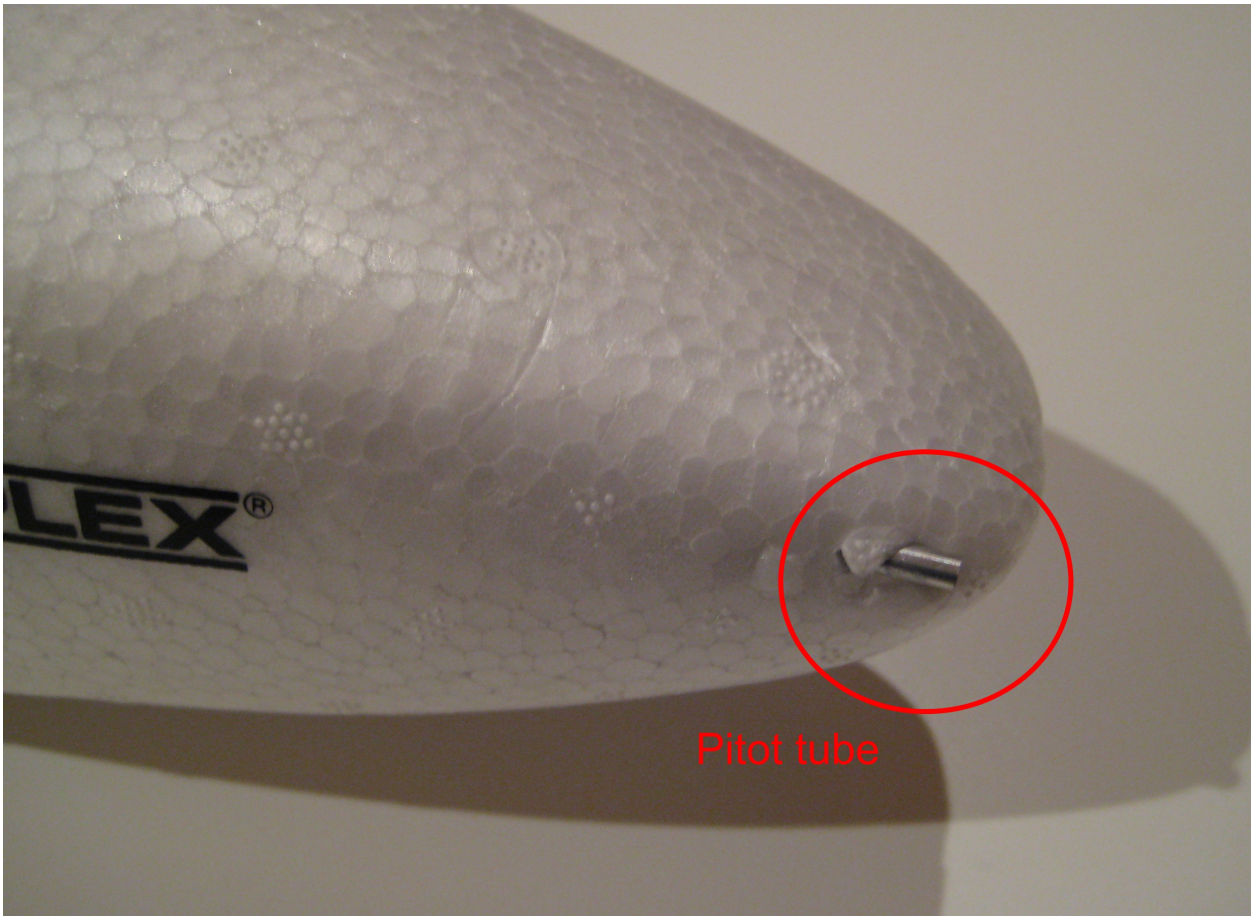
Use velcro or double-stick tape to put the sensors on the outside of the aircraft's body, with the XY sensor mounted diagonally. The cable can face forwards or backwards, but if you place it facing backwards, you must change setting 1-2 in the configuration file to 1 (see Appendix 2). Here's where I put mine. I just use an Xacto knife to carve a little flat spot in the foam where the XY and GPS modules go. I mount the XY in the diagonal position as shown in the picture, with the cable facing forward. The Z goes on the side, taking care to ensure that its view of the ground and sky is not obtructed by the wing when it's in place.



[Optional] If you want, you can test your sensors once they're in place and hooked up to ArduPilot. Here's a very simple Arduino program that will test your XY and Z sensors. Just load it on ArduPilot (make sure the board is powered and the GPS is not connected). With the FTDI cable connected, click on the serial monitor icon in Arduino and make sure the speed is set for 57600bps. You should read values around 512 with max min of 300 and 700 if you move it around. If you stop moving them the values should be steady. If the values are jumping around a lot or are very low your sensors may be damaged. Remember that sensor readings inside and near heat sources (like your hand) are nothing like the real thing outside. But it's still a good way to confirm that your sensor is working right. Once you're finished, reload the ArduPilot code.

If you're using the uBlox5 module with the helical antenna, it should be mounted vertically, with the antenna pointing to the sky. It can be inside your equipment area and does not need a clear view of the sky (just don't mount it under metal or carbon fiber). Here's how I have it mounted in a Superstar:

Now for the pitot tube. You'll need to push the tube through the foam in the cockpit so it protrudes into the airstream. I put mine pretty far in the nose as shown below, to get the cleanest airflow, but if (when!) I crash I'm going to bend that tube. You may want to mount yours a bit further back in the nose if you're expecting lots of hard landing. Connect the silicon tubing the aluminum tube and twist/push it through the foam until it emerges where you want it.



Connect the other end of the tubing to the top nipple of the pressure sensor on the expansion board. Leave the bottom nipple open (it will record ambient pressure in the cockpit).

If you are using ArduPilot in an aircraft with the propeller in the nose, the pitot tube must be mounted out on one wing, at least a foot from the fuselage to be outside the prop flow.

ArduPilot Settings

Now it's time to tell ArduPilot about the placement and equipment decisions you've made.

In the above case, with the XY cable facing forward, we modify the following line in the ArduPilot.h file (in any text editor, such as Windows Notepad) as follows:

```
//1-2
#define REVERSE_X_SENSOR 0 //Thermopile XY Sensor, 1 = cable behind, 0
= cable front
```

If you're using the EM406 GPS module, which uses the SirfIII chipset, make sure the following line is set to SIRF mode (this is the default):

```
//1-7
#define GPS_PROTOCOL 1 //0 = NMEA, 1=SIRF, 2=uBlox, Choose protocol,
uBlox only for PRO's please.
```

If you're using the uBlox module, that line should be changed to this:

```
//1-7
#define GPS_PROTOCOL 2 //0 = NMEA, 1=SIRF, 2=uBlox, Choose protocol,
uBlox only for PRO's please.
```

Checking for reversed servos

Now it's time to check your system and see if any servos need to be reversed. This is something you can do on the ground (outside, ideally on a sunny day) by tilting the plane and watching the effect on the control surfaces. First, put the field calibration jumper on D6 and GND pins you soldered above, or attach the bind plug to the cable you soldered to those holes. Then power on the autopilot, waiting for GPS lock as described in the field calibration section below. Once that's done, remove the jumper or bind plug.

At this point you should be able to hold the aircraft at arm's length and roll and pitch it and see the rudder/aileron and elevator responding to the sensors. They should move in the opposite direction from the direction you're tilting the aircraft: if you pitch the plane down, the elevator should go up. If you roll to the left, the rudder or ailerons should move to try to roll the plane back to the right.

Indoors, you can also simulate this with your hand (no need to go through the field calibration process above): placing your palm in front of a pair of sensors on the fore, aft, port or starboard sides simulates the IR signature of that side tilted down to the ground (the ground tends to have a "hotter" IR signature than than the sky). So if you put your hand in front of the XY sensor (which looks to the autopilot like the plane is pitching down), the elevator should go up. If you put your hand behind the sensor (simulating pitching up), the elevator should should go down. Likwise for the two sides--the rudder should turn in the opposite direction from the side your hand is on).

If any control surface movement seems reversed, you need to tell the autopilot to reverse the movement of that servo. Do so with the following lines

```
//1-4
#define REVERSE_ROLL 0 //To reverse servo roll.
//1-5
#define REVERSE_PITCH 0 //To reverse servo pitch.
```

[optional] Ground Testing with the Fixed Direction Mode

If you want to test the autopilot further on the ground, we have included a special software mode to allow this. In previous version of the software, we had special "NorthEast" code that would attempt to always steer the plane north-east for testing purposes. In version 2.2 and above, this mode can be enabled with a simple settings change in the EasyStar.h (or whatever airframe you're using) file:

Set the following lines as per the below:

```
//9-1
#define FAKE_BEARING 1 //If set to 1, will fake the bearing and will
try to go always head to the defined DESIRED_FAKE_BEARING
//9-2
#define DESIRED_FAKE_BEARING 45 //Will try to go NorthEast, you can
change that to 0 = NORTH, 90 = EAST, 180 = SOUTH, 270 = WEST or
whatever!

//9-6
#define WALK_AROUND 1 //Must be "0" to test the GPS and heading
against the servo and "1" for normal operation
```

[Walk-Around mode disables stabilization, turning ArduPilot into a navigation-only autopilot. This is also useful if you want to use ArduPilot in ground vehicles that don't require stabalization, or if you want to use ArduPilot in an aircraft that already has its own stabilization system, such as the FMA CoPilot]

Save the file, then open the Arduino IDE, load the ArduPilot code and download it to ArduPilot.

Now walk outside with your plane (with all the RC stuff hooked up) and power up, waiting until you get a GPS lock (solid blue LED). Once you've got that, toggle the autopilot on and start walking around at a brisk pace in big circles. You should see the rudder (assuming you're using the EasyStar; ailerons if you you're using a plane with them) try to steer you so you're going NorthEast.

When you're done with the ground testing, return the autpilot to its normal mode by editing this line as follows and re-uploading the code to ArduPilot:

```
//9-1
#define FAKE_BEARING 0 //If set to 1, will fake the bearing and will
try to go always head to the defined DESIRED_FAKE_BEARING
```

Setting your autonomous modes

ArduPilot has two modes: programmed waypoints and return-to-launch (RTL). In the first one, you enter the GPS Latitude and Longitude and altitude coordinates as waypoints in the setup utility. In the second, you do not enter in any waypoints and the aircraft just returns to the Lat/Lon it was at when you first powered on the board (at your launch location).

By default, the code is set up to be enabled by transmitters with a three-position toggle switch (usually channel 5 or 6) and the three modes are as follows:

	Mode LED (green)	**MUX LED (red)**	**Function**
Down position:	Off	Off	Autopilot off
Middle position:	On	On	Waypoint mode
Up position:	Off	On	RTL mode

(If you only have a two-position toggle, it's possible to simulate the middle position by mixing together two channels on your RC transmitter, so, for example, channel 5 up and channel 6 down is "middle" while channel 5 up and channel 6 up is "up". Please consult your RC system's manual for instructions on how to do this). You can also achieve a middle position if you have a proportional dial on one channel, and you just turn it half way).

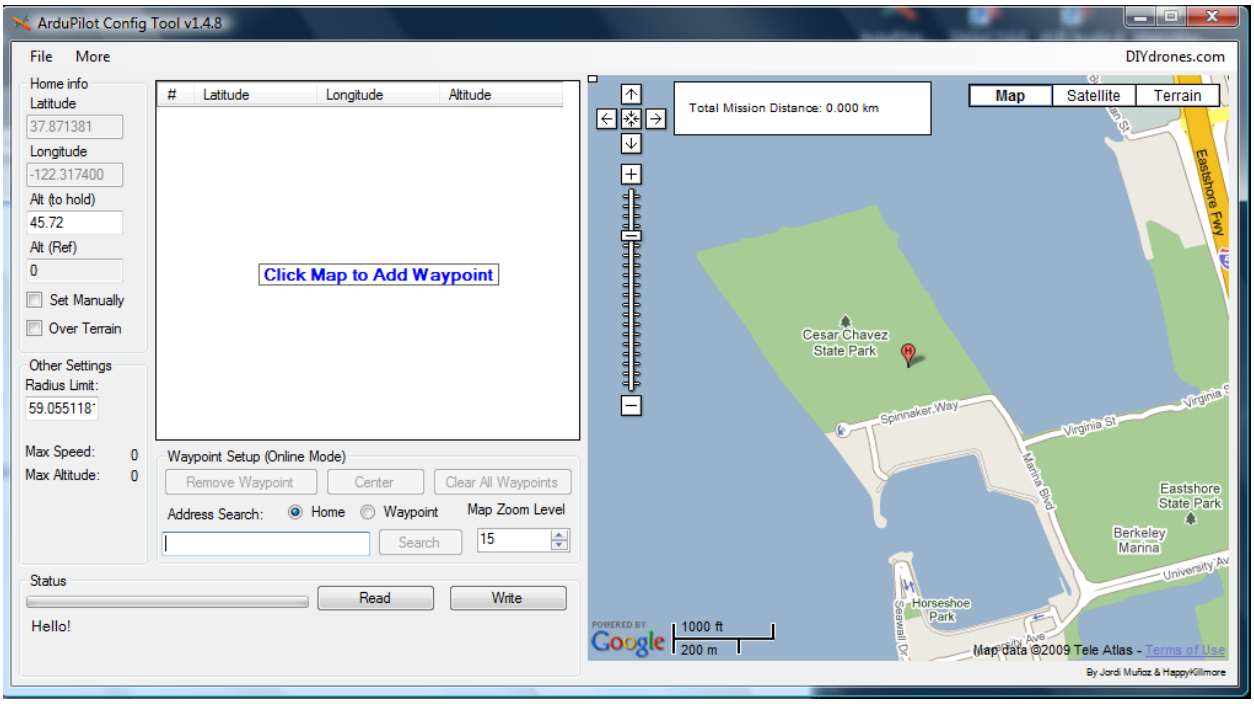
If you want to change the behavior of the toggle switch, do so with this line in the ArduPilot.h file:

```
//1-6
#define RADIO_SWITCH_ACTION 0 // 0: TX Switch centered = waypoint mode
& full = RTL mode. 1: TX Switch centered = RTL & full = waypoing mode.
```

Using the Setup Utility

Once you've got the basic autopilot set up, it's time to start planning missions. Download the ConfigTool.zip file from the Google Code repository and decompress it to your desktop. [If you have any trouble running the software, make sure that your PC is running Microsoft .Net Framework 2.0]. Connect your FTDI cable and power on your board as if you were going to upload code, powering on the board and disconnecting the GPS to avoid serial conflicts.

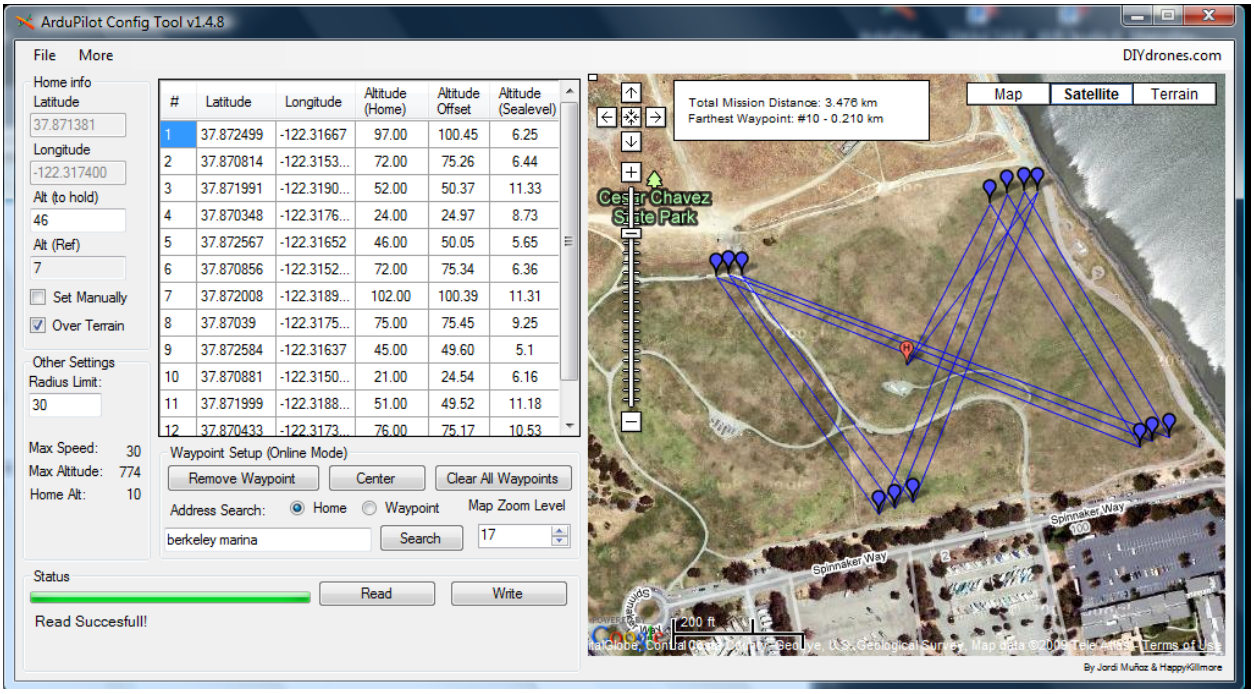
Windows will probably ask you to approve the software accessing the Internet once or twice, click okay each time. It should look something like the below. If not, close the program and reopen it.



Now select the port assigned to your FTDI cable and the ArduPilot board you've got (green board is ATmega168; red board is ATmega328). Initialize the board by clicking on "write" before doing anything else.

Inputting waypoints is easy--just click on the map. You can add up to 48 waypoints.

Here's what mine looks like with a 14-waypoint pattern with 3D (lat, lon and altitude) waypoints:



Once you've entered all the waypoints, click "write" again to store them in ArduPilot's non-volatile memory (they will be retained even after you power down). You can test to see if they've been entered correctly by clicking "read". Your waypoints should now display on the map in the utility. Note: if you haven't already done a field setup process to record your home position, you should enter that manually.

With Over Terrain unchecked, you can just enter an altitude and start clicking your waypoints. The ArduPilot will then try to maintain that altitude above the home altitude. So if you put in 150 and you have feet selected, it will try to hold 150 feet above home regardless of what the ground looks like.

With Over Terrain checked, you can enter that same 150 ft in the altitude and it will ask a government website for the altitude at each of your waypoints (shown in the altitdue (sealevel) column plus the 150 (which would be shown in the Altitude Offset column which equals the Altitude (Home) column. This is what actually gets passed to the ArduPilot. So it will try to fly 150 ft above all the invidivual waypoints, not just above the home waypoint.

The "radius limit" defines how closely the aircraft has to come to the waypoint to be considered a hit.

The "Max Speed" and "Max Altitude" data displays are a form of datalogging--they show the maximum speed and altitude achieved in the most recent flight. You can clear them with the "Clear Max Alt/Speed" menu item in the More menu.

"Roll Trim" and "Elevator Trim" are to set your autopilot to compensate for aircraft tendencies to drift one way or another. This is most useful if your XY sensor isn't mounted quite level.

The Config Utility can add "smart" altitudes for each waypoint, which is to say that it knows the terrain for any location and how far it is above sea level. So you can plan a course and if

you set the altitude of a waypoint as 100 meters, for example, it will be 100 meters above that location, not just 100 meters above your launch location. This is especially useful for hilly terrain. If you don't intend to enter 3D waypoints, clear the "Over Terrain" checkbox and two of the three altitude columns will disappear.

You can save and load as many missions as you want. All other usual Google Maps features are enabled, as well as the ability to work in feet or meters.

Because waypoints are retained in non-volatile memory (so it will be retained, even when you power down) and the config utility requires an Internet connection, people usually plan their mission and load the waypoints at home, before they go to the field.

[optional] Adding wireless telemetry

Adding wireless telemetry is not difficult and can extend the capabilities of your UAV immensely.

The first thing to keep in mind is that you should use Xbee modules in a different frequency range than your RC equipment.

If you have 72Mhz RC gear, you can use 2.4Ghz Xbee modules. In that configuration, we use these Xbee Pro wireless modules with a Adafruit adapter board on the aircraft side, and a Sparkfun USB adapter board on the ground/laptop side. But please note that the only reason to use 2.4Ghz Xbee equipment is if you also want to use 2.4Ghz video transmission; otherwise, it's better to use 900Mhz Xbees (see next paragraph), which have longer range.

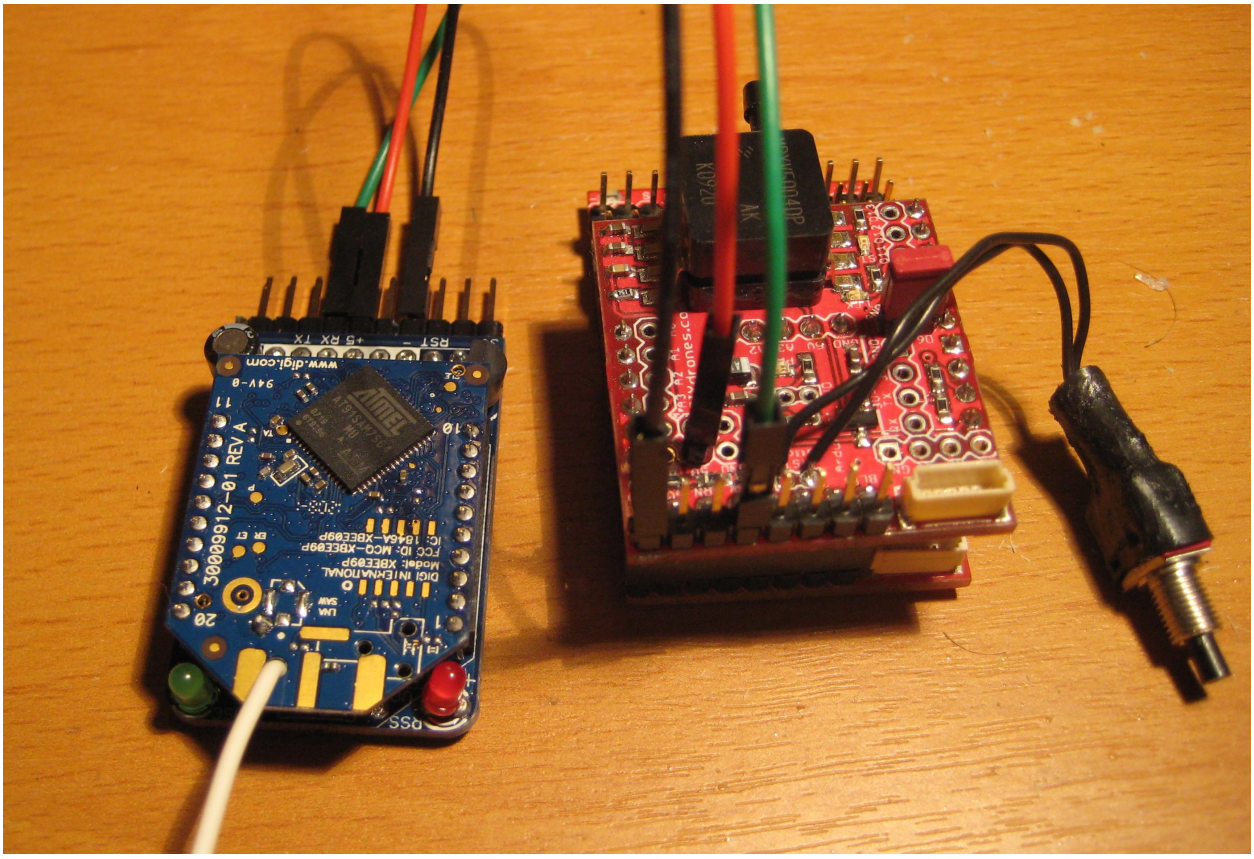
If you have 2.4Ghz RC gear, you should use 900Mhz Xbee modules. In that case, we use this Xbee Pro with the wire antenna for the aircraft, and this Xbee Pro with a SMA antenna connector (and a good 900Mhz antenna) on the ground, with the same adapter boards as above. Alternatively, you can use our all-in-one ArduStation hardware on the ground and dispense with the adaptor and laptop entirely on that side.

Next, you need to set up your Xbee modules. They ship with a default of 9600bps, which you must change to match the ArduPilot's serial speed. That depends on which GPS module you're using. The EM406 runs at 57600bps, while the uBlox runs at 38400bps; set your Xbee modules to match your GPS module. If you are using a different module in NMEA mode, set the serial speed to whatever you've set the code to use with the GPS module (the default is 9600)

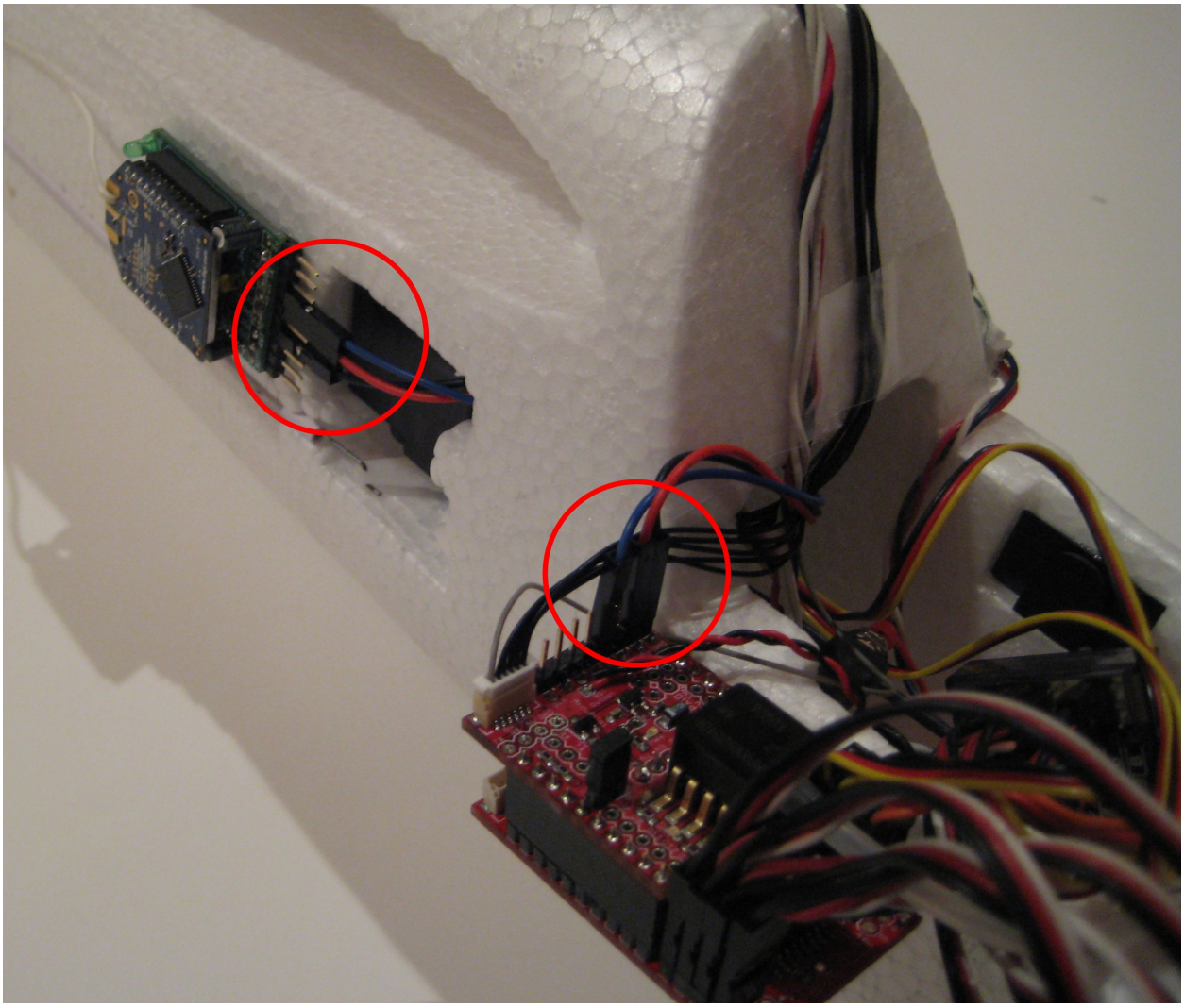
Connect each one of the them to the Sparkfun USB adapter board, plug the USB cable into your PC, and use Digi's X-CTU utility to select the right serial port and communicate with them. Remember to initially set the utility to 9600bps to contact the new Xbee modules, and than after you've changed the speed, change the utility's serial speed accordingly. You should also give the modules unique Network IDs so they will be paired. Just use any 3-digit number, and just make sure you have set it the same on both modules. (**Note:** If you will be flying near other UAV planes make sure to verify the Network IDs are unique and not used by others in your vicinity.)

On the ArduPilot side, use three jumper wires to connect the pins shown below.
--Xbee RX to ArduPilot Shield FTDI port TXO

--Xbee 5v+ and GND to the GND pins next to the ArduPilot Shield FTDI port and the special 5v pin you solder on in the shield assembly step.



It will end up looking like the below when you've got it connected on the plane (this picture shows an alternative pin to use to get Xbee power from the ArduPilot Shield; it's noisier so we generally recommend the pin used in the description above instead). I thread the jumper cables in through the servo chamber; if you find that the proximity of the data cables is making your servos chatter, place the cables further away from the servos.



On the ground side, connect the other Xbee module to your laptop with a USB cable.

That's it. If you open up a terminal program on your laptop (you can use the Arduino IDE's serial monitor for this, too), select the correct serial port, and set the baud rate to whatever you set the Xbee modules to above. Once you do this, you should see ArduPilot telemetry coming in. Anytime there is a "Serial.println" in the code, that data will be sent through the Xbees to the ground. You can record any data you want, and even datalog from the ground! You can also open the Ground Station software, setting the right port and baud speed) and it should begin to show ArduPilot data.

Additionally, if you want to test the range of your Xbee link, connect the plane-side Xbee module's RX and TX pins together to create a loopback circuit and use the X-CTU utlitity's range test function. For the modules we are using you should get around a mile.

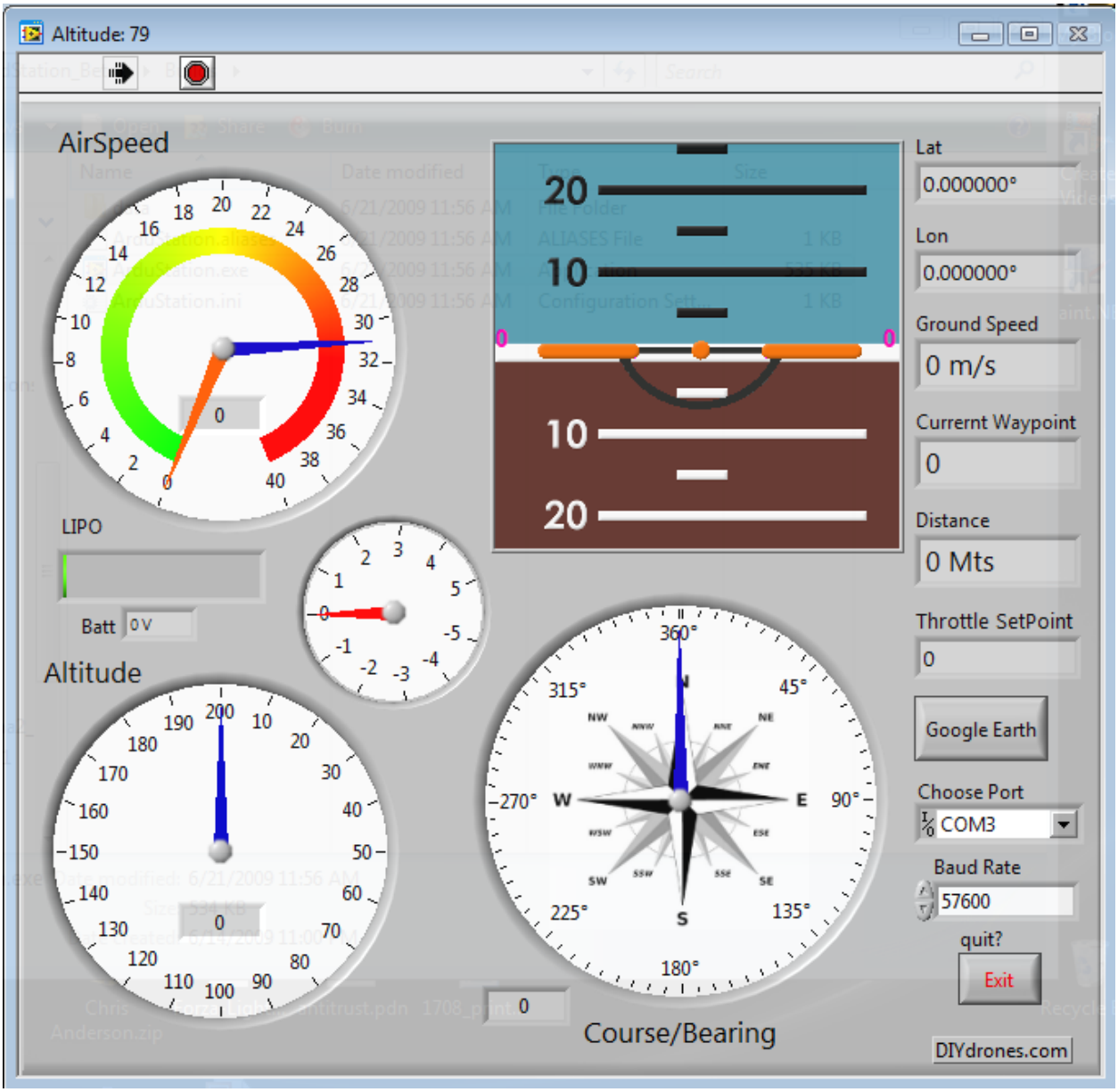
IMPORTANT NOTE: Sometimes Xbee modules get corrupted due to signal coming in before power on bootup. To avoid this, ALWAYS disconnect the signal wire to the onboard Xbee adapter before powering up. Only reconnect them after the rest of the UAV has power. If

you're finding that yours stops working (green LED on Adafruit adaptor doesn't come on), instructions to reload the firmware are here. Ensure that you've selected the right Xbee types or the firmware won't load (the 900Mhz ones recommended above are "XBP09-DP"). After you reset it to its default settings and confirm that it's working, make you sure you reset its speed to 57k and ID number to match your other module.

[optional] Setting up the Ground Station

Once you have the Xbee telemetry set up, you can use the ground station. (Note: if you are using our ArduStation hardware instead, please see those instructions and disregard this section). Download the executable code from the Google Code repository. It's called "GroundStation.zip". Unpack the directory to your desktop.

You'll also need the LabView runtime engine and serial drivers. Download the runtime engine here and the drivers here. Install them. Now you can run ArduStation.exe in the groundstation folder. After telling Windows to allow the software access to the Internet, you should see a screen like the following:



Ensure that your ground-based Xbee and the USB adapter board are connected and plugged into your PC. Select the correct serial port in the Ground Station software and set the serial speed to match the speed you set your Xbee modules to (which are, in turn, determined by your choice of GPS module). If you are using the EM406 GPS, you can leave it at the default 57600. If you are using the uBlox, set it to 37800. Click on the little check box in the top left corner of the window after changing any data. If ArduPilot is running, you should start to see the data displayed in real time on the ground station.

If you press the Google Earth button and have Google Earth installed (if not, you can get it free here), it will open in a separate window and the position of your aircraft will be shown

by an aircraft icon. This updates in real-time, so that window will serve as a moving map of your aircraft's current location.

If you are using the hardware-based ArduStation instead of this software, please see the operating instruction for that here.

Field calibration

ArduPilot 2.2 and above (with Z sensor):

This version of the software auto-calibrates the IR sensors. The only thing you need to do at the field is to record the "home" GPS position in EEPROM memory, which will not be erased when you reset or power down the board. To do so, follow these instructions:

1. Ensure that the autopilot is in manual mode (channel five toggle off) and GPS is connected.
2. Place the bind plug or jumper cap on the field calibration pins. Ensure that your aileron and elevator sticks are in the center position, and then apply power to the board.
3. If you have an EM406 GPS module, the yellow status LED will blink for a few seconds as the autopilot programs the GPS for binary mode (the uBlox module does not need such a programming step). Next the blue lock GPS LED will blink rapidly, which means the autopilot is waiting for GPS lock. You can now remove the jumper (or return the switch to the normal postion).
4. When the autopilot has established a GPS lock the blue LED will turn solid and the the autopilot will start moving the rudder/ailerons and elevator until you reset the board. **Note 1**: If you are doing this for the first time, or the first time in a few weeks, you may find that it takes as much as 15 minutes to get a sat lock. Also, if you're doing this indoors, you may never get a sat lock and the blue LED will not stop blinking. In that case, try again outside. **Note 2:** The ArduPilot puts the EM406 into binary mode, which disables the red LED on the GPS module itself, so it will remain very dimly lit and will not blink. **Note 3**: If you have an EM406 and it's not going into binary mode, check line 1-7 in the configuration file; it should be set to 1; if you have a uBlox, ensure that that line is set to 2.
5. You are now ready to fly! Press the reset button on the ArduPilot board to restart it (your calibration settings are saved in permanent memory and will not be erased).

ArduPilot 2.1 (no Z sensor):

This version requires that the IR sensor be calibrated in the field, on arriving and if the weather conditions change significantly:

1. [As above]
2. [As above]
3. [As above]
4. [As above]
5. Holding the aircraft without obscuring the thermopile sensor, point the nose at the ground. Switch the autopilot on with your RC toggle switch. The elevator will move, signaling that the sensor has been calibrated. You can now remove the jumper or bind plug.
6. You’re now ready to fly! Switch the aircraft back into manual mode for launch, and press the reset button on the ArduPilot board to restart it (your calibration settings are saved in permanent memory and will not be erased).

Sensor calibration FAQ:

Q: Are the sensors calibrated only when the bind plug/jumper is on or also with every reset?

A: Only when you insert the bind plug/jumper, then it will be recorded in the eeprom. If you restart you don't have to calibrate again (over short periods of 15 mins, battery change for example).

Q: Does the sensor calibration happen after GPS lock or before?

A: The answer is after. The calibration happens exactly when you remove the bind plug, takes like 1 second.

Q: What happens after you take off the bind plug? Any additional calibration?

A: Well, it will read the Z sensor for calibration and store all the values in the eeprom, which takes no time at all. Just be sure the airspeed sensor is not against the wind (in strong winds).

Q: Should the airplane be held in the air while the sensor calibration is going on?

A: The airplane can be in the ground. Just stay away from the z sensor when removing the bind plug.

Q: Any other calibration tips?

A: Always switch to autopilot when you are in ground, to see if the elevator and rudder responds. Remember that the Z sensor calibrates over time. If you leave the aircraft (even in manual) the sensor will calibrate better. Also try to apply pressure on the pitot tube to test the throttle vs airspeed.

Fly-By-Wire Mode

ArduPilot includes a "fly-by-wire" mode, which stabilizes RC flight and returns the aircraft to level flight when you release the sticks, in a similar way to the the FMA Co-Pilot (yes, when you get an ArduPilot, you get a FMA Co-Pilot for free!). To use fly-by-wire mode, simply disconnect the GPS. ArduPilot also automatically switches into stabilization mode if it ever loses GPS lock. (Note that even in autonomous flight, the plane will read and respond to your commands somewhat, so you can "nudge" it.)

In this mode, we recommend that you plug your ESC directly into your RC receiver's throttle channel, rather than into ArduPilot. This will give you manual control of the throttle during fly-by-wire mode. If you leave the ESC plugged into ArduPilot, the software will control your throttle, attempting to maintain altitude, but you will not be able to control the throttle yourself.

If you're finding that you seem to have little or no manual control in this mode, change settings 2-1 and 2-2 in the configuration file. A good starting number for the gains is 8.

Tuning ArduPilot for your aircraft

Despite what you may have heard, there are no plug-and-play autopilots in the world ;-) Even the best autopilots need to be tuned for different airframes, configurations and use patterns, and doing so is a bit of a black art. ArduPilot is distributed tuned for a standard EasyStar in low wind conditions, and you should be able to use it pretty much as is for that. But just as every RC plane requires a little trimming to fly well, you'll find that your autopilot may peform better if you tweak various settings.

Our recommendation is that while you're first setting up ArduPilot for your aircraft, you do NOT use it to control the throttle. Rather than connecting the ESC through the ArduPilot, connect it straight to your RC receiver as usual. That way you can control the throttle manually while ArduPilot is controlling stabilization and navigation. The main reason for this is that it's very difficult to tune aircraft in the wind, especially if the throttle settings are not sufficient to make headway (when the aircraft is heading into the wind with a insufficient throttle setting, its ground speed may drop to zero, which will confuse the GPS navigation, which is based on ground speed). Manually keep the throttle high enough to make headway into the wind, and you'll find it much easier to observe and tweak stabalization and navigation gains.

For aircraft other than the EasyStar, you'll definitely need to change setting to get acceptable performance. Over time, we'll release additional [airframe].h settings files for other aircraft, starting with the EasyGlider, SuperStar EP and Funjet. Community members will no doubt share files for their own aircraft and over time this tuning process will get easier.

[The ArduPilot 2.3 code does not use the D term in the PID, so you only need to modify the P (proportional) and I (integrator terms). You can learn more about PID theory here.]

Here are some general guidelines on modifying the PID gain settings in Appendix 2:

First, the best way to set up stabilization is to do so in Fly-By-Wire mode. Disconnect the GPS on the ground, and then when you switch the autopilot on in the air, it will simply stabilize your RC controlling. Try it with your hands off the stick and then see how it recovers from rolls and pitching.

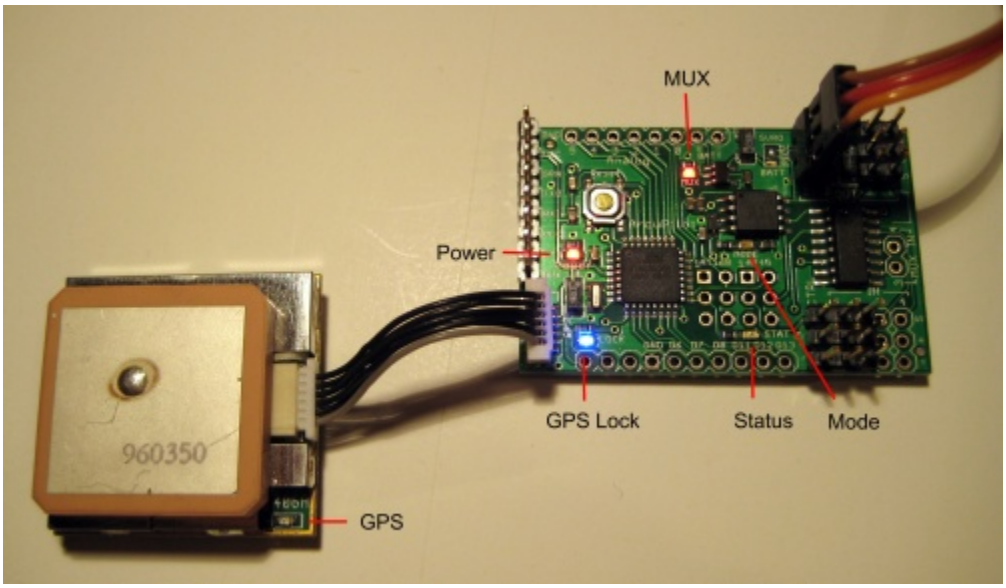
A general rule of thumb is to increase the proportional term until the plane starts to wing rock (roll) or porpoise (pitch), then reduce the P term to half that value. Then increase the I term until sufficient stability is achieved.

You can start with the default settings in the EasyStar.h file, but if you really want to optimize the autopilot for your aircraft and not overcontrol it. We recommend you start from very low values and build up. For example P=0.1 and I=0. And then increase P slowly until you see better response (.1).

Next, you can test navigation by trying RTL (Return to Launch) mode. This is the default mode for the up position of your RC transmitter's toggle switch. Point your plane away from you and switch into auto. It should return to you and circle overhead. Try from various directions, so you can see how the plane handles navigation into the wind, downwind, and crosswind. If everything checks out, you're ready to try waypoints.

[More on this to come]

Appendix 1: Correct LED behavior



The correct ArduPilot LED displays are as follows. "GPS LED" refers to the one on the EM406 module. The uBlox module does not have a LED. See the above picture for reference:

[NOTE for those using the EM406 GPS module: when you first plug in your GPS, the red LED on it will glow brightly when it is powered on. But when you go through the first setup procedure with ArduPilot (using the jumper/bind plug) the code will program the GPS into binary mode. This disables the LED (it still glows very faintly), which is perfectly normal. If for any reason you want to return your GPS module to its default NMEA mode, just unplug it and leave it for a week or so. The internal capacitor will discharge and it will return to its factory defaults.]

First power-up of board, straight from the factory:

Power:	**On**
Stat:	**Flashing in sequence**
GPS Lock:	**Flashing in sequence**
MUX:	**Off unless a RC receiver channel is connected to CTRL pins and that channel is toggled on**
Mode:	**Off**

ArduPilot 2.x code loaded, autopilot in manual mode, configuration jumper/bind plug on, GPS connected

Power:	**On**
Stat:	**Will blink twice as GPS is programmed into binary mode, then blink steadily**
GPS Lock:	**Flashing while it attempts to aquire a lock; solid afterwards**
MUX:	**Will turn on briefly during boot, then stay off**

Mode:	**Off**
GPS LED:	**Off**

ArduPilot 2.x code loaded, autopilot in auto mode 1 (middle position of toggle switch), jumper removed, GPS connected

Power:	**On**
Stat:	**Flashing (indicates loop running correctly)**
GPS Lock:	**Flashing while it attempts to aquire a lock; solid afterwards**
MUX:	**On**
Mode:	**On**
GPS LED:	**Off**

ArduPilot 2.x code loaded, autopilot in auto mode 2 (high position of toggle switch), jumper removed from digital pins 6&7, GPS connected

Power:	**On**
Stat:	**Flashing (indicates loop running correctly)**
GPS Lock:	**Flashing while it attempts to aquire a lock; solid afterwards**
MUX:	**On**
Mode:	**Off**
GPS LED:	**Off**

Appendix 2: User-configurable settings (ArduPilot 2.2 and above)

When compiling the code, ArduPilot reads all user-configureable settings from a header file called EasyStar.h in the \\arduino-0016\hardware\libraries\ArduPilot_Headers directory.

If you want to create a custom configuration file for your own airframe, say the Superstar, you can rename the configuration file and change this line in the first tab that calls it. So, for instance, in the Superstar example, you'd change that line to:

```
#include <ArduPilot_Headers\superstar.h> //Loading the  settings.
```

The file is plain text and can be edited in any text editor. The following are the lines in the file, and what they do:

```
  /*************************************/
 /*ArduPilot Header file, good luck!*/
/*************************************/

//Hardware Configuration
//0-1
#define SHIELD_VERSION 1 // Old (red) shield versions is 0, the new (blue) shield version is 1

// Airframe settings
//1-1
#define ALT_HOLD_HOME 1 //What altitude you want to hold for home, 0 = the one you are when you switch to RTL, 1 = the altitude defined by the configuration tool.
//1-2
#define REVERSE_X_SENSOR 0 //XY Thermopiles Sensor, 1 = cable behind, 0 = cable in front
//1-3
#define MIXING_MODE 0 //Servo mixing mode 0 = Normal, 1 = V-tail (v tail not tested yet).
//1-4
#define REVERSE_ROLL 1 //To reverse roll servo, change to -1
//1-5
#define REVERSE_PITCH 1 //To reverse pitch servo, chance to -1
//1-6
#define RADIO_SWITCH_ACTION 0 // 0: TX Switch centered = waypoint mode & full = RTL mode. 1: TX Switch centered = RTL & full = waypoint mode.
//1-7
#define GPS_PROTOCOL 2 // 0 = NMEA, 1=SIRF, 2=uBlox, Choose protocol
//1-8
#define ATTITUDE_RATE_OUTPUT 250 // The output rate of attitude data in milliseconds. Useful if you want to increase the output rate.
```

```
//1-9
#define POSITION_RATE_OUTPUT 4  //This number will be multiplied by ATTITUDE_RATE_OUTPUT, the result is the refresh rate in milliseconds.
//1-10
#define REMEMBER_LAST_WAYPOINT_MODE 0 // If set to 1 will remember the last waypoint even if you restart the autopilot. If 0, will start from zero every time you restart the system.
//1-11
#define INTPUT_VOLTAGE 5200.0 // voltage in millis your power regulator is feeding your ArduPilot. Measure and set this to have an accurate pressure and battery level readings. (you need a multimeter to measure this of course).
//1-12
#define REVERSE_THROTTLE 0 // 0 = Normal mode. 1 = Reverse mode...


// Fly by wire settings
//
//(Note: If you disconnect the GPS you will fly by wire.)
//ALWAYS leave your stick's centered when you ArduPilot is booting up.

//2-1
#define FLY_BY_WIRE_GAIN_ROLL .5 //Decrease the value to increase the response of the sticks. DESIRED_ROLL = //STICK_POSITION*FLY_BY_WIRE_GAIN_ROLL
//2-2
#define FLY_BY_WIRE_GAIN_PITCH .5 //The same as roll.
//2-3
#define FLY_BY_WIRE_SPEED_SETPOINT 20 //The airspeed you want to hold in fly by wire mode.
//2-4
#define GPS_ERROR_SPEED_SETPOINT 3 // In -m/s; , in case of GPS failure the airplane will enter into stabilization mode only and will try to maintain the airspeed set here.
//2-5
#define REV_FLY_BY_WIRE_CH1 1 //-1 will invert it
//2-6
#define REV_FLY_BY_WIRE_CH2 1 //-1 will invert it


//Autopilot PID gains.
//(Note: All the PID control loop gains and limits...)
//3-1
#define SERVO_AILE_MAX 2400 //Range of Ailerons
//3-2
#define SERVO_AILE_MIN 600
//3-3
#define SERVO_ELEV_MAX 2400 //Range of Elevator
//3-4
#define SERVO_ELEV_MIN 600

//HEADING GAINS
//4-1
#define head_P .7 //Heading error proportional (same used to move the rudder)... DO not add too much or you will oscillate left and right.  (drunk driver effect)
//4-2
#define head_I .1 //heading error integrator. Do not add too much or you will overshoot.
```

```
//4-3
#define head_D 0 //Derivative not used, but someday....
//4-4
#define head_error_max 35 //35 The maximum output in degrees to control the roll
setpoint
//4-5
#define head_error_min -35 //-35 The min output in degrees to control the roll setpoint

//ROLL GAINS
//5-1
#define roll_abs .2 //Set point absolute...(Not Used)
//5-3
#define roll_P .35 //roll PID proportional
//5-3
#define roll_I .35 //roll PID integrator
//5-4
#define roll_min -25 //PID output limit in servo degrees
//5-5
#define roll_max  25 //PID output limit in servo degrees
//5-6
#define roll_Integrator_max 10 //Limit the integrator, to avoid overshoots
//5-7
#define roll_Integrator_min -10

//PITCH GAINS
//6-1
#define pitch_P .65 //Pitch Proportional
//6-2
#define pitch_I .35 //Pitch integrator
//6-3
#define pitch_min -25 //Pitch limits
//6-4
#define pitch_max  25
//6-5
#define pitch_Integrator_max 10 //Pitch integrator limits
//6-6
#define pitch_Integrator_min -10
//6-7
#define PITCH_COMP .30 //<------Very important, Pitch compensation vs. Roll bank angle.

//THROTTLE GAINS
//7-1
#define throttle_max 1800 //Servo range In milliseconds.
//7-2
#define throttle_min 1200 //
//7-3
#define throttle_dead_zone 20 //In percent %
//7-4
#define throttle_absolute 3 //Absolute
//7-5
#define throttle_kp 3 //Proportional
//7-6
#define throttle_ki 1 //Integrator
//7-7
```

```
#define throttle_max 85 //Limits
//7-8
#define throttle_Integrator_max 70 //Integrator limit.


//More PID gains for altitude and speed.
//8-1
#define ALTITUDE_ERROR_MAX 10 //
//8-2
#define ALTITUDE_ERROR_MIN -10 //
//8-3
#define ALTITUDE_ERROR_PITCH_PROPORTIONAL 1.5 //Altitude error proportional, pitch setpoint
//8-4
#define ALTITUDE_ERROR_PITCH_MAX 15 //The limit of pitch travel (in degrees) used for altitude hold, if you add to much you may stall...
//8-5
#define ALTITUDE_ERROR_PITCH_MIN -15 //The limit of pitch travel (in degrees) used for altitude hold, if you add too much you may dive
//8-6
#define AIRSPEED_CENTRAL 30 //Airspeed  central point in m/s, normal flight... This value is the lowest airspeed that makes your plane flight steady.
//8-7
#define ALTITUDE_ERROR_AIRSPEED_PROPORTIONAL 3
//8-8
#define ALTITUDE_ERROR_AIRSPEED_MAX 30
//8-9
#define ALTITUDE_ERROR_AIRSPEED_MIN -10


  /*****************/
 /*Debugging Stuff*/
/*****************/

//9-1
#define FAKE_BEARING 0 //If set to 1, will fake the bearing and will try to always head to the defined DESIRED_FAKE_BEARING
//9-2
#define DESIRED_FAKE_BEARING 45 //Will try to go NorthEast, you can change that to 0 = NORTH, 90 = EAST, 180 = SOUTH, 270 = WEST or whatever!
//9-3
#define FAKE_GPS_LOCK 0 ////If  set to 1 will jump the GPS lock process to set home position. FOR TESTING ONLY!
//9-4
#define PRINT_WAYPOINTS 1 //If set to 1, at bootup will print all the waypoints set in the eeprom!
//9-5
#define TEST_THROTTLE 0 // If set to 1 will test the throttle increasing the speed slowly.
//9-6
#define WALK_AROUND 1 //Must be "0" to test the GPS and heading against the servo and "1" for normal operation
//9-7
#define CALIBRATE_SERVOS 0// Use to move the servos center, left and right or center right and left. You must adjust using 3-1 and 3-2.
```

```
//9-8
#define TEST_SENSORS 0 // Set 1 for enable, overwrite the servos with the raw IR sensors data, to test orientation. This will overwrite everything.
//9-9
#define PRINT_EEPROM 1 // If set to 1, sends content of the EEPROM to the serial port for debugging purpose
```

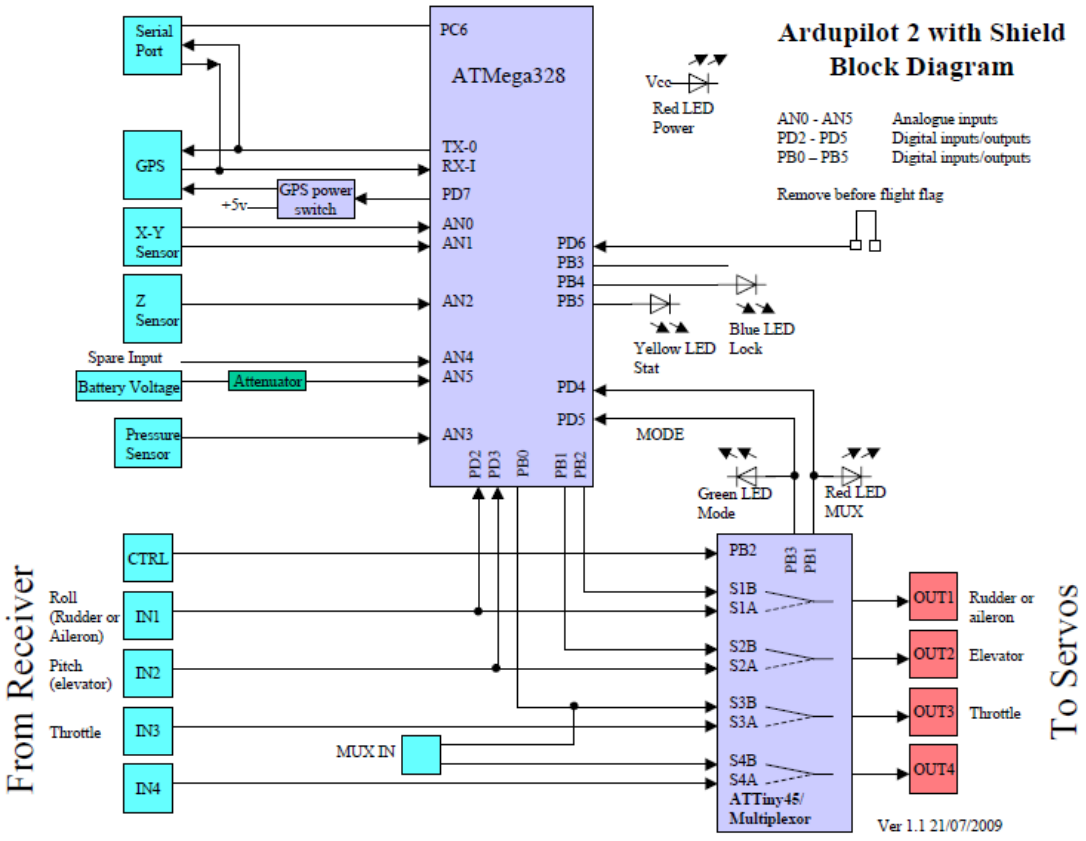
Appendix 3: Theory of operation

The ArduPilot theory of operation is essentially the same as that of the Paparazzi open source autopilot, which is nicely documented here.

If you follow that link, you will learn:

--How thermopiles work
--How PID loops work
--How cross-track waypoint following works
--etc

Appendix 4: Hardware diagram



(Thanks to Peter Seddon)

Appendix 5: Software architecture

(coming)

Appendix 6: Troubleshooting

Problem	Cause/Solution
Blue LED (GPS lock) does not come on (with jumper removed)	You do not have GPS lock. Reattach jumper and reset board. The GPS shoud flash until GPS lock is achieved. If it continues flashing for more than 15 minutes, you probably have set up ArduPilot for the wrong GPS module (see settings in Appendix 2) or have a connection problem with your GPS module (look closely at the pins in the GPS connector and see if one is bent). Once GPS lock is achieved you can remove the jumper and reset the board. The blue GPS lock LED should come on quickly.
Cannot upload code with Arduino IDE (IDE reports errors such as "avrdude: stk500_getsync(): not in sync: resp=0x78")	Please run through the debugging tips here
LED on EM406 GPS module is not on	This is normal when it is in SIRF binary mode (default for ArduPilot). If you have switched to NMEA mode after ArduPilot has previously programmed the EM406 into binary mode, you will have to wait about a week for its internal capacitor to expire, returning it to its default NMEA mode. Then the LED will come on again
You've added Lat and Lon coordinates to the configuration utility and pressed Write, but when you Read, you see zeros for some of them	You must hit "Enter" in each field after entering the coordinate.
Little or no manual control in "fly-by-wire" mode	Decrease the number in configuration settings 2-1 and 2-2. A good starting point for the gains is 8

Appendix 7: Glossary

2.4 Ghz: The frequency used by digital (spread spectrum) radio communications in our applications, including 2.4Ghz RC, bluetooth and some video transmission equipment. This is a different band than the older 72 Mhz band that is used for analog RC communications. To avoid radio frequency conflict is it often a good idea to use 72 Mhz radio equipment when you are using 2.4 Ghz onboard video transmitters, or use 900 Mhz video when using 2.4 Ghz RC equipment.

AHRS: Altitude Heading Reference System. An IMU (see below) plus the code to interpret the output from its sensors to establish a plane's XYZ and heading orientation.

AGL: Altitude above ground level

AMA: Academy of Model Aeronautics**.** The main US model aircraft association. Generally hostile to amateur UAVs, which are banned on AMA fields. But each AMA chapter and field may have slightly different policies, and it's possible to test airframes and some technology on AMA fields without violating the association's rules.

Arduino: An open source embedded processor project. Includes a hardware standard currently based on the Atmel Atmega168 microprocessor and necessary supporting hardware, and a software programming environment based on the C-like Processing language. Official site is here.

BASIC Stamp: A simple embedded processor and programming environment created and sold by Parallax. Often used to teach basic embedded computing and the basis of our autopilot tutorial project. Parallax also makes the very capable Propeller chip, which is the basis of the AttoPilot autopilot and others.

Bootloader: Special code stored in non-volatile memory in a microprocessor that can interface with a PC to download a user's program.

COA: Certificate of Authorization. A FAA approval for a UAV flight. See this for more.

DCM: Direction Cosine Matrix. A algorithm that is a less processing intensive equivalent of the Kalman Filter. See this for more.

Eagle file: The schematic and PCB design files (and related files that tell PCB fabs how to create the boards) generated by the free Cadsoft Eagle program. This is the most common standard used in the open source hardware world, although, ironically, it's not open source software itself. Needless to say, this is not optimal, and the Eagle software is clumsy and hard to learn. One hopes that an open source alternative will someday emerge.

ESC: Electronic Speed Control. Device to control the motor in an electric aircraft. Serves as the connection between the main battery and the RC receiver. Usually includes a **BEC**, or Battery Elimination Circuit, which provides power for the RC system and other onboard electronics, such as an autopilot.

FPV: First-person view. A technique that uses an onboard video camera and wireless connection to the ground allow a pilot on the ground with video goggles to fly with a cockpit

view.

FTDI: A standard to convert USB to serial communications. Available as a chip for boards that have a USB connector, or in a cable to connected to breakout pins. FTDI stands for Future Technology Devices International, which is the name of the company that makes the chips.

GCS: Ground Control Station. Software running on a computer on the ground that receives telemetry information from an airborne UAV and displays its progress and status, often including video and other sensor data. Can also be used to transmit in-flight commands to the UAV.

Hardware-in-the-loop simulation: Doing a simulation where software running on another computer generates data that simulates the data that would be coming from an autopilot's sensors. The autopilot is running and doesn't "know" that the data is simulated, so it responds just as it would to real sensor data. Hardware-in-the-loop uses the physical autopilot hardware connected to a simulator, as opposed to simulating the autopilot in software, too.

I2C: A serial bus that allows multiple low speed peripherals, such as sensors, to be connected to a microprocessor. See this for more.

IDE: An integrated development environment, such as the Arduino editor/downloader/serial monitor software. Often includes a debugger.

IMU: An inertial measurement unit. Usually has at least three accelerometers (measuring the gravity vector in the x,y and z dimensions) and two gyros (measuring rotation around the tilt and pitch axis). Neither are sufficient by themselves, since accelerometers are thrown off by movement (ie, they are "noisy" over short periods of time), while gyros drift over time. The data from both types of sensors must be combined in software to determine true aircraft attitude and movement to create an AHRS (see above). One technique for doing this is the Kalman filter (see below).

Inner loop/Outer loop: Usually used to refer to the stabilization and navigation functions of an autopilot. The stabilization function must run in real-time and as often as 100 times a second ("inner loop"), while the navigation function can run as infrequently as once per second and can tolerate delays and interruptions ("outer loop").

INS: Inertial Navigation System. A way to calculate position based on an initial GPS reading followed by readings from motion and speed sensors using dead reckoning. Useful when GPS is not available or has temporarily lost its signal.

ICSP: In Circuit Serial Progammer. A way to load code to a microprocessor. Usually seen as a six-pin (two rows of three) connector on a PCB. To use this, you need a programmer, such as this one, that uses the **SPI** (Serial Peripheral Interface) standard.

Kalman Filter: A relatively complicated algorithm that, in our applications, is primarily used to combine accelerometer and gyro data to provide an accurate description of aircraft attitude and movement in real time. See this for more.

LOS: Line of Sight. See VLOS below.

LiPo: Lithium Polymer battery, aka LiPoly. Varients include Lithium Ion (Li-Ion) battery. This battery chemistry offers more power and lighter weight than NiMh and NiCad batteries.

NMEA: National Marine Electronics Association standard for GPS information. When we refer to "NMEA sentences", we're talking about ASCII strings from a GPS module that look like this: $GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

OSD: On-screen display. A way to integrate data (often telemetry information) into the real-time video stream the aircraft is sending to the ground.

PCB: Printed circuit board. In our use, a specialized board designed and "fabbed" for a dedicated purpose, as opposed to a breadboard or prototype board, which can be used and resused for many projects.

PIC: Pilot in Command. Refers to a FAA requirement that UAVs stay under a pilot's direct control if they are flying under the recreational exemption to COA approval. See Line of Sight above. (Not to be confused with the PIC processor series by Microchip)

PID: Proportional/Integral/Derviative control method. A machine control algorithm that allows for more accurate sensor-motion control loops and less overcontrol. See this for more.

SiRF III: The standard used by most modern GPS modules. Includes SiRF III binary mode, which is an alternative to the ASCII-based NMEA standard described above.

Sketch: The program files, drivers and other code generated by the Arduinio IDE for a single project.

Thermopile: An infrared detector. Often used in pairs in UAVs to measure tilt and pitch by looking at differences in the infared signature of the horizon fore and aft and on both sides. This is based on the fact that there is always an infrared gradient between earth and sky, and that you can keep a plane flying level by ensuring that the readings are the same from both sensors in each pair, each looking in opposite directions.

UAV: Unmanned Aerial Vehicle. In the military, these are increasingly called Unmanned Aerial Systems (**UAS**), to reflect that the aircraft is just part of a complex system in the air and on the ground. Ground-based autonomous robots are called Unmanned Ground Vehicles (**UGVs**) and robot submersibles are called Autonomous Underwater Vehicles (**AUVs**). Robot boats are called Unmanned Surface Vehicles (**USVs**).

VLOS: Visual Line of Sight. The pilot's ability to see an aircraft from the ground well enough to control it, without the use of artificial visual aids (aside from glasses). Required by FAA regs.

WAAS: Wide Area Augmentation System. A system of satellites and ground stations that provide GPS signal corrections, giving up to five times better position accuracy than uncorrected GPS. See this for more.

Xbee: The commercial name of the recommend ZigBee-compatible radio modems commonly used by amateur UAVs

ZigBee: A wireless communications standard, which has longer range than bluetooth but lower power consumption than WiFi.