ME 100 (UAV) - Final Report

By: Connor Lee, Parth Shah, Jean Turban, Chris Bradley, Nick Buoniconti

Table of Contents

Introduction

The Caltech Unmanned Aerial Vehicle (UAV) Engineering Club was founded September 2014. The first two quarters of the 2014-2015 school year were used to raise money and find interested members of the student body. The mission of the club is to foster interest in UAVs, something that has garnered lots of attention in industry these past couple years. The applications of manned and unmanned vehicles have increased exponentially and having Caltech students involved in pertinent project is the goal of the club. As of June 2015 there were 23 members of the UAV Club. Of these 23 members, 8 of the members were considered active and 5 of them participated in the ME 100 class during the Spring 2015 quarter. These five students were: Chris Bradley (ME ’17), Nicholas Buoniconti (ME ’17), Connor Lee (CS ‘ 17), Parth Shah (ME ’17), and Jean-Alexandre Turban (ME/CS ‘ 16).

Traditionally the choice for an aircraft is between a copter and a plane. When it comes to attempting to fly one autonomously, the logical choice seems to be copters. The simplest and most common copter is the quadcopter. It utilizes a straightforward design and is highly maneuverable while still providing a decent amount of thrust to carry necessary attachments. For this ME 100 project we used the 3DR DIY Quadcopter Kit. This was a kit that was highly customizable if the user desired but also could be constructed into a basic x-mode quadcopter. It is also important to note that quadcopters are widely available and therefore when problems arise it is likely not for the first time. This made forums a very useful place to go and read up on similar hurdles other copter enthusiasts had and how they overcame them.

The initial goal of the ME 100 class was “To build a drone that can follow an RC toy car on the ground autonomously. The drone will perform in an open environment (so not dealing with object collision) and follow an object of a specific color to make it easy to distinguish” (ME 100 Syllabus). The expectation at this point was to be able to build the quadcopter in a couple weeks at the beginning and then allow the software team to work for the rest of the term on developing successful code and algorithms to find the RC car while the hardware team built a second improved generation of the quadcopter. As the quarter progressed and problems came up the goal was redefined on 6/10/15 to be “New Goal: Have quadcopter takeoff autonomously (3 ft off ground). Once stable and hovering, roll tennis ball underneath quadcopter. Quadcopter camera should recognize object and should trigger autonomous landing sequence” (ME 100 Syllabus). At this time there have been discussion of what the second generation of the quadcopter should look like and include but due to a lack of time no substantial progress was made on the next iteration.

Hardware

To complete the task at hand the plan was to build a quadcopter and mount a computer and a camera on it. After doing some research and reading forums of individuals who also attempted to make quadcopters autonomous, the basic on-board computer of choice seemed to be the Raspberry Pi. The camera of choice normally tended to be a GoPro, but because the Raspberry Pi already came with a simple USB camera, that was the one used over the course of this quarter. The quadcopter also had a compass with GPS and 915 MHz telemetry mounted on it.

As mentioned above for the ME 100 class a quadcopter was chosen to be the choice of aircraft for the desired goals. The 3DR DIY Quadcopter Kit was selected as a result of the flexibility present in the kit. With a couple more arms and motors it can be changed from a quadcopter to an octocopter very easily. For this first generation UAV, an x-mode quadcopter was desired and constructed. The kit also came with a Pixhawk controller. From online reviews it seemed as if that the Pixhawk was a top of the line controller with the other option being the APM 2.6 controller. Because the Pixhawk was already included with this quadcopter kit, it was the one used for this project.

Quad Hardware Components

In this segment we will look at the various components selected for the quad, analyze why they were selected and look at how to work through set up. This section in conjunction with the receiver/transmitter and mission planner segments should be followed to gain understanding of how quadcopter was constructed and linked with our control board.

Frame Selection

We selected an X-wing quad copter. We choose this setup as we thought it provided a lot of functionality while remaining easy enough to program for newcomers. The X-wing quadcopter has 4 motors, and they are setup in an X shape with two arms extending from the body of the frame to the motors on each side. The frame had a lot of useful features such as the hollow arms that enabled us to keep the motor wires on the inside of the quadcopter. A drawback to the frame was its durability and inaccessibility to the electronics, which we will discuss later on. The frame was made of a fairly lightweight material, but the design flaws in the frame held the drone back in certain areas. A goal of future research and development for the club is to design a more efficient and effective base frame. Read the future work portion of the document to get a feel for the direction we are hoping to go with the frame design.

Power Distribution Board (PDB)

The first key component to our quad setup was the power distribution board (PDB). This board relays how much power will go to each of our four electronic speed control motors (ESC). Depending on how many motors you want to use, the PDB can be switched out or modified to accommodate different motor configurations. The PDB works in line with the Pixhawk Flight Controller (Pixhawk) to control the different spin rates of the motors. It looks like this:

The Pixhawk will make all of our calculations and process what speeds the motors should spin at to achieve the correct flight path. We control the Pixhawk through receiver or autonomous controls and our inputs are taken in by the Pixhawk and used in calculations of spin rates. The Pixhawk will relay that information to the control board through wire connections. It is essential that these wires are connected to the correct input slots on the PDB. We had some challenges with analyzing the motor configurations as online tutorials often had inaccurate wire setups. It is essential to hook up the wires that go from the speed controller to the PDB to the correct slots. Power will come in from another wire from the battery that goes to the distribution board and it will split this power amongst the four motors and send it to them through additional cables. Pin connections for the ESCs are also made in the back of the Pixhawk; correctly hooking all of these connections up is essential for proper power distribution, spin rate, and ultimately flight path. Some pitfalls we ran into in this section was incorrect programming of the Pixhawk and incorrect wiring of the PDB based on faulty image walkthroughs on the 3DR website. Think through these stages and make sure the way your board is wired makes sense for the functions you want it to serve. We found that the easiest way to wire it correctly was to label your motors according to Figure 1, and to simply connect the motors to the matching number on the board.

Figure 1: How to number your quadcopter motors for easy connection to PDB. This assumes you are using an "X" configuration.

Pixhawk

The Pixhawk as discussed briefly before was the chosen control board for our drone. We decided upon the Pixhawk because of its flexibility in design features and its ability to work through the Mission Planner flight software. Together you can utilize the Pixhawk and Mission Planner setup to do almost anything you want to on the drone. The Pixhawk has easy connection locations to the basics such as the GPS hookup, telemetry setup, and ESC connections, but also has some spots to plug in additional items. You can utilize this board to plug in other external hardware items such as sonar and, in our case, a raspberry pi onboard operating system. The raspberry Pi was utilized for autonomous flight and computer vision (CV) recognition, but please read the computer portion of the document for an in depth look at that part of the project.

There are many unique features the Pixhawk can be programmed to employ. Because of the easy pin connections, external hardware can be plugged in fairly easily to the board which makes this board's customizability the best of any on the market. Setting up the Pixhawk was fairly straightforward. The firmware could be downloaded through the Mission Planner software. The calibration of the Pixhawk was completed in the steps that we outlined thoroughly in the Mission Planner module. The most challenging part of the general Pixhawk setup is ensuring that all of the pins are connected to the correct spots on the board. As discussed in the PDB segment, incorrect wiring of the pin connections to the PDB can lead to incorrect power distributions and may lead to awkward flight paths. In our case the board was wired 90 degrees off (1 connection to the right, based mistakenly on the online tutorial) which led to the onboard gyros in the Pixhawk trying to correct flight path by pushing the copter further off course. This led to tipping and eventually crashes. Utilize a tether system to keep the quad attached to the ground until you are certain your pin corrections are all correct, otherwise you may face some rough crashes in the testing phase.

Another cool feature about the Pixhawk, as discussed just above, is that the Pixhawk has self-correcting gyros on board. These gyros function by analyzing the current state of evenness of the quads axes and putting more or less power into the spin rates of the ESCs depending on the desired corrections. By spinning the motors faster one side will rise higher than another and this leads to turning, rotation, and change in height. The gyros basically ensure that no matter what happens in the take-off phase, once the quad is in the air, it will level out (assuming your connections are correct). You can help the accuracy of the gyros by calibrating correctly in the Mission Planner stages and by utilizing foam pads to limit the amount of vibrations the Pixhawk receives from the frame.

Global Positioning System (GPS)

The GPS we used was a GPS and compass in one. The GPS is a critical piece of the Drone and requires a lot of attention to ensure it works correctly. When working inside of a classroom we would often fail to get a GPS lock or our GPS would give us a location of somewhere in Africa (we were in Pasadena). Confounded by these problems we tackled the issues with the PDB wiring first and came back to our struggles with the GPS later. It turns out that if the GPS is indoors or anywhere near electronics it will likely fail to get a GPS lock. When testing the GPS it is best to go outdoors and bring only the laptop and cord to connect to your laptop, otherwise you could face interference. One useful components to the drone frame we purchased was that it came with a GPS mast. This mast was very useful in the project as it separates the GPS from the rest of the electronic components below. Simply attaching the GPS to the frame will not do as this additional 3 inches of spacing is pivotal. Once outdoors we were able get a GPS lock and design flight paths in Mission Planner, which the drone correctly followed. A blinking green light means that the GPS is locked and that you simply need to arm the drone, which we will get to later. You can check the status of the GPS lock by connecting to Mission Planner and looking in the top left status bar. If it says “no GPS fix” then your Pixhawk is not setup right yet or your need to get outdoors. If it says “no 3D fix” then you need to get away from electronics or get outdoors because the GPS is working it just can’t get a fix because there is too much interference from these external factors.

Electronic Speed Controlled Motors (ESC)

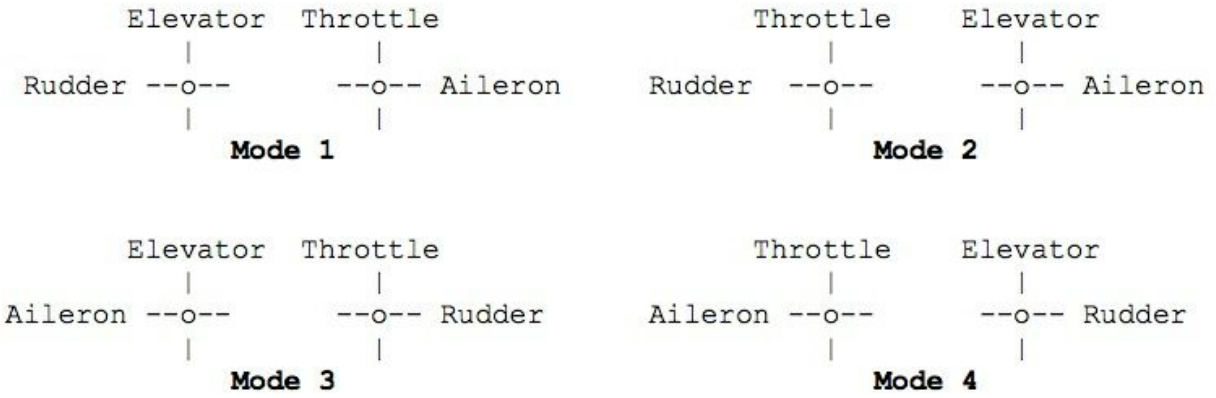
These were the motors selected for the quad setup. As touched upon briefly in the PDB segment, these motors work by receiving amounts of power from the PDB and spinning at a given rate. These motors were selected because of the ease and feasibility of utilizing battery power as opposed to gas. Gas powered motors would have been difficult to manage, dangerous, and far too costly for our setup. The far cleaner Lipo battery setup is a much safer and sensible power method. These motors are basically attached to the frame via screws and then to propellers through a nut and screw setup. Removing the propellers for testing and other purposes is quite easy and is one reason why we selected this motor setup. In order to start flying the Pixhawk you must complete ESC calibration. This is the last step in the calibration process. Please look at the modules for Mission Planner setup and Radio Controller setup before attempting the ESC calibration. Calibrating the ESCs consists of plugging the battery in with the thrust joystick in its max position. Then bring the joystick to the bottom position and the drone will make a couple of short beeps indicating the calibration is complete. Once this step is complete the drone must be armed and then it can fly. Note that you only need to calibrate the ESC’s once, although it is fairly easy so you may repeat it as often as you like.

Arming the ESCs is fairly simple as you just push the thrust joystick to its right most bottom most position. Then drone should beep and the flashing light should turn either solid blue or solid

green. Solid blue indicates that you have armed successfully but do not have a GPS lock. A solid green light indicates that you have successfully armed and that you have the GPS lock. Any other light configuration indicates that you failed the arming or ESC calibration process. To troubleshoot this problem plug the Pixhawk into the Mission Planner and use the status window to analyze the issues.

Quad Receiver and Transmitter

Now we will explore the transmitter and receiver for the quadcopter. This is a key component and is necessary even for autonomous flight. Our team used a Turnigy 9x Transmitter with a FrSky DJT module that communicated with a FrSky V8FR-II receiver. Our setup also included a PPM-Sum encoder, which we will explore in depth later. The transmitter we purchased was “Mode 2”, which indicates that the throttle channel is on the left joystick. Different modes (shown below) can be purchased based on preference.



Choosing a Receiver and Transmitter:

When browsing the Ardupilot FAQ there are three mainly suggested transmitter (Tx) and receiver (Rx) combinations for use with the Pixhawk. We personally ended up using a Turnigy 9x transmitter with a FrSky DJT module that communicated with a FrSky V8FR-II receiver. The transmitter and receiver were chosen due to availability, price, and versatility. The Turnigy 9x gives us access to 8 RC channels and is heavily documented in online forums. Other suggested models like the FrSky Taranis are twice the price of the Turnigy 9x and less readily available. The Taranis does come with more options (such as 16 channels) and is easier to set up, however these qualities did not justify the price when keeping our task in mind. One important thing to keep in mind when choosing a receiver and transmitter is whether they have built in PPM-Sum (or cPPM) capabilities. This aspect is covered extensively in the following sections and is critical to flying with a Pixhawk controller. Lastly, when choosing your receiver and transmitter, it is worth considering whether you want telemetry capabilities. Telemetry setup is not necessary for manual or autonomous flight but is useful in logging flight data. We opted to have hardware that could handle telemetry in case we decided to try it.

Modding the Turnigy 9x to Work With FrSky

The FrSky module was used primarily because it is compatible with PPM and the Turnigy 9x stock module does not support native PPM. In addition, the module had the added benefit of giving us telemetry capabilities and increasing our manual flying range to 2 kilometers. Furthermore, the module is compatible with a wide variety of FrSky PPM-Sum receivers, which made our system as a whole more versatile.

In order to use the FrSky transmitter module and receiver with the Turnigy 9x however, some physical modifications must be made to the transmitter. The standard Turnigy 9x comes with its own built in transmitter module (Figure 2) and must be replaced by the FrSky module.

Figure 2: The Turnigy 9x with the stock transmitter module can be seen on the left. The updated 9x with the FrSky module is seen on the right.

In order to do this we began by taking the screws off the stock module and removing it from the cartridge. We found that the module was connected to the antenna through a thin cable that ran through the inside of the 9x. Inside the module there is a piece of PCB with the cable connections (Figure 3) - you must unsolder these in order to free the module and antenna.

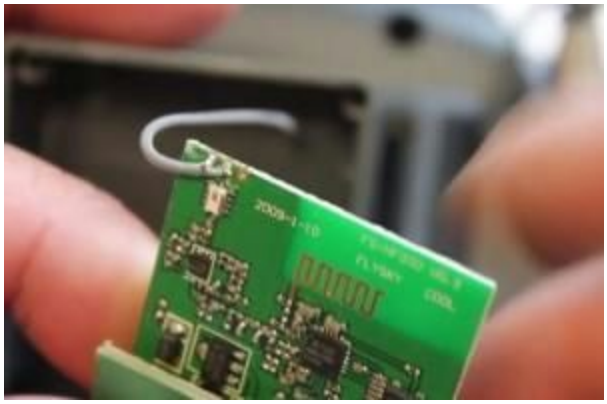


Figure 3: The PCB board housed inside the stock transmitter module. The thin wire connection on the upper left runs through the 9x and to the antenna. This must be unsoldered in order to remove the transmitter module and antenna.

In order to fit the FrSky module in the 9x we also had to remove a plastic lip (Figure 3) that prevents the FrSky module from fitting in the cartridge properly. Thus we proceeded to open up the Turnigy 9x. From the inside, we pulled back the PCB board that houses the connector pins, thus leaving the plastic lip by itself. It is important to keep the pins straight and undamaged, as otherwise the transmitter module will not fit.. With the pins clear of the plastic lip, we cut off the lip (Figure 4).

Figure 4: The cartridge of the 9x. On the left we can see the connector pins being housed by the plastic lip. The picture on the right is after removing the pins and clipping the lip. (Note: this picture as well as the PCB picture was taken from http://blog.oscarliang.net)

We then reassembled the transmitter, taking care to put everything back the way it was. For more detailed instructions, we suggestion looking here:
https://pixhawk.org/peripherals/radio-control/turnigy_9x_frsky_mod

Adding the PPM Sum Encoder

The Pixhawk is not capable of using PPM (Pulse Position Modulation) or PWM (Pulse Width Modulation) signal which is standard in most transmitters and receivers and instead takes in a PPM-sum signal. While PPM requires each signal to have its own channel, PPM-Sum has the benefit of combining multiple signals per channel. If the hardware is PPM-sum compatible out of the box, then your Rx and Tx will take care of unscrambling the PPM-Sum so that your system works as intended. It is important to note that both Rx and Tx must both be PPM compatible, and that the receiver must be PPM-Sum compatible. We ran into issues and delays when we failed to realize our V8FR receiver did not have built in PPM-Sum capabilities. Luckily, this was an easy fix as one can simply purchase a “PPM Encoder”to use with the receiver, which goes ahead and converts the PPM signal into PPM-Sum before it passes it on to the Pixhawk. We purchased our encoder directly from 3DR robotics and simply followed their instructions to install it. If we were to choose a new receiver, we would likely opt for the FrSky D4R-II : a lightweight receiver that has built in telemetry and PPM-Sum capabilities (rendering the PPM encoder useless and making our quadcopter cleaner).

Correct Settings for Quadcopter Flight

Before we could fly our quadcopter, we had to update the settings on our transmitter so it knew what sort of vehicle we were flying. We simply followed YouTube video tutorials in order to achieve this but the gist of it is as follows:

Under System Settings:

- Create new mode by clicking Mode Select and selecting the second entry.
- Rename the mode by clicking Mode Name and renaming it to “Quad”
- Set mode type as “Acro” (generally used for planes)
- Set Stick as “Mode 2”(assuming your transmitter is also Mode 2)

Under Func. Settings

- Go to “Aux-Ch”(auxillary channel) and set Ch5 to “Gear”

In order to make sure these settings take effect, make sure to copy over your updated settings into the primary slot. The Turnigy 9x has a built in setting that allows you to easily change your primary settings to any user-defined profile. Occasionally, the Turnigy 9x does not save the settings so make sure that all your changes take effect by going back and double checking your selections

The Turnigy 9x under Acro mode allows for up to 6 modes which can be toggled through the various switches on the transmitter. For our purposes, we simply used two flight modes: Stabilize and Auto. These were toggled by flipping Ch.5 on/off and allowed us to transition from flying the quadcopter manually to autonomously by simply flipping the switch. We made sure to

check that our two modes and our five channels worked properly by connecting to Mission Planner and going to the respective “RC” and “Flight Modes” tabs (Figure 4.)

Figure 5: Using Mission Planner, one can ensure that all the flight modes and channels are working properly. Mission Planner shows changes on the RC transmitter in real time which allows you to toggle channels to see the response.

While in Mission Planner, if you note that one of the channel is reversed, then simply change the channel settings within the transmitter. Under Func. Settings there is an option for reversing channels

Issues and Fixes

With a project as ambitious as ours, we were bound to run into a few problems with the implementation of our plan. While it can be, and often was, frustrating to troubleshoot such a

complex and multifaceted machine as the quadcopter we built for this course, it is important to remain engaged and persevere through the setbacks. This section will describe some of the mechanical issues we ran into as an example of either how to overcome them or how we might improve our design for the second generation.

Assembly

In the process of figuring out the wiring of the power distribution board and the pixhawk, we needed access to all the quadcopter's electronics. Unfortunately, the 3DR design stored those components within the two main carbon fiber plates of the body, locked in by dozens of screws. Each time we wanted to make a change to the wiring, we had to disassemble the entire quadcopter, which cost us valuable time and effort. There was no real way for us to solve this issue, but we can plan to avoid it in the future. Our next design will keep this frustration in mind and make sure the electronics are easy to access when making adjustments. We must still keep in mind the durability of the design, as our system must be able to survive the inevitable crash. That being said, it should not be difficult to find a happy medium.

Durability

While the electrical system was able to sustain a significant beating without failing, the same cannot be said about some of the physical aspects of the quad copter.

Legs

The four supports of the quad copter underwent significant stress during the test flight stage of development, where we flew the copter while making adjustments to the motor calibration. Needless to say, there were quite a few crashes with some pretty fast speeds. In one such instance, the quadcopter was brought down hard on concrete to avoid nearby pedestrians. In the course of the landing, several screws in the legs popped out and were lost in the brush, forcing us to rethink how we support the quad copter. In the end, we decided to reinforce the existing structure with duct tape, wrapping it several times around each leg. In all subsequent tests after adding the duct tape, the legs have been sturdier than ever, allowing us to test the limits of the quadcopter without fear of breaking it. Our future design will likely feature shorter legs, so they are less likely to break in the first place. The original legs consisted of two carbon fiber angled "L" pieces that connected at various points through screws and plastic spacers. These legs are not very sturdy and have a lot of flex. They also tend to lose screws if you come down to hard. We suggest using legs that have a sturdy shaft, and a rubbery/bouncy tip.

GPS

Much like the legs of the quadcopter, the GPS system also took a beating when we tested with human pilots. The transmitter/receiver is mounted several inches above the top surface of the quad copter, supported only by a thin, hollow plastic rod. This left the GPS very exposed, and

broken almost immediately. As a quick fix, we repeatedly glued the rod back to its mount with mixed results. It seemed like every other time we flew, the GPS would be dangling off the quad copter by the time it landed. Finally, we realized we should look back to duct tape as a possible solution, and ended up taping the rod several times over. However, this is clearly not a long term solution. It will be tough to fix this problem in our future design however, as the GPS must be a certain height above the quadcopter so the electronics don't interfere with the signal. To deal with the issue, we might consider a sturdier mount that doesn't break as easily, perhaps made of aluminum, or magnesium. Then again, the fact that the GPS snaps at the rod helps ensure that the GPS module itself is not damaged, so we'll see.

One thing about our group that allowed us to succeed and meet our goals was the fact that we were able to think outside the box when it came to solving the problems we ran into. Moreover, the experience we gained in dealing with this iteration of the quadcopter will enable us to build an even better prototype for the next round.

Software

Mission Planner Usage and Issues

The Pixhawk selection led us down the natural path of choosing Mission planner as our programming platform as these two are intertwined. Pixhawk works well with Mission planner and there are unique features in Mission Planner's interface that were extremely useful. Here is a quick outline and guide that will take you through the benefits of using Mission Planner, outline some pitfalls we encountered, and indicate the steps needed to get drone flights operational.

Calibration

For this project we had an x wing quad. In mission planner there are a lot of preset frames that you can choose from. Our frame orientation of the X-wing quad was preset in the system and is simply selected from a display menu. One unique feature of Mission planner is its ability to support custom frames. In our future plans we will be attempting to build some non-traditional drone frames and thus this feature will be essential for us and can be an extremely helpful tool to keep in mind. Ability to be creative with the drone was pivotal to many of our hardware selections, and you will see that theme throughout this Mission Planner walkthrough.

Before getting too far into the calibration process you must download the proper firmware onto the Drone. If the firmware does not download check the USB connection and make sure the Pixhawk is receiving power from the laptop. The battery does not need to be plugged in for any parts of this process, as the Pixhawk should be receiving power from the cable that connects it to the computer. By simply entering the setup wizard and choosing the proper frame, you will be prompted with the firmware update process. Once everything is good on the board, the Pixhawk will notify you that the firmware is up to date with a flurry tone and with a message on the laptop.

The next piece of Calibration was setting up the orientations of everything. This entails a two-step process. The first stage of this is setting up the drone's x-y-z coordinate plane by resting the drone on each axis of rotation. This can be done by following the visual guide in Mission Planner's set-up wizard. Additionally a nice feature of Mission Planner is that you can re-calibrate any part of the drone's calibration without having to go through the entire calibration process. The second stage of this orientation set-up is to rotate the quad around each axis of rotation. This can again be done in the setup wizard. You must hit every portion of the 3D sphere on screen or you may be stuck in this phase. Keep rotating around different axes of rotation until enough of the sphere is filled in otherwise the calibration may be inaccurate. There are helpful YouTube videos online for this stage if your technique is poor.

Once this portion has been calibrated, you can move on to programming your radio controls. This can only be done once your receiver has been properly bound. Please see the binding process section of the document if you run into issues with binding the controller to the Pixhawk as this is a delicate process. Additionally make sure you either have the Lipo battery disconnected or the propellers disconnected before programming RC to ensure no accidental flight occurs. Once everything is safe and ready for RC testing utilize either the wizard tool or go to RC calibration and click calibrate. Now you must move the sticks and switches you are calibrating to their extreme conditions (both highest and lowest positions). The mission planner should respond by moving the bounds of the different channels in response to your stick and switch movements. If this does not happen you most likely have an issue with your controller setup or your binding setup (again look at the binding setup if you get lost here). Once you have finished this setup you can move on to the customization portions of the calibration process. The earlier steps must be done on any quad, tri, etc. that you want to fly using the mission planner interface. Failure to do these steps will likely result in a failure to arm response tone and light which will be either a green or blue slow blink (depending on whether or not you have a geo-lock). Now we can get into the customization steps which is the most useful and open part of the software. There are many different flight modes you can program from hover to land to anything in between. You can set up these flight modes by selecting one of the modes from the available menu that mission planner provides or by programming your own. Some useful modes to have for our project were a land mode as we are inexperienced flyers and the mode that switches the vehicle from autonomous to human controlled flight (in case we wanted to step in). Other useful items to play with in the software include setting up a geo-fence, designating a battery fail-safe, and designing a sonar plan. Basically anything you could think of using with your drone can be done with some research and simple programming on Mission Planner. We looked in to utilizing the geo-fence as a way of keeping the drone in a set area. As long as the drone has a GPS lock you should be able to utilize this geo-fence. Another useful feature is the battery failsafe which will cut off the drone's battery power or initiate a land sequence if the battery is critically low. This is obviously important for occasions in which you are doing extensive testing with the drone. The point of this section is to basically let you know what Mission Planner has to offer and to show you the ways you can get creative with your drone and get the most out of the available technology. I urge you to explore all of these customizable

features before even designing your drone, as there may be certain items you want to incorporate after viewing the interface.

A very important piece of the Pixhawk flight is its utilization of the GPS. Any good drone must have a functioning GPS on board as the GPS can be utilized for numerous features from navigation to protecting the drone from hitting buildings or entering restricted air zones. We encountered numerous issues with the GPS as there are a lot of factors that can influence the GPS's ability to function. Please see the GPS part analysis in the Quad components segment for more details. We were able to utilize the GPS feature to get our drone to take off, fly selected paths at selected heights and land completely autonomously in this project.

Image Processing

Raspberry Pi 2

While the pixhawk could control and direct the movement of a drone, it could not be used for computer vision. Instead, a microcomputer was used as an onboard image processor. Specifically, the Raspberry Pi 2 was chosen for this role.

The core components of the Raspberry Pi 2 are the 900MHz quad-core ARM Cortex-A7 CPU, and its 1GB RAM. In addition, it has 4 USB ports, 40 GPIO pins, full HDMI port, ethernet port, camera interface (CSI), and a Micro SD card slot. The cost per unit is around $40, making it a relatively cheap compared to other competing models. The primary reason for using the RPI2 is the cheap cost per unit while having enough processing power and ram to perform intensive computations and analysis. The RPI2 is very light, allowing it to be mounted to the drone without hindering the drone's flight dynamics. In addition, the RPI2 can use a variety of UNIX operating systems. A replaceable Micro SD card allows the user to increase the memory of the computer.

In this project, we increased the storage of the RPI2 to 32GB and used the basic Raspbian Wheezy OS. Raspbian is a UNIX based platform and is very similar to Ubuntu in terms of package managing and other installations. To install Raspbian, the Raspbian image must be downloaded and written to a Micro SD card from a computer. Recommended size for the SD card is at least 4GB. For a project of this nature, our recommendation is at least 16GB (OpenCV and other libraries require a lot of space). Although the RPI2 has many perks, it also had numerous problems. Using a standard android phone charger (5V, 1000mA) causes the battery to heat up and fry the board. While others have had success with using a standard android phone charger, we recommend getting 5V charger with at least an output current of 2000mA. Another reason that our RPI2 might have had malfunctions is the static charges that it comes into contact with while being carried from place to place. In order to solve this issue, be sure to carry it in a static-free bag or a case.

A better alternative to the RPI2 is the ODroid. It has a better CPU and has much more RAM allowing for faster processing. The cost is around $70 - $160, which is more expensive than the RPI2.

Standard Raspberry Pi Camera

The Raspberry Pi camera is a 5 megapixel camera that weighs only 3 grams. It costs $25 and connects directly to the RPI2 camera port. It has many other features, which are listed here: https://www.raspberrypi.org/documentation/hardware/camera.md.

This camera module was chosen for our project mainly because it could easily integrate with the RPI2 and because of its light weight. Although the documentation and reviews states that the camera takes pictures with good resolution, the number of frames to achieve relatively good resolution is ~100-200. Subsequently, it takes around 5-7 seconds to generate a quality picture. Of course, an autonomous drone must "see" faster than that in order to effectively detect objects. In this project, each image consisted of 15 frames, which were enough to roughly determine a colored object's location.

OpenCV

OpenCV in C++ was used for all image processing tasks. The most time consuming part of making the drone autonomous was installing and integrating the libraries correctly. OpenCV takes 4-6 hours to build on the RPI2. Note that OpenCV must be installed first before Raspicam. CMake also needs to be installed before OpenCV.

To install: (Details here: http://robertcastle.com/2014/02/installing-opencv-on-a-raspberry-pi/)

1. Ensure the RPI2 is up to date.
    sudo apt-get update
    sudo apt-get upgrade
2. Download OpenCV online from GitHub or a mirror site.
3. Unzip the folder.
4. Change directory into the unzipped folder, make a folder called "release" and go into the newly created directory.
    cd opencv-x.x.x
    mkdir release
    cd release
    ccmake ../
5. Build and install OpenCV
    make
    sudo make install

Raspicam

Raspicam is a C++ API that allows a C++ program to directly take pictures with the Raspberry Pi camera module. Since we used OpenCV in C++ to process images, generating images with C++ code streamlines the vision into a single platform process.

To install: (Detailed instructions here: https://github.com/cedricve/raspicam)

1. Clone the repository.
   git clone https://github.com/cedricve/raspicam .
2. Build it.
   cd raspicam
   mkdir build
   cd build
   cmake ..

Drone API

The Drone API is a library that allows the RPI2 to command the pixhawk (subsequently, commands the drone) using Python scripts. MAVLink must be installed and open for the Drone API to properly work.

To install: (Detailed instructions here: http://dev.ardupilot.com/wiki/droneapi-tutorial/)

1. Get dependencies.
   sudo apt-get install pip python-numpy python-opencv python-serial python-pyparsing python-wxgtk2.8
2. Install Drone API.
   sudo pip install droneapi

MAVLink

MAVLink serves as the bridge between the RPI2 and the pixhawk. Essentially, it allows the RPI2 to send commands to the pixhawk. Along with installing MAVLink on the RPI2, the pixhawk and the RPI2 needs to be connected. This is done as shown below.

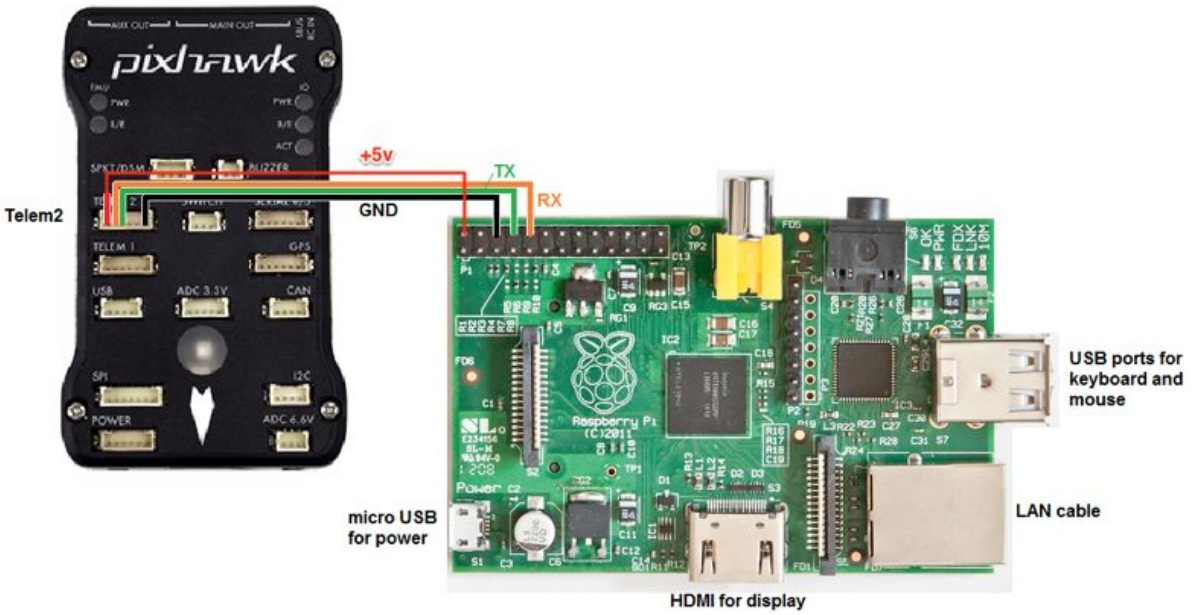


Figure 6. Diagram of pixhawk - RPI2 serial port connection. Notice that there only 4 wires are used for this connection. The 5V wire also powers the RPI2 when the pixhawk is connected to the LiPo battery.

To install: (Detailed instructions here: http://dev.ardupilot.com/wiki/companion-computers/raspberry-pi-via-mavlink/)

1. Get updates, install NumPy, MAVLink, and MAVProxy,
   sudo apt-get update
   sudo apt-get install screen python-wxgtk2.8 python-matplotlib python-opencv python-pip python-numpy
   sudo pip install pymavlink
   sudo pip install mavproxy
2. Disable the serial port messages.
   sudo raspi-config
3. Reboot RPI2

Object Recognition

In this project, the drone needed to recognize a red object and land. To recognize red objects, OpenCV was used in conjunction with Raspicam to take and process images. Thus, the basic algorithm is to take a picture using the camera module, storing it as a matrix of pixels. A color range is specified and a bit mask is applied to the matrix in order to get rid of the pixels that aren't within our color range. This is called applying a threshold to the image. Once that is done, we search the matrix for subset of the remaining bits that is greater than a pre-specified radius. This is done to get rid of excess noise in our data. A boolean value is returned, indicating whether an object has been detected or not.

The actual coded algorithm can be found in this repository:
https://github.com/connorlee77/RPiVisionAPI

Integrating with Drone API

The object detection algorithm must be called from a program that commands the actual movement of the drone. The Drone API was used to make the drone fully autonomous. For this project, the drone needed to takeoff, hover and detect objects, and land when appropriate. Thus, the basic algorithm behind our command scripts was to arm the drone, take off, and start running our object detection algorithm. When a red object is detected, the drone will land. Note that the drone must be armed in STABILIZE or GUIDED modes. A pseudocode of the commands is as follows:

```
//Arm the drone.
vehicle.armed = true

//Takeoff.
vehicle.commands.takeoff(height)

//Detect objects.
while no red object detected:
        is_object_detected()
//Land.
vehicle.mode = VehicleMode("RTL")
```

Since our object detection algorithm was written in C, the command scripts must call the executable file and read the outputted string from the C file. This was done using Popen and PIPE. An example of how this is done is available here:
https://github.com/connorlee77/RPiVisionAPI/blob/master/test.py.

Flying the Drone

Having the command script integrated with the object detection algorithm, we can now fly the drone autonomously. To do this, connect to MAVProxy by entering:

sudo –s
mavproxy.py --master=/dev/ttyAMA0 --baudrate 5760

This should launch MAVProxy. This means that the RPI2 which contains our command script is connected to the pixhawk. Allow the console to load for a couple of seconds. To start the drone, change directory into the file where the command script is located. Then enter:

api start command_file.py

Experiments

After building and calibrating the quadcopter we were able to fly it manually. With a bit more tinkering and spending some time on Mission Planner we were able to get the quadcopter to fly autonomously based on GPS coordinates. Once these flights had been successful we moved forward to mounting the Raspberry Pi. Unfortunately we were never able to successfully autonomously fly the quadcopter but we had small successes once mounting the Raspberry Pi. It was able to write and execute scripts in live time, even while flying. The one hurdle was that the quadcopter would not take off autonomously. Initially it seemed as if the issue was with autonomously arming but after fixing that bug on one of the last days the lift off error continued to hinder the autonomous flight. We decided to simulate the take off by hand and lift it to the hovering height of three feet. Then while holding the quadcopter it was able to execute all the remaining steps. The scripts wrote correctly in live time and then when the ball passed under the camera they told the quadcopter to land.

Taking off and flying autonomously using Mission Planner:
https://www.youtube.com/watch?v=6tPfZTecLmQ

Taking off and flying manually:
https://www.youtube.com/watch?v=GTC4JCbGQB0

Calibration of drone using a weighted platform:
https://www.youtube.com/watch?v=0NGv7lUkWkE

Conclusion

Though we were not able to meet our goal with 100% success, we are still very happy with what accomplished. Besides taking off autonomously, our quadcopter was successfully able to recognize a red tennis ball that rolled under it and trigger the landing sequence. We believe that given a few more weeks we could have met our goal completely.

Were we to repeat the term, there are a few things which we would have done differently. Hardware setbacks such as waiting for the PPM encoder were time costly and could have been avoided with more research or by planning ahead better. Also, some flight failures could have been avoided with more careful reading of the Pixhawk documentation - at the time we were so eager to get the quadcopter flying that we skipped some crucial steps.
The experience has been invaluable and taught us many skills that will be undoubtedly useful in our future endeavors. Through troubleshooting and failures, we have learned the basics of quadcopter flight and image processing, as well as the basics of the RC hobby world. On top of these, the ten week time limit we had taught us a lot regarding problem solving and making do with what's available. With these experiences under our belt, we now feel comfortable designing and creating a quadcopter from scratch that utilizes the most cost efficient parts and task-suited components while maintaining a versatile and powerful design. Additionally, we now feel comfortable porting our experiences to other fields: whether it's applying the vision software to ground systems for security purposes, or designing a small vehicle to complete a given task.

We are not done with our project just quite yet. These ten weeks served as a great platform from where we could familiarize ourselves with quadcopters and image processing. Now that we know the basics, we believe there are many useful and interesting projects that we could pursue as further work. Some of our current ideas include creating a quadcopter security system that utilizes facial recognition, or perhaps building a quadcopter that is suited for flying through landscapes to detect old war mines. The capabilities of the combination of UAV's and machine learning are endless. Major challenges arise in areas such as battery life, cost, and processing power but technology is improving day-to-day making this one of the biggest up and coming fields.

At this point we would like to take a moment to thank everyone who helped us as well as our sponsors. We would like to thank Dr. Perona (Caltech) for being our mentor and helping guide our progress. We would also like to thank Kairos and Lightspeed Ventures for helping fund our project.

Resources

Materials needed/used:

Item:	Price:	Description:	Quantity:
Mission Planner	Free	http://ardupilot.com/downloads/	x
ESCs	$10	20 amp brushless ESC for quadcopter	5
Frame	$120	ArduCopter 3DR Quad Frame (main frame, arms, landing gear)	1
Wires	$15	Power cables and Signal Cables for PDB-APM. Female/Male XT60 Adapters. This includes various cables	1
uBlox GPS+Compass	$80	Compatible with Pixhawk, allows for autonomous GPS flight	1
Pixhawk Flight Controller	$200	pen source platform for autonomous flight control, includes gyro and barometric sensors	1
Propellers	$8	APC Propeller set (10X47)	2
Batteries	$35	APM Power Module XT60	2
Motors	$10	IRIS (850Kv) AC2830-358	4
Splitter to quadcopter (power control)	$15	Quadcopter power distribution board	1
Receiver	$40	FrSky DJT module	1
Transmitter	$60	Turnigy 9x	1
Vision API	Free	OpenCV	x
PPM Encoder	$20	3DR PPM encoder	1

Key Resources:

Ardupilot Multicopters: If you are interested in getting into multi-copters, and especially if you are thinking of using Pixhawk, this is the most critical resource. It is well worth reading through the entire link. It has basic information about what parts are and what they do, as well as detailed

tutorials on how to set up and use Pixhawk. You will avoid many headaches if you read this manual in advance.

Quick Hardware Reference Links:

Suggested Tx and Rx from Ardupilot: Quick link for what receivers and transmitters are compatible and suggested with the Pixhawk
Detailed Turnigy 9x and FrSky mod: How to physically change your Turnigy 9x so that it works with the FrSky transmitter module
Binding Receiver and Transmitter (Turnigy and FrSky):
Making Home Made Bind Plug for Binding Transmitter

Quick Software Reference Links:

Various flight modes: Explanations of the various possible flight modes. The primary ones used in this project were Stabilize and Auto
http://hackaday.com/2014/06/20/autonomous-balloon-popping/
Camera -> RPi -> Mavlink -> Pixhawk
http://www.raspberrypi.org/forums/viewtopic.php?t=77333&p=551394

For Fun:

Professional builds examples - these quadcopters are built by heavy multicopter enthusiasts and are good examples of quality builds. Many include parts lists and pictures of assembly as well as flight videos.
General info - a nice wiki style page containing basic info and links for quadcopters