CORNELL UNIVESITY

Autonomous Quadcopter Docking System

ECE M.ENG. Design Project Final Report

Sima Mitra
Advisor: Bruce Land

Spring 2013

Abstract

The goal of this project was to design the systems and algorithms necessary to allow a quadcopter to autonomous locate and land on a station target. The purpose of this system was to outline the framework for a quadcopter based data collection or surveillance system that copes with the relatively short battery life of these highly mobile devices by consistently landing the AAV safely in a designated location to be recharged. The 3D Robotics ArduCopter was chosen as the quadcopter platform since it is capable of autonomously hovering in place and is capable of carrying a payload, such as the camera used to determine the location of the dock. A system was devised such that the quadcopter can correctly determine the location of a target ground station while hovering and then land when above the target. Only commercially available components and free software were used to so that the entire docking system is easily accessible to future researchers and UAV enthusiasts.

Executive Summary

In this project a system was designed to autonomous land a quadcopter using software and materials easily accessible to students and UAV hobbyists. The 3DRobotics Arducopter quadcopter was chosen after a rigorous selection process for it rugged design, carrying capacity, degree of autonomy supported, open source software, and preexisting community. A control system was designed using a UBS radio and open source program MAVProxy/MAVLink after the original software was proven inadequate.

When purchasing an RC controller to test and calibrate the Electronic Speed Controllers (ESCs) of the quadcopter proved to be prohibitively expensive, I instead chose to use USB Xbox 360 gaming controller to control the quadcopter manually thought the telemetry radio already being used for inflight commands by altering an existing MAVProxy modules for joystick control. There was no precedent for calibrating the ESC without a controller but I was able to create a method to do so.

While Arducopters are often used to take pictures, there is little precedent for using a camera to control the quadcopter automatically. To save on cost, an old Android Smartphone was attached to the quadcopter to provide a live video feed. To control the quadcopter a new module was created that integrates with MAVProxy that uses the computer vision library SimpleCV to search for a marker to identify the target area in which to land. A custom module was written for the control program that will instantiate a separate thread that views the video feed of an android smart phone and uses computer vision to search for a large red rectangle. If the marker is detected and centered in the video, an RC override command is set to cause MAVProxy to land the quadcopter.

With system it is possible for a quadcopter to autonomous land in a target area and thus be retrieved for charging. All software used is free and all control programs and computer vision libraries used are open source and python based for better accessibility to students and UAV enthusiasts.

Table of Contents

Introduction

Radio controlled helicopters and planes are highly valued for their ability to remotely search wide areas without the risks of traditional manned aircraft and have high potential for use in remote sensing applications, surveillance, and scientific research. As computing power has increased, so have the autonomous abilities of these devices. Autonomous aeronautical vehicles, or AAVs, have a wide appeal due to their maneuverability, speed, and wide range compared to land based devices. These AAVs are particularly useful for situations that are too dangerous for humans, such as in disaster relief, surveillance, or radiation level detection. This proposal will be focusing on developing systems for autonomous multicopters, specifically quadcopters.

One of the main disadvantages of remote autonomous and unmanned systems is their extremely short battery life. For multicopters, the tradeoff between battery capacity and battery weight can result in a flight time that may be as little as minutes. The short battery life of these devices is one of the main reasons that currently make their widespread deployment unfeasible. It also means that current autonomous systems must periodically break from their missions and be retrieved by their human supervisors to be recharged. By developing a system where by a multicopter could sense its battery level and return to a charging bay to which it could automatically connect, the degree of human supervision required decreases. An autonomous multicopter could be sent to take images or collect data, pause when its battery needed to be recharged, and then return to its mission when it was fully recharged.

Docking stations also allow for more multicopters to be potentially deployed. For example, many quadcopters could deploy to an area to collect data, and as their batteries were depleted they could return to charge at a free station. This could be coordinated such that a certain number of copter are always flying, which would be advantageous for certain applications like surveillance or other situations when it is imperative that data collected is continuous but it is difficult to build an equally widespread stationary sensor array.

Design Problem

While improving the mechanics, aerodynamics, and reliably of Multicopters is an active research area, the basics of quadcopter design is a solved problem. Instead of building in a quadcopter, the focus of this project was to explore what could be done with an autonomous quadcopter. Currently, the main constrain on multicopters is their battery life, which can limit the flight time to as little as 10 minutes. This problem extends to all multicopters and often is invariant of the scale of the machine: for example the Seeed Studios Crazyflie [1], a prototype nano-quadcopter that fits in the palm of your hand, can achieve 7 minutes of flight time with its 170mAh Li-Po battery [1], which is comparable to the 5-7 minute flight time of the quadcopter used for this project 3DRobotics Arducopter with the recommended 2200mAh Li-Po battery [2]. Just as gas stations extend the range of an automobile, one solution is to provide designated recharging areas for these multicopters.

This leads to another real-world design constrain: cost. Multicopters are not simply RC toys and their price reflects their high performance hardware. Similarly, extreme multicopter acrobatics and controls research is often performed using a motion capture systems like VICON [3], which provides exact 3D localization [4]. These systems can cost tens of thousands of dollars, thus limiting their use to large research institutions. However, there is still an avid community of multicopters and Unmanned Aerial Vehicles (UAVs) enthusiasts exploring this field without specialized equipment. Therefore, another design constrain of this project was to implement this system cost effectively and only using commercially available components and free or open source software in the hopes that this project could contribute back to the amateur community.

Since the battery chosen was a 3 cell Li-Po and it cannot be recharged safely without all 6 balanced charge contacts connected to a Li-Po battery charger, it was determined that the quadcopter would simply land in an area where it could be retrieved and manually charged, not unlike a gas-station provides a pit-stop for cars.

Design Solution Evaluation

Quadcopter Choice

One of the first major design choices made was choosing the type of multicopter to be used in this project. A quadcopter platform was chosen because in general they are capable of hovering in place, robust, well balanced for the amount of lift they generate, and are widely used in the UAV community.

Since the goal of this project was to explore the use of quadcopters rather than design a quadcopter, it was determined that it would be best to choose a pre-existing quadcopter that was affordable and relatively-easy to construct since most amateur UAV uses, myself included, can be expected to have access to soldering tools, but not welding or laser-cutting equipment. This meant that the quadcopter chosen would either be fully assembled or be part of a kit. Another major constrain considered was the size of the quadcopter. While small quadcopters have the advantage of being easy to transport and lower priced, it was essential that the platform chosen be able to accommodate any additional hardware, sensors, and their power sources without over-burdening the quadcopter. This excluded tiny 'toy' quadcopters, but small to medium sized devices would provide enough lift to support such peripherals. This selection process narrowed down the quadcopter platforms available at the time to two possible candidates: the Parallax ELEV-8 [5] and the 3DRobotics Quad-C Frame ArduCopter [6].

Another aspect considered was the degree of prebuilt autonomy of the quadcopter. Since one of the design constrains was that the project platform needed to be accessible to students and amateurs, it would be preferred if the quadcopter already supported some autonomy, such as the ability to hold

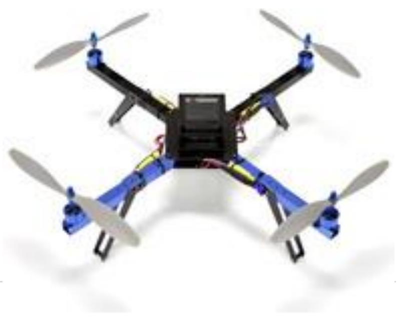


Figure 1: 3D Robotics ArduCopter [6], the quadcopter chosen for this project.

position and hover unaided. In this regard, multicopters in the ArduCopter family have a distinct advantage. The ArduCopter [7], is a new open source Arduino-based multicopter platform developed by the UAV enthusiasts of the DIY Drones [8], a community devoted to amateur UAVs. The goal of the ArduCopter project is to support a diverse set of features, from stabilized manual flight to automatic waypoint visiting using GPS. The 3D Robotics ArduCopter is controlled via the ArduPilot Mega 2.5 (APM 2.5) board [6], which supports all of these features. The ArduCopter it is open source and anyone can download, develop, and contribute back to the community. Due to its preexisting community and open-source software, the 3D Robotics ArduCopter [9] was chosen as the quadcopter base for this project and a 3D Robotics ArduCopter kit [6], as opposed to the more expensive pre-built quadcopter, was purchased.

Controls

The next major design choice was the control system for the quadcopter. On the ArduCopter page they suggest purchasing an RC controller for manual control and ESC calibration, and using the Mission Planner Utility [10] for autonomous functions such as waypoint vising and for tuning advanced parameters of the ArduCopter's autopilot. This proved to be a challenge, as it appeared most of the instruction for calibrating the quadcopter and enabling manual failsafes seems to require having an RC controller even when the quadcopter was intended for autonomous flight only. Additionally, RC controllers are expensive: the suggested controller [2], the Spektrum DX7s [11], cost more than half of the price of the quadcopter kit. Further evaluation showed that advanced RC controllers could cost up to 1.5 times the price of the quadcopter. While an RC hobbyist might already have such a device, it is an expensive accessory that would be barely used when focusing on autonomous projects. I chose to not purchase the RC controller and instead create a way to emulate the failsafes and ESC calibration without it. Directions for ESC calibration without an RC controller are included later in this report in the ESC calibration without an RC Controller section.

Another challenge arose when I began implementing the autonomous functions of the quadcopter. While the Mission Planner Utility provides a visually appealing user interface, it ultimately proved inadequate. While the Mission Planner claimed to be capable of using python scripts for autonomous actions, in the latest version of the program it seemed like this feature was not well supported and no simple scripting examples existed. Certain essential features, such as the accelerometer calibration, simply did not work. While the accelerometer calibration was fixed by reverting back to an older version of the program, it became clear a better control method was need.

Instead, I chose to use MAVProxy [12], a command line interface for UAVs that is the backbone of QGroundControl [13], an open source ground control station for small autonomous unmanned systems. MAVProxy uses Micro Air Vehicle Marshaling/Communication Library, or MAVLink [14], to communicate with the ArduPilot Mega 2.5 control board on the quadcopter. MAVProxy [15] and MAVLink [16] are both open source python based utilities, which fit well with my desire to use free and open software. Directions for using MAVProxy are included later in this report in the Using the MAVProxy and custom modules section. This meant the on-board Autopilot did not need to be altered and the quadcopter could be control using a ground station computer over a telemetry link [17] using a 915 MHz 3D Robotics USB radio [18] that is both well supported by the APM 2.5 and uses a frequency in the ISM band.

Another advantage of MAVProxy is it support of additional modules which can be used to control camera gimbals and other peripherals. While I had intended to use the position hold and altitude hold modes of the quadcopter to fly it fully autonomously, I found that these features were difficult to debug. Not having access to an RC controller, I used module called joystick included in the MAVProxy code based in Modules folder. This module allows a USB gaming controller to emulate an RC controller over the telemetry radio and is also supported in the Mission Planner [19]. I purchased a Logitech F310 USB Xbox controller [20] and edited this module by added an entry for an Xbox controller in the look-up table (see the Modified controller code: mavproxy_joystick.py section of the Appendix). This mapped the analog axis of the Xbox to the pitch, roll, yaw and throttle controls. I also mapped some of the additional buttons on the controller to switch the mode of the quadcopter (such as Stabilize, Position

Hold, Altitude hold, and Land). This allows the quadcopter to be flown manually in Stabilize mode, which uses the accelerometer and a PID loop to stabilize the quadcopter in flight.

Target Sensing

The next major hurdle was choosing how the quadcopter would find and identify a charging station. While the quadcopter I purchased did include a GPS, this would only be accurate up to a few meters and alone would not be capable of landing on a small target.

It was decided that for this system the ground station computer would handle flying as well as landing the quadcopter, as opposed to altering the quadcopter's autopilot or having a separate controller present on at the target directing the quadcopter because this would consolidate the controls to a single computer.

While several other methods of identify the target were initially considered, such as having the quadcopter search for modulated light with a specialized receiver, it was decided that the quadcopter would use computer vision to identify a marker at the target. This would provide the added bonus that the quadcopter would also have a camera for general use. While simple cameras like the CMU cam [21] were initially entertained, the simple solution I arrived at was to use an old Android smartphone running the free app called IP webcam [22], as many people have an old smart phone they are no longer using. The application allows you to view the video stream from phone over Wi-Fi by accessing the phone's IP address. For the project I used an old HTC Incredible 2 [23] which was donated for free.

Since the computer used had Wi-Fi, this allowed for a dedicated link for the camera that did not interfere with the control program for the quadcopter by keeping the high bandwidth video information on a separate channel and by having all of the computer vision done by the computationally more powerful ground station computer (a laptop).

The target identification was handled by using SimpleCV [24] an open source python based computer vision library. SimpleCV supports advanced macros to do face detection, QR code detection, and connected component detection. With SimpleCV it is very easy to view the video stream from the Android smart phone if the base computer is on the same network. As a python library, SimpleCV is relatively simple to interface with the MAVProxy controls for the quadcopter.

Target choice

The marker for the station chosen would need to be unique in the area the quadcopter was surveying. Initially, the quadcopter was going to use QR codes to identify a station as they are unique do not look like naturally occurring terrain. However, in the process of testing it was discovered that the QR code detector used in SimpleCV is too slow for the frame rate of the IP webcam and if the camera was moving or vibrating this would be render the image too blurry for it to be recognized. Instead, a script was implemented that detects large red rectangles, which were unique enough in the indoor and lawn-covered outdoor environments tested.

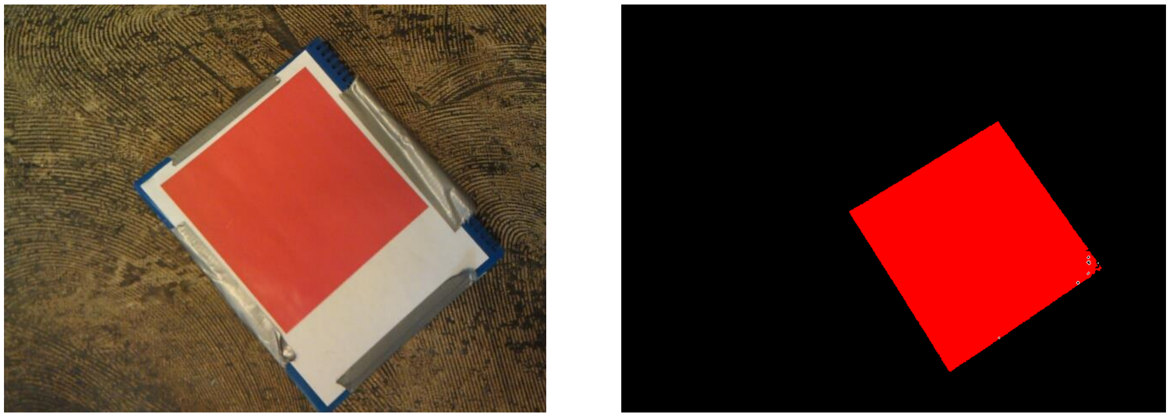


Figure 2: Right: The marker used in this project. Left: The thresholded view of the target used by the algorithm.

Final Design

Overview

In the final design a 3DRobototics quadcopter is controller using the ArudCopter ArduPilot Mega 2.5, an autopilot board that handles that stabilized flight, integrates the local sensors such as GPS, sonar and battery monitoring as well as radio communication [6]. This quadcopter was chosen for it size, payload capacity, preexisting community and open-source software (see Quadcopter Choice section). The quadcopter’s flight is controlled using a ground station computer over a 915 MHz communication link [17] using a 3D Robotics radio [18], which was chosen based on its compatibly with the APM 2.5 (see Controls).

The quadcopter is controlled using a command prompt interface by the program MAVProxy [12], which initiates a MAVLink [14] communication link to the quadcopter over the radio. The MAVProxy program was chosen for its support of control scripts and extra modules (see Controls). This ground station computer can change the mode of the quadcopter to be flown under manual control using a USB gaming controller [20] to emulate an RC controller [19] and by setting the flight mode of the quadcopter to Stabilize (see Using the MAVProxy and custom modules for how to use this module and the Modified controller code: mavproxy_joystick.py section of the Appendix). It can also switch to any one of the enabled flight mode from the terminal, such as Return-to-Launch, Position Hold, Altitude Hold, Land, etc.

An Android smartphone is placed on the bottom of the quadcopter and the video feed of the phone is accessible at the phone’s local Wi-Fi IP address using a free app called IP webcam [22]. This camera system was chosen because it is very easy to interface with, is supported by SimpleCV and would likely work with any smartphone running a similar application (see Target Sensing). The quadcopter is able to sense the location it must land at using a custom MAVProxy module mavpoxy_android.py that I wrote (see Android Camera module for detecting target: mavproxy_android.py in the Appendix). This module uses the computer vision library SimpleCV [24] to search for a large red rectangles in a separate thread from MAVProxy. A simple red square was chose as the target because this feature is simple to detect but was unique enough in all of the environments tested to avoid misidentification (see Target choice). This program also writes a log file that timestamps when commands are executed for debugging purposes. If a large red rectangle is detected in the center of the video, an RC override command is set to cause the quadcopter to enter the flight mode LAND, which will cause the quadcopter to quickly descend to the ground and will slow down the motors as the quadcopter detects it is no longer moving and level on the ground (see Android Camera module for detecting target: mavproxy_android.py in the Appendix). With this system a flying quadcopter could be triggered to land in a relatively small area, allowing for it to manually recharged.

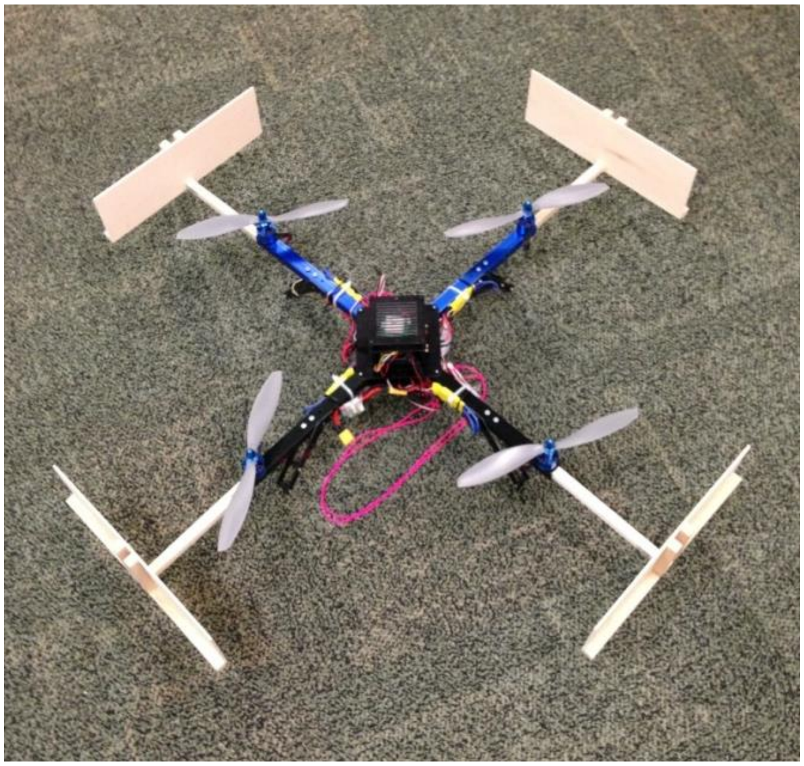
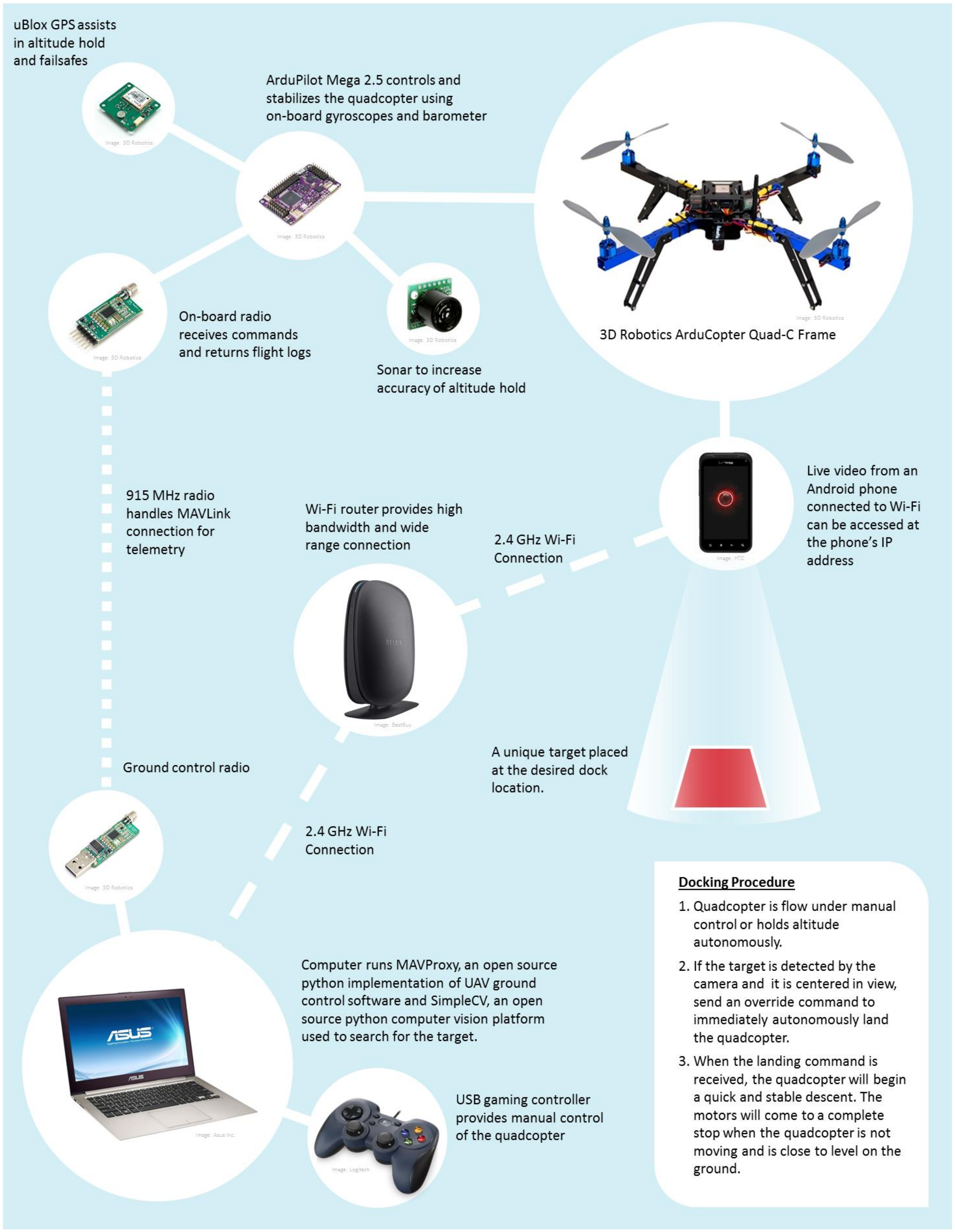


Figure 3: The constructed quadcopter with added pieces to protect the propellers during test flights.

A small change was made to the frame of the 3D Robotics ArduCopter to place wooden 'baffles' on the ends of the arms of the quadcopter. This protects the propellers, nearby objects, and people during testing and flight. The baffles were made from ¾ in square dowels that were placed in the hollow ends of the quadcopters arms. The outer size pieces are 4 inches high which protects the propellers when the quadcopter is flipped upside down. Since the location of the motors was not changed, the baffles have a limited effect on the flight of the quadcopter and no changes were made to the PID loop parameters.

During testing the quadcopter was always anchored to a weight on the ground using a tether, clipped in either in the center of the quadcopter or on opposite feet of the quadcopter. Originally a small lead puck as used due to its low profile, but the quadcopter proved it was capable of pulling this weight. Two large cinder blocks were later used during testing, which proved to be more than enough to secure the quadcopter.


uBlox GPS assists in altitude hold and failsafes
ArduPilot Mega 2.5 controls and stabilizes the quadcopter using on-board gyroscopes and barometer
3D Robotics ArduCopter Quad-C Frame
On-board radio receives commands and returns flight logs
Sonar to increase accuracy of altitude hold
915 MHz radio handles MAVLink connection for telemetry
Wi-Fi router provides high bandwidth and wide range connection
2.4 GHz Wi-Fi Connection
Live video from an Android phone connected to Wi-Fi can be accessed at the phone's IP address
A unique target placed at the desired dock location.
Ground control radio
2.4 GHz Wi-Fi Connection
Computer runs MAVProxy, an open source python implementation of UAV ground control software and SimpleCV, an open source python computer vision platform used to search for the target.
USB gaming controller provides manual control of the quadcopter

Docking Procedure

1. Quadcopter is flow under manual control or holds altitude autonomously.
2. If the target is detected by the camera and it is centered in view, send an override command to immediately autonomously land the quadcopter.
3. When the landing command is received, the quadcopter will begin a quick and stable descent. The motors will come to a complete stop when the quadcopter is not moving and is close to level on the ground.

Figure 4: Overview of the final design implemented.

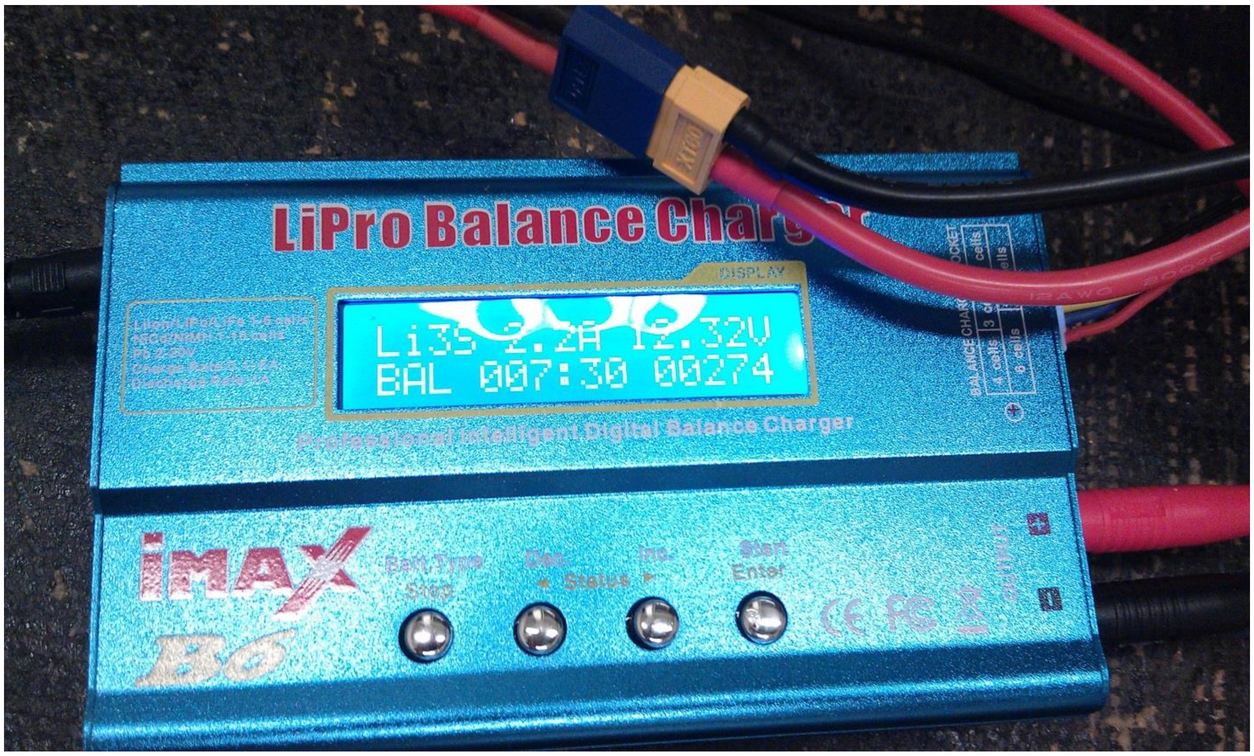


Figure 5: The iMax B6 was chosen because it is affordable battery charger than can balance charge a multicellular Li-Po battery, such as the 3 cell battery that powered the quadcopter.

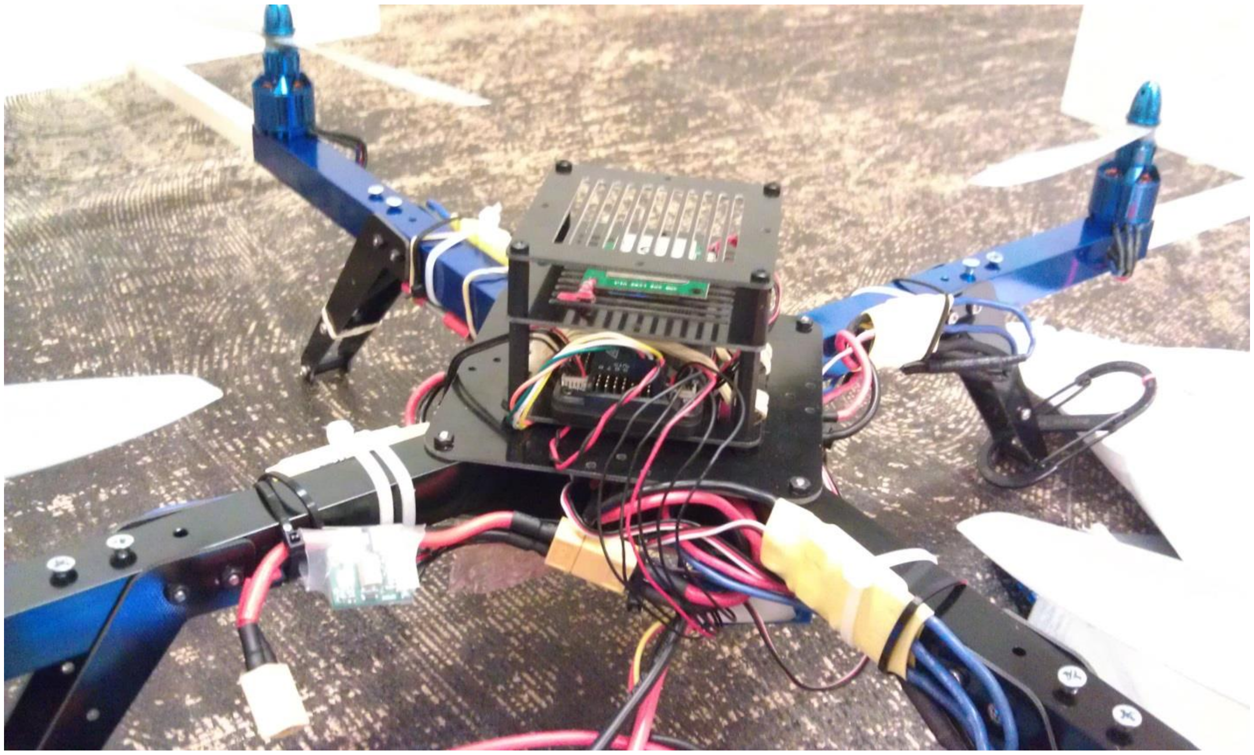


Figure 6: The center of the quadcopter, showing the battery connector and ArduPilot control board.

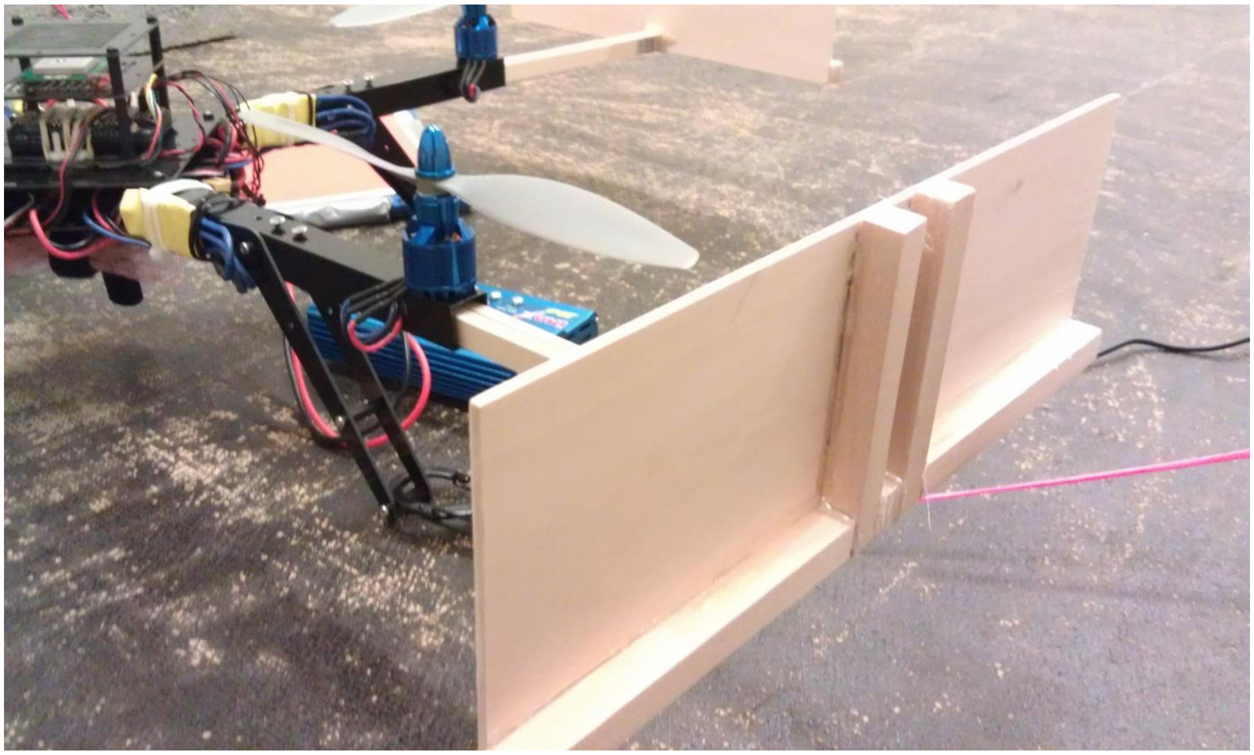


Figure 7: These propeller protectors were designed to prevent the propellers from ground on uneven take offs, nearby objects when in flight and the ground when the quadcopter flips over.

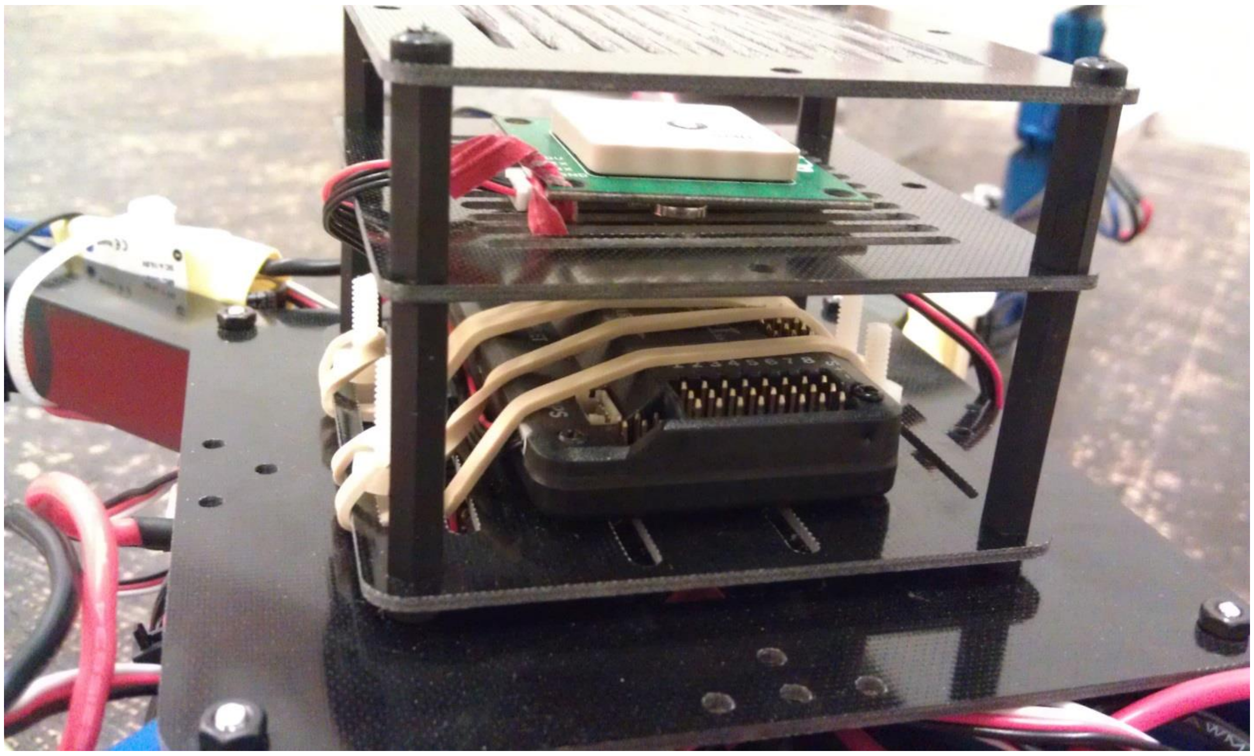


Figure 8: The ArudPilot Mega board strapped in with rubber bands for quick access with GPS mounted above.

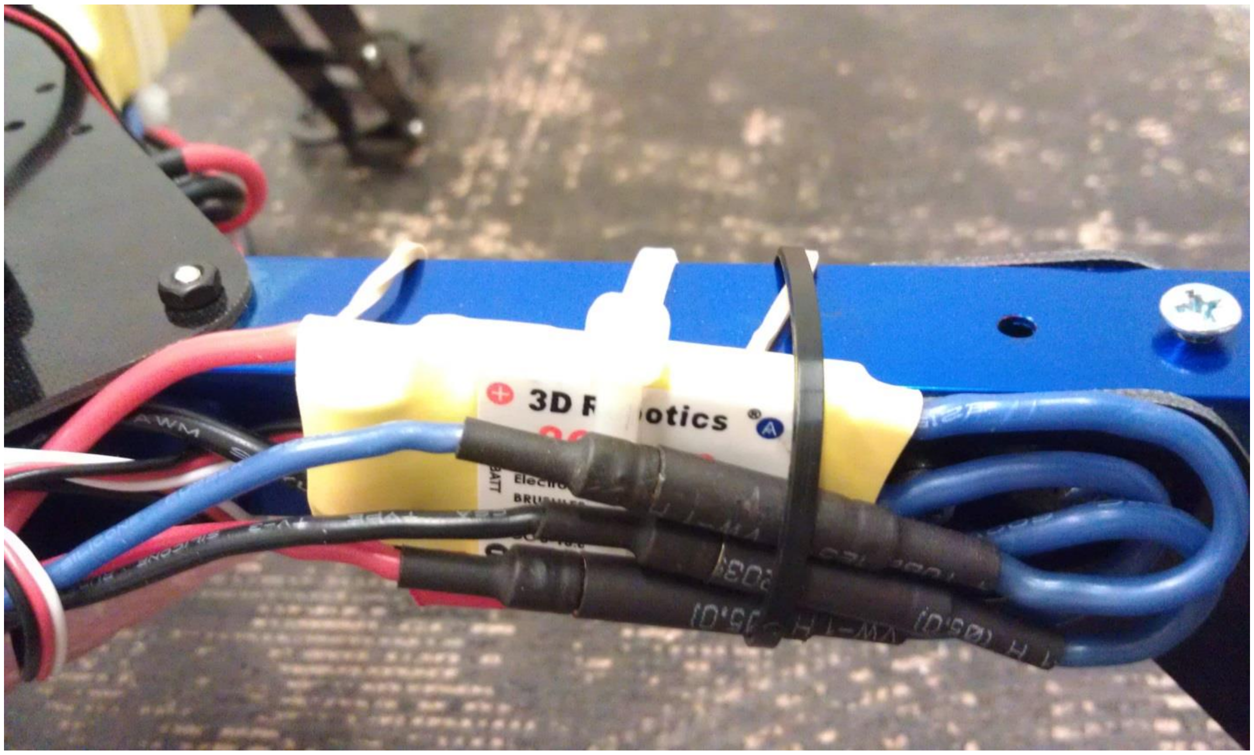


Figure 9: Electronic Speed Controllers (ESCs) for the quadcopter's motors were attached to the arms of the quadcopter.

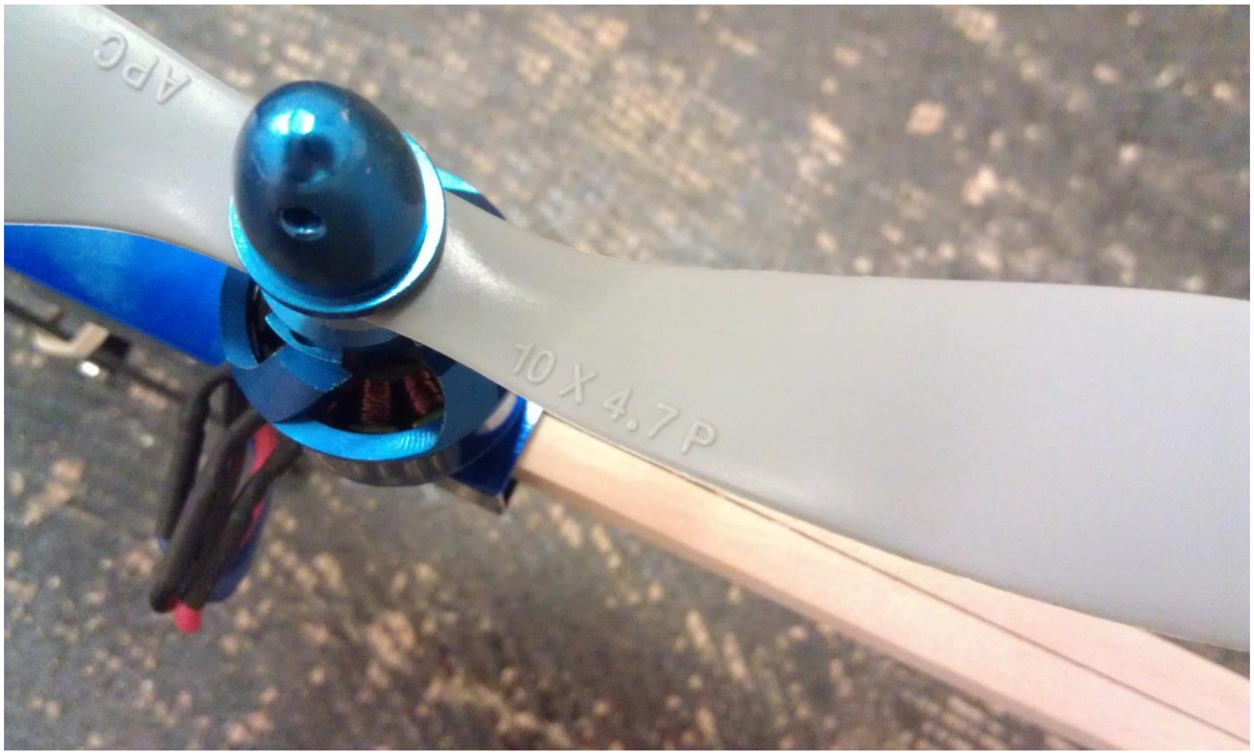


Figure 10: Propeller mounted to brushless motor. Propellers should be mounted with text facing up.

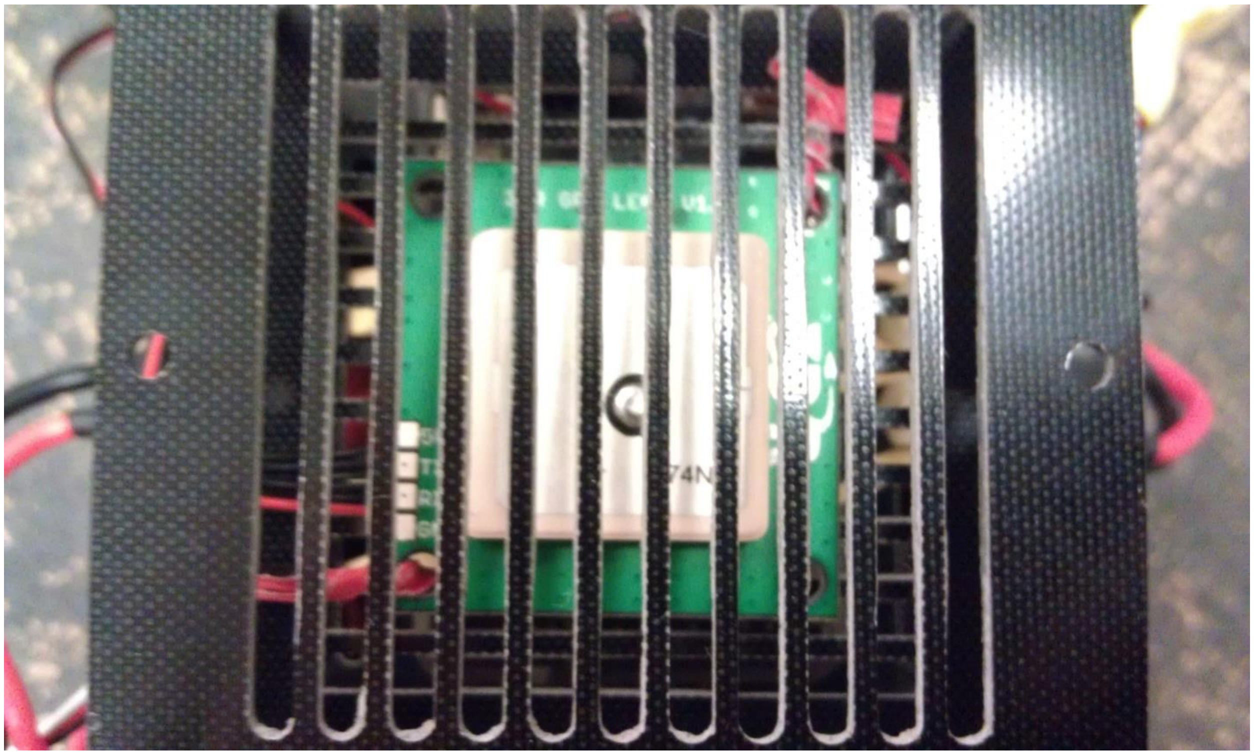


Figure 11: uBlock GPS is protected by the fiberglass top.

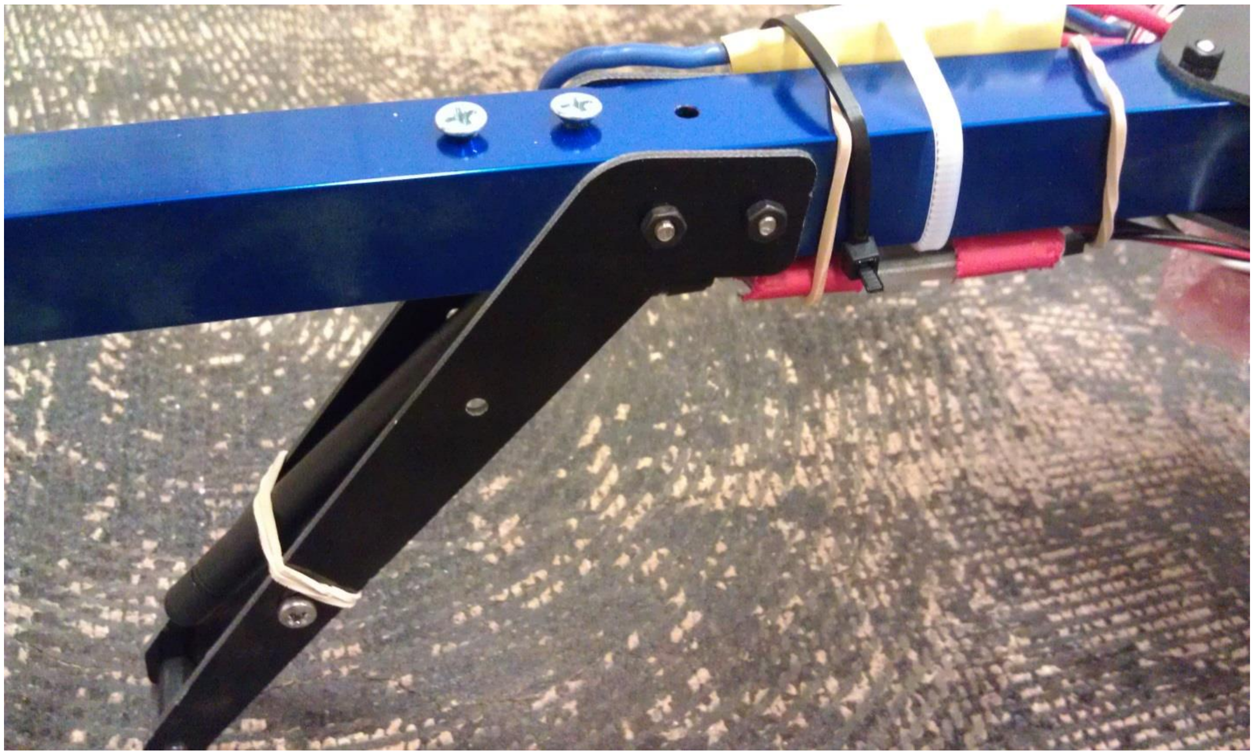


Figure 12: The quadcopter's radio should be mounted as far from the GPS as possible to reduce interference. I chose to mount the radio in the quadcopter's leg to protect it during crashes.

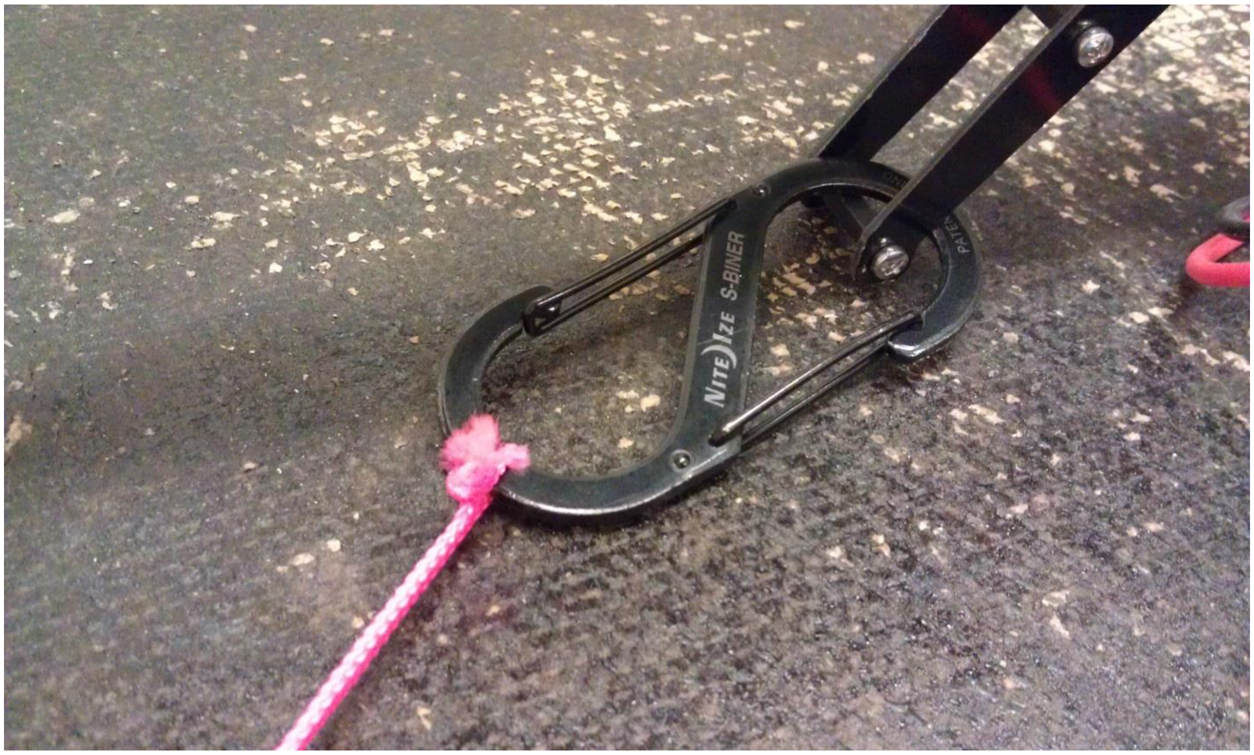


Figure 13: During testing the quadcopter was tether during testing.

Instructions

Notes on Assembly instructions

The assembly instruction for the 3DRobotics ArduCopter can be found here [25]. Some additions hints include:

- The assembly instruction are often value on how to connect the motors
  - The order of the bullet connectors on the ESCs cannot be determined at assembly time. Simple test the quadcopter with props on (and tied down) by holding a piece of paper above them. If the paper is pushed up, switch two of the wires.
  - The Red-black-white 3 wire connectors are plugged in upside down on the power distribution board.
  - When connecting the 4 wire connect to the APM, the yellow wire should be in the first position.
- The instructions tend to switch the ordering of the motors, sometimes clockwise counting then and other times diagonally numbering them. Your quadcopter is correctly wired if you run the Motors test [26] in the command line interface (CLI) and each motors spins for a short time, moving from the first motor at the front right size of an X-configuration quadcopter counter-clockwise around the frame. Do not try to match the diagram, watch the video instead.
- The propellers should have the writing facing *upward*.
- Some kits include propeller mounts that press fit onto the motors shaft, while others screw directly to the outer motor casing. If possible, avoid the first kind, as they will fit poorly and have a tendency to come off in flight but will shear at the base of the bolt if over-torqued.
- Accelerometer and ESC calibration is mandatory. A well calibrated quadcopter will lift of the ground smoothly in stabilized mode.

Software downloads

- Download the SimpleCV 1.3 superpack [27] and run.
- Under Control Panel\System and Security\System, open Advanced System Settings. In systems settings dialog box, got to the Advanced tab and click on Environment Variables. Added the following:

PATH: C:/Python27/;C:/Python27/Scripts/;C:/OpenCV2.3/opencv/build/x86/vc10/bin/;

PYTHONPATH: C:/SimpleCV1.3/files/opencv/build/python/2.7/;C:/OpenCV2.3/opencv/build/python/2.7/;

- Download the QR code reader module [28]
- Download the MAVProxy [15], MAVLink [16], and all their required windows libraries by follow these directions [12].
- Install IP webcam [22] on your android phone, or use the webcam of your computer (cam = Camera(0))

ESC calibration without an RC Controller

While it is high encouraged that an RC controller is used in addition to a telemetry radio, ESC calibration [29] can be done using MAVProxy and a gaming joystick. The quadcopter should not have props on or be securely held down in a rig.

1. Disconnect the battery XT60 connector and remove the pair of red and black wires that goes to the APM.
2. Connect to the APM to the ground control computer over USB. This will power the board. Connect to the APM over USB by typing:
   ```
   python mavproxy.py --master=COM4 --baudrate=115200 --
   quadcopter --aircraft=quadcopter
   ```
   (Comport will vary so check the comport in the device manager or in the Mission Planner)
3. Load the joystick module by typing:
   ```
   module load joystick
   ```
4. Put the throttle stick to full and hold it.
5. Connect the Battery. There will first be several beep indicating the number of cells present in your battery (3 in this case), then after a pause, two short beeps together. After the two short beeps, drop the throttle stick to zero. If calibration has occurred, there will be a confirmation tone. Moving the throttle stick a little should cause the motors to spin.

Using the MAVProxy and custom modules

1. Start a new windows command prompt and navigate to the MAVProxy folder.
2. Start MAVProxy by typing:
   ```
   python mavproxy.py --master=COM3 --baudrate=57600 --
   quadcopter --aircraft=quadcopter
   ```
   (Comport will vary so check the comport in the device manager or in the Mission Planner)
3. Wait until a MAVLink is established and the message indicating that the parameters have been written to the APM. Do NOT interrupt the MAVLink until this is done, as it important failsafes are being written to the APM. If you interrupt, the MAVLink will become corrupted and a GPS enabled quadcopter will enter RTL mode and attempt to 'fly home' when first armed.
4. Load the Android Camera module by typing:
   ```
   module load android
   ```
5. Wait for SimpleCV to finish loading. A window will appear showing a thresholded image of the largest red object in view of the camera. If this step fails, check that the IP webcam is running, the IP address is correct in the android module, and that both the camera and computer as connected to the same Wi-Fi network.
6. Load the joystick module by typing:
   ```
   Module load joystick
   ```
7. If a matching joystick is found the module will return the name and the number of axis (should be 5). Toggle the joystick to see the values update on the terminal.
8. Arm the quadcopter by typing: ARM
9. The quadcopter will initialize gyros. Wait for this process to complete.
10. When the quadcopter returns ARMED you are ready to fly!
11. Always disarm the quadcopter before approaching by typing DISARM
12. Close the command prompt instance before starting a new MAVLink. This will insure the camera detection thread terminates.

Results

Using this system, the quadcopter has successfully landed using visual targets (see webpage video). In timing tests where the quadcopter was flying under manual control, the quadcopter is able to land from hovering 60 cm in the air in about 0.6 seconds after the Land command was received. The timelog recorded by the android module during each flight shows the amount of time elapsed between each subsequent command, showing that the quadcopter is able to determine the target is centered and ready to land in 10 milliseconds or less (see Appendix III – Example Time log from android camera module).

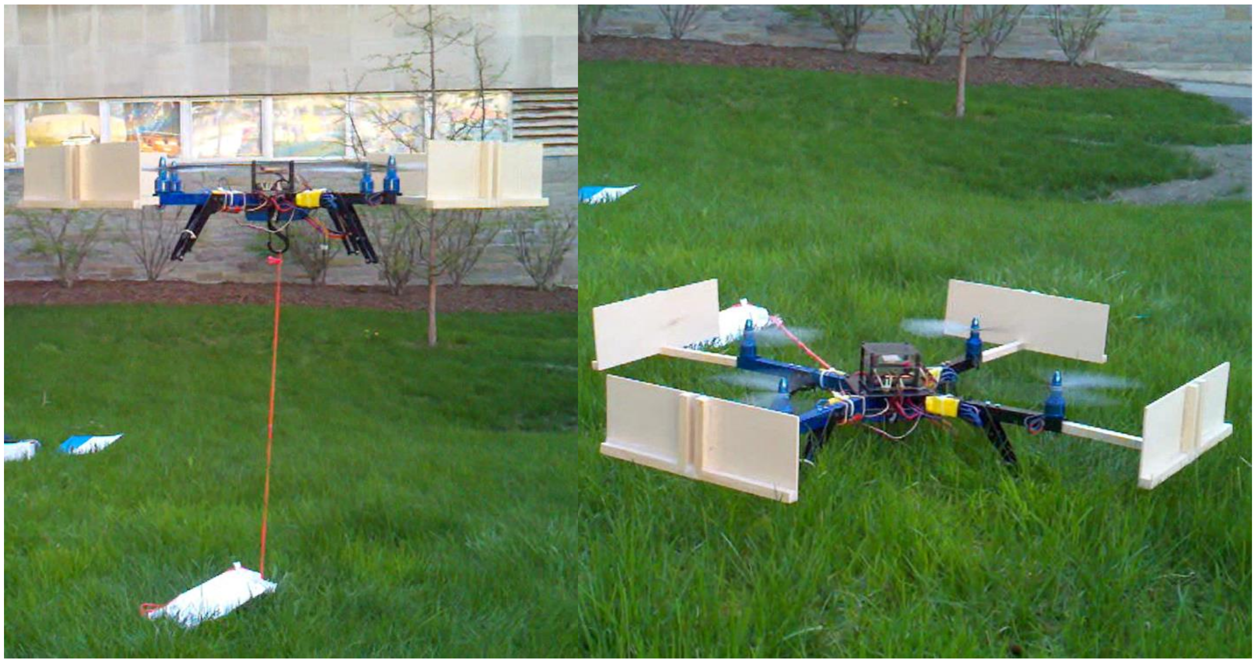


Figure 14: Left: the Quadcopter flying on a tether. Right: The quadcopter landing after seeing the red target.

The type of target used can greatly increase the time required to recognize it, thus large red squares proved to be unique object in both indoor and outdoor locations. But, as with any computer vision task, the lighting conditions can significantly change the appearance of the targets. While the Android smart phone's camera is able actively adjust, it does so much more slowly than the control loop's execution, causing the quadcopter to sometimes miss the target. The camera showed best performance indoors with indirect natural light and overhead lights are present or in complete darkness using the built in back LEDS for illumination.

One of the greatest limiting factors of the system is the update rate of the video as the video stream accumulates lag as it continues to film. This can be mitigated by having a local dedicated Wi-Fi router and by removing applications running in the background of the Android phone, but it proved to be difficult to control the quadcopter in real time.

While it had been hoped that the quadcopter could be set to fly fully autonomous using the Position Hold or Altitude Hold flight modes, this proved to be too difficult to test with. Due to the large size of the quadcopter, latency in the communication, lack of precision and drift in the sensors, when the quadcopter was attempting to hold altitude or position it would often drift to the end of its tether and then become unstable. Since a large enough outdoor area with Wi-Fi (or power outlets for a router) and clear GPS signal proved difficult to find, testing was done by flying the quadcopter under manual control and always on a short tether, both indoors and outdoors. The quadcopter proved more than capable of flying with the extra weight of the Android phone and wood propeller protectors fabricated for it.

It has also been hoped to integrate battery level sensing into this control program, but MAVProxy did not seem to integrate with this feature correctly and would often return nonsensical results. Similarly, a sonar module was added to attempt to augment the quadcopters PID loop control at low altitude but was ultimately removed since, the sonar would often, but not always, return the wrong distance until it detected an object 15 cm or further away, at which point it be fairly accurate. This proved to be detrimental to the operation of the quadcopter as the sonar is attached to the frame only 6cm above the ground and therefore when taking off from the ground the sonar would incorrectly return that that the quadcopter was 600 cm in the air until it detected its first object.

While much autonomous robotics research is conducted using complex and prohibitively expensive motion capture systems, this docking system provides proof that using off the shelf parts, an old smart phone, and free open source software can accomplished complex computer vision and autonomous vehicle control tasks that student, researches and hobbyists can all benefit.

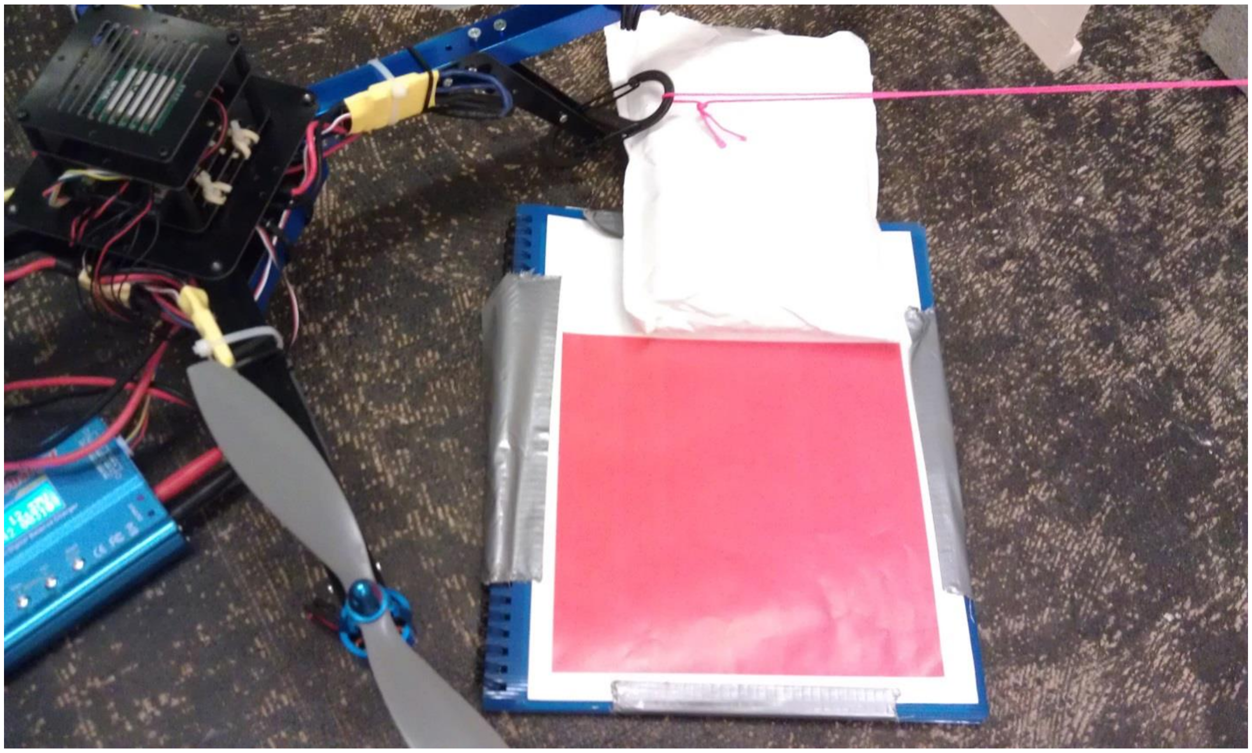


Figure 15: Quadcopter after having landed near the target.

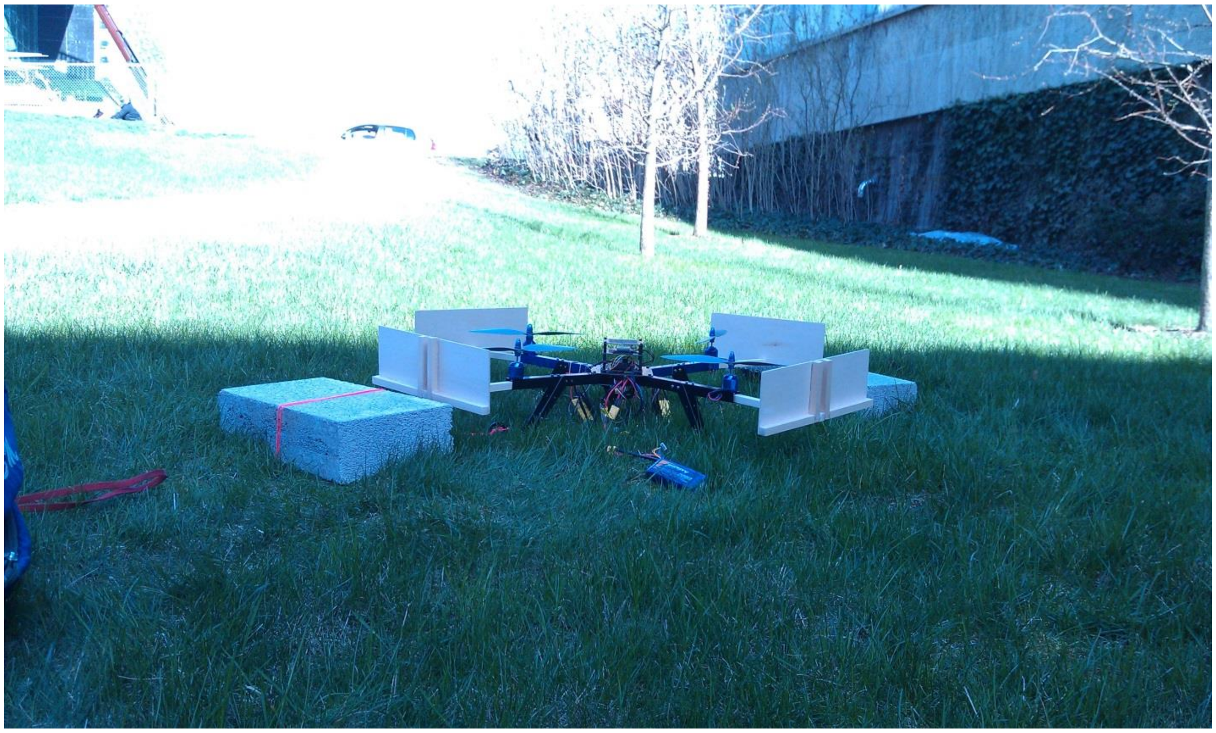


Figure 16: Quadcopter testing rig outdoors.

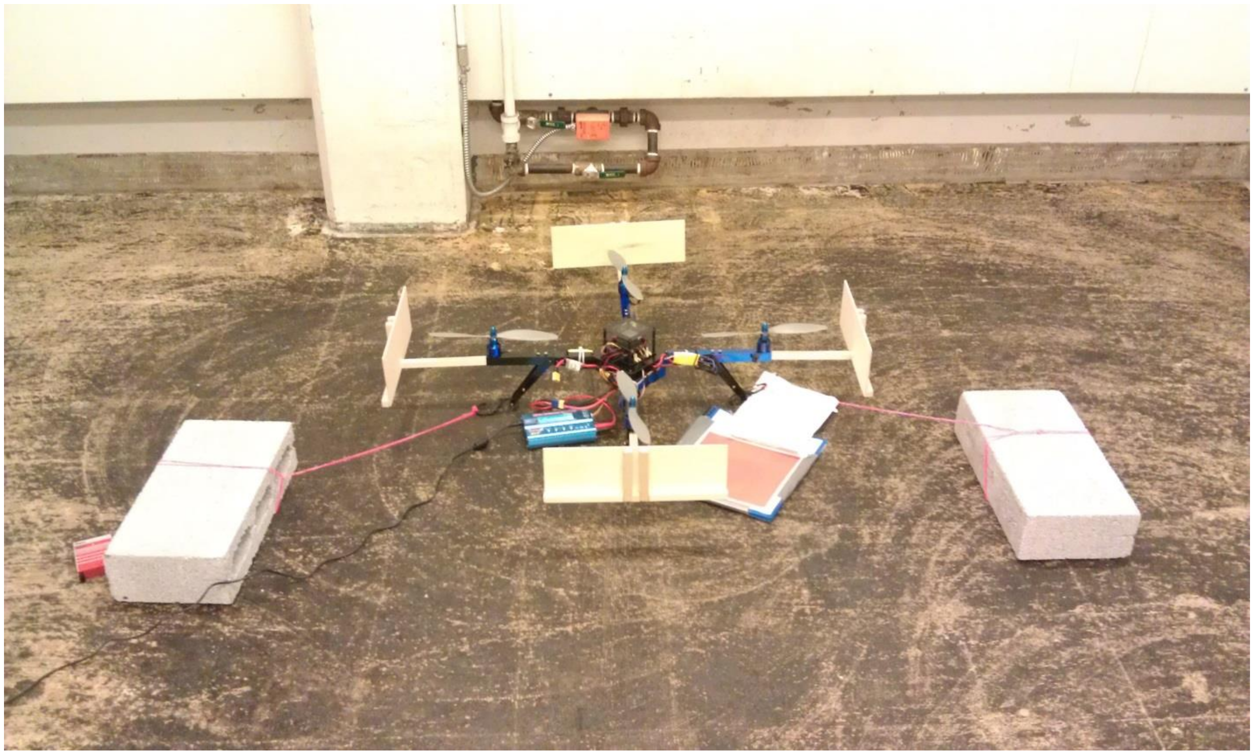


Figure 17: Quadcopter testing rig outdoors.

Appendices

Appendix I – Python Scripts Used

Android Camera module for detecting target: mavproxy_android.py

```
#Find Station using android camera
from SimpleCV import *
import time
import math
import thread
from time import sleep

mpstate = None

def cameraDetection():
    global mpstate
    disp = Display()

    lasttime = time.time()
    filename = "Timelog" + str(lasttime) + ".txt"
    f = open(filename, 'w')

    currenttime = time.time()
    f.write("Got to camera detection " + str(currenttime-lasttime) +"\n")
    lasttime = currenttime

    #cam = Camera(0)
    #OR
    #address = 'http://10.32.37.119:8080/videofeed'
    address = 'http://192.168.2.4:8080/videofeed'
    cam = JpegStreamCamera(address)
    print address

    currenttime = time.time()
    f.write("Camera stream started " + str(currenttime-lasttime) +"\n")
    lasttime = currenttime

    #f.close()

    xmin = 0 #310
    xmax = 640 #420
    ymin = 0 #160
    ymax = 480 #320
    blobmin = 2000 #2000
    while disp.isNotDone():
        override = mpstate.status.override[:]

        img = cam.getImage()
        if img != None:
            red_img_dist = img.colorDistance((255,0,0))
            bin_img = red_img_dist.binarize(100)
            blobs = bin_img.findBlobs()
            if blobs != None:
                biggestblob = blobs[-1]
                biggestblob.draw(color =(255,0,0))
                #print(biggestblob.area())

                currenttime = time.time()
                f.write("Fround red blob of size " + str(biggestblob.area())
+ " " + str(currenttime-lasttime) +"\n")
                lasttime = currenttime

                if biggestblob.area() > blobmin:
                    loc = biggestblob.centroid()
                    print("Found red blob located at" + str(loc))

                    currenttime = time.time()
                    f.write("Fround red blob located at " + str(loc) + " " +
str(currenttime-lasttime) +"\n")
                    lasttime = currenttime

                    if biggestblob.isRectangle(tolerance = 0.051):
                        print("Found a large red rectangle!")
                        #biggestblob.drawHull(color=(0,255,0), alpha=0)
                        distance = 12389*math.pow(biggestblob.length(),-
0.976)
                        print("The ground is " + str(distance) + " cm away")

                        currenttime = time.time()
                        f.write("Fround red rectangle, ground is " +
str(distance) + " cm away " + str(currenttime-lasttime) +"\n")
                        lasttime = currenttime

                    if (biggestblob.x>xmin) and (biggestblob.x<xmax):
                        if (biggestblob.y>ymin) and (biggestblob.y<ymax):
                            print("Blob is centered, attempting to LAND")
                            #StartAdd
                            override[4] = 1000
                            print("Got here to override")
                            mpstate.status.override = override
                            mpstate.override_period.force()

                            currenttime = time.time()
                            f.write("Target is centered, sending LAND command
" + str(currenttime-lasttime) +"\n")
                            lasttime = currenttime

                            #EndAdd
            bin_img.save(disp)
    f.close()
#StartAdd
def name():
    '''return module name'''
    return "android"

def description():
    '''return module description'''
    return "Android Camera Computer Vision"

def mavlink_packet(pkt):
    pass

def init(_mpstate):
    global mpstate
    print("Start Camera init")
    '''initialise module'''

    mpstate = _mpstate
    thread.start_new_thread(cameraDetection, ())
    # state = module_state()
    # mpstate.androidcam_state = state
    print("End camera init")
```

Modified controller code: mavproxy_joystick.py

```
#!/usr/bin/env python
'''joystick interface module

Contributed by AndrewF:
  http://diydrones.com/profile/AndrewF

'''

import pygame, fnmatch
from time import sleep

mpstate = None

class module_state(object):
    def __init__(self):
        self.js = None

'''
A map of joystick identifiers to channels and scalings.
Each joystick type can control 8 channels, each channel is defined
by its axis number, the multiplier and the additive offset
'''
joymap = {
    'CarolBox USB*':
    #
http://www.hobbyking.com/hobbyking/store/__13597__USB_Simulator_Cable_XTR_Aer
oFly_FMS.html
    # has 6 usable axes. This assumes mode 1
    [(3, 500, 1500),
     (0, 500, 1500),
     (1, 700, 1500),
     (4, 500, 1500),
     (5, 500, 1500),
     None,
     (2, 500, 1500),
     (5, 500, 1500)],

    'Sony PLAYSTATION(R)3 Controller':
    # only 4 axes usable. This assumes mode 1
    [(2, 500,  1500),
     (1, -500,  1500),
     (3, -1000, 1000),
     (0, -500,  1500)],

    #'Controller (XBOX 360 For Windows)':
    'Controller (Gamepad F310)':
    # only 4 axes usable. This assumes mode 1
    [(0, 500,  1500),  #CH 1 - roll
     (1, 500,  1500), #CH 2 - pitch
     (3, -1000, 1000), #CH 3 - throttle
     (2, -500,  1500)]  #CH 4 - yaw
    # [(2, 0,  1500),
     # (1, 0,  1500),
     # (3, -1000, 1000),
     # (0, 0,  1500)]
}

copter_axis = ["ROLL", "PITCH", "THROTTLE", "YAW", "MODE"]

def idle_task():
    '''called in idle time'''
    state = mpstate.joystick_state
    if state.js is None:
        return
    for e in pygame.event.get(): # iterate over event stack
        #the following is somewhat custom for the specific joystick model:
        override = mpstate.status.override[:]
        for i in range(len(state.map)):
            m = state.map[i]
            if m is None:
                continue
            (axis, mul, add) = m
            if axis >= state.num_axes:
                continue
            v = int(state.js.get_axis(axis)*mul + add)
            v = max(min(v, 2000), 900)
            print copter_axis[i] + ": " + str(v)
            override[i] = v

        if state.js.get_button(6):
            override[4] = 1000 #the back button
            print "SENDING LAND MODE"
        elif state.js.get_button(7):
            override[4] = 2000 #the start button
            print "SENDING STABILIZE MODE"
        elif state.js.get_button(3):
            override[4] = 1300 #yellow button Y
            print "SENDING ALT HOLD MODE"
        elif state.js.get_button(0):
            override[4] = 1400 #green button A
            print "SENDING LOITER MODE"
        if override != mpstate.status.override:
            mpstate.status.override = override
            mpstate.override_period.force()

def name():
    '''return module name'''
    return "joystick"

def description():
    '''return module description'''
    return "joystick aircraft control"

def mavlink_packet(pkt):
    pass

def init(_mpstate):
    '''initialise module'''
```

```
global mpstate
mpstate = _mpstate
state = module_state()
mpstate.joystick_state = state

#initialize joystick, if available
pygame.init()
pygame.joystick.init() # main joystick device system

for i in range(pygame.joystick.get_count()):
    print("Trying joystick %u" % i)
    try:
        j = pygame.joystick.Joystick(i)
        j.init() # init instance
        name = j.get_name()
        print 'joystick found: ' + name
        for jtype in joymap:
            if fnmatch.fnmatch(name, jtype):
                print "Matched type '%s'" % jtype
                print '%u axes available' % j.get_numaxes()
                state.js = j
                state.num_axes = j.get_numaxes()
                state.map = joymap[jtype]
                break
    except pygame.error:
        continue
```

Appendix II – Other Python Scripts

findRed.py – find any red items in view of the IP webcam and prints to the command prompt

```
#red example
from SimpleCV import *
import time
import math

disp = Display()
address = 'http://192.168.2.2:8080/videofeed'
print address
cam = JpegStreamCamera(address)

while disp.isNotDone():
    img = cam.getImage()
    red_img_dist = img.colorDistance((255,0,0))
    only_red = img - red_img_dist
    #img = img.scale(90, 90)

    RGB_only_red = only_red.meanColor()
    print(RGB_only_red)
    if RGB_only_red[2] > 15:
        print("Found RED!")
    img.save(disp)
```

cameraExample.py - QR code finding

```
from SimpleCV import JpegStreamCamera, Display
import time
import math
distance = 0
#visible distance seems to range from 400 to just over 50 pixels
disp = Display()
address = 'http://192.168.2.2:8080/videofeed'
print address
cam = JpegStreamCamera(address)

while disp.isNotDone():

    img = cam.getImage()
    barcodes = img.findBarcode()
    if barcodes != None:
        for b in barcodes:
            print("Barcode found!")
            distance = 12389*math.pow(b.length(),-0.976)
            print("The ground is " + str(distance) + " cm away." )
                      if (b.x>310) and (b.x<420):
                if (b.y>160) and (b.y<320):
                    print (b.length())
                    barcodes[0].show()
                    b.draw()
            img.save(disp)
```

Appendix III – Example Time log from android camera module

```
Got to camera detection 0.0
Camera stream started 0.00200009346008
Fround red blob of size 129310.5 4.64199995995
Fround red blob located at (422.08797171665617, 331.79111389510774) 0.000999927520752
Target is centered, sending LAND command 0.00600004196167
Fround red blob of size 220099.0 0.115000009537
Fround red blob located at (372.109560546239, 254.63202619427315) 0.0
Target is centered, sending LAND command 0.0090000629425
Fround red blob of size 238102.0 0.123999834061
Fround red blob located at (360.98825923343776, 244.6876562705619) 0.00100016593933
Target is centered, sending LAND command 0.00999999046326
Fround red blob of size 270070.5 0.123999834061
Fround red blob located at (340.30871630432296, 248.92323115630919) 0.0
Fround red rectangle, ground is 22.6741633209 cm away 0.00999999046326
Target is centered, sending LAND command 0.00100016593933
Fround red blob of size 259545.5 0.24599981308
Fround red blob located at (361.5463653450615, 237.53009202625358) 0.0
Target is centered, sending LAND command 0.00999999046326
Fround red blob of size 127128.0 0.0930001735687
Fround red blob located at (490.39389040966586, 211.97989034673714) 0.000999927520752
Target is centered, sending LAND command 0.010999917984
Fround red blob of size 48062.5 0.0759999752045
Fround red blob located at (494.65676983094926, 84.9879982661465) 0.0
Fround red rectangle, ground is 48.1382131232 cm away 0.00300002098083
Target is centered, sending LAND command 0.0
Fround red blob of size 48062.5 0.069000005722
Fround red blob located at (494.65676983094926, 84.9879982661465) 0.0
Fround red rectangle, ground is 48.1382131232 cm away 0.00300002098083
Target is centered, sending LAND command 0.0
Fround red blob of size 48062.5 0.0720000267029
Fround red blob located at (494.65676983094926, 84.9879982661465) 0.000999927520752
Fround red rectangle, ground is 48.1382131232 cm away 0.00200009346008
Target is centered, sending LAND command 0.0
Fround red blob of size 1062.0 445.447999954
Fround red blob of size 1062.0 0.0859999656677
Fround red blob of size 1062.0 0.0789999961853
Fround red blob of size 9917.5 2740.14700007
Fround red blob located at (551.9570960423493, 314.1480043693807) 0.000999927520752
Target is centered, sending LAND command 0.00100016593933
```

Works Cited

[1] S. Studios, "Crazyflie Nano Quadcopter Kit 10-DOF with Crazyradio (BC-CFK-02-A)," 2013. [Online]. Available: http://www.seeedstudio.com/depot/preorder-crazyflie-nano-quadcopter-kit-10dof-with-crazyradio-bccfk02a-p-1365.html. [Accessed May 2013].

[2] DIYDrones, "ArduCopter: Purchasing," DIYDrones, 30 December 2012. [Online]. Available: https://code.google.com/p/arducopter/wiki/Purchase. [Accessed May 2013].

[3] Vicon Motion Systems and Peak Performance Inc., "VICON," 2013. [Online]. Available: http://www.vicon.com/. [Accessed May 2013].

[4] Vicon Motion Systems and Peak Performance Inc., "University of Pennsylvania GRASP Lab Engineering Case Study," [Online]. Available: http://www.vicon.com/company/documents/UPENNJan13.pdf. [Accessed May 2013].

[5] Parallax Inc., "ELEV-8 Quadcopter Kit," 2013. [Online]. Available: http://www.parallax.com/Store/Robots/FlyingPlatforms/tabid/964/ProductID/799/List/0/Default.aspx?SortField=ProductName,ProductName. [Accessed May 2013].

[6] 3DRobotics Inc., "3DR ArduCopter Quad-C Frame + Optional electronics Kit," 3DRobotics Inc., 2013. [Online]. Available: http://store.3drobotics.com/products/3dr-arducopter-quad-c-frame-kit-1. [Accessed May 2013].

[7] DIY Drones, "Arducopter - The Full-Featured Multicopter UAV!," DIY Drones, [Online]. Available: https://code.google.com/p/arducopter/. [Accessed May 2013].

[8] C. Anderson, "DIY Drones - the leading community for personal UAVs," DIY Drones, 2013. [Online]. Available: http://diydrones.com/. [Accessed May 2013].

[9] Udrones Robotics Creations, " ArduCopter 3DR Quad D - Ready-to-Fly," 3D Robotics, 2013. [Online]. Available: http://www.udrones.com/product_p/acrtf2.htm. [Accessed May 2013].

[10] DIY Drones, "Mission Planner Utility," DIY Drones, 2012. [Online]. Available: https://code.google.com/p/arducopter/wiki/AC2_Mission. [Accessed May 2013].

[1 3DRobotics , "Spektrum DX7," 3DRobotics , 2013. [Online]. Available: http://store.3drobotics.com/products/spektrum-dx7s-7-ch-transmitter-with-ar8000.

1] [Accessed May 2013].

[1 QGroundControl, "MAVProxy," QGroundControl, [Online]. Available:
2] http://qgroundcontrol.org/mavlink/mavproxy_startpage. [Accessed May 2013].

[1 QGroundControl, "QGroundControl - Ground control station for small air/land/water
3] autonomous unmanned systems," QGroundControl, [Online]. Available:
http://qgroundcontrol.org/. [Accessed May 2013].

[1 QGroundControl, "MAVLink Micro Air Vehicle Communication Protocol," QGroundControl,
4] [Online]. Available: http://qgroundcontrol.org/mavlink/start. [Accessed May 2013].

[1 Mavproxy, "Mavproxy," [Online]. Available: https://github.com/tridge/MAVProxy. [Accessed
5] May 2013].

[1 L. Meier, "GitHub: Mavlink," 2012. [Online]. Available: https://github.com/mavlink/mavlink.
6] [Accessed May 2013].

[1 DIY Drones, "Using wireless data modules for telemetry and in-flight commands," 19
7] February 2013. [Online]. Available: https://code.google.com/p/arducopter/wiki/Telem.
[Accessed May 2013].

[1 CMUcam, "CMUcam: Open Source Programmable Embedded Color Vision Sensors,"
8] [Online]. Available: http://www.cmucam.org/. [Accessed May 2013].

[1 P. Khlebovich, "IP Webcam," Google Play Store, 1 January 2013. [Online]. Available:
9] https://play.google.com/store/apps/details?id=com.pas.webcam&hl=en. [Accessed May
2013].

[2 Sight Machine, Inc., "Simple CV: Computer Vision platform using Python.," Sight Machine,
0] Inc., [Online]. Available: http://simplecv.org/. [Accessed May 2013].