Letter of Transmittal

Group 1215
April 3, 2012

Jon Nakane
6224 Agricultural Road
V6T 1Z1
Vancouver, BC

Dear Mr. Jon Nakane,

This final report is for a self sponsored project to develop a quadrotor flying machine and onboard obstacle recognition feature.

This report will include background information on the quadrotor, the flight mechanics, description of our mechanical design, open source software that we utilise, the object recognition algorithm, the on board electronics components and testing results of the project.

We appreciate that you have taken your time to help us with this project and review our report. We hope this final report will meet your approval.

Sincerely,

Jia Ming Liang, Daniel L. Lu, Richard Lee

Quaffle
Design of an Autonomous Quadrotor

A scalable design based on commercial parts and rapid prototyping, featuring automatic object detection and flight control software.

Lee, Richard
Liang, Anson
Lu, Daniel Lawrence

Project 1215: ENPH 459
Engineering Physics Project Lab
University of British Columbia

April 3, 2012

Executive Summary

Our project is a self-sponsored project in which we construct an autonomous quadrotor flying vehicle that is capable of identifying and tracking targets. We present a new design for a low-cost quadrotor chassis that is modular and scalable. It can be created with current rapid-prototyping technologies such as the personal 3D printer and the water-jet cutter. To demonstrate the feasibility of constructing such a quadrotor, we have selected and procured several commercial off-the-shelf components including the motors, microprocessor, inertial measurement unit, battery, and rangefinder. The scale of the quadrotor is comparable to commercial remote-controlled aircraft, at about 50 cm wide. Using lightweight components such as carbon fibre chassis and lithium polymer battery, the quadrotor is light in weight and weighs 1.3 kg fully loaded.

The control systems requisite for stable flight are explored. The objective of autonomous flight is achieved by using computer algorithms to independently control the thrust magnitude of each motor depending on feedback from the inertial measurement unit. Due to the nature of the quadrotor design, this is sufficient for manoeuvring in all axes (hovering, vertical takeoff and landing, yaw, pitch, roll, etc) and requires no extra moving parts. The flight software is directly obtained from the AeroQuad project, an open-source software suite for remote-controlled quadrotors.

The project features the novel use of an onboard digital camera in conjunction with image processing software to detect and avoid obstacles. Because machine vision is computationally expensive, the processing is offloaded to an external computer. The downward-facing digital camera on the quadrotor chassis feeds a real time video feed wirelessly to an external computer which applies one of three modes of image processing and returns commands to the quadrotor via a Bluetooth connection.

The ultimate goal of fully autonomous flight wherein the quadrotor can automatically detect and follow targets is achieved partially: we were able to develop a working quadrotor prototype and working computer vision software, but were unable to integrate the two systems due to limited time.

Contents

List of Figures

List of Tables

1 Introduction

QUAFFLE is the project name for a new quadrotor unmanned aerial vehicle (UAV). Our project focuses on the complexity of developing such a machine and developing an onboard object recognition system that enables it to detect obstacles or search for targets.

1.1 Organisation of project report

This report will detail all the essential parts of the project that are necessary for developing a quadrotor and computer vision.

The development of this project will be separated into four main parts, flight control software, object recognition software, quadrotor mechanical design and the electronic components on the quadrotor. The flight control software section will include the description of how the quadrotor flight mechanics work and what is required for the software to control it. It will include a discussion on the Kalman filter algorithm that is used to filter noise in the electronic instruments and a description of the open-source AeroQuad software that we integrated to our quadrotor. The object recognition software part will include the main features we implemented and the algorithms behind the software. The quadrotor mechanical design section will include our design specification, choice of materials, and analysis on the mechanical system. The electronics section will justify our decisions in choosing the commercially available electrical parts and how we integrate all the electronics together in our quadrotor.

The section on the testing protocol will include tools we used for electronics test, calibration and PID tuning. It will also include section on and our indoor flight test. The results section will include the analysis on the design of the quadrotor and object recognition software. A description of the outcome of testing will be included in the results section, as well as a discussion on issues we have faced for this project. The most important results on the design of the quadrotor and object recognition software are summarised in the conclusion section.

The project deliverables section will include what will be delivered at the end of the project, financial summary, as well as ongoing commitment after the project is finished. Finally, the recommendation section will discuss potential further improvements to the Quaffle project.

1.2 Background and motivation

Quadrotors have been in existence ever since the 1920s when pioneers Etienne Omichen, George de Bothezat, and Ivan Jerome constructed the first quadrotors in history. However, the early attempts suffered from several drawbacks preventing widespread adoption, such as pilot workload and power-to-weight ratio. As such, for the most part of the 20th century they were relegated to only being a novelty with no practical use.

In the 21st century, quadrotors have seen a resurgence in popularity, primarily as small, unmanned remote-controlled aircraft for both military and civilian use. With the prolifera-

tion of affordable electronics such as digital cameras and computer microcontrollers, all the drawbacks that plagued the older designs are resolved. The concerns of pilot workload were solved by using a cheap computer microchip, such as the open-source Arduino, to finely control the thrust of each rotor; and the issue of power was solved by using efficient brushless electric motors on a chassis made from lightweight composite materials. Advances in computer vision technology also herald a new generation of autonomous quadrotors which may use image processing techniques to analyse their environment and make decisions on their own.

There are many advantages to using a quadrotor compared to other aircraft such as helicopters and fixed-wing aircraft. Compared with helicopters, a quadrotor has a simpler mechanical design since the rotors do not tilt and there is no need for the complex linkages and shafts in a helicopter swashplate - instead, a quadrotor can manoeuvre in all degrees of freedom simply by varying the thrust of each rotor. The quadrotor typically has counter-rotating propellers which cancel each other's angular momentum and angular thrust, negating the need for an additional tail rotor. Since there are four main rotors, a quadrotor's propellers each have less kinetic energy than a single helicopter rotor for the same total lift, reducing the chance of serious accidents or injury. Meanwhile, compared with fixed-wing aircraft, rotorcraft such as quadrotors have the obvious advantage of vertical take-off and landing, the ability to hover in one spot, and superior manoeuvrability in low-speed situations.

These advantages render unmanned quadrotors ideal for many purposes, such as surveillance, aerial photography, cargo transport.

1.3 State of the art

1.3.1 Flight control

Modern technology has enabled very affordable components and software for building quadrotors. Of the many attempts at creating a quadrotor, some projects are open-source and intended for a do-it-yourself (DIY) audience, such as Arducopter and AeroQuad. Arducopter is a family of autopilot multi-rotorcraft designed to be controlled by an Arduino microcontroller, with a wide array of features for UAV purposes including stabilised control, positioning hold using a global positioning system (GPS), altitude hold, automated takeoff and landing, waypoint programming, mounted camera stabilisation, telemetry display, and mission planning. AeroQuad is another project comprising of a software suite to control an Arduino quadrotor aircraft. Whilst AeroQuad does not support fully autonomous flight as Arducopter does, it implements several key functionalities such as stable hovering and control. Both AeroQuad and Arducopter are compatible with custom-designed quadrotors as long as they use a compatible Arduino microcontroller, sensors, and motors.

1.3.2 Obstacle detection

Using sensors such as a GPS, rangefinder, and barometer, a quadrotor is capable of positioning itself and perform simple autonomous tasks such as flying from one position at point A to

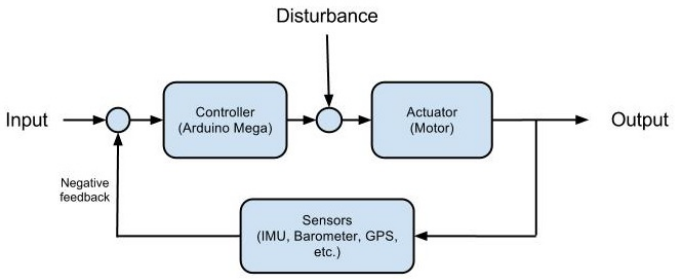

Figure 1: Block diagram illustrating control system of Quaffle.

another at point B. Assuming that the path between A and B is free of obstacles, a quadrotor may simply fly in a straight line. However, if obstacles exist between A and B, the task of navigating around them becomes nontrivial. Presently, there is much research being done in this area. At the University of Pennsylvania, Daniel Mallinger et al has created a motion-tracking system allowing indoor quadrotors to perform aggressive and accurate manoeuvres such as navigating around obstacles and landing at oblique angles [1]. He has also created a system for multiple quadrotors to coordinate together in tasks such as building a structure. Another similar project is at the Institute for Dynamic Systems and Control of the Swiss Federal Institute of Technology at Zurich, which likewise allows aggressive manoeuvres and coordination between multiple quadrotors [2].

However, these projects rely on external sensors, which severely limits the practical applications of the technology in outdoors situations. Hence, it is necessary to develop an onboard sensor system that allows the quadrotor to independently detect obstacles and automatically establish the correct flight path. Part of our research will focus on developing object-recognition technology that enables the quadrotor to detect obstacles using only a normal onboard digital camera.

1.4 Scope of project

The project Quaffle will involve two main parts: building a stable flying quadrotor and implementing an onboard sensor system that can detect obstacles. The stability of a flying quadrotor requires a complete control system with feedback (Figure 1), consisting of input, controller, actuator, sensors, disturbance, and output.

The input of this control system can be a radio remote controller or a wireless command from a computer. The controller used in Quaffle is an Arduino Mega processor. The actuator consists of four brushless electric motors which are each controlled by an independent electronic speed controller (ESC). The sensors consist of an inertial measurement unit, barometer, and rangefinder. A GPS module is not included in Quaffle, although we may add it in the future. The outputs of the control system are the quadrotor position and orientation, as well as their derivatives, including velocity, acceleration, angular velocity, and angular acceleration.

In order to recognize and trace targets, the software that we are developing needs to not

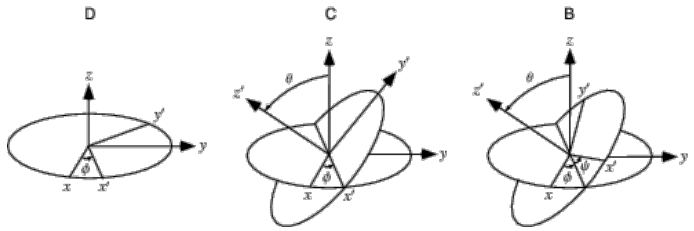

Figure 2: Coordinates for representing position and orientation of the quadrotor in three dimensions.

only recognize image fast since Quaffle will fly at a high speed, but also recognize object with a high accuracy. With the amount of time given, we chose to use Open Source Computer Vision (OpenCV) library, because it is free to use with many powerful features built in. Since OpenCV works best with C++ or C#, we chose C# to work with since C# has more APIs built in to work with than C++. During the development of this software, Visual Studio 2010 has been used since it allows us to debug in real time while implement the software.

2 Design of the quadrotor

2.1 Flight control software

2.1.1 Theory of flight control

A quadrotor can have movement in 6 degrees of freedom by just altering the speed of each of its four rotors. All four rotors provide lift to the quadrotor. Since the rotors are in two counter-rotating pairs, the angular momentum is cancelled in regular operation. The position and orientation of the quadrotor can be expressed using the generalised coordinate system:

$$\text{longitude}\, x, \text{latitude}\, y, \text{altitude}\, x, \text{roll}\, \theta, \text{yaw}\, \phi, \text{pitch}\, \psi, \tag{1}$$

where the tuple x, y, z represents the centre of mass position in Cartesian coordinates and θ, ϕ, and ψ are Euler angles that represent the orientation (Figure 2). To obtain position x and y we require a GPS module; for altitude z we can use a rangefinder (for low altitudes) or a barometer (for high altitudes). To obtain the Euler angles, we use the inertial measurement unit, which includes an accelerometer, gyrometer, and magnetometer.

To control the quadrotor we will first assume that the quadrotor has its centre of mass at its geometric centre. Three modes of flight operation [5] are illustrated in Figure 3. To hover or fly upwards, the rotors are simply powered on at the same speed. To yaw clockwise or counterclockwise, the rotor pair spinning in one direction will be set at a higher speed than the pair spinning in the opposite direction. To move horizontally, the quadrotor tilts towards its direction of motion by adjusting roll and pitch. When the quadrotor is thus tilted, the rotors provide also a horizontal component of force, enabling horizontal movement. Since the dynamics of rotor thrust is not linear (Section 2.4.4) and the mass distribution of the

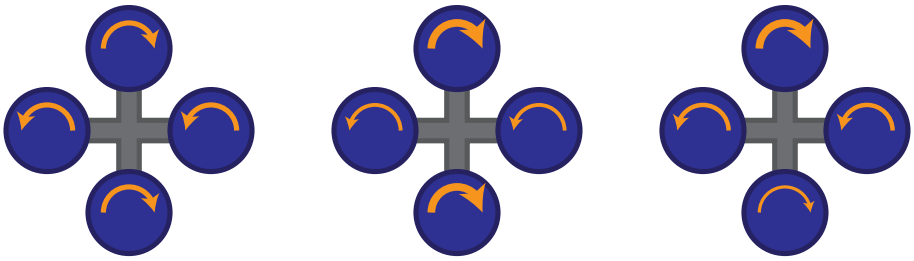

Figure 3: Modes of movement by adjusting rotor speed. From left to right: Hover/move up and down; Yaw right; Move forward (↓).

quadrotor and moment of inertia can be complex, we use PID control with proportional gain, integral gain, and derivative gain. The inertial measurement unit (Section 2.4.3) reports the acceleration, angular acceleration, and the orientation of the quadrotor.

2.1.2 Kalman filter

The inertial measurement unit is susceptible to noise due to mechanical vibrations, electrical and thermal noise, etc. Hence, using the IMU for feedback control requires some noise filtering to mitigate spurious readings. We employ a software Kalman filter for this purpose.

The Kalman filter is an iterative filtering algorithm that predicts the output using measurement history. It consists of two major steps when used for sampling data: the prediction step and correction step. The ultimate goal of these two steps are used for calculating a value known as the Kalman gain. The greater the Kalman gain, the more confidence will be placed on the measured value; conversely, with a smaller Kalman gain, the output will be weighted more towards the predicted result.

To speed up our development process, we utilise the Kalman filter library provided by AeroQuad. The generic implementation of a Kalman filter algorithm is as such[3]:

1. Read previous measurements/predictions.
2. Calculate predicted value $x(t) = Ax(t-1) + B \cdot u$.
3. Read new measured value.
4. Compute new measured value and predict value difference.
5. Calculate covariance.
6. Calculate the Kalman gain.
7. Correct prediction.
8. Calculate covariance of prediction error.

2.1.3 AeroQuad

The bulk of flight control implementation for Quaffle uses AeroQuad. AeroQuad is an open-source project for remote-controlled quadrotor development. One of the reasons why we chose AeroQuad is because it is very flexible for configuring the software to suit different systems. AeroQuad has an extensive feature list relevant to our needs [9]:

- Multiple flight configurations are supported:
  - Quad ×, Quad +, Quad Y4
  - Tri, Hex ×, Hex +, Hex Y6
  - Octo ×, Octo + and Octo ×8
- Multiple flight angle estimation algorithms supported:
  - DCM (best with magnetometer)
  - ARG (best with no magnetometer)
  - MARG (experimental)
- Flight options supported:
  - Heading hold with magnetometer or gyro
  - Altitude hold with barometer
  - Altitude hold with ultrasonic sensor (best for low altitude hold and terrain following)
- Enhanced battery monitoring options:
  - Enable auto descent
  - Specify battery cell count
  - Integration with On Screen Display (OSD)
- Multiple receiver options:
  - 6 or 8 channel receivers supported
  - PWM receivers
  - PPM receivers
  - PPM using hardware timer
- Telemetry options:
  - Wireless telemetry on dedicated serial port
  - OpenLog binary write

- Camera stabilization support:
  - Dedicated servo channels for roll, pitch, yaw
- Custom OSD support for MAX7456:
  - Specify video standard to use
  - Specify callsign to display
  - Built in attitude indicator
  - Display altitude in feet/meters
  - OSD system which allows remote PID tuning
- Advantages with the AeroQuad software architecture:
  - Easier troubleshooting of hardware with individual tests programs for each sensor.
  - Updated task scheduler to clearly define timing of critical tasks
  - Sensor classes are separated into their own libraries. If you want to add a new sensor, add a library based on
  - the defined functions for seamless integration into the flight software
  - Flight control functions are also separated into their own libraries with well defined interfaces and properties such that if new features are added or enhanced, there will be minimal impact to the overall Flight Software performance.

We use the following AeroQuad configuration for Quaffle:

- Quad+
- DCM (with magnetometer)
- Altitude hold with barometer
- 6 channel receivers support

AeroQuad also provides a graphical user interface on PC to help the user configure the quadrotor, calibrate sensors, and tune PID values. With this feature, we are able to quickly set up our robot to test flight after putting together all mechanical and electrical components.

2.2 Object recognition software

2.2.1 Overview of object recognition

Object recognition is instrumental in Quaffle to enable to detect and avoid obstacles as well as interact with targets. We use the OpenCV library, an open-source image processing library to implement computer vision in Quaffle. The object recognition is offloaded from

Quaffle onto a personal computer such as a laptop, using a wireless digital video camera mounted on Quaffle that is capable of transmitting video in real time.

Object recognition programs typically store certain templates (slave images), of items that it is intended to detect, and attempt to detect and locate these items in an input image frame (master images). The object recognition software in Quaffle takes real time images from the digital video camera as input and process each frame using computer vision algorithms to determine whether or not the current frame contains the object it is attempting to detect. The output, if the algorithm succeeds in detecting the object, is simply the coordinate pair x, y indicating the position of the object in the frame.

Three modes of object recognition have been developed: a general purpose object recognition algorithm for finding any pattern using speeded up robust feature (SURF) and scale invariant feature transform (SIFT) method; facial recognition using Haar-like features (HLF); and laser tracking using brightness differences. The basic software flow is described in Figure 4 and each of the three modes of object recognition will be separately discussed and analysed in Sections 2.2.3, 2.2.4, and 2.2.5. The user may choose any one of the modes to use.

2.2.2 Image capture from camera

To process the real time video, individual frames must be captured from the camera. In Figure 5, the code snippet for handling this is shown. Once the start camera button is clicked, a new instance of the `Capture` object is created, if it has not already been created. The class `Capture` is implemented in a built-in library within `EMGU.CV.Util` that captures images from any camera and video source.

Once the camera is started, an internal timer that is bundled to the camera is also started and set to tick once every 40 ms. In other words, the camera will take a picture every 40 ms, supplying images at a framerate of 25 fps. The framerate is chosen to be as high as possible without overloading the processor due to the limitations in computing power available to us. A lower framerate is not desired since it will cause the system to less responsive.

At the instance the image is taken, the algorithm will try to match the image to the master image that is previously stored in the program with any of the three methods that will be described in Sections 2.2.3, 2.2.4, and 2.2.5.

2.2.3 Object recognition using SIFT and SURF

A major library in OpenCV is the SIFT library, which stands for scale invariant feature transform. For any object in the image, interesting points on the object can be extracted to provide a good feature description of the object. These points, extracted from the master image, can then be used to identify the same object when it is in a test image containing many other objects. Given any master image, this method is able to detect, match, and describe local features that are similar in the master and slave images. The SIFT algorithm stores the interesting features of the master image and those of the slave image using a matrix for each. This allows transformations on these features to be done by means of matrix operations. Operations such as scaling, transposition, and rotation can be performed easily - hence the

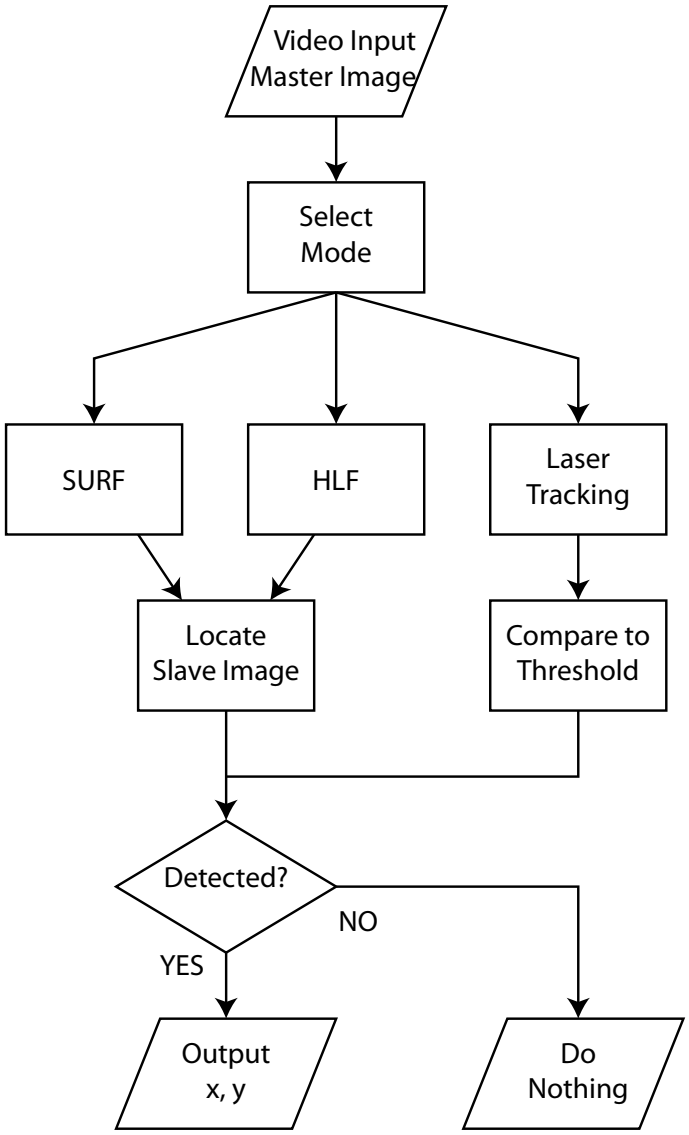

Figure 4: Flowchart illustrating the procedure for object recognition.

```
private void btnStartCam_Click(object sender, EventArgs e)
{
        try
        {
                #region if capture is not created, create it now
                if (_capture == null)
                {
                        try
                        {
                                _capture = new Capture();
                        }
                        catch (NullReferenceException excpt)
                        {
                                MessageBox.Show(excpt.Message);
                        }
                }
                #endregion
```

Figure 5: Code snippet for fetching current frame from video feed.

SIFT has the desirable quality of being scale and rotation invariant, meaning that it can successfully detect objects even when they are rotated and scaled to different sizes.

Prior to implementing real time object recognition, we developed a simple program to use the SIFT method on still images (Figure 6). Testing with various images showed that the algorithm performs with great accuracy and precision, albeit with long computation time. In this the test shown in the figure, the slave image was simply cropped from the master image and rotated. As seen in Figure 6, 1473 ms elapsed to locate the slave image (top right) in the master image. The slow speed of the algorithm makes it unsuitable for real time object recognition that is required for a quadrotor, hence, instead of SIFT, we use the SURF method.

The speeded up robust feature (SURF) method is extremely similar to SIFT, but is about ten times faster as its name implies. For a typical 1080p (1920 $\times$ 1080 pixels), the software is able to detect and recognise the object within 30 ms on average. In Figure 7 the code for implementing object recognition using SURF is shown. As seen in Figure 7, the software stores interesting points in the variable `observedKeyPoints`, which are then mapped to a matrix named `observedDescriptors`. To match the two matrices, we employ a built-in brute force matching object in OpenCV called `BruteForceMatcher`. Finally, `Features2DTracker` is used to confirm thet uniqueness of the interesting points found. Notice also that Figure 7 includes a stopwatch to measure the speed of the algorithm. The final time will be displayed on the graphical user interface (GUI) for the convenience of the user. In addition, if the computing time is longer than 80 ms (which is double the duration for the video frame as explained in Section 2.2.2), the current image recognition process will be aborted to conserve computational resources. The reason is that, if all of the images take more than 80 ms to process, the program would become unresponsive.

Once the object is found, the position of the found object will be returned, and it will be drawn on the GUI to be displayed to the user. In the demo program, the user may train the software on the object to recognise by simply drawing on the master image displayed

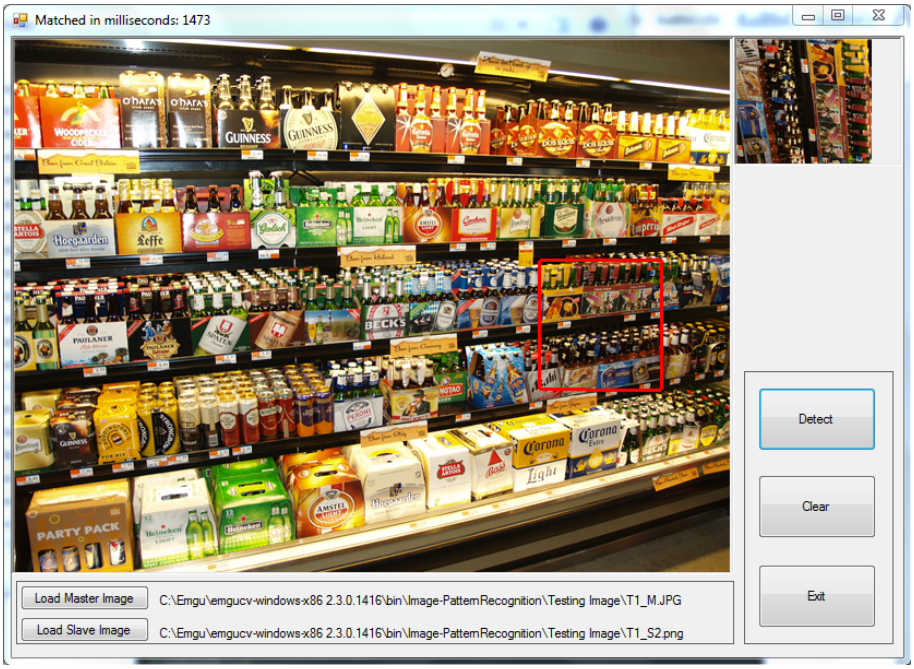

Figure 6: Screen capture showing the demo program for object recognition using the SIFT method.

in the GUI to select the region of the camera frame to save as the slave image. As seen in Figure 8, the software is able to recognise four different objects simultaneously with less than two-thirds CPU usage on an Intel Core 2 Duo computer.

2.2.4 Facial recognition using HLF

The method of Haar-like features HLF is a pattern recognition pattern designed specifically for facial recognition. A Haar-like feature takes adjacent rectangular regions at a specific location on an image, and computes the sum of pixel intensities in that region and calculate the differences between them. These differences will then be sued to categorise the subsection, which is then itself subdivided into smaller subsections. The recursive operation of the algorithm allows it to be very fast. The HLF method is specifically written for facial recognition because it is well known that the darkest region on a human face is the eyes and the brightest region is the cheek area. A common Haar-like feature for generic facial detection would put two rectangular boxes above the eye and cheek area. The total triangle made by these three boxes would represent the area of the human face.

However, facial detection is not the same as facial recognition, since it cannot distinguish between different faces. Fortunately, it is also possible to use the Haar method to recognise different faces. The program stores an image of a known face to a database, and then invokes a neural network built into OpenCV in a separate thread to start extracting information about the new face that the program has just learnt (e.g., the ratio of the distance between the two eyes to the distance between the eyes and nose). All this information is then stored

```
watch = Stopwatch.StartNew();

// extract features from the observed image
observedKeyPoints = surfCPU.DetectKeyPointsRaw(observedImage, null);
Matrix<float> observedDescriptors = surfCPU.ComputeDescriptorsRaw(observedImage, null,
    observedKeyPoints);

BruteForceMatcher matcher = new BruteForceMatcher(BruteForceMatcher.DistanceType.L2F32);
matcher.Add(modelDescriptors);
int k = 2;
indices = new Matrix<int>(observedDescriptors.Rows, k);
dist = new Matrix<float>(observedDescriptors.Rows, k);
matcher.KnnMatch(observedDescriptors, indices, dist, k, null);

mask = new Matrix<byte>(dist.Rows, 1);

mask.SetValue(255);

Features2DTracker.VoteForUniqueness(dist, 0.8, mask);

int nonZeroCount = CvInvoke.cvCountNonZero(mask);
if (nonZeroCount >= 4)
{
        nonZeroCount = Features2DTracker.VoteForSizeAndOrientation(modelKeyPoints,
            observedKeyPoints, indices, mask, 1.5, 20);
        if (nonZeroCount >= 4)
                homography = Features2DTracker.GetHomographyMatrixFromMatchedFeatures(
                    modelKeyPoints, observedKeyPoints, indices, mask, 3);
}

watch.Stop();
```

Figure 7: Code snippet for SURF method.

Figure 8: Screen capture showing the demo program for object recognition using the SURF method. Two different objects are being detected and tracked in real time simultaneously.

```
Stopwatch watch;
//String faceFileName = "haarcascade_frontalface_default.xml";
String faceFileName = "haarcascade_frontalface_alt.xml";
String eyeFileName = "haarcascade_eye.xml";

//Read the HaarCascade objects
using (HaarCascade face = new HaarCascade(faceFileName))
using (HaarCascade eye = new HaarCascade(eyeFileName))
{
        watch = Stopwatch.StartNew();
        using (Image<Gray, Byte> gray = image.Convert<Gray, Byte>()) //Convert it to
            Grayscale
        {
                //normalizes brightness and increases contrast of the image
                gray._EqualizeHist();

                //Detect the faces  from the gray scale image and store the locations as
                    rectangle
                //The first dimensional is the channel
                //The second dimension is the index of the rectangle in the specific channel
                facesDetected = face.Detect(
                        gray,
                        1.1,
                        10,
                        Emgu.CV.CvEnum.HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
                        new Size(20, 20));
        }
        watch.Stop();
}
```

Figure 9: Code snippet for HLF method.

in an XML file for future use.

A code snippet for using HLF to detect faces is shown in Figure 9. As can be seen, the generic face template `haarcascade_frontface_alt.xml` is loaded before using the Haar's face method. Once the database is loaded, the program has all the information for HLF to recognise a human face. A `HaarCascade` object is initialised twice for face and eye. The built-in function from OpenCV library `Detect` is used to check if any face is detected in the current captured frame. Two parameters can be set to tune the speed and accuracy of the face detection algorithm. The `Detect` function takes in as arguments the colour of the image, accuracy of detection, the number of times to iteratively run face detection, the detection type, and size of image stored. The value we are using for accuracy is 1.1, which allows the program to yield the correct result 70% of the time. The accuracy is not very high, but to increase the accuracy would require extra computational power. In order to increase the accuracy of face detection, we run it ten times for each image. This improves the consistency of recognising the same face correctly. Once a face is drawn, the location of the face is returned and a rectangle is drawn in the GUI outlining the face (see Figure 10, showing two faces being recognised concurrently with just over half of CPU usage).

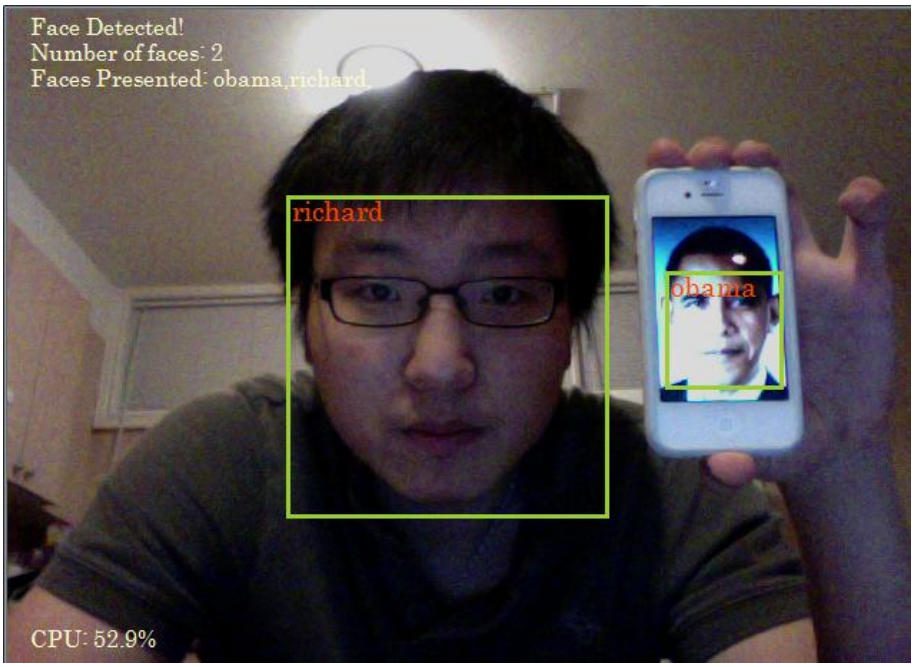

Figure 10: Screen capture showing the demo program for facial recognition using the HLF method. Two different faces are being detected and tracked in real time simultaneously.

2.2.5 Laser tracking

Laser tracking is a mode whereby the software can follow a laser beam shining on any surface. The goal for laser tracking is to allow Quaffle to track any target on the ground simply by shining a laser on it. This useful for two reasons: first, a human operator can instruct Quaffle to follow a target by pointing a laser beam at it; second, we can use this to debug Quaffle's ability to follow targets.

2.2.6 Integration with flight control

So far in Sections 2.2.3, 2.2.4, and 2.2.5 we have described how the various computer vision algorithms identify and locate the target as a set of coordinates on the camera image frame. To connect this with the quadrotor Quaffle, we implement a serial class to communicate with the Arduino Mega microcontroller on Quaffle. Quaffle is equipped with a Bluetooth shield that enables communication with other Bluetooth-enabled devices such as a laptop computer. The serial class is initiated on program startup with a user-specified COM port for communicating with Quaffle. The location of the recognised image is then sent in real time to Quaffle in a specific format: The x and y coordinates are sent as two integers representing the percentage ratio they are to the width and height of the image respectively. In Quaffle, the Arduino board may then process the information and use them as needed.

```
public void LaserTracking()
{
        bool brightnessFound = false;
        float brightest = 0;
        int xPos = 0, yPos = 0;

        for (int y = 0; y < curCameraBM.Height; y += 5)
        {
                for (int x = 0; x < curCameraBM.Width; x += 5)
                {
                        byte red, green, blue;
                        red = curCameraBM.GetPixel(x, y).R;
                        green = curCameraBM.GetPixel(x, y).G;
                        blue = curCameraBM.GetPixel(x, y).B;

                        float brightness = (299 * red + 587 * green + 114 * blue) / 1000;

                        if (brightness > threshold)
                        {
                                if (brightness > brightest)
                                {
                                        brightest = brightness;
                                        xPos = x;
                                        yPos = y;
                                        brightnessFound = true;
                                }
                        } // (brightness > _mForm.threshold)
                } // x loop
        } // y loop
}
```

Figure 11: Code snippet for laser tracking.

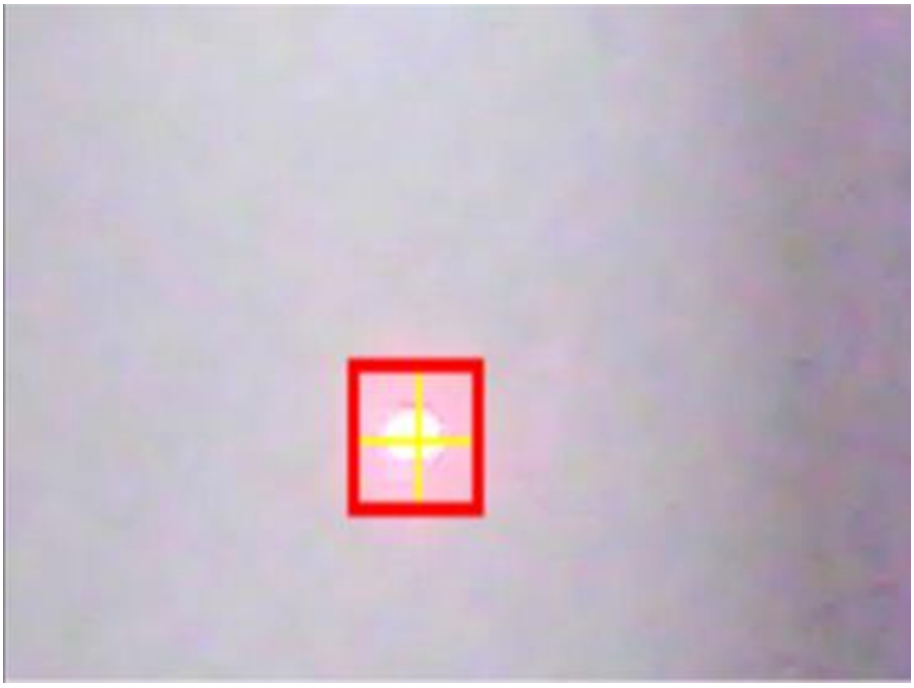

Figure 12: Screen capture showing the successful location of laser beam.

```
Point middle = new Point(0, 0);
string serialOut = "";
foreach (MCvAvgComp f in facesDetected)
{
        middle.X = f.rect.Location.X + f.rect.Width / 2;
        middle.Y = f.rect.Location.Y + f.rect.Height / 2;

        double Xprop = (100 * middle.X / curCameraBM.Size.Width) / 10;
        double Yprop = (100 * middle.Y / curCameraBM.Size.Height) / 10;
        //serialOut = "x" + Xprop.ToString("N0") + "y" + Yprop.ToString("N0");

        serial.Write(serialOut);
}
```

Figure 13: Code snippet for serial communication from object recognition software to Quaffle.

2.2.7 Multithreading

Computational performance is of utmost importance to computer vision software since image processing is highly computationally intensive, but computer vision requires real time performance. The HLF method, for example, can take up to 500 ms to process a single frame, and the resulting 2 fps frame rate is far too slow for successful implementation in a flying machine. Fortunately, there are ways to improve the performance of the program by multithreading. The latest version of the software launches a new thread every time a frame is taken, so that if the algorithm gets stuck analysing a certain frame, then subsequent frames are not affected. Tests on an Intel Core 2 Duo laptop indicate that the program is able to recognise four images simultaneously with no significant amount of delay.

With multithreading, synchronisation can be an issue. Since 25 threads are created per second, each thread needs to fetch the resource of the master image at different times. A central manager algorithm has been developed to manage each thread to process in order. Moreover, a synchronous program method has been adopted, so that when one thread is accessing some resource, the resource is locked and only available to that particular thread. If another thread attempts to access the same resource, a copy of that resource will be made to be used on the second thread.

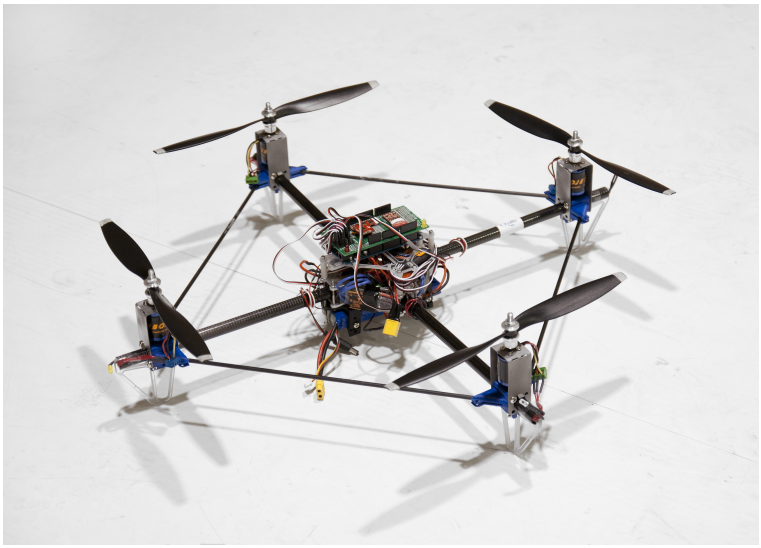

Figure 14: Photograph of completed Quaffle prototype.

2.3 Mechanical design

2.3.1 Overview of design

The chassis of Quaffle is designed to be rigid but lightweight. It consists primarily of carbon fibre tubes connected at the corners by 3D printed plastic parts and at the centre by polycarbonate and aluminium components fabricated using a water-jet cutter. The design emphasizes modularity and scalability as well as low cost and ease of fabrication and assembly. The size of the quadrotor can be easily changed by using carbon fibre tubes of different lengths, and the central assembly can be expanded to install additional instruments simply by adding more layers on top.

2.3.2 Carbon fibre tubes

Carbon fibre tubes were selected because of their strength and light weight. Quaffle uses two different types of circular carbon fibre tubes: a thicker tube for the main quadrotor arms and four thinner auxiliary tubes connecting them to each other. The four arms are constructed using cellophane-wrapped unidirectional carbon fibre tubes with an outer diameter of 14.04 mm and an inner diameter of 12.45 mm. In between each pair of two adjacent arms, a thinner unidirectional carbon fibre tube (diameter 4.76 mm) connects them near the end (Figure 16. The purpose of these tubes is twofold: first, when combined with the quadrotor corner part in Section 2.3.3, they ensure that the motors point upwards since otherwise there is no easy way to do so given the circular cross-sectional profile of the arms; second, they add strength to the entire chassis and mitigate low-frequency vibrations.

Since the corner components (motor and landing gear) are not connected to the central assembly except through the carbon fibre tubes, adjusting the lengths of the carbon fibre tubes allows one to change the size of the entire quadrotor without having to redesign the

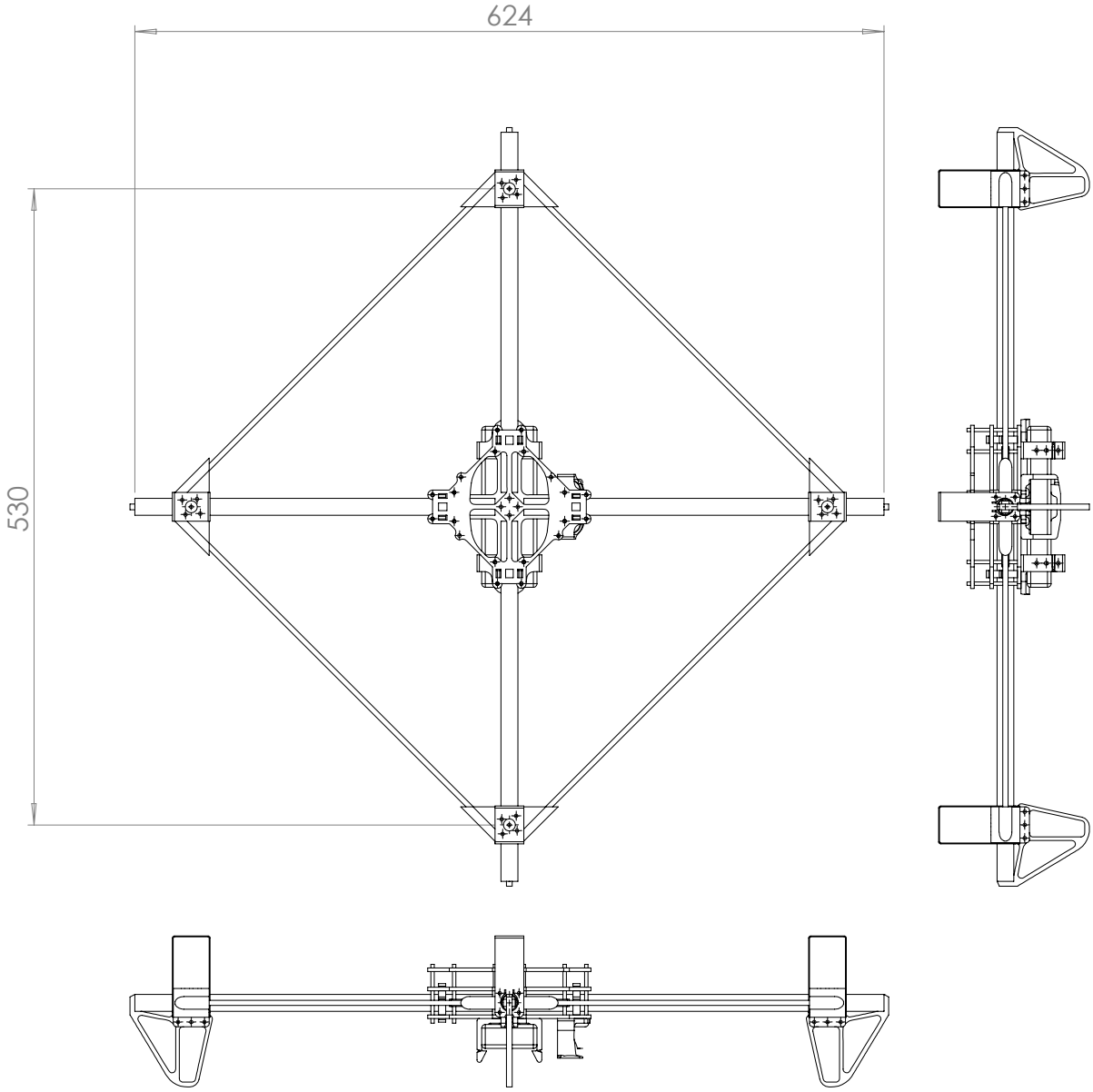

Figure 15: Schematic of Quaffle chassis not including motors, rotors, and electrical components. Units are in millimetres.

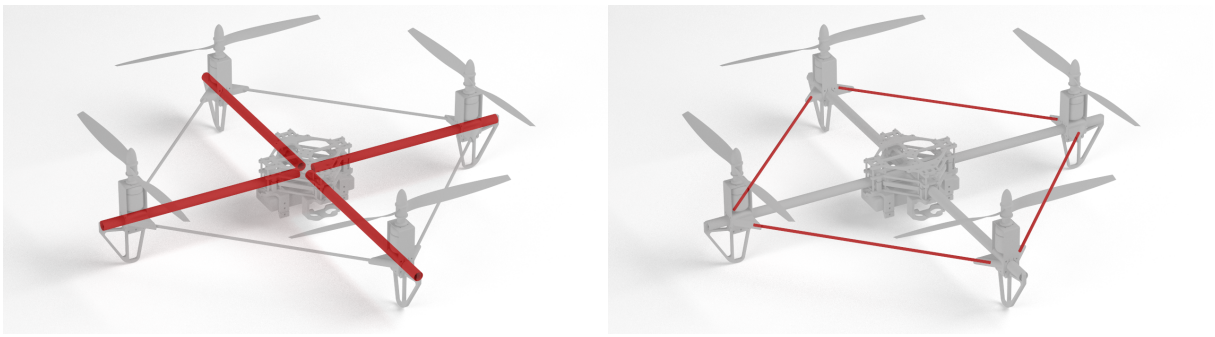

Figure 16: Diagram indicating the position of carbon fibre tubes on the Quaffle chassis. Highlighted in red are the main arms (left) and the auxiliary tubes (right).

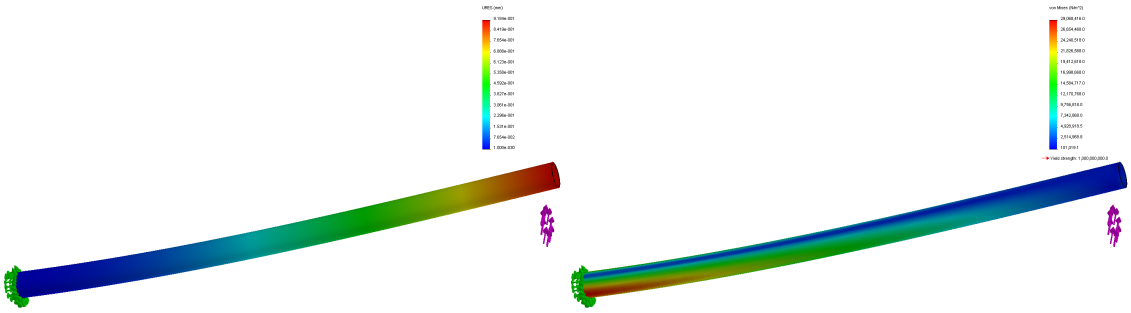

Figure 17: Results of finite-element analysis of carbon-fibre tube used in main arm. For an applied force of 10 N, the maximum deformation (left) is approximately 0.9 mm and the maximum Von Mises stress (right) is 29 MPa. The deformation scale here in this figure is approximately 32.68. The material properties are listed in Table 1. The simulation was done using Solidworks 2012.

other components. For our project, the desired diameter of Quaffle was approximately 60 cm excluding rotors; hence, we used the length of 300 mm for the four main arms (Figure 15). The thinner auxiliary carbon fibre tubes connect the corners diagonally and, by geometry, are approximately $\sqrt{2}$ times the length of the main arm. However, in reality our design allows the arms to extend slightly beyond the motors and the quadrotor corner part (Section 2.3.3) also extends sideways, hence the length of the auxiliary tubes used is 366 mm. In general, the relationship between the lengths of the carbon fibre tubes can be described in the following way:

$$l = \sqrt{2}L - 58\,\text{mm} \tag{2}$$

where L is the length of the main arms and l is the length of the auxiliary tubes.

The material properties of the unidirectional carbon fibre tube in Table 1 means that the 14.04 mm carbon fibre tube used in the main arm of the quadrotor is more than strong enough for our purposes. As seen in Figure 17, under 10 N of load, the arm does not deform more than 0.9 mm and the maximum Von Mises stress is 29 MPa, giving a factor of safety

Table 1: Mechanical properties of unidirectional carbon fibre tubes [4].

Property	**Value**
Density	1.60 g/cc
Young's modulus (axial)	135 GPa
Young's modulus (transverse)	1 GPa
Ult. tensile strength (axial)	1500 MPa
Ult. compressive strength (axial)	1200 MPa
Ult. tensile strength (transverse)	50 MPa
Ult. compressive strength (transverse)	250 MPa

of more than 30. The motors are not capable of producing more than 10 N of thrust, so the design is strong enough. Furthermore, this simulation did not take into account the contribution from the auxiliary tubes, so it is likely to underestimate the strength of the design.

2.3.3 3D-printed components

Quaffle features several parts fabricated by a 3D printer. 3D printing is both affordable and capable of very complex parts which would otherwise require specialised machine tools or assemblies of multiple components. The Engineering Physics Project Lab owns an Up! personal portable 3D printer that is capable of printing in layers of 0.20 mm thick (Figure 19). Using 3D printer technology, we were able to create a part hereafter known as the quadrotor corner. Each of the four quadrotor corners joins the its corresponding main arm with the adjacent auxiliary carbon fibre tubes and provides a position to securely install the motor mount and landing gear (Figure 18). A more detailed schematic of the quadrotor corner is shown in Figure 20.

In addition to the quadrotor corner, Quaffle also contains 3D-printed parts to hold the battery and to hold the rangefinder (Figure 21). These parts attach to the underside of the polycarbonate central assembly (see Section 2.3.4). The battery holder comes in two identical parts to hold the battery at both ends. Each of the two parts extends slightly beyond the battery so that a screw may be used to prevent the battery from sliding out at each end. However, the battery may be easily removed simply by removing one of the said screws and sliding it out.

The rangefinder holder is a single part that is designed to fit the rangefinder. Two large holes in the underside of the rangefinder holder are designed to accommodate the emitter and receiver of the ultrasonic rangefinder.

2.3.4 Polycarbonate components

Polycarbonate parts play an important role in the chassis of Quaffle. The central assembly consists of stacked layers of 3 mm thick polycarbonate cut using a water-jet cutter 22. Of these, the bottom two layers firmly secure the carbon fibre main arms of the quadrotor to

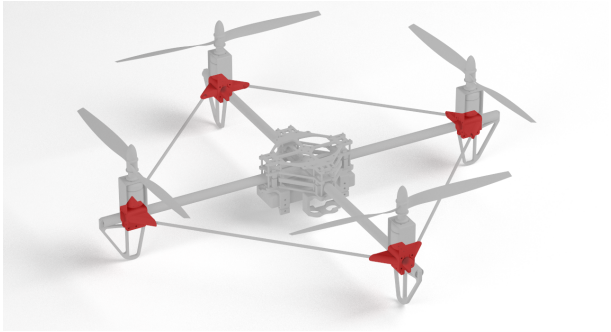

Figure 18: Diagram indicating the position of 3D-printed quadrotor corners on the Quaffle chassis, highlighted in red.

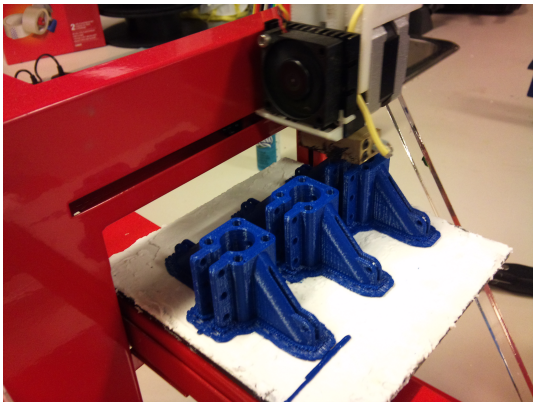

Figure 19: Photograph of three quadrotor corners being printed using the Up! PP3DP.

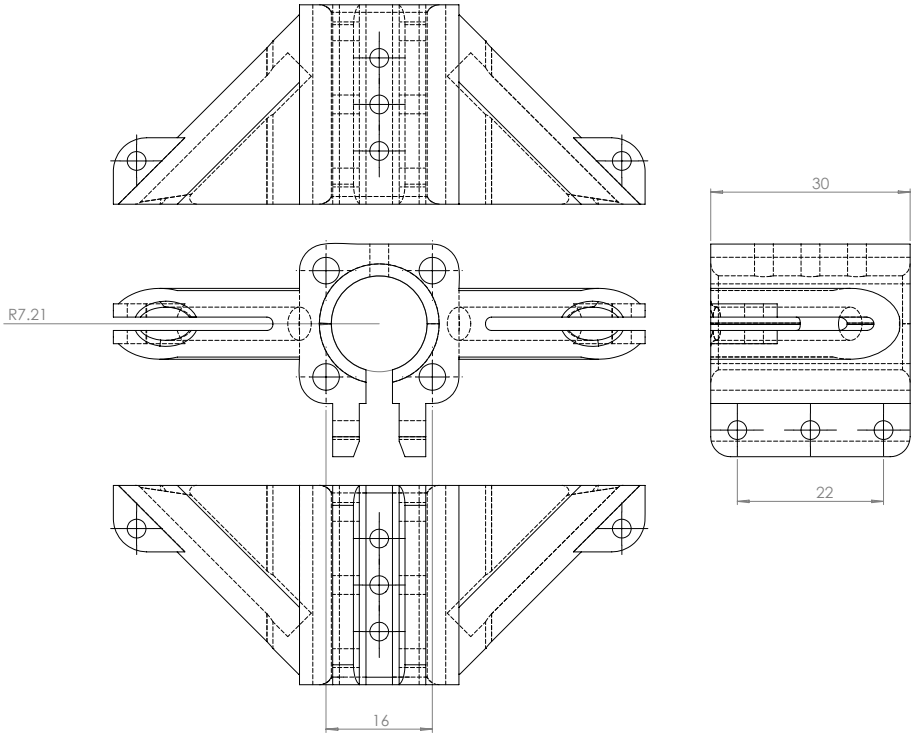

Figure 20: Schematic of quadrotor corner. Units are in millimetres.

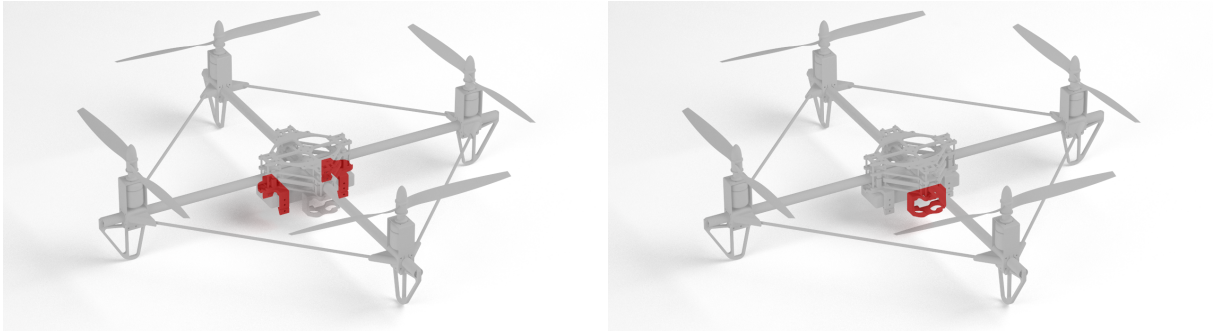

Figure 21: Diagram indicating the position of battery holders (left) and rangefinder holder (right) on the Quaffle chassis, highlighted in red.

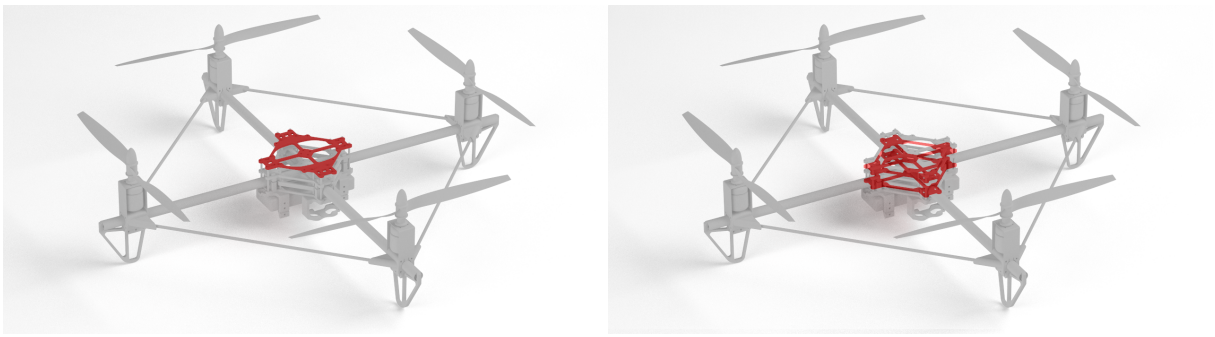

Figure 22: Diagram indicating the position of Arduino Mega mounting platform (left) and polycarbonate portion of central assembly (right) on the Quaffle chassis, highlighted in red.

an aluminium core, providing a solid platform on which instruments and electronics can be installed. The two layers secure the carbon fibre arms by sandwiching it with semicircular clamps (Figure 23). Although the unidirectional carbon fibre tube is very resilient to axial loads, it is weak transversely (Table 1); but the semicircular clamps, combined with the relatively flexible nature of polycarbonate allow the forces to be distributed evenly, thereby securing the tubes without damaging them.

An upper layer is designed to fit the Arduino Mega 2560 microcontroller. All the sensors and instrumentation required for Quaffle are installed on the AeroQuad shield which is directly mounted on the Arduino Mega board. Since this layer does not carry significant load, polycarbonate was chosen for its light weight and aesthetic appeal. The design of this layer is extremely similar to the other two layers in the central assembly. If needed, additional layers similar to this can be stacked on top or on the bottom for extra instrumentation. There is no limit to adding extra layers on top insofar as it remains a reasonable amount and does not interfere with flight control mechanics. Adding extra layers on the bottom is very limited due to the length of the landing gear and care must be taken so that the battery does not hit the ground.

The space between the upper layer and the lower two layers can contain additional batteries for powering specific electronic devices. The length of the edge of the lower layers is slightly longer than the electronic speed controller, therefore allowing a snug fit when the speed controller is attached to the side of the central assembly using tape.

The landing gear is also created using polycarbonate using the water-jet cutter (Figure 24). Polycarbonate was chosen for the landing gear because it is light, is resistant to impacts (unlike acrylic, which is brittle and tends to shatter), is sufficiently soft to avoid damaging ground surfaces, and has the aesthetic appeal of being transparent, thereby creating the illusion of hovering even when sitting on the ground.

2.3.5 Metal components

While the use of metal is to be avoided in general to minimise weight, Quaffle does feature some metal components. Notably, the crux of the central assembly is machined out of 4.8

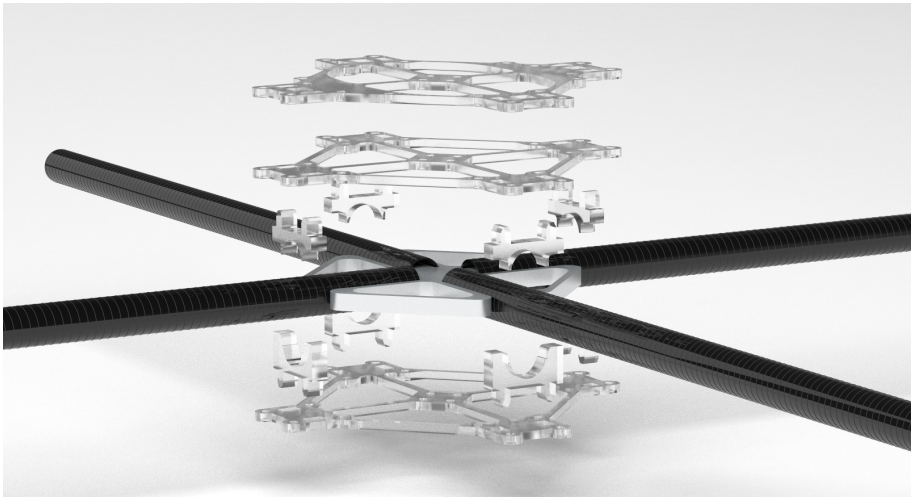

Figure 23: Render showing close-up exploded view of central assembly. The three polycarbonate layers from Figure 22 are clearly shown, as well as the semicircular clamps used to secure the carbon fibre tubes.

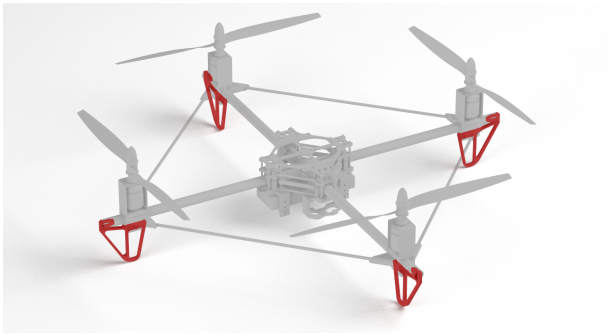

Figure 24: Diagram indicating the position of the landing gear on the Quaffle chassis, highlighted in red.

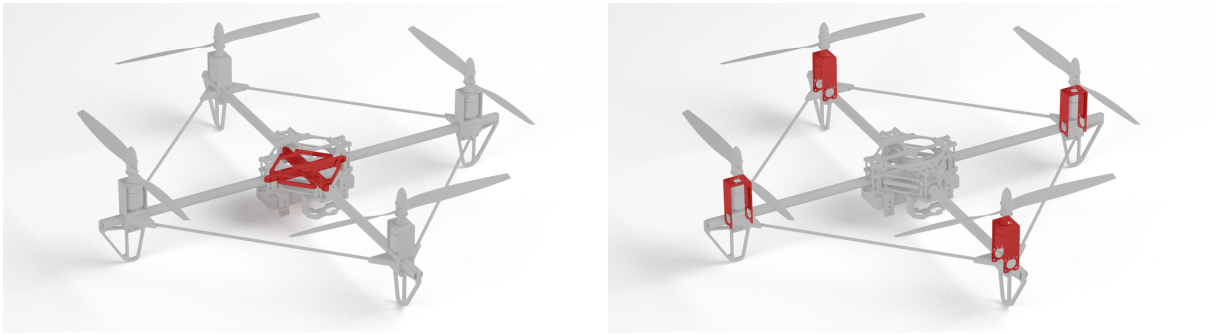

Figure 25: Diagram indicating the position of central aluminium piece (left) and motor mounts (right) on the Quaffle chassis, highlighted in red.

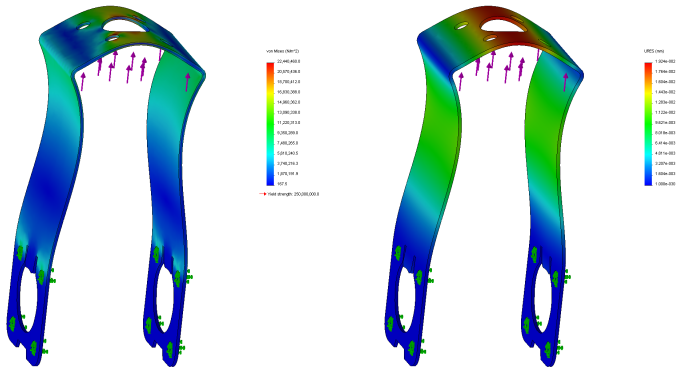

Figure 26: Results of finite-element analysis of motor mount under axial load. For an applied force of 10 N, the maximum deformation (left) is approximately 0.01 mm and the maximum Von Mises stress (right) is 22 MPa. The deformation scale here in this figure is approximately 352. The material is mild steel. The simulation was done using Solidworks 2012.

mm sheet aluminium using the water-jet cutter. Obviously, it is very important for the core of the aircraft to be as solid as possible or else each arm may wobble. The four carbon fibre arms are secured to this aluminium piece using polycarbonate, as was described in Section 2.3.4.

The motor mounts are cut out of 20-gauge (0.75 mm thick) sheet steel using the water-jet cutter. They are designed to enclose the entire motor because the motor shaft of the motor that we purchased could not be reversed, so the motor could only be mounted at the top, therefore requiring such a tall design because the carbon fibre tubes would not allow otherwise. However, we shall see in Section 4.1 that this design causes issues with vibration. A simulation for axial load was performed in Figure 26 and it was found that the motor would not produce any significant distortion or stresses.

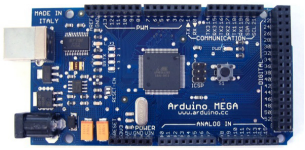

Figure 27: Photograph of Arduino Mega 2560 microcontroller.

2.4 Electronic components

2.4.1 Overview of electronics

Selecting the appropriate electronic components for a quadrotor is challenging because of the vast amount of options available on the market. Also, having selected the components to use, we must then learn how to use them. A quadrotor flying vehicle requires several essential electronic components:

1. A central processing unit to perform all the required computation.
2. Sensors to measure orientation and position.
3. Motors for propulsion.
4. Wireless communication for remote control.
5. Battery for providing power.

The constraints on our purchasing decisions include price, size, and weight. In each of the following sections, we will discuss the specifications desired for the respective electronic component.

2.4.2 Arduino Mega microcontroller

The Arduino Mega 2560 (Figure 27) is an open-source microcontroller that contains the ATMega 2560 processor. We chose this microcontroller for several reasons, but primarily it is because it is an open-source development platform, and therefore there are many software libraries available to use. Importantly, most of the electronics used in Quaffle have Arduino library support, so we can test our electronic parts without spending time implementing drivers etc. It was mentioned in Section 1.3.1 that there are some open-source projects for quadrotors designed for Arduino. As such, this choice of platform allows us to mitigate the time spent reinventing the wheel. The programming language of Arduino is based on the Processing programming language (which is in turn based on Java), which we are familiar with. Other reasons to choose the Arduino Mega 2560 include affordability and the fact that it has sufficient processing power for flight control (as demonstrated in numerous other quadrotor projects).

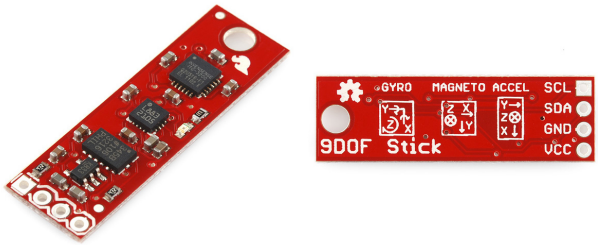

Figure 28: Photograph of SparkFun 9DOF inertial measurement unit.

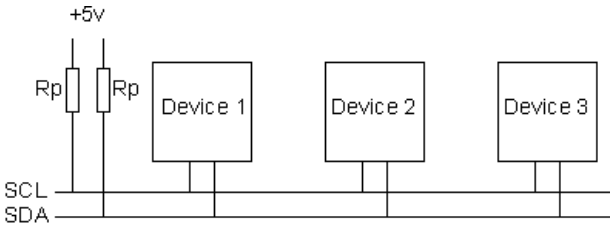

Figure 29: Schematic of correct wiring for I^2C devices [6].

2.4.3 Inertial measurement unit

The inertial measurement unit used in Quaffle is a SparkFun 9 degrees of freedom IMU (Figure 28). Despite the low cost and small size of the chip, it provides accurate orientation measurement, especially when used in conjunction with a noise filter such as a Kalman filter. The small size of the chip makes it easy to install on the quadrotor.

The SparkFun 9DOF IMU communicates using a serial interface known as the inter-integrated circuit (I^2C). This protocol contains four lines: supply voltage, ground, serial data line (SDA) and serial clock (SCL). The SCL line is used for data synchronisation and the SDA line is used for data transmission. This method of serial interfacing is capable of connecting to multiple devices just through the SDA and SCL lines. The number of devices that can be connected is limited by the host device; in this case, the Arduino Mega 2560 allows I^2C to connect up to 112 devices (the Arduino Mega 2560 contains 127 addresses, but 15 are reserved for internal operations). The SDA and SCL lines are "open drain" drivers; hence, in order to read high, it must be connected to a pull up resistor. The diagram in Figure 29 demonstrates how to properly wire the I^2C connection for multiple devices [6].

The AeroQuad software takes readings from the IMU at a rate of 50 Hz. The sampling rate should not be any higher because this ensures that the quadrotor is able to respond to the new measurements before it takes the next measurements. The SparkFun 9DOF IMU is capable of up to 16 MHz sampling rate.

To read the data through I^2C the software follows the following protocol sequence [6]:

1. Send start sequence.

2. Send I^2C address of the device we are reading with the R/W bit set to low to indicate we are writing to the device.

Figure 30: Photograph of Turbojet 880 KV brushless motor used (rotor not shown).

Table 2: Specifications of Outrunner 880 KV brushless motor [7].

PWM duty cycle	**25%**	**50%**	**75%**	**100%**
Current (A)	1.5	6.1	14.2	20
Power (W)	17.5	70	160	210
Thrust (g)	230	650	1290	1380

3. Send internal register address that we would like to read.

4. Send a start sequence again.

5. Send I^2C address of the device we are reading with the R/W bit set to high to indicate we are reading from the device.

6. Read data byte from the device.

7. Send stop sequence.

2.4.4 Motors and electronic speed control

The rotors are the only actuator for a quadrotor, so it is essential to ensure that the rotors can support the weight of the entire robot and have sufficient performance for flight. Our target weight of the robot is about 1.3 kg. We chose 3-phase brushless motors because of their excellent power output, small size, and light weight. The use of a electronic speed controller (ESC) allows the rotational speed to be adjusted more accurately using pulse width modulation (PWM) on the voltage input.

To choose a proper brushless motor, we must first estimate the desired amount of lift that the motor can generate, given the propeller size. A Turbojet 880 KV brushless motor with a 12×45 propeller was chosen (Figure 30); the specifications for a similar motor, an Outrunner 880 KV motor, are listed in Table 2 and used for analysis of motor characteristics [7].

We determine the duty cycle required to lift our 1.3 kg quadrotor equipped with four of these motors by multiplying the thrust in Table 2 by four. This is then plotted on a graph

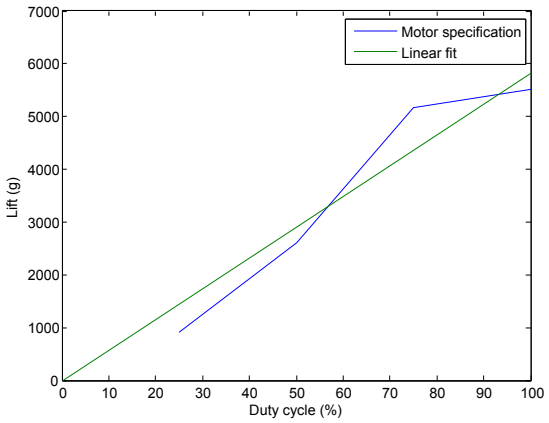

Figure 31: Graph of motor lift with respect to input duty cycle for an 880 KV motor with 12×45 propeller.

shown in Figure 31. For a simple approximation of lift with respect to duty cycle, we fit a linear function. Using least-squares regression we obtain that:

$$F \approx 58.24D \tag{3}$$

where F is the total lift force (g) and D is the PWM duty cycle (percent). For the hovering condition, F is set to be the weight of the quadrotor and we can calculate that the requisite PWM duty cycle required is 22%. A higher duty cycle will allow the quadrotor to fly upwards; however, limitations in total current will limit the maximum thrust.

To determine the amount of current that can be drawn, we also plot the total current for the four motors against duty cycle in Figure 32. We use a quadratic least-squares fit to interpolate between points, and we find:

$$I \approx 0.006D^2 + 0.2271D \tag{4}$$

where I is the drawn current and D is the duty cycle. Using this, we determine that at a duty cycle of 22% for hovering, it will draw about 7.9 A of current.

When we assembled the quadrotor and began tests, we found that the duty cycle required for lift is 30% instead of 22%. The total current drawn varies from 10 A to 15 A. Two reasons for the discrepancy between this and our theoretical values exist. Firstly, the specifications used to derive our expected duty cycle and current are for a different model of motor (the Outrunner as opposed to the Turbojet), which, despite being very similar, can still have different performance especially seeing as the Turbojet is cheaper. Secondly, some parts of the chassis of Quaffle may disrupt the airflow behind the propeller, such as the quadrotor corner discussed in Section 2.3.3. This would reduce the lift and overall efficiency of the system.

The Turbojet 880 KV brushless motor comes with a 30 A Turbojet ESC (Figure 33). This ESC has the following list of features:

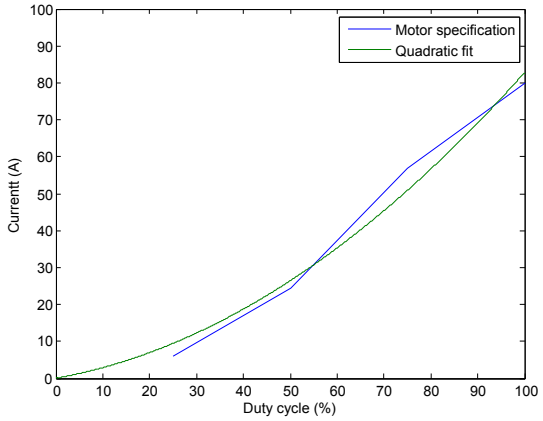

Figure 32: Graph of motor current with respect to input duty cycle for an 880 KV motor with 12×45 propeller.

1. Low resistance
2. High rate PWM
3. Support for up to sixteen cells
4. Soft start ramp up
5. Low torque start
6. User programmable brake
7. Auto motor cutoff with reset
8. Throttle range self-adjusting
9. Auto shut down when signal is lost
10. Low voltage auto setting based on battery
11. Safe power arming program ensures that motor will not run accidentally when aircraft is powered on

The ESC takes as input the controller PWM output and the main power from the battery; and the output of the ESC is the three-phase power to the motor (Figure 34). To reverse the spinning direction of the motor, it suffices to switch two of the three wires for three-phase power.

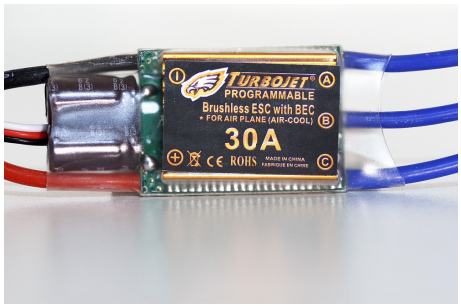

Figure 33: Photograph of electronic speed controller for Turbojet 880 KV brushless motor used (rotor not shown).

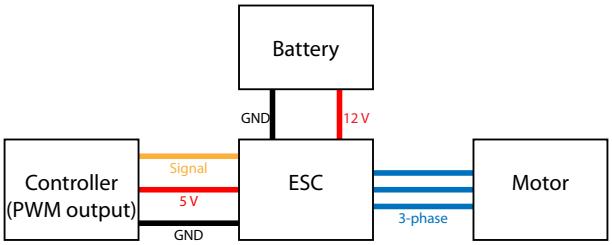

Figure 34: Wiring schematic of electronic speed controller for Turbojet 880 KV brushless motor.

2.4.5 Wireless communication

Quaffle can communicate with external devices using two methods. Firstly, we have a 2.4 GHz remote controller so that we can perform manual test flights before automatic flight control has been fully developed. The remote controller that we use is a HobbyKing remote controller with 7 channels. To connect this device to the quadrotor, we attach the 7-channel receiver to the analog inputs on the Arduino Mega 2560.

To use this remote controller with the AeroQuad software, we configure the remote control to AP mode for airplanes. For the left joystick, the vertical axis controls the average throttle

Figure 35: Photograph of HobbyKing 7 channel remote controller (left) and receiver (right).

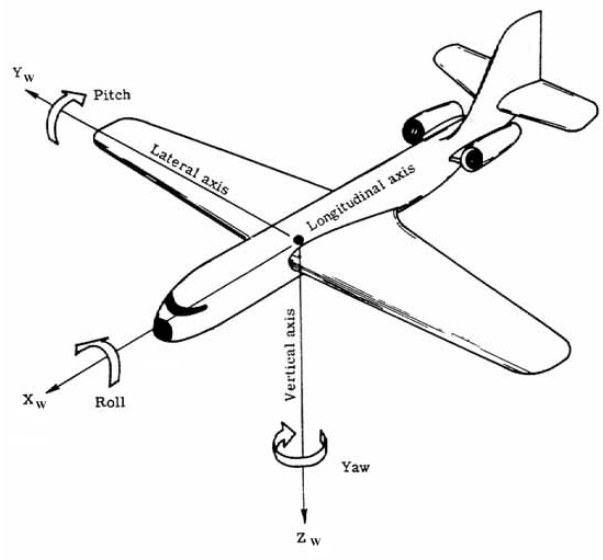

Figure 36: Diagram indicating the yaw, pitch, and roll directions for a fixed-wing aircraft [8].

power of the four motors and the horizontal axis controls the yaw direction of the quadrotor. For the right joystick, the vertical axis controls the pitch and the horizontal axis controls the roll. These controls are analogous to that of a fixed-wing aircraft depicted in Figure 36.

The second form of wireless communication available to Quaffle is a 2.4 GHz Bluetooth receiver. Whereas the aforementioned remote controller can only send analog data to Quaffle, the Bluetooth chip can send digital information, making it suitable for communication with a computer program such as the object recognition program described in Section 2.2. Bluetooth allows 8 bits of data transfer at a time once the two devices have established the connection. The controlling device is called the master, and the device being controlled is called slave. Bluetooth on the quadrotor also allows any device with Bluetooth capability to connect to it. This means that we can use our laptop and smartphone to control it since most of these devices have Bluetooth enabled nowadays. To connect to a Bluetooth device, we use the following procedure:

1. Enable Bluetooth on the slave device.
2. Use the master device to scan for the slave device.
3. When the slave device has been found on the master device, the master device will require a 4-digit pin specified by the slave device. Then the connection is established.

Bluetooth has the advantage of being very cheap when compared to other types of wireless serial communication. However, Bluetooth has a very limited range of operation (usually limited to around 10 m). Also, there is typically some delay in receiving the data, which can

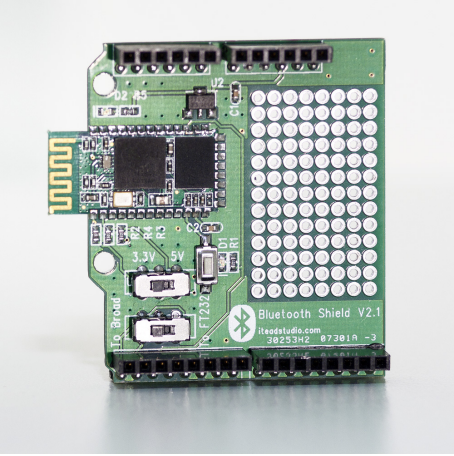

Figure 37: Photograph of Bluetooth shield.

potentially be fatal to a flying robot. Despite these disadvantages, we will use the Bluetooth shield (Figure 37) for early development purposes.

We have managed to develop a communication protocol to control the motor speed using console input. For example, to set the duty cycle for Motor 1 to 20%, one enters `motor 1 20`. The source code for this can be found in Appendix A.

2.4.6 Battery

For a flying machine such as a quadrotor, the obvious choice of battery is a lithium ion polymer (LiPo) battery. The LiPo battery has the highest energy density per weight amongst affordable battery options, and are quite inexpensive. However, care should be taken when charging and discharging LiPo batteries, since an overcharge can lead to a dangerous explosion and an overdischarge can lead to other forms of battery failure. Hence, we use a specific LiPo charger while charging the battery since these chargers have a voltage level monitor to prevent overcharging.

In Section 2.4.4 we have determined that the motor draws about 10 A to 15 A of current when in flight. For a target flight time of 10 minutes (0.167 hours), the desired battery capacity would be around 1.6 Ah to 2.5 Ah. We also require a battery voltage of 11.1 V as specified by the Turbojet ESC.

Thus the battery of our choice is a 3-cell Zippy LiPo 11.1 V battery shown in Figure 38. The battery capacity is 2650 mAh, exceeding our desired specifications. When a constant 15 A current is drawn, we measured that it lasts around 11 minutes before the battery level runs below its failure level.

2.4.7 Installation of electronic instruments

We use the AeroQuad shield v2.1 (Figure 39) to combine the electronics. This shield is specifically designed to stack on the Arduino Mega 2560 and provides an easy connection

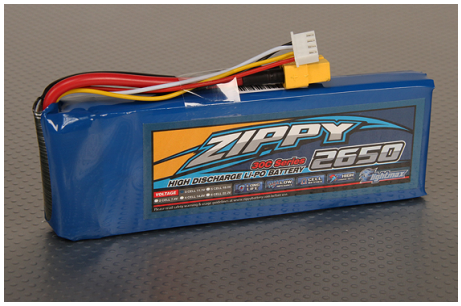

Figure 38: Photograph of 3-cell Zippy LiPo 11.1 V battery.

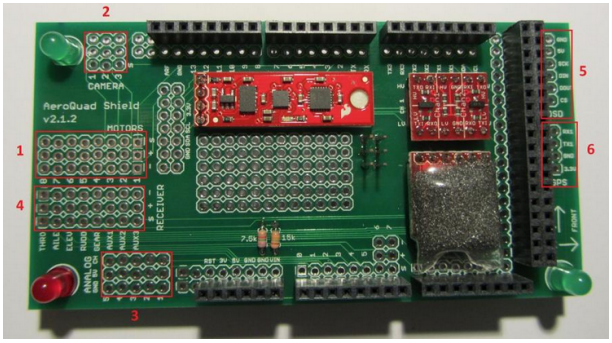

Figure 39: Photograph of AeroQuad shield v2.1. The labelled parts are: 1. Eight outputs for regular PWM ESCs for brushless motors; 2. Three outputs for servo control; 3. Analog inputs for other sensors such as an ultrasonic rangefinder; 4. Radio receiver channel inputs; 5. Connection pins for MAX7456 OSD; 6. Connection pins for GPS.

to the Sparkfun 9DOF IMU. The shield also contains PWM and remote receiver output pins which allow us to save time wiring up the ESC and remote control channel connections. Ultimately, the shield provides an easy and clean connection to all lour electronic devices which reduce the chance of human error and speed up the development of Quaffle.

For our prototype we have only used the PWM output and the radio receiver channel input, as listed in Table 3.

2.4.8 Power distribution

The power distribution for Quaffle is very important since the motors draw a large amount of current. For normal operation we expect a power draw of around 15 A for all motors, and definitely not exceeding 15 A for each motor. Hence, we decided to use 20 AWG wires for each motor. Also, to ensure that the ESC operates safely, we have installed a 30 A fuse to prevent any individual ESC from drawing a current exceeding 30 A. The Arduino Mega is connected directly to a separate 9 V battery. The reason for connecting the Arduino Mega to a separate battery is because the Arduino always needs to be turned on before the ESC can be turned on to ensure that the ESCs are properly initialised by the Arduino board. Figure 40 demonstrates how we wire the batteries to the electronics.

Table 3: Connection of devices to AeroQuad shield.

AeroQuad shield pin	Device connected
PWM 1	Front motor
PWM 2	Back motor
PWM 3	Right motor
PWM 4	Left motor
`aile`	Receiver channel 1
`elev`	Receiver channel 2
`thro`	Receiver channel 3
`rudd`	Receiver channel 4

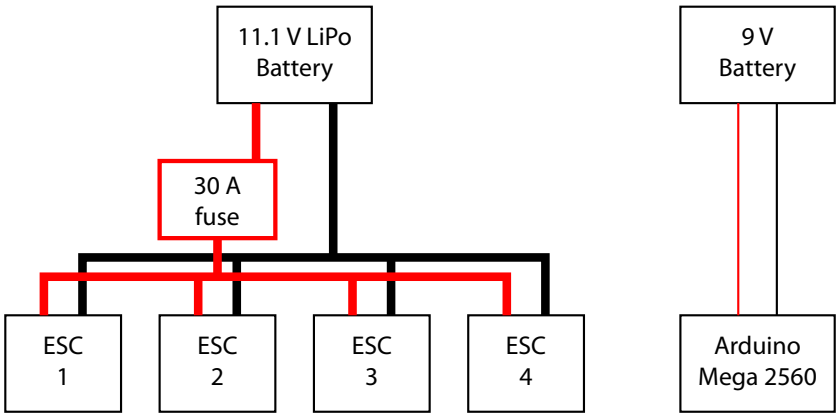

Figure 40: Block diagram showing how we wire the batteries to the electronics. The thicker red and black lines indicate 20 AWG wires.

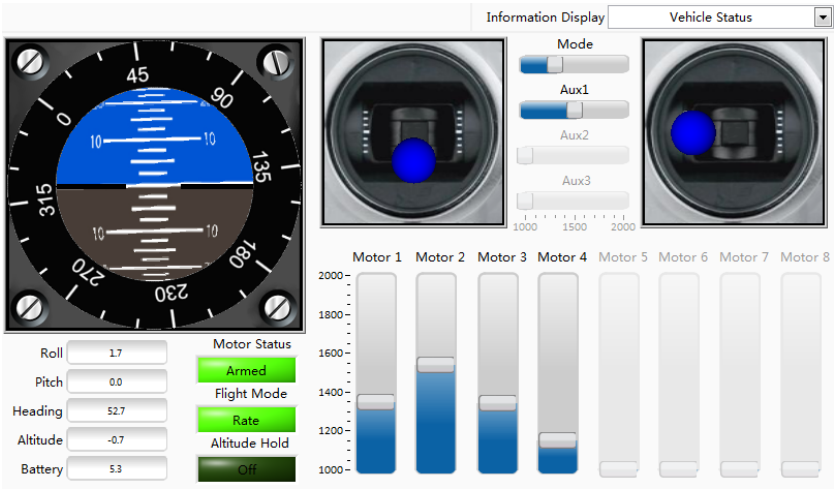

Figure 41: Screen capture of AeroQuad GUI. This status display will provide information from IMU (Roll, Pitch and Yaw/Heading), ESC speed (Motors 1, 2, 3 and 4) and the Radio remote control joystick position.

3 Testing protocol

To successfully develop a working flying quadrotor, we need to have all onboard electronics tested, calibrated and tuned. The Aerroquad software allows manual tuning for the PID control; hence, we need to eliminate as many variables in our system as possible. We also need to fly the quad-rotor to ensure the tuning parameters are correct by visual observation of the flying performance. This section of the report will discuss the method we use for testing, calibrating and tuning the electronics and flying the quad-rotor.

3.1 Electronics test, calibration, and tuning of control parameters

The AeroQuad configurator provides a GUI program to help with setting up the quadrotor for flight. The first step to set up the quadrotor is initialize the Electrically Erasable Programmable Read-Only Memory (EEPROM) on the Arduino Mega. This is necessary to allow the calibration and tuning data to permanently store in Arduino Mega processor. Once the EEPROM is initialized, we can then calibrate the IMU, transmitter and ESC. The purpose of calibrating the IMU is because the IMU may not lie perfectly flat on the quadrotor; hence, the quadrotor must be placed on a level flat surface during this calibration. For transmitter and ESC calibration, the AeroQuad only needs to check the range of input and output values.

The AeroQuad GUI allows has a feature that allow developer to check the status of all the electronic parts on the quad-rotor (Figure 41). With this feature, we can test all the electronics easily.

The last step before putting the quadrotor to flight test is tuning the PID. The AeroQuad software uses a manual PID control which means there are proportional, integral and derivative gain for every data point that needs to be calculated to control the speed of the

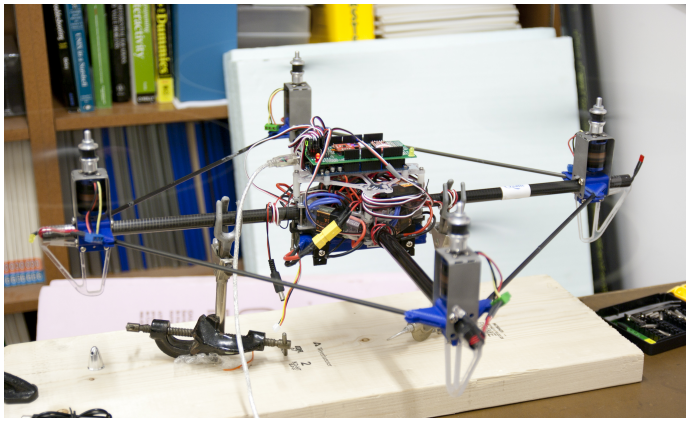

Figure 42: Photograph of test station for rotation in one axis.

motor. In our case, the PID controlled data are roll, pitch and yaw. A rough value of PID must be tuned because if these values are far from the ideal value, the quadrotor will be impossible to fly properly.

The typical procedure to calibrate these PID values are first setting the integral and derivative value to 0, then slowly increase proportion from 0. To tune the PID for one out of three rotational axes, we need to fix the other two axes of the quad-rotor so that it is constrained to move in only the axis of interest (Figure 42. Then, we adjust the PID value of the free axis by slowly increasing the proportional value. For example, if we fix the yaw and pitch direction, we can tune the roll axis PID value. We need to set the motor throttle to lift off position, then we can try to change the roll by hand. With a proportion value roughly close to the ideal value, we will feel some stiffness of from the roll axis because the quad-rotor is trying to resist external forces. To mitigate or eliminate the oscillation of the quadrotor trying to get back to balance, we have to adjust the derivative value. We can produce the oscillation by manually pulling the roll axis off balance then let it try to recover itself. We will observe a critical damping of the oscillation if the derivative value is correct. We can do the same thing for the pitch axis to tune the pitch PID values. For tuning the yaw axis, we have to let the quadrotor fly in midair and also adjust proportion value first then derivative value. It is recommended not to adjust the integral value, we will adjust this value only if the quadrotor start to get off balance and unable to recover over some period of time.

3.2 Indoor flight test

The flight test is the most difficult part of this project because this is when the entire control system put to work together. The first flight has a high chance of failing because it is impossible for one to come up with the perfect model to simulate the entire control system. With so many variables that could cause flight failure, it is a good idea to put protection on the quadrotor during the first flight test and eliminate as much external disturbance as possible.

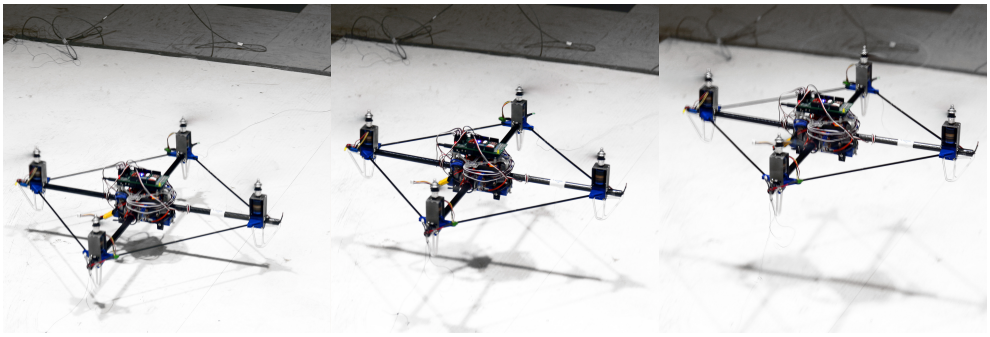

Figure 43: Photographs showing the takeoff of Quaffle to enter stable hovering. The three frames are taken at 3 fps with a shutter speed of 1/40 s.

To minimize the external disturbance, we decided to test our robot indoors so that it is not affected by wind. We also tie a string to each corner of the quadrotor so that it will not fly out of bounds and consequently crash into any object in the indoor environment. However, during our first few flight tests, the quadrotor can only hover for a few seconds before losing balance immediately. A possible reason for instability is ground effects caused by extremely low altitude hovering. Using digital cameras to record videos and images during the flight test, we find that the propeller is susceptible to significant vibrations of a few centimetres at its tips (Figure 44. This is another possible reason for poor flight performance. The vibrations will be analysed quantitatively in Section 4.1.

4 Results and discussion

The project is intended to design a quadrotor flying machine and integrate the obstacle detection program on it. We have not been able to integrate these two components due to our limited time, so we will separate the discussion here into two parts: the design of the quadrotor and object recognition.

4.1 Design of quadrotor

The entire quadrotor consists of three main parts: software, chassis, and electronics. The AeroQuad open-source project was successfully installed in Quaffle. Porting the software to Quaffle was very easy because has very similar hardware to the official AeroQuad quadrotor.

We have looked at the available lift that the motor we purchased can provide. With four Turbojet 880 KV brushless motors and 45 $\times$ 12 propellers, the quadrotor can produce 1.3 kg of lift at about 30% throttle input. At this rate, we measure that the motors will altogether draw about 10 A to 15 A of current from the battery. The 3-cell Zippy LiPo 11.1 V battery has a capacity of 2650 mAh and can provide about 11 minutes of flight time. Preliminary testing show that the robot is capable of lifting off the ground easily. However, since we have not had sufficient time to tune the PID control parameters to perfection, the robot can only

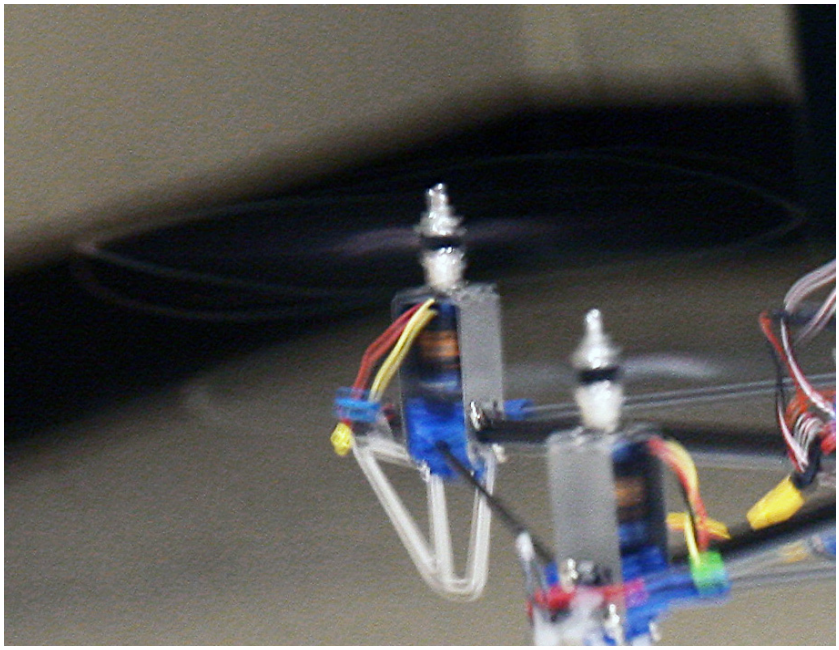

Figure 44: Close-up of photograph of Quaffle hovering in mid air. Notice the path traced by the white tips of the rotor against the dark background. The shutter speed is 1/40 s.

hover for a few seconds before losing balance.

The chassis has proven to be very strong. Even with some crashes from a height of 1 m, the robot has survived with no noticeable damage. However, the motor mount is susceptible to vibrations. In Figure 45 we find that the distortion under a 1 N horizontal load is 0.04 mm. To calculate the natural frequency, we use the equation below, where k_{eq} is the equivalent spring constant and m_{eq} is the equivalent mass, here assumed to be the mass of the motor.

$$\omega_n = \sqrt{\frac{k_{eq}}{m_{eq}}} \tag{5}$$

$$= \sqrt{\frac{1\,\mathrm{N}/0.04\,\mathrm{mm}}{110\,\mathrm{g}}} \tag{6}$$

$$= 477\,\mathrm{rad/s} \tag{7}$$

$$\approx 4500\,\mathrm{rpm} \tag{8}$$

The natural frequency of vibration, also known as the resonant frequency, is 4500 rpm. The application of even a small oscillatory force close to the resonant frequency can yield vibrations of a large amplitude. To find the angular speed of our motor, we observe that there are four faint rotor trails visible in Figure 44 over a period of 1/40 s. Since the rotor

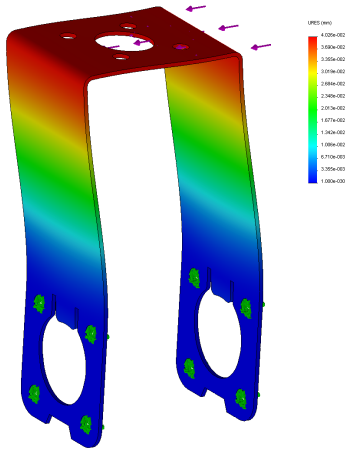

Figure 45: Results of finite-element analysis of motor mount under transverse load. For an applied force of 1 N, the maximum deformation is approximately 0.04 mm. The deformation scale here in this figure is approximately 168. The material is mild steel. The simulation was done using Solidworks 2012.

has two blades, we can estimate that the rotor has turned 2 revolutions in 1/40 s.

$$f = \frac{2}{1/40\,\mathrm{s}} \tag{9}$$

$$= 80\,\mathrm{Hz} \tag{10}$$

$$= 4800\,\mathrm{rpm} \tag{11}$$

As we can see, the rotor velocity is close to the resonant frequency, yielding the vibrations. It should be noted that the motor mount was not originally designed in this "tall" configuration - instead, it was supposed to be only a few millimetres above the quadrotor corner. The motor shaft would then be reversed so that the motor could be mounted on the bottom with the rotor on top. However, our attempt to reverse the shaft on the Turbojet 880 KV motor failed. With the "short" motor mount, the spring constant k_{eq} would be much higher, therefore moving the resonant frequency far away from the motor speed.

4.2 Object recognition software

During the four month period, a fairly accurate recognition software has been produced by using the popular OpenCV library. the software is able to use three modes at user's request to do object recognition for recognizing any generalized pattern, facial recognition for recognizing faces, or laser tracking for tracking bright spot. There are a few limitations such as various ways of fetching images and computation power.

Since the camera has to fly with the UAV, at first we thought of doing the abstruse computer vision computation on the Arduino board. Given the low processing power of

Arduino board, however, we had to offload the computation to a nearby computer with a wireless camera. Due to the bandwidth issues with video streaming through a radio frequency serial port, the image quality is rather low, which makes the object recognition difficult. Without a high quality image, the SURF method may not be able to extract enough valuable key points for recognition. In addition, offloading images to a computer means there has to be a ground station nearby wherever the UAV is flying to, which is a major drawback for autonomy.

Due to the nature of computer vision, it takes lots of computation power. The best solution for our case is to get a high processing power micro-board of that is small and light enough to fit on the Quaffle chassis. An example would be the motherboard of an ultraportable laptop equipped with a powerful processor. Such a device would be ideal since we will not need to offload images through radio to a nearby computer as the device is sufficiently powerful to perform its own image processing. However, this is beyond our budget.

5 Conclusion

The objective of this project is to design a quadrotor and to investigate the possibility of using object recognition as onboard obstacle detection/object tracking for quadrotor. With limited time frame for this project, we have not managed to succesfully integrate the quadrotor with the object recognition software. However, we have successfully developed a quadrotor prototype and a object recognition software separately.

After the four month period, our quadrotor is able to fly with a remote controller. It is able to complete all of the manoeuvres we outlined in Section 2.1.1. The quadrotor may be classified as a 4 channel aircraft: altitude, roll, pitch and yaw. We have also determined that our robot is capable of hovering for around 11 minutes with the 3-cell LiPo battery we chose for this project. We have not managed to achieve stable balancing of the quadrotor when hovering. After some test flights while we tune PID control parameters, we determine that the lifting speed at which the rotor spins is too close to one of the natural frequencies of vibration, which causes the rotor to vibrate. To solve this issue, we must redesign the motor mounts and possibly purchase smaller propellers and motors.

The object recognition software written in OpenCV is a complete software suite for our Quaffle that is able to track any given object using SURF method, any generic face using HLF, any specific face using the integration of HLF and a neural network, and lastly any laser point shining on a surface. The software is able to continuously run on an Intel Core 2 Duo computer constantly recognizing four objects simultaneously with 65% of CPU usage, which is a good result considering that computer vision requires a large amount of computational power. The software was tested with a 720p (1280×720 pixels) USB camera. The recognizing result has a 95% accuracy rate and 0.01% false alarm rate. If a wireless camera is used instead, the accuracy would decrease dramatically due to the low quality of the camera and bandwidth constraints on video streaming. A serial class has also been developed to output the detecting result of the recognition software to the Arduino board. Our next step would

Table 4: Summary of deliverables.

Deliverable	Accomplished (Yes/No)	Comments
Prototype flyable UAV	Yes	Flyable with remote control
Separate software package that can be installed and used on any UAV with Arduino controller	Yes	The software package can be uploaded to any UAV with Arduino controller; a second piece of C# machine vision software can be installed on any computer to perform real time object recognition
Demo video of the performance of our UAV	Yes	Footage of indoor test flight
Engineering recommendation report	Yes	N/A
Documents describing all components in the system	Yes	N/A
Complete design drawings	Yes	N/A

be to devise an algorithm on the Arduino side to take input from the serial port and control the aircraft accordingly when a target is found.

6 Project deliverables

6.1 List of deliverables

Over the four month period, we have completed all the goals that we have outlined in our proposal. A summary of deliverables is shown in Table 4.

6.2 Financial summary

The financial summary is shown in Table 5. The total self-sponsored cost is $350.19 and the total for the project lab is >$110.

6.3 Ongoing commitments by team members

Even though this project is self-sponsored, we would still like to improve Quaffle after the ENPH 459 course is complete. We will continue to work on the following items, but not limited to it:

1. Electrical

Table 5: Financial summary of components used in Quaffle.

Component	**Quantity**	**Total price ($)**	**Purchased by**
SPA 9DoF IMU	1	109.21	self-sponsored
Arduino Mega 2560	1	36.49	self-sponsored
AeroQuad shield v2.1	1	34.80	self-sponsored
Lithium polymer battery	1	9.42	self-sponsored
IMAX B6 charger/discharger for LiPo	1	24.99	self-sponsored
Stackable Bluetooth shield	1	22.60	self-sponsored
880 KV Turbojet motor with ESC and bullet head	4	100.08	self-sponsored
12×45 Propeller (pair)	2	10.12	self-sponsored
9 V battery adapter	1	2.48	self-sponsored
9 V batery	1	N/A	project lab
Wireless camera	1	55.00	project lab
Remote controller & receiver	1	55.00	project lab
3D printed parts	N/A	N/A	project lab
Carbon Fibre tubes	2	N/A	project lab
Aluminium, steel, and polycarbonate	N/A	N/A	project lab

- Remote control test of the UAV in an open outdoor environment.
- Tune PID control parameters to improve flight performance.
- Implement flight algorithm to allow autonomous flying without manual remote control.

2. Object recognition

   - Implement software to control camera rotation and stabilisation in flight using servo motors.
   - Add-ons for the flight control algorithm to take into consideration object recognition outputs. Once the signal has been sent from the object recognition software confirming the target has been found, the UAV will stop searching and fly towards the target.

3. Mechanical

   - Redesign of the landing gear to absorb more landing impact and increase ground clearance to allow mounting of camera.
   - Gimballed camera mount able to rotate camera in two axes.
   - Styrofoam bumper for safety and to mitigate rotor damage in case of crashes.
   - Cover for central assembly to protect electronic components.
   - Redesign of motor mount to mitigate vibrations.

7 Recommendations

Since the Quaffle project is self sponsored, this section would be only intended for the further candidate that might be sponsored by us to continue working on our UAV project.

The main goal of the Quaffle for this limited four month period is to design, fabricate, and prototype a flyable quadrotor vehicle that is able to fly, hover, and land safely. We also wanted to develop a computer vision software that allows the aircraft to perform object recognition, facial detection, and laser tracking. We have successfully completed our goal for the four month period. There are many further development that we would like to see on our Quaffle since it has a great potential for autonomous flying, surveillance, rescue, and even for military applications. The following are a few goals for people who might be interested in continue working on our project:

1. Total integration of the object recognition and flight control software.

   Right now, flight control software does not take into any consideration of the result of the object recognition. We recommend the integration of the two. For example, when a target has been found, the UAV would hover and track the target, and possibly attempt to land on it.

This objective is mostly implemented in software so the potential team should have extensive programming experience. They should not only focus on the expansion of the already existing object recognition software, but also write new classes in the flight control algorithm to parse in serial results from object recognition and control the UAV. The objective should be completable within a 4 month period.

2. Further development on the UAV

   The Quaffle is a rather big machine to work with. It is possible that a smaller quadrotor will be more manoeuvrable and easier to develop, so a smaller version of Quaffle using smaller motors may be useful.

   This objective has all of mechanical, electrical, and software design. The potential team should compose of well-rounded engineering students, such as Engineering Physics students. They should focus on the building and prototyping of a smaller version of Quaffle while maintaining all of the functionalities. Circuit boards for various functions such as power distribution can be created using the same design but smaller in size. The software can be the same as the one currently being used, but with well-tuned PID control parameters for stable flight. The objective should be completable within a 4 month period.

There is still a lot of improvement that we can make to Quaffle; with time, funding, and good people, Quaffle has lots of potential to become the next generation of UAV.

8 References

[1] Mellinger, D. (2010). *Quadrotor.* Retrieved April 2012, from University of Pennsylvania `https://fling.seas.upenn.edu/ dmel/wiki/index.php?n=Main.Quadrotor`.

[2] Lupashin, S., D'Andrea, R. (2012). *A state-of-the-art platform for motion control research.* Retrieved April 2012, from Institute for Dynamic Systems and Control at ETH Zurich `http://www.idsc.ethz.ch/Research_DAndrea/FMA`.

[3] Angelosanto, G. (2008). *Kalman Filtering of IMU Sensor for Robot Balance Control.* Massachusetts Institute of Technology.

[4] *Carbon Fibre Tubes.* (n.d.). Retrieved April 2012, from Carbon Fibre Tubes Ltd: `http://carbonfibretubes.co.uk/technology.html`

[5] DiCesare, A., Gustafson, K., & Lindenfelzer, P. (2009). *Design Optimization of a Quad-Rotor Capable of Autonomous Flight.* Worcester Polytechnic Institute.

[6] *I2C Tutorial.* (n.d.). Retrieved April 2012, from Robot Electronics: `http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html`

[7] *Motor AC2836-358, 880Kv.* (n.d.). Retrieved April 2012, from DIYDRONES: https://store.diydrones.com/Motor_AC2830_358_880Kv_p/ac-0004-05.htm

[8] *Pointing an Instrument on an Airborne Platform.* (n.d.). Retrieved April 2012, from http://mtp.mjmahoney.net/www/notes/pointing/pointing.html

[9] *The Open Source Quadcopter.* (n.d.). Retrieved April 2012, from Aeroquad: http://aeroquad.com/

Appendix A: Bluetooth Serial Communication Code

```
/*
        Project: Quaffle - Quadrotor UAV
        By: Anson Liang, Daniel L. Lu, Richard Lee
        Date: Jan 16, 2012

        File:Cmd.pde
        - This file contains functions for communicating with Arduino
*/

#define CMD_DEBUG_SERIAL 0

#define CMD_STRING_SIZE 16
#define CMD_MAX_ARGS 5
#define CMD_NUM_COMMANDS 7


#define CMD_PRINT(arg1, arg2, arg3, arg4)\
        {Serial.print("CMD:: "); Serial.print(arg1); Serial.print(" "); Serial.print(arg2);
            Serial.print(" "); Serial.print(arg3); Serial.print(" "); Serial.println(arg4);}


/* ---------- global variables ---------- */
char* cmd_ptr;
int cmd_len;
int cmd_argCnt;
int cmd_trigger = 0;
int cmd_args[5] = {0};// five args max, 'a' is arg counter
int cmd_motorInitFlag = 0;

extern int global_select;

enum cmd_list{
        //This is an enum of all the commands.
        START,
        ALT,
        MOTOR,
        IMU,
        HOVER,
        READY,
        HELP
};

char cmd_keyword[CMD_NUM_COMMANDS][CMD_STRING_SIZE] = {
        "start",
        "alt",
        "motor",
        "imu",
        "hover",
        "ready",
        "help"
};

/* ---------- public functions ---------- */

/*
        cmd_Read()
        - Read serial input from user
*/
void cmd_read()
{
        if( Serial.available() > 0 ) {  // command length is 6 bytes
                delay(100); // A delay must take place to avoid serial read error
```

```
                char* cmdbuf = (char*)malloc(sizeof(char) * CMD_STRING_SIZE);
63              char c = 'n';
                int i = 0;
65              while( Serial.available() && c != '\n' ) {  // buffer up a line
                        c = Serial.read();
67                      cmdbuf[i++] = c;
                }
69
                i = 0;
71              while( cmdbuf[++i] != ' ' ) ; // find first space
                cmdbuf[i] = 0;          // null terminate command
73              cmd_ptr = cmdbuf;
                cmd_len = i;          // length of cmd
75              char* s;
                char* argbuf = cmdbuf+cmd_len+1;
77              int a = 0;
                while( (s = strtok(argbuf, " ")) != NULL && a <= CMD_MAX_ARGS ) {
79                      argbuf = NULL;
                        cmd_args[a++] = (byte)strtol(s, NULL, 0); // parse hex or decimal
                            arg
81              }

83              cmd_argCnt = a;          // number of args read

85              if(pCharSize(cmd_ptr) != cmd_len)
                {
87                      CMD_PRINT("Command Error!","","","");
                        Serial.flush();
89                      cmd_trigger = 0;
                        return;
91              }
                cmd_trigger = 1;
93
                if(CMD_DEBUG_SERIAL)
95              {
                        CMD_PRINT("cmd:", cmd_ptr, "cmd_len:", cmd_len);
97                      CMD_PRINT("# arg:", cmd_argCnt, "","");
                        for(int i = 0; i < cmd_argCnt; i++) {
99                              CMD_PRINT("arg", i, ":", cmd_args[i]);
                        }
101             }

103             Serial.flush();

105     }
}
107
/*
109     cmd_talk()
        - tell Arduino what to do according to what is read
111 */
void cmd_talk()
113 {
        int cmdIndex;
115     for(cmdIndex=0; cmdIndex<CMD_NUM_COMMANDS+1; cmdIndex++){
                if(strcmp(cmd_keyword[cmdIndex], cmd_ptr , cmd_len)) break;
117     }
        //CMD_PRINT("cmdIndex", cmdIndex, "", "");
119     switch(cmdIndex){
                case START:
121                     //code for start;

123                     break;
                case ALT:
125                     //code for alt;
```

```
                        break;
127                 case MOTOR:
                        //code for motor;
129                     if(cmd_motorInitFlag) {
                            motor_control(cmd_args[0], cmd_args[1]);
131                     }else{
                            CMD_PRINT("CMD_Error:_please_initialize_system_first", "", "
                                ", "");
133                     }
                        break;
135                 case IMU:
                        imu_setValue(cmd_args[0], cmd_args[1]);
137             break;
                case HOVER:
139                     hover_command(cmd_args[0], cmd_args[1], cmd_args[2]);
                        break;
141                 case CMD_NUM_COMMANDS:
                    default:
143                 //invalid command
                    //handle invalid command here...
145                 ;
        }
147
}
```