Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Ashley J. Robinson

30th April, 2013

A small scale quadcopter platform for robotic swarm development

Project supervisor: Professor Steve R. Gunn
Second examiner: Dr Richard M. Crowder

A project report submitted for the award of
MEng Electronic Engineering with Artificial Intelligence

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for the award of
MEng Electronic Engineering with Artificial Intelligence

by Ashley J. Robinson

The increasing popularity of quadcopters as intelligent agents in the swarm robotics community warrants development of suitable hardware platforms. Meanwhile the scale of standard quadcopters is decreasing as smaller components become less expensive and more readily available. This project report describes the research and development of a small scale quadcopter with the intention of deployment as a hardware platform for robotic swarm research. Drawing on recent designs the approach is to use a circuit board as the frame but also to keep the propellers in the same plane as the board which is novel for a quadcopter of this scale.

The amount of thrust produced by the final design proved incapable of lifting the weight of the craft but the work done serves as an ideal starting point for the next design revision. This should seek to optimise both mass and available thrust. Control firmware and additional hardware peripherals are also developed along with analysis software.

Contents

List of Figures **ix**

List of Tables **xiii**

Listings **xv**

Abbreviations **xvii**

Symbols **xviii**

Acknowledgements **xxi**

1 Introduction **1**
1.1 Flying circuit boards . . . 2
1.2 Why quadcopters? . . . 2
1.3 Motivation for specification . . . 3
1.4 Contributions . . . 4

2 Background research **5**
2.1 Quadcopter theory . . . 5
2.1.1 Flight . . . 6
2.2 Sensor data . . . 8
2.2.1 Accelerometer . . . 8
2.2.2 Gyroscope . . . 9
2.2.3 Application . . . 9
2.3 Control theory . . . 10
2.3.1 Complementary filter . . . 11
2.3.2 Kalman filtering . . . 12
2.4 Swarm robotics . . . 13

3 Design **15**
3.1 Optimal circuit board configuration . . . 15
3.2 Motors and propellers . . . 18
3.3 Motor housing . . . 18
3.4 Battery . . . 19
3.5 System architecture . . . 20

3.6 Communications . . . 21
3.7 Base station . . . 22
3.8 Project planning . . . 23
3.8.1 Risk management . . . 23

4 Implementation 25
4.1 System components . . . 25
4.2 Motor and propeller selection . . . 27
4.3 Quadcopter circuit board design . . . 28
4.3.1 Component layout . . . 28
4.3.2 Additional features . . . 28
4.4 Quadcopter manufacture . . . 30
4.5 Motor housing fabrication . . . 31
4.6 Expansion hardware . . . 33
4.7 Base station hardware . . . 33
4.8 Prototype manufacture . . . 34
4.9 Firmware development . . . 34
4.9.1 Data acquisition . . . 35
4.9.2 Filtering . . . 37
4.9.3 Secure digital card communication . . . 39
4.10 Base station software development . . . 40
4.11 Full system integration . . . 41

5 Testing and tuning 43
5.1 Hardware alterations . . . 44
5.1.1 Motor drivers . . . 44
5.1.2 System architecture . . . 45
5.2 Hardware setup for firmware verification . . . 46
5.3 Sensor readings . . . 47
5.4 Control loop optimisation . . . 48

6 Evaluation 49
6.1 Performance . . . 49
6.1.1 Thrust to weight ratio . . . 49
6.1.2 Control implementation . . . 50
6.2 Production cost . . . 52
6.3 Project planning: a retrospective . . . 52

7 Conclusions and future work 53
7.1 Archaeopteryx improvements . . . 53
7.2 Formalisation . . . 53
7.3 Propeller optimisation . . . 54
7.4 Weight loss . . . 55
7.5 Final thoughts . . . 56

A Project brief **61**

B Archive description **63**

C Printed circuit board configuration options **65**
C.1 Circular 65
C.2 Tapered square 68
C.3 Clover 70
C.4 Graph generation 70

D Battery comparison **73**

E Software development **75**

F Printed circuit board design files **77**
F.1 Archeopteryx rev.A 77
F.2 Base station transceiver 86
F.3 BlackBox 89

G Hardware modifications **91**

H First propeller **93**

I Project planning **95**
I.1 Act I - actual 95
I.2 Act II - planned 96
I.3 Act II - actual 97

J Firmware listings **99**

K Software listings **103**

List of Figures

1.1 A formation of 20 micro quadrotors in flight. Taken from Kushleyev et al. (2012). 1
1.2 Crazyfile (Rev.B). Taken from Bitcraze. 2
1.3 Frames of reference. Taken from Zhang et al. (2009) 3

2.1 The inertial and body frames of a quadcopter. Adapted from Luukkonen (2011) 5
2.2 Resolving forces 7
2.3 Rotation in xy plane 7
2.4 An accelerometer and gyroscope package with axis notation 8
2.5 Direct calculation of pitch from magnitude only data 9
2.6 Open-loop control 10
2.7 Closed-loop control. Adapted from Sa and Corke (2011) 10
2.8 Basic complementary filter. Adapted from Colton (2007) 11
2.9 Recombination of sensor data 11
2.10 The ongoing discrete Kalman filter cycle. The time update projects the current state estimate ahead in time. The measurement update adjusts the projected estimate by an actual measurement at that time. Taken from Welch and Bishop (2001). 12
2.11 The true value of the random constant is given by the solid line, the noisy measurements by the cross marks, and the filter estimate by the remaining curve. Taken from Welch and Bishop (2001). . . 12
2.12 Schematic diagram of a simple reflex agent. Taken from Russell and Norvig (2010) 13

3.1 Clover layout definition 16
3.2 Clover layout scaling 17
3.3 Motor housing (Revision A) 18
3.4 Motor housing (Revision B) 19
3.5 A comparison of lithium polymer batteries on the market. Raw data held in table D.1 20
3.6 The simplest manifestation of the required system 20
3.7 Complete ideal data flow within the system 21
3.8 An initial prototype of the base station user interface with dummy data 22

4.1 Implemented system architecture 25

4.2 Chosen propellers. Clockwise rotation on the left and anti-clockwise on the right . . . 27
4.3 Motor and propeller combination . . . 27
4.4 Fabricated board. Top side. . . . 30
4.5 Fabricated board. Bottom side. . . . 31
4.6 Fabricated motor housing (Revision B) . . . 32
4.7 Motor housing (Revision C) . . . 32
4.8 The BlackBox expansion option . . . 33
4.9 Transceiver stick . . . 34
4.10 Serial peripheral interface read operations from MPU6000. Values held in table 4.3 . . . 36
4.11 Magnitude response of 25 coefficient finite impulse response filters with a normalised cut-off frequency of 0.437π rads/sample . . . 38
4.12 Archaeopteryx top . . . 42
4.13 Archaeopteryx bottom . . . 42

5.1 Annotated screen capture of an analogue trace (top) and logic analyser (bottom) connected to serial peripheral interface pins of the MPU6000 . . . 44
5.2 Microchip MTD6505 - functional block diagram. Taken from Microchip (2011) . . . 45
5.3 Serial peripheral interface architecture repair . . . 46
5.4 Hardware setup . . . 47
5.5 Visual feedback of attitude . . . 47

6.1 Real-time plots . . . 51

7.1 Propeller/motor/housing sketch . . . 55

C.1 Circular layout . . . 66
C.2 Circular layout scaling . . . 67
C.3 Tapered square layout . . . 68
C.4 Tapered square layout scaling . . . 69

E.1 User interface communications . . . 76

F.1 Schematic page 1. . . . 77
F.2 Schematic page 2. . . . 78
F.3 Schematic page 3. . . . 79
F.4 Schematic page 4. . . . 80
F.5 Schematic page 5. . . . 80
F.6 Design view. Top copper layer looking down. Intermittent line represents ground polygon boundary. . . . 82
F.7 Design view. Bottom copper layer looking down. Intermittent line represents ground polygon boundary. . . . 83
F.8 Layout view. Top layer looking down. . . . 84
F.9 Layout view. Bottom layer looking up. . . . 85

F.10 Base station transceiver Schematic 86
F.11 Design view. Top copper layer looking down. 87
F.12 Design view. Bottom copper layer looking down. 87
F.13 Layout view. Top layer looking down. 88
F.14 Layout view. Bottom layer looking up. 88
F.15 BlackBox Schematic 89
F.16 BlackBox circuit board layout 89

G.1 Three disconnected tracks on the board. Copper section removed with scalpel 91
G.2 HK - XP 3A 1S 0.7g micro brushless electronic speed controller . . 91

H.1 First idea for a bespoke propeller 93

I.1 The actual activities carried out from 30/09/12 to 10/12/12 95
I.2 The planned activities from 06/01/13 to 19/05/13 96
I.3 The actual activities that took place from 06/01/13 to 01/05/13 . 97

List of Tables

3.1 Risk register 24

4.1 Chosen constraints for clover circuit board layout 28
4.2 Layout reasoning 29
4.3 Acceleromter values read from MPU6000. As seen in figure 4.10 . . 36
4.4 Frame composition 40

6.1 Section weights 50

7.1 Improvements for revision B 54

D.1 A sample of lithium polymer batteries on the market 73

F.1 Component list for revision A 81

Listings

4.1 Data acquisition from MPU6000 35
4.2 Implementation of an finite impulse response filter on an Atmel microcontroller 38
4.3 BlackBox log creation and population 39
B.1 README.txt 64
C.1 Scaling comparison graph generation 70
E.1 Example log file. Generated from user interface run in figure 3.8 . . 75
J.1 Main from control microcontroller 99
J.2 Main from transceiver microcontroller 101
K.1 Main form from base station software 103

Abbreviations

3D Three Dimensional.

AWGN Additive White Gaussian Noise.

BLDC Brushless Direct Current.

CAD Computer Aided Design.

CPU Central Processing Unit.

FIR Finite Impulse Response.

GUI Graphical User Interface.

I^2C Inter-Integrated Circuit.

I/O Input/Output.

IC Integrated Circuit.

IIR Infinite Impulse Response.

ISP In-System Programming.

JTAG Joint Test Action Group.

LED Light Emitting Diode.

LiPo Lithium Polymer.

MEMS Microelectromechanical System.

MISO Master In Slave Out.

MOSEFT Metal Oxide Semiconductor Field Effect Transistor.

MOSI Master Out Slave In.

MSB Most Significant Bit.

PC Personal Computer.

PCB Printed Circuit Board.

PID Proportional-Integral-Derivative.

PWM Pulse Width Modulation.

QPROP Propeller/Windmill Analysis and Design Engine.

RC Radio Controlled.

SCLK Serial peripheral interface CLocK.

SD Secure Digital.

SMD Surface Mount Device.

SPI Serial Peripheral Interface.

UART Universal Asynchronous Receiver/Transmitter.

uDFN Micro Dual Flat No-Lead.

USB Universal Serial Bus.

VTOL Vertical Take Off and Landing.

Symbols

β	Propeller hole separation
δ	Recession radius
ϵ	Printed circuit board diameter
θ	Pitch of quadcopter (angle in xz plane)
μ	Printed circuit board safety rim width
ρ	Propeller radius
ϕ	Roll of quadcopter (angle in yz plane)
ψ	Yaw of quadcopter (angle in xy plane)
ω_c	Cut off frequency
f	Force produced by propeller and motor
F	A required force
g	Acceleration due to gravity ($9.81ms^{-2}$)
$H(z)$	Digital filter transfer function
m	Mass of the quadcopter
$X(z)$	Digital filter input
$Y(z)$	Digital filter output
$\mathbf{A}$	Current state prediction matrix
$\mathbf{B}$	Measurement mapping matrix
$\mathbf{K}$	Controller vector
$\mathbf{H}$	Quadcopter plant vector
$\mathbf{V}$	Actual orientation vector
$\mathbf{V}^*$	Required orientation vector
$\mathbf{X}$	Measured x-axis vector
$\mathbf{Y}$	Measured y-axis vector
$\mathbf{Z}$	Measured x-axis vector
x, y, z	The spatial dimensions
x_B, y_B, z_B	Spatial dimensions relative to body of quadcopter

Acknowledgements

I would like to thank Professor Steve Gunn for all his help and inspiration through such an interesting project. Mark Long for all his assistance with the printing of the motor housing. Jeff Hooker for his technical assistance with circuit construction. Dr Geoff Merret for accepting my request for a budget increase. My parents, Mike and Rachel Robinson, for care packages and logistics. Finally a big thanks to all my friends at university and the lads at 10 Sherborne road for keeping me distracted.

Statement of Originality

All the work herein is that of my own effort however particular resources were called upon during the completion of this project.

Technical documentation from Atmel, FTDI, Invensense and Microchip which includes recommended circuit implementations and software protocols for their products.

Component footprint models distributed with EAGLE version 6.3 with additions from Sparkfun and Adafruit libraries.

Software libraries which include FatFs from ELM, AVR Libc and the System Namespace distributed with Visual Studio 2010.

Chapter 1

Introduction

Quadcopters[1] are aerial Vertical Take Off and Landing (VTOL) vehicles that use four columns of lift to achieve flight. The craft is known for being based upon simple principles, when compared to other rotor aircraft, and has recently become more popular because of the increase in consumer processor specifications which are needed to implement the necessary control algorithms. Many open-source designs already exist for standard size quadcopters, such as Lim et al. (2012), but more recent implementations are favouring smaller scale designs as the necessary hardware overheads scale down and the efficiency of components increases.

Swarm robotics is an interesting approach to multi-agent systems. A biologically inspired swarm uses many simple robots to complete complex tasks (Mohan and Ponnambalam, 2009). Disassociating a quadcopter from the complex control and flight dynamics it can be considered as quite a simple robot which can be integrated into such a swarm; figure 1.1.

FIGURE 1.1: A formation of 20 micro quadrotors in flight. Taken from Kushleyev et al. (2012).

[1] Also known as quadrocopters or quadrotors

1.1 Flying circuit boards

A conventional quadcopter has a main body from which four arms extend to hold the motors and propellers but when the objective is to make the vehicle as small as possible this renders a traditional approach inefficient. Area overheads for the arms and motor housing not only increase dimensions but also weight. This requires larger amounts of lift to achieve stable flight therefore adding weight to the design. This vicious cycle continues until the design is beyond the constraints of the specification. In an attempt to keep copters as small as possible designs exist based around a single Printed Circuit Board (PCB) which is used as the frame as well as the medium for circuit construction. Figure 1.2 contains a prototype for the open source Crazyfile nano quadcopter which was developed throughout 2012 and went on sale as kit in February 2013. Aimed at hobbyists this is a prime example of a single board solution.

FIGURE 1.2: Crazyfile (Rev.B). Taken from Bitcraze.

1.2 Why quadcopters?

Robots are used for tasks that humans don't want to work on which can be because they are too dangerous or simply too tedious. Quadcopters have an advantage when it comes to dangerous situations because of their freedom of movement; unlike grounded robots that may encounter issues with terrain in such situations. Swarms of quadcopters have been applied to research in the field of post-disaster infrastructure repair (Alvissalim et al., 2012) along with search and rescue (Almurib et al., 2011). They can also track movements and coordinate sub-swarms of ground agents. Figure 1.3 shows a quadcopter configured to coordinate ground units using a downward facing camera and attempting to resolve the local image

plane to the ground plane which contains the sub-swarm agents. All swarms referenced in this section are developed either using bespoke agent architecture or adapted from hobbyist consumer grade quadcopters hence the need for versatile development platforms.

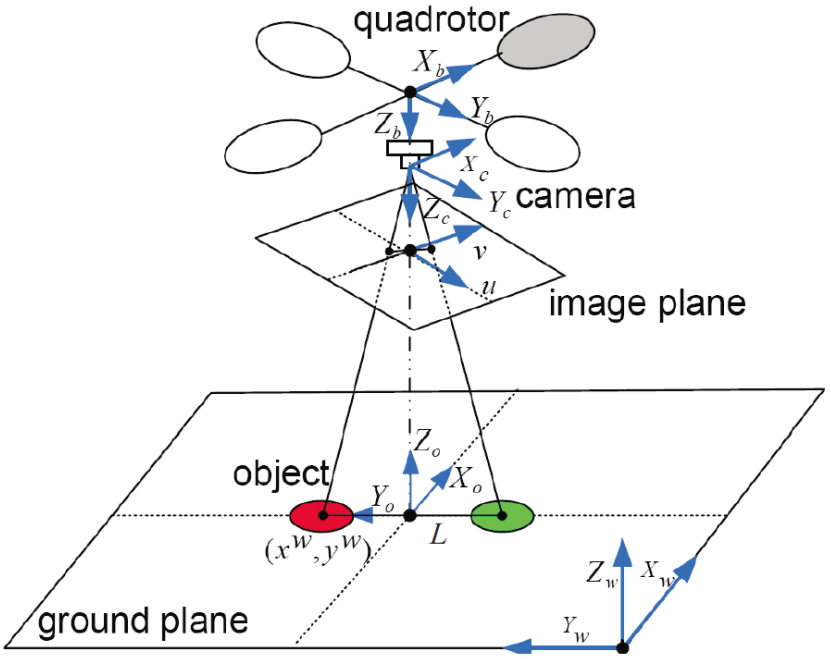


FIGURE 1.3: Frames of reference. Taken from Zhang et al. (2009)

1.3 Motivation for specification

An audience for a small scale hardware platform therefore exists in robotic swarm research. Reducing the size offers benefits such as indoor testing and a reduction in the amount of mechanical components which tends to be the majority of expenditure during such a project. This design would act as a platform for researchers to add their own hardware in the form of sensors and actuators for specific branches of swarm development. A standardised form of wireless communication should also be implemented to allow each craft to communicate not only with a base station coordinator but also with every other quadcopter in the swarm. This requires an implementation of a firmware kernel to stabilise the craft and handle any other low level functionality but still interface with user applications therefore disassociating any control or hardware concerns. The initial project brief is held in appendix A.

1.4 Contributions

Skills possessed at the beginning of this project come from a background in electronic engineering. This experience was repeatedly drawn upon throughout completion but also built upon by engaging in new challenges. The most tasking of which was to design a two layer PCB of non-trivial dimensions using EAGLE Computer Aided Design (CAD) software. Previous experience helped with designing the system architecture but the process of PCB manufacture itself yielded great learning outcomes. Implementation and testing revealed successes and failures which increased understanding of practical electronics. This will assuredly improve performance in future engineering endeavours.

The project also required a multidisciplinary skill set which was not completely present from the start. Physics, control theory and mechanical engineering were all areas which lacked the same standard of background experience as that in electronics. As the project evolved so did the need for proficiency. Three Dimensional (3D) printing using Autodesk CAD software enabled a new competence which was successfully integrated with the other mechanics of the design. Physics and control theory were combined to produce design models and increase understanding of the task.

Chapter 2

Background research

2.1 Quadcopter theory

Four propellers attached to motors create vertical lift which allows the vehicle to move off the ground. Figure 2.1 shows how the aircraft is arranged. Two pairs of counter rotating propellers are placed opposite each other and then connected by an internal frame. The upward force generated by the motors is variable and therefore allows control of the three angles of pitch (θ), roll (ϕ) and yaw (ψ); acting in the xz, yz and xy planes respectively.

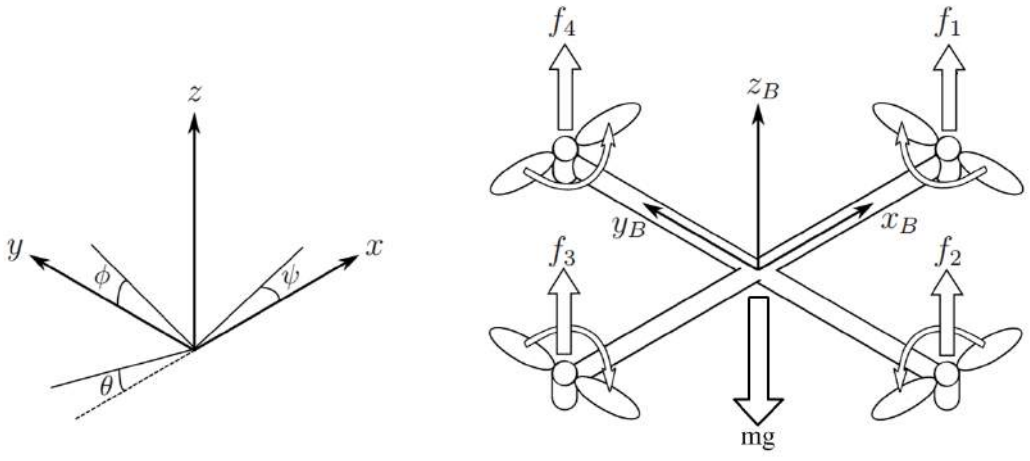


FIGURE 2.1: The inertial and body frames of a quadcopter. Adapted from Luukkonen (2011)

2.1.1 Flight

The hardware design process can only constrain the variables in figure 2.1 to a maximum. The control software will manipulate these such as to achieve stable flight but to become airborne it obvious that equation (2.1) must be satisfied. Where the total force of the motors negates the overall weight. Assuming all four motor and propeller combinations are the same this is just four times the individual thrust output. Practically however these constraints are not feasible. The quadcopter moves by altering angles and therefore reducing the resultant force against gravity. The general rule of thumb, recommended by Büchi (2011), is to have a possible thrust of at least twice the weight of the quadcopter.

$$f_1 + f_2 + f_3 + f_4 = 4f > mg \tag{2.1}$$

Considering the copter only in the xz plane means resultant forces now rely upon the pitch. To achieve movement in the positive x direction the angle θ should be positive and in the trivial case the z axis position will remain constant. The required angle is achieved by changing f_1 and f_3 then by using θ the forces along each axis can be resolved; as described by figure 2.2 and equations (2.2), (2.3) and (2.4). Considering the known mass and the required force along the x axis yields the values of θ and F needed to achieve level flight; (2.5) and (2.6).

$$F = F_1 + F_2 + F_3 + F_4 \tag{2.2}$$

$$F_a = F\cos(\theta) \tag{2.3}$$

$$F_b = F\sin(\theta) \tag{2.4}$$

$$F = F_b + mg \tag{2.5}$$

$$\theta = \arctan\frac{F_b}{mg} \tag{2.6}$$

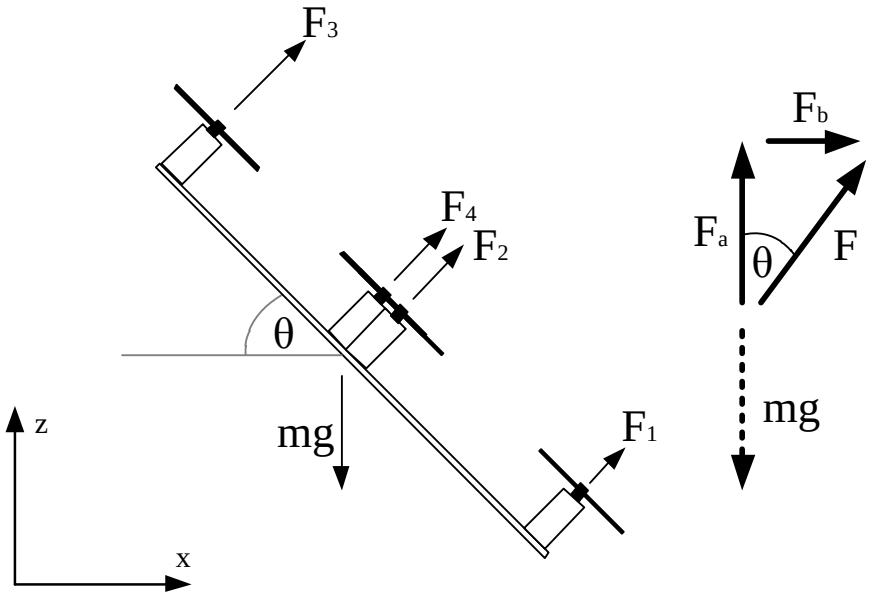
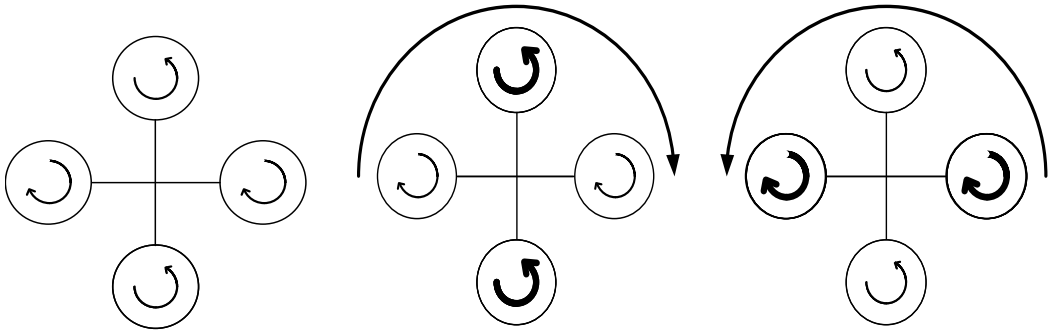


FIGURE 2.2: Resolving forces

This means the vehicle proceeds in the positive x direction until it reaches a maximum speed when the force of the air resistance is equal to that of F_b. The forces produced will be contested by the air resistance but sensor feedback will take care to retain the pitch. The two forces not in use (F_4 and F_2) can be used to increase the overall force but also control the roll of the aircraft. This gives the copter the ability to move throughout three dimensions and also, by resolving the total angular velocity of the propellers, allows the vehicle to rotate; figure 2.3.



(a) No rotation (b) Clockwise rotation (c) Anti-clockwise rotation

FIGURE 2.3: Rotation in xy plane

2.2 Sensor data

The critical function of a quadcopter is to measure the two angles of pitch and roll. The main sensors on the market which can be used as angle measurement devices are accelerometers and gyroscopes. These come in Microelectromechanical System (MEMS) Integrated Circuit (IC) packages which can easily be integrated into an electronic design; see figure 2.4. In theory these could both be used individually to calculate the orientation of the object which the package is mounted upon but in practice the sensors are noisy so both devices are used to improve quality.

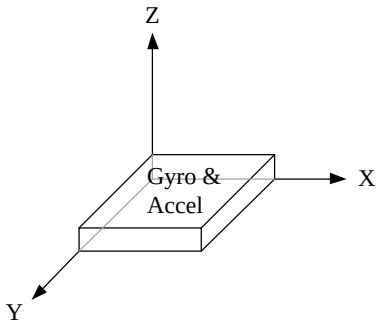


FIGURE 2.4: An accelerometer and gyroscope package with axis notation

2.2.1 Accelerometer

Three spatially orthogonal vectors of acceleration are produced by the device ($\mathbf{X_a}$, $\mathbf{Y_a}$ and $\mathbf{Z_a}$) and assuming only gravity is acting upon them these vectors will resolve to give a single vector of magnitude $1g$ in the direction of gravity; as described in equation (2.7). The dot product of the gravitational vector with each of the respective composite vectors will now yield the angle between gravity and the direction in which said vector points. Equations (2.8) and (2.9) shows how this can be used to calculate pitch and roll respectively.

$$\mathbf{X_a} + \mathbf{Y_a} + \mathbf{Z_a} = \mathbf{g} \tag{2.7}$$

$$\theta = \arccos \frac{\mathbf{X_a}.(\mathbf{X_a} + \mathbf{Y_a} + \mathbf{Z_a})}{||\mathbf{X_a}||\ ||(\mathbf{X_a} + \mathbf{Y_a} + \mathbf{Z_a})||} = \arccos \frac{\mathbf{X_a}.\mathbf{g}}{||\mathbf{X_a}||\ ||\mathbf{g}||} \tag{2.8}$$

$$\phi = \arccos \frac{\mathbf{Y_a}.(\mathbf{X_a} + \mathbf{Y_a} + \mathbf{Z_a})}{||\mathbf{Y_a}||\ ||(\mathbf{X_a} + \mathbf{Y_a} + \mathbf{Z_a})||} = \arccos \frac{\mathbf{Y_a}.\mathbf{g}}{||\mathbf{Y_a}||\ ||\mathbf{g}||} \tag{2.9}$$

2.2.2 Gyroscope

As with the accelerometer the gyroscope produces three orthogonal vectors ($\mathbf{X_g}$, $\mathbf{Y_g}$ and $\mathbf{Z_g}$) but in this case each one accounts for the angular velocity which leads to equation (2.10). The acceleration of each vector at time T. The constant term produced by integration can negated if at time 0 the package is not accelerating in any direction.

$$\mathbf{i_a} = \int_0^T \mathbf{i_g}\, dt, \;\; \mathbf{i_a} = 0|_{t=0}, \;\; \mathbf{i} \in \mathbf{X}, \mathbf{Y}, \mathbf{Z} \tag{2.10}$$

2.2.3 Application

In practice this theory is not directly implementable. The only data produced by the accelerometer is magnitude so it can no longer be considered in vector form. The approach is to find the magnitude of the first two vectors in a plane orthogonal to third vector, shown in figure 2.5, then by using standard trigonometry equation (2.11) follows.

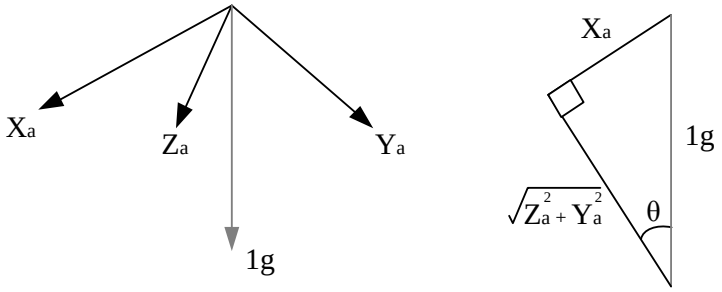


FIGURE 2.5: Direct calculation of pitch from magnitude only data

$$\theta = \arctan \frac{X_a}{\sqrt{Y_a^2 + Z_a^2}} \tag{2.11}$$

The gyroscope initialisation can also be an issue because gravity won't allow for the accelerometer to read zero in all directions. This can be fixed by placing the package on a flat surface and taking the constant offset magnitude, for the axis orthogonal to the surface, as $1g$. The surface would have to be truly perpendicular to gravity which of course is impossible. This problem along with other noise issues is addressed by the control theory.

2.3 Control theory

The open-loop control topology of the system is contained in figure 2.6 and as discussed in section 2.2 sensors are used to measure the attitude and angular velocity of the attitude. There is another system to consider now, the quadcopter itself, how it responds to change in the motors and how these changes can be measured accurately. This will be known as the quadcopter plant (**H**).

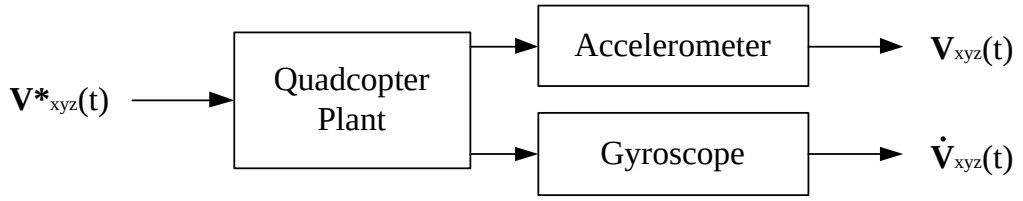


FIGURE 2.6: Open-loop control

The system acts upon a required three axis input vector ($\mathbf{V*}_{xyz}$), as shown in figure 2.6, where the two output vectors are from each of the sensors. The closed loop block diagram, figure 2.7, feeds the two sensor output vectors back which are subtracted from the input to form an error vector. The two controllers $\mathbf{K_1}$ and $\mathbf{K_2}$ are configured such as to minimise said error vector. These can take the form of standard Proportional-Integral-Derivative (PID) controllers. The unknown block contains the process to merge the acquired data. The question remains how to incorporate the two measurements. A nested loop is one option or feedback can be contained in just one path once a good estimation of the orientation is calculated. The two approaches considered in sections 2.3.1 and 2.3.2 should be implemented chronologically subject to overall progress.

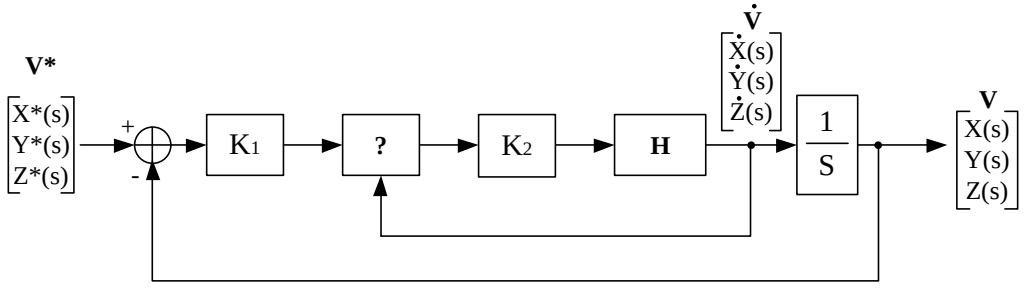


FIGURE 2.7: Closed-loop control. Adapted from Sa and Corke (2011)

2.3.1 Complementary filter

The most basic approach to achieving an improved output from the accelerometer and gyroscope is to use a complementary filter. It removes some of the noise which plagues the raw data and is much less complex than a Kalman filter (see section 2.3.2) which of course means it requires less processing time. The filter however just cuts out information to be processed as it is assumed to contain mainly noise but in practice this actually means some useful information is also being disregarded. Figure 2.8 contains an overview of the filter. A gyroscope and accelerometer reading is taken and after the gyroscope reading is integrated the data is cut using two filters then additively combined. This data is then ready to be processed which reveal the angles of orientation.

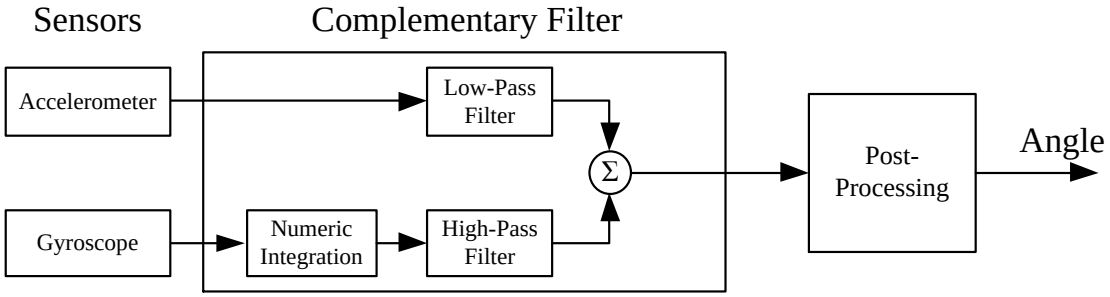


FIGURE 2.8: Basic complementary filter. Adapted from Colton (2007)

The two filters remove the high frequency and low frequency components of the accelerometer and gyroscope respectively. This results in the recombination shown in figure 2.9. The angular frequency, ω_c, is used to tune the response in order to remove as much noise as possible but retain useful information.

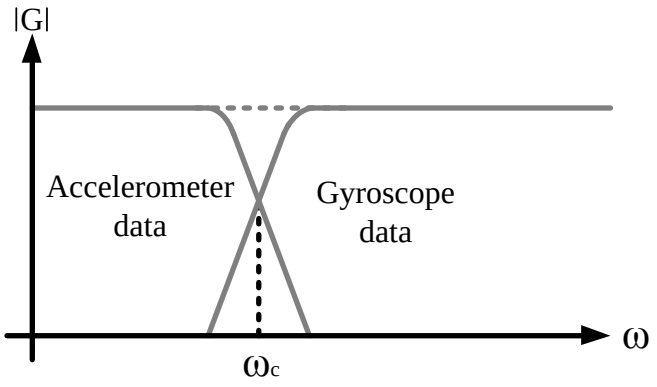


FIGURE 2.9: Recombination of sensor data

2.3.2 Kalman filtering

The Kalman filter (Kalman, 1960) is capable of compensating for noisy sensor readings by using a model which can predict the behaviour of that which is to be measured. This prediction along with the actual noisy measurements provides meaningful data. Using equation (2.12) the filter can be configured as shown in figure 2.10 to estimate a discrete time controlled process. The state $x \in \Re^n$ is the estimation vector at step k and $z \in \Re^n$ is the measurement. Variables v and w represent the inputs to the process at each step along with Additive White Gaussian Noise (AWGN). The matrix $\mathbf{A}$ is used to predict the current state from the last state and $\mathbf{B}$ can be used to map the current state to a measurement.

$$x_k = \mathbf{A}x_{k-1} + w_{k-1}\,, \quad z_k = \mathbf{B}x_k + v_k \tag{2.12}$$

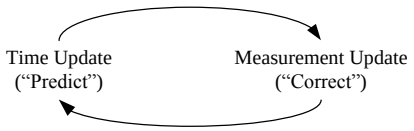


FIGURE 2.10: The ongoing discrete Kalman filter cycle. The time update projects the current state estimate ahead in time. The measurement update adjusts the projected estimate by an actual measurement at that time. Taken from Welch and Bishop (2001).

A filter example is shown in figure 2.11. An arbitrary constant voltage is chosen and measured with noise that is modelled as AWGN with a standard deviation of 0.1 around this constant value. The filter output begins far from the actual value but slowly converges towards it as iterations increase.

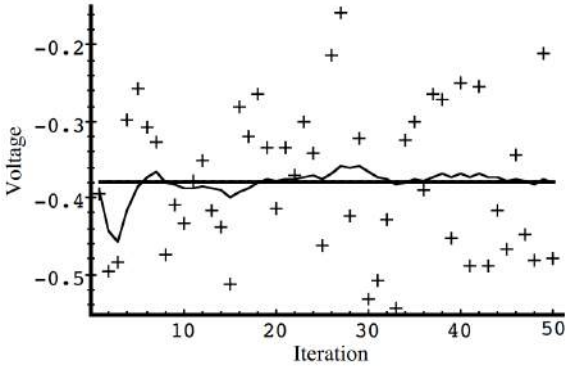


FIGURE 2.11: The true value of the random constant is given by the solid line, the noisy measurements by the cross marks, and the filter estimate by the remaining curve. Taken from Welch and Bishop (2001).

2.4 Swarm robotics

An intelligent agent consists of an agent program which produces an output based on some input, defined by its behaviour, but also an architecture on which the program can be executed. This architecture is composed of sensors and actuators which allow the agent to interact with its surrounding environment. The program and architecture must both be present to form an intelligent agent. Figure 2.12 contains a simple reflex agent. The agent can take in information from the environment and then act upon this information due to a set of condition-action rules.

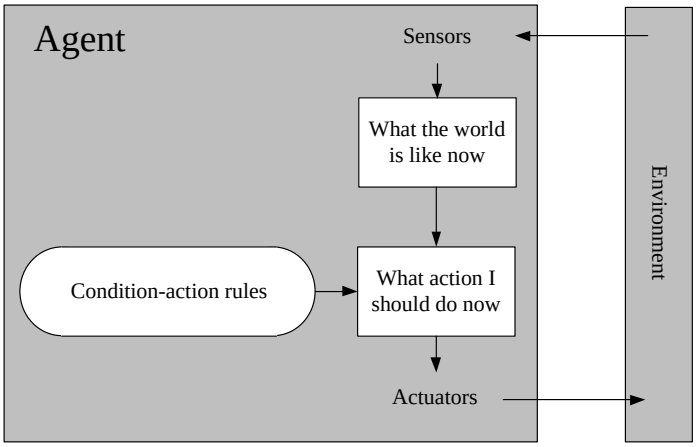


FIGURE 2.12: Schematic diagram of a simple reflex agent. Taken from Russell and Norvig (2010)

A group of individual autonomous agents working together is called a swarm. Alone each is quite simple but when working towards a common goal the agents, through self organisation, can carry out tasks more efficiently than a single more complex agent. The understanding of swarm interaction is largely inspired by biology which contains many natural swarms such as ants or birds. These swarms are not controlled by a single coordinator but each agent in the swarm has to ability to communicate with other agents in the vicinity. So by understanding this behaviour robotic swarms can be programmed to solve different problems (Mohan and Ponnambalam, 2009).

Chapter 3

Design

3.1 Optimal circuit board configuration

Previous designs for small scale quadcopters that use a PCB as the frame simply attach the motor on top of the board. This is a reasonable solution as it reduces the frame size as much as possible while retaining the distance between the propellers. The solution proposed here differs because the propellers are enclosed by the circuit board, similar to larger scale designs, and the propellers placed in the same plane. The advantages include increasing the space available for components to be placed and protecting the propellers from collisions which of course is a large risk when tuning a swarm of quadcopters.

The three configuration options considered are titled circular, tapered square and clover. Appendix C contains detailed definitions of each layout, the modelling equations for usable/total area and the graphs used for comparison between configurations with $\mu = 5mm$ and $\delta = 15mm$. This however is not true for the clover layout where the layout definition is contained in figure 3.1 and comparison graphs in figure 3.2. The clover layout was chosen as the configuration to implement because of higher area efficiency for lower radius propellers. The center of each layout provides the core space for most of the circuitry to be housed while the four areas between the propeller holes can be used to house circuitry that does not require many connections. The rim of the PCB has been excluded as possible placement area to prevent damage to components when in flight.

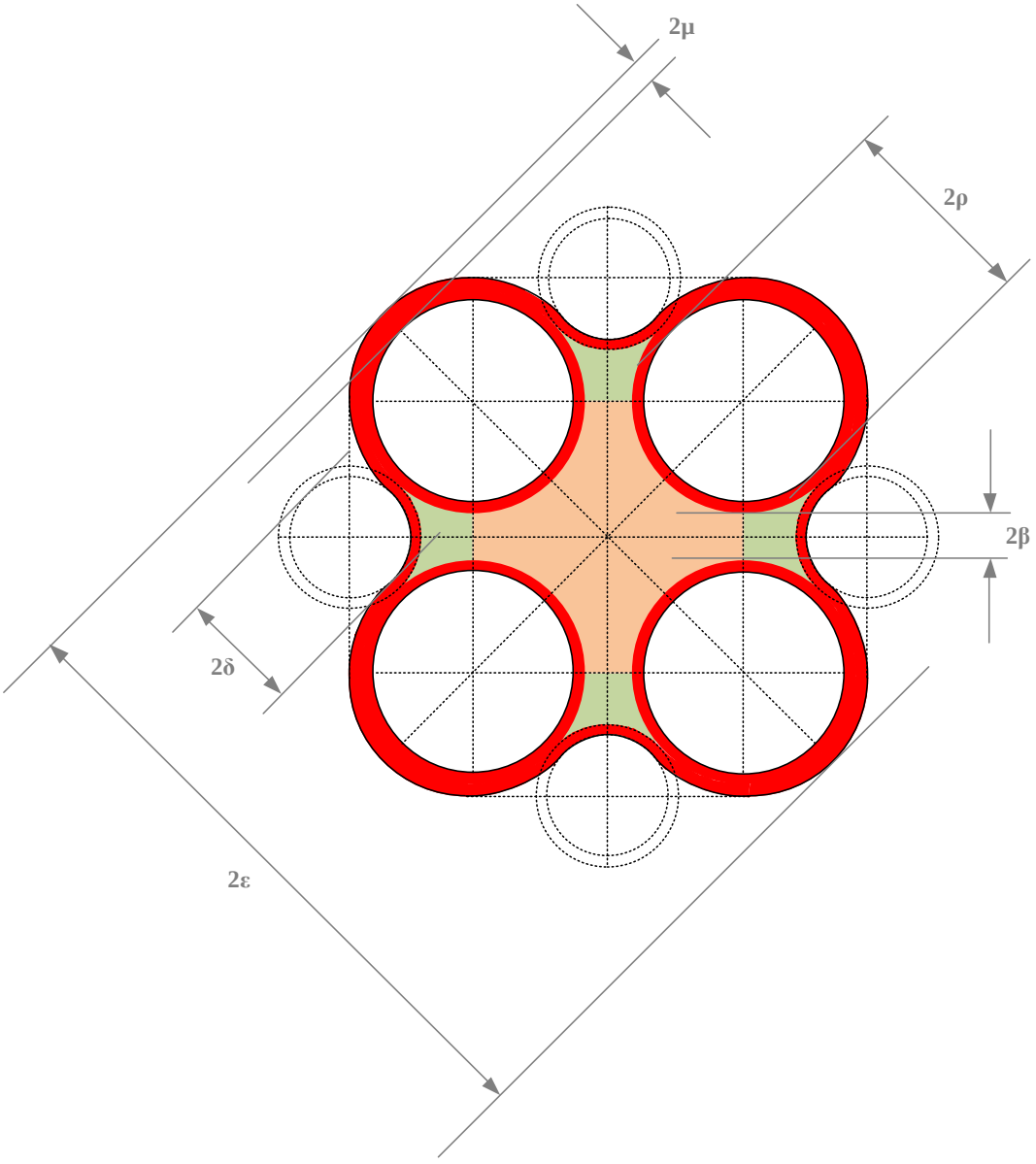


FIGURE 3.1: Clover layout definition

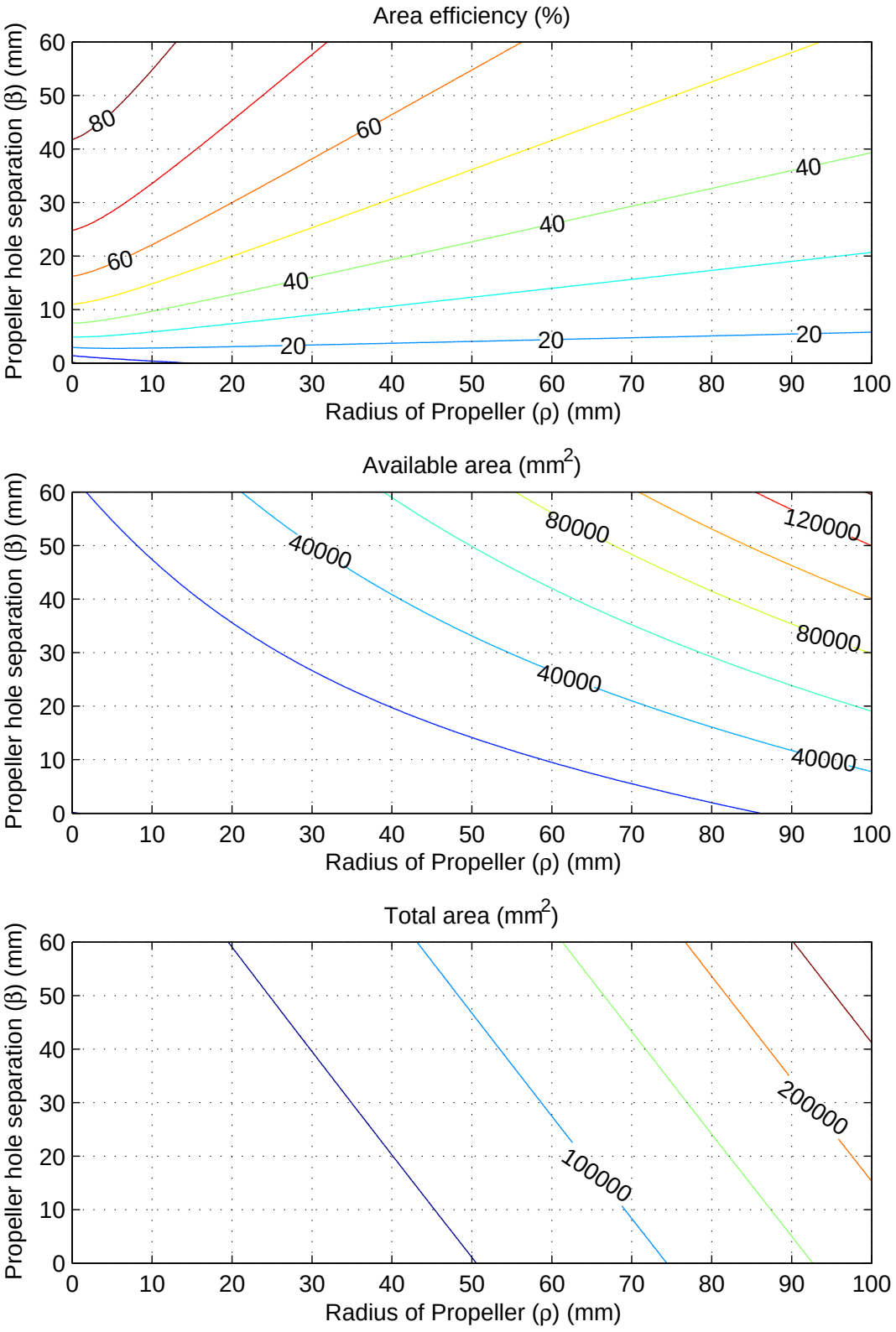


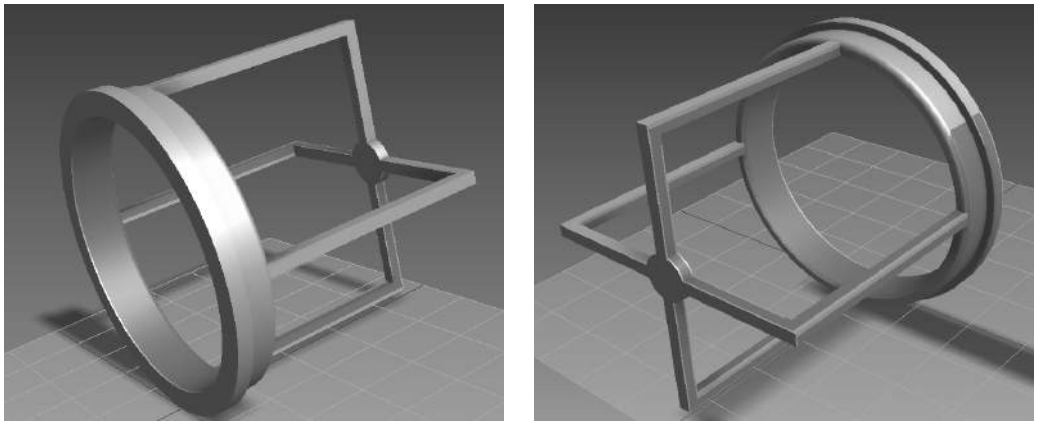
FIGURE 3.2: Clover layout scaling

3.2 Motors and propellers

The motors used drive the propellers and therefore produce thrust must be efficient, compact and powerful. The best solution is Brushless Direct Current (BLDC) motors which meet all these criterion (Bai, 2011). These typically require a three phase drive system which increases the system complexity. The motors shaft can then fitted with an appropriate propeller which in this case needs to be matched with a counter rotating partner therefore reducing the overall rotational force. The challenge is to source propellers that provide enough lift for the whole design. Larger propellers will increase the overall weight by increasing the propeller hole radius hence PCB area therefore more lift is required.

3.3 Motor housing

The propellers must be situated in the same plane as the PCB but this means, when using conventional motors, a section of each motor will protrude from the PCB. The dimension specification is subject to large variance due to PCB options therefore this part will be custom fabricated using a 3D printer. Figure 3.3 contains the first idea for mounting the motor to a PCB. A circular rim is designed to slot into a hole in the PCB of a slightly larger radius radius which is then held in place by a lip around the edge of the rim. The base of a motor is then held in place by the four struts attached to the rim.

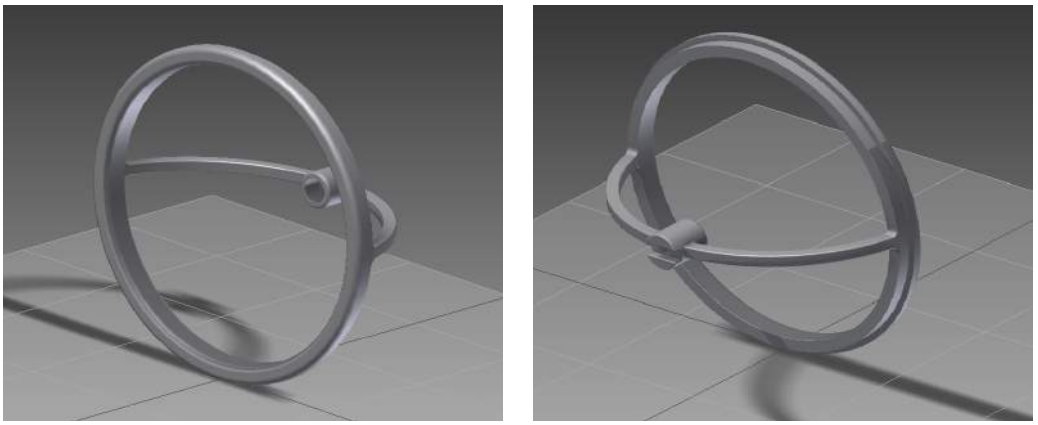


(a) Front side (b) Back side

FIGURE 3.3: Motor housing (Revision A)

This design is revised in figure 3.4 where only two of the arms remain and the base now contains a recess to fit the motor along with raising the whole design from the

ground with a small footing. Perpendicular edges are also removed from the whole design apart from where the rim is designed to sit flush against the PCB edge. The whole unit can be tuned to fit different size PCBs and motors as required. Rapid revisions in design concepts are made possible by using Autodesk 3D CAD software.



(a) Front side (b) Back side

FIGURE 3.4: Motor housing (Revision B)

3.4 Battery

The quadcopter requires a local power supply which of course can be produced by a battery. The battery needs to be of low weight to be incorporated into the design. The lifetime of the battery would ideally be in order of hours but this is not a necessity for a first prototype. Lithium Polymer (LiPo) batteries offer the best energy to weight ratio on the market and suffer from a very low self discharge so for that reason they are popular among model aircraft hobbyists (RMAC).

Linear least squares regression using obtainable LiPo battery data produces an energy to mass ratio of 423J/g which is shown in figure 3.5 and calculated using equation (3.1). The chosen battery will be taken from the sub 50g selection depending on system mass and required lifetime. The potential difference of such batteries is 3.7V which is enough to provide a standard regulated 3.3V power supply.

$$Energy\ to\ mass\ ratio = \frac{Voltage \times Charge}{Mass} \tag{3.1}$$

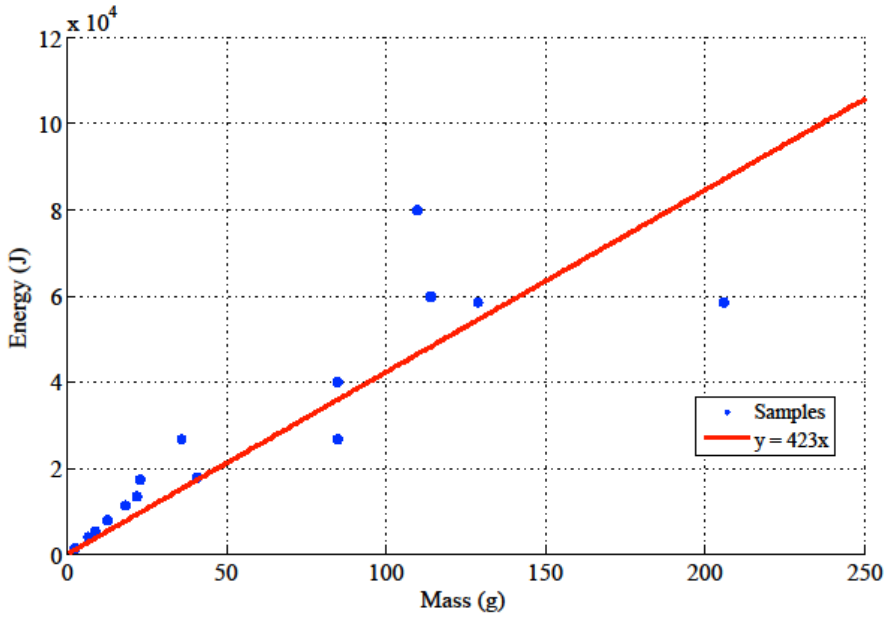


FIGURE 3.5: A comparison of lithium polymer batteries on the market. Raw data held in table D.1

3.5 System architecture

The backbone of the system is a processor that can read sensor data and act upon this data by adjusting the speed at which connected motors rotate thus altering the thrust because of the attached propellers (figure 3.6). The processor adjusts to given sensor readings as per its program and the cycle continues in order to maintain flight.

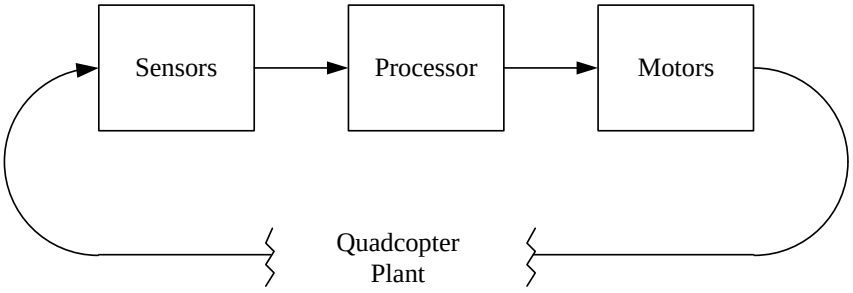


FIGURE 3.6: The simplest manifestation of the required system

The basic solution can be improved by adding a wireless link. The decision was taken not to interface the transceiver directly with the existing processor in an attempt to reduce its functional responsibilities therefore freeing up processor time for a tighter control loop. The transceiver module can only be accessed via another

processor allowing data to be buffered or manipulated before being sent to/received from the actual transceiver. Figure 3.7 contains the ideal data flow within the system where the feedback is now contained by the physical environment. Additional changes include separating the sensor array into an accelerometer and gyroscope (as discussed in section 2.2), separating the motors into four modules and adding an expansion port. The expansion is intended to encourage hardware alterations by making it as easy as possible to interface with the processor.

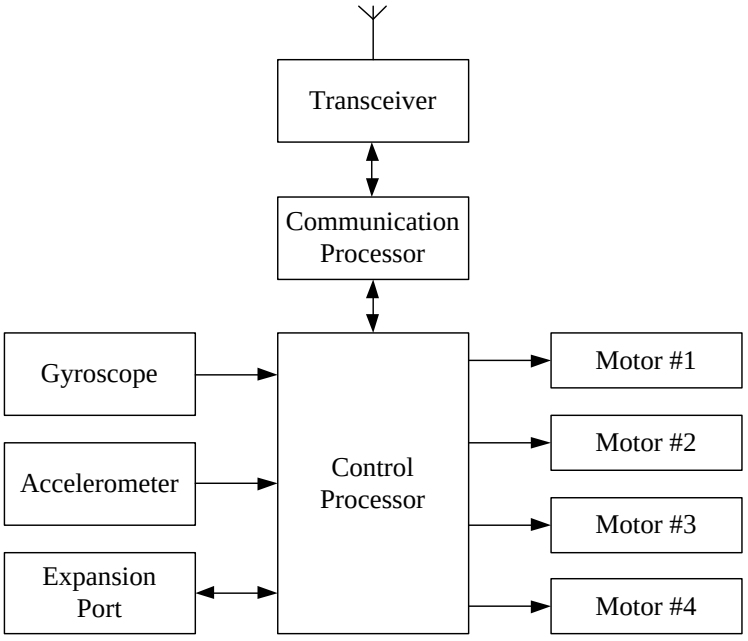


FIGURE 3.7: Complete ideal data flow within the system

3.6 Communications

As specified the communications protocol needs to be capable of supporting point-to-point through to fully connected topologies. The ZigBee[1] family of transceivers are best suited for this application. The technology is of low complexity and has become increasingly popular in multi-agent systems (Zhang and Gao, 2011). It supports a wide range of topologies, is widely available and built upon the IEEE:802.15.4 standard which is aimed at integration into portable systems with little to no battery (energy harvesting) consumption. The short range is the main drawback but in this application that is not an issue.

[1]This refers to the waggle dance of honey bees; a natural swarm.

3.7 Base station

The base station must contain a human user interface to communicate with an individual quadcopter or to coordinate a swarm. Initially the software was developed to communicate with a single craft. This can be achieved by using a general purpose Personal Computer (PC) with the appropriate software and hardware additions. The application should provide real-time information about the state of the quadcopter but also log this data for recall and analysis. The chosen language to develop this Graphical User Interface (GUI) is C# due to experience and the ability to produce graphical programs for Windows operating systems.

An initial design used dummy data produced by the user to simulate the type of data which could be expected from the implemented system. Figure 3.8 contains the initial concepts for the base station with two sliders used to generate dummy data. Listing E.1 contains the first three entries in the log file which is produced by the screen capture in figure 3.8. Entries are made every 100 milliseconds but to reduce the file size only changes in the data are recorded. A possible warning system is used to recorded when the data may have broken a set threshold where log data is human readable at this prototyping stage. The first chart is used to plot values of inclination against time. The second chart plots the two values against each other so a level recording would be at the origin. These are somewhat analogous of the type of instrumentation found in aircraft cockpits.

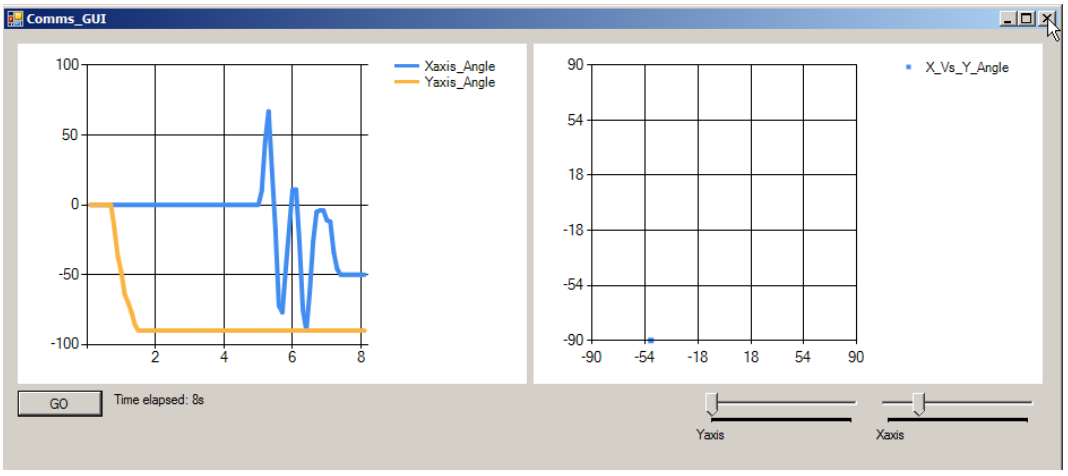


FIGURE 3.8: An initial prototype of the base station user interface with dummy data

3.8 Project planning

As an individual project the planning was orientated around a single person workload but ordered such as to increase demonstrable functionality of the end product. This is based around the spiral model proposed by Boehm (1988). Appendix I contains Gantt charts which plot the activities carried out during the project. The first (figure I.1) contains the initial activities; these were not planned from the start as research and design opened many new areas of work but towards the end of this period the project became more structured. This chart therefore serves as a review of natural beginnings of the project. The second chart produced (figure I.2) is a speculation on the coming events in the project, which are mainly down to implementation, and therefore a working guide. The activities here are largely dependent on the reception of the manufactured PCB.

3.8.1 Risk management

To minimise project risks they must first be identified. The risk register in table 3.1 contains the main risks along with an estimation of likelihood and impact to the project. These two properties are multiplied to produce a risk score which allows them to be addressed in order.

The project also requires development of software and CAD files. This inherently places work at risk of damage by corrupted data or loss by faulty storage mediums. Files are therefore periodically backed up from a local file store to a remote server which is held in a different building. This greatly reduces the risk of simultaneous damage from either device malfunctioning or local environmental issues.

Risk	**Likelihood (1 - 5)**	**Impact (1 - 5)**	**Score (1 - 25)**	**Mitigation**	**Contingency**
Quadcopter doesn't fly	1	5	5	Modelling and simulation	Prove concept and design a 2^{nd} revision
Rev.A PCB doesn't function	3	2	6	Design rule checks and 3^{rd} party reviews	Review design and then submit rev.B
Rev.B PCB doesn't function	1	5	5	Thorough evaluation of rev.A	Prove concept theoretically
Quadcopter is damaged	2	5	10	Test in crash proof environment	Order more than one PCB and make sure components are in large reserves

Table 3.1: Risk register

Chapter 4

Implementation

4.1 System components

Using the system architecture described in section 3.5 the design was brought from theory into practice by sourcing suitable components. Some of these components introduced more complexity into the design by requiring additional functionality but some extra features are included just to make the system easier to test. Figure 4.1 contains an overview of the system implementation. A list of all major components is contained in table F.1.

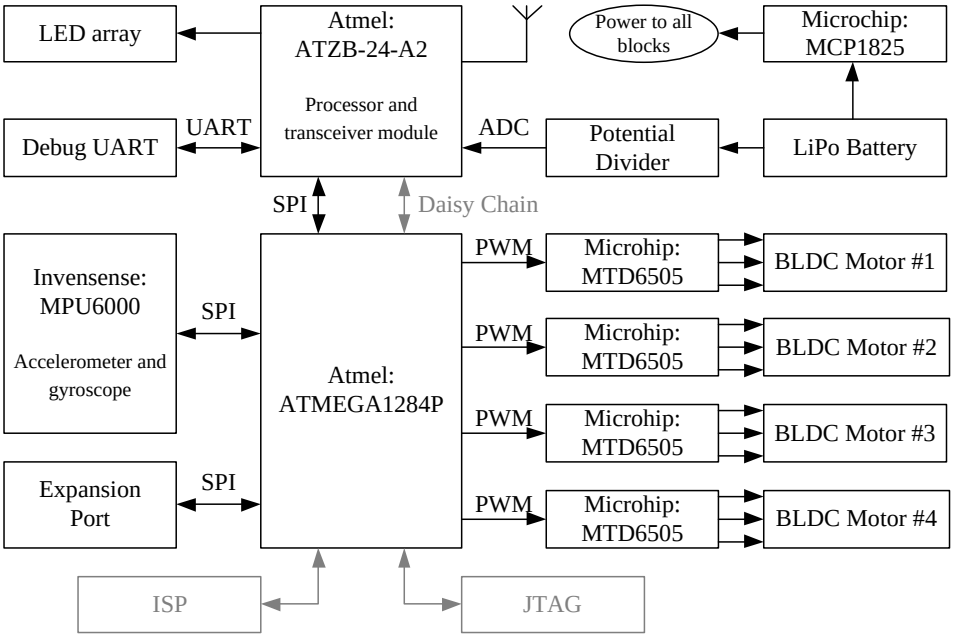


FIGURE 4.1: Implemented system architecture

The processors required for controlling the craft and buffering the transceiver communication are naturally implemented as microcontrollers. As the firmware to be implemented is not yet available programming interfaces are required so they can be programmed in system. These take the form a Joint Test Action Group (JTAG) interface (2×5 0.1 inch pin header) which is daisy chained to program both microcontrollers and an In-System Programming (ISP) interface (2×3 0.1 inch pin header) connected only to the ATMEGA1284P. The reason for the superfluous interface is because the ISP connections are direct and unlikely to fail whereas JTAG relies on more connections and both microcontrollers functioning as intended. This way in the event of a failure in the JTAG interface the system retains the ability to be programmed partially.

The functionality of the transceiver microcontroller has increased. A connection to the battery via a potential divider can alert the system if the voltage drops such that it might risk losing the 3.3V supply rails. A Universal Asynchronous Receiver/Transmitter (UART) connection for assistance with debugging and data transfer has been added. A Light Emitting Diode (LED) array which can be used as immediate visual feedback as to the status of the system has been connected. Using a single module instead of two separate ICs has also reduced the size of this section of the design and complexity in its integration. A dual chip antenna is also included as part of the ATZB-24-A2 package which removes the need to design and implement this section. The modules contains an ATMEGA1281 microcontroller and a AT86RF230 2.4GHz ZigBee transceiver.

Driver ICs are used to bridge the connection between the microcontroller and the motors. These are MTD6505 drivers from Microchip designed to drive BLDC fans for laptops. The motors require a 3-phase sinusoidal signal and while this is possible to produce with the microcontroller it is simpler to use the ICs and also less computational complexity is required from the control processor.

The two sensors required are available in the same package which reduces complexity and area of the design. The chosen device is the Invensense MPU6000 which is a 3-axis gyroscope and accelerometer that produces 16-bit signed values for each of the six sensors. The range of which is user programmable to be traded off for resolution. This chip is chosen because it uses a Serial Peripheral Interface (SPI) interface which is simpler and faster than the alternative Inter-Integrated Circuit (I^2C) interface model because of the burst read function. The chip also contains a temperature sensor and can be upgraded with an additional magnetometer IC.

4.2 Motor and propeller selection

As mentioned in section 2.1 a quadcopter requires two sets of counter rotating propellers. This is a simple component to source for a standard size multi-rotor aircraft but for a small scale design the choices are rather limited. This provoked an initial approach of 3D printing the propellers which of course requires no supply chain and could be matched to the exact specifications of the PCB and motor. This however did not prove feasible due to unforeseen complexities in the process of modelling propellers (discussed further in section 7.3).

The chosen propellers are available as spares for a model aircraft and have a diameter of $44mm$. A full set is contained in figure 4.2 which also shows why the component must be mirrored to achieve counter rotation. These are then paired with the motor shown in figure 4.3 which is a 3 phase BLDC motor.

FIGURE 4.2: Chosen propellers. Clockwise rotation on the left and anti-clockwise on the right

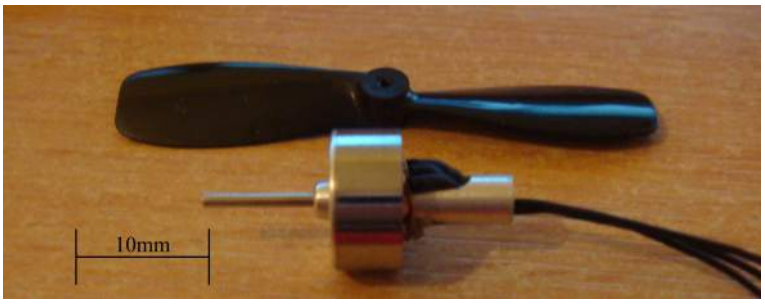


FIGURE 4.3: Motor and propeller combination

4.3 Quadcopter circuit board design

The entire system described in section 4.1 and additional support components must be placed on a board which adheres to the specification for a clover layout in figure 3.1. There are four variables to adjust which include the propeller radius (ρ), propeller hole separation (β), safety rim width (μ) and the recession radius (δ). The propellers have a radius of $22mm$ but there must be additional space for the motor housing to be attached and air to pass though therefore a suitable overcompensation of $30mm$ leaves a gap of $8mm$ to place the housing and negate any edge effects. A propeller hole separation of $8mm$ allows twice that for the transceiver module to fit between. Setting the safety rim proves difficult for a first revision because without crash testing the structural integrity cannot be measured; it is therefore best to overcompensate with a value of $2.5mm$. The recession radius is defined by the amount of components and the central section provides more than enough space to set this to $9mm$. These values are contained in table 4.1 which when substituted into equations (C.5) and (C.6) yields a usable area of $3001mm^2$ and a total area of $5454mm^2$ which is an approximate area efficiency of 55%.

Constraint	**Symbol**	**Value**
Propeller hole separation	β	8mm
Propeller radius	δ	30mm
Safety rim width	ρ	2.5mm
Recession radius	μ	9mm

Table 4.1: Chosen constraints for clover circuit board layout

4.3.1 Component layout

The center of gravity must ideally remain at the center of quadcopter; equidistant from all four propellers. To achieve this components of equal weight are placed opposite each other in an attempt to counter balance any turning force about the center of the PCB. This however must be compared against placement due to functionality; such as programming headers which need to remain accessible. Table 4.2 contains placement reasoning of clusters of components.

4.3.2 Additional features

The PCB contains some additional features intended for either expansion on the design or for testing purposes. The two headers spaced either side of the center

Component(s)	**Placement reasoning**
ATMEGA1284P, 12MHz crystal, JST header, MCP1825, and reset switch	Placed on the western side of the board to counter balance the transceiver module.
Transceiver	Placed as far as possible to the eastern side of the board to prevent interference with the antennas on the module.
ISP Header	Placed on the northern side of the board to counter balance the JTAG header on the southern side.
JTAG Header	Placed on the southern side of the board to counter balance the ISP header on the northern side.
LEDs	In the four corners of the board to make more visible and this part of the board would be used by nothing else otherwise as it is too thin. The safety rim is ignored because of the non-critical nature of these components.

Table 4.2: Layout reasoning

of the board are designed to house a child board via two 2×8 socket headers. The distance between the center of these two headers is $33.02mm$ (13×0.1 inch spaces). As the spacing grid is standard prototyping board can be used.

There is also a silkscreen on both the top and bottom layers. This contains information on signals and components. In addition this contains information to warn the user of moving blades, identify LEDs and revision data. The final piece of information segments the board into four using cardinal directions as an analogy. A ground plane polygon is flooded throughout the board to reduce ground bounce. This can interfere with the radio communications so it is only spread around the side of the transceiver which doesn't contain the chip antennas.

Small holes are placed between the propellers to use during control tuning; cord can be fixed to the board and then elevated which allows for safe balance testing. Capacitors and resistors used in the design are implemented in 0805 and 1206 packages respectively. The LEDs chosen are also in 1206 dimensions. These consume a small amount of area but are not so small as to affect construction by hand.

4.4 Quadcopter manufacture

The PCB (which became known as Archaeopteryx[1]) was fabricated as a two layer, 1.6*mm* thick board composed of FR-4 grade epoxy laminate sheets by a company called PCBCART. All design files for this board are held in appendix F.1. Two scans of the top and bottom of the PCB are contained figure 4.4 and 4.5 respectively.

FIGURE 4.4: Fabricated board. Top side.

[1]A late Jurassic period bird that may have developed a form of four winged flight (Longrich (2006)). The name is also pronounced with a "copter" sound (*ar-kee-op-tr-iks).*

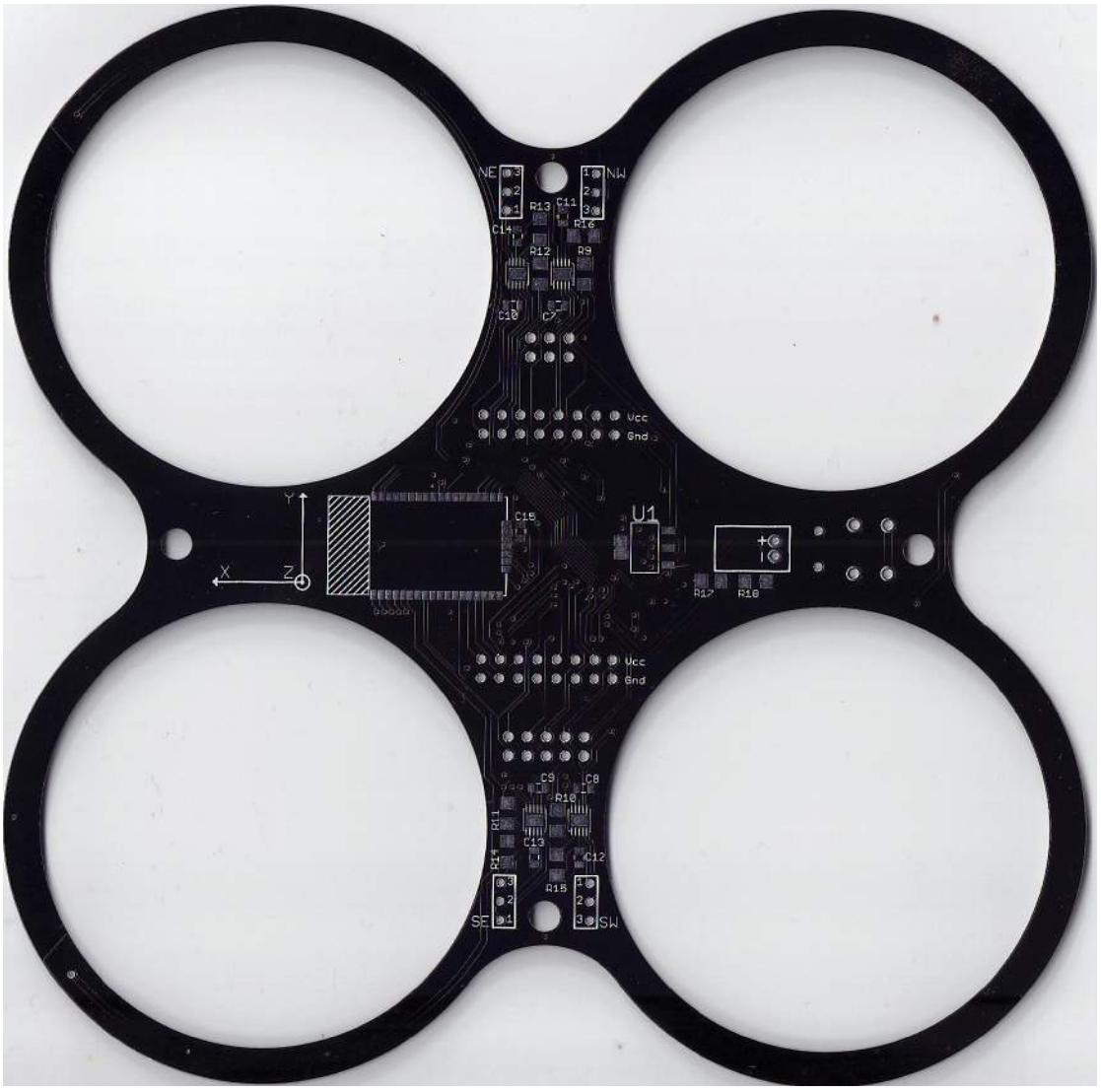
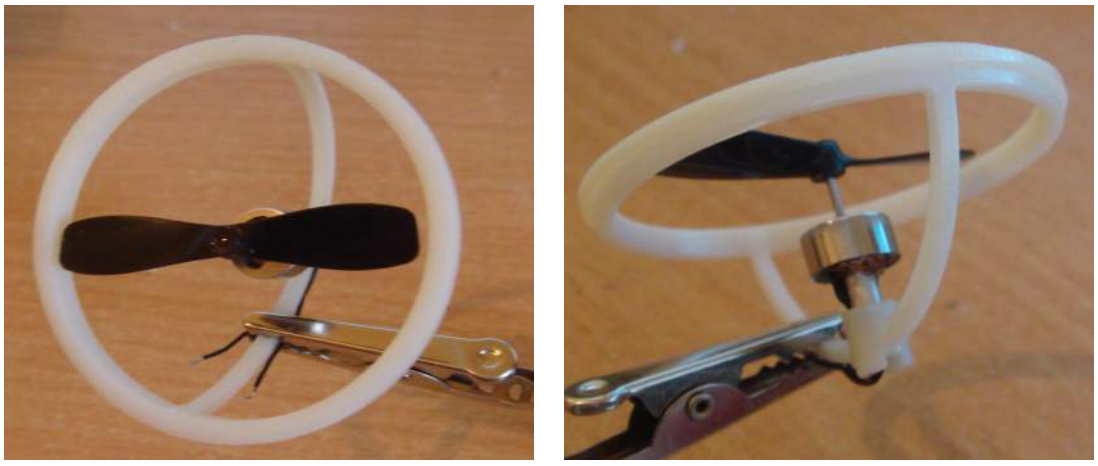


FIGURE 4.5: Fabricated board. Bottom side.

4.5 Motor housing fabrication

The first prototype of the housing to be fabricated was revision B which can be seen holding a motor and propeller combination in figure 4.6. The scale of this prototype is based around only the motor and its purpose is to test the validity of the design once fabricated. This revision proved successful because it retains the motor in place and has enough room for the propeller to move freely. The design does however have quite a high profile. It can be seen in figure 4.6(b) that the propeller is placed towards the end of the motor shaft to keep it in line with the rim. The struts also proved to provide too little support to the center.

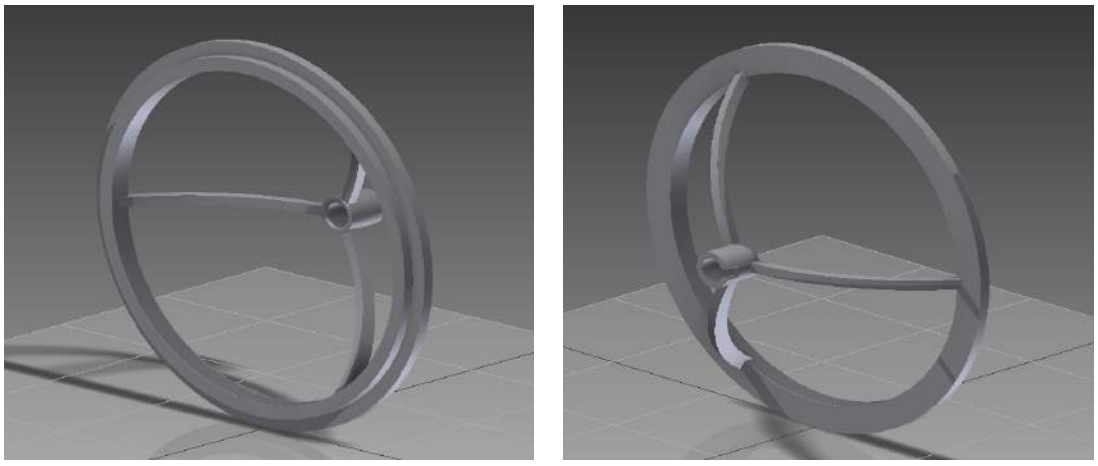
Revision C contains changes which address the issues found in revision B and also new additions to better suit the now fabricated PCB. It can be seen from



(a) Front side (b) Back side

FIGURE 4.6: Fabricated motor housing (Revision B)

figure 4.7 that there are now three struts attaching the rim to the base of the motor to provide more support. The arc in struts has also been altered to provide a smaller distance from the motor fixture to the plane of the rim hence reducing the profile of the quadcopter. New additions include changing the radius and width of the rim to match the considerations in section 4.3 and also bringing the lip of the rim to the same side as motor. This means the housing can be fixed on the underside of the PCB and avoid conflicting with components on the top side.

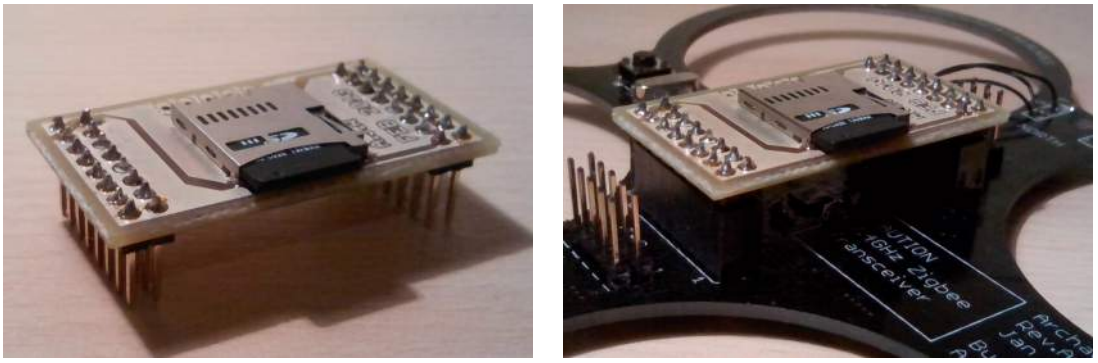


(a) Front side (b) Back side

FIGURE 4.7: Motor housing (Revision C)

4.6 Expansion hardware

BlackBox[2] is a simple manifestation of a possible hardware expansion option. A micro Secure Digital (SD) card acts as a store for data during flight and afterwards it can be read by a PC and examined. This is done rather than simply sending the data to the base station because of the large quantity generated at each iteration of the control loop. Figure 4.8 shows the module standing alone and attached to the system. Schematic and layout for this module is contained in appendix F.3.



(a) Stand alone (b) In system

FIGURE 4.8: The BlackBox expansion option

4.7 Base station hardware

The hardware for the transceiver is based on the ATZB-24-A2 transceiver package and a UART to Universal Serial Bus (USB) bridge to enable easy interfacing with the standard means of computer Input/Output (I/O). The stick also contains a JTAG header to program the microcontroller and two LEDs to indicate traffic or to use for general debugging. Figure 4.9 holds the fabricated board. When connected to the USB port the stick draws power from the standard USB 5V supply and uses the bridge IC, which contains a level converter, to provide a potential difference of 3.3V for the transceiver package. All circuit descriptions and board designs are contains in appendix F.2.

[2]A flight data recorder makes a log of instructions sent to electronic systems on an aircraft and is often referred to as a "black box" even though they are usually orange in colour.

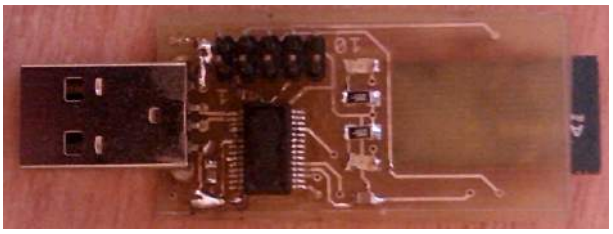


FIGURE 4.9: Transceiver stick

4.8 Prototype manufacture

PCB manufacture is an expensive process therefore the main quadcopter board was the only PCB fabricated to a high standard. The two other boards (sections 4.6 and 4.7) were fabricated using a free service provided by Spirit circuits. This is a tracks and holes only double sided board. The quality is far superior to any hand-made PCB methods, allowing for fine pitch Surface Mount Devices (SMDs), but not as of the same quality as the quadcopter board and doesn't support a silkscreen.

4.9 Firmware development

The main control processor keeps the craft stable during flight where the approach implemented is that which is discussed in section 2.3.1. The microcontroller takes a reading from the accelerometer which is then sent through a low pass filter. In parallel a reading from the gyroscope is integrated and then sent through a high pass filter. Finally the output of both filters is used to estimate orientation. This can be used to drive the motors after which data can be dumped to both the expand port and transceiver microcontroller. This must all be implemented in code for which the C language is used to develop low level functions on Atmel microcontrollers.

4.9.1 Data acquisition

Data from the sensor IC is gathered using a SPI connection. Each read/write from the chip takes at least two SPI transfers; the first Most Significant Bit (MSB) of the first byte indicates a read or write operation with the seven remaining bits containing the register address and the second byte contains the data (InvenSense, 2012). Burst read/writes are supported so when necessary more transfers will occur for extra data bytes. Listing 4.1 burst reads the six accelerometer registers.

```
1  int16_t accel_X,accel_Y,accel_Z;
2  void MasterInit(void)
3  {
4     PORTB |= _BV(PB0);//MPU6000_/CS not active
5     PORTB |= _BV(PB1);//EXPAND_/CS not active
6     SPCR = _BV(SPE)|_BV(MSTR)|_BV(SPDR0);//Enable SPI, Master, set fclk/16
7  }
8  uint8_t SPI_Swap(uint8_t data)
9  {
10    SPDR = data;
11    while(!(SPSR & _BV(SPIF)));//Wait for transmission complete
12    return SPDR;//Data has been swapped therefore returned
13 }
14 int16_t Get_two_bytes()
15 {
16    uint8_t high,low;
17    high = SPI_Swap(0xFF);//Dummy Swap to get data out
18    low = SPI_Swap(0xFF);//Dummy Swap to get data out
19    return ((high << 8)|low);//Smash bytes together
20 }
21 void MPU6000_Read_Accel(uint8_t start_address)
22 {
23    uint8_t high,low;
24    PORTB &= ~_BV(PB0);//GYRO_/CS active
25    //Begin reading from start register
26    SPI_Swap(0x80 | start_address);//MSB is high for a read op
27    accel_X = Get_two_bytes();
28    accel_Y = Get_two_bytes();
29    accel_Z = Get_two_bytes();
30    PORTB |= _BV(PB0);//GYRO_/CS not active
31 }
```

Listing 4.1: Data acquisition from MPU6000

Figure 4.10 contains a screen capture from a logic analyser attached to the connections between the host microcontroller and the MPU6000. The 750KHz clock is used to transfer the first register address byte (with the MSB high for read operations) into the chip then six data bytes out of the chip. These bytes are upper and lower parts of a two's complement 16-bit number therefore the upper byte is shifted and placed in a bitwise OR operation with the lower byte.

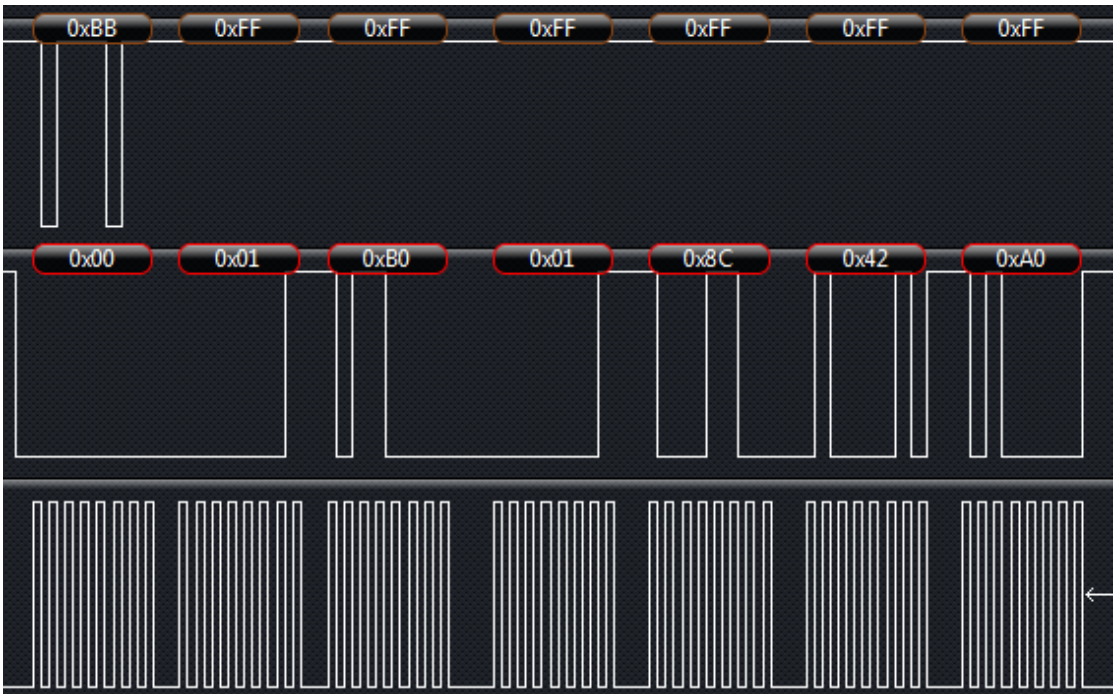


FIGURE 4.10: Serial peripheral interface read operations from MPU6000. Values held in table 4.3

An example of values read from the IC are contained in table 4.3. The values once cast represent the acceleration in the three dimensions of the chip. These readings were taken when the chip was configured for maximum resolution so each is scaled down by 16384 (= 2^{14} therefore shift right 14 places). The z-axis reading indicates approximately 1g and because the craft is stationary this means it is on a surface perpendicular to gravitational force. Substituting the values into equation (2.11) also indicates a pitch of zero (see equation (4.1)).

Byte	**Register (Name)**	**Address (Hex)**	**Value (Hex)**	**Value once cast (Decimal)**	**Value once scaled (g) (3 s.f.)**
1	ACCEL XOUT H	0x3B	0x01	See below	See below
2	ACCEL XOUT L	0x3C	0xB0	432	0.0264
3	ACCEL YOUT H	0x3D	0x01	See below	See below
4	ACCEL YOUT L	0x3E	0x8C	396	0.0242
5	ACCEL ZOUT H	0x3F	0x42	See below	See below
6	ACCEL ZOUT L	0x40	0xA0	17056	1.04

TABLE 4.3: Acceleromter values read from MPU6000. As seen in figure 4.10

$$\theta = \arctan \frac{0.0264}{\sqrt{0.0242^2 + 1.04^2}} = 0.0254\, rads \tag{4.1}$$

4.9.2 Filtering

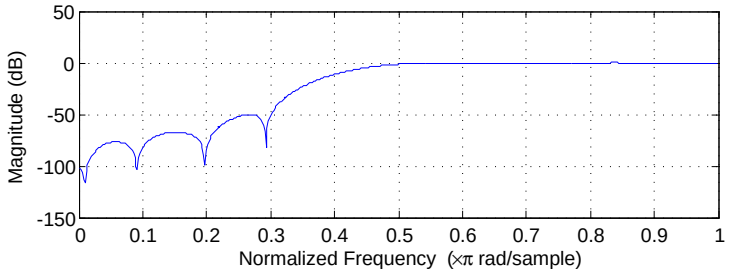
Basic Finite Impulse Response (FIR) filters are suitable for this application because they are simple and inherently stable. The control loop must also be kept short in duration to take enough samples. MATLAB can be used to generate the filter coefficients as required for each of the filters. The generic solution for an FIR filter is contained in equation (4.2)(Rogers, 2012) but as the output is now complied from two different inputs the approach can be considered as equation (4.3). The filters are designed as such that if both X_1 and X_2 were the same signal the output would also be approximately the same because the full spectrum has been reconstructed.

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{i=0}^{N-1} b_i z^{-i} \tag{4.2}$$

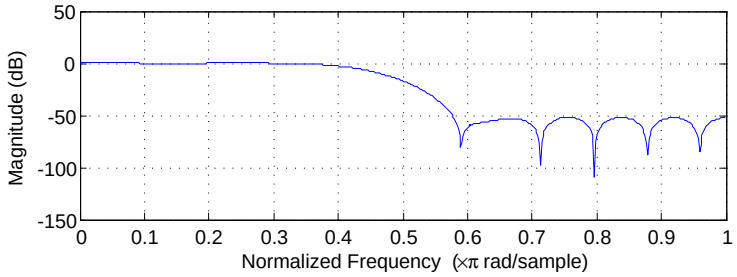
$$Y(z) = X_1(z) \sum_{i=0}^{N-1} c_i z^{-i} + X_2(z) \sum_{i=0}^{N-1} d_i z^{-i} \tag{4.3}$$

As a starting point for the filter a 10Hz separation frequency was chosen and with an initial sample rate of 45.776Hz (= 12MHz/(1024 $\times$ 256), divided from system clock). This requires a normalised cut off frequency of 0.437π rads/sample. Frequency plots of two 25 coefficient filters are contained in figure 4.11. This is just one example of how the process can be used to tune the design until suitable parameters are fitted. The code held in listing 4.2 is the implementation of this filter in C to be placed on the main control processor. The timer generated interrupt takes a sample which then passes it through the filter kernel to produce an output. The approach requires six filters in all so the performance is critical especially because of the large number of multiply-accumulate operation required.

These filters are a suitable implementation because they have large stop-band attenuation however the transition band width is not ideal. This can be improved by increasing the order of the filter but this comes with a pay off in delay which will make the control loop less responsive as the data is not as accurate.



(a) High pass filter



(b) Low pass filter

FIGURE 4.11: Magnitude response of 25 coefficient finite impulse response filters with a normalised cut-off frequency of 0.437π rads/sample

```
1  double high_coeffcients[] = {0.0015,-0.0016,-0.0041,0.0016,0.0123,0.0035,
2  -0.0265,-0.0230,0.0435,0.0759,-0.0574,-0.3065,0.5616,-0.3065,-0.0574,0.0759,
3  0.0435,-0.0230,-0.0265,0.0035,0.0123,0.0016,-0.0041,-0.0016,0.0015};
4  double pipe[N];
5  double data_out,data_in;
6  double filter_kernel(double input){
7     uint8_t n;
8           double output;
9     output = input*high_coeffcients[0];
10    for(n = 0;n < N;n++){//Calculate output
11       output += pipe[n]*high_coeffcients[n + 1];
12    }
13    for(n = 1;n < N;n++){//Update pipe
14       pipe[n] = pipe[n-1];
15    }
16    pipe[0] = input;//New value in pipe
17    return output;
18 }
19 void init_sample_timer(){
20    TIMSK1 |= _BV(TOIE1); //Enable overflow interrupt
21    TCCR1B |= (1 << CS00); //Set up timer at F_CPU/1024
22 }
23 ISR(TIMER1_OVF_vect){//Overflow an 8-bit timer = 256
24    data_out = filter_kernel(data_in);//Sample rate = 45.776Hz
25 }
```

LISTING 4.2: Implementation of an finite impulse response filter on an Atmel microcontroller

4.9.3 Secure digital card communication

Communication with the SD card is achieved via a SPI connection. The low level implementation for this protocol is taken care of by FatFs (Electronic Lives Manufacturing, 2012) which is a generic software package aimed at small embedded systems. FatFs has been ported for the ATMEGA1284P. The nature of BlackBox functionality however only requires the SD card to be written to which reduces complexity. The log initialisation and population routines are held in listing 4.3. This avoids overwriting a previous log with instance coding. Notice how the work area is constantly mounted and unmounted to avoid causing a corrupted work space if the power is cut during a write operation. The sector is kept as 0 because no other sector is needed. A possible danger for tighter control loops is that the card may run out of storage so it is recommended to use SD cards of the maximum size which is two gigabytes (high capacity not supported for FatFs protocol).

```
1  FATFS fs;//Work area
2  FIL fdst;//File object
3  char path[20];
4  //Create a new log file, once per run?
5  void openBlackBox (void)
6  {
7     uint8_t index = 0;//256 logs
8     disk_initialize(0);//initialise sector 0
9     f_mount(0, &fs);//mount sector 0
10    while(index < 255)
11    {
12       sprintf(path,"0:arch%d.log",index);//Don't overwite a previous log
13       if(f_open(&fdst, path, FA_CREATE_NEW) == FR_EXIST){//Is the path occupied?
14          index++;//increment the last digit in the path
15       }else{
16          break;//Free spot, create log
17       }
18    }
19    f_close(&fdst);
20    f_mount(0,NULL);//unmount sector 0
21 }
22 //Always run openBlackbox first
23 void writeBlackBox(uint8_t *data,uint8_t length)
24 {
25    f_mount(0, &fs);//Mount sector
26    f_open(&fdst, path, FA_OPEN_EXISTING | FA_WRITE);//Open existing for writing
27    f_lseek(&fdst, f_size(&fdst));//Go to end of file
28    f_write(&fdst,data,length,0);//Write data
29    f_close(&fdst);
30    f_mount(0,NULL);//Unmount sector
31 }
```

Listing 4.3: BlackBox log creation and population

4.10 Base station software development

The software is designed as such to interface with transceiver stick or debug UART connection using the same protocol. The software requests a packet of data from link by continuously sending dummy bytes and waiting for a reply; eventually a frame of 64 bytes is then sent back to the base station. Table 4.4 contains the contents of a frame inbound to the base station.

Byte(s)	**Name**
0...1	ACCEL XOUT (H...L)
2...3	ACCEL YOUT (H...L)
4...5	ACCEL ZOUT (H...L)
6...7	TEMP OUT (H...L)
8...9	GYRO XOUT (H...L)
10..11	GYRO YOUT (H...L)
12..13	GYRO ZOUT (H...L)
14...17	FIL ACCEL X (3...0)
18...21	FIL ACCEL Y (3...0)
22...25	FIL ACCEL Z (3...0)
26...29	FIL GYRO X (3...0)
30...33	FIL GYRO Y (3...0)
34...37	FIL GYRO Z (3...0)
38...41	PITCH (3...0)
42...45	ROLL (3...0)
46...49	YAW (3...0)
50...53	TIME STAMP (3...0)
54...57	MOTOR (3...0)
58	Battery
59...63	Debug/peripheral

Table 4.4: Frame composition

The first 14 bytes is simply the latest sample of raw data from the sensors. The next 24 bytes is the latest output from the filters and therefore the information seen by the quadcopter control routine. These are originally float data types and have therefore been decomposed to 4 bytes for transmission where upon reception can be reconstructed. The same is true for values of roll, pitch and yaw which are placed in bytes 38 to 49. A time stamp is placed in the frame as a 32-bit unsigned integer which counts the number of control loop iterations and can therefore be used to calculate system on time. This is large enough to avoid overflow as even if the loop was $1\mu s$ in duration it would take 1.2 hours of system on time to break. Four bytes are added for the current rotational speed of each motor. Finally the transceiver module appends the packet with a byte of data on the state of the

battery voltage before interfacing with communication link. The remaining bytes of data are left for either the peripheral device to use or for additional data required for debugging purposes.

Different aspects of the base station software were added in parallel as the design began to increase its functionality. A tabbed layout was used to operate the three main operations of the interface. The main area plots real-time data taken from the quadcopter. Four charts are used to plot the raw and filtered output from both the accelerometer and gyroscope. Another plot holds the three axis orientation that the system has calculated from the readings and filtering. Temperature, time stamp and battery voltage are printed to the GUI along with a bar chart of motor output. This can be seen in figure 6.1.

The second tab allows a user to review all communication ports on the host PC and select which port the UART connection has been established. The Baud rate can also be selected to match that of the peripheral device. When the connection is activated the program constantly sends dummy data bytes and checks the PC hardware buffer for incoming packets. Once a byte is received the program waits until the buffer is full with the remaining bytes in the frame. These are displayed in a raw format to the user and also constructed format for immediate verification. See figure E.1 for a running example. The final section allows data to be read from a log file for analysis. This can be copied from either the SD card to the PC or from a previous session of the base station.

4.11 Full system integration

Finally bringing all the hardware aspects together results in figure 4.12 and 4.13 with some additional components and tweaks discussed in section 5. The propellers are fixed to the motor shafts using epoxy resin and same is also true for the motor housing to the PCB. The three wires for the motors are soldered straight to their respective pads.

FIGURE 4.12: Archaeopteryx top

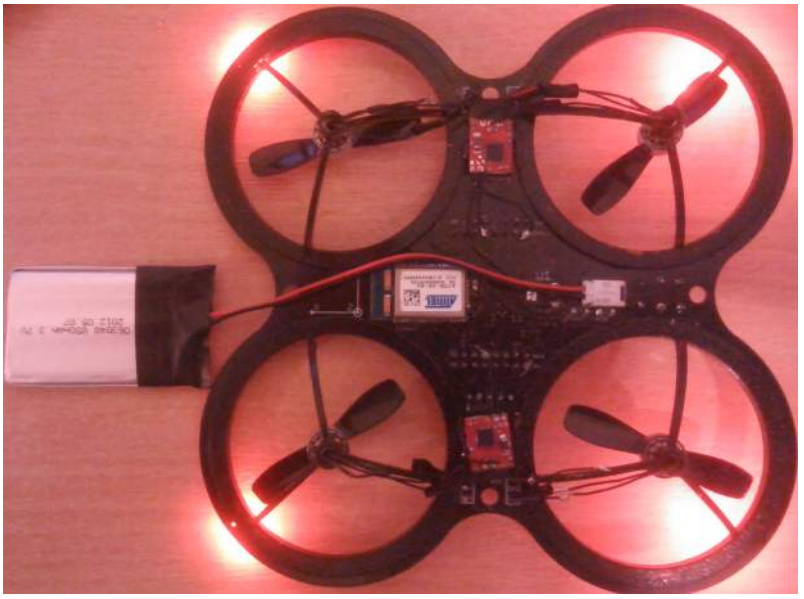


FIGURE 4.13: Archaeopteryx bottom

Chapter 5

Testing and tuning

Testing was carried out parallel to some firmware and software development as this was required for functionality testing. The testing cycle is as such; add to the board, test new additions (electrical characteristics and/or data processing) and repeat until all components are present and functional. The first components placed on the PCB were the two microcontrollers along with the programming headers, support components and the power supply. These were verified over both the ISP and JTAG connections to be functional and therefore programmable. The four LEDs were then added and also proven to be functional by programming the microcontroller to which they are connected.

The next IC to be soldered on to the board was the MPU6000 which did not function as expected. This revealed an issue with the SPI connection, fixed in section 5.1.2, but a temporary fix was found by setting the SPI pins on the ATZB-24-A2 module as inputs therefore preventing data collisions. The MPU6000 then exhibited strange functionality as read/write operations functioned only intermittently. The problem was traced to incorrectly soldered pins when an analogue oscilloscope test revealed a dropping voltage level on Master In Slave Out (MISO) traffic from the IC. Figure 5.1 contains the traces used to draw this conclusion. The expansion SPI port and debug UART were else successfully tested during this period.

Re-soldering the MPU6000 fixed the issue and the next item to test was the motor drivers. These immediately presented a problem because of their Micro Dual Flat No-Lead (uDFN) $0.5mm$ pitch packaging which proved difficult to solder and verify connections. These were eventually solder and checked with a low magnification microscope but when connected to the motors and respective micrcontoller

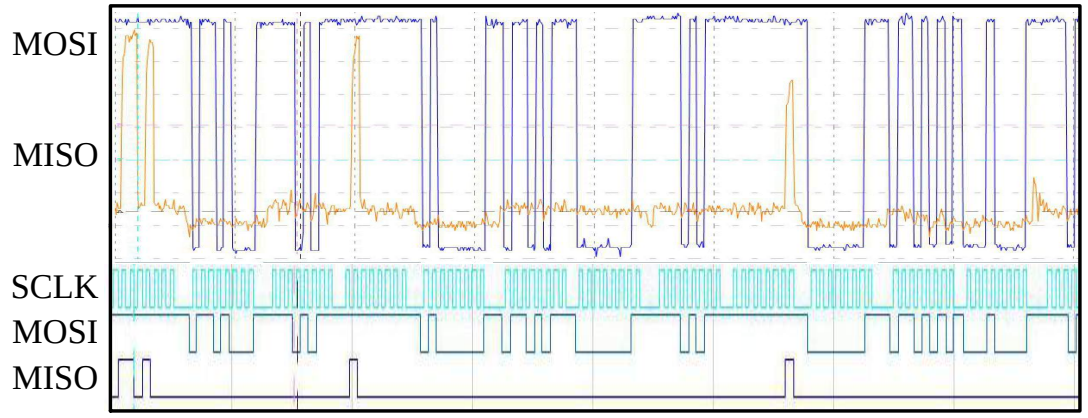


FIGURE 5.1: Annotated screen capture of an analogue trace (top) and logic analyser (bottom) connected to serial peripheral interface pins of the MPU6000

produced no turning force upon a motor. The output of the chip was reviewed when disconnected from the motor and revealed three out of phase signals which suggested that the current available to drive the motor was not large enough. Thin long tracks on the PCB attributed somewhat to this but when attaching additional wires in parallel with these tracks in an attempt to decrease resistance the motors only moved with very low torque.

5.1 Hardware alterations

5.1.1 Motor drivers

The ICs used to drive the motors are aimed at small BLDC motors typically attached to a fan. Figure 5.2 shows how the chip is configured. A Central Processing Unit (CPU) is used to produce three sinusoidal inputs to Metal Oxide Semiconductor Field Effect Transistor (MOSEFT) pairs.

Introducing a major issue into the critical control loop the two options for remedying the problem were to either change the motors or the drivers. The motors were chosen because of their success in Radio Controlled (RC) hobby aircrafts and had already been designed to work with the housing and propellers. The drivers were chosen because of their small package size and simplicity. The fastest solution proved to replace the drivers with electronic speed controller circuits proven to work with motors already. The chosen driver PCB is contained in figure G.2 and it can be seen how it is similar to the original driver IC but simply bigger thus able to source more current for the motors. These speed controllers use a

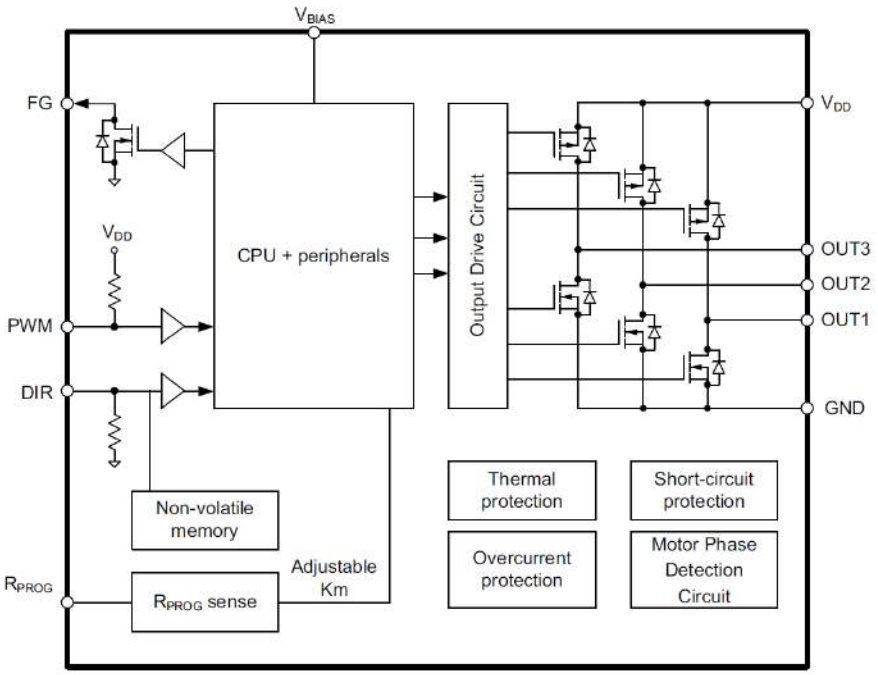


FIGURE 5.2: Microchip MTD6505 - functional block diagram. Taken from Microchip (2011)

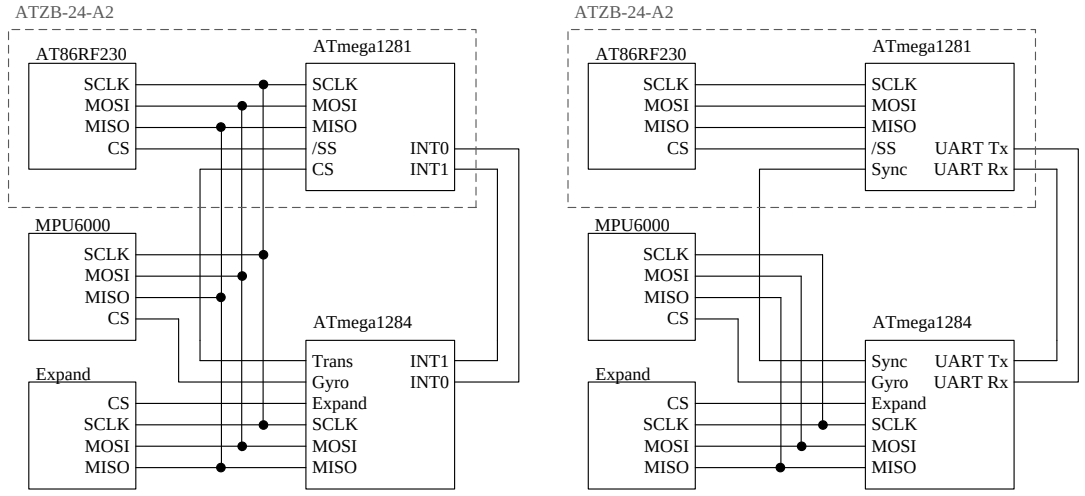
50Hz Pulse Width Modulation (PWM) signal to enable linear control of a 3-Phase BLDC motor by varying the duty cycle between approximately 10% and 20%. At $10mm$ by $13mm$ the boards fit in the propeller hole separation gap of $16mm$ so two were placed top layer and two on the bottom layer in the region now empty from the first driver ICs.

The original input signals required for each motor driver was a PWM channel and a single logic input for directional control of the motor. This is a trivial task for the Atmel microcontroller as the arbitrary frequency PWM specification only called for a change in duty cycle which would map to rotational speed. The new 50Hz PWM specification is strict on frequency and duty cycle. This has been implemented by using a timer and handler code to shape the output waveform as required.

5.1.2 System architecture

When tested the SPI communication throughout the board did not function as expected. This was traced back to the design phase when the decision to implement a SPI only serial link was taken. This required all digital components to interact

using a strict protocol. Atmel chips do support hardware based SPI communication but require explicit use of the slave select pin whenever a microcontroller is initiated in slave mode. This design did not use the correct pin for slave select as this was connected to the AT86RF230 IC inside the module (figure 5.3(a)). The original intention was to use SPI and switch master mode using interrupts for rapid dual processing and as a form of mitigation in controlling the sensors. The architecture was repaired by removing the global SPI tree and using two separate trees with a UART connection between the two microcontrollers (figure 5.3(b)). This was made possible by soldering wires between the appropriate connections on the ICs and removing the copper tracks for MISO, Master Out Slave In (MOSI) and Serial peripheral interface CLocK (SCLK) entering the ATZB-24-A2 module (shown in figure G.1). The chip select line was left intact as a synchronisation line.



(a) Original (b) Fixed

FIGURE 5.3: Serial peripheral interface architecture repair

5.2 Hardware setup for firmware verification

The microcontrollers are programmed over a JTAG interface and to achieve this an AVR Dragon was chosen. This tool allows for daisy chain programming of AVR devices (required) and supports in-system breakpoint features to assist with development. The setup in figure 5.4 contains the PCB connected to the AVR Dragon and also a UART to USB cable to assist with data verification. This setup is also used to verify the functionality of base station software.

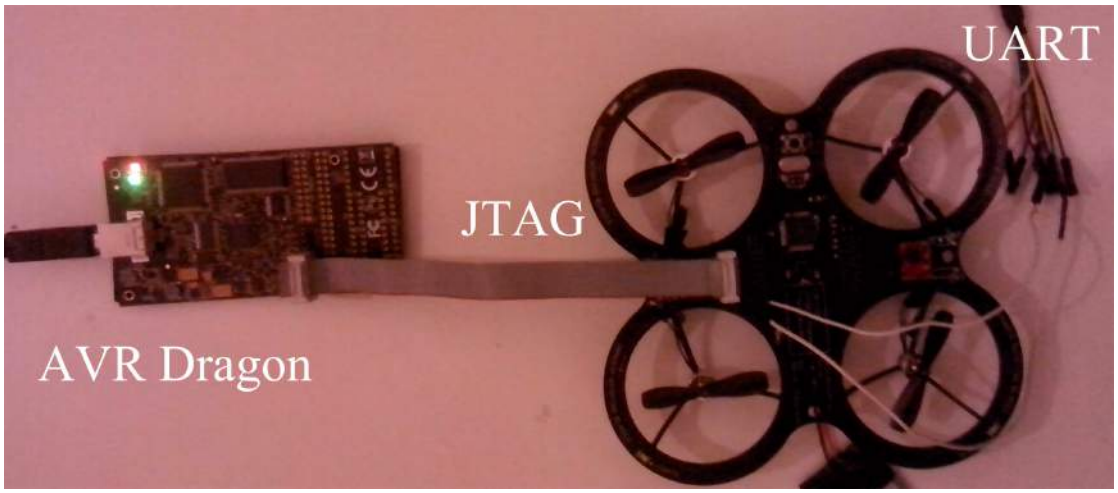
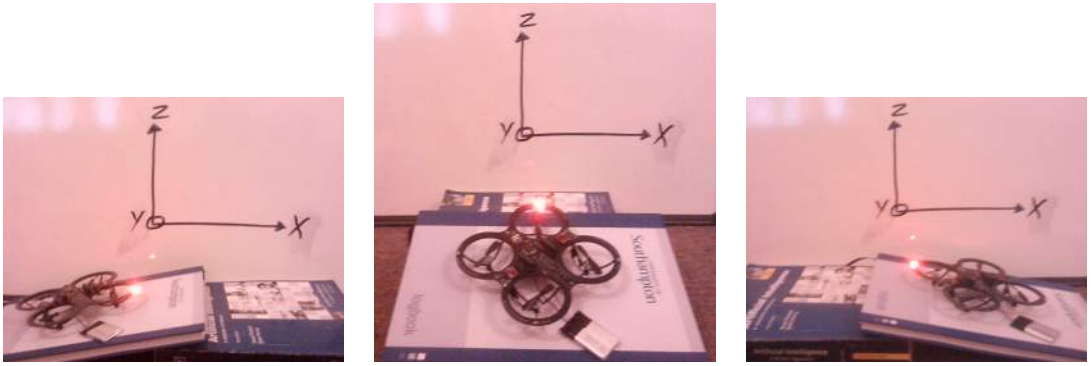


FIGURE 5.4: Hardware setup

5.3 Sensor readings

Initially data from the sensors was viewed via the four LEDs at the corners of the board. Values of pitch and roll map to appropriate LEDs and through visual feedback it is clear the system is reading valid sensor data. Figure 5.5 contains three of the four possible states for visual feedback. As the book changes in orientation so does quadcopter as it remains fixed in place at the feet. The sensor data be viewed in more detail by either writing it to the SD card, transferring it over the debug UART or ideally sending it to the USB transceiver via the wireless ZigBee connection.



(a) Positive pitch and roll (b) Positive roll and negative pitch (c) Negative roll and pitch

FIGURE 5.5: Visual feedback of attitude

5.4 Control loop optimisation

All functions to be executed on the control microcontroller are contained inside a timer driven interrupt for which the frequency defines the coefficients for the filters and the overall response of the system. This can be improved by making the execution time as small as possible. The main functions are to read data and drive the motors which in themselves are not processor heavy unlike the control calculations between these two steps. These initially consist of floating point calculations which require multiple clock cycles to complete but where possible these can be replaced with integer operations or bit shifting which take a lot less time.

The other functions this microcontroller is concerned with can be negated in terms of time dependency so instead of executing inside the interrupt function they can be spread across multiple intervals between interrupts. BlackBox may however be used in the loop to gain a sample of every piece of data collected. UART communication can be done outside the interrupts causing lag only in the transceiver module which is why these two modules were separated. This lag is reduced by increasing the baud rate from the initial 9600Bd to that the fastest stable rate which is 57600Bd throughout the system.

Chapter 6

Evaluation

6.1 Performance

The main design in this project was the quadcopter PCB which functioned as described in the schematics with flaws appearing only in the initial designs. All layout design, e.g. footprint models and circuit routing, was successfully implemented.

System integration proved challenging as bringing together many different designs into a singular functional unit rendered unforeseen complications. The motor and propeller combinations did not produce enough thrust for the body of the craft. This could have been prevented by increasing modelling and analysis of variation during the design phase. Unfortunately weight estimations proved difficult as specification of both the PCB and motor housing remained unknown. The additional hardware alterations also increased weight.

The wireless link remains untested as this was not a basic requirement to achieve further functionality. The transceiver stick however behaves as designed. When attached to the computer it identifies itself as a serial port and can exchange data with the ATMEGA1281 microcontroller inside the transceiver module.

6.1.1 Thrust to weight ratio

Attaching the quadcopter to a test rig upon a set scales allowed for an estimation in the thrust produced. The craft is placed upside down to allow for better airflow

therefore lift is manifested as an increase in weight. Table 6.1 contains individual component weight, total weight and the change in the test rig weight. The thrust to weight ratio is calculated using equation (6.1) which reflects the inability to achieve lift. These results are an estimation and more thrust should be achieved in open space. The motor housing remains fixed to the board but using the last revision as a lower estimation the weight of the board with only original components comes to 354.34N.

Item	**Weight (N)**
Revision B motor housing	23.64
BLDC motor	19.62
Motor driver modification board	5.20
Quadcopter (no battery)	548.18
Test rig (motors off)	1349.37
Test rig (all motors on full power)	1564.70

Table 6.1: Section weights

$$\frac{Thrust\ (N)}{Weight\ (N)} = \frac{\Delta\, Test\ rig\ weight}{Quadcopter\ weight} = \frac{1564.70 - 1349.37}{548.18} = 0.39 \qquad (6.1)$$

6.1.2 Control implementation

The firmware produced for the design uses the complementary filter approach discussed in section 2.3.1. This could not be tested fully but test plans allow for partial tuning for a one dimensional case by using the stabilisation holes in the PCB. Removing the need for a thrust to weight ratio greater than one as the craft can be suspended in mid-air and balanced using the thrust that is available.

Before this testing can take place further tuning is required to gain a more reliable reconstruction of the filtered signals. It can be seen from the screen capture of the base station (figure 6.1) that a better filter implementation is required. The high pass filter used on the integrated gyroscope measurement requires a sharper transition band to remove the drift but retain the high frequency information. The low pass filter for the accelerometer suffers because of such a low cut off frequency and removes very little high frequency content from the signal. This can be improved by using an Infinite Impulse Response (IIR) filter to improve transition sharpness. Improving the method of integration will also help to remove drift from the measurement contribution of the gyroscope. This can be done by using best fit curves or trapezoid area approximation rather than the current approximation of rectangles.

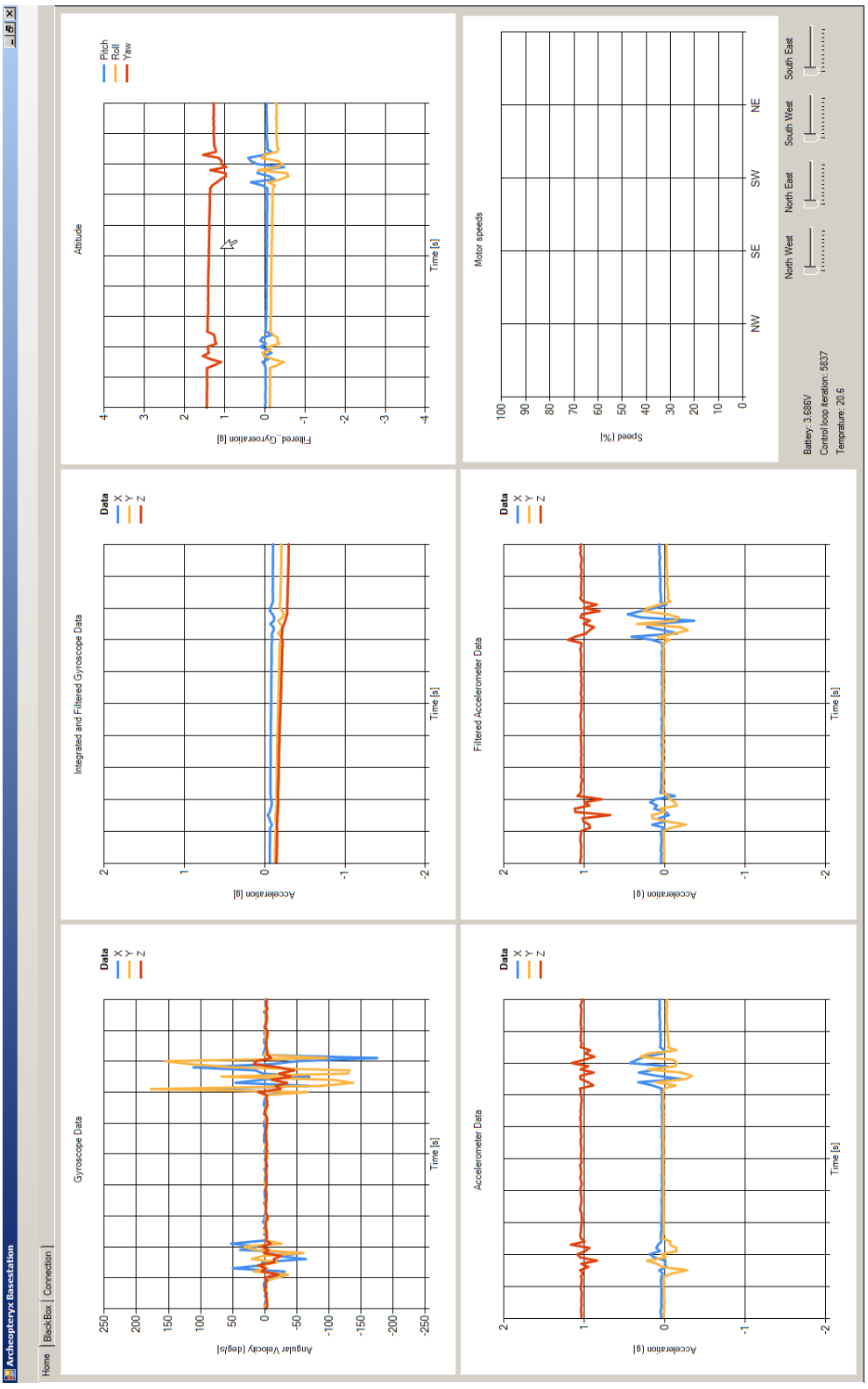


FIGURE 6.1: Real-time plots

6.2 Production cost

This prototype unit cost a total of £328.13 for the basic components with additional costs for some of the hardware modification features; see table F.1. This price can be reduced for the perspective audience when considering the larger production of PCBs and bulk purchase of components.

6.3 Project planning: a retrospective

The Gantt chart in figure I.3 shows the actual activities which took place instead of those planned in figure I.2. They are similar with respect to the first PCB submission but when testing revealed the need for more time and components the lag was used to work on future tasks. This is why a modular addition was implemented so early when it was intended to be an optional extra towards the end of the project.

The risk register considered in table 3.1 reflected the actual issues which arose during implementation. The concern with the first PCB revision was justified as it did not function as required but the contingency strategy, revision B PCB, proved ineffective with diminishing budget and time. The inability to fly was a large influence on the project but as a low likelihood was estimated the risk score was relatively low. This score should have been greater as much more mitigation actions should have been put into place.

Chapter 7

Conclusions and future work

"Any sufficiently advanced technology is indistinguishable from magic."

Arthur C. Clarke

This project proved very ambitious. Reviewing the original project brief contained in appendix A it can be seen that some of the initial ideas are greatly over specified. Breaking the project down in waypoints however allowed some of the specification to be achieved. The progress made serves as a great starting point for continuing development on this idea with considerable ground already covered and a lot of experience gained.

7.1 Archaeopteryx improvements

Some features of the PCB can be improved for the next revision. These would address some of the hardware changes discussed in section 5.1. Table 7.1 contains the major changes but apart from these most of the board can remain the same. Dimensions and component placement proved a great success as the center of gravity of the design remains at the origin of the board.

7.2 Formalisation

All code written for the quadcopter as of current is quite specific and only drawn together for this prototype. The aim in the future is to make this accessible and

Improvement	Reasoning
Replace board level communication protocol with that in figure 5.3(b).	This fix proved successful and therefore should be made part of the design.
Replace the driver ICs with either new ICs capable of driving enough current or an microcontroller/MOSEFT configuration.	This will prevent additional circuitry from being added manually.
Separate power supplies for logic devices and motor drivers.	Avoid voltage rail ripples.
Reduce PCB safety rim width.	The width chosen provided more than enough support.
Label every output on headers and reduce size if possible.	These headers are for expansion and debugging. A lot of the testing has been done.
Remove ISP header.	JTAG communication worked fine through out the board.

Table 7.1: Improvements for revision B

as easy to manipulate for researchers. Therefore code should be restructured and layered for interfacing with the user application. This will require formal documentation and published restrictions on processing capabilities. The transceiver is also yet to be tested as part of the design and like the code this will require evaluation.

7.3 Propeller optimisation

The intention has always been to make the copter as compact as possible, keeping every component aligned in one plane but due to physical limitations and sourcing issues this has not been possible. The motors protrude from the bottom of the craft and therefore housing was used to keep the propellers aligned. This solution resolved the issue but an ideal approach is to replace the propeller/motor/housing combination with a single fan type configuration; as seen in figure 7.1. This was unfeasible to purchase during the project because of required clockwise/anticlockwise matching for the propellers but using the 3D printer it would possible to create such components. The BLDC motor can now be placed in the center and by removing the shaft a printed propeller hub can be placed around the core. The three wires for each phase can then run along the supports where they will meet through hole connections in the new PCB. Making it $6mm$ thick covers a $1.6mm$ board, the BLDC motor height and the need for an adhesive rim.

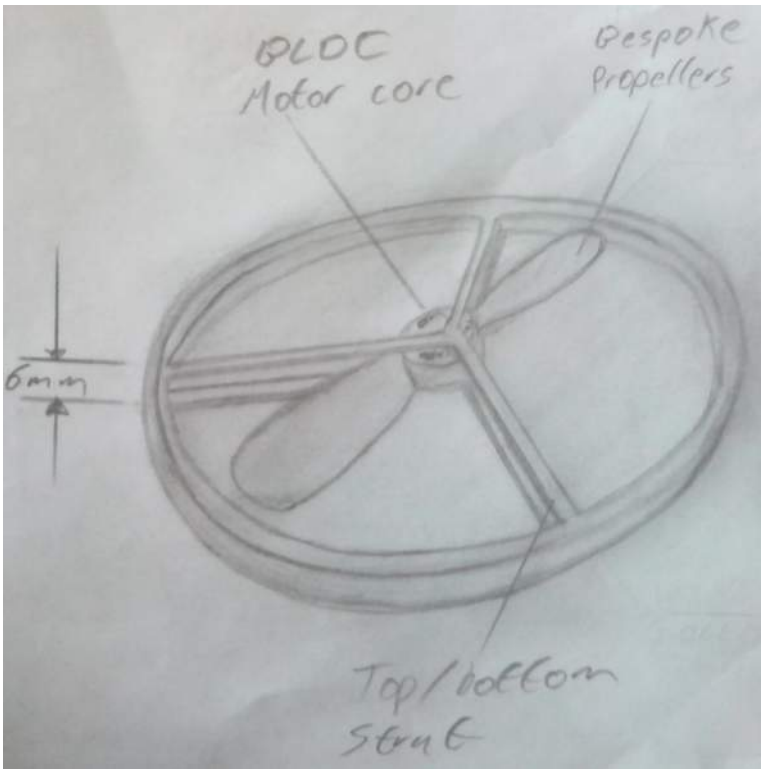


FIGURE 7.1: Propeller/motor/housing sketch

This would also incorporate blades better suited to motor and size of the quadcopter. An idea during the design phase was to model the propellers as such to optimally match them with the motors (first idea held in figure H.1). This can be achieved using a Propeller/Windmill Analysis and Design Engine (QPROP) (Drela, 2007). Designed to source the best motors on the market to optimally match with a certain propeller it can instead be used backwards to converge on an optimally matching propeller for a motor. The propeller can then be 3D printed and incorporated into the fan configuration for the next quadcopter.

7.4 Weight loss

Providing more lift with different propellers is one catalyst for flight but losing weight is also beneficial. For a second design a lot more weight can be lost and yet the structural integrity retained. Area not populated with tracks or silkscreen on the PCB could simply be removed as they bring nothing to craft except weight.

7.5 Final thoughts

This concept has nothing to which it can be directly compared as there are no publicised implementations of similar low profile quadcopters. Only the thrust to weight ratio proved to be the downfall with all other hardware systems in working order. Further decreasing the profile and increasing the available thrust will certainly aid in reaching the specified criterion of this project. This is an exciting implementation with scope for deployment in an increasingly popular field of research.

Bibliography

H.A.F. Almurib, P.T. Nathan, and T.N. Kumar. Control and path planning of quadrotor aerial vehicles for search and rescue. In *SICE Annual Conference (SICE), 2011 Proceedings of*, pages 700–705, 2011.

M.S. Alvissalim, B. Zaman, Z.A. Hafizh, M.A. Ma"sum, G. Jati, W. Jatmiko, and P. Mursanto. Swarm quadrotor robots for telecommunication network coverage area expansion in disaster area. In *SICE Annual Conference (SICE), 2012 Proceedings of*, pages 2256–2261, 2012.

Autodesk. Autodesk inventor: 3D CAD software. accessed: 18/04/13.

Lin Bai. Electric drive system with bldc motor. In *Electric Information and Control Engineering (ICEICE), 2011 International Conference on*, pages 359–363. Dept. of Electr. and Comput. Eng., Purdue Univ. Calumet, Hammond, IN, USA, April 2011.

Bitcraze. Bitcraze: Crazycopter. accessed: 6/12/12.

B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61 –72, may 1988.

Roland Büchi. *Fascination Quadrocopter*. Herstellung und Verlag, 2011.

Shane Colton. The balance filter, 2007. accessed: 25/03/13.

Prof. Mark Drela. QPROP: Propeller/Windmill Analysis and Design, December 2007. accessed: 14/04/13.

EAGLE. Easily Applicable Graphical Layout Editor. accessed: 24/04/13.

Electronic Lives Manufacturing. Fatfs - generic fat file system module, 2012. accessed: 03/04/13.

IEEE:802.15.4. Standard for local and metropolitan area networks. 2011. IEEE P802.15 Working Group.

InvenSense. MPU-6000 and MPU-6050 product specification. 2012. revision 3.3, accessed: 08/04/13.

Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

A. Kushleyev, D. Mellinger, and V. Kumar. Towards a swarm of agile micro quadrotors. In *Robotics: Science and Systems (RSS)*, 2012.

Hyon Lim, Jaemann Park, Daewon Lee, and H.J. Kim. Build your own quadrotor: Open-source projects on unmanned aerial vehicles. *Robotics Automation Magazine, IEEE*, 19(3):33 –45, sept. 2012.

Nick Longrich. Structure and function of hindlimb feathers in archaeopteryx lithographica. *Paleobiology*, 32(3):417–431, 2006.

Teppo Luukkonen. Modelling and control of quadcopter. 2011.

Microchip. Sinusoidal sensorless 3-phase brushless dc fan motor driver, November 2011. accessed: 19/03/13.

Y. Mohan and S.G. Ponnambalam. An extensive review of research in swarm robotics. In *Nature Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 140 –145, dec. 2009.

PCBCART. PCB & assembly direct sale from China. accessed: 09/04/13.

RMAC. Rand model aeronautic club: Batteries explained. accessed: 18/04/13.

Professor Eric Rogers. Digital filter design. *ELEC3029 - Signal Processing*, 2012. Part III, Semester I, University of Southampton.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010.

Inkya Sa and Peter Corke. Estimation and control for an open-source quadcopter. 2011.

Spirit. Go naked. accessed: 08/04/13.

Greg Welch and Gary Bishop. An introduction to the kalman filter. 2001.

Baoding Zhang and Shulan Gao. The study of zigbee technology's application in swarm robotics system. In *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on*, pages 1763 –1766, aug. 2011.

Tianguang Zhang, Wei Li, M. Achtelik, K. Kuhnlenz, and M. Buss. Multi-sensory motion estimation and control of a mini-quadrotor in an air-ground multi-robot system. In *Robotics and Biomimetics (ROBIO), 2009 IEEE International Conference on*, pages 45–50, 2009.

Appendix A

Project brief

10^{th} October 2012

A quadcopter is an aircraft that achieves flight using four propellers oriented such as to create four vertical columns of lift. These propellers are independently controlled to set the yaw, pitch and roll of the craft. Movement is achieved by varying the orientation and overall power output. This design will be used in a multi-agent swarm that can complete complex tasks in an indoor environment.

Each quadcopter will use sensors to gather data on orientation and because the behaviour of the motors, propellers and environment are known feedback can be used to achieve stable flight. This can only be made possible by using a controller that reacts fast enough to maintain the desired flight angles. Secondary sensors will also be used to improve how an individual agent will perceive the environment. These sensors should be attached as optional modules to a customisable base model allowing the end user to create a diverse swarm. The focus of the design should always remain on minimising the scale and overall cost for each unit.

Individual aircraft will require a wireless communication link which should be at least half duplex. The link will range from a simple point-to-point connection when only one quadcopter is active to an adaptive fully connected network when an entire swarm is active. Data transfer via the connection will serve as either manual control for the swarm/individual or as an interrupt for a program held locally. In both cases flight data will be periodically dumped to a base station which should take the form of bespoke software running on a PC which has the ability to communicate wirelessly using the same protocols.

The growth of the project should be based on a spiral model following chronological waypoints so if tasks prove too ambitious there will still be a functional final result.

1. Construct and test a prototype that can achieve stable flight
2. Create a user interface to manually control the device from a base station
3. Implement modular additions that can increase an agent's perception of the environment.
4. Construct and test a production grade quadcopter
5. Replicate the final design in order test swarm interaction
6. Program the swarm to complete basic tasks

Appendix B

Archive description

- 3D_Printing
  - revA
  - revB
  - revC
- Code
  - Archeopteryx
  - Basestation_MCU_ATMEGA1281
  - Control_MCU_ATMEGA1284P
  - Transceiver_MCU_ATMEGA1281
- Datasheets
- Figures
- Meeting_log
- PCB_Design_Files
  - Archaeopteryx_blackbox_revA
  - Archaeopteryx_revA
  - Archaeopteryx_USB_transceiver_revA
- Brief.pdf
- Progress_report.pdf

- Project.pdf
- Project.tex
- README.txt

```
-3D_Printing
   The three revisions of the motor housing in Autodesk Inventor

-Code
   All firmware and software development code
   Archeaopteryx => C#
   Basestation_MCU_ATMEGA1281 => C
   Control_MCU_ATMEGA1284P => C
   Transciver_MCU_ATMEGA1281 => C

-Datasheets
   All the datasheets for components used in the project.
   Mostly PDFs but webpages are also stored locally and as hyperlinks

-Figures
   All figures used to compiles the project report and drawing package files

-Meeting log
   Time indexed meeting agendas

-PCB_Design_Files
   All PCB CAD files for the design in EAGLE

-Brief.pdf
   The brief submitted in October 2012

-Progress_report.pdf
   Progress report submitted in December 2012

-Project.pdf
   The electronic version of the final report

-Project.tex
   LaTeX file used to produce the final report

-README.txt
```

Listing B.1: README.txt

Appendix C

Printed circuit board configuration options

C.1 Circular

$$Usable\ area = \pi(\epsilon - \mu)^2 - 4\pi(\rho + \mu)^2 \tag{C.1}$$

$$Total\ area = \pi\epsilon^2 \tag{C.2}$$

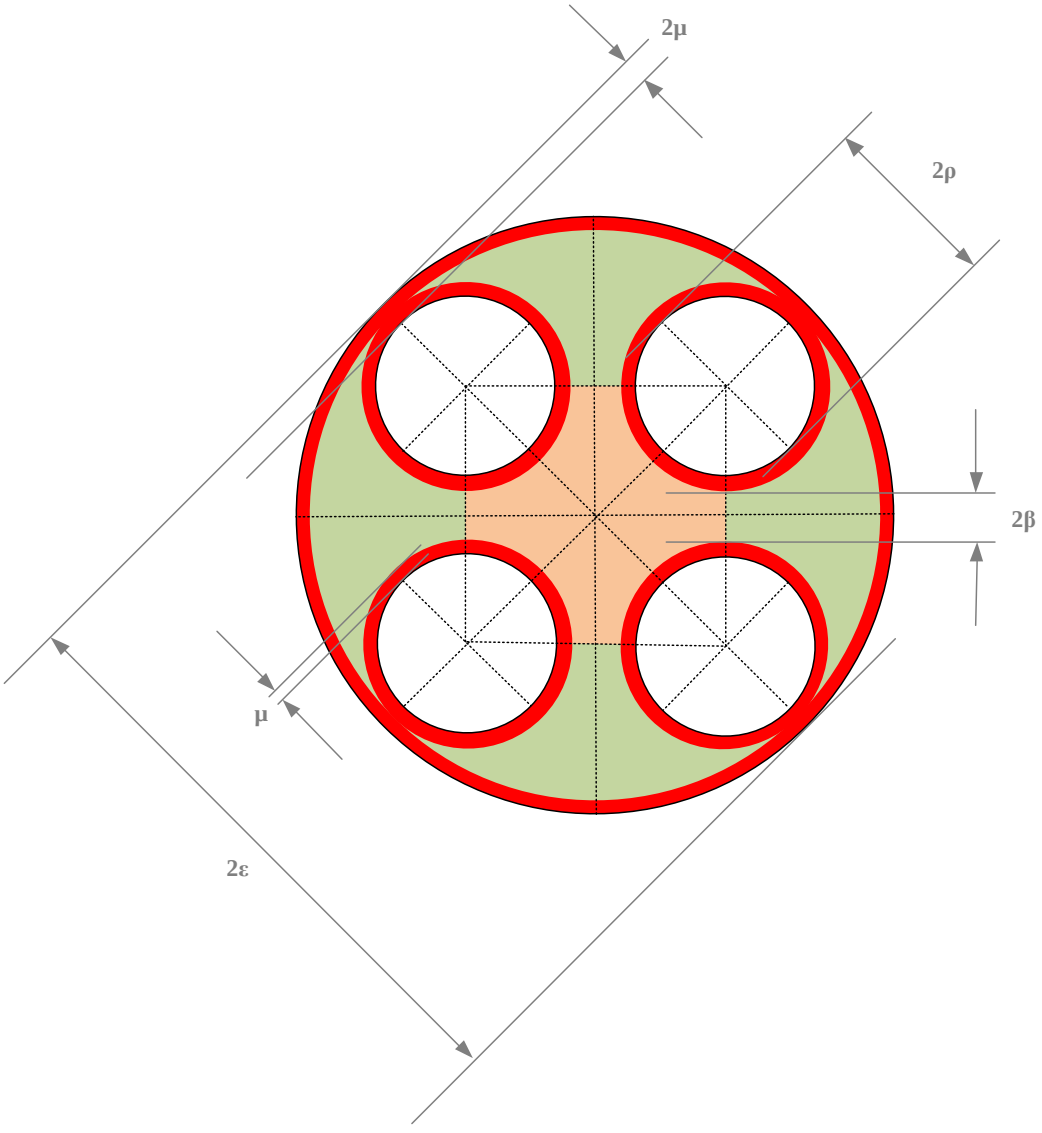


FIGURE C.1: Circular layout

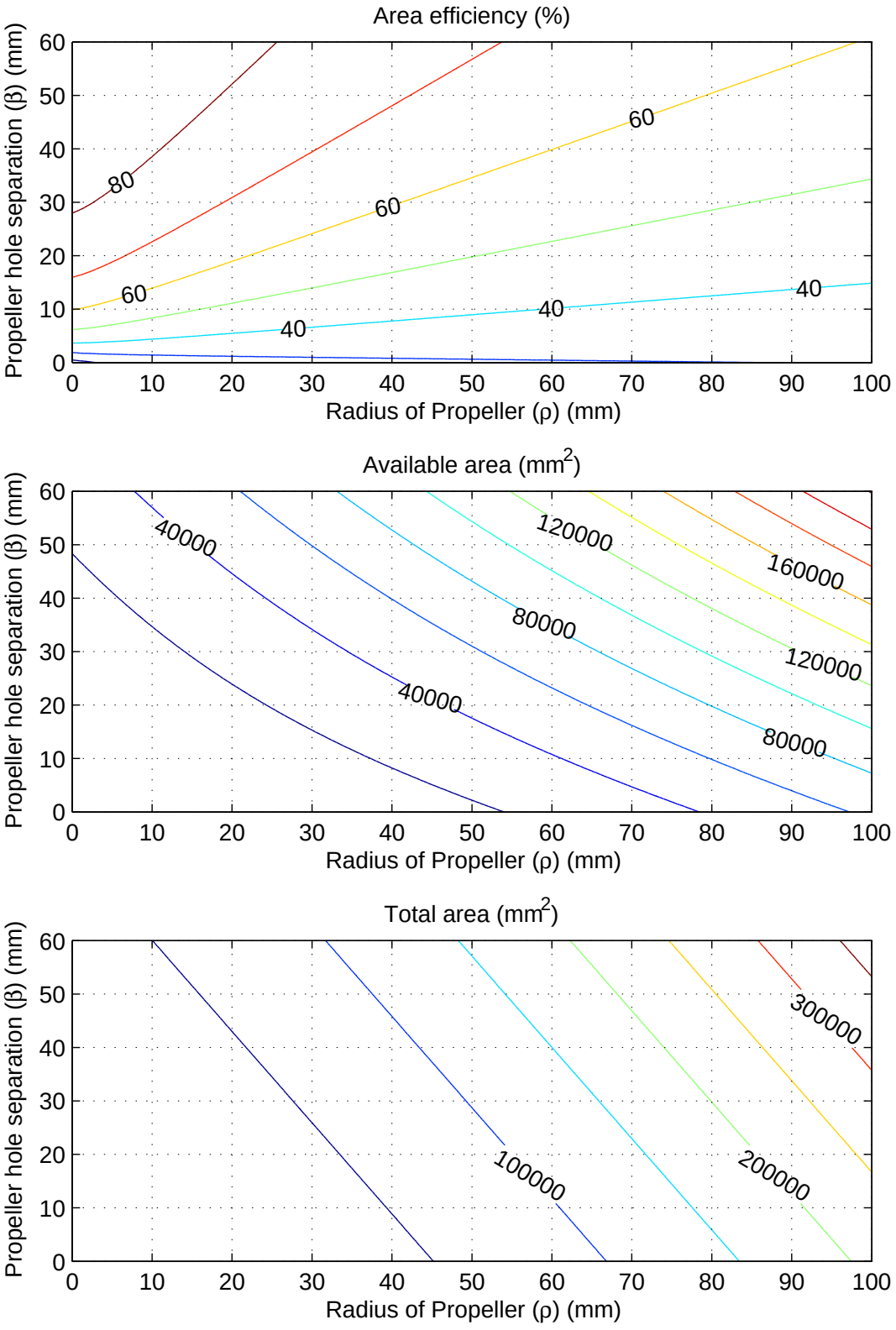


FIGURE C.2: Circular layout scaling

C.2 Tapered square

$$Usable\ area = (4\rho + 4\mu + 2\beta)^2 - (4 - \pi)(\rho + \mu)^2 - 4\pi(\rho + \mu)^2 \tag{C.3}$$

$$Total\ area = (4\rho + 6\mu + 2\beta)^2 - (4 - \pi)(\rho + 2\mu)^2 \tag{C.4}$$

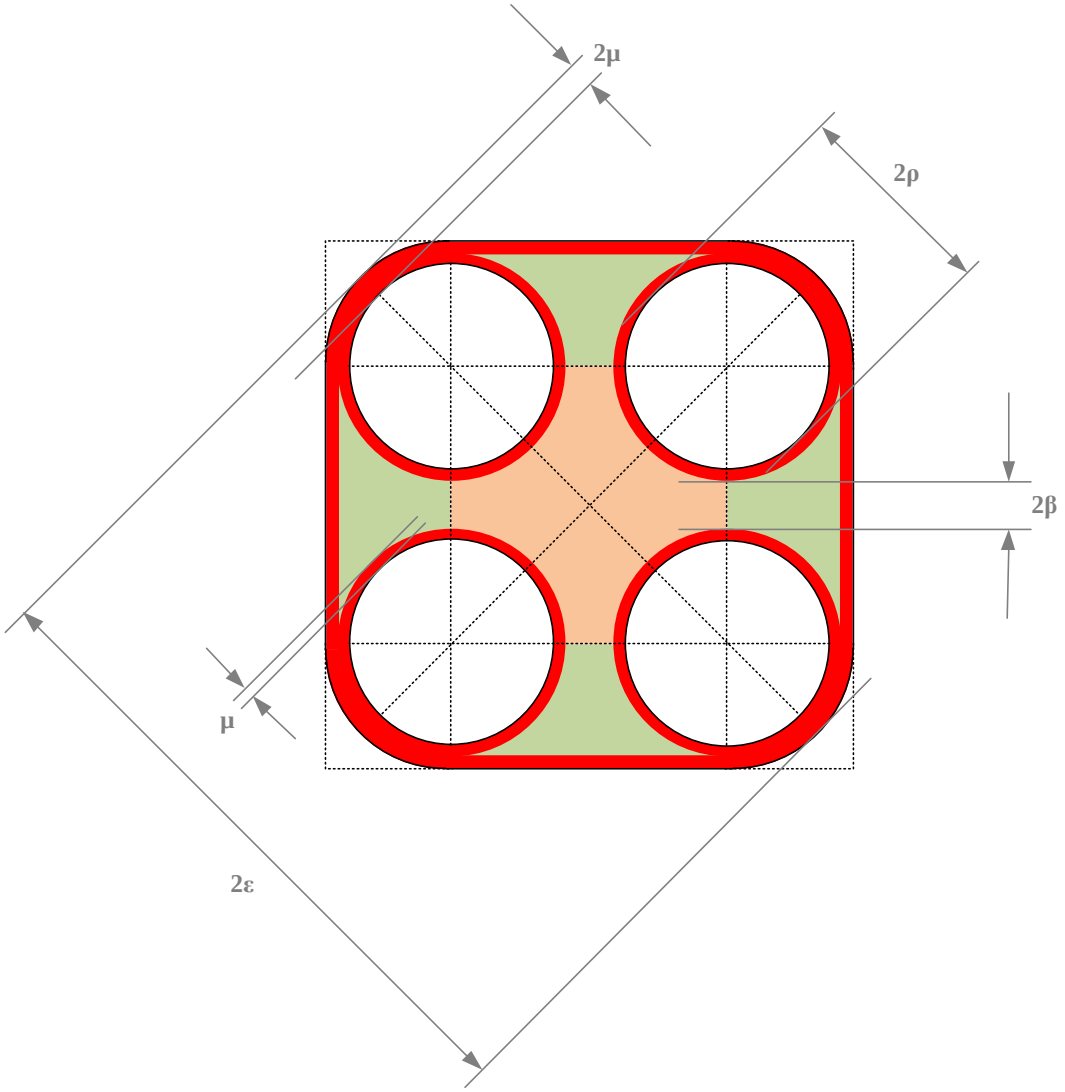


FIGURE C.3: Tapered square layout

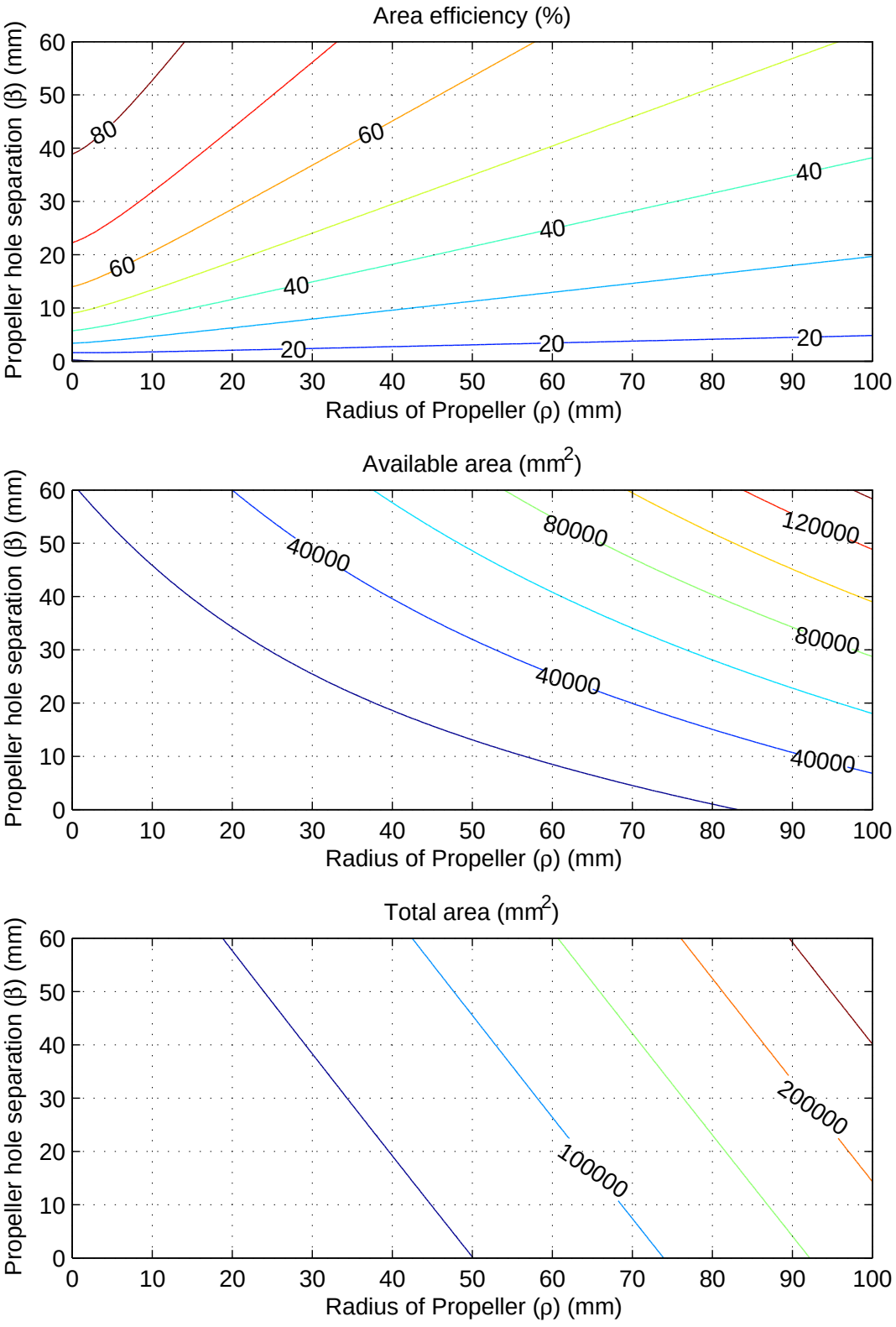


FIGURE C.4: Tapered square layout scaling

C.3 Clover

$$Usable\,area \approx (4\rho+4\mu+2\beta)^2-(4-\pi)(\rho+\mu)^2-4\pi(\rho+\mu)^2-\delta(\rho+\mu+\beta) \quad \text{(C.5)}$$

$$Total\,area \approx (4\rho+6\mu+2\beta)^2-(4-\pi)(\rho+2\mu)^2-\delta(\rho+\mu+\beta) \quad \text{(C.6)}$$

C.4 Graph generation

```
%Ashley Robinson
%08/12/12
%Layout.m

clear%last run

%-----Small range
mu = 5 %edge thickness(mm)
delta = 15 %you are receding(mm)

%-----PCB parameter vectors
rho = [0:0.5:100]%(mm)
beta = [0:0.5:60]%(mm)

y = 1;
for i=rho
   x = 1;
   for j=beta
   epsilon = (2.414*i) + (3.414*mu) + (1.414*j);

   top_circ(x,y) = (pi*((epsilon - mu)^2)) - (4*pi*((i + mu)^2));
   bottom_circ(x,y) = pi*epsilon*epsilon;
   eff_circ(x,y) = 100*top_circ(x,y)/bottom_circ(x,y);

   top_roun(x,y) = ((4*i +4*mu+2*j)^2)-((4-pi)*((i+mu)^2))-(4*pi*((i + mu)^2));
   bottom_roun(x,y) = ((4*i + 6*mu + 2*j)^2) - ((4 - pi)*((i +(2* mu))^2));
   eff_roun(x,y) = 100*top_roun(x,y)/bottom_roun(x,y);

   top_clov(x,y) = ((4*i+4*mu+2*j)^2)-((4-pi)*((i+mu)^2))-(4*pi*((i mu)^2))-(
    delta*(i+mu+j));
   bottom_clov(x,y) = ((4*i+6*mu+2*j)^2)-(( -pi)*((i+(2*mu))^2))-(delta*(i+mu+j))
    ;
   eff_clov(x,y) = 100*top_clov(x,y)/bottom_clov(x,y);

   x = x + 1;
   end
   y = y + 1;
end

figure(1);
%----------Circular
subplot(3,1,1);
```

```
41 [C,h] = contour(rho,beta,eff_circ);
42 set(h,'showtext','on','TextStep',get(h,'LevelStep')*2);
43 grid on
44 title('Area efficiency (%)');
45 xlabel('Radius of Propeller (\rho) (mm)');
46 ylabel('Propeller hole separation (\beta) (mm)');
47
48 subplot(3,1,2);
49 [C,h] = contour(rho,beta,top_circ);
50 set(h,'showtext','on','TextStep',get(h,'LevelStep')*2);
51 grid on
52 title('Available area (mm^2)');
53 xlabel('Radius of Propeller (\rho) (mm)');
54 ylabel('Propeller hole separation (\beta) (mm)');
55
56 subplot(3,1,3);
57 [C,h] = contour(rho,beta,bottom_circ);
58 set(h,'showtext','on','TextStep',get(h,'LevelStep')*2);
59 grid on
60 title('Total area (mm^2)');
61 xlabel('Radius of Propeller (\rho) (mm)');
62 ylabel('Propeller hole separation (\beta) (mm)');
63
64
65
66 figure(2)
67 %----------Round
68 subplot(3,1,1);
69 [C,h] = contour(rho,beta,eff_roun);
70 set(h,'showtext','on','TextStep',get(h,'LevelStep')*2);
71 grid on
72 title('Area efficiency (%)');
73 xlabel('Radius of Propeller (\rho) (mm)');
74 ylabel('Propeller hole separation (\beta) (mm)');
75
76 subplot(3,1,2);
77 [C,h] = contour(rho,beta,top_roun);
78 set(h,'showtext','on','TextStep',get(h,'LevelStep')*2);
79 grid on
80 title('Available area (mm^2)');
81 xlabel('Radius of Propeller (\rho) (mm)');
82 ylabel('Propeller hole separation (\beta) (mm)');
83
84 subplot(3,1,3);
85 [C,h] = contour(rho,beta,bottom_roun);
86 set(h,'showtext','on','TextStep',get(h,'LevelStep')*2);
87 grid on
88 title('Total area (mm^2)');
89 xlabel('Radius of Propeller (\rho) (mm)');
90 ylabel('Propeller hole separation (\beta) (mm)');
91
92
93 figure(3)
94 %----------Clover
95 subplot(3,1,1)
96 [C,h] = contour(rho,beta,eff_clov);
97 set(h,'showtext','on','TextStep',get(h,'LevelStep')*2);
98 grid on
```

```
 99 title('Area efficiency (%)');
100 xlabel('Radius of Propeller (\rho) (mm)');
101 ylabel('Propeller hole separation (\beta) (mm)');
102
103 subplot(3,1,2);
104 [C,h] = contour(rho,beta,top_clov);
105 set(h,'showtext','on','TextStep',get(h,'LevelStep')*2);
106 grid on
107 title('Available area (mm^2)');
108 xlabel('Radius of Propeller (\rho) (mm)');
109 ylabel('Propeller hole separation (\beta) (mm)');
110
111 subplot(3,1,3);
112 [C,h] = contour(rho,beta,bottom_clov);
113 set(h,'showtext','on','TextStep',get(h,'LevelStep')*2);
114 grid on
115 title('Total area (mm^2)');
116 xlabel('Radius of Propeller (\rho) (mm)');
117 ylabel('Propeller hole separation (\beta) (mm)');
```

Listing C.1: Scaling comparison graph generation

Appendix D

Battery comparison

Vendor	**Mass (g)**	**Energy (KJ)**	**Energy Density (J/g)**
Sparkfun	2.65	1.4652	552.9056604
OneCall	7	3.996	570.8571429
Sparkfun	9	5.4	600
OneCall	13	7.992	614.7692308
Sparkfun	18.5	11.322	612
Sparkfun	22	13.32	605.4545455
OneCall	23	17.316	752.8695652
Sparkfun	36	26.64	740
OneCall	41	17.982	438.5853659
Sparkfun	85	26.64	313.4117647
OneCall	85	39.96	470.1176471
Sparkfun	110	79.92	726.5454545
Sparkfun	114	59.94	525.7894737
OneCall	129	58.608	454.3255814
Sparkfun	206	58.608	284.5048544

Table D.1: A sample of lithium polymer batteries on the market

Appendix E

Software development

```
************************************************
LOG FILE

User:           Ashley
Computer Name: ASHLEY-PC
Date:           08-04-13
Time:           1438-14
************************************************
=> Entry:         0
   Date:          1438-19
   Time elapsed: 0.8s
   Xais angle:    0
   Yaxis angle:   -17

=> Entry:         1
   Date:          1438-19
   Time elapsed: 0.9s
   Xais angle:    0
   Yaxis angle:   -37

=> WARNING ============> Entry:         2
                         Date:          1438-19
                         Time elapsed: 1s
                         Xais angle:    0
                         Yaxis angle:   -48
   Message:
   Yaxis has exceeded 45 degrees tilt
```

Listing E.1: Example log file. Generated from user interface run in figure 3.8

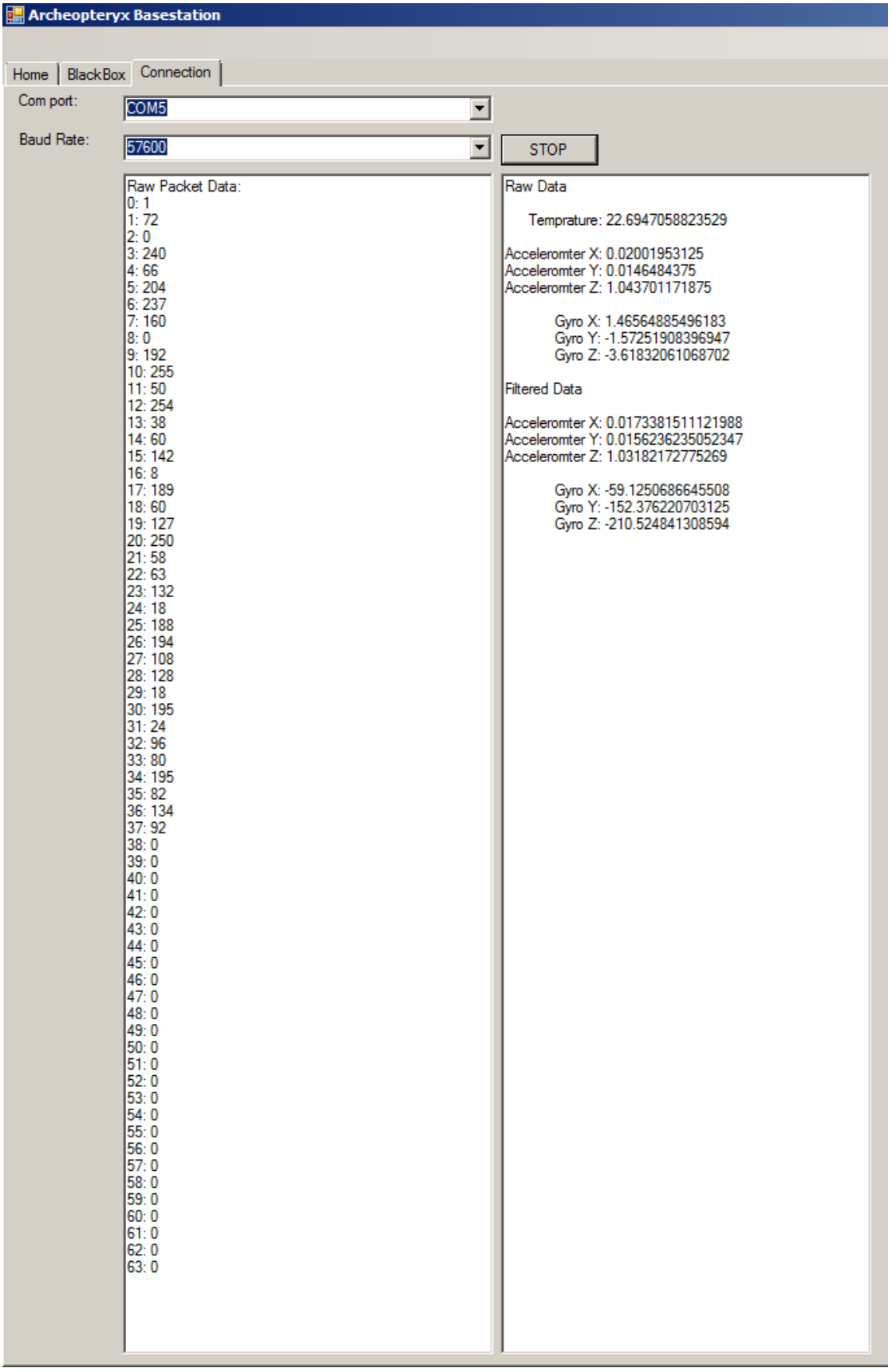


FIGURE E.1: User interface communications

Appendix F

Printed circuit board design files

F.1 Archeopteryx rev.A

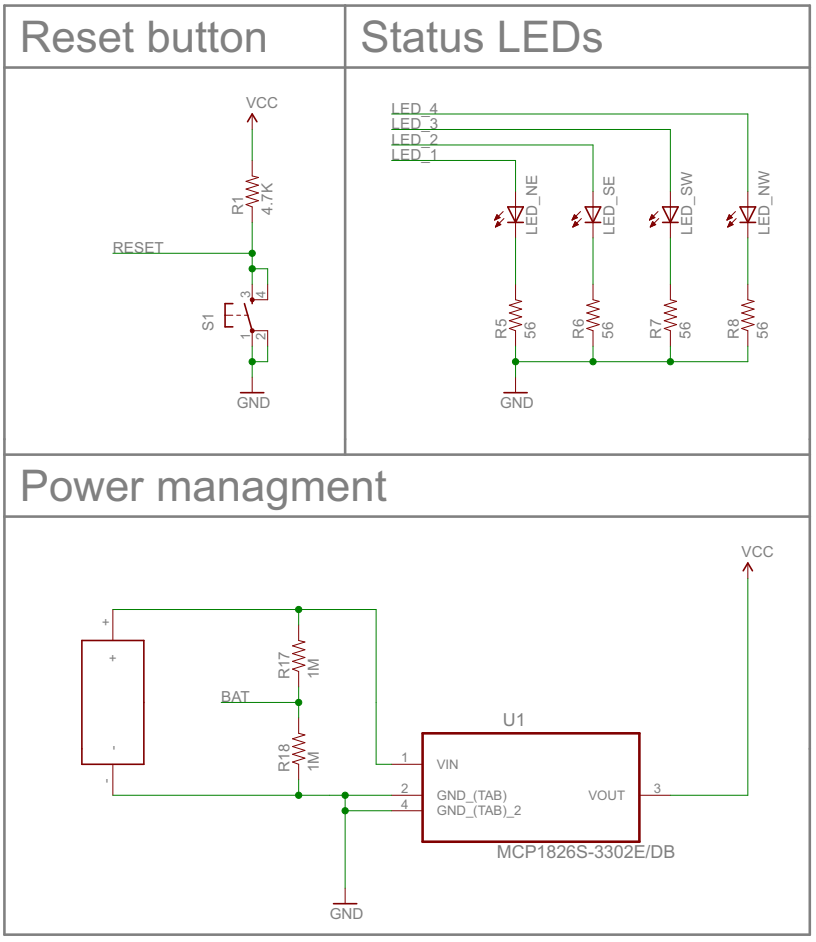


Figure F.1: Schematic page 1.

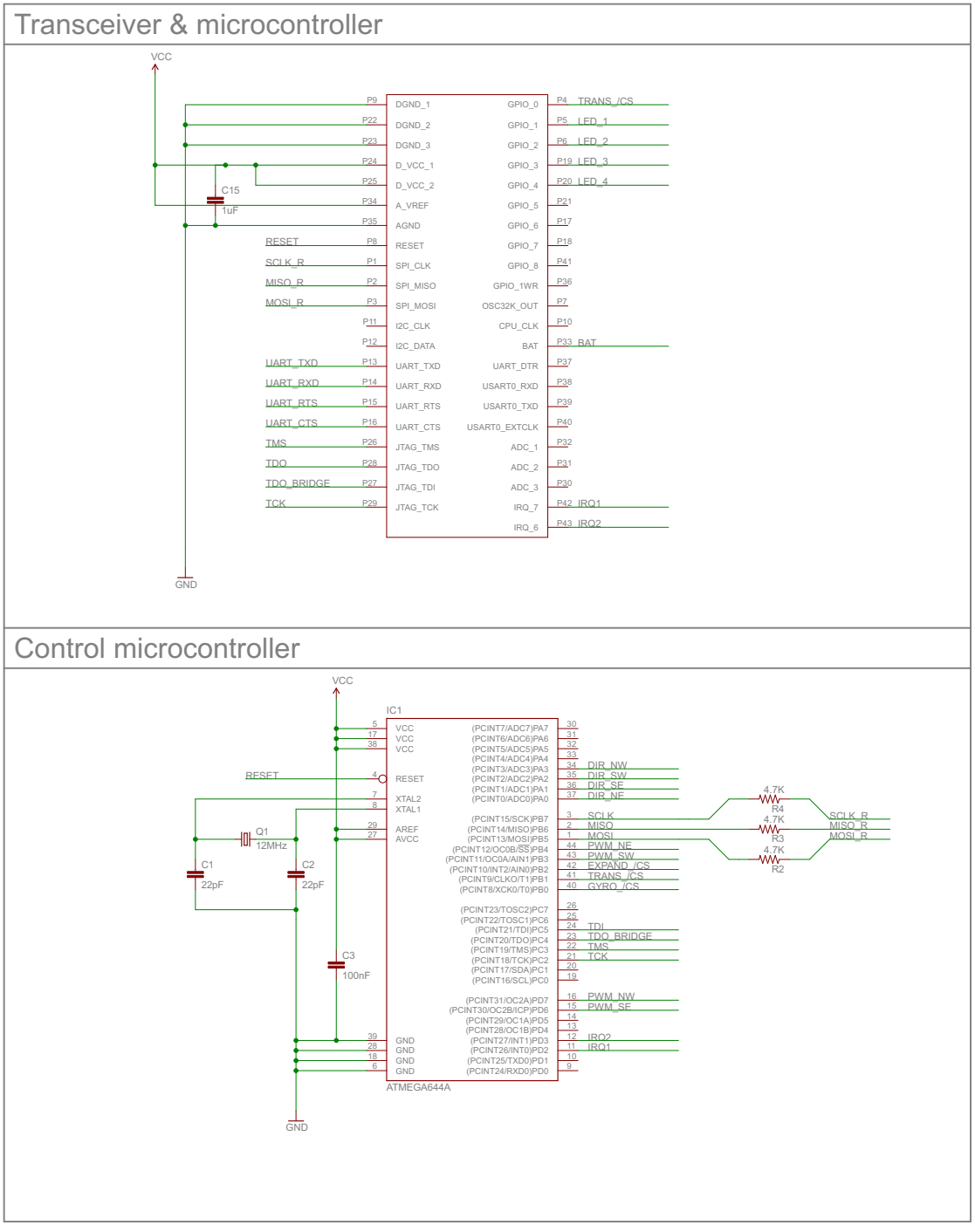


FIGURE F.2: Schematic page 2.

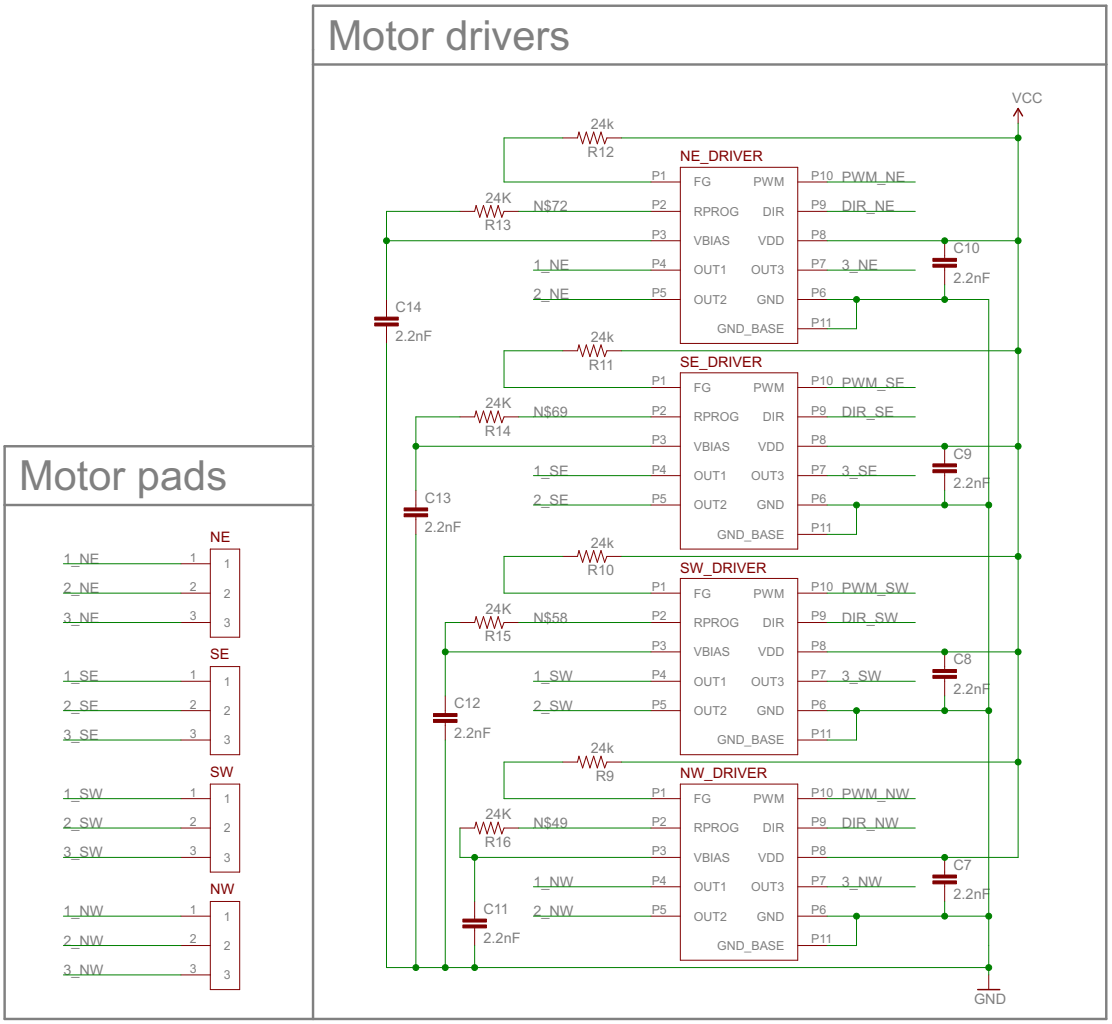


FIGURE F.3: Schematic page 3.

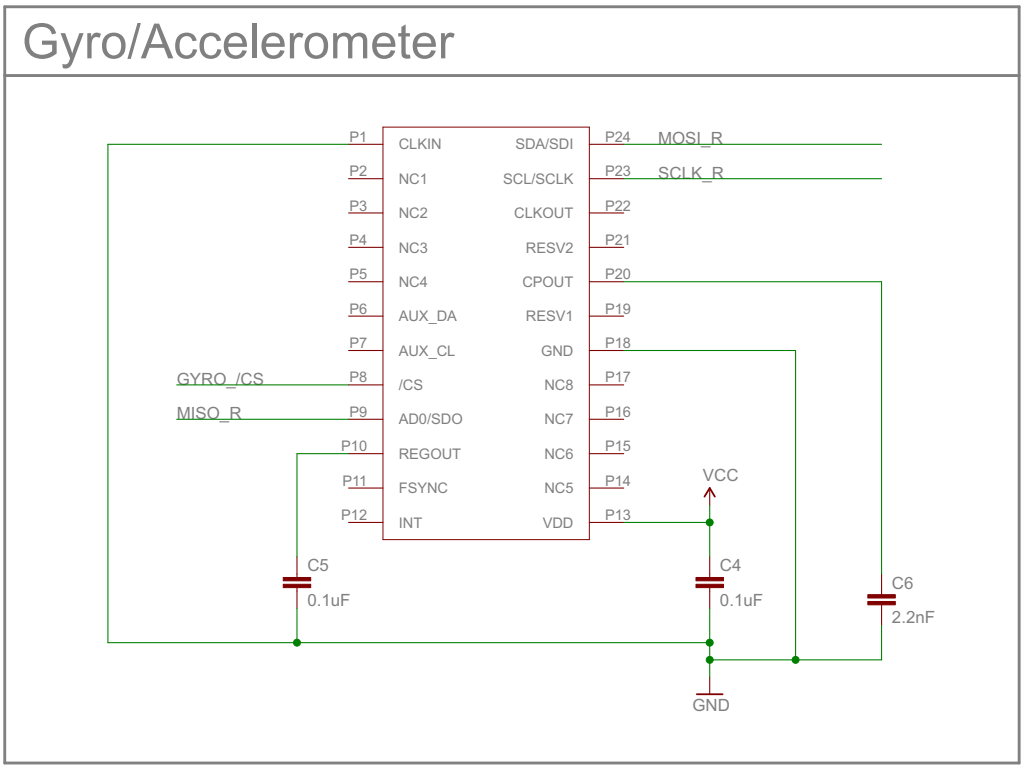


FIGURE F.4: Schematic page 4.

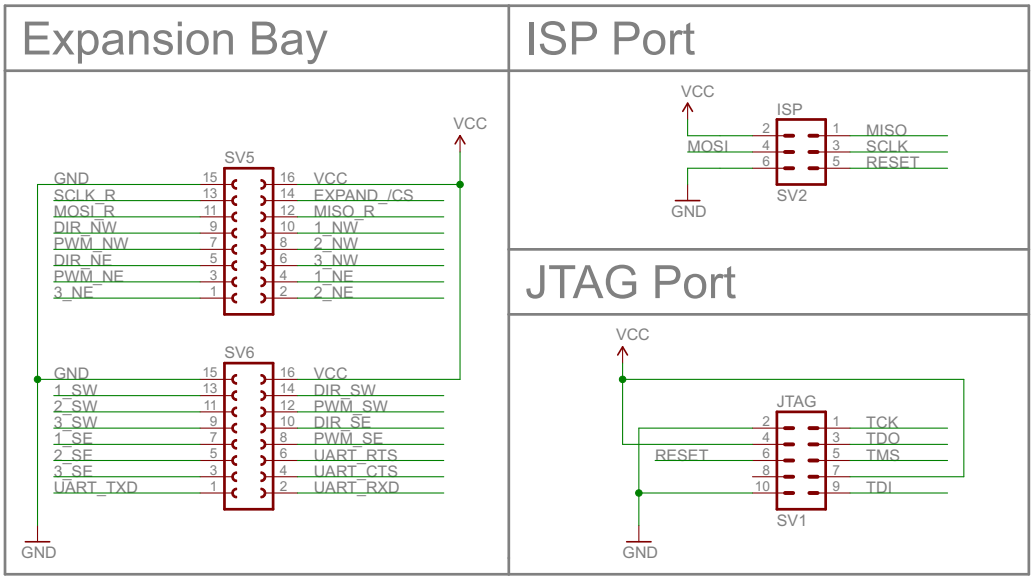


FIGURE F.5: Schematic page 5.

Component	**Function**	**Vendor**	**Quantity**	**Cost(Each)(£)**
ATMEL: ATMEGA1284P-AU	Microcontroller	OneCall	1	3.36
ATMEL: ATZB-24-A2	Microcontroller, Transceiver and Antenna	OneCall	1	22.78
Brushless Outrunner 7700kv	BLDC motors	ElectriFlite	4	8.92
INVENSENSE: MPU-6000	Gyroscope and Accelerometer	OneCall	1	28.73
Microchip: MTD6505 (Pack of 5)	Motor driver	RS Components	1	0.484
Microchip: MCP1825	Power mangagment	OneCall	1	0.68
PCB Fabrication	Mounting and circuitry (10 boards)	PCBCART	1	119.30
SILVERLIT R/C X-TWIN PLANE SPARE PROPELLERS (Pack of 2)	Propellers to be attached to motors	Wonderland Models	2	3.99
Sparkfun: 850mAh Battery	Power	Sparkfun	1	5.93
3D Printed Motor Housing	Structure	ECS: In house	4	25.80
			Total	328.13

Table F.1: Component list for revision A

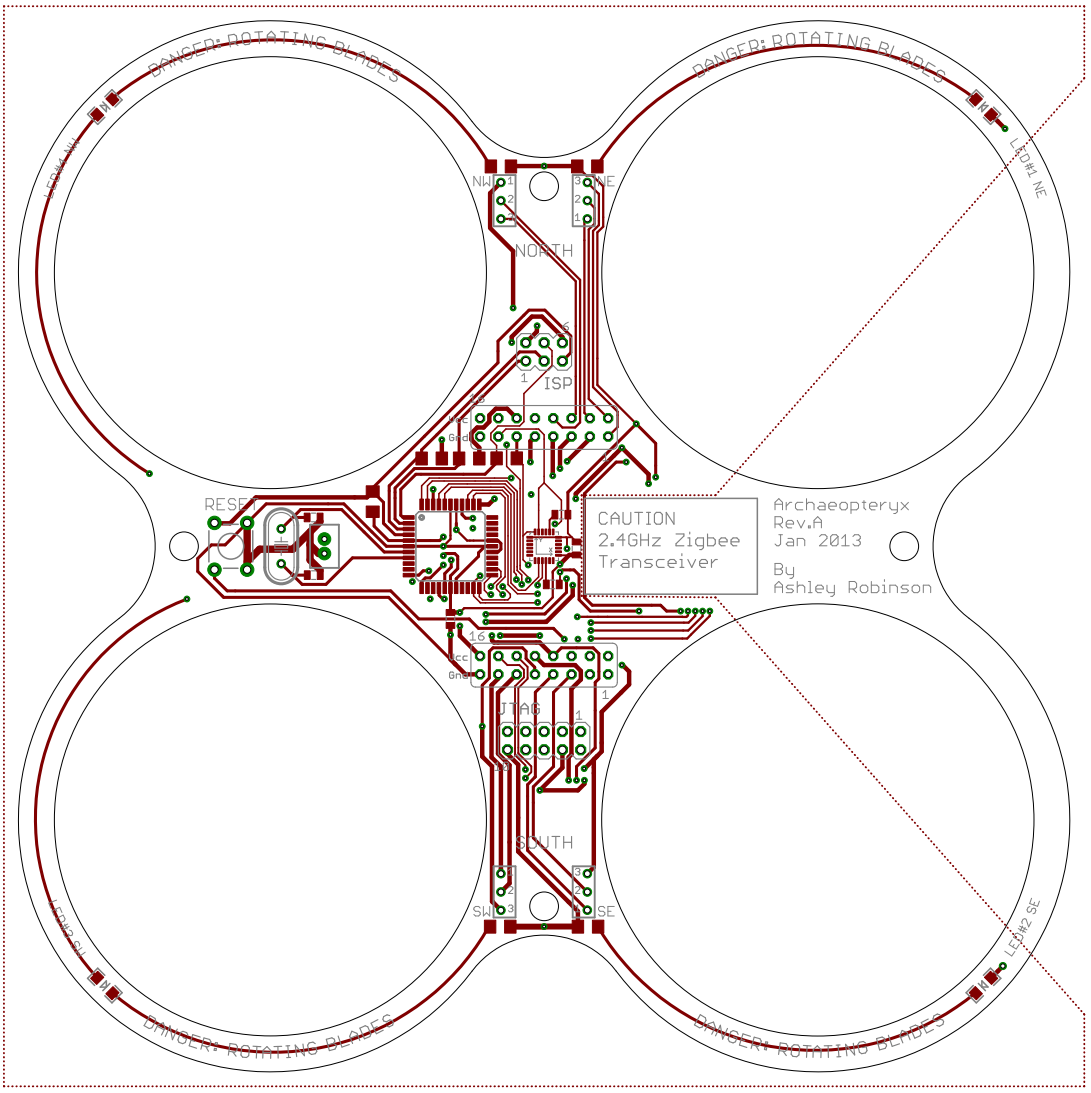


FIGURE F.6: Design view. Top copper layer looking down. Intermittent line represents ground polygon boundary.

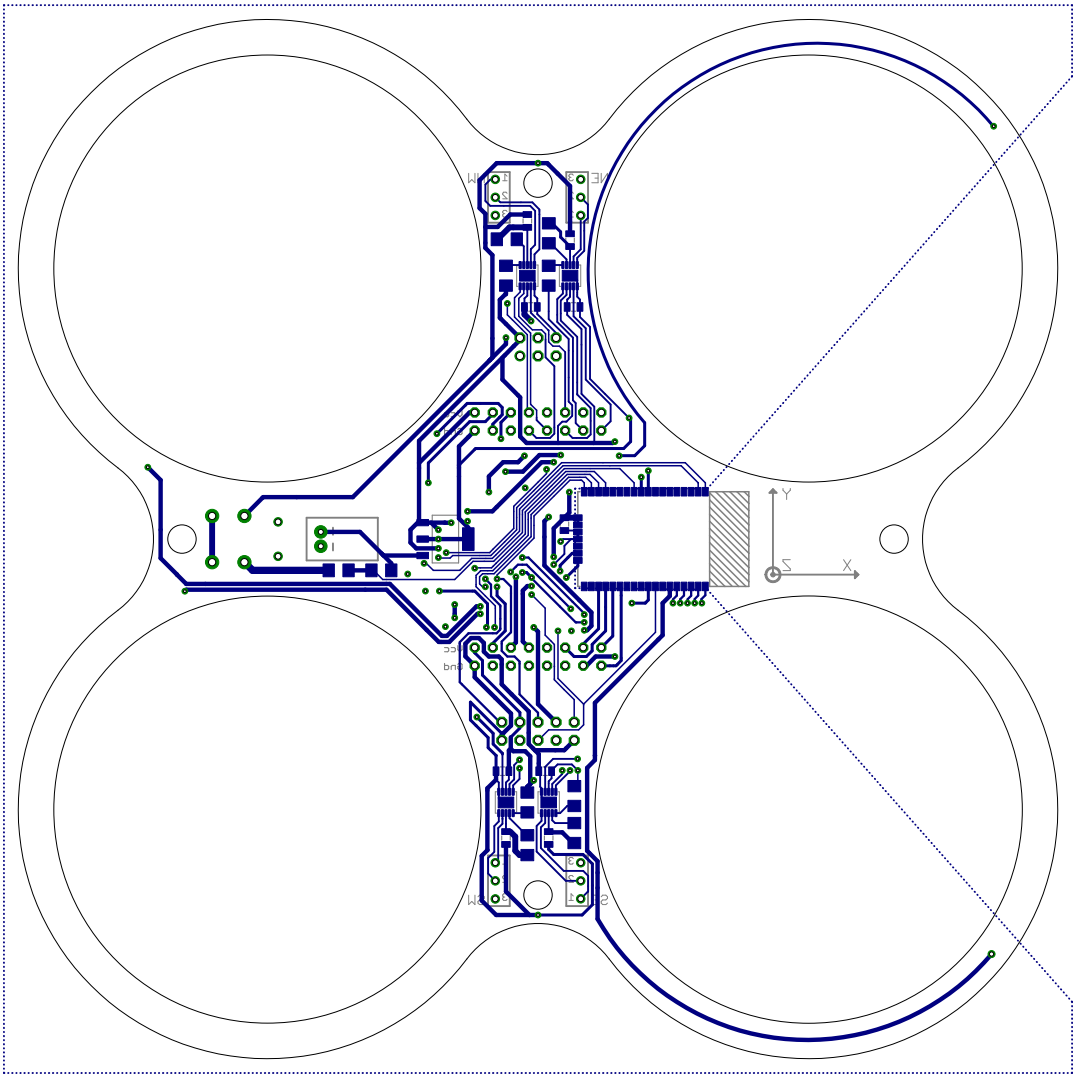


FIGURE F.7: Design view. Bottom copper layer looking down. Intermittent line represents ground polygon boundary.

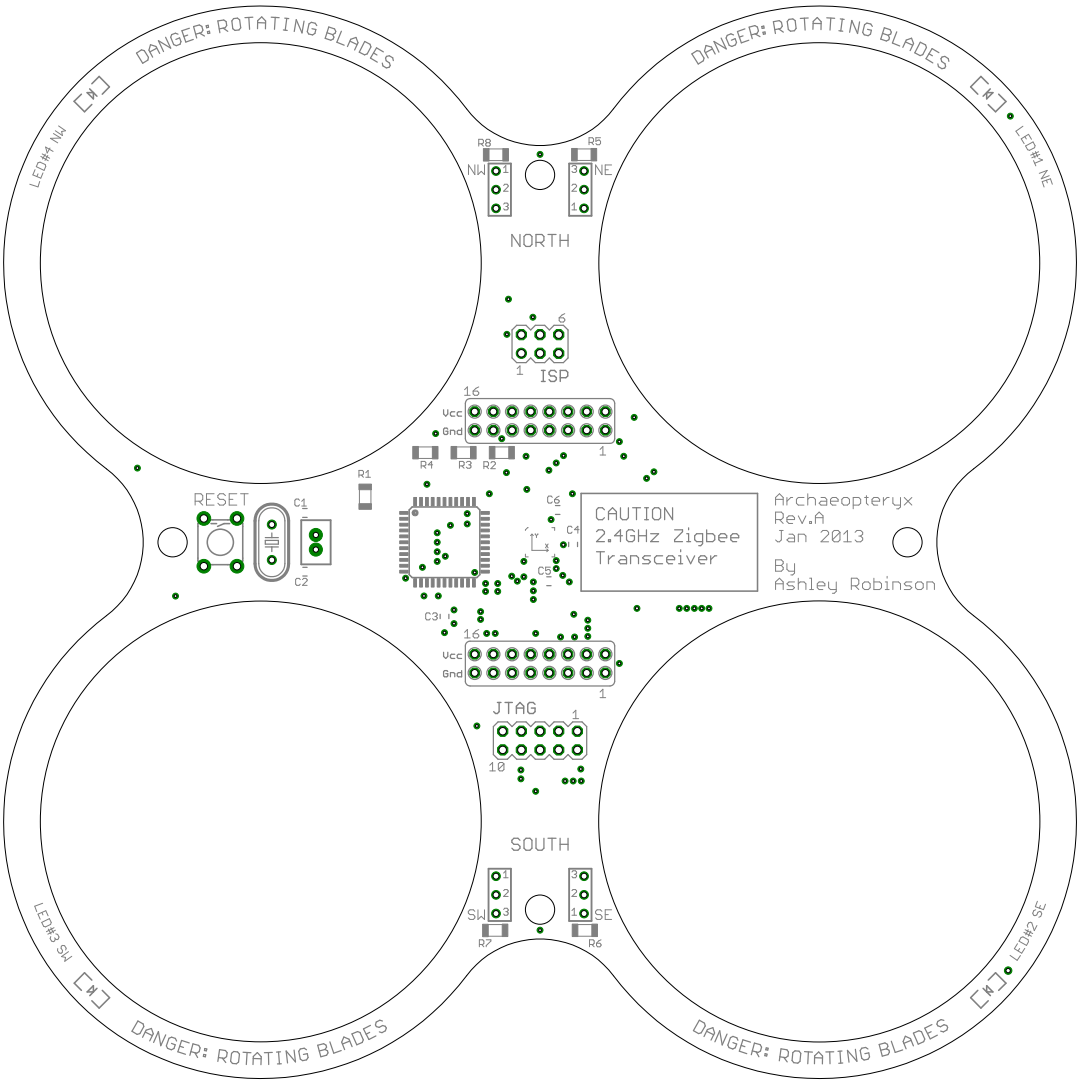


FIGURE F.8: Layout view. Top layer looking down.

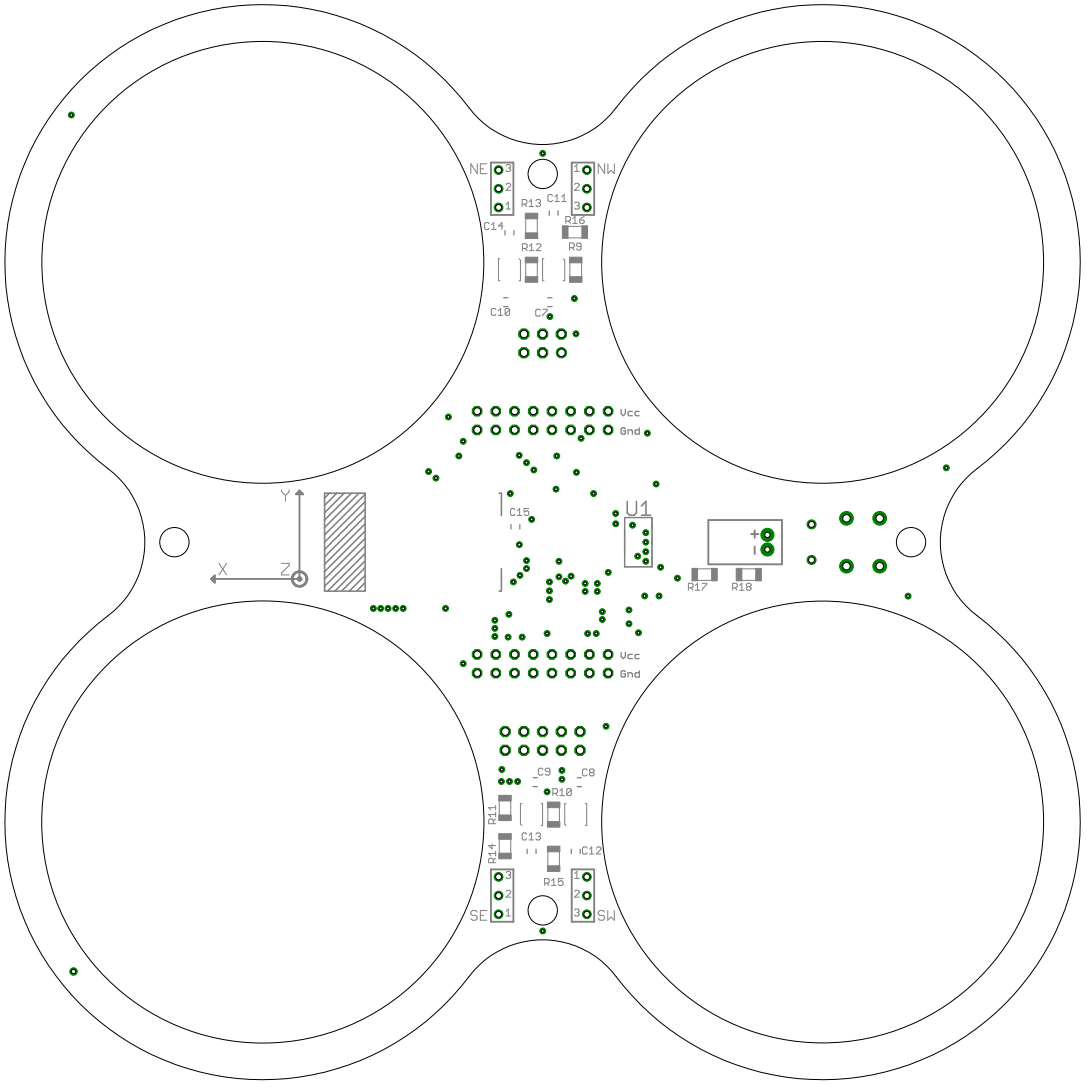


FIGURE F.9: Layout view. Bottom layer looking up.

F.2 Base station transceiver

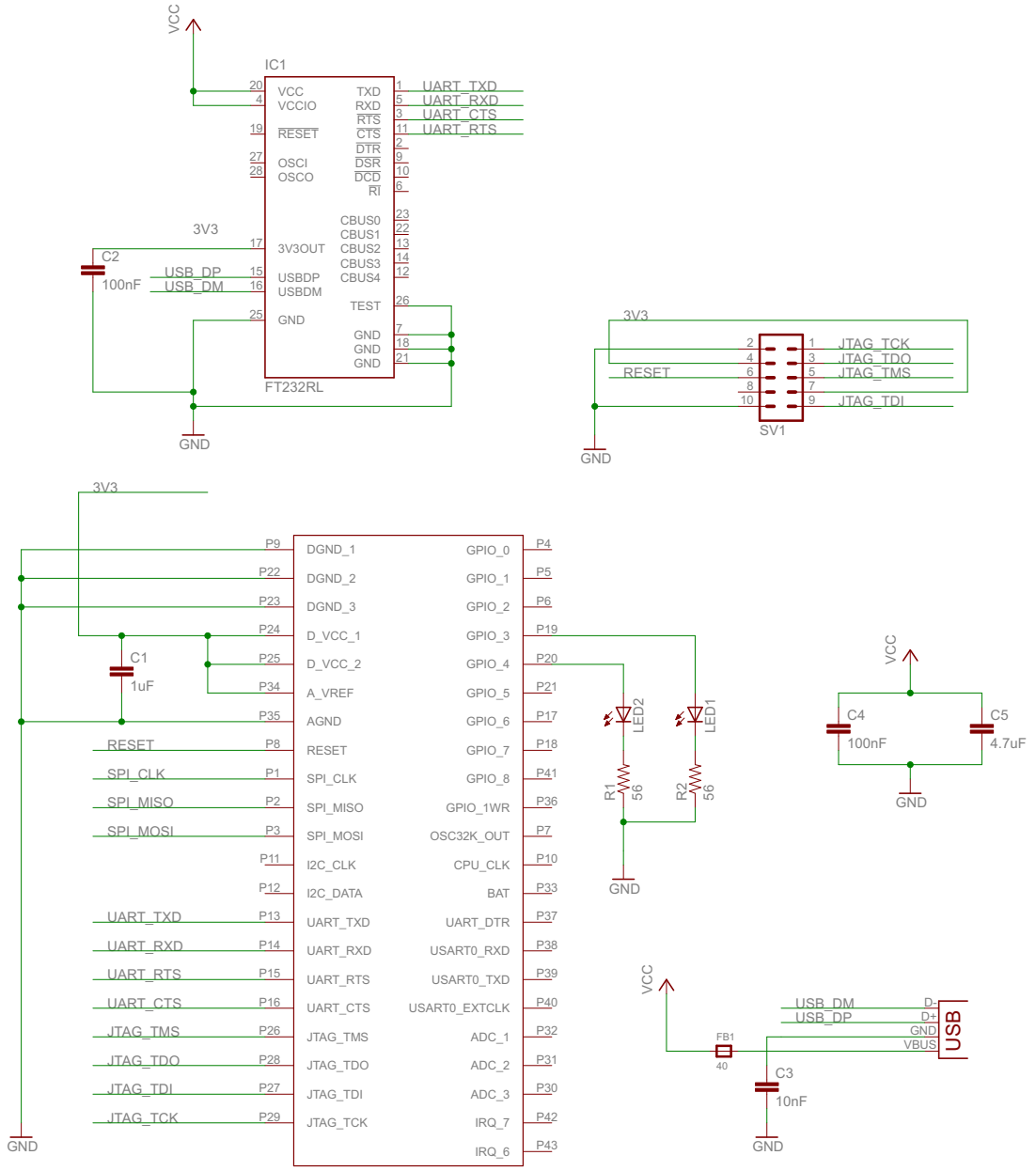


FIGURE F.10: Base station transceiver Schematic

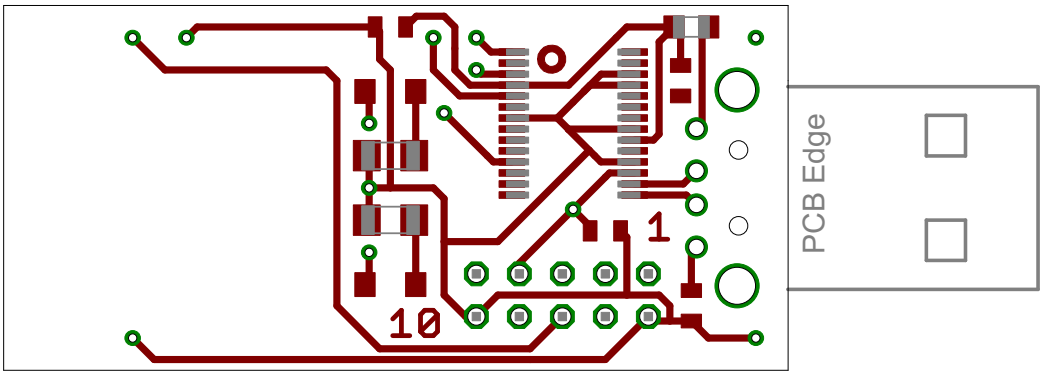


FIGURE F.11: Design view. Top copper layer looking down.

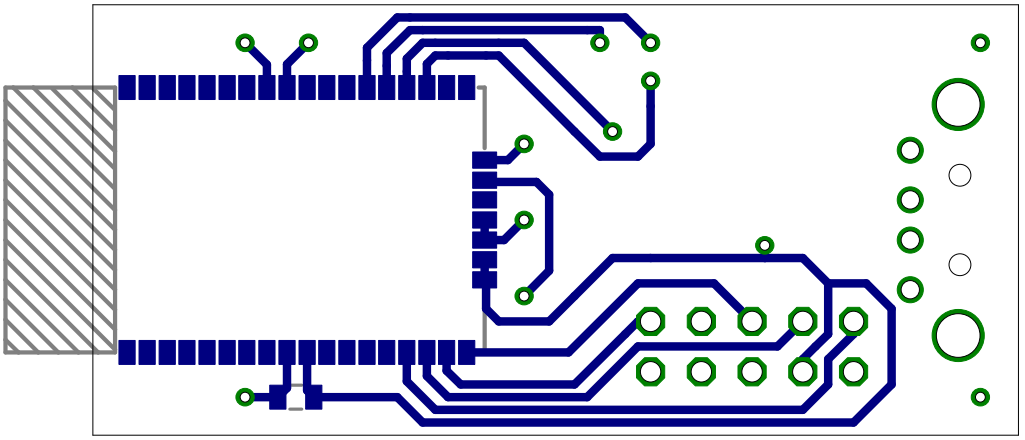


FIGURE F.12: Design view. Bottom copper layer looking down.

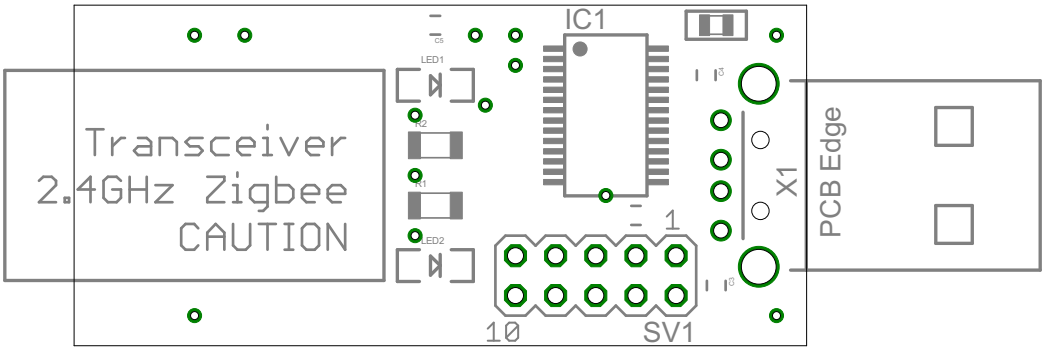


FIGURE F.13: Layout view. Top layer looking down.

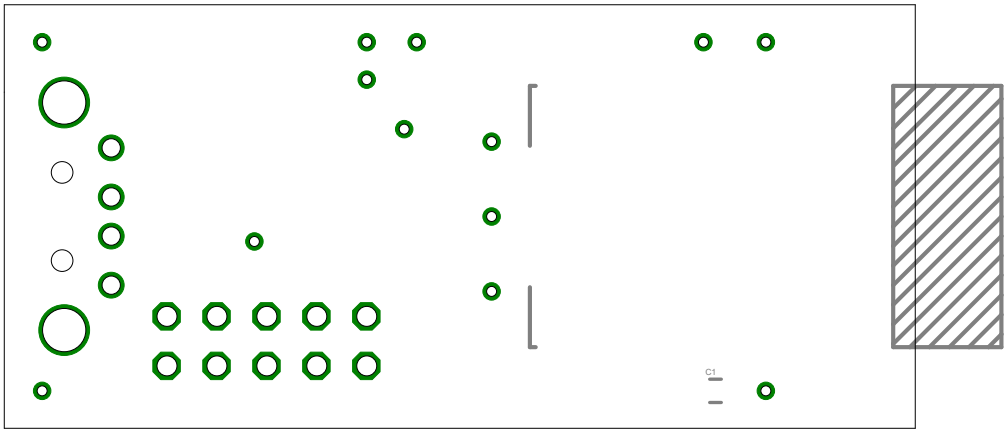


FIGURE F.14: Layout view. Bottom layer looking up.

F.3 BlackBox

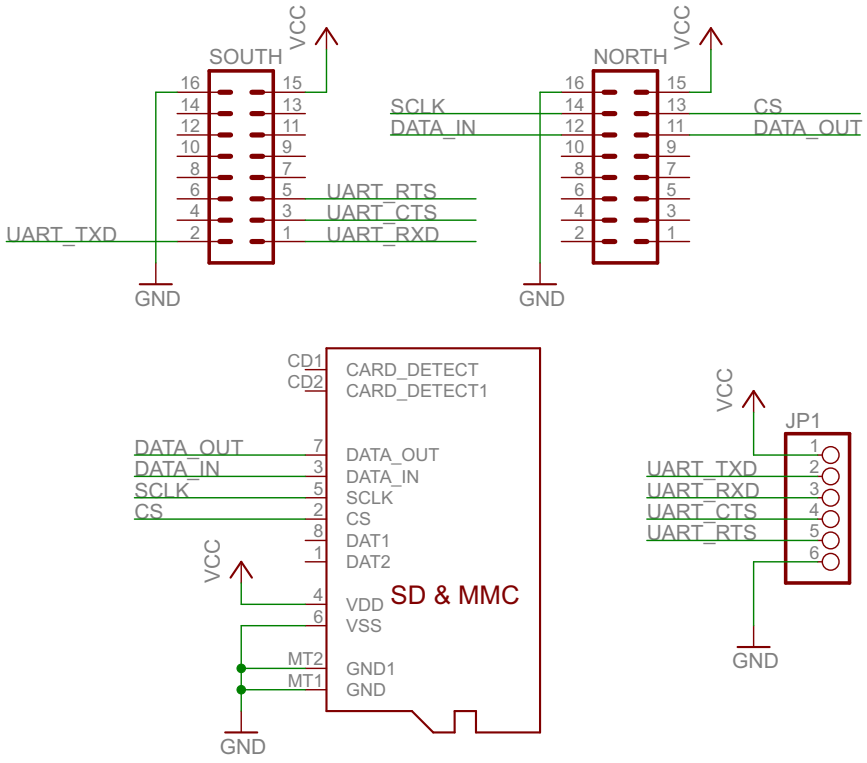


FIGURE F.15: BlackBox Schematic

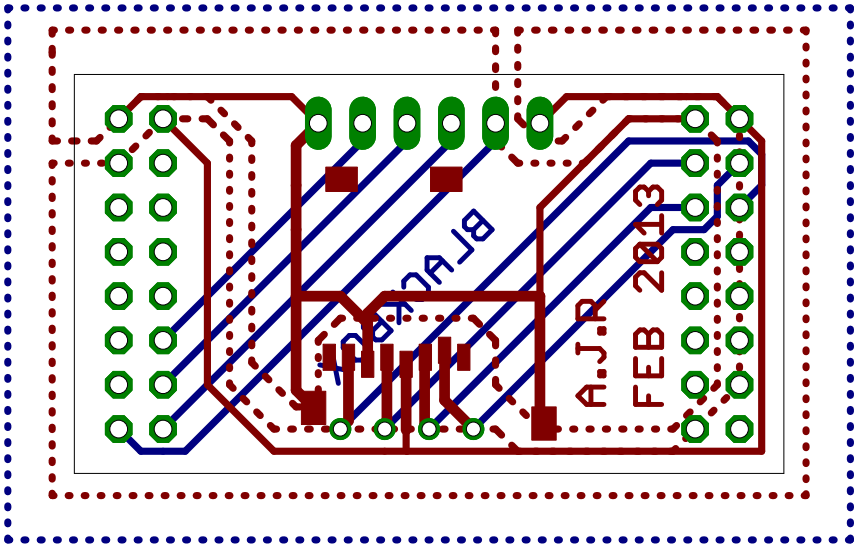
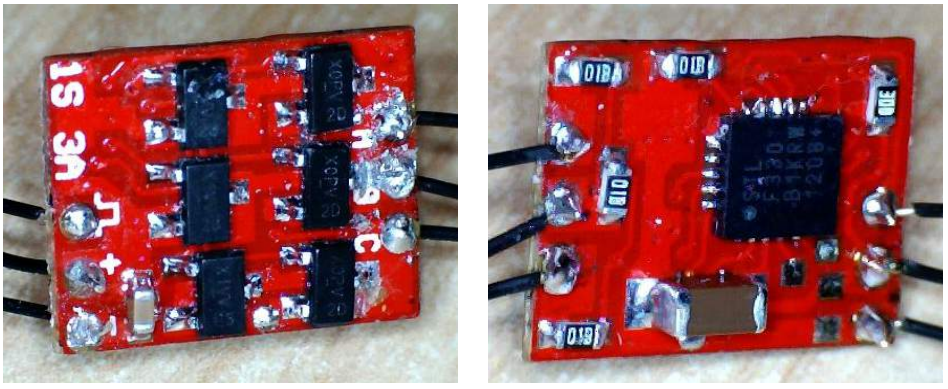


FIGURE F.16: BlackBox circuit board layout

Appendix G

Hardware modifications

FIGURE G.1: Three disconnected tracks on the board. Copper section removed with scalpel



(a) Back - MOSFETs (b) Front - microcontroller (silicon labs F330)

FIGURE G.2: HK - XP 3A 1S 0.7g micro brushless electronic speed controller

Appendix H

First propeller

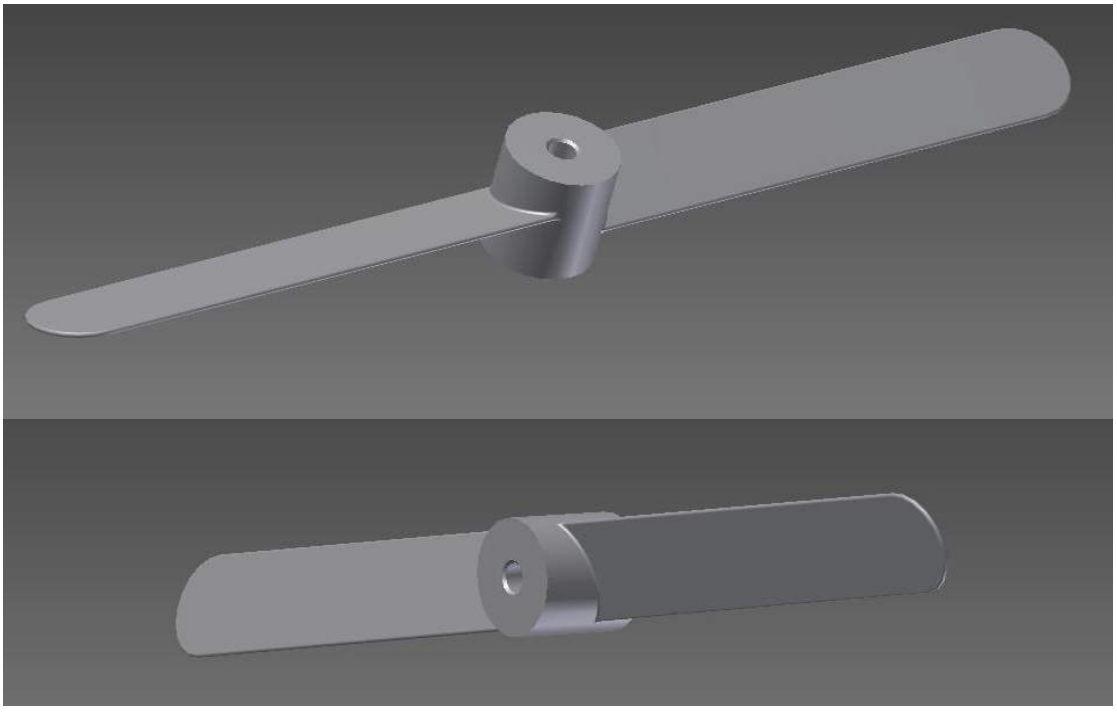


FIGURE H.1: First idea for a bespoke propeller

Appendix I

Project planning

I.1 Act I - actual

ID	Task Name	Start	Finish	Duration
1	Brief finalisation and submission	01/10/2012	12/10/2012	2w
2	Background research	01/06/2012	24/10/2012	20w 4d
3	Initial design	24/10/2012	30/11/2012	5w 3d
4	3D printer research	24/10/2012	07/11/2012	2w 1d
5	Area efficiency modelling	31/10/2012	21/11/2012	3w 1d
6	Component prototyping	31/10/2012	21/12/2012	7w 3d
7	First 3D printer prototype	21/11/2012	28/11/2012	1w 1d
8	Design optimisation	21/11/2012	05/12/2012	2w 1d
9	Rev.A Quadcopter PCB design	05/12/2012	09/01/2013	5w 1d
10	Progress report creation and population	07/11/2012	12/12/2012	5w 1d
11	Second Examiner meeting and feedback	21/11/2012	21/11/2012	0w

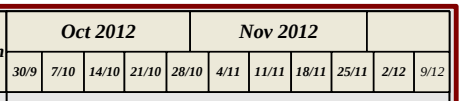


FIGURE I.1: The actual activities carried out from 30/09/12 to 10/12/12

I.2 Act II - planned

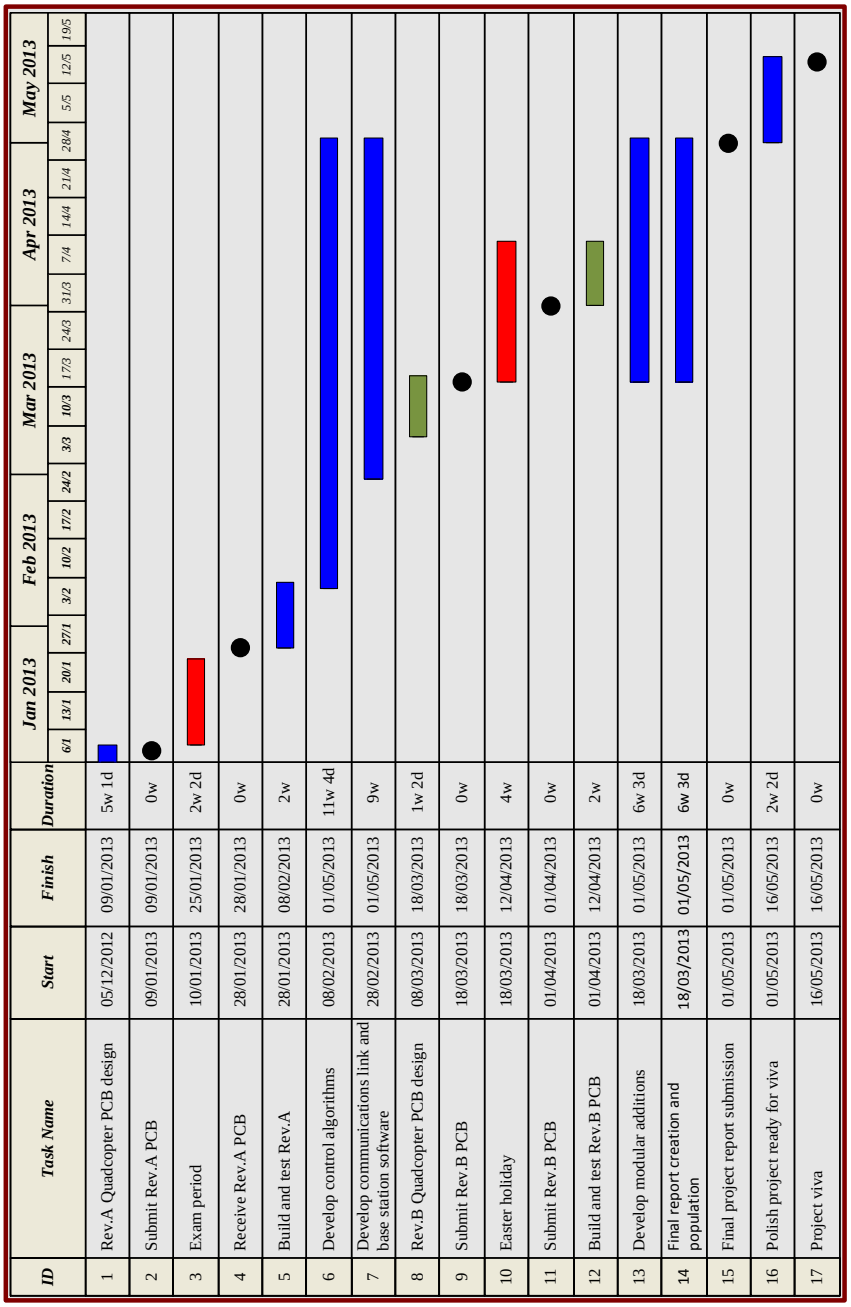


FIGURE I.2: The planned activities from 06/01/13 to 19/05/13

I.3 Act II - actual

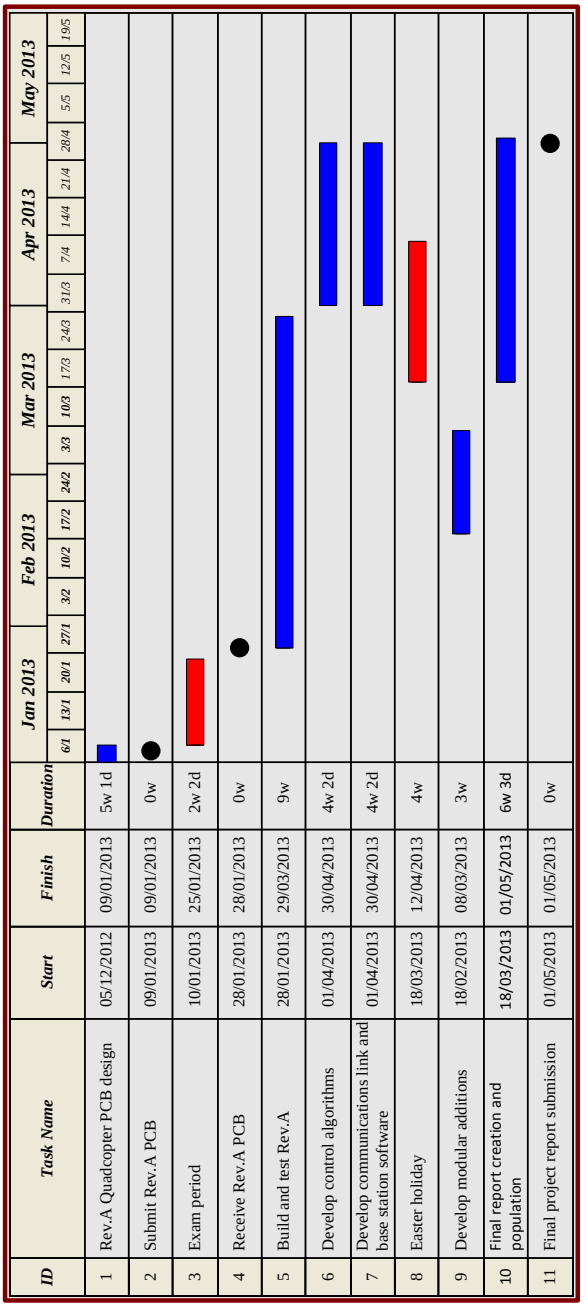


FIGURE I.3: The actual activities that took place from 06/01/13 to 01/05/13

Appendix J

Firmware listings

```
1  /*
2   * Control.c
3   *
4   * Created: 11/04/2013 02:19:13
5   *  Author: Ashley
6   */
7
8  #include "Control.h"
9
10 //Control loop
11 ISR(TIMER1_COMPA_vect){
12         PORTB |= _BV(PB2);//Use Exapnd_/CS to check control loop time
13         //Get new sample
14         UpdateSample();
15         //filter
16         accel_filtered_x = filter_kernel(AccelX,AccelX_pipe,low_coeffcients)/
       accel_scale;
17         accel_filtered_y = filter_kernel(AccelY,AccelY_pipe,low_coeffcients)/
       accel_scale;
18         accel_filtered_z = filter_kernel(AccelZ,AccelZ_pipe,low_coeffcients)/
       accel_scale;
19         gyro_filtered_x = filter_kernel(GyroX,GyroX_pipe,high_coeffcients)/
       gyro_scale;
20         gyro_filtered_y = filter_kernel(GyroY,GyroY_pipe,high_coeffcients)/
       gyro_scale;
21         gyro_filtered_z = filter_kernel(GyroZ,GyroZ_pipe,high_coeffcients)/
       gyro_scale;
22         //reconstruct
23         X = accel_filtered_x + gyro_filtered_x;
24         Y = accel_filtered_y + gyro_filtered_y;
25         Z = accel_filtered_z + gyro_filtered_z;
26         //orientate
27         pitch = atan2(X,(sqrt(square(Y) + square(Z))));
28         roll = atan2(Y,sqrt(square(X) + square(Z)));
29         yaw = atan2(Z,sqrt(square(X) + square(Y)));
30         speed[0] = 0;
31         speed[1] = 0;
32         speed[2] = 0;
33         speed[3] = 0;
```

```
34          //Calc speeds
35
36          //Update motors
37          PWM_NW_motor(speed[0]);
38          PWM_SE_motor(speed[1]);
39          PWM_SW_motor(speed[2]);
40          PWM_NE_motor(speed[3]);
41          //time stamp update
42          time_stamp++;
43          PORTB &= ~_BV(PB2);
44  }
45
46  int main(void)
47  {
48          uint8_t i;
49          uint8_t guess;
50          //volatile uint8_t command[] = {2,2,2,2};
51
52          setup_outputs();
53          init_motors();
54          init_uart();
55          MasterInit();
56          init_MPU6000();
57          init_sample_timer();
58          init_motor_timers();
59
60          uart_tx(0x00);//Tell transceiver you're ready
61          time_stamp = 0;
62
63          sei();
64
65          while(1)
66          {
67                  //Package data between interrupts, not vital
68                  //COntro stuff
69                  pack_a_double(14,accel_filtered_x);
70                  pack_a_double(18,accel_filtered_y);
71                  pack_a_double(22,accel_filtered_z);
72                  pack_a_double(26,gyro_filtered_x);
73                  pack_a_double(30,gyro_filtered_y);
74                  pack_a_double(34,gyro_filtered_z);
75                  pack_a_double(38,pitch);
76                  pack_a_double(42,roll);
77                  pack_a_double(46,yaw);
78                  //integer time stamp
79                  pack_an_int(50,time_stamp);
80                  //motor speeds
81                  buffer[54] = speed[0];
82                  buffer[55] = speed[1];
83                  buffer[56] = speed[2];
84                  buffer[57] = speed[3];
85                  //58 hold battery, put in transceiver
86                  //59..63 => user/debug
87                  //Send it
88
89                  if(PINB & _BV(PB1)){//PB1 is high
90                          for(i =0;i< packet_size;i++){
91                                  uart_tx(buffer[i]);//Send packet
```

```
92                          }
93                  }
94          }
95  }
96
97  void pack_a_double(uint8_t start,double value){
98          conversion breakdown;
99          breakdown.fla = (float)value;//Cast to float
100         buffer[start] = breakdown.bytes.byte3;
101         buffer[start + 1] = breakdown.bytes.byte2;
102         buffer[start + 2] = breakdown.bytes.byte1;
103         buffer[start + 3] = breakdown.bytes.byte0;
104 }
105 void pack_an_int(uint8_t start,uint32_t value){
106         conversion breakdown;
107         breakdown.integer = value;//Cast to float
108         buffer[start] = breakdown.bytes.byte3;
109         buffer[start + 1] = breakdown.bytes.byte2;
110         buffer[start + 2] = breakdown.bytes.byte1;
111         buffer[start + 3] = breakdown.bytes.byte0;
112 }
```

Listing J.1: Main from control microcontroller

```
1   /*
2    * TRANSCEIVER.c
3    *
4    * Created: 08/04/2013 23:55:02
5    *  Author: Ashley
6    */
7
8
9   #include "Transceiver_hardware.h"
10
11  #include <avr/io.h>
12  #include <avr/interrupt.h>
13  #include <util/delay.h>
14
15  #include "AT86RF230.h"
16
17  volatile uint8_t buffer[packet_size];
18  volatile uint8_t command[command_size];
19
20  int main(void){
21          setup_outputs();
22          init_spi_master();
23          init_uart_debug();
24          init_uart_control();
25          init_Bat_ADC();
26
27          PHY_INIT();//Clock sour of 8MHz from transceiver
28
29          command[0] = 0;
30          command[1] = 0;
31          command[2] = 0;
32          command[3] = 0;
33
34
```

```
35          uint8_t i = 0;
36          uint8_t guess = 0;
37          LEDs(0xFF);//Waiting to sync indicator
38          rx_control();//Wait for control to start
39      while(1){
40                  PORTB |= _BV(PB5);//Need packet therefore high
41                  for (i = 0;i < packet_size;i++){
42                          buffer[i] = rx_control();//Wait for control MCU to reply
43                          PORTB &= ~_BV(PB5);//push low, timeout looks for this
44                  }
45
46
47                  UpdateOrientation(buffer);
48                  buffer[58] = Read_Bat_ADC();
49
50                  if(rx_debug()){//if packet received before timeout
51                          for (i = 0;i < packet_size;i++){
52                                  tx_debug(buffer[i]);
53                          }
54                          for(i=0;i<command_size;i++){
55                                  command[i] = rx_debug();
56                          }
57                  }
58          }
59  }
```

Listing J.2: Main from transceiver microcontroller

Appendix K

Software listings

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.ComponentModel;
5  using System.Data;
6  using System.Drawing;
7  using System.Linq;
8  using System.Text;
9  using System.Windows.Forms;
10 using System.IO;
11 using System.IO.Ports;
12
13
14 namespace Archeopteryx
15 {
16     public partial class BaseStation : Form
17     {
18         private const uint packet_size = 64;
19         private const double bat_scale = 0.02578125;
20         byte[] buffer = new byte[packet_size];
21         byte[] command = new byte[4];
22         private uint counter = new uint();
23         private string path;
24         private SerialPort UART = new SerialPort();
25
26         private double temp, accelX, accelY, accelZ,gyroX,gyroY,gyroZ;
27         private double filtered_accelX,filtered_accelY,filtered_accelZ,
       filtered_gyroX,filtered_gyroY,filtered_gyroZ;
28         private double pitch, roll, yaw;
29         private UInt16 motor_NE, motor_SE, motor_SW, motor_NW;
30         private UInt32 iter_loop;
31
32         public BaseStation(){
33             InitializeComponent();
34             WindowState = FormWindowState.Maximized;
35             //com ports around?
36             if (SerialPort.GetPortNames().Length == 0){
37                 no_com_label.Visible = true;
38             }else{
```

```
39                  no_com_label.Visible = false;
40              }
41              //Default for UART
42              Baud_comboBox.SelectedIndex = 12;
43              //Look for files in same directory
44              path = Directory.GetCurrentDirectory();
45              path_label.Text = "File location: \n" + path;
46              //Start timer
47              counter = 0;
48              Step_timer.Enabled = true;
49              Step_timer.Start();
50              //Take chare of charts
51              Accel_chart.ChartAreas[0].AxisY.Maximum = 2;
52              Accel_chart.ChartAreas[0].AxisY.Minimum = -2;
53              Accel_chart.ChartAreas[0].AxisY.Interval = 1;
54              Accel_chart.ChartAreas[0].AxisY.Title = "Acceleration (g)";
55              Accel_chart.ChartAreas[0].AxisX.Maximum = 20;
56              Accel_chart.ChartAreas[0].AxisX.Minimum = 0;
57              Accel_chart.ChartAreas[0].AxisX.Interval = 10;
58              Accel_chart.ChartAreas[0].AxisX.Title = "Time [s]";
59              Accel_chart.ChartAreas[0].AxisX.LabelStyle.Enabled = false;
60              Gyro_chart.ChartAreas[0].AxisY.Maximum = 250;
61              Gyro_chart.ChartAreas[0].AxisY.Minimum = -250;
62              Gyro_chart.ChartAreas[0].AxisY.Interval = 50;
63              Gyro_chart.ChartAreas[0].AxisY.Title = "Angular Velocity (deg/s)";
64              Gyro_chart.ChartAreas[0].AxisX.Maximum = 20;
65              Gyro_chart.ChartAreas[0].AxisX.Minimum = 0;
66              Gyro_chart.ChartAreas[0].AxisX.Interval = 10;
67              Gyro_chart.ChartAreas[0].AxisX.Title = "Time [s]";
68              Gyro_chart.ChartAreas[0].AxisX.LabelStyle.Enabled = false;
69              Filtered_Accel_chart.ChartAreas[0].AxisY.Maximum = 2;
70              Filtered_Accel_chart.ChartAreas[0].AxisY.Minimum = -2;
71              Filtered_Accel_chart.ChartAreas[0].AxisY.Interval = 1;
72              Filtered_Accel_chart.ChartAreas[0].AxisY.Title = "Acceleration (g)";
73              Filtered_Accel_chart.ChartAreas[0].AxisX.Maximum = 20;
74              Filtered_Accel_chart.ChartAreas[0].AxisX.Minimum = 0;
75              Filtered_Accel_chart.ChartAreas[0].AxisX.Interval = 10;
76              Filtered_Accel_chart.ChartAreas[0].AxisX.Title = "Time [s]";
77              Filtered_Accel_chart.ChartAreas[0].AxisX.LabelStyle.Enabled = false;
78              Filtered_Gyro_chart.ChartAreas[0].AxisY.Maximum = 2;
79              Filtered_Gyro_chart.ChartAreas[0].AxisY.Minimum = -2;
80              Filtered_Gyro_chart.ChartAreas[0].AxisY.Interval = 1;
81              Filtered_Gyro_chart.ChartAreas[0].AxisY.Title = "Acceleration [g]";
82              Filtered_Gyro_chart.ChartAreas[0].AxisX.Maximum = 20;
83              Filtered_Gyro_chart.ChartAreas[0].AxisX.Minimum = 0;
84              Filtered_Gyro_chart.ChartAreas[0].AxisX.Interval = 10;
85              Filtered_Gyro_chart.ChartAreas[0].AxisX.Title = "Time [s]";
86              Filtered_Gyro_chart.ChartAreas[0].AxisX.LabelStyle.Enabled = false;
87              Attitude_chart.ChartAreas[0].AxisY.Maximum = 4;
88              Attitude_chart.ChartAreas[0].AxisY.Minimum = -4;
89              Attitude_chart.ChartAreas[0].AxisY.Interval = 1;
90              Attitude_chart.ChartAreas[0].AxisY.Title = "Filtered_Gyroeration [g]"
    ;
91              Attitude_chart.ChartAreas[0].AxisX.Maximum = 20;
92              Attitude_chart.ChartAreas[0].AxisX.Minimum = 0;
93              Attitude_chart.ChartAreas[0].AxisX.Interval = 10;
94              Attitude_chart.ChartAreas[0].AxisX.Title = "Time [s]";
95              Attitude_chart.ChartAreas[0].AxisX.LabelStyle.Enabled = false;
```

```
96
97
98              Motor_chart.ChartAreas[0].AxisY.Maximum = 100;
99              Motor_chart.ChartAreas[0].AxisY.Minimum = 0;
100             Motor_chart.ChartAreas[0].AxisY.Interval = 10;
101             Motor_chart.ChartAreas[0].AxisY.Title = "Speed (%)";
102             Motor_chart.ChartAreas[0].AxisX.Maximum = 5;
103             Motor_chart.ChartAreas[0].AxisX.Minimum = 0;
104             Motor_chart.ChartAreas[0].AxisX.Interval = 1;
105             //labels
106             bat_label.Text = "Battery";
107             temp_label.Text = "Temperature";
108             iter_label.Text = "Control loop iteration: 0";
109         }
110         private void Files_comboBox_Click(object sender, EventArgs e){
111             string[] files = Directory.GetFiles(path);//Find files in that
        directory
112             Files_comboBox.Items.Clear();//Get rid of last add
113             foreach (string file in files){
114                 Files_comboBox.Items.Add(file.Replace(path,""));//Remove the
        directory path when selecting
115             }
116         }
117         private void Load_button_Click(object sender, EventArgs e){
118             byte[] load_buffer = File.ReadAllBytes(path + Files_comboBox.Text);
119
120             raw_log_textBox.Text = load_buffer.Length.ToString();
121
122         }
123         static double CalcTemp(byte upper, byte lower){
124             return ((double)Smash_Two_Bytes(upper,lower)/340) + 36.53;
125         }
126         static double CalcGyro(byte upper, byte lower){
127             return ((double)Smash_Two_Bytes(upper, lower) / 131);
128         }
129         static double CalcAccel(byte upper, byte lower){
130             return ((double)Smash_Two_Bytes(upper, lower) / 16384);
131         }
132         static short Smash_Two_Bytes(byte upper, byte lower){
133             return BitConverter.ToInt16(new byte[2] { (byte)upper, (byte)lower },
         0);
134         }
135         private void com_comboBox_Click(object sender, EventArgs e){
136             com_comboBox.Items.Clear();
137             if (SerialPort.GetPortNames().Length == 0){
138                 no_com_label.Visible = true;
139             }else{
140                 no_com_label.Visible = false;
141                 foreach (string port in SerialPort.GetPortNames()){
142                     com_comboBox.Items.Add(port);
143                 }
144             }
145         }
146         private void go_button_Click(object sender, EventArgs e){
147             if (go_button.Text == "GO"){
148                 go_button.Text = "STOP";
149                 //setup port
```

```
150                 UART.PortName = com_comboBox.Items[com_comboBox.SelectedIndex].
    ToString();
151                 UART.BaudRate = int.Parse(Baud_comboBox.Items[Baud_comboBox.
    SelectedIndex].ToString());
152                 UART.Open();
153             }else{
154                 go_button.Text = "GO";
155                 Raw_textBox.Text = "";
156                 UART.Close();
157             }
158         }
159         private void View_buffer(){
160             //Raw data
161             Raw_textBox.Clear();
162             Raw_textBox.Text = "Raw Packet Data:";
163             for (int i = 0; i < packet_size;i++ ){
164                 Raw_textBox.Text += Environment.NewLine;
165                 Raw_textBox.Text += i.ToString() + ": " + buffer[i].ToString();
166             }
167             //Run calculations
168             temp = CalcTemp(buffer[7], buffer[6]);
169             accelX = CalcAccel(buffer[1], buffer[0]);
170             accelY = CalcAccel(buffer[3], buffer[2]);
171             accelZ = CalcAccel(buffer[5], buffer[4]);
172             gyroX = CalcGyro(buffer[9], buffer[8]);
173             gyroY = CalcGyro(buffer[11], buffer[10]);
174             gyroZ = CalcGyro(buffer[13], buffer[12]);
175             //Reconstruct filter floats
176 
177             double NaN_test = 0;;
178             byte[] to_float_accelX = {buffer[17],buffer[16],buffer[15],buffer
    [14]};
179 
180             NaN_test = (double)System.BitConverter.ToSingle(to_float_accelX, 0);
181             if(NaN_test != double.NaN)
182             {
183                 filtered_accelX = NaN_test;
184             }
185             byte[] to_float_accelY = {buffer[21],buffer[20],buffer[19],buffer
    [18]};
186             NaN_test = (double)System.BitConverter.ToSingle(to_float_accelY, 0);
187             if (NaN_test != double.NaN)
188             {
189                 filtered_accelY = NaN_test;
190             }
191             byte[] to_float_accelZ = {buffer[25],buffer[24],buffer[23],buffer
    [22]};
192             NaN_test = (double)System.BitConverter.ToSingle(to_float_accelZ, 0);
193             if (NaN_test != double.NaN)
194             {
195                 filtered_accelZ = NaN_test;
196             }
197             byte[] to_float_gyroX = { buffer[29], buffer[28], buffer[27], buffer
    [26] };
198             NaN_test = (double)System.BitConverter.ToSingle(to_float_gyroX, 0);
199             if (NaN_test != double.NaN)
200             {
201                 filtered_gyroX = NaN_test;
```

```
202             }
203             byte[] to_float_gyroY = { buffer[33], buffer[32], buffer[31], buffer
        [30] };
204             NaN_test = (double)System.BitConverter.ToSingle(to_float_gyroY, 0);
205             if (NaN_test != double.NaN)
206             {
207                 filtered_gyroY = NaN_test;
208             }
209             byte[] to_float_gyroZ = { buffer[37], buffer[36], buffer[35], buffer
        [34] };
210             NaN_test = (double)System.BitConverter.ToSingle(to_float_gyroZ, 0);
211             if (NaN_test != double.NaN)
212             {
213                 filtered_gyroZ = NaN_test;
214             }
215             byte[] to_float_pitch = { buffer[41], buffer[40], buffer[39], buffer
        [38] };
216             NaN_test = (double)System.BitConverter.ToSingle(to_float_pitch, 0);
217             if (NaN_test != double.NaN)
218             {
219                 pitch = NaN_test;
220             }
221             byte[] to_float_roll = { buffer[45], buffer[44], buffer[43], buffer
        [42] };
222             NaN_test = (double)System.BitConverter.ToSingle(to_float_roll, 0);
223             if (NaN_test != double.NaN)
224             {
225                 roll = NaN_test;
226             }
227             byte[] to_float_yaw = { buffer[49], buffer[48], buffer[47], buffer
        [46] };
228             NaN_test = (double)System.BitConverter.ToSingle(to_float_yaw, 0);
229             if (NaN_test != double.NaN)
230             {
231                 yaw = NaN_test;
232             }
233             byte[] to_int_time = { buffer[53], buffer[52], buffer[51], buffer[50]
         };
234             iter_loop = (UInt32)System.BitConverter.ToUInt32(to_int_time, 0);
235 
236             //Data
237             data_textBox.Clear();
238             data_textBox.Text =  "Raw Data" + Environment.NewLine + Environment.
        NewLine;
239             data_textBox.Text += "      Temprature: " + temp.ToString() +
        Environment.NewLine;
240             data_textBox.Text += Environment.NewLine;
241             data_textBox.Text += "Acceleromter X: " + accelX.ToString() +
        Environment.NewLine;
242             data_textBox.Text += "Acceleromter Y: " + accelY.ToString() +
        Environment.NewLine;
243             data_textBox.Text += "Acceleromter Z: " + accelZ.ToString() +
        Environment.NewLine;
244             data_textBox.Text += Environment.NewLine;
245             data_textBox.Text += "             Gyro X: " + gyroX.ToString() +
        Environment.NewLine;
246             data_textBox.Text += "             Gyro Y: " + gyroY.ToString() +
        Environment.NewLine;
```

```
247             data_textBox.Text += "             Gyro Z: " + gyroZ.ToString() +
        Environment.NewLine;
248             data_textBox.Text += Environment.NewLine;
249             data_textBox.Text += "Filtered Data" + Environment.NewLine +
        Environment.NewLine;
250             data_textBox.Text += "Acceleromter X: " + filtered_accelX.ToString()
        + Environment.NewLine;
251             data_textBox.Text += "Acceleromter Y: " + filtered_accelY.ToString()
        + Environment.NewLine;
252             data_textBox.Text += "Acceleromter Z: " + filtered_accelZ.ToString()
        + Environment.NewLine;
253             data_textBox.Text += Environment.NewLine;
254             data_textBox.Text += "             Gyro X: " + filtered_gyroX.
        ToString() + Environment.NewLine;
255             data_textBox.Text += "             Gyro Y: " + filtered_gyroY.
        ToString() + Environment.NewLine;
256             data_textBox.Text += "             Gyro Z: " + filtered_gyroZ.
        ToString() + Environment.NewLine;
257             data_textBox.Text += Environment.NewLine;
258             data_textBox.Text += "Attitude:" + Environment.NewLine + Environment.
        NewLine;
259             data_textBox.Text += "Roll: " + roll.ToString() + Environment.NewLine
        ;
260             data_textBox.Text += "Pitch: " + pitch.ToString() + Environment.
        NewLine;
261             data_textBox.Text += "Yaw: " + yaw.ToString() + Environment.NewLine;
262
263
264         }
265         private void Step_timer_Tick(object sender, EventArgs e){
266             counter++;
267             Accel_chart.ChartAreas[0].AxisX.Maximum = counter;
268             Gyro_chart.ChartAreas[0].AxisX.Maximum = counter;
269             Filtered_Accel_chart.ChartAreas[0].AxisX.Maximum = counter;
270             Filtered_Gyro_chart.ChartAreas[0].AxisX.Maximum = counter;
271             Attitude_chart.ChartAreas[0].AxisX.Maximum = counter;
272             if (counter > 100)
273             {
274                 Accel_chart.ChartAreas[0].AxisX.Minimum = counter - 100;
275                 Gyro_chart.ChartAreas[0].AxisX.Minimum = counter - 100;
276                 Filtered_Accel_chart.ChartAreas[0].AxisX.Minimum = counter - 100;
277                 Filtered_Gyro_chart.ChartAreas[0].AxisX.Minimum = counter - 100;
278                 Attitude_chart.ChartAreas[0].AxisX.Minimum = counter - 100;
279             }
280             if (UART.IsOpen){
281                 //-----------------------------------------Request and
        recieve data
282                 buffer[0] = 0xAA;//Don't send zero
283                 UART.ReadExisting();//Get rid of any left over data in buffer
284                 while (UART.BytesToRead == 0){
285                     UART.Write(buffer, 0, 1);//Keep requesting a packet
286                 }
287                 UInt16 fail = 0;
288                 while ((UART.BytesToRead < packet_size) & (fail < 10000)){//Wait
        for packet to build up
289                     fail++;
290                 }
291                 if (fail != 10000)
```

```
292                 {
293                     if (UART.BytesToRead == packet_size)
294                     {
295                         UART.Read(buffer, 0, Convert.ToInt32(packet_size));//Is
    the packet out of sync, don't lsiten
296                         View_buffer();
297                     }
298                     UART.Write(command,0,4);
299                     try
300                     {
301                         if (((Decimal)accelX < Decimal.MaxValue) & ((Decimal)
    accelX > Decimal.MinValue))
302                         {
303                             Accel_chart.Series["X"].Points.AddXY(counter, accelX)
    ;
304                         }
305                         if (((Decimal)accelY < Decimal.MaxValue) & ((Decimal)
    accelY > Decimal.MinValue))
306                         {
307                             Accel_chart.Series["Y"].Points.AddXY(counter, accelY)
    ;
308                         }
309                         if (((Decimal)accelZ < Decimal.MaxValue) & ((Decimal)
    accelZ > Decimal.MinValue))
310                         {
311                             Accel_chart.Series["Z"].Points.AddXY(counter, accelZ)
    ;
312                         }
313                         if (((Decimal)gyroX < Decimal.MaxValue) & ((Decimal)gyroX
     > Decimal.MinValue))
314                         {
315                             Gyro_chart.Series["X"].Points.AddXY(counter, gyroX);
316                         }
317                         if (((Decimal)gyroY < Decimal.MaxValue) & ((Decimal)gyroY
     > Decimal.MinValue))
318                         {
319                             Gyro_chart.Series["Y"].Points.AddXY(counter, gyroY);
320                         }
321                         if (((Decimal)gyroZ < Decimal.MaxValue) & ((Decimal)gyroZ
     > Decimal.MinValue))
322                         {
323                             Gyro_chart.Series["Z"].Points.AddXY(counter, gyroZ);
324                         }
325                         if (((Decimal)filtered_accelX < Decimal.MaxValue) & ((
    Decimal)filtered_accelX > Decimal.MinValue))
326                         {
327                             Filtered_Accel_chart.Series["X"].Points.AddXY(counter
    , filtered_accelX);
328                         }
329                         if (((Decimal)filtered_accelY < Decimal.MaxValue) & ((
    Decimal)filtered_accelY > Decimal.MinValue))
330                         {
331                             Filtered_Accel_chart.Series["Y"].Points.AddXY(counter
    , filtered_accelY);
332                         }
333                         if (((Decimal)filtered_accelZ < Decimal.MaxValue) & ((
    Decimal)filtered_accelZ > Decimal.MinValue))
334                         {
```

```
335                             Filtered_Accel_chart.Series["Z"].Points.AddXY(counter
    , filtered_accelZ);
336                         }
337                         if (((Decimal)filtered_gyroX < Decimal.MaxValue) & ((
    Decimal)filtered_gyroX > Decimal.MinValue))
338                         {
339                             Filtered_Gyro_chart.Series["X"].Points.AddXY(counter,
     filtered_gyroX);
340                         }
341                         if (((Decimal)filtered_gyroY < Decimal.MaxValue) & ((
    Decimal)filtered_gyroY > Decimal.MinValue))
342                         {
343                             Filtered_Gyro_chart.Series["Y"].Points.AddXY(counter,
     filtered_gyroY);
344                         }
345                         if (((Decimal)filtered_gyroZ < Decimal.MaxValue) & ((
    Decimal)filtered_gyroZ > Decimal.MinValue))
346                         {
347                             Filtered_Gyro_chart.Series["Z"].Points.AddXY(counter,
     filtered_gyroZ);
348                         }
349                         if (((Decimal)roll < Decimal.MaxValue) & ((Decimal)roll >
     Decimal.MinValue))
350                         {
351                             Attitude_chart.Series["Roll"].Points.AddXY(counter,
    roll);
352                         }
353                         if (((Decimal)pitch < Decimal.MaxValue) & ((Decimal)pitch
     > Decimal.MinValue))
354                         {
355                             Attitude_chart.Series["Pitch"].Points.AddXY(counter,
    pitch);
356                         }
357                         if (((Decimal)yaw < Decimal.MaxValue) & ((Decimal)yaw >
    Decimal.MinValue))
358                         {
359                             Attitude_chart.Series["Yaw"].Points.AddXY(counter,
    yaw);
360                         }
361                     }
362                     catch { }
363 
364                     Motor_chart.Series["motors"].Points.Clear();
365                     Motor_chart.Series["motors"].Points.AddXY("NW",(byte)buffer
    [54]*10);
366                     Motor_chart.Series["motors"].Points.AddXY("SE",(byte)buffer
    [55]*10);
367                     Motor_chart.Series["motors"].Points.AddXY("SW",(byte)buffer
    [56]*10);
368                     Motor_chart.Series["motors"].Points.AddXY("NE",(byte)buffer
    [57]*10);
369 
370                     if((temp).ToString().Length > 4){
371                         temp_label.Text = "Temprature: " + (temp).ToString().
    Substring(0,4);
372                     }else{
373                         temp_label.Text = "Temprature: " + (temp).ToString();
374                     }
```

```
375                     string battery = ((float)buffer[58] * bat_scale).ToString();
376                     if (battery.Length > 5){
377                         bat_label.Text = "Battery: " + battery.Substring(0,5) + "
    V";
378                     }else{
379                         bat_label.Text = "Battery: " + battery + "V";
380                     }
381                     iter_label.Text = "Control loop iteration: " + iter_loop.
    ToString();
382                 }
383             }
384         }
385 
386         private void NW_trackBar_Scroll(object sender, EventArgs e)
387         {
388             command[0] = (byte)NW_trackBar.Value;
389         }
390 
391         private void NE_trackBar_Scroll(object sender, EventArgs e)
392         {
393             command[1] = (byte)NW_trackBar.Value;
394         }
395 
396         private void SW_trackBar_Scroll(object sender, EventArgs e)
397         {
398             command[2] = (byte)NW_trackBar.Value;
399         }
400 
401         private void SE_trackBar_Scroll(object sender, EventArgs e)
402         {
403             command[3] = (byte)NW_trackBar.Value;
404         }
405     }
406 }
```

Listing K.1: Main form from base station software