Dissertation for Masters of Science in Engineering

Design of a Kite Controller

for Airborne Wind Energy

Matteo Milandri

February 2015

University of Cape Town
Faculty Engineering and the Built Environment
Electrical Engineering Department

Supervised by

Samuel Ginsberg

Dr. Ian de Vries

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own.

Matteo F. Milandri

Abstract

Airborne wind energy is a field of technology being developed to make use of the vast, renewable wind power resource which is above the reach of traditional wind turbines, without the need for a large tower. Much analytical research has been undertaken in recent years to better understand the problem space. However, there are relatively few working systems that demonstrate their functioning and can be compared with simulations and theory.

Off-grid power systems still rely heavily on diesel generators, so devices that tap renewable energy sources with similar ease of deployment and lower cost of energy would help this sector to reduce its reliance on expensive, polluting, fossil fuels. The development of these systems is often performed by teams with business interests leaving little open access content available regarding the design process of such devices or the data that they provide.

A kite control pod has been designed for the remote control of a standard kitesurfing kite and a prototype has been demonstrated stably flying such a kite on a fixed length tether. This pod and kite would be tethered to a winch and as the kite flies across the wind, the lift force generated is applied to the winch which is reeled out and electrical power generated. Once fully extended, the tether would be reeled in with the kite de-powered, using some of the generated energy, stored in a battery. This system can then be used as a test bed for the further development of a compact, autonomous, airborne wind energy system for off-grid applications.

Contents

Abstract ii

1. Introduction 1
- 1.1. Airborne wind energy . . . 1
- 1.2. Scope of study . . . 2
  - 1.2.1. Limitations . . . 2
  - 1.2.2. Report overview . . . 3
- 1.3. User requirements . . . 3
  - 1.3.1. Functional description . . . 4
- 1.4. Contextual considerations for AWE projects . . . 5
  - 1.4.1. Energy . . . 5
  - 1.4.2. Benefits of airborne wind energy . . . 5
  - 1.4.3. Deployment locations and flight duration . . . 6
  - 1.4.4. Regulatory considerations . . . 6
  - 1.4.5. Safety . . . 7
- 1.5. Terminology . . . 7

2. Literature review 9
- 2.1. A brief history of kites . . . 9
- 2.2. Physics of wind power . . . 9
  - 2.2.1. Apparent wind . . . 10
  - 2.2.2. Cross-wind flight . . . 11
  - 2.2.3. How electrical power is generated . . . 11
  - 2.2.4. Scaling to larger systems . . . 12
- 2.3. Meteorology of wind power . . . 12
  - 2.3.1. Stratification of wind . . . 13
  - 2.3.2. Wind power and availability . . . 13
  - 2.3.3. Saturation wind power or total power available . . . 14
- 2.4. Active airborne wind energy projects . . . 14
- 2.5. Challenges to commercialisation . . . 18
  - 2.5.1. Technical challenges . . . 18
  - 2.5.2. Regulatory approval . . . 19
  - 2.5.3. Demonstrable safety . . . 19
  - 2.5.4. Funding requirements . . . 19
  - 2.5.5. Competition from other energy sources . . . 19

2.6. Kite control mechanisms/systems as found in literature 20
2.6.1. Categorisation of AWE systems 20
2.6.2. Launching and landing 22
2.6.3. Sensors 23
2.6.4. Flight software and control theory 23

3. Specification **25**
3.1. Basic operation 25
3.1.1. Remote controlled actuation 25
3.1.2. Launching / Landing 26
3.1.3. Flight 26
3.2. System features 27
3.2.1. Actuation 27
3.2.2. Traction force transmission 28
3.2.3. Powering the pod 28
3.2.4. Control input 28
3.2.5. Feedback 29
3.2.6. Dimensions and mass 29
3.2.7. Mechanical integration 29
3.2.8. Flight time 30
3.2.9. Safety release mechanism 30
3.2.10. Ground station data aggregation and display 30
3.3. Environmental stresses 31
3.3.1. Wind range 31
3.3.2. Impact 31
3.3.3. Sand and dust ingress 31
3.3.4. Temperature range 31
3.3.5. Water immunity 31
3.3.6. System and subsystem reliability 32

4. Design **33**
4.1. Initial design exploration 33
4.2. Mechanical structure and integration 34
4.2.1. Baseplate strength 34
4.2.2. Packaging and mechanical layout 35
4.2.3. Wire harnesses 35
4.2.4. Enclosure 36
4.2.5. Mass budget 36
4.3. Actuators 37
4.3.1. Motors and winch assemblies 37
4.3.2. Position sensors 38
4.3.3. Motor drives and controllers 39
4.4. Energy storage and management 40
4.5. Embedded system architecture and microcontroller selection 41

4.6. Control input . . . 42
4.7. Telemetry . . . 42
4.7.1. Link budget . . . 42
4.7.2. Kite to ground data . . . 43
4.7.3. Ground to kite data . . . 43
4.8. Sensors . . . 43
4.8.1. GPS . . . 43
4.8.2. Inertial motion unit . . . 44
4.8.3. Barometer and temperature sensors . . . 44
4.9. Electrical integration . . . 44
4.9.1. System coordination . . . 45
4.9.2. Power supply . . . 45
4.9.3. Grounding strategy . . . 45
4.9.4. Sources of electromagnetic interference . . . 45
4.9.5. Sharing the battery power supply . . . 45
4.10. Embedded software design . . . 46
4.10.1. Motor controller software . . . 46
4.10.2. System controller software . . . 46
4.11. Auxiliary systems . . . 47
4.11.1. Energy harvesting . . . 48
4.11.2. Safety release . . . 49
4.12. Ground station . . . 49
4.12.1. Display and logging of controller data . . . 49
4.12.2. Ground based measurements . . . 50

5. Implementation **51**
5.1. Electronic circuit boards . . . 51
5.1.1. Motor drive PCB . . . 51
5.1.2. Inrush limiter and power distribution board . . . 56
5.1.3. System master controller . . . 56
5.1.4. Ground station circuit . . . 59
5.2. Electro-mechanical components . . . 60
5.2.1. Actuators . . . 60
5.2.2. Battery . . . 61
5.2.3. Wind power generator . . . 61
5.2.4. Safety release mechanism . . . 62
5.2.5. Wiring harnesses . . . 62
5.3. Mechanical aspects . . . 62
5.3.1. Base structure and layout . . . 62
5.3.2. Packaging and assembly . . . 63
5.3.3. Outer housing . . . 63
5.3.4. Attachment to the kite and tether . . . 65
5.4. Main controller embedded software . . . 65
5.4.1. Real-time operating system . . . 65

5.4.2. RC controller input interpretation . . . 66
5.4.3. Drive node communication . . . 66
5.4.4. Telemetry . . . 66
5.4.5. Message integrity . . . 67
5.4.6. Sensors . . . 67
5.4.7. GPS . . . 68
5.5. Motor controller embedded software . . . 68
5.5.1. PWM generation for drive switches . . . 69
5.5.2. Interrupt sources . . . 70
5.5.3. Message received . . . 71
5.5.4. Node reply message . . . 71
5.5.5. Feedback control calculation . . . 71
5.6. Ground station software . . . 72
5.6.1. Code functions . . . 72
5.6.2. Data display . . . 73

6. Testing **75**
6.1. Sub-component performance characterization . . . 75
6.1.1. Actuators . . . 75
6.1.2. Motor controllers . . . 77
6.1.3. Battery energy capacity and power delivery . . . 78
6.1.4. Telemetry and control link . . . 79
6.1.5. Safety release effectiveness . . . 79
6.2. Integration tests . . . 80
6.2.1. Battery-controller-motor . . . 80
6.2.2. Control via telemetry and RC controller . . . 81
6.3. Field tests . . . 82
6.3.1. Ideal weather conditions . . . 82
6.3.2. Equipment required . . . 82
6.3.3. Pre-flight checks . . . 82
6.3.4. Procedure . . . 83
6.3.5. Data to be recorded . . . 84
6.4. Results of field tests . . . 84
6.4.1. Test 1 . . . 84
6.4.2. Test 2 . . . 85
6.4.3. Test 3 . . . 87
6.4.4. Test 4 . . . 88
6.4.5. Analysis of data from test 4 . . . 90

7. Conclusion **91**
7.1. Results . . . 91
7.2. Future prospects for AWE . . . 92
7.3. Further work . . . 92

Acknowledgments **93**

Bibliography **94**

Nomenclature **100**

A. Appendix **101**
A.1. Attached media . . . 101
A.2. Datagram Tables . . . 102
A.2.1. Ground to kite message . . . 102
A.2.2. Kite to ground message . . . 103
A.2.3. Controller to node message . . . 105
A.2.4. node to controller message . . . 105
A.3. Java code for the Ground Station . . . 105
A.3.1. Kite ground station class . . . 105
A.3.2. Telemetry class . . . 109
A.3.3. Data dissemination . . . 113
A.3.4. Data types . . . 115
A.3.5. Data logging . . . 118
A.3.6. Ground station GPS receiver . . . 121
A.4. Embedded code . . . 125
A.4.1. System master controller . . . 125
A.4.2. Motor drive node firmware . . . 145

List of Figures

1.1. HAWT compared to AWE (author's own diagram) . . . 2
1.2. Diagram of whole system (author's own diagram) . . . 4
1.3. Terminology of the wind window (author's own diagram) . . . 8

2.1. Global wind power density($\frac{kW}{m^2}$) percentiles [1] . . . 14
2.2. Photograph of Makani Power's wing in flight[2] . . . 15
2.3. Photograph of Altaeros BAT prototype in flight[3] . . . 16
2.4. KitePower's 25m^2 kite and control pod [4] . . . 17
2.5. Ampyx Technology concept [5] . . . 17

3.1. Layout of standard kite bar and lines . . . 30

4.1. Photograph of motors mounted onto a test plate . . . 38
4.2. Photograph of cells selected . . . 41

5.1. Motor drive power electronics schematic . . . 52
5.2. Motor drive microcontroller schematic . . . 54
5.3. PCB Layout . . . 56
5.4. Photograph of drive mounted in position with wires attached . . . 57
5.5. Inrush limiter schematic (author's own diagram) . . . 57
5.6. Photograph of the system control circuit . . . 58
5.7. The motion sensor circuit mounted on foam . . . 60
5.8. Photographs of modifications made to allow for position sensing . . . 61
5.9. Photograph of packaging of the system . . . 64
5.10. Flow diagram of motor node software . . . 69
5.11. Control Loop Diagram . . . 70
5.12. Screen-shot of ground station application . . . 74

6.1. Speed of retraction versus voltage applied . . . 76
6.2. Current required for different retraction forces . . . 77
6.3. Thermal image of motor drive under load . . . 78
6.4. Set-point tracking of the position controller . . . 81
6.5. Prepared test set-up . . . 83
6.6. Photograph of the second test . . . 86
6.7. Photograph of the third test . . . 87
6.8. Photograph of kite and controller in flight during test 4 . . . 88
6.9. Photographs taken during test 4 . . . 89

1. Introduction

Wind energy, whilst growing in importance in the supply of energy, is predominantly captured using horizontal axis wind turbines (HAWT) in the so-called *Danish style* configuration [6]. These typically have three blades and a generator on top of a tower extending up to around 150m built on a firm foundation. The wind field, however, increases in strength and consistency with altitude which has provided the motivation to build taller and taller turbines to exploit this resource. The abundance of high altitude wind energy has, for many years, been inaccessible due to the height and size limitations inherent in most wind energy harvesting technologies [7] these are compared in Figure 1.1. Primarily due to wind drag over land, the wind speed in a given location will be slowest close to land and faster higher up [8, 9, 10]. Conventional wind turbines are limited by the costs involved in building the tall, strong structures to support them. This construction challenge is increased when the turbine is built in the ocean with submerged foundations. Foregoing the tower altogether has been a compelling possibility, made available by structures that are both able to capture mechanical energy from the wind and remain airborne themselves, either by using some of the wind power to do so or by being lighter than air. Over the last decade, a number of technologies have been invented, integrated and adapted in pursuit of realizing methods of harvesting this high altitude wind energy [6].

1.1. Airborne wind energy

In the process of developing a concept to be pursued it is important to explore and present information on the broader context in which this project exists. The field of airborne wind energy or as it was previously called, high altitude wind power (HAWP) is still in its infancy and is defined by Moritz Diehl as follows: "Airborne wind energy regards the generation of usable power by airborne devices"[6]. AWE encompasses research in several disciplines such as renewable energies, aeronautics, robotics and control, meteorology and electrical engineering. Several approaches to wind power extraction have been examined, most exploiting either lift or drag power of a tethered wing. Lift mode power involves the wing lift force being used for traction of either a vehicular load or a winch, paying out the tether. Drag mode power involves using small turbines to introduce extra drag onto the wind capture wing. This power generated by the turbines is then transmitted down a conductive tether to the base station. Both of these approaches harness cross-wind power, where the lift force generated by the wing is amplified by the high speed (usually several times the wind speed) flight across the wind.

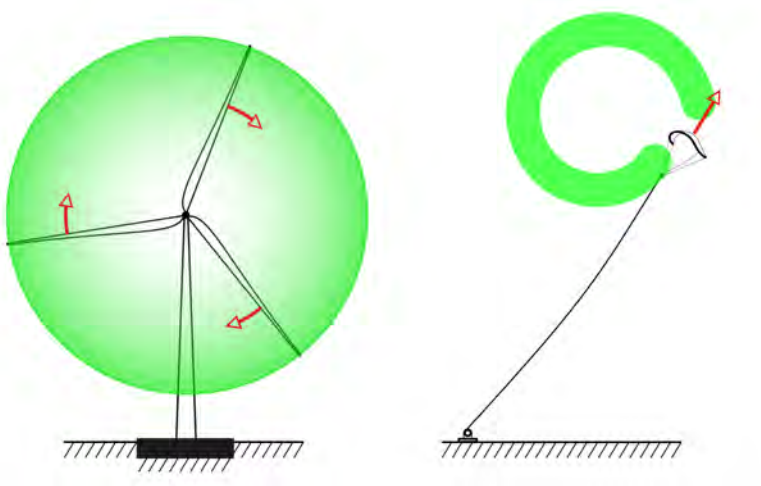


Figure 1.1.: HAWT compared to AWE (author's own diagram)

There are other approaches to harnessing high-altitude wind in an airborne manner such as the use of tethered airships with turbines or autorotating gyrocopters. While these have some niche uses, they fail to take advantage of the benefits of cross-wind motion and thus do not have as great a potential as the concepts described above.

1.2. Scope of study

This masters dissertation undertakes the design and prototype manufacture of some of the equipment required to realize one such high altitude wind power generator. The system in question is designed to interface with a human-operable power kite as the wind-energy capture wing and entails the design of a robotic controller capable of flying the kite while being attached to a tether. The process is documented in the form of this thesis and is presented to record the decisions, trade-offs and attempts made in producing a working prototype. This will answer questions regarding the complexity and feasibility of the implementation of such a system..

The system will be tested to demonstrate function and will be deemed to be functional if the control pod, attached to a kitesurfing kite is able to be launched, flown in a steady state and landed without loss of control or other failure.

1.2.1. Limitations

This study is limited to the design of the kite control robot and the prototyping and testing thereof. In the context of airborne wind energy research, this is a component of a so called pumping mode kite generator and while it is designed to be integrated into such a system, demonstration of this has been deemed beyond the scope of this project.

The study also does not extend into the development of a dynamic model or control law for autonomous flight control of a kite. No integration with a winch or other power generating or storage system is to be investigated. The flight testing of the control pod is to be done with a fixed tether as the winch designed in a complimentary study will not be ready during the duration of this research. The demonstration of the working system will also be limited as only photographs thereof are presented here. Video footage is available on a CD included with the printed version.

1.2.2. Report overview

The documentation includes the following chapters:

1. Introduction, this chapter, a presentation of the broad context of this thesis.
2. An overview of the current state of the art in AWE
3. A system level specification
4. Design documentation and motivations behind major component selection
5. Notes from the implemented design and prototyping process
6. A report of the tests undergone and the performance of the system
7. The report is concluded with an evaluation of the system as built and a discussion on how the system could be extended in future work.

1.3. User requirements

In this dissertation we develop a robotic control pod, used to fly a leading-edge-inflatable kite-surfing kite. It is to be both a prototype of a system that could be commercialized and a test platform for the further development of other aspects of a full AWE system, like what is shown in Figure 1.2. This system is a model in that it is scaled down in some important respects. It is smaller, cheaper, with fewer functions and less robust than if it were to be part of a system that would be used to produce electrical power over an extended period of time. It is to be self-contained in terms of power supply and sensor/actuator placement as opposed to being integrated into a wing structure and is to provide a direct method of remote control of a kitesurfing kite. It is to attach to the standardized bar and loop interface of such a kite to enable remote control of it. This system is to be used to characterize the kite control method and to provide a platform for the comparison of different kite designs. The data gathered from field tests performed using this system are also a useful goal of this project, providing a variety of possible insights into kite dynamics and embedded systems development.

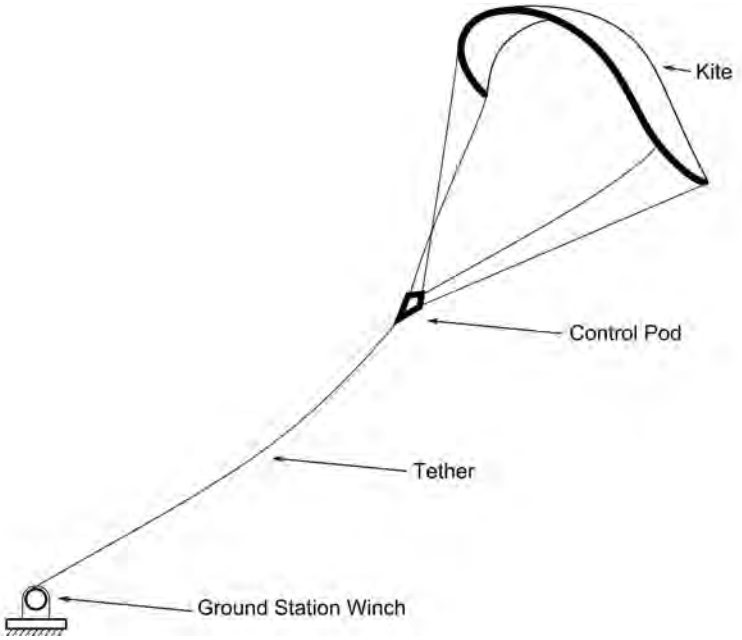


Figure 1.2.: Diagram of whole system (author's own diagram)

1.3.1. Functional description

The kite is to be flown by this device and provide a human controller sufficiently fast response times to steer it in a stable flight path. The kite and controller are attached to the ground station by a tether, to be attached to a winch in the power producing version. As the line gets unreeled from the winch, a drag force is applied to the drum and the excess energy is converted to electrical power to drive a load or charge batteries. The kite is to be flown up away from the winch that is fixed to the ground until the tether is completely unwound. At this point the kite's control lines must be released, reducing the lift on the kite and allowing the winch to pull the kite and tether back in. Once the line is sufficiently wound in (at the lower limit of the cycle) the process is repeated with the kite controller retracting the rear kite lines and thus generating lift once again with the kite.

The controller and the winch system on the ground will need to have a communication channel to signal operating data and changes of state between them. Crucial data will include upper and lower flight profile limits, safety warnings and failure-to-safe countermeasures and data relevant to the optimization of the power generation during the whole generation cycle. This communication channel will also allow an operator on the ground to be able to take control of the kite or to initiate any of the stages of flight. Data regarding kite position and dynamics and wind speed, direction and consistency will also be needed for kite power maximization.

1.4. Contextual considerations for AWE projects

This project fits into the broad context of wind power and aspects of the following fields of study and implementation inform and affect the deployment of viable airborne wind energy systems.

1.4.1. Energy

The world's energy mix is changing at an unprecedented rate [11], partially due to the rising prices of traditionally utilized sources and partially to the greater demand for non-carbon dioxide producing (or polluting) sources of energy. This change provides an opportunity for novel concepts for energy production to enter the market. The prices per kilowatt hour of produced energy for these renewable sources have been showing a strong trend down, towards and in some cases, below the prices of power available on national grids. This is especially relevant where the prices of fossil fuels are continuing a trend of increasing prices (of long-term averages).

Reliance on electrical energy sources that are by nature, intermittent such as photovoltaic solar panels and wind turbines can create problems for electricity grids that are not designed to accommodate them. Thus, renewable energy sources with a higher capacity factor such as hydroelectricity and potentially airborne wind energy will become increasingly important as methods of increasing the contribution of renewable energy sources without major changes to most grids.

1.4.2. Benefits of airborne wind energy

When compared to conventional wind power, AWE provides some important benefits. Firstly, the possibility of much lower material and logistics costs. As shown in [6], large scale AWE systems could have an airborne mass of 15 - 20 times less than just the blades of HAWT systems with equivalent rated power.

Secondly, the operational altitude of airborne systems can be much higher than that of traditional wind turbines, catching faster, more consistent wind and allowing for a higher capacity factor. This is because the wind over the land is less affected by ground based obstacles at higher altitudes. The altitude can also be continuously optimised for maximum continuous power during operation.

Lastly the replacement of the large tower with a tether leads to simpler, cheaper, transport to site and construction. The system would therefore also be able to be completely removed during storms and grounded for maintenance. The generator and gearbox can be situated on the ground in certain configurations and not at the top of the tower as is the case in HAWT systems.

1.4.2.1. Wind Power

Wind power is predominantly captured using triple-bladed horizontal-axis wind turbines and the largest of these are reaching a blade tip height of 220m and a rated capacity of 8.0 MW. While growing in height and capacity over time, these turbines are still unable to reach wind outside of the boundary layer which extends to about 500m altitude [1].

Also, due to the low altitudes of operation, the vast majority of the viable wind sites to HAWT systems are near coasts or offshore, where there are fewer obstacles and the wind is slowed much less due to friction. In contrast to this, higher altitude operation allows AWE systems to be viable, at possibly any location, greatly increasing the applicability of wind power.

1.4.2.2. Off Grid Power

Airborne wind energy use in off-grid and islanded-grid settings shows great potential and could be used to reduce the reliance of these systems on diesel generators. These systems are usually too small to benefit from the cost-effectiveness of large wind turbines and power stations and could benefit from power production systems that can scale to their needs. Several teams are designing AWE systems with an output power around 60kW that can fit into a truck [12].

1.4.3. Deployment locations and flight duration

Airborne wind generators can be used in a much larger variety of sites, especially further inland than would be appropriate for horizontal axis wind turbines due to their greater operating altitude allowing access to more consistent, powerful wind. There is however, concern [13] over their general applicability close to roads, buildings and human activity due to the danger of impact of the wind capture wing or the fast moving high tensile tether in the event of a system failure. This safety in operation is crucial both for site selection and to allow for a number of these systems to be deployed in a given area. Such a system would also be required to operate with little human oversight for long periods of time to be cost-effective as a power producing system. Therefore, autonomous control of the airborne wing is an important part of any viable system. The majority of AWE systems would require some low level wind for the launching of the kite. This is sometimes absent even when there is wind at higher altitudes and must be taken into consideration in the site planning of such a project.

1.4.4. Regulatory considerations

The aerospace/air-traffic regulations regarding tethered obstacles need to be considered with regard to the allowable altitudes and locations of deployment. These obstacles are

usually marked with red spheres with a diameter of at least 60cm according to the Civil Aviation Authority of South Africa [14]. A set of static marker kites has been proposed by EnerKite [15] to ensure that the kite lines are visible to aviators without the need to add drag to the kite tether by adding spherical markers to it. During the testing to be performed as part of this study, an altitude limit of 300m will be imposed to avoid airspace obstruction.

1.4.5. Safety

Any applied technology must be done so in a manner that mitigates risk of injury or damage to individuals and property in the event of a system failure. The solutions under investigation in the field of AWE will create potential risk to low-flying aeroplanes as a tethered obstacle as well as a risk of crashing into person or property on land. The fail-to-safe-state functioning of the components and integrated system must be proven before any such system can be deployed without stringent keep-out zones around them. This requirement of demonstrable safety is more crucial to new or so-called alternative technologies as a precedent has not yet been set as to what is acceptable and regulators will tend to be conservative in their endorsement of new projects. In this project, the test site will be cleared of bystanders and a safety power-release mechanism will be employed during testing to mitigate this risk.

1.5. Terminology

In the discussions that follow, certain terminology pertaining to kite flight in the wind field will be used. This is depicted in Figure 1.3 and explained below.

Wind direction / heading The wind direction is labelled by where it is coming from. i.e. a southerly wind would be moving directly northwards.

Downwind Having the same direction as the wind velocity vector.

Wind window The quarter-sphere, with the ground station as the reference point being its centre, where the kite can naturally fly.

Apparent wind The velocity of the air relative to the kite. This equals the wind velocity vector minus the kite velocity vector (both being relative to ground).

Leading edge and traction lines These lines are attached to the leading edge of the kite wing and transfer the majority of the lift force. They are held at a fixed length as the trailing edge lines are adjusted to control the kite.

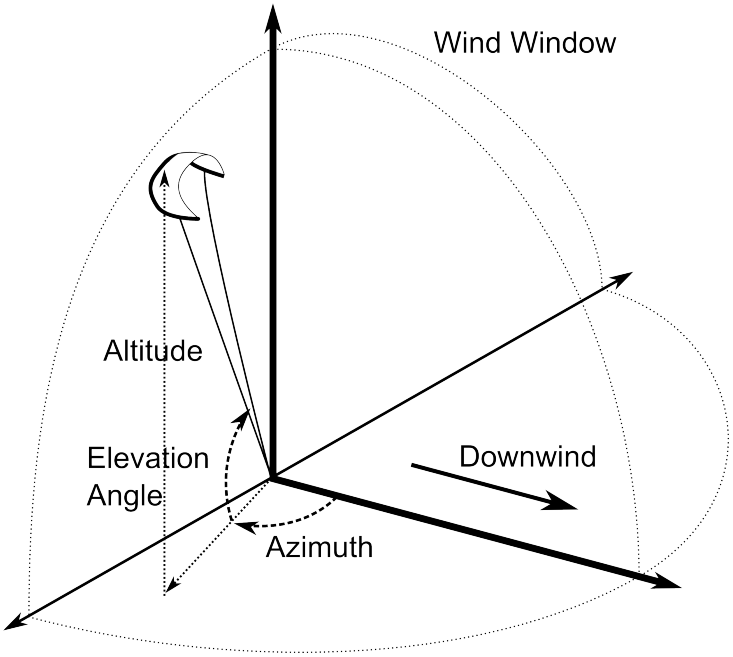


Figure 1.3.: Terminology of the wind window (author's own diagram)

Trailing edge and control lines The lines to the rear of the kite wing that can be adjusted in position to change the angle of attack of each side of the wing independently.

Elevation angle and bearing The elevation angle is the angle between the tether and the ground plane and the azimuth angle is the angle between the bearing of the kite and the downwind direction (positive in clockwise or right-hand direction). These and the tether length provide polar coordinates which give position.

2. Literature review

In order to provide a firm theoretical base for this study, a review of the current trends and academic output pertaining to airborne wind energy capture has been undertaken. As shown below, the field is relatively new even though kites have been used for centuries for other purposes.

2.1. A brief history of kites

Some of the earliest known uses of kites have been traced back to the 4th century in China [16]. They seem to have originated as flying toys but soon were used for airborne reconnaissance over the city of SongCheng. Between 1750 and 1950, kites were the primary platform for airborne meteorology up to altitudes of 5 000 meters and these measurements were crucial in the weather forecasting state of the art of the time [17]. Kites were also used to pull boats and wagons but the control of the kites was difficult, especially in the presence of turbulence. Kites and testing of tethered wings were also instrumental in the development of the aeroplane [the papers of Orville and Wilbur Wright] [17].

In 1976 a patent was granted to Peter Payne and Charles McCutchen for a "self-erecting windmill" [18] and then in 1980 a paper by Myles Loyd is published in the Journal of Energy on "Crosswind Kite Power" [19], now considered to be the start of academic exploration of airborne wind energy, especially in quantifying its potential.

Over the last century, the design of the kites in use has also changed substantially from the box kites and r-kites used for meteorology to the wide variety in use today. Much of this development has been done by kite enthusiasts and more recently, by companies involved in designing equipment for kitesurfing and kite drawn buggies.

2.2. Physics of wind power

Wind is the movement of a body of air, flowing down a pressure gradient due to uneven heating of the atmosphere by the sun. This movement of the air and its associated mass embodies kinetic energy that can be captured using a device to impede the flow. The energy can be output by fixing the apparatus to a point relative to the ground.

The power available from a body of air can be calculated from the kinetic energy KE contained in the airmass with mass m and velocity v:

$$KE = \frac{1}{2}mv^2 \tag{2.1}$$

and the volume V of the air flowing through area A per unit time Δt can be described by:

$$V = v_{wind}A\triangle t \tag{2.2}$$

and therefore the mass m of the air with density ρ per unit time is:

$$m = \rho v_{wind}A\triangle t \tag{2.3}$$

Combining these and given that power $P = \Delta KE/\Delta t$

$$P = \frac{1}{2}\rho A v_{wind}^3 \tag{2.4}$$

The power that can be extracted from the wind is proportional to the swept area but also to the cube of the wind speed therefore a doubling of the average wind speed can lead to an 8 fold increase in energy output.

Not all the kinetic energy can be extracted though (or the air behind the turbine would be stationary and pile up), only up to 16/27 of the total, known as the Betz limit [20].

2.2.1. Apparent wind

The lift force is proportional to the coefficient of lift C_L and the area of the kite but the square of the apparent wind speed v_a. Thus if the kite is moving fast though the air, in figure 8 patterns for example, the lift generated can be significantly higher than that of a stationary wing with only the wind velocity appearing over it.

$$F_{Lift} = \frac{1}{2}\rho A_{wing} C_L v_a^2 \tag{2.5}$$

The effect of crosswind motion is also present in conventional (HAWT) turbines where the majority (more than 50% from the last 30% of the wing) of the power is generated near the tip of the wing [6]. The tip speeds of these turbines can reach 320m/s or 94% of the speed of sound.

2.2.2. Cross-wind flight

The apparent wind on a kite can be increased by flying in a cross-wind path and this has been shown by Myles Loyd [19] to produce a factor of up to $\frac{1}{2}\left(\frac{C_L}{C_D}\right)^2$ (between 10 and 100 times) more power output than 'static' kites pulling against a winch. Where C_D is the coefficient of drag of the wing. This is partly from the effect of covering a large swept area during the cycle (disrupting a certain volume of the wind field). The higher the speed of the wing, the greater the lift force and the greater the drag force. Thus the ratio of the coefficient of lift C_L over the coefficient of drag C_D is an indicator of the effectiveness of a particular wing in generating wind power.

Wind power equation gives a limit to the maximum power available from a wind field for a given wind speed and wing geometry:

$$P < \frac{2}{27}\rho A_{wing} v_w^3 C_L \left(\frac{C_L}{C_D}\right)^2 \tag{2.6}$$

Equation 2.4 gives a figure for the amount of power in a certain cross-sectional area of the wind where Equation 2.6 defines a relationship between wing performance and output power.

2.2.3. How electrical power is generated

In a HAWT, power is generated by imposing an extra drag force on the movement of the blades. This drag counteracts the lift force driving the blades and is taken off by applying a torque, slowing the blades rotation. The movement in the opposite direction to the applied force provides an output power through the gearbox and generator to the grid. Airborne wind turbines, in the absence of a fixed tower and nacelle, must generate power in other ways.

2.2.3.1. Ground based generation and traction systems

Two main main methods are used for power generation, the first is to use the lift force generated by the airborne wing, transmitted as a tension in the tether to unwind a winch attached to a generator or to pull a vehicle (such as a boat or rail car on a track). The generator and winch can then be on the ground rather than at the top of a tower. The operation of this traction based system consists of two phases: the traction and retraction phases. These would be started once the kite reaches a suitable altitude. The traction phase where the kite lift is increased and the tether is unreeled from the winch. The traction force times the speed of payout is the mechanical energy that can be tapped by a generator. The lift produced by the kite is increased by flying circles or figure 8 patterns across the wind. Once the tether is fully extended, a short retraction phase is

required to bring the tether back in. The kite is usually flown to as close to vertical as possible and de-powered fully while the tether is retracted to reduce the force required. As there is up to 20% of the cycle time that consumes power, several generating systems in the same farm or energy storage is required to enable consistent power output.

2.2.3.2. Airborne generators and induced drag

The other method of power generation is to use small turbines attached to the flying wing to introduce a drag that harvests power from the flight of the wing. This power is then transmitted down a conducting tether to the ground-station. During operation, the tether length is static and the kite is flown in loops across the wind. This method is typically referred to as drag mode. Other methods such as SkyWindPower's flying electric generators are using essentially the same overall concept but their position remains relatively static.

2.2.4. Scaling to larger systems

The objective of most of the commercial endeavours to design AWE systems is to scale their demonstration units up to a size that would produce significant energy for the electrical grid. There is great theoretical potential for scaling up AWE systems to output power greater than the largest conventional wind turbines, even up to an estimated 40MW for a single wing [19]. At these power levels, the tethers would be about 12cm in diameter with a kite wingspan of about 100m. The kite mass would become a limiting factor as the wing area increases with the square of the span (for a certain aspect-ratio) but the mass increases cubically with an increase in wing span, leading to a diminishing return for larger systems [6].

2.3. Meteorology of wind power

Wind is caused by uneven heating of the atmosphere by the sun. When heated, the air expands, rises and cooler air flows in to fill the space below it. The direction of movement is changed by the centripetal acceleration caused by the rotation of the earth, known as the Coriolis force. The available wind resource varies widely by location and surface topology. As the energy extractable from the wind is proportional to the cube of the wind speed, knowing where and when the wind blows is important to any effort to extract power from it [21].

A brief summary of the meteorology of wind is given here to show the vast untapped potential available above the operating altitudes of current wind turbine technology.

2.3.1. Stratification of wind

Near the ground, friction from obstacles and terrain cause the wind to become turbulent and slow down. This effect typically extends up to an altitude of about 500m above the ground and makes up the so-called boundary layer of the atmosphere [22]. With an increase in altitude, the wind is less disrupted so the wind speed increases to a maximum at about 12km. Above this altitude, the air pressure has dropped to low enough to make it difficult to extract meaningful energy from the wind.

There are two related approaches for determining the average wind in the boundary layer, above a reference point. The:

- Log wind profile (in neutral stability conditions): $u_z = \frac{u_*}{K} \ln\left(\frac{z-d}{z_0}\right)$ where the expected wind velocity u_z at an altitude of z meters is related to u_* which is the shear velocity. $K = 0.41$ and z_0 is the surface roughness (in meters) (0.1 for open fields). d is the height of the 'zero plane' (usually about 2/3 of the height of the tallest obstacles) [21]
- wind profile power law $v_x = v_r \left(\frac{z_x}{z_r}\right)^{\alpha}$ is used for estimating average wind velocity v_x at an altitude z_x relative to the known average wind velocity v_r at another (typically lower) altitude z_x. The exponent α is empirically derived and equals about 0.143 over land and 0.11 over open water [20].

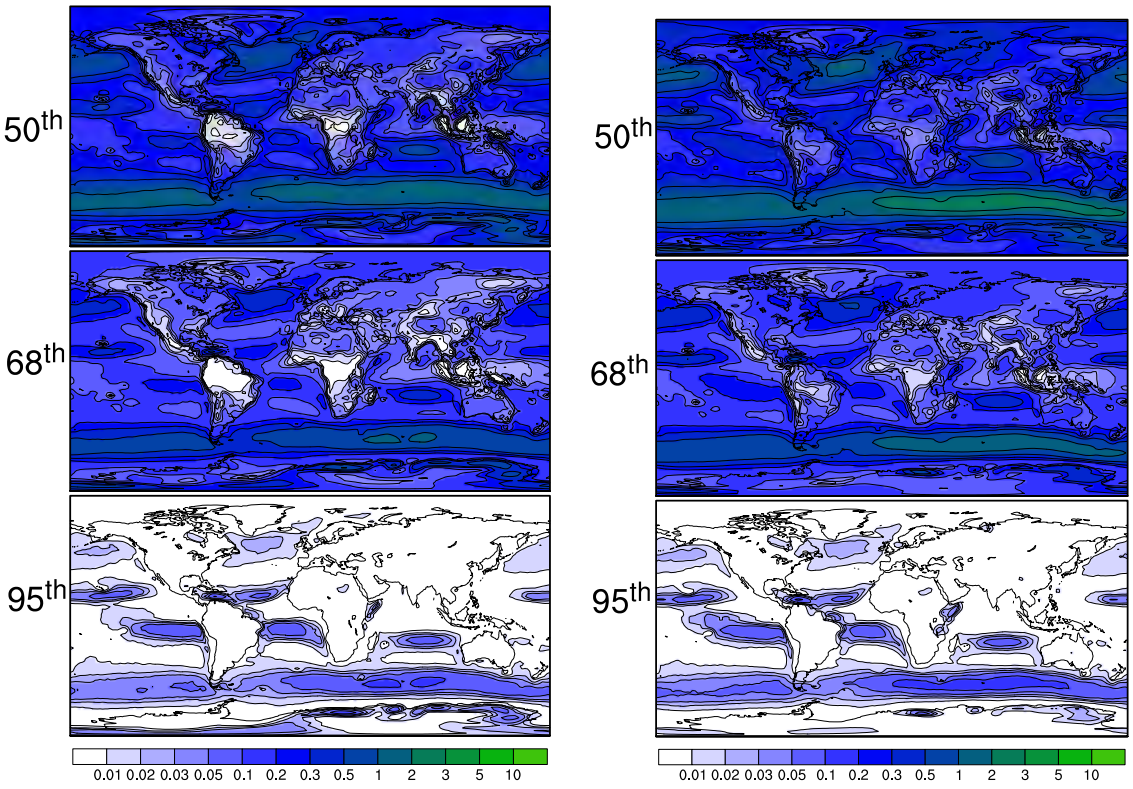
The dynamics of this boundary layer are interesting as we want to access winds at an altitude of between 300 and 500m

This wind speed profile is due to the drag effect of obstacles on the land. Below altitudes of about 500 meters, the wind in the so called boundary layer is affected by drag and turbulence leading to lower velocities [22]. Due to this, most favourable wind sites are near or in the ocean. As the wind above 300 meters and especially above 1km altitude is less affected by the resistance present near the land, it is much more consistent and wind speed usually increases with increasing distance from stationary obstacles.

The charts shown in Figure 2.1 are a reanalysis of radiosonde data from 1979 to 2009. They provide estimates of wind power density at different altitudes for at least 50, 68 and 95 percent of the time, presented by Caldera and Archer [1]. Areas over the land show higher average wind power densities at altitudes of 500m than at 80m. The difference between wind over the land and ocean is much less pronounced at these elevated altitudes.

2.3.2. Wind power and availability

Part of the reason for a higher average wind power at the upper edge of the boundary layer is due to the more consistent movement of unimpeded air. This more consistent wind allows for a much higher capacity factor as a turbine can be used closer to its nominal output level for more of the time. Another reason for higher capacity factor is the operational altitude flexibility available to airborne systems [23].



(a) At 80m altitude **(b)** At 500m altitude

Figure 2.1.: Global wind power density($\frac{kW}{m^2}$) percentiles [1]

2.3.3. Saturation wind power or total power available

In a study done by Mark Jacobson and Christina Archer, it was calculated that the power available in the jet-streams, at 10km altitude contains approximately 380 TW of available power [24]. This amounts to 30 times more power than civilisation uses at the moment, showing that with the right developments, wind power alone could drive humanity's energy needs for the foreseeable future. This study only estimates the power available at 100m and at 10km altitudes but also notes that more power would be available by spreading the turbines over different altitudes.

2.4. Active airborne wind energy projects

Much progress has been made in the field of AWE since the 1980's, most of it centred around universities or special interest research projects. There are, however, no commercially viable systems available yet. These systems are outlined below and summarised in Table 2.1.

In order to contextualise this project, the other active projects in the field of AWE are discussed including the benefits and potential pitfalls of their chosen solutions.

KiteNRG The Italian team have developed a 60kW mobile prototype that they are testing with a two-line ram-air kite [25]. They have optimized the flight path of a kite used to pull a boat in any direction relative to the wind [26]. They have also contributed actively to the development of control methods and optimization of the generating capacity of kites [27, 28]

Makani Power Is using drag mode power generation and have a system capable of automatic launching, landing and flight. Flight testing is continuing but it seems that there are certain regulatory issues that are holding up more long term tests [2]. The current version of the Makani power plane has four turbines, mounted on the wing shown in Figure 2.2. These are used to introduce a drag to the craft as it flies in circles across the wind. The turbines can also be used as propellers for powered ascent and descent. The tether is power conducting and transmits the generated power to the ground.

Figure 2.2.: Photograph of Makani Power's wing in flight[2]

In 2011 Makani power merged with Joby energy [29], another company investigating a similar concept.

SkySails Their traction power system has primarily been designed to tow ships. They have a prototype 200m^2 ram-air kite that is being tested in the field. They have presented conceptual details of an electricity generating version at AWEC 2013 and showed off their kite in the field. They have a 55kW functional demonstrator model including automated launch and recovery systems and an automated flight control system [30].

Sky Windpower Have designed an auto-rotating gyrocoptor with 4 turbines attached to a tether. They have demonstrated autonomous takeoff, power generation and landing.

They are also touting benefits of such a device for surveillance etc. No progress visible in the last year [31].

Altaeros Energies Developing a tethered, lighter-than-air wind turbine as depicted in Figure 2.3. This has been deployed at a site in Alaska for testing in harsh and remote environments. Their systems are designed to be used where normal wind turbines would be impractical due to the logistics associated with them [3]. They do not seem to make use of crosswind power generation, instead opting for a structure that accelerates the air past the turbine in the centre.

Figure 2.3.: Photograph of Altaeros BAT prototype in flight[3]

Enerkite Started in 2010, demonstrated a working, 30kW system at the AWEC conference in Berlin in September 2013. Currently testing it over longer flight times with an autonomous control and launch and recovery mechanism and are using 3-line control. They seem to be actively developing their system with videos of testing in December 2014 [12].

Delft University of Technology (Laddermill) KitePower This team, after being started by the late Wubbo Ockkels, are using a single line pumping mode system with a leading-edge inflatable kite and a flying control pod. Now lead by Roland Schmehl, they have made notable contributions to the computational modelling of flexible wing kites and are have a 20kW technology demonstrator system [32] shown in Figure 2.4.

Ampyx Power Are using fixed wing gliders that are very aerodynamically efficient in a pumping mode power generation with a single tether. The control of the wing is done with ailerons on the tail of the plane as shown in Figure 2.5.

Twingtec Are developing a new form of inflatable glider using a hybrid solid-inflatable technology based on a tensile structure called tensarity [8]. This they are using in a 50kW pumping mode kite power generator [33] using a dual-line control system.

Figure 2.4.: KitePower's 25m^2 kite and control pod [4]

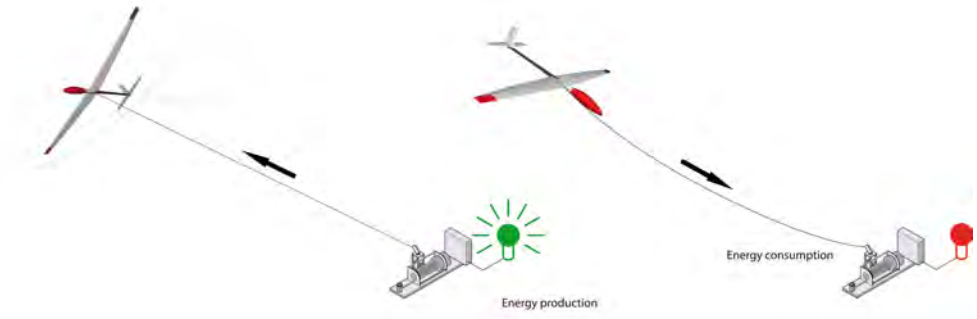


Figure 2.5.: Ampyx Technology concept [5]

Omnidea are designing a traction mode system using the Magnus effect to generate lift with a rotating, helium filled cylinder [34]. This system does not take advantage of crosswind flight and requires a conducting tether to power the rotation of the cylinder. This conductor must be include two connections through the rotating winch of the ground station.

Others There are about 50 or so groups globally working on airborne wind energy in its various forms [6] with many investors behind them so this application of technology is going to remain interesting and become more competitive for some time.

Group Name	Wing Type	Generator Type	Tether Type
KiteNRG	ram-air kite	ground based winch	dual tether, variable length
Makani Power	composite fixed wing	on-wing generators	conductive, fixed length
Skysails	ram-air kite and control pod	traction	conductive, fixed length
Sky WindPower	auto-rotating gyrocopter	airborne generators	conductive, fixed length
Altaeros Energies	Helium filled, ducted turbine	airborne generator	conductive, fixed length
Enerkite	ram-air kite	ground based winch	dual tether, variable length
Kite Power	leading edge inflatable kite	ground based winch	variable length
Ampyx Power	fixed wing glider	ground based winch	variable length
Twingtec	hybrid fixed and inflatable glider	ground based winch	variable length
Omnidea	helium filled cylinder	ground based winch	dual, conductive, variable length

Table 2.1.: Comparison of AWE approaches

2.5. Challenges to commercialisation

Despite the quantity of concepts and projects in the AWE sector, there are still significant obstacles to the commercialisation of AWE systems [35].

2.5.1. Technical challenges

There are several major, unresolved technical risks that must be dealt with by any project team attempting to make a viable product using AWE. Some of these have addressed by the teams above but the lack of test systems running continuously for times greater than several days at most shows that much more is to be done.

- Ensuring robustness to extreme gusts of wind
- Automatic launching and landing (even in high wind) [36] for maintenance and to avoid forecasted weather events
- Position control to allow for many of these systems to be operated near each other without collisions [37]
- Lightening protection (although meteorology kites have demonstrated this ability) possibly meeting the ISO 61400-24 standard for lightening protection but with special focus on avoiding burning the tether.
- Flight control failure (of either avionics or control surfaces) and subsequent fall of wing and tether [15].
- Accurate predictions of tether life for pre-emptive replacement.
- Optimisation of flight path for power output, especially in stratified and turbulent wind [38].
- Reliability and lifespan of the materials used. Resistance to wear and tear from sun, wind and water [39].

- Cost price of the final product: new high-tech equipment is to developed and commercialised at a competitive price.

2.5.2. Regulatory approval

The wings and tethers of AWE systems reach into the airspace during normal operation and so consideration must be given to aviation regulations regarding tethered obstacles.

A precedent has also not been set for whether AWE systems should be permitted to operate near roads or occupied buildings and would need to be ratified in the future. Currently, each organisation developing an AWE system has to acquire permission from the local authorities for each time they test their system [12], further slowing and complicating the development task [40]. Bodies such as the AWEC seek to form a unified representative for the airborne wind energy community [41].

Each system that has been developed will be required to pass some sort of standard test and have airspace markers similar to other aviation obstacles [14]. Marking the kite or possibly the wind site may be necessary. Some teams have proposed using stationary marker kites.

2.5.3. Demonstrable safety

As part of gaining regulatory and public approval, any project that is working towards field trials and longer term testing needs to demonstrate a degree of fault tolerance and fail-to-safe behaviour. Makani wants to have a system working for more than a year before releasing the product to market [35]. This will probably form part of the regulatory process for bringing a new device to market. It will be tested far from flight-paths and people until it has proven robust and safe behaviour in even the most extreme weather conditions.

2.5.4. Funding requirements

As this technology is still considered relatively high-risk due to the lack of viable implemented concepts. This means that the technology is reaching the so-called valley of death [35], where risk is still too high but capital requirements are climbing and are growing too large for angel investors or venture capital. This is often a stage in a new technology where the multiplicity of the options is reduced and only a few technological approaches survive.

2.5.5. Competition from other energy sources

In order for a new technology to grow and become adopted, it must provide benefits when compared to the other available options. The various sources of power, both on

and off the grid are becoming ever more closely competed. As the price of photovoltaic power has fallen dramatically in the last decade, and is starting to undercut the price of traditional sources such as coal and nuclear. Solar power can also be deployed in nearly every part of the globe. The levelized cost of energy for conventional wind power has continued to fall as larger turbines are built with new materials. Near the end of 2014, the price of oil dropped to pre-2006 levels and has stabilized there through the northern hemisphere winter. This puts pressure on renewable investments but overall the price shocks provide an incentive for the transition towards renewable energy sources that do not have fluctuating fuel costs. New natural gas is coming online at a relatively low price in the United States because of recent developments in hydraulic rock fracturing technology which has enabled access to large shale gas reserves and is part of the reason for the lower oil price. These changes force those developing AWE technologies to demonstrate significant advantages in their methods, to get interest and investment.

2.6. Kite control mechanisms/systems as found in literature

To aid in deciding between various design options, discussed in later chapters, an overview of methods currently employed is given here.

2.6.1. Categorisation of AWE systems

Several methods of control of the kite have been proposed and some used in technology demonstrators in the 5 to 80kW power range.

2.6.1.1. Two line, ground based control.

A two-line kite can be steered by changing the lengths of the lines relative to each other but the angle of attack cannot be controlled actively but rather, depends on the position of the kite in the wind window and its velocity. A system using such a kite typically uses two winches, driven independently or connected by a gearing. This allows the lines to be reeled out or in while being steered continuously. When one of the lines is shortened, the drag on that side of the wing is increased, slowing that side and turning the kite towards it. The lack of control of the kite's angle of attack reduces the controller's ability to depower the kite during the retraction phase of a traction mode power generation system.

The original laddermill concept [42] used this method but stacked many kites into a ladder-like formation.

2.6.1.2. Three line, ground based control.

In this method of control, the two lines as in the previous case are connected to the two sides of a flexible kite and a third line is connected through a bridle to the leading edge of the kite. In this configuration, the central line accepts the majority of the lift tension exerted by the kite. The two side lines now act merely as control lines. All three lines can be unreeled but the control is effected by the difference in length between the central line and the line on each side. This has the advantage that the angle of attack and thus, lift on the kite for any given wind speed can be controlled, lengthening both of the side lines would reduce the kite's angle of attack and in short, reduce its lift. A draw-back of this method is the additional drag caused by the presence of three lines as opposed to just one. EnerKite uses this method in their 30kW demonstrator [12].

2.6.1.3. Solid wing with airborne generator

This approach is to use a rigid wing with small, high-speed turbines on it, anchored to the ground with a static length, power conducting tether. The turbines induce a drag, optimized for power production on the typically circular flight of the wing in the wind field. This power is transmitted down the tether at a high voltage to reduce resistance losses over the length which can be up to 1km. Using a solid wing allows for much higher lift/drag ratios than with a fabric wing [43] which, according to Equation 2.6, allows for a greater generation capability for a wing of the same area. The drawbacks to this approach are mainly the extra complexity of the conducting tether and the extra weight added to the wing by the turbines and generators, it is however, a very promising method.

Makani power make use of this approach [2] and have a demonstrator that can autonomously launch and land using the turbines in a quadcopter vertical take-off and landing configuration by supplying the wing with power through the tether.

2.6.1.4. Single line tether with control pod.

The challenges created by bringing all three of the control lines to the ground such as drag and control line droop and stretch can be partially mitigated by using a kite control pod, attached to the tether, close to the kite. The control lines are then brought together at this point and control is remotely actuated from there. This pod is then connected to a single traction tether and winch. The control commands are then sent to the control pod via a data cable in the tether or over a radio link. This pod does however add mass to a point just below the kite and so its inertia can cause stability issues when the kite changes direction rapidly.

Skysails uses this method [30] but only has differential control of the trailing edge of their kites and so do not adjust the angle of attack of their kites. This is possibly because they do not need to have a repeating reel-in phase which requires low lift as the traction

force is being used to pull ships. The KitePower team also makes use of a pod but can individually control the two tailing edge lines in relation to the leading edge traction lines and so can both steer the kite and adjust its angle of attack [7].

2.6.1.5. Rigid wings with controllable ailerons.

In order to increase the L/D ratio of the flying wing, some teams, most notably Ampyx power [5], have opted for using solid glider-like tethered wings in a pumping mode system. These wings weigh more than fabric kites but are more robust and can generate more power for the same wing area. They usually have tails, much like gliders with remote controlled ailerons for flight control. These systems also have the advantage of being able to completely remove their lift during the retraction phase and be reeled in very fast at low power. This is very important for consistent power output in a pumping mode generator. TwingTec is building a hybrid soft/hard wing to make use of both the benefits of fixed wings and the lightweight nature of fabric wings[8].

2.6.1.6. Single line tether with movable attachment points on kite edges.

As described and characterized in [44], an arrangement where the control points of a two line kite can be remotely controlled and moved closer to the leading or trailing edge of the kite independently provides another method of kite control. In the demonstrated configuration, the two lines were then joined 10-20m from the kite and the rest of the tether was a single line, wound around a winch, used for power generation. This method has different dynamics but very similar overall control methodology to the dual line arrangement but does have a method of de-powering the kite as both sides' lines can be moved towards the leading edge of the kite at the same time. It does however have the challenges of mounting the controlled slider mechanisms to the sides of the kite.

2.6.2. Launching and landing

The autonomous, semi-supervised launching and landing of these airborne systems is an important part of their operation as they are usually too large to be handled by humans and may be required to land suddenly due to partial failure or indecent weather. There are three main approaches [45]:

Rotary launching, downwind landing This method of launching uses a boom, rotating in the horizontal plane to pick up the kite or wing and then swing it around till it has sufficient velocity to fly up into the wind field as the tether is extended.

Mast based Skysails has pioneered this method using a telescoping mast that allows the ram-air kite to fill and the control pod to take over before release and tether extension. On landing, the tether is reeled all the way in and the leading edge of the kite is attached to the end of the mast. The kite is then collapsed and the tower retracted into a storage enclosure.

Powered vertical take-off and landing Systems such as Makani power's wing and Sky Wind Power's auto-rotating gyrocopter can have power supplied through their conductive tethers to allow for powered flight from the ground station and up into the wind stream [2]. From there, they would transition into wind-borne flight and begin producing power. The reverse can be done for landing.

2.6.3. Sensors

The feedback required to build a autonomous flight controller comes from several sources, some airborne, some in the ground station. Data from these sensors is fused to form a model of the kite kinematics and then fed into the controller. The airborne sensors include inertial measurement such as acceleration, rotation and orientation, position sensing from a GPS module and environmental sensing such as apparent wind velocity and air pressures on the upper and lower sides of the wing [46].

The sensing at the ground station determines the tension on the tether and the bearing and elevation angle of the kite relative to the tether. The software in the ground station is also present to display some of the flight details to the operators and is recorded for analysis.

2.6.4. Flight software and control theory

The autonomous control of a kite in a power generating system is beyond the scope of this project but as the system is being developed with further work possible in this direction, the design can benefit from an understanding of the needs and constraints of the control system.

Much work has been done in the past few years on modelling kite dynamics and the development of control laws and algorithms with the following in mind:

- Firstly, stable flight, following a set of way-points [38]
- Active optimisation of the flight path for maximum power generation over the full cycle [47, 48].
- Robust response in the event of gusts and turbulence
- Autonomous launching and landing sequences [49, 45].

The sensor data discussed above is brought together and the processing of the control is usually done by a ground station computer so using a low-latency telemetry link is important for pod and wing based actuation systems.

Model predictive control is used by many of the teams surveyed [50, 51, 52] with some suggesting a 'direct' method [25] which records human operator's control input and the resultant kite flight and uses a neural network to automatically determine a suitable control law. This method obviously requires many hours of training data.

3. Specification

The information presented in the review of the prior art and project planning is used to define the features to be included in the product specification. This chapter presents the requirements of the functional elements of the kite control unit (KCU) as a whole. The following design chapter then determines the appropriate components and architectural decisions to achieve this. The specification is used as a baseline to which the prototype can be compared to determine the degree to which it fulfils these requirements. The performance of any one of the characteristics of this device can vary substantially from component selection through to system integration. The pre-commissioning and field testing will also refer to this specification in the preparation and performance of the acceptance test procedure.

3.1. Basic operation

The kite control unit to be designed below is a remote-controlled pod, used to fly a kitesurfing kite. It is to replace the control input to the kite usually performed by the human operator and thus must attach to the standard bar-and-loop interface of such a kite. This KCU shall attach to a tether rope that will provide the kite and pod's anchor to the ground. The KCU is controlled by a human operator via a radio link and in addition to the visual feedback, data is transmitted to a computer on the ground for logging and display. Power for the actuation of the control lines and any on-board processors or sensors in the control pod shall be carried on-board or generated in flight. This flight will be performed by actuating the kite control lines. The difference in the positions of the two lines provides the steering control input while the average positions of the lines determines the angle of attack of the kite and thus is related to the lift generated by it. More detailed information regarding the test preparation is covered in the testing chapter.

3.1.1. Remote controlled actuation

The control input, as received by the human-operated remote control radio shall be effected by the actuators contained in this KCU in order to provide the differential and common-mode movements in the kite control lines to steer the kite and change its angle of attack relative to the wind. This will allow a human operator to fly the kite while

only having access to the visible dynamics of the kite and being able to manipulate the control stick of the radio controller.

3.1.2. Launching / Landing

The kite and control pod is to be launched and landed in a manner similar to the launching of a typical surf kite. That is: the kite is positioned at right angles to the wind direction relative to the control pod and held by an assistant so that the lines are taut and the control input can have effect. Once the kite is under the control of the pod, it is released to be flown up to above the control pod. From this position the tether can be reeled out as needed and the kite and controller will be able to fly together.

Landing follows the reverse of this process, the tether is reeled in until the control pod is on the ground. The kite is then flown to the ground at the side of the wind window

3.1.3. Flight

The control of the flight of the kite is achieved by selective braking of either side of the kite through differential movement of the control lines. The line that is retracted will slow the side of the kite that it is attached to and thus cause a rotation of the kite. This rotation is in the clockwise direction (from below) if the right-hand line (facing downwind) is shortened and in the anti-clockwise direction if the left-hand control line is shortened or if the right-hand line is lengthened in relation to it. Through this steering control, the kite can be flown anywhere in the downwind quarter hemisphere from the tether point at the ground station (see section 1.5). The aerodynamic lift generated by the kite is used to keep the kite airborne and to provide a traction force on the tether.

A hand-held controller is used to input the desired control set-points for the positions of the trailing edge lines of the kite.

3.1.3.1. System preparation

Before launching the kite, several activities are required:

1. Weather predictions are to be checked for favourable conditions during flight time
2. Attaching controller to kite bar and lines
3. Ensuring software is logging relevant data (including wait for GPS to lock and zeroing the barometric altitude offset)
4. The tether is to be anchored properly

3.1.3.2. Interpretation of flight data

The data reported to the ground station computer is presented to give the operator feedback on the following data to aid in the flying of the kite:

1. Force on the control lines
2. Position of the kite and KCU relative to the ground station position including tether length and angle to ground (theta)
3. The dynamic motion of the kite controller (acceleration and orientation) relative to the ground.

3.1.3.3. Disabling the kite in high wind

The operator, upon noticing errant behaviour or system failure would be responsible for activating the safety de-power mechanism. The kite and controller will then descend in an uncontrolled manner. Once they are on the ground, retrieval of the various components and note-taking from the situation is important.

3.2. System features

The specific characteristics of the finished system are listed below.

3.2.1. Actuation

1. The controller shall be able to control the kite with a tension of 15kg on each of the control lines.
2. Given the dimensions of a standard bar and lines, each line can have a change in length of 500mm, independent of the other line.
3. The static load on each of the control lines up to 50kg shall have no adverse effects on the actuators.
4. The peak actuation power available shall be sufficient to satisfy the following conditions:
    a) The maximum force of 15kg (150 N) is exerted during the full travel.
    b) Peak actuation power shall be available over the full travel distance of the cable of 500mm.
    c) The total actuation shall take less than three seconds to complete. i.e. at least 167mm/s travel speed.

   d) The on-board power supply shall be capable of providing for two actuators in these conditions simultaneously.

5. Given the fact that the force is transmitted through cables, only a fraction of the actuation power will be required in the release (wind out) direction.

3.2.2. Traction force transmission

To generate power, the lift force of the kite will be transmitted to the winch via the tether. Thus, if the selected design forms part of this chain, it shall have a tensile strength of at least that of the tether or 500kg, whichever is greater. This provides a factor of 5 margin for a power generation system that, to produce a peak power of 10kW, would require a traction force of 100kg and reel out at a rate of 10m/s.

3.2.3. Powering the pod

1. Power for actuation and embedded electronics shall be supplied by a battery included in the pod.
2. The power supply shall be capable of supplying worst-case current to the motors without the voltage dropping low enough to interfere with the operation of the drives or the other electronics. This maximum current is based on the current drawn by the actuators under maximum load with all the other continuous loads present.
3. The power to control the pod shall be available for one hour of normal flight.
4. The total energy required can be stored in its entirety or optionally shall be provided by means of on-board power generation to supplement the battery energy.
5. If this on-board power generation is greater than the average used by the kite controller this would enable unconstrained flight times.

3.2.4. Control input

The device shall have a means of receiving control input from an operator.

1. The latency from user input to the beginning of the actuated response shall be sufficient to be perceived to be real-time. For user interfaces, a system responding in under 100ms is perceived to be instantaneous [53].
2. The data rate of the communication radio link shall be sufficient to transport the required data to the control pod without bottlenecks. No message shall be delayed by more than 100ms.
3. The range of the data link shall have a gain margin of at least 20dB at the maximum distance expected during testing.

4. The sensitivity/performance of the data link shall not be reduced below the minimum specified above when the kite and KCU is at any of the positions of the operating cycle or when the kite is being launched or landed.

3.2.5. Feedback

In addition to the visual feedback received by the operator regarding the flight of the kite, sensor data from the kite control pod shall be measured and transmitted back to the ground station. Data regarding the following shall be reported:

1. The acceleration and rotation of the KCU in three dimensions.
2. The orientation in three dimensions of the KCU with regards to the earth's magnetic field.
3. The ambient and battery temperature.
4. Ambient air pressure at the control pod and thus its altitude relative to the ground station.
5. GPS receiver data to determine position relative to the ground station.
6. Current delivered to the motors by the controllers, indicating the tension in the control lines.
7. The position of each of the control motors' spools and lines i.e. the length of cable unspooled.

3.2.6. Dimensions and mass

As the KCU is to be attached in-line with the tether, about 24m from the kite, the size and mass of the KCU will affect both the aerodynamic drag on the kite's flight and the kinematics of changing the direction of flight rapidly. Thus,

1. The device shall have a maximum size of 500mm x 500mm x 300mm
2. The device shall have a mass of less than 5 kilograms.

3.2.7. Mechanical integration

The device shall provide suitable means for attachment to the standard bar and lines arrangement of a kite as depicted in Figure 3.1. This includes attachment to the leading edge lines that transmit the lift force generated by the kite to the tether, the trailing-edge control lines, attached to the actuators and an attachment point on the opposite side for the tether to the ground station.

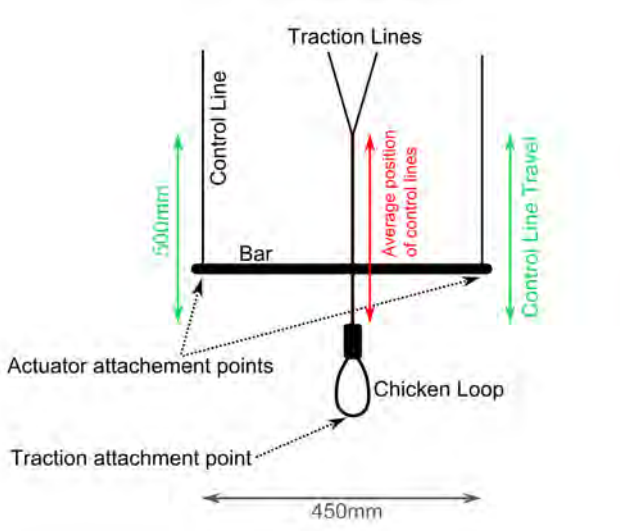


Figure 3.1.: Layout of standard kite bar and lines

3.2.8. Flight time

1. The device shall be able to provide sustained control of the kite for 1 hour.

3.2.9. Safety release mechanism

A mechanism to remove the majority of the lift-force (de-power) from the kite and KCU shall be included to ensure that the kite can be landed safely in the event of kite or KCU malfunction. The mechanism shall be engaged independently from the normal functions of the KCU.

1. The release mechanism shall be able to release the main power lines on command.
2. The structure shall withstand 150kg (1500N) load statically.
3. The mechanism shall be able to release under 800N load.

3.2.10. Ground station data aggregation and display

The data collected by the flight computer and transmitted over the telemetry link shall be displayed in a manner that makes it possible for the operator to fly the kite without visual feedback.

1. The data is to be displayed in a readily understood format, clearly indicating important flight parameters.
2. The data is to be stored for further analysis and system characterisation.

3.3. Environmental stresses

The conditions in which the kite and controller are to be tested will be controlled to avoid extremes but some robustness is required. If this system were to be modified for use over extended periods of time, further measures would be required to ensure reliable performance.

3.3.1. Wind range

1. The module shall be capable of maintaining stable control of the kite over a range of wind speeds, from 10 knots to 40 knots (5 to 20 m/s).
2. The stability of control shall not be compromised by sudden fluctuations in wind velocity, these fluctuations can be up to 50% of the average velocity.

3.3.2. Impact

The device shall withstand impact with sand from a free-fall height of two meters as might be experienced during launching or landing of the system.

3.3.3. Sand and dust ingress

As this unit will primarily be tested on a beach, reasonable ingress protection shall be implemented to avoid it getting clogged with sand while the preparations for flight are being made.

3.3.4. Temperature range

1. The device shall operate continually in ambient air temperatures from 0°C to 50°C and be exposed to extended periods of direct sunlight.
2. This range of temperatures shall not alter the operation of the device by more than 10% of any of the other specified values.

3.3.5. Water immunity

1. The KCU shall not be adversely affected by water spray, mist and condensing moisture but will not be required to be waterproof. i.e. it will have an ingress protection level of at least IP44, preferably IP55 [54].

3.3.6. System and subsystem reliability

1. The system shall be designed to survive 10 hours of flight and given an approximately 3 second travel time of the actuators, there can be up to 12000 actuations of the control lines in the prototype's life.
2. The system shall have some means of reporting errant behaviour or the loss of function of a subsystem or part thereof.
3. The KCU shall monitor available system power and predict remaining flight time in order to pre-empt a loss of control due to depletion of available power and to land the kite before such a mode is realised.
4. The KCU computer shall be capable of resetting during flight to minimize the effect (and time elapsed before restored function) of a software crash or watchdog time-out.

4. Design

The design of a device involves bringing together the specifications and available components and the determining what is needed to achieve the worst-case loads and requirements. This design starts with the overall structure of the robot and then provides details of the decisions made regarding the subsystems and components. The KCU is split into modules that logically separate its functions so that the individual specifications can be tested during the early stages of prototyping, reducing the effect of changes being required.

The design of the subsystems of the kite controller depend on the constraints of the full system and thus must be designed with the integrated whole in mind. This includes the selection of the actuators and the structural elements that provide the framework for the inclusion of the remaining systems.

4.1. Initial design exploration

The control of a kite-surfing kite was specified in the requirements and thus ruled out several of the approaches discussed in section 2.6. These kites have three lines: two for steering and attitude control and one for transmitting the majority of the traction force. This control can be effected in several ways but can be categorised either as ground based control with multiple tethers or airborne control with a single tether.

Ground based control requires a winch that has three spools that can be wound in or out independently of each other. The different tensions in the lines will result in differing elongation and bowing due to wind drag. Due to this the kite may be required to be redesigned to balance the nominal tensions in the lines. This means that ground based control is not ideally suited to being attached to a variety of standard kites. Secondly, the control of the kite and the power generation capability are interlinked and cannot be designed and tested separately, further complicating the overall system design.

Airborne control can be performed on the structure of the kite or in a pod, attached within several meters of the kite. Any control that is integrated into the structure will require major modification of the kite. Furthermore, only certain kites would have a shape or bridle system suitable for the types of actuation that would be possible here.

Modifying a kite to optimise it for power generation is a promising possibility but out of the scope of this project.

Airborne control of the kite using a pod suits this project as it is most similar to the manner in which these kites are typically controlled. This allows for a variety of kites to be tested with this system and for the design effort to be focussed on controlling the kite in flight. One area of concern however is regarding the dynamic movement of the pod, both rolling on the axis parallel with the tether and swinging out when the kite changes direction suddenly. Both of these will be addressed in this design and tested to ensure that they are suitably mitigated to avoid affecting controllability.

4.2. Mechanical structure and integration

The force generated by the lift of the kite is to be transmitted through the structure of the controller thus, it must provide both the strength to transmit that force and the points to which all of the other components or brackets mount. The base structure of the pod could be constructed using several different structures but a single aluminium plate is used as this simplifies the layout of components and force calculations. It provides a plane of symmetry about which the two similar halves can be built.

4.2.1. Baseplate strength

The maximum force to be transmitted through the structure of the pod is limited by the breaking strength of the tether. This is attached to a winch with the ability to keep the traction force below a specified maximum to avoid line breakage and to meet the 500kg specification in subsection 3.2.2. During the testing of the prototype, the pod will be tethered to a sandbag with a mass of 50kg. Therefore, even under sudden loading and acceleration, the worst-case force should not exceed 150kg or 1500N.

The minimum cross-sectional area (A) of material required, to support this load given a yield strength (σ) of at least 170MPa as specified in [55] for 6005 aluminium alloy.

$$F_{max} = A\sigma \tag{4.1}$$

From Equation 4.1 the minimum area to have a yield force of at least 5kN (~500kg) is $A = \frac{F}{\sigma} = \frac{5k}{170M} = 29.41mm^2$.

A 2mm thick aluminium plate has been selected, which requires a width of at least 15mm. Given the uneven loading resulting from attaching the tether to a hole drilled in the plate, an area with a factor of 2 larger should be sufficient to not overstress the material around the hole. A cross-sectional area of at least 30mm^2 around the mounting points on the bottom and top of the pod should be provided.

4.2.2. Packaging and mechanical layout

4.2.2.1. Actuator mounting and winch position

The first components to be placed in the KCU structure are the actuators, with their integrated winches. The cables exiting the winch feed point do so through a flexible guide to avoid kinking if they are not aligned with the directions of the kite control lines.

The actuators are the heaviest components in the system and along with the battery, their positions define the centre of mass of the KCU. The centre of mass is important as if it is located within the line with the power lines of the kite and the tether the rotation of the KCU due to changes in direction can be greatly reduced.

4.2.2.2. Battery mounting and protection

The cells of the battery are fragile and are at risk of explosion or fire if they are punctured or deformed substantially and so must be protected in the way they are mounted and by use of extra structure around them. They also contribute in a significant way to the mass of the KCU and so their location must be chosen with the device's centre of mass in mind.

4.2.2.3. Mounting of electronics

Once the structurally important components are in place, the PCBs of the drive and control electronics are mounted on brackets, attached to the central structure. The motor controllers are mounted close to the motor terminals in an effort to reduce EMI generated in the motor brushes and switching from being effectively transmitted to the other nearby electronics.

4.2.3. Wire harnesses

The integrity of the wiring between the different circuit boards and other components is crucially important to the reliability and continued operation of the system. Usually a single poor or intermittent connection is enough to cause the system to be severely impeded or to fail outright. The cables, therefore, need to be well made and where connectors do not lock in place, glue is to be applied to keep them there. The cables running between modules are: The power cables: from the battery to the power distribution board then splitting off to each of the circuit boards. The RS485 bus cable running between the motor drives and the system controller. The various sensor cables such as the position sensors and inertial motion unit (IMU) and global positioning system (GPS) receiver data cables and the motor cables from the motor drives.

4.2.4. Enclosure

The enclosure or fuselage's primary purpose is to protect the electronics and other sensitive components from damage due to impact or sand and dust and water spray. It also plays a role in reducing the overall aerodynamic drag of the KCU during flight. The specification subsection 3.3.5 requires splash resistance. This can be relaxed in the design of the prototype KCU as the system would not be flight tested in weather conditions with a risk of rain. The kite and KCU falling into open water should also be avoided so extra care needs to be taken on the coast. It would be ideal for the final version (beyond the scope of this project) to be fully waterproof as this would allow for testing the capability of the system for the towing of boats.

The enclosure is to be made mostly of 1mm aluminium sheet, covering closed cell foam. This provides some shock absorption and avoids stressing the internal structure unnecessarily. This sheet will be supported in some places with thicker bent aluminium, especially around the electronics and extending past the battery. Sections of vacuum formed perspex are used where radio antennas need to be mounted without protruding out of the enclosure.

4.2.4.1. Ease of assembly/disassembly

As this is a prototype that will be adjusted and re-worked often during its manufacture, ensuring that it is easy to assemble and disassemble will help expedite the prototyping phase. This is done by using commonplace fasteners and ensuring that the fasteners are accessible i.e. avoiding the need for special tools to reach tight spaces.

4.2.4.2. Aerodynamic drag

The outside of the enclosure will be moderately streamlined but this is not a key parameter given that the system will not be flown at high velocities during testing. The addition of a tail fin is suggested to avoid the device swinging on its axis due to sudden changes of direction.

4.2.5. Mass budget

The KCU flies along with the kite while it is performing it pumping power cycle. The mass of the kite and that of the KCU contribute to the losses of the system and therefore directly affect the system's efficiency or more precisely its effectiveness at power generation. The components that embody the most mass are the actuators, the battery and the structure and housing. For the actuators, described below, the components chosen have as much integrated into a single unit as possible, removing the need for extra brackets to hold the components in line with each other. The battery is made up of high-power Lithium-ion cells which have a high specific energy. The structure and the

housing have been designed to be made out of aluminium and to reduce weight wherever feasible without compromising on strength or rigidity. The total mass budget specified in subsection 3.2.6 is 5kg for the KCU.

4.3. Actuators

The control of the kite is enacted by the change in position of the control bar. As this control bar is connected to the kite using lines, the control is assumed to always be done under some tension. Loss of tension in these lines constitutes a loss of control in the kite and this would be regardless of the controllers used. The actuation mechanism can therefore use winches as no extension force is necessary or even makes sense. The main factors taken into account in the choice of means of actuation are its power density (power per unit volume); specific power (power per unit mass); robustness and simplicity of mounting and mechanical complexity (if the components are separate: spool, gearbox, motor).

4.3.1. Motors and winch assemblies

The objective of this study is the design of a system that is sufficiently able to control a kite-surfing kite, in terms of the force and velocity of the control effort required. These kites have been designed to be human operated so a force of greater than 10kg (~100N) on each line is not expected for extended periods. The force on the control lines varies with apparent wind on the kite and is a function of the lift generated by the kite. The kite is designed to minimize this force as it tires out the operator of the kite and is referred to as bar pressure. It is typically of a maximum of 30kg (~300N)

While the control effort is bounded by the maximum force to be applied, the distance travelled for a full actuation defines the total energy required for said actuation. As with the case of the tension of the control lines on the bar, the actuation distance is also a parameter designed into the kite to improve its operability by the kite surfer. Thus, the maximum travel of any commercially available kite surveyed for in this study has been measured to be approximately 65cm. This is the distance from the closest point of travel to the harness loop to the maximum extension of the operator's arms. In order to maintain control of the kite in turbulent conditions, rapid actuations are frequently necessary but these usually involve the differential actuation of the bar associated with providing steering input to the kite. The maximum total travel time is specified to be three seconds in subsection 3.2.1.

$$P = \frac{Fd}{t} \tag{4.2}$$

Therefore $P = \frac{300*0.5}{3} = 50W$ per motor should be sufficiently powerful to provide the control effort required. This will require an electrical effort of up to 30% larger i.e.

65W of electrical power. Electrical converters are typically the most efficient when the voltages and currents are in the same order of magnitude. The voltage supplied to the motors defines their maximum speed and from testing in subsection 6.1.1, a minimum voltage of 11.5V is required for a travel time of 3 seconds under no load. Further optimisation of these values will require investigation into other parameters, not least the design considerations of the power electronics stage used to drive these motors.

Several motors, gearboxes and spools were identified and their integration was considered but was ultimately rejected due both to the cost of the components and the design/manufacturing effort of combining them into assemblies. A combined commercially available unit was then sought and automotive window winder motors were identified as ideal . These actuators are readily available and already commercialized for low cost and high reliability and can be bought as a pre-assembled winch the ones selected are shown in Figure 4.1. Window lift assemblies are typically located in the doors of vehicles which are classified as 'wet zones' so these motors are also reasonably water resistant and watertight connectors can be used with them. They are designed for a long lifetime in moderately high vibration environments.

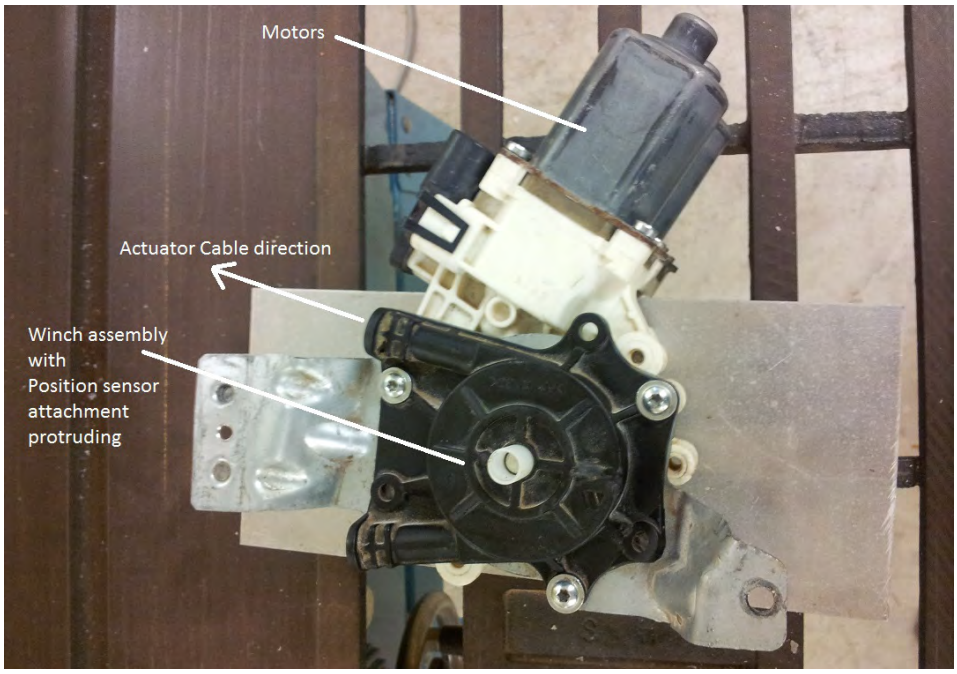


Figure 4.1.: Photograph of motors mounted onto a test plate

4.3.2. Position sensors

A measurement of the position of the actuators is required by the position control system in order to provide feedback control of the actuators. The challenge is to sense the position of the end of the cable, part of which is wound onto the winch. The full

extent of the travel of the winch is 5 rotations. The position of the winch can be sensed by resistive or optical means. Slotted disk encoders can report change in position but not absolute position. Absolute position optical encoders are however too expensive to be viable for this project. Multi-turn potentiometers are much more cost effective and thus selected for this function. They do suffer mechanical damage if forced to turn past their end-stops and thus the software of the controller must reliably avoid this. The ten-turn variant was chosen to ensure that these end-stop positions are well outside of the operating area.

The signal from the potentiometers is measured by the motor control microcontroller with a resolution of 10 bits as this translates to sub 2mm resolution of the position which is sufficient for this application.

4.3.3. Motor drives and controllers

The control effort is to be applied to each of the motors by means of a motor drive system. Several commercially available modules were evaluated regarding the design criteria. The basic specifications of drive current in excess of 20A, up to a voltage of 20V were readily available. The applicability of the control method to this application however, was not as straightforward. The most common form of motor driver for these small applications is designed for three-phase synchronous motors or small DC motors, used in model aeroplanes, cars and helicopters. A circuit would be required to read the position sensor and generate the control signal. The embedded software in these devices would also have to be modified to drive DC motors. For these reasons, a motor driver and controller was designed as part of the project rather than buying part of it.

4.3.3.1. Features of a motor controller node

The function of position control of the motors was integrated into one circuit board. The microcontroller on this board interprets the commands communicated over the control bus and applies these set-points to the software control law. The position measurements of the actuators are obtained from a potentiometer mounted onto the cable spooling mechanism and used as feedback for closed-loop position control. Motor current is also measured and reported over the communication bus along with motor position.

4.3.3.2. Converter topology

The most easily applicable topology for this two quadrant motor drive is the common H-bridge. It comprises of four switches, all of which must be rated to operate under the full load. 30A, 30V MOSFETs are sufficient for this purpose and as there are P-channel FETs available with this rating, there is no need to bootstrap the drive to N-channel FETs on the high side. Each switch is connected to its own driver which is in turn

controlled by output pins of a microcontroller so that any switching waveform can be generated in software. This also allows for the switching to be suspended while the set-point for the current is within a small band around zero to save power.

4.3.3.3. Microcontroller selection

Due to the low processing load and single output set of waveforms to be generated, a simple 8-bit microcontroller was selected. The MC9S08GT16A, made by Freescale was chosen due to the availability of experience regarding this microcontroller. It has a 16 bit timer module with PWM suitable to this application. It also includes a 10-bit ADC for measurement of the position sensor signal and current measurement value.

4.3.3.4. Current sensing and force feedback

The current delivered to the motors by the drive is measured during operation and is reported over the serial bus. Knowing the characteristics of the winch motors allows for the determination of the torque generated by them knowing the current through them. This is used to determine the tension in the kite steering lines which is useful in determining some of the flight characteristics of the kite.

4.4. Energy storage and management

As specified in subsection 3.2.8 the on-board battery power should enable at least an hour's operation. To determine the capacity of these batteries therefore, an estimate must be made as to the load current drawn from them. Peak current occurs when both actuators are winding in simultaneously and can be between 15 and 25A. This actuation takes less than three seconds for full travel but is unlikely to happen often during operation. Small steering movements are performed much more often as the kite is flown. Assuming constant steering action (one motor pulls in while the other pays out) at half of full motor torque, 5A would be drawn continuously.

As the mass of the system is constrained and the batteries are a notable portion of it, Li-ion batteries are suitable due to their high specific energy. The cells chosen have a capacity of 4 Amp-hours at a nominal voltage of 3.7 volts and are shown in Figure 4.2. This is sufficiently close to capacity requirement stated above and as that is expected to be an over-estimate, this characteristic is left to the testing phase to verify. The battery comprises of four such cells, connected in series to provide a nominal voltage of 14.8V and a maximum voltage of 17.0V. This battery has a nominal energy capacity of $14.8V \times 4Ah = 59.2Wh(213kJ)$.

A scaled version of the battery voltage is measured by an ADC in the main system controller. Cell balance and charge protection are handled by the battery charger and

Figure 4.2.: Photograph of cells selected

are thus not needed on the kite controller at this stage. A temperature probe is mounted inside the battery casing and the voltage produced is connected to another ADC channel of the microcontroller.

The cells selected have a power rating of 50C that is $50 \times 4(Ah) = 200A$ instantaneous current capability without incurring damage. This is far above the ratings of the other components of the system and so the fuses would blow long before the battery is affected. A 30A fuse is included before the positive battery terminal.

4.5. Embedded system architecture and microcontroller selection

The KCU is centred on an embedded computer that aggregates the sensor data and parses the control inputs for communication with the motor control nodes. Processors that can fulfil the functions required are available in many different devices from many different manufacturers. The features required include an ADC with at least 12-bit resolution, three asynchronous serial ports, one I^2C port and several timer capture channels. A 32-bit core is preferable due to the number of mathematical calculations being performed and the code execution speed improvement over 8-bit cores. The flash and RAM sizes are not a limiting factor as cores with resources ample for this application are available.

The development environment available for a particular core is also important in the selection as good tools can greatly reduce the software development time-line and complexity. This includes peripheral driver code templates for standard peripheral functions. Support for a real-time operating system is also useful for this application to support the development of concurrently functioning software, reducing the risk of timing related issues later in the development path. An ARM Cortex M0+ microcontroller made by Freescale semiconductor has been selected both because it fulfils the above feature requirements and because there is a hardware development board available that aids in system prototyping. The microcontroller has a part number of MKL25Z128CLK4 and operates at up to 48MHz with 128KB of flash and 16KB of RAM.

4.6. Control input

A standard 'RC' controller has been selected for operator input as it provides a ready-made, robust, hand-held controller. These devices are extremely cheap and standardized. The receiver, outputs a digital pulse stream, 20ms apart with a pulse width of 1-2ms. There are four outputs on the model selected and two of these are connected to the system controller and fed into a capture/compare timer module to determine the pulse widths. The RC receiver is powered from the 3.3V rail of the microcontroller so the input signals can be read directly without dividing down from the typical 5V signals.

4.7. Telemetry

The RC controller was chosen to provide a robust low-latency control input to the KCU. It however, only provides communication in one direction. A bi-directional RF transceiver, the *Nordic semiconductor* nRF905 module is included on the master controller to provide a return path for the data measured in the KCU. The pair of communication modules provide a serial data stream and operate in the 900MHz ISM unregulated band. These modules send 32 byte packets in each direction as the RF link is actually half duplex so there are performance issues in sending smaller packets.

4.7.1. Link budget

The nRF905 module datasheet [56] specifies the output power to be 10dBm and the receive sensitivity to be -100 dBm. From these figures and knowing that omnidirectional monopole antennas are to be used, a link budget can be calculated to determine an estimate of the maximum distance between the telemetry transceivers while maintaining almost(less than 5% error rate) continuous communication.

$$P_{RX} = P_{TX} + G_{TX} - L_{TX} - L_{FS} + G_{RX} - L_{RX} \tag{4.3}$$

where transmitted power is $P_{TX} = 10dBm$ and both TX and RX antenna gains are $G_{TX} = G_{RX} = 2dBi$, the losses are estimated to be less than 1dB on each side and the freespace loss can be calculated using

$$FSPL = (\frac{4\pi d}{\lambda})^2 \tag{4.4}$$

where λ is wavelength in meters and d is the line-of-sight distance between the antennas. This can be converted to decibels and scaled to

$$FSPL(dB) = 20\log_{10}(d) + 20\log_{10}(f) + 32.45 \tag{4.5}$$

for distances in kilometres and frequencies in MHz.

Therefore at distances of 1km and a frequency of 866MHz the path loss is -91.2dB. So to sum up, $P_{RX} = 10 + 2 - 1 - 91.2 + 2 - 1 = -79.2dB$ which when compared to the receiver sensitivity, gives a link margin of 20 dB which should be sufficient for reliable communication in direct line of sight.

4.7.2. Kite to ground data

The majority of the data being measured in the KCU is needed, reported and recorded in the ground station. The system controller sends packets of 32 bytes each to the ground station in response to the receipt of a packet from the ground station. The data packet is described in appendix subsection A.2.2 but in summary the data transmitted are the motor currents, motor positions, controller inputs, battery voltage, GPS location, KCU Acceleration (3-Axis), pressure(altitude) and system status.

4.7.3. Ground to kite data

In addition to the direct control input, the operator can adjust some of the other functional parameters of the control pod. This can be used to tune control input sensitivity and override certain default configurations such as the maximum and minimum positions of the actuators and the maximum speed with which the actuators can be driven. Table A.1 describes the data contained in the packets sent in this direction.

4.8. Sensors

In order to provide this study with performance and motion dynamics data, several sensors are implemented in the controller and the measurements are sent back to the ground station via the telemetry link for live display, and stored for later analysis. An off the shelf, encapsulated GPS module has been selected which outputs NEMA0189 packets over a TTL level serial connection. A motion sensing daughter-board has been selected as the inertial motion unit which is connected via I2C.

4.8.1. GPS

The GPS receiver provides an absolute position measurement and is connected to a UART module on the system-controller and reports several standard *NMEA 0183* sentences once per second over a TTL level serial connection. These readings indicate latitude and longitude coordinates, altitude and GPS time (in UTC) and some derived figures such as velocity and heading. This position information is used to derive the

relative position of the kite controller to the ground station. Figures of note to the operator are the angle of the kite to the wind, the angle of elevation and the tether length. These can be calculated from the difference between the KCU and ground-station GPS readings.

4.8.2. Inertial motion unit

To determine motion information at a higher sample rate than the GPS can provide, an IMU comprising of an accelerometer, a magnetometer and a gyroscope, each of which having three sensing axes - in the X, Y and Z directions for the former two and pitch, yaw and roll for the latter. Once this information is filtered and synthesized, the full motion of the controller can be monitored and recorded. These sensors are all mounted on a daughter-board that is held in place on a piece of foam, acting as a vibration buffer.

Magnetic orientation According to the *National Geophysical Data Center* the earth's magnetic field intensity in Cape Town, South Africa, where the development and testing is taking place, is 26 000 nano Tesla. It has an inclination of 66° up (negative) and a declination of 25° west (negative) [57]. This can be used to offset the measurements made by a three dimensional magnetometer to determine the orientation of the robot.

4.8.3. Barometer and temperature sensors

Barometric pressure sensors are used to determine altitude relative to the start position more accurately than the GPS receiver and is included in the IMU daughter-board. The part is the BMP05 made by Bosch sensortech. It also reports air temperature and includes a measurement IC so that the information is accessible over the I^2C bus.

The temperature of the battery is measured using a LM35 analog temperature sensor with the signal measured by the ADC of the master controller. If the battery temperature starts to rise rapidly, the operator would be warned of the situation before system failure became likely.

4.9. Electrical integration

Once the details of each of the electrical subsystems is defined, the integration of them into a functional whole is considered. This affects the implementation of each subsystem as the interfaces between them, physical, communication and power provide this compatibility.

4.9.1. System coordination

The actions of the subsystems can be controlled either in a centralised or distributed architecture. The core functions of the system are centred around the motor control and feedback and the balance of functions that can be all coordinated by a single processor. The system will therefore be designed around a single main processor, handling radio communication, sensor reading and commands for the motor control nodes which will be considered to be self-contained in function.

4.9.2. Power supply

The main controller requires 3.3V and 5V supplies but is supplied with 12-17V battery power. A buck converter module is used to reduce the battery voltage to 5V in an efficient manner. The 3.3V is supplied from the 5V using a low drop-out voltage regulator for the microcontroller and other low power electronics.

4.9.3. Grounding strategy

Power is distributed from a single point close to the battery, thus, a star based grounding pattern is employed to avoid ground loops. This ground is joined to the chassis close to the power distribution board so that the chassis can be effectively used as a shield for interference. Each electronic module's power supply incorporates decoupling capacitors to this ground to reduce the inter-module interference. As the pod is not electrically connected to any other system while it is operational, no other interface compliance is required.

4.9.4. Sources of electromagnetic interference

Each of the modules will be subjected to and must be robust to electrical interference. The majority of this will come from the other modules in the KCU as it is a closed system(battery powered and enclosed in metal) and far from other sources of noise. Notable sources of noise are: the motor brushes - as the commutator switches between sets of windings there are spikes in the voltage; the drive switching noise - the sharp/high speed transients in the power switching can cause spikes on the supply voltage. The buck regulator providing 5V power to the master controller also would introduce some switching noise. The power supply to each subsystem will use decoupling capacitors to improve the robustness to supply voltage fluctuations.

4.9.5. Sharing the battery power supply

Dips in the power supply due to high loading would possibly disrupt the other modules. This can be mitigated by using batteries with low internal resistance and separate, thick

conductors to the modules. Ensuring that low voltage circuitry is properly decoupled will also help it survive the transients that have an effect through their respective voltage regulators.

4.10. Embedded software design

As mechatronic systems become more advanced and complex, software, running on an embedded processor becomes increasingly important in providing both robust and flexible solutions. In the KCU, there are two embedded software programmes which are run on the system controller's microcontroller and the motor control circuit's microcontroller. Both programs are written in C and compiled to be run on their respective processors.

4.10.1. Motor controller software

Each of the motor controller's circuits include an 8-bit Freescale microcontroller that executes the software written for it. This software is responsible for generating the switching waveforms for the motor control, measuring the analog voltages output by the position and current sensors and receiving and transmitting messages on the RS485 bus.

The code routines are predominantly interrupt driven, firing off small sections of code to update the state of the internal variables before going back to spooling for 'flags'. A periodic interrupt triggers the ADC readings and the updating of the PWM duty cycle, with the required calculations regarding scaling of the ADC readings and the position control loop being triggered to be run outside of the interrupts by flags set within them.

As each character is received on the serial port from the RS485 bus, an interrupt is fired that builds up a message string for interpretation. If the address of the message match the address of the node, the message is processed, using its values to update the internal registers and a reply is generated and put into the outgoing buffer.

This software is written into the flash of both of the motor controller's microcontrollers with their different address bits being the only differentiating information for them to be addressable over the RS485 bus.

4.10.2. System controller software

The system controller has several different concurrent functions so a Real Time Operating System (RTOS), namely, *FreeRTOS [58]* has been implemented to improve the time independence of these tasks. This allows one function to be paused, waiting for further information, without interrupting the other functions of the system and so improving the modularity of the functions performed by this software. Increasing the modularity of software functions reduces their interdependency and so makes the system more robust to partial failure. For example, if the GPS receiver stops providing data, the timing of the telemetry communications is not affected.

Control input The RC receivers pulse train input is measured by a timer which fires an interrupt on each edge of the signal. The times between rising and falling edges for each on the input channels is recorded and at that point, a request to perform that calculation is put onto the task's queue for processing outside of the interrupt context. This processing provides automatic ranging of the control inputs as the absolute values could drift over time. The output of this automatic range adjustment is used to linearly interpolate the motor position set points before they are sent to the motor drive nodes.

RS485 bus communication The system controller as the master on the RS485 bus, initiates all communication. Every 50ms, a message is sent to each of the motor controller nodes containing the new position or speed set-point (depending on the mode it is in) and the position limits for that controller. The motor controller replies with its latest position and current measurements. These measurements are passed on to the telemetry task for reporting to the ground station.

Sensor reading There are a variety of sensors connected to the system controller that all need to be measured periodically. The accelerometer, gyroscope, magnetometer and barometer are connected to an I^2C bus. On start-up, they must be initialized to measure continuously and polled to receive those measurements. Some of the sensors such as the barometer require additional calculations to be performed before usable values can be reported to the telemetry task for transmission to the ground station.

GPS communication The GPS receiver sends a set of text messages once per second that are interpreted by this task. Once a location message is received, the latitude, longitude and altitude variables are updated.

Telemetry The telemetry task waits for messages from the ground station indicating the operational mode and various other configuration values. It replies to these messages with the data sent to it by the other other tasks such as sensor information or motor positions.

Safety release A signal in the message from the telemetry task can also indicate that the operator has requested that the safety release mechanism be activated. This message would then be sent over a digital signal output to the line release subsystem.

4.11. Auxiliary systems

The following systems fulfil their own, isolated function and although they connect to other parts of the KCU, they do not partake in the functioning of the other systems.

4.11.1. Energy harvesting

The KCU is battery powered and thus has a limited time of operation before requiring landing and recharging. On-board power generation could be used as a method of supplying the KCU's power requirements and therefore increasing this operational window and possibly making it unconstrained by the battery's capacity. Effective use of this generation can also reduce the energy storage capacity requirement and possibly help reduce overall weight. One way of removing the battery entirely is to deliver the power required through the tether. Although there are notable systems making use of this, such as SkySails' kite control pod, it is not considered viable here as it requires a tether with at least two electrical conductors in it.

There are several options for this energy harvesting including PV solar, wind and kinetic energy recovery. Kinetic energy recovery is done by means of a mass that slides along a rail or an off-centre mass on a shaft. It is deemed undesirable due to the relatively large mass required for meaningful power capture and for the way this mass interacts with the dynamics of the pod. The use of solar panels for this generation has several advantages such as its lack of moving parts and the manner in which they can be integrated into the body of the KCU.

Good solar panels are available with a typical output power of 100W per m^2 [59] thus if panels covered the entire KCU which is designed to have dimensions of about 300mm by 300mm by 200mm and so a projected area of between 0.06 and 0.09 m^2. The power output would thus be between 6 and 9W, which at 15V would provide a charging current of approximately 500mA. This current is less than the estimated current consumption of the KCU as discussed in section 4.4 above to be 2A and the minimum required for the generation to fully power the device. One way to increase the area of the panels and with it, their power output would be to put the panels onto the wind capture wing. This is however, out of the scope of this project as it is concerned with using standard inflatable kites and not rigid wings

Finally there is the option of including a small propeller either on the leading side of the controller or at the end of a tail. This benefits from the high apparent wind at the KCU due to the cross-wind flight path used in the ground based power generation. A propeller with a blade length of 125mm is selected which has two blades and a swept area of 0.05 m^2. Without knowing the $\frac{C_l}{C_d}$ of the propeller, it is difficult to calculate the power output of such a device but assuming that an output of half of the Betz limit is feasible, an estimate can be derived as follows. The power in the wind mass per unit area is $P = \frac{1}{2}\rho A v^3$ and the Betz limit states that no more than $\frac{16}{27}$ of that can be extracted [60]. Thus, half would be $P = \frac{8}{27}\frac{1}{2}\rho A v^3$ if 30W is required continuously then the apparent wind velocity must be at least 15m/s or 30 knots which is easily achievable in wind as low as 8 knots.

Thus, for cross-wind flight, an on-board generator is a viable option but its inclusion is not crucial to the testing of the other functions of the pod.

4.11.2. Safety release

A secondary mechanism needs to be available to reduce the kite's lift force in event of malfunction of the primary control actuators. In kitesurfing, this is typically done by releasing the control bar to the point where the kite's rear lines go slack and the angle of attack is therefore parallel to the wind. The only connections to the control lines are the actuator winches therefore, making them able to be released would complicate the normal functioning of the KCU and so becomes an inelegant solution.

Thus a mechanism can be built to release the tether connection from the control pod so that the tether is only joined via a usually slack secondary connection to a point further up on the leading edge lines. This has the effect of disrupting the kite's flight and it would probably fall in an uncontrolled manner. The safety release function is only included to avoid the untenable situation of a malfunction causing an uncontrolled, full force and power flight. This component consists of a clasp mechanism and a servomotor and would be mounted on the lower side of the KCU where the tether is connected. The activation of the safety release mechanism is communicated the radio link between the release trigger circuit at the ground station and the KCU. Providing a separate radio link for this improves functional redundancy.

For the low altitude and static testing this would be replaced by a fixed link and the safety de-power action is implemented by using a separate line between a point on the leading edge lines and the ground.

4.12. Ground station

The purpose of the ground-station is to provide the human operator with a means of inputting control set-points and information regarding the flight. All this is to be logged for analysis after the testing. As described in section 4.6 above, the actuator set-points are input by means of a typical hobby RC controller with this signal being received directly by the KCU. Initially, the majority feedback regarding the flight of the kite would be visual, while the testing still uses short tethers and the distance between the operator and the kite is less than 100m. As the kite gets further away this will get more difficult and the information relayed over the telemetry link thus becomes more important.

4.12.1. Display and logging of controller data

The ground station side telemetry module is connected to a laptop, running software for the display of the flight data as well as providing controls for the operator to control certain flight parameters and override some control inputs.

4.12.1.1. Data received from the KCU

As described in appendix subsection A.2.2, the KCU reports data at a rate of 10Hz and includes the following values

- Actuator positions displayed as a pair of vertical bars.
- Control set-points (processed from the RC receiver in the KCU) displayed as bars next to the positions to show actuator position tracking.
- Acceleration in the X, Y and Z planes will initially be displayed as a bar graph with three bars and a graphical representation of the motion of the kite is left to further work.
- Barometric pressure and air temperature are displayed as numeric values with the altitude being derived from the pressure.
- GPS coordinates of the KCU is used to calculate the position of the KCU relative to the ground. Top and side view graphics depicting altitude, elevation angle and azimuth angle and tether length can be generated.
- Motor current measurements indicate the forces on each of the actuators from the control lines of the kite. These are displayed in a line graph.
- Battery voltage is displayed as a number on the front page of the application with a separate tab for a line graph of the voltage over time.

The data is logged for post-flight analysis and a subset of it that is relevant to the operator is displayed on a graphical user interface (shown in Figure 5.12) by means of graphs and diagrams and numerical values.

4.12.1.2. Data transmitted to the KCU

In addition to the control set-points sent by the RC controller, some commands are transmitted from the ground station to provide the KCU with limits for the motor's positions, maximum speed and mode. A motor drive duty override value is also sent by this means and is used mainly for testing in the motor controller's speed control mode (the other being position control).

4.12.2. Ground based measurements

Data from several sensors on the ground are also received by the ground station software. These include a GPS receiver for relative position calculation and in the future, an anemometer and vane for wind velocity measurement. They are connected to the computer by USB and are logged alongside the data from the kite by the ground station software.

5. Implementation

Once the broader design decisions had been made, attention could be given to the component choices and building up the subsystems one at a time and integrating them into a whole. Some of the tests in the following chapter would have been done on the component level before system integration, for example: the characterisation of the motors and winch assemblies. This chapter documents the as-built designs and some of the adjustments made during the prototyping of the parts and assembling the KCU. In some cases, such as with the system management circuit, a mock-up was implemented first to make the development of the embedded software and the interconnected systems easier. Once the changes and issues had been addressed, the final version could be built much more quickly. This iterative prototyping approach assisted in exploring the problem-space and and avoiding time spent on technical dead-ends.

5.1. Electronic circuit boards

The electronics of the system are centred on three circuit boards, namely the system controller and the two motor drive circuits. There are other electronic components but they are all connected to these boards for control and integration.

5.1.1. Motor drive PCB

Once the motors and batteries had been selected and the motors characterised as in subsection 6.1.1 and the maximum expected load on the drives calculated, the circuit design could be finalised and the PCB laid out. As discussed in the design, the drive circuits are required to supply the motor with a controlled voltage up to the input voltage maximum of 20V and up to a current of 20A in either direction. The choices of the specific components making up each of the circuits are given below.

5.1.1.1. Switches and switch drivers

The H bridge is made up of two symmetrical half bridges, each connected to one terminal of the motor. The high-side switches (T1 and T2 in Figure 5.1) are SUD45P03 P-channel MOSFET MOSFETs which have a 45A continuous current capability and a 30V maximum drain-source voltage. The low-side switches (T3 and T4) are IRF1010

N-channel MOSFETs, rated for 60V and 84A. They have 8mΩ and 12mΩ on-resistances respectively at a gate-source voltage of 10V. Therefore, the worst-case current of 20A would cause an I^2R power dissipation of $20^2(0.012 + 0.008) = 8W$ in the switches. In parallel with each MOSFETs source and drain is a free-wheeling diode, an S1A capable of 1A and 100V reverse blocking, connected in parallel and the same direction as the body diode of the FET. Each half bridge has a TC4428 dual power MOSFET driver IC made by Microchip which has one inverting driver, connected to the high-side switch and one non-inverting driver connected to the low side switch. In this way, a high input into the driver always means a switch is ON. A 22Ω resistor is connected between the output of each driver and the gate of the respective MOSFET. This limits the current into the gate to $\frac{17V}{22\Omega} = 0.77A$, about half of the rated 1.5A capability of the drivers. The P-channel device will not be switching so the I^2R losses will be the total loss of this device. To charge the 4nF input capacitance of the MOSFETs to their maximum threshold voltage of 4V at 0.77A therefore takes up to

$$Q = \int i(t)dt = CV$$

so $t \approx CV/i$ which is $t = 4nF \times 4V \div 0.77A = 20ns$.

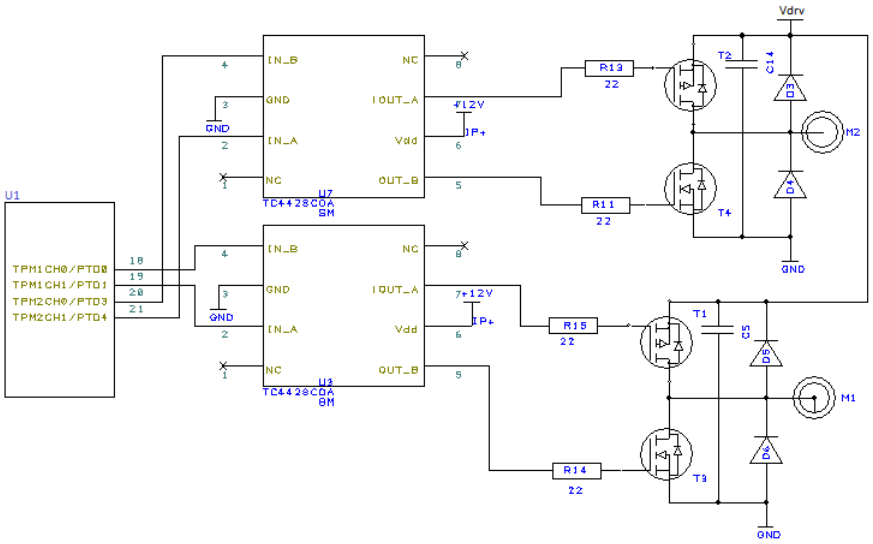


Figure 5.1.: Motor drive power electronics schematic

5.1.1.2. Internal power regulation

The power supply to the drive circuit from the 4-cell lithium-ion battery can be from 12V to 17V but the microcontroller requires a voltage of 3.3V and the RS485 transceiver, a 5V supply. As the digital electronics on this circuit require less than 50mA, it is feasible to use linear regulators for these supplies. The LM1117 adjustable regulator is used for

both of the voltages with the 3.3V regulator being daisy-chained off the 5V rail. These regulators have an internal reference of 1.25 volts and a resistor divider is used to set the voltage using the formula:

$$V_{out} = 1.25(1 + \frac{R_1}{R_2}) + 50\mu A \times R_2 \tag{5.1}$$

With this, the resistors for the 5V rail were calculated as R1 = 2.2kΩ and R2 = 6.2kΩ and for the 3.3V rail R1= 3.3kΩ and R2 = 2.2kΩ. The power supply is decoupled with 10µF ceramic capacitors close to the inputs of the regulators and 1µF capacitors close to their outputs.

5.1.1.3. Drive current measurement

The current flowing through the motor and drive is measured using an isolated hall-effect linear current sensor, the ACS711 made by Allegro. The output signal of this sensor is based on the current through the path from pins IP+ to IP- and scales with the supply voltage, which is 3.3 volts in this case. When no current is present, the output is $\frac{V_{supply}}{2}$ or 1.65V and has a slope of 55mV per Amp. The output therefore is $V_{supply} - 0.25V$ when 25A is flowing from pin IP+ to IP- and 0.25V when 25A is flowing from pin IP- to IP+. This is ideal for the ADC's inputs which use the same supply as a reference. The actual motor current is calculated from the raw ADC value in the motor controller software and reported over the control communication bus.

5.1.1.4. Motor position sensing

The winch position sensing is done by connecting its shaft to a 10-turn potentiometer, mounted on a bracket on the side of the actuator housing. The potentiometer's resistor is connected across the same supply as is going to the ADC reference pins and so the output, connected to an ADC input, is proportional to potentiometer position only and can scale with the supply voltage. The potentiometer output or tap voltage measured with a resolution of 10bits, after averaging and thus splits the full travel of the pot into 1024 counts or 102.4 counts per revolution of the cable winch. As the nominal winch diameter is 46mm this equates to a position resolution of the cable tips of $46 \times \pi \times \frac{10}{1024} = 1.4mm$. A 100kΩ pull-down resistor is also connected to the ADC input to ensure that if the sensor becomes disconnected, the ADC will read a value of zero and not remain floating. The zero reading is interpreted as sensor failure by the software and the motor drive is disabled in this case to protect the cable winch from over-winding in either direction.

5.1.1.5. Connections to the microcontroller

The two half bridges' drivers are connected to the PWM outputs of TPM1 and TPM2 of the microcontroller respectively this is shown in Figure 5.1. The rest of the circuit is

shown in Figure 5.2 where the analogue signals from the position and current sensors are connected to the ADC channels, ADP2 and ADP1 respectively. The RS485 transceiver is connected to the Serial Communication Interface (SCI2) on pins PTC0 and PTC1 with the transmit enable of the driver being output from PTC5. The node address jumpers are connected to PTA2 - PTA5 and are configured as inputs. A 32kHz quartz crystal is connected to the XTAL and EXTAL clock driver/input pair.

The microcontroller is powered from the 3.3V supply with 100nF and 10µF capacitors close to each of the supply pins. The analogue reference is also connected to this supply. The debug adapter header is also brought out onto the 6-pin header as required for programming and in circuit debugging. This header exposes the processor's debug pin, reset pin and Vcc and ground.

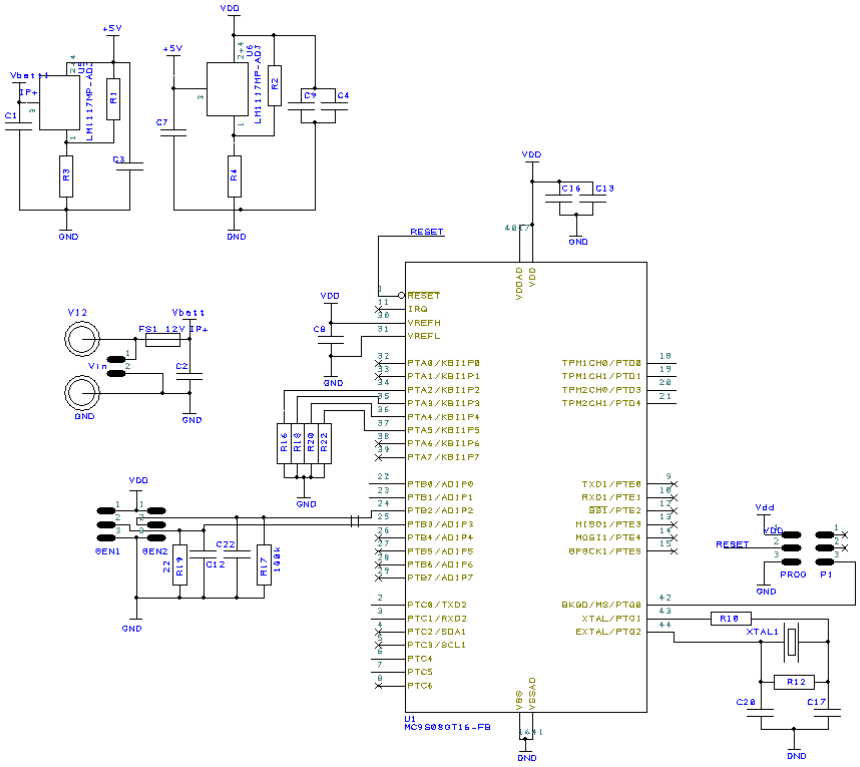


Figure 5.2.: Motor drive microcontroller schematic

5.1.1.6. Communication with host

The motor control nodes communicate with the system controller over a RS485 bus. Each device on this bus has a device address, set with solder jumpers connected to four of the port pins of the microcontroller. Each of these pins can be connected to Vcc or Ground to signify a 1 or a 0 in that bit of the slave address. With these four inputs, 16

addresses can be set. This hardware address set-up avoids an individual address having to be written into the flash of the microcontroller. The system controller masters this bus and is the only node that can transmit on the bus without first receiving a request. It sends commands and gets replies in a call-response manner.

A RS485 transceiver, the MAX485ESA from Maxim Semiconductor is used for signal translation to the bus and space on the circuit board is made for termination resistors to bias the bus correctly. These can be populated if needed. In the bus as it has been implemented here, one of the motor nodes has termination resistors (a pair of 820Ω resistors pulling the lines high and low and a 150Ω resistor between them) and the system controller, at the other end of the bus has the other.

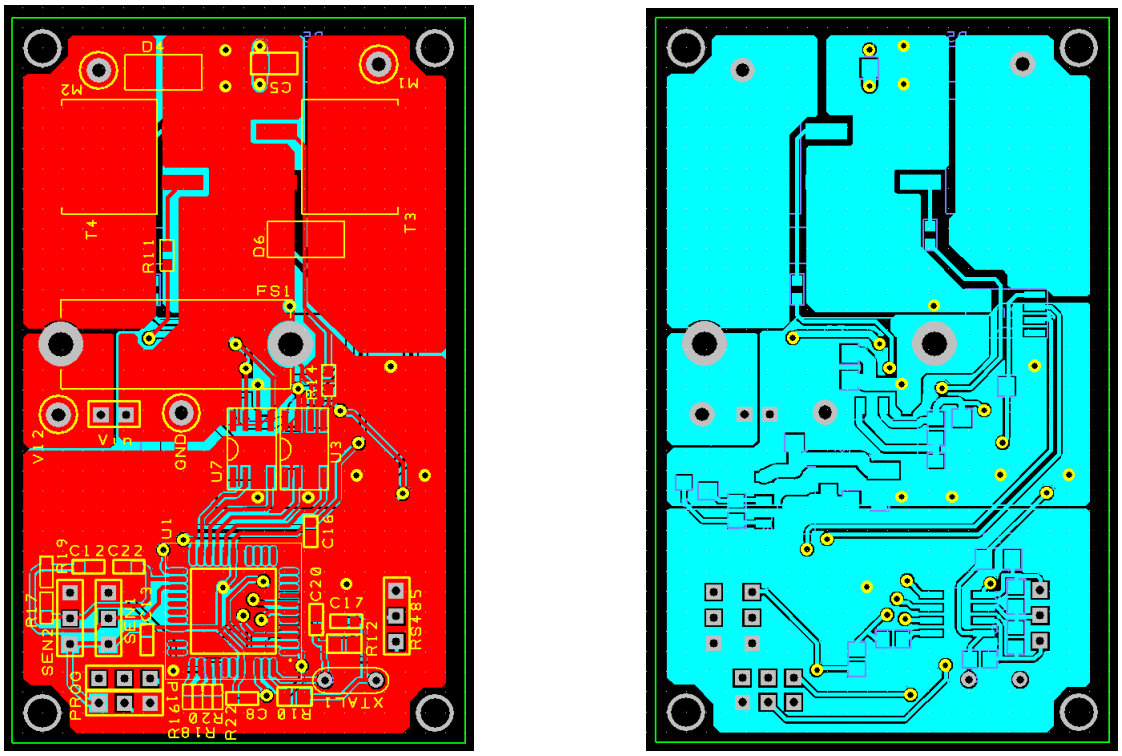
5.1.1.7. PCB layout

The three main constraints on the drive circuit boards are: to make use of a two layer board to make prototyping easier, to not require active cooling or heat-sinks and to make the board relatively small to fit in close to the motors. The size was the least strict constraint but it definitely needs to be smaller than 10cm in either direction.

The layout was started on the high-power side of the board and the components that would be exposed to fast transients or high currents are grouped together. The ground net is split at the power connector to avoid ground loops between the high current, switching side and the more sensitive microcontroller and analogue signal side. The MOSFETs are placed on large pads to help spread the heat generated in the devices so that the need for extra cooling would be minimised. The supply and ground areas are also poured in overlapping sections on either side of the board to improve capacitive coupling between them. Space is left between the supply traces and the outputs to avoid coupling too much noise across. Decoupling capacitors are placed close to each of the components that they supply and between the supply pours close to the switches. The result of this layout is shown in Figure 5.3 which, once fabricated from this design was populated and can be seen in Figure 5.4 below.

5.1.1.8. PCB manufacture, population and power on tests.

The boards made for the first prototype, which ended up, after a few minor modifications, working well enough to be used for the remainder of the project were milled using UCTs circuit board milling machine. The board was populated in parts. First the linear regulators and fuse were populated. The board was powered on to check that the correct voltages were present and that there were no shorts. Next the MOSFETs and drivers and current sensor were populated and the power on test repeated. Finally the microcontroller and RS485 transceiver and the remainder of passives and headers were populated and the power on tests repeated before the microcontroller was programmed. Once these tests had been completed, the board was programmed and progressively tested with all of the active functionality.



(a) Top Copper **(b)** Bottom Copper

Figure 5.3.: PCB Layout

5.1.2. Inrush limiter and power distribution board

An issue was found with the motor drives in which they sometimes suffered MOSFET failures when attached to the battery voltage. The inrush, and the inductance on the leads were causing a voltage spike to appear on the supply rails and this voltage is above some of the components specified limits. This issue was resolved with the addition of a inrush limiting circuit on the power distribution circuit board. On start-up, there is a 10Ω resistor in series with the supply. After 50 milliseconds, a MOSFET is switched on to bypass this to avoid having too much supply resistance. A 0.05Ω resistor remains in series, along with up to 24mΩ of the IRF1010 N-channel MOSFETs.

This board includes a 25A fuse and another MOSFET that is switched on by the low-current power switch, this also powers an LED to indicate that the system is powered. The switch and LED, indicated by the red square in the schematic in Figure 5.5 were mounted out the chassis to be accessible when the device is fully assembled.

5.1.3. System master controller

The system control circuit provides connectivity between the various subsystems and the master microcontroller. A FRDM-KL25Z development circuit board, containing

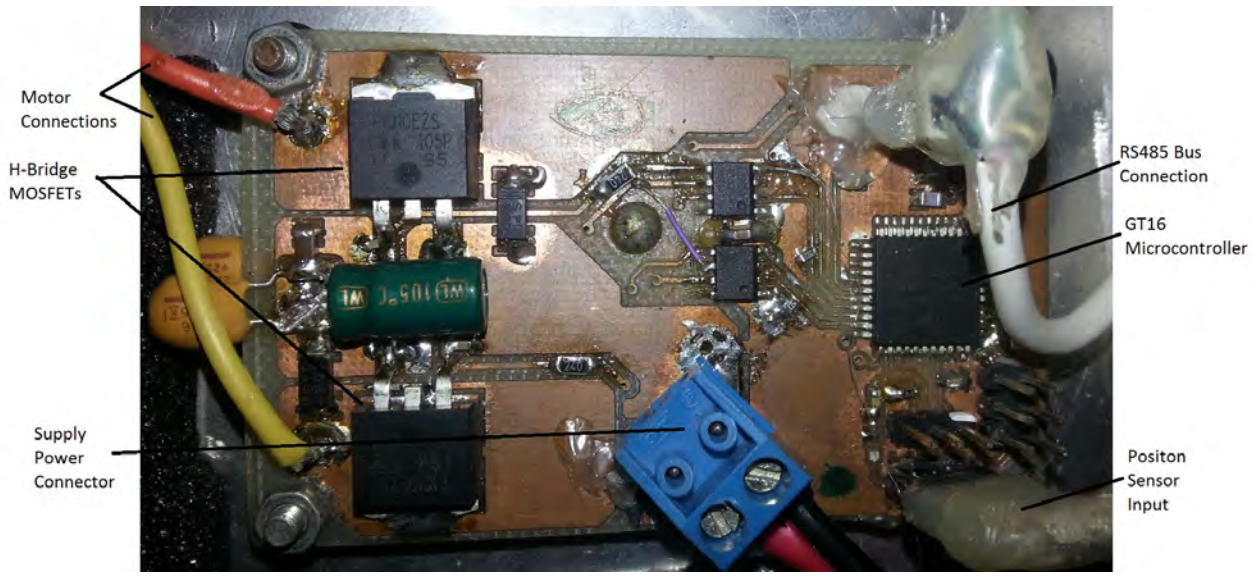


Figure 5.4.: Photograph of drive mounted in position with wires attached

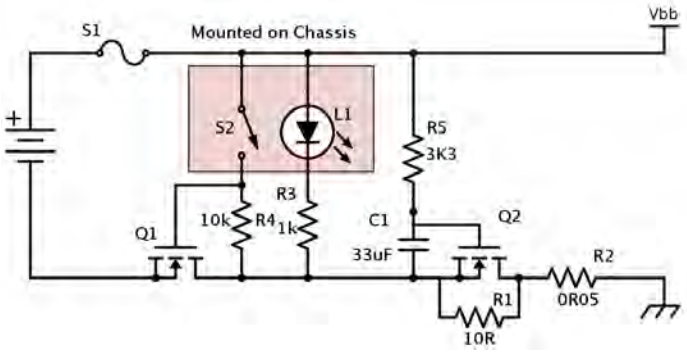


Figure 5.5.: Inrush limiter schematic (author's own diagram)

a Freescale KL25Z Cortex M0+ microcontroller and a built-in OpenSDA debugging companion chip was used. This development kit mounts onto a connectivity motherboard with headers to connect the GPS, IMU, telemetry module, RS485 bus, RC receiver input and the battery temperature sensor. A resistor divider, connected to a 16-bit ADC now measures the battery voltage which is converted down to 5V with a switch-mode buck regulator module.

5.1.3.1. Telemetry daughter board

A nRF905 [56] RF transceiver module is mounted onto a pair of 2.54mm pin headers and the RF antenna is connected directly to it as is visible in the top left of Figure 5.6. It is supplied with 3.3V and connected to a UART of the microcontroller to facilitate the communication for the telemetry link. The modules are then set-up with a transmit and receive address of 033 on radio channel number 200 and the baud rate is set to

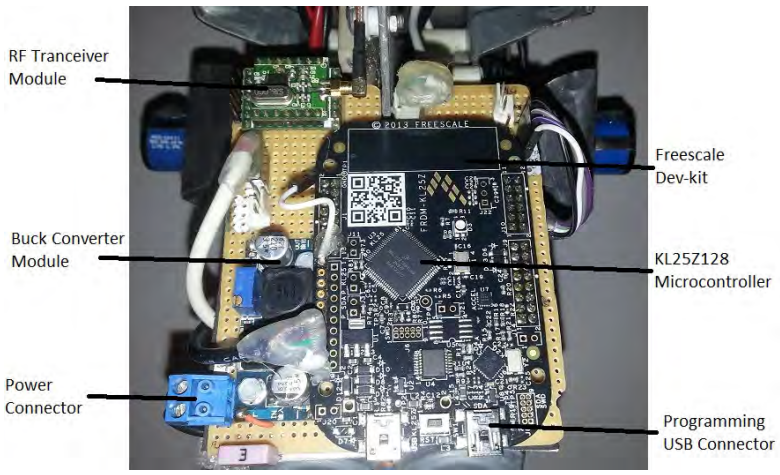


Figure 5.6.: Photograph of the system control circuit

19200. The configuration of the module is changed by grounding the configuration pin and sending new parameters over the serial port.

5.1.3.2. Power supply

The battery voltage which ranges from 12V when flat to 16.8V when fully charged is converted down to the 5V supply rail for the master controller board by a buck regulator module. A stand-alone pre-qualified adjustable module was used to save power compared to a linear regulator and to reduce risk and development time of the system. The 5V rail is further regulated to 3.3V to supply the microcontroller. This is done using a 1117 low drop-out (LDO) regulator.

5.1.3.3. RS485 transceiver

The same MAX485, RS485 transceiver that has been used on the motor control nodes is implemented here for communication with them. It is powered from the 5V rail and is connected to a UART of the microcontroller as well as a digital output pin that controls the data direction. This pin is set high while the master controller is transmitting and low when it is ready to receive data from one of the nodes.

5.1.3.4. Radio controller input

The radio control module outputs four standard “RC servomotor control pulse-width-modulated signals with an on-time of 1 to 3 milliseconds and an off-time of 20 milliseconds. Two of these signals are connected to a capture-compare timer module of the microcontroller and represent the up-down and left-right channels of the right-hand stick of the radio controller.

5.1.3.5. Battery voltage measurement

The battery voltage is divided down to the range measurable by the ADC of the microcontroller using a resistor divider with the values of 22kΩ and 3.9kΩ giving an output of 3.3 at 22V. This is far in excess of the typical maximum battery voltage of 17V but still has a resolution of 50µV in the range of 12 to 17V because of the 16-bit ADC present on this microcontroller. The worst-case current consumption of this resistor divider is 660µA.

5.1.3.6. GPS receiver module

The GPS receiver that has been selected is a sealed module with a GoTop receiver and integrated antenna. This device communicates over a CMOS level asynchronous serial interface. As the device needs to detect extremely faint satellite signals, it has been mounted on the top of the kite control pod, outside of the enclosure. The system controller board includes a 4-pin connector, to which the cable from the receiver connects. The module is powered with a 3.3V supply over the power and ground connections with the other two connections being serial receive and transmit signals.

5.1.3.7. Motion sensor module

The movement of the kite control pod: acceleration, orientation, rotation and altitude are measured with MEMS sensor chips shown in Figure 5.7. They are mounted on a sensor board that is separate to the system controller PCB and is connected to it by an I^2C interface. The details of the sensors used and the achievable resolution and accuracy thereof is discussed below in subsection 5.4.6 when the software interface to these devices is presented.

5.1.4. Ground station circuit

The ground station circuit comprises of a nRF905 module and antenna and a GPS module with integrated antenna. Both of these modules are connected to a computer via USB and use a USB-serial converter to interface with their CMOS level serial signals.

These circuits are mounted in a polycarbonate enclosure with the 866MHz antenna extending out of it. The addition of an anemometer and wind vane to measure wind velocity would be very useful but during the testing the wind speed and direction are recorded manually.

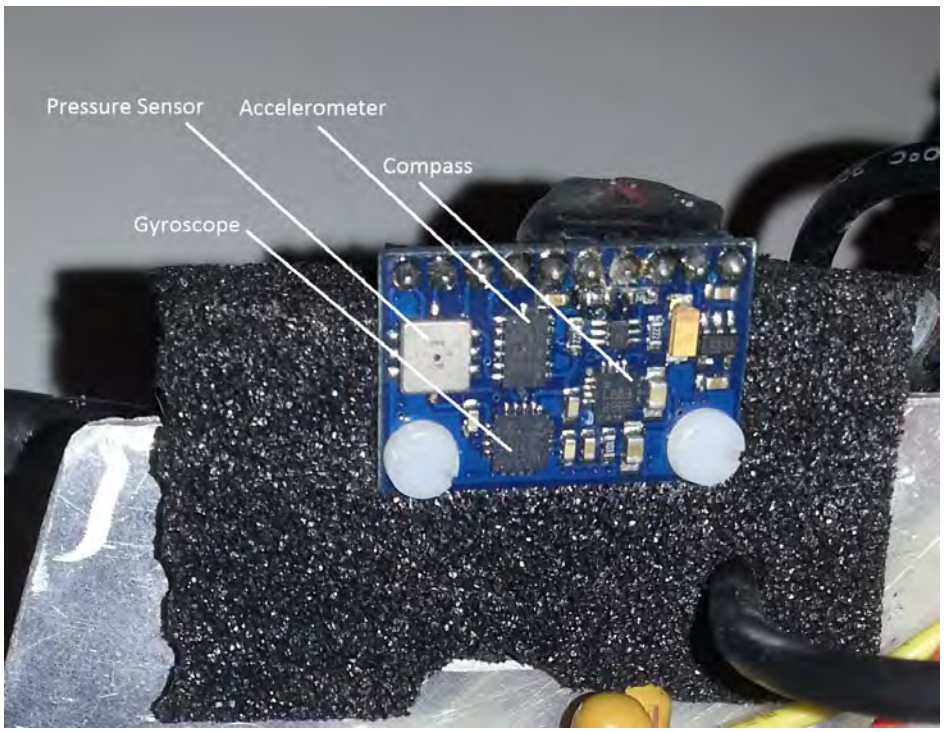


Figure 5.7.: The motion sensor circuit mounted on foam

5.2. Electro-mechanical components

5.2.1. Actuators

The actuators, as discussed previously are components in the window lift assembly from a car door. The ones used came from a scrapped *Smart* car and were acquired as full assemblies with track for the window and two cables coming out of each winch. The winches were disassembled and the required components removed from them. The components used are: the motor, the winch and gearbox assembly, one of the cables and cable guides and the winch housing. the opening for the other cable was blocked with glue to avoid ingress of sand or water into the winch.

5.2.1.1. Winch modification for position sensing

The cable winch spools are in an enclosed cavity, meaning that the measurement of the position of the spool must either be done inside the housing or a linkage must be added to bring the position to the external sensor. The latter is the approach taken as the multi-turn potentiometer is too large to be mounted internally. The winch spool rotates on a fixed shaft so a sleeve has been machined to couple with the spool on the inside and rotate on the same shaft. The hole in the housing had to be bored out to make space for the sleeve and a bracket was made to hold the potentiometer in line with the

(a) Modified winch housing and spool **(b)** Modified potentiometer and bracket

Figure 5.8.: Photographs of modifications made to allow for position sensing

shaft as shown in Figure 5.8. The potentiometer is also modified with a slot cut into the end of the shaft. This slot fits over the bolt, added to the middle of the sleeve providing a rotational coupling but still allowing for some misalignment or movement.

5.2.2. Battery

The battery is an electrical component but its mechanical aspects are important to the construction of the system. The battery consists of four Li-ion pouch cells they are stacked together and connected in series. The cell to cell connections are made by directly soldering one cell's positive terminal to the next cells negative terminal. The most positive and most negative terminals are exposed to supply power to the rest of the system. A secondary, 5 terminal connector is included with one wire soldered to each of the cell terminals (3 common and 2 at ends) is used to attach the cell balancing battery charger.

A fuse holder has been soldered onto the positive terminal and fitted with a 30A fuse. The other side of the fuse and the negative terminal have female pin terminals soldered to them to be easily connected to the power distribution board that has the mating terminals on its leads. The whole battery has been enclosed in a plastic coated foam casing and is mounted on a bracket that is bolted onto the base plate on the opposite side to where the motors protrude.

5.2.3. Wind power generator

A small motor and propeller are mounted on the end of the tail. The motor is a permanent magnet 3-phase synchronous type so a 3phase rectifier and buck-boost converter is used

to connect it to a charging circuit for the batteries. The propeller has two blades and a blade length of 120mm and so a swept area of 0.05m^2. The motor has 12 poles which increases the output frequency by a factor of four and can be directly connected to the propeller without a gearbox. This has not been included in the initial field tests as the tail would obstruct ground handling.

5.2.4. Safety release mechanism

The safety mechanism is mounted at the point where the tether connects to the KCU. It is made up of a slot and a clasp that can be rotated out of the way to release the line. A secondary line, attached to the tether, skips the KCU and attaches further up the traction lines of the kite. If the release is activated, the leading edge lines are shortened by 2m and the kite is completely de-powered. The clasp is actuated by a servomotor, connected to it by a linkage so that the servomotor can be mounted in a more convenient and less exposed location. This module would complicate initial field trials so has been built but not installed for testing.

5.2.5. Wiring harnesses

There are 12 wires that have to be routed between modules. Of that 3 are power cables, one RS485 bus, two position sensor wires, two motor leads. IMU wire, GPS wire, Radio Controller wire, telemetry antenna cable, the connectors are glued in place for most of these while some have clips to keep them connected. They are mostly point-to-point routed and held down with glue and cable-ties where needed but not in a way that would hinder assembly or disassembly.

5.3. Mechanical aspects

Optimising the mechanical design of the system is important for several reasons. Firstly, as the system must be light and compact, careful consideration has been given to ensuring all of the components fit. Secondly this can directly affect the ease or difficulty with which the components are manufactured and the system assembled. Finally this optimisation can help reduce mechanical or electrical complexity, leading to a more robust and possibly cheaper system.

5.3.1. Base structure and layout

The system is designed to allow the traction force of the kite to be transmitted through a central plate, forming the core structure of the controller and down the tether line. The control actuators are to be symmetrically placed around this structure. The layout

of the system is influenced by the directions of the cables and ensuring the centre of mass remains at the centre of the pod. The motors are counterbalanced by the battery mounted to a bracket, bolting onto the baseplate. The other components are either mounted directly to this plate or onto brackets that are bolted to it.

5.3.2. Packaging and assembly

A side view of the device as assembled is shown in Figure 5.9. In this image, the protective cover on the side is removed for visibility of the system components.

Motor drive mounting The motor drives are mounted on either side of the base plate with the motor output side close to the motor terminals to get short motor leads.

System controller bracket The system controller is the largest board in the KCU as there is little spare space on the base plate, it is attached using a bracket and mounted in the space below the motors that protrude out from the cable winch assemblies.

Power board placement The power distribution board is mounted on the lower part of the base plate near the battery terminals, has the power harness soldered to it and once it is mounted, this is routed up past the motors to the drives and over to the system controller.

Battery mounting The battery is mounted on its own bracket on the far side of the baseplate to the motors. This placement is in part, to counterbalance the motors mass to align the centre of mass of the KCU with the line going from the tether attachment point to the leading edge lines' attachment point at the top of the KCU.

Position sensor brackets The position sensors are mounted onto the outside of the cable winches to keep them in line with the axis of the spool.

5.3.3. Outer housing

The housing of the KCU is supported by two bent aluminium plates 1.6 mm thick that, covering the bottom and top, extend up and down respectively and bolt onto the brackets included in the actuator assemblies. Four wooden bars are screwed into the corners between these top and bottom plates. Closed cell foam is inserted between the wooden blocks on all four sides with holes and cutaways where necessary to accommodate components and an antenna. This foam is present to absorb loading and avoid the bending of the position sensors or their brackets in case of impact. The foam is covered

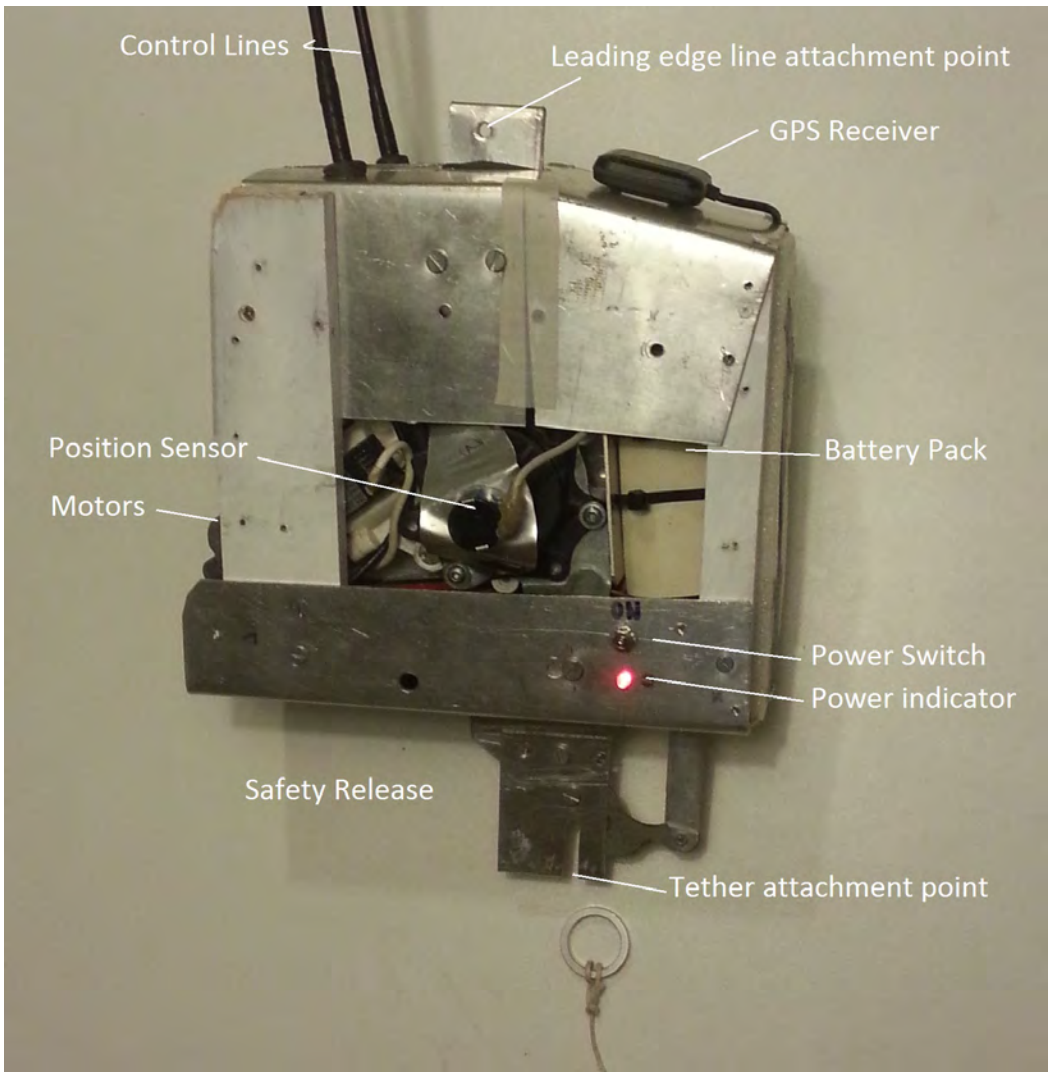


Figure 5.9.: Photograph of packaging of the system

with thin aluminium sheeting, secured onto the wooden bars with self tapping screws. This housing fully encloses the electronics and other assemblies and protects them from sand and water splashes but not fine dust and immersion.

The front and rear covers are made of perspex and the telemetry and radio controller receiver antennas are mounted so under them so that the signal will not be blocked. This also ensures that these antennas are more protected from impact and damage than if they protruded from the casing. The power switch and power indication LED are mounted into the lower cover in a position that is accessible but reasonably protected from damage due to impact by the bulged side protection covers.

5.3.4. Attachment to the kite and tether

The interfaces between the controller and the kite lines is the control bar. The KCU is attached to the ends of the bar with short ties at the end of the actuator cables. The leading edge lines go through the middle of the bar and to a loop that is where the majority of the kite lift force is transmitted. This loop is tied onto an attachment point on the top of the KCU.

The tether is attached to a ring that is clipped into the safety release mechanism on the lower side of the pod, from there a slack piece of the line continues past the pod and is attached to a point on the leading edge lines of the kite, 1.5m from the pod. When the safety release mechanism is activated, the tension in the tether is transmitted straight to the kite, reducing the angle of attack of the kite past the point of zero lift so that the tension in the tether is released. During initial testing, the sandbag is tied directly to the baseplate holes that would be used to bolt the safety release on.

5.4. Main controller embedded software

The software of the main controller coordinates the communication between the various parts of the system. It also proved useful as a tool for testing and experimentation during the development of the components that connect through it before the system had been fully integrated.

5.4.1. Real-time operating system

The implementation of *FreeRTOS* facilitated the separation of the functions of the system controller into separate tasks. Each task, running much like a thread in a fully fledged operating system has been developed to be semi-independent. This modularity of functions decouples the timing of each function from that of all of the others, relying on the task scheduling system of the RTOS to execute whichever code needs to be run at that moment. Interprocess communication is enabled by the use of semaphores, protecting the data structures from race-conditions regarding reading and writing values. *FreeRTOS* requires the inclusion of three source files and one header file into the project and can be downloaded and used under a modified GPL licence. The header file is used for configuration such as the heap size, memory management method employed and the time interval between ticks. It requires 4kB of flash and minimal RAM for its functioning with the majority of the heap being used for tasks stacks. The heap size is set to 4096 bytes.

Task set-up The tasks are created with medium priority so they will be scheduled in order with no single task overriding the others. Each task is given a stack of 256 bytes

and the *FreeRTOS* documentation states that they each use an additional 74 bytes for task scheduling.

5.4.2. RC controller input interpretation

The software developed for the interpretation of the RC controller input is listed in subsubsection A.4.1.4 and works as follows. The PWM signal from the RC receiver triggers an interrupt on both edges of each signal. On each interrupt, a timer value is recorded and the timer restarted. These values are used by a state machine to calculate the on-times of the two signals as they are proportional to the left-right and up-down positions of the control stick of the transmitter.

The *radio_control_main_service* is run as a task and keeps track of the highest and lowest counts measured for a particular channel and the calculation returns a percentage value that is a linear interpolation as a percentage of the latest measured value between this maximum and minimum. When the device starts up, the control stick must be moved to the extremes of each axis to set these bounds as this allows the inconsistencies of the controller to be factored out.

5.4.3. Drive node communication

The system controller is connected to the motor drives by a RS485 bus and the communication with them is handled by the *bus_master_main_service* function as listed in subsubsection A.4.1.2. This task implements the master state machine of the communication protocol. The message structure of the protocol and exact payload is defined in Appendix Table A.4 and the packets to and from the node are 8-bytes long. This task polls each of the slave nodes on the bus once every 50 milliseconds to send commands including motor set-points and control parameters and the slave responds with its measured position and current through the motors.

Every 25ms, a timer sets a flag, initiating this communication. A message for a specific motor node is composed and a DMA transfer is started to send the data out over the serial port. As the slave replies, each byte triggers a serial port interrupt which packs the bytes into the reply message and puts that message on a queue for processing once it is complete. The reply data is then copied out into the slave's data structure and the task waits for the next timer trigger. If there is no response from the slave node, the status byte of that slave's data structure has a bit raised to indicate a communication error. The node data structure gets transferred to the ground station via the telemetry link for logging and display to an operator.

5.4.4. Telemetry

The telemetry task handles the receipt of messages from the ground station and replying with data from the control pod. Each message has a constant length and after a start

of frame character (0xFF) is packed into a receiving buffer from the receive interrupt on UART1. Once the message has been received it is put onto a queue for processing by the task. The received CRC8 is compared to the one calculated from the other received bytes.

The packet structure of the protocol which is listed in subsection A.2.1 defines one message that is repeatedly sent with a period of 10Hz from the ground station. Section subsection A.2.2 defines two messages to be sent from the controller . The first is a standard reply to the ground-station message and the other is appended on after the reply once every ten messages so is normally sent at 1Hz. The data is sent in packets of 31 bytes each as the nRF905 transceiver modules are optimised for throughput with this packet size.

The data being sent to the controller includes updated limits for the motor positions, optional set-points for the motor positions and speeds among other configuration data.

5.4.5. Message integrity

The telemetry and node communication protocols both use a CRC for message integrity verification. The CRC-8 algorithm with a polynomial of $P = X^8 + X^2 + X$ has been shown to be good at detecting single and double bit errors in messages shorter than 32 bytes with undetected errors having a probability of less than 10^{-11} [61]. Thus extra processing overhead incurred in calculating a CRC checksum over a 1's complement addition is deemed acceptable for a reliable message reception.

5.4.6. Sensors

The *run_sensors* function, listed in subsubsection A.4.1.6 is invoked by the start task function of the RTOS. There are four sensors connected to the system controller over an I^2C communication bus . This task also triggers ADC readings of the battery pack voltage.

The I^2C devices are:

Accelerometer ADXL 345 is a triple axis accelerometer made by *Freescale* with a range of up to $\pm$16G and a resolution of 4 mG per LSB and a device address of 0x53. The data from this sensor is integrated to determine the motion of the control pod.

Barometer BMP085 is a MEMS barometer, made by *Bosch Sensortech* with a pressure range of 300 to 1100 hPa (~9000m above, to ~500m below sea level) and temperature sensor with an absolute accuracy of $\pm$1°C and a resolution of 0.1°C and a device address of 0x77. This is used to measure the altitude of the pod. It is accurate to 0.5m of altitude relative to the altitude of the launch site.

Gyroscope L3G4200D is triple axis gyroscope, made by *STMicroelectronics* with a range of up to 2000 degrees per second and a resolution of 16 bits and a device address of 0x69. The gyroscope is used to measure yaw rotation to determine the spinning of the pod.

Magnetometer HMC5883L A triple axis magnetometer made by *Honeywell* for low-field magnetic sensing and compass application and a device address of 0x1E. The 12-bit ADC enables a 1° to 2° compass heading accuracy. When the data from the triple axis magnetometer, is fused with that from the triple axis accelerometer, both the acceleration of the device and its orientation can be determined.

A timer trigger sets of the I^2C and ADC measurements. When each of these measurements are complete, its data is stored and the next reading is started. The task handles the calculations that are required to convert the raw values into engineering units and to determine the derived units such as altitude from pressure.

5.4.7. GPS

The GPS is connected to UART 2 with a baud rate of 9600 and the receive interrupt is triggered each time a byte is received from the GPS receiver. The receiver transmits standard NMEA 0183 protocol sentences to the controller over the serial connection. The message that contains comma separated ASCII values for the latitude, longitude, altitude and time data that the controller is listening for starts with $GPGGA and ends with a newline character. Once this message is received, the text is parsed to determine the integer values which are stored in the GPS data structure for transmission to the ground station. The code to perform this function is listed in subsubsection A.4.1.5.

5.5. Motor controller embedded software

The software that has been written to implement the functions of the motor controllers, listed in subsection A.4.2 is less complex than the main controller software and can be almost completely interrupt driven. Figure 5.10 depicts the logical flow of the different processes in response to polling two flags in the main loop or in response to hardware interrupts.

The control law used in this software is shown in 5.11a below and was developed around a level digitizer shown in 5.11b

This control loop has been implemented in the microcontroller software to allow for further control design if necessary.

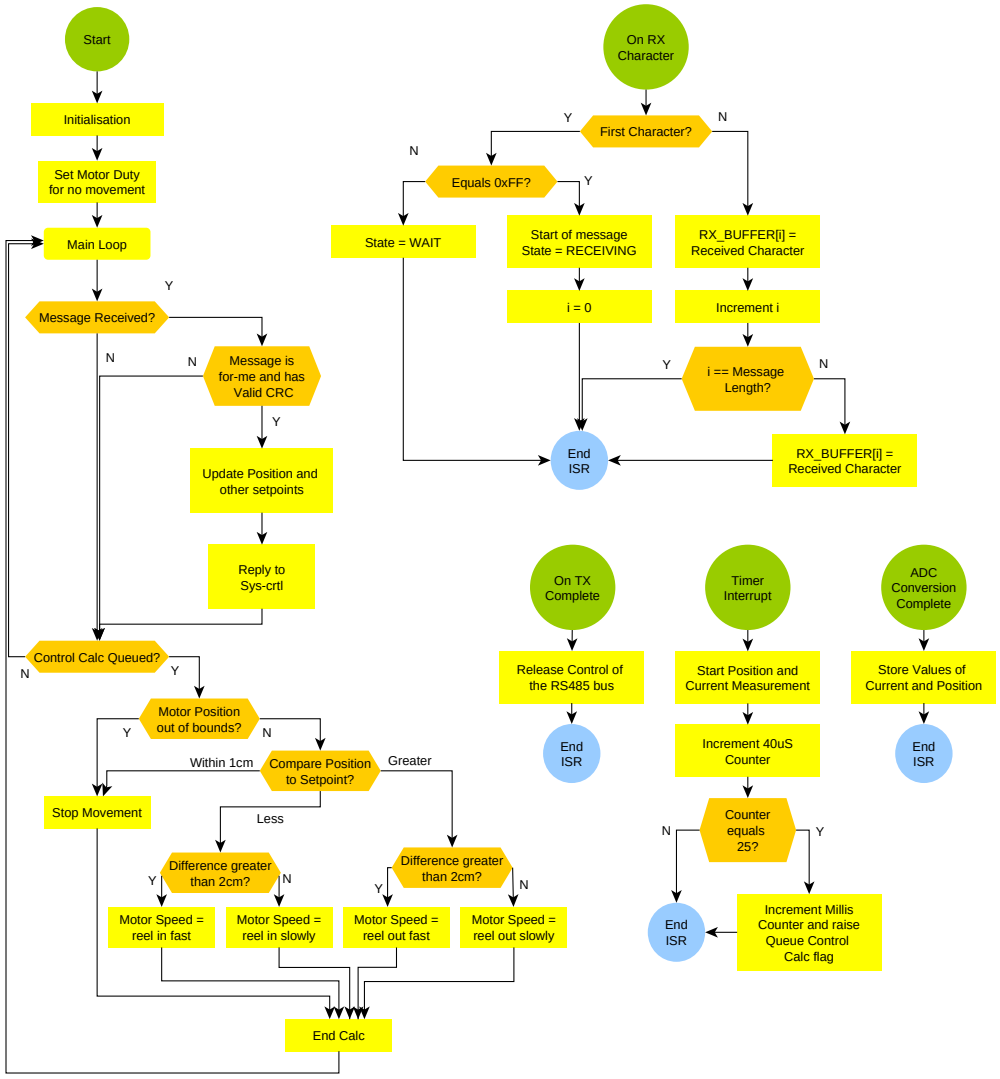


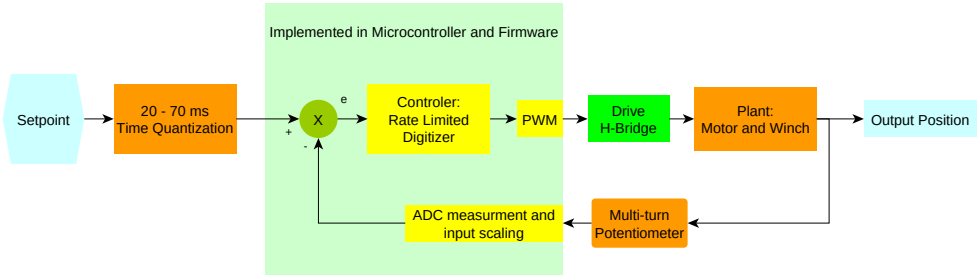
Figure 5.10.: Flow diagram of motor node software

5.5.1. PWM generation for drive switches

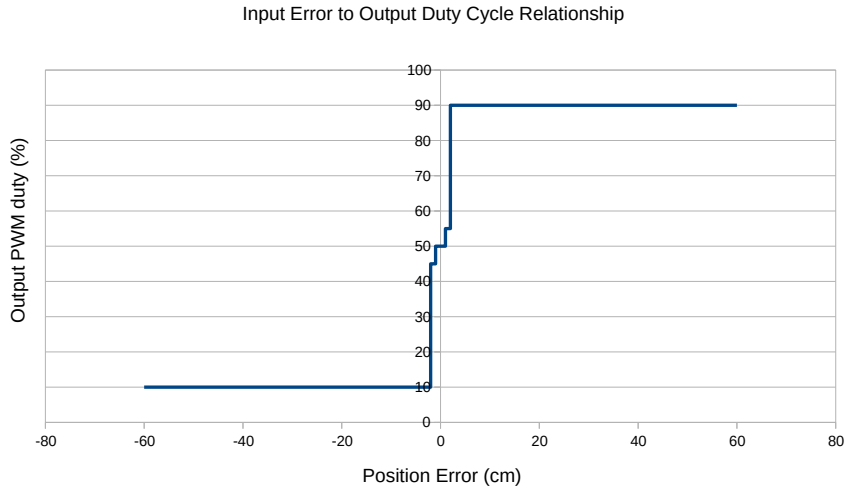
The four signals that drive the switches of the motor drive are connected to pins 18 to 21 as shown in the schematic snippet in Figure 5.1

Pins 19 and 21 drive the low side switches and 18 and 20 drive the high side switches. Care must be taken to avoid both switches of either half-bridge being on at the same time as this would short the supply bus. This function was carefully written and thoroughly tested before the rest of the development could continue.

The switching regime chosen is that of a bi-directional buck converter rather than unipolar or bipolar switching as this is simpler to implement with the hardware available on the microcontroller and because this essentially stops the switching when there is no



(a) Control loop diagram



(b) Input error to duty cycle

Figure 5.11.: Control Loop Diagram

duty input, saving power.

5.5.2. Interrupt sources

The microcontroller hardware in use has been set up to call functions in the code when certain events are detected. The following are implemented:

- Timer interrupt every 40µS - The PWM output is updated and a counter is incremented for the millisecond tick.
- ADC conversion complete - The measured value is stored.
- Character Received on RS485 - The receive state machine packs these into a message buffer for processing once the whole message is received.
- Transmission complete on RS485 - The bus communication direction is set from transmit back to receive

5.5.3. Message received

Each commands message from the host, defined in Table A.4, contains updated set-points, position limits and a drive mode setting. The receipt of the message triggers an interrupt for each byte which is packed into a receive array by a state machine as listed in subsubsection A.4.2.2. Once the message is received and verified using a CRC8 algorithm as described in subsection 5.4.5 above and the recipient address of the message is compared to the hardware address of the node. If there is a match, the message is used to update the relevant variables, otherwise it is discarded.

5.5.4. Node reply message

On receipt of a message addressed to a particular node, a reply is sent to the controller consisting of the node's status, the motor position and the measured motor current, the structure of which is given in Table A.5

5.5.5. Feedback control calculation

The position control of the actuators is implemented in a feedback control loop. The position set-point is updated by the main controller over the communication bus and the current position is measured using an ADC connected to the multi-turn potentiometer. The DC motors are linear in most of their operating range in each direction with a discontinuity around the zero velocity point as determined from the test results in subsection 6.1.1. The whole winch assembly is highly non-linear however due to the difference in the forces required for retraction and extension of the cable. A further dead-band, associated with a small position error was introduced to save power when little movement is required. Lastly, the control loop is simplified by the worm-gear because the motor stops the cable movement when there is no voltage applied to the motor. This is implemented as listed in subsubsection A.4.2.2 as a proportional bang-bang controller, where large errors (greater than $\pm$30mm) cause fast movement in the direction of decreasing error, small errors (greater than $\pm$6mm) cause slower movement and the motor is left off for errors less than $\pm$6mm. Checks are done for motor position out-of-bounds which could be caused by a mechanical issue where the winch is over-wound or set-up incorrectly or a wiring issue where the position is read as 0. The motor is stopped if any of these checks fail to avoid further damage.

This control method performed well both in laboratory testing and field trials and a maximum rate limit was added later as a setting sent from the ground station software.

5.6. Ground station software

The software on the ground station computer has been designed to provide information on the flight of the kite and storage of the data transmitted from the kite. The Java language was chosen for this application over the original test program which was written in LabView due to licensing issues encountered with the latter. Extra code libraries were included for graphing and serial port communication to make Java viable for this purpose. Python would have also been suitable but has a less able graphic user interface (GUI) development environment.

5.6.1. Code functions

As the ground station software design centres on the GUI, providing feedback to the user and receiving commands, a class handling the GUI and the events associated with it is the start point for the program. Separate functions have been modularised as much as possible and the program consists of the following classes :

5.6.1.1. KiteGroundStation

This is the program entry point, it instantiates the GUI which has been built using Java. It also instantiates the telemetry class for communication over this, the logging class for writing of test data to file and several LineGraph objects for some of the data display work. These other classes return data to this class when available for painting when the GUI updates. The code is listed in subsection A.3.1

5.6.1.2. Telemetry

This class instantiates a SerialPort object and sets it up with a baud rate of 19200 for communication with the serial port and attaches a listener to the interrupt generated by the receipt of a byte on this port shown in subsection A.3.2. A timer is instantiated to generate an interrupt every 100ms and while a sending boolean variable is set to true, a message is packed and sent on each of these interrupts. The message, whose protocol is defined in subsection A.2.1 is 31 bytes long and if received correctly by the KCU, triggers a response message. As each byte of this response are received, the receive interrupt calls the Telemetry class' event handler which packs it into a message. Once the whole message has been received, a CRC8 check is done on it and if it passes, the data is packed into the KiteData class for use in the KiteGroundStation class.

5.6.1.3. Data logging class

All of the data that is received over the telemetry link is pushed into a queue by the main class for storage to disk by the logging class listed in subsection A.3.5. This queue

which is 120 items long, allows for data to be buffered for several seconds between each write operation, reducing load on the hard-drive. Once the queue/buffer has filled up past 60 items, it is flushed to disk and stored as a CSV file. One file is created for each test and starts with a header used for test notes and other static data.

5.6.1.4. CRC-8 class

The Checksum verification is done for each received message in the telemetry class and a CRC8 byte is attached to each transmitted message. This class was adapted from *LibFLAC*[62] and the CRC8 C algorithm used in the embedded code and it uses the same polynomial. It uses a table-lookup method of CRC calculation .

5.6.1.5. LineGraph class

Having good graphing functionality is important to a live display of the flight data so along with choosing Java as the language with which to write this program, a suitable graphing tool had to be found or made .

JFreeChart is a free and open-source library for Java that provides a variety of data visualisations such as graphs and dials. The LineGraph class has been written as a wrapper for the library, tying together much of the customisation needed in using the *JFreeChart* line graph class.

5.6.1.6. Second GPS receiver

A second GPS receiver is connected to the ground station computer to provide a reference point for comparison with the moving point of the kite and pod. This receiver also transmits *NMEA-0183* GPS messages processed as shown in subsection A.3.6 and has been included as a fixed-position receiver can be used to offset the drift error that can occur in the receiver moving with the kite control unit.

5.6.2. Data display

Beyond the initial testing where the KCU is several meters above the ground, the kite and KCU may be too far for the operator to see all the details of the flight needed to control it properly so the sensor data must be displayed in a manner useful to the operator.

Kite position The display of the 3D position on a 2D screen poses certain challenges, a 3D rendering would look accurate but possibly provide little precision as to the position of the kite. A better method would be to provide two cross sections, one facing downwind or looking from above, the other from the side, showing tether length and altitude angle and altitude. The altitude is also displayed as a numeric value in meters.

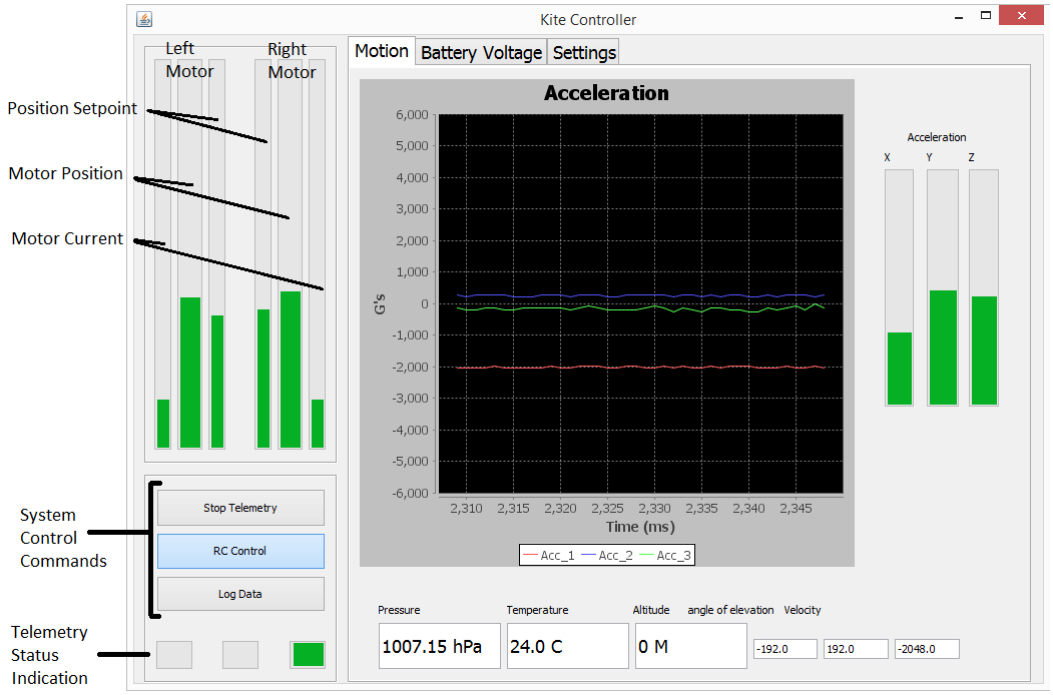


Figure 5.12.: Screen-shot of ground station application

Actuator movement The position set-points of the controller and the resulting movement of the actuators are plotted on a set of bar-graphs so that the set-point tracking is clearly visible and the kite steering and power input can be seen.

Force on the control winches The motor current in each of the motors is proportional to control line tension during reel-in of the line, this current is displayed alongside the motor position bars.

Battery voltage The battery voltage of the KCU is plotted on a line graph continuously during the flight and can be monitored to see how quickly the battery is discharging.

Raw telemetry data One of the tabs in the application main screen displays an annotated listing of the raw data received from the kite. This was used during early testing and debugging of the telemetry link.

6. Testing

To verify the prototype design and manufacturing, the subsystems' functions are measured and compared to the design's specified values. If the subsystems perform sufficiently well then the system is integrated and tested as a whole. This being the measurable result of the work of the previous chapters shows the system functioning as required and gives values to some of the predicted performance figures. The lab tests are thus part of the development and the field tests show the whole system's performance and thus these tests perform complementary functions in the the development and prototyping of this system.

6.1. Sub-component performance characterization

The subsystems are to be tested individually. This allows for function to be split up and tested separately, ignoring inter-module effects till the system is tested as a whole. This allows for means by which the inter-module effects can be measured (deviation from a previously attained performance). An example of this behaviour would be the reduction in bus voltage due to battery internal resistance when under load of the motors versus the motors being tested using a bench power supply as source. The measurements produced by the IMU and the GPS were shown to be valid but were not extensively tested during this process but the raw values were recorded as part of the flight testing.

6.1.1. Actuators

The actuators including motors, worm gear and spool are tested to determine the voltage versus no load speed relationship and the current versus tension relationship.The former will provide an indication as to the maximum travel speed that can be expected from these motors given the battery voltage used in the system. The latter, to determine how much current will be required to actuate under a variety of loads. The results of these tests are useful in setting reasonable values for the loading of other components in the system, given measurements and certain assumptions regarding of the actuation speeds and forces required to control a kite.

Test set-up The velocity versus applied voltage is done for the whole travel of the spooled cable which is 600mm. The cable is fully extended and a voltage, from 2V to

20V is applied while the travel time is measured with a stop watch. The travel time is converted to travel velocity by $V = \frac{0.6}{t}$ in m/s.

The current test is done by mounting the winch assembly (or holding it in a vice) with the cable guide pointing straight down. The end of the cable is then attached to a bucket which is progressively filled with water, to provide a constant mass. The resulting current load while lifting and lowering the bucket is recorded. The force in Newtons can be derived from the equation $F = m.G$ where G = 9.81(N/kg). For each run, the current in the motor is increased till the winch starts to move, the current is then recorded and mass is increased.

Results When the motor had no load, travel speed of the actuator was recorded as shown in Figure 6.1 and it follows that the speed increases at roughly 16mm/s per volt applied.

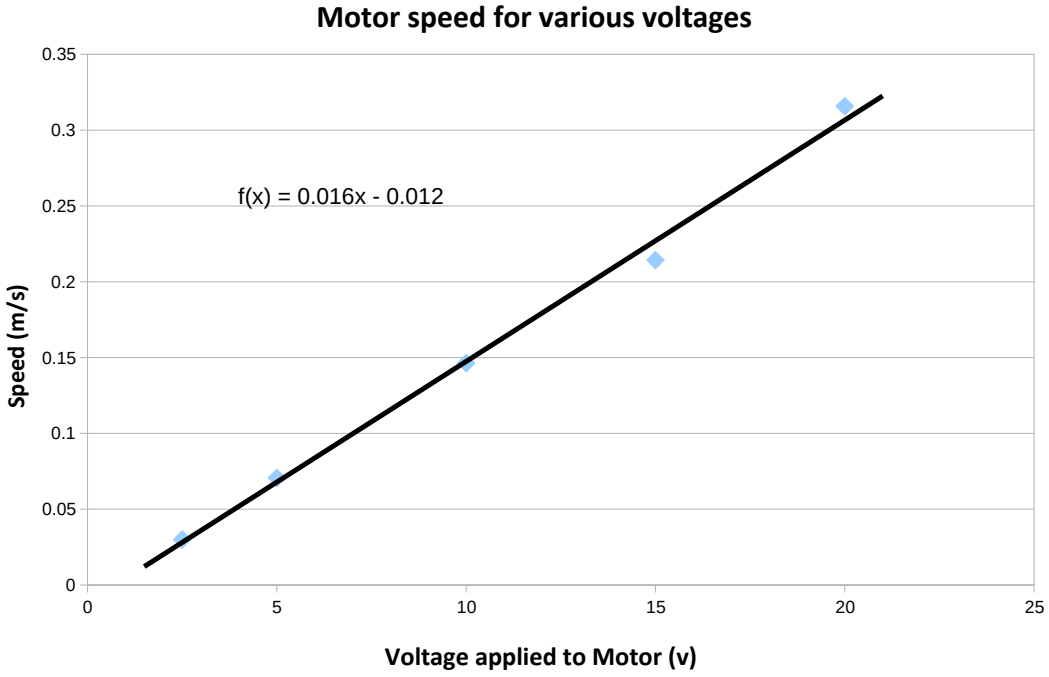


Figure 6.1.: Speed of retraction versus voltage applied

As the static load force on the motors was increased, the current required to retract the winch against this force was measured and the results are presented in Figure 6.2 for both motors. The linear interpolation of the results for both motors were almost exactly collinear and besides an offset of 1.3A, for each 10N increase in force, the current required increases by 0.41A. Therefore the full force of 150N from subsection 3.2.1 would require $I = 0.41(150/9.81) + 1.3 = 7.57A$. The data presented for these tests are a single set of measurements with a linear regression line overlaid, further testing would improve the trustworthyness of the measurements made but is left for further investigation.

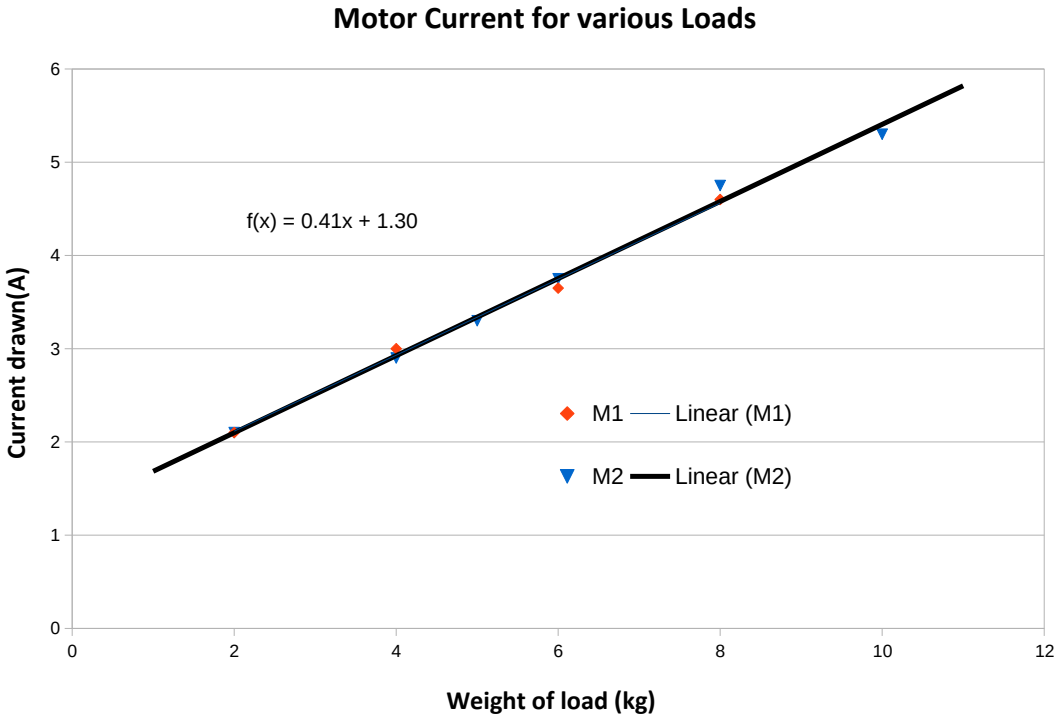


Figure 6.2.: Current required for different retraction forces

6.1.2. Motor controllers

The motor controllers are tested by being connected to an actuator, pulling on a dummy load. This test is to verify the performance of the motor controller designed, over the full range of the load. This is primarily to check that high currents in the controller do not cause it to overheat or give rise to functional issues such as noise being coupled to parts of the circuit that may cause unintended behaviour.

Test set-up The actuator is set up in the same way as when its current test was being performed, including the bucket. The motor controller is commanded to raise and lower the bucket 5 times for 10kg load. The circuits are to be powered from a bench power supply to eliminate the effect of the battery voltage dipping under load. This will test the motors and drives together as it is a more representative test than testing the drives on a resistive load.

Results The thermal image of the board at the end of the cycle is used to check for hotspots on the circuit board. Component over-temperature is a sign of an error in the circuit design so these thermal images can be useful in diagnosing design issues and verifying power dissipation . The area in red in Figure 6.3 is the part of the board

with the MOSFET drivers and the voltage regulators on the lower side. Both of these contribute to a part of the board reaching 45°C which is hotter than the rest of the board but acceptable during operation. It does indicate that the power consumption of the circuit can be reduced by reducing the drive current to the MOSFETs.

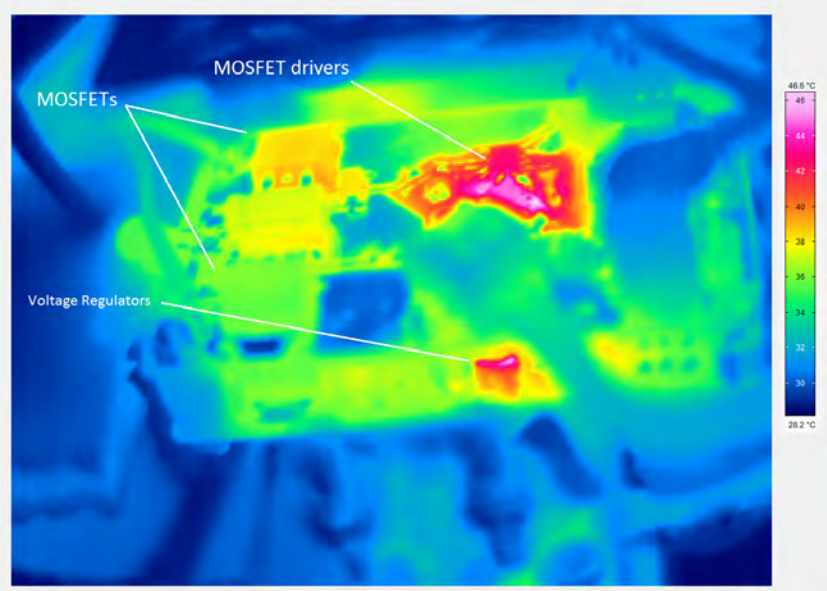


Figure 6.3.: Thermal image of motor drive under load

6.1.3. Battery energy capacity and power delivery

The current and energy capacity of the battery, although not directly specified, can be useful in determining the expected flight time with this battery. Since it is impossible to predict the exact load profile of the KCU, the flight time specification will have to be demonstrated in tests with a typical loading pattern.

Test set-up The fully charged and rested battery is connected to the controllable load and discharged to empty in half an hour. The cell voltages will be manually measured during the test and the battery will be considered flat when there is 3V across the most discharged cell.

Results From 4.2V per cell resting, the battery was connected to a load and discharged at 8A, i.e. to drain the battery in half an hour. The test was stopped when the first cell reached 3V. At this point the battery had delivered 4.1Ah.

The battery was rested and the open circuit voltage was 3.27V, 3.57V, 3.42V and 3.39V after 5 minutes. A load of 20A was then applied to the battery and it maintained it for

5 seconds before the weakest cell reached 3V. The battery was rested for a further 5 minutes and a current of 10A was loaded which it maintained for 15 seconds.

This test shows that this battery pack meets its specified capacity of 4Ah and is capable of producing bursts of higher current, even when almost completely flat.

6.1.4. Telemetry and control link

Before flying the kite and KCU system, a reasonable measure of antenna function is required for both the telemetry and control radio links. There is no received signal strength indication (RSSI) information provided by either of the radios so no acceptable figure can be defined. The test then will only check that the radios operate as expected up to a distance along the ground of at least double that of the maximum tether length used during flight. This is to be tested at several different antenna angles.

An initial test of 100m is to be performed before system integration as a sanity check on the components used with greater distances tested as the flight envelope is increased.

Test set-up A minimum set of data must be looped back through the telemetry link and the RC receiver connected to servomotors if the system is not in a functional state, otherwise the as-built KCU can be used. At the determined distance between the KCU and ground station, proper function must be maintained through a rotation of each of the antennas in as many combinations make sense but at least in all six directions (placing the pod with each of its sides facing the remote site in turn) for a variety of orientations of the other unit.

Results The telemetry communication was maintained at a distance of 50m without noticeable decline in throughput. The servomotors continued to be controlled by the RC controller as well during this test.

Paced out in several different orientations, the communication starts to be interrupted regularly at about 75m in some orientations and 100m in others. This is sufficient for initial testing but would need to be replaced when longer, higher altitude tests are performed. This is much less than calculated so the cause for signal degradation should be found before further development is done. A different telemetry transceiver with a signal strength indication would also enhance this testing.

6.1.5. Safety release effectiveness

As the safety release mechanism is not used during the normal operation of the KCU it must be shown to function correctly even at the maximum tension expected on the leading edge lines. It is also important to verify that there is no bending or warping of the components at this load. The maximum test load is 150kg as this is well above what is expected during testing but possible in the field.

Test set-up The controller base plate with safety release and servo mounted are fixed in place on a test bench. A spring gauge is used to measure the force applied but as the gauge only measures to 25kg, the line is looped 4 times in a pulley arrangement. A load is applied to the other side of the pulley until the gauge reads 10kg and the safety release mechanism is engaged. The test is repeated for a reading of 20kg and 25kg. The test is passed if the safety release mechanism manages to release the link in all of the tests and there is no noticeable deformation of the components of the mechanism.

Note This test was not done as it required the integration of the servomotor into the body of the KCU which was not done before field trials. The safety release mechanism was not included in the build that was tested as a whole as robustness and compactness was emphasises and another method of safety de-power could be effectively employed.

6.2. Integration tests

Once the components had been tested individually, the system was built up progressively with additional components and software integrated for each new test. Much of the software development was done during this iterative testing phase and was checked to be working before each set of new features were added.

6.2.1. Battery-controller-motor

As part of the system integration, larger and larger parts of the full system are connected and tested together. The objective of the test is to ensure that the loads required can be managed on battery power.

Test set-up Is the same as for the motor drive test but now with the battery supplying the power. The actuator is driven to lift and lower a 10kg bucket 5 times. The test is then repeated with both actuators and controllers present and two 10kg buckets.

Results When this was first connected, the inrush current damaged some of the MOSFETs in the drive. This gave rise to the addition of the inrush current-limiting power distribution board.

The boards were then repaired and the test passes with a current of 15A being drawn during the lifting of both 10kg buckets.

6.2.2. Control via telemetry and RC controller

Once the software in the system controller and ground station application had reached a level of functionality where commands could be sent from the ground station application to the system controller to override the speed and position control set-points. The RC controller is then used to input the position set-points, showing the functioning of the auto-limit-set and the rest of the interactions. The position control latency is also to be noted.

Test set-up The KCU is suspended by its leading edge line connection point from a mounting on the wall or roof and the actuator cables are attached to bungee cords or an elastic, fixed to the same point. The tether mount point is attached to a bench or weight to stop the set-up from swinging. This test set-up was left assembled for the duration of the embedded development period and used to verify the functioning of each new firmware update.

Results The system responded consistently to the control inputs from the computer, both in speed and position control mode. This test was performed numerous times while the system software was being developed and done at least once before each software release for field testing. The data displayed in Figure 6.4 compares the control set-point (blue line) to the measured position of the controller (orange line). The absolute position tracking is good and the travel time is approximately three seconds for large transients.

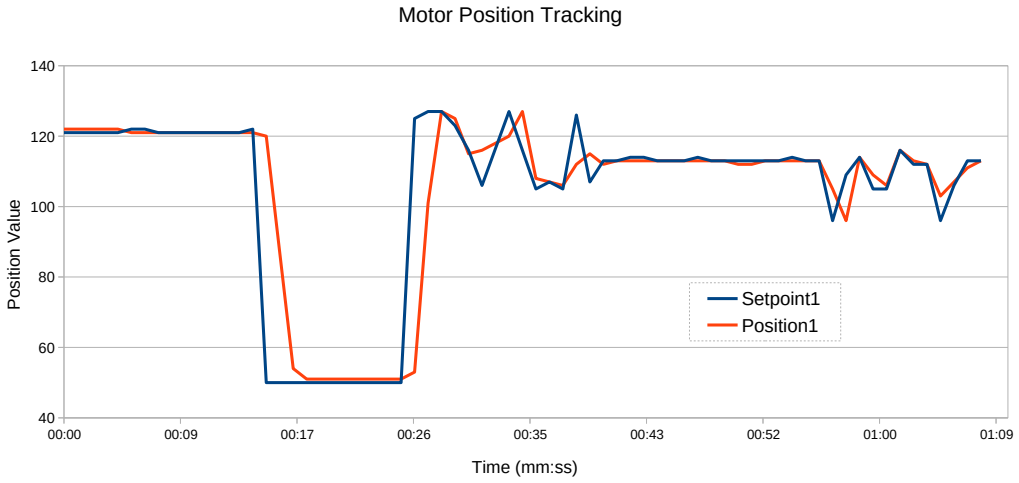


Figure 6.4.: Set-point tracking of the position controller

6.3. Field tests

The testing in the field is crucial to this work and constitutes a fulfilment of some of the requirements as stated in the scope of this study. As this will be outside of the facilities of a lab, careful considerations of equipment to be included as well as plans to mitigate the effects of certain failures are required. In particular as this is the first opportunity for the controller to be tested with an actual power kite, special attention must be paid to the issues of safety and non-damage to equipment. The test shall be performed on a beach, ideally with the wind parallel to the coast and 12-18 knots in speed. The controller shall be tethered to a sandbag with a mass of 50kg. An extra line shall be attached to one of the control lines so that if the kite is released/torn free/overpowered, it can be turned side-on to the wind and landed. As the controller is attached to a sandbag and not a winch, the kite will remain at the same altitude of about 25m for the duration of the test and the controller shall be up to three meters off the ground.

If this product were to be produced in volumes and to be used unattended or by untrained individuals, more rigorous testing would be required.

6.3.1. Ideal weather conditions

Wind speed: 12 - 18 knots (6 - 9 m/s) and with low levels of gustiness. At these wind speeds, the kite will be properly supported by the wind, even in the event of a lull to half the wind speed. The kite will also not be producing more tension than the weight of the anchor at the upper end of this wind range.

Wind direction parallel to beach (or to longest dimension of test area) so that there is lots of unobstructed downwind space.

6.3.2. Equipment required

Kite control unit and RC controller, Laptop, ground-station electronics, kite and lines, tether, sand bag and a hand scale.

6.3.3. Pre-flight checks

Power up the controller and connect to the computer via the telemetry link.

- Low battery detection - ensure that the battery voltage indicates that the state-of-charge of the battery is above 80% i.e. that there will be sufficient energy to perform the test.
- Wait for and verify the GPS lock - that the GPS receiver is determining the device location and has sufficient satellites in view to continue providing position information.

- Zero the barometric altitude calculation. The barometric pressure at ground level is used as a reference to determine the altitude during the flight. This is not crucial for tests with short, fixed-length tethers.
- Check that both control line actuators are moving freely and correctly for the control input. Specifically note if the lengths of the two lines are different when the steering input is in the middle. If so, adjust the offset for this input.
- Activate the safety release system and check that it is actuated correctly and that the green LED on the trigger unit indicating function and sufficient battery is illuminated.

6.3.4. Procedure

At least two people are required for this test, one to control the kite and one to launch/-land the kite and monitor the laptop software.

The ground station anchor point, being a sandbag with at least 50kg of sand in it is filled and the tether is tied between it and the tether attachment point of the KCU.

The kite is inflated and the standard bar-and-lines are attached to it. The cable ends of the actuators of the KCU are attached to the ends of the bar using the ties provided and the 'chicken loop' is attached to the traction point of the KCU body as shown in Figure 6.5.

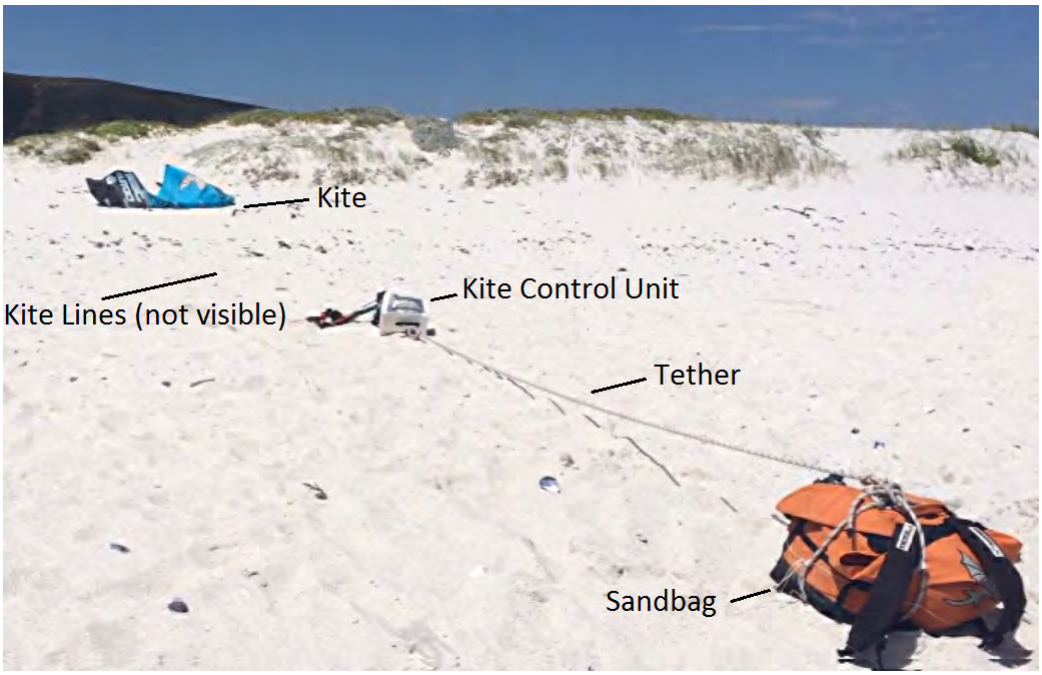


Figure 6.5.: Prepared test set-up

The kite is then held at the edge of the wind window while the chain of control from the input to the kite is checked.

Once the kite is stable and under the control of the operator of the KCU, the assistant releases the kite and it is flown up to the vertical position.

On completion of the intended flight, the kite is flown to the side of the wind window where it is landed by the assistant as a reversal of the launching process. If there is any failure, or other interruption of the test, it is likely that the safety release will be activated or the kite flown into the ground, in either case, the assistant is to retrieve the kite and fasten it against the wind.

6.3.5. Data to be recorded

During the test, data will be recorded by the ground station computer's software. Further information, in the form of test notes are to be taken by the operator.

6.3.5.1. By the test operator

- Take note of weather, wind speed and direction, location and time of day.
- Video footage and photographs are to be taken to be included in the test reporting.
- Wind speed logs either with an anemometer on site or by getting information from a nearby wind metering website.
- Notes are to be taken on the performance of the system, especially, any issues with the KCU and recommendations for improvements thereof.

6.3.5.2. By ground-station software

Data logging of the measurements made by the KCU and ground-station is useful for later analysis of the flight and system performance.

6.4. Results of field tests

6.4.1. Test 1

This test was done in headless mode where there was no ground station computer available to display and store the data. No photographs were taken during this test as the operator and assistant were preoccupied with the flight itself.

Wind 15 - 19 Knots, 17ks Average, SSE direction

Location Sunset beach, Cape Town. Just north of the Bursa Ave. parking lot.

Date 19 January 2014 at 1 PM

Kite Best Guroo 6m^2, standard set-up, lowest power.

6.4.1.1. Results

Launch went as planned, maximum de-power. Once overhead, control was maintained for about 5 seconds before the controller stopped responding. The kite then dragged the controller and 50kg sandbag down the beach before crashing into the sand just to the right of downwind of the controller. No further flights were attempted at the time.

Recovery The kite was recovered immediately (it had filled but was facing downward). The controller suffered only minor impacts with no structural damage. It was however, powered off and upon closer inspection, the bouncing had caused the power connector to become disconnected.

Notes

1. when the bar delta is too high (+25cm) the bar stops sliding on the traction line. The control method must be changed to limit the possibility of this.
2. The safety line shall be attached to the pilot in the future so that they can chase the kite as far as need be but when they stop, the kite spills and lands.

Recommendations

1. Move left control elevation input to right stick and blend the differential and common mode control actuation into a single stick.
2. Glue in the power connector.
3. Ensure there are more assistants to help take photographs of the subsequent tests.

6.4.2. Test 2

This test was also done without telemetry data being displayed and recorded on a laptop as while the software was ready, the laptop running it had been damaged and was not available.

Wind 12- 15 knots, average 14kts, SSE

Location Sunset Beach, Cape Town

Date 29 January 14 at 7:45 pm

Kite Best Guroo 6m^2, standard set-up, lowest power.

Figure 6.6.: Photograph of the second test

6.4.2.1. Results

The control of the kite was much more intuitive than the first test. This is important as there is no force feedback to the operator, they rely on visual feedback alone. Improved controllability resulted in a longer test flight time of about 3 minutes. The wind was light and at times, during lulls, there was too little wind to keep the kite flying properly so it was landed and the test aborted there. It was noted on landing that the cable from one of winches had slipped out of its groove and would have become jammed if the flight had continued.

Cell voltages Before test: all four within 20mV of 4.05V or about 85% state of charge; after test: all within 20mV of 3.85V i.e. approximately 70% SoC.

Notes Controller with the modified control input calculation, using only one stick of the RC controller worked very well. This was the first version of the algorithm that lacked the ability to adjust the steering of the kite when both winches were fully retracted. Kite launched and landed in a controlled manner.

Recommendations

1. Modify winch to avoid cable jamming.
2. Change control method to ensure steering ability at all points of the common mode input range.
3. Improve the crash shielding

6.4.3. Test 3

Figure 6.7.: Photograph of the third test

Wind 18 to 25 knots (9-13m/s) South-south-east

Location Sunset Beach, Cape Town

Date Sunday 11 October 2014 at 15:00

Kite Best Guroo 6m^2, standard setup, lowest power.

Payload ~100kg sand bag.

6.4.3.1. Results

System was controllable and flew for approximately 30 seconds before the traction force generated by a cross-wind manoeuvre caused the tether swivel to snap. The controller and kite then had to be retrieved about 50m down the beach because the secondary de-power line snapped under the shock loading. The data logging laptop was set up but due to poor telemetry signal because of obstacles between the KCU and receiver. The ground station computer was stationed in a vehicle, about 80m from the controller and little data was received for logging.

Recommendations:

1. Remove the swivel from the tether configuration as it is only needed where a long winch-spooled tether is used to avoid twisting.
2. Move ground station into direct line of sight to the kite controller
3. Use a stronger safety line to ensure that the kite will be de-powered properly if there is similar mechanical failure.

6.4.4. Test 4

Figure 6.8.: Photograph of kite and controller in flight during test 4

Wind 12 - 15 knots South-East

Location Derdesteen, 2km north of Big Bay, Cape Town

Date Sunday 9 November 2014 at 14:00h

Kite Best Guroo 6m^2, standard set-up, lowest power.

Payload ~50kg sand bag.

Set-up For this test, data was being recorded but the ground station GPS receiver was not used as it did not provide useful information.

Kite controller outer housing had been rebuilt prior to this test to integrate the antennas into the body of the robot rather than protruding from it. Crash protection has also been improved.

3m tether from 50kg sandbag was used and the separate safety connection to operator was stronger than in previous tests.

Results Successful test with much longer flight time. The test was ended after about 10 minutes of flight as the wind had become more gusty i.e. there was no failure that caused test termination.

Data recording stopped after about four minutes of flight because the laptop that was performing the function went to sleep but this was only discovered during post processing.

The KCU device as a whole weighs 3.85kg which is below the specified 5kg.

Recommendationsset-up

1. Data recording period too low(1Hz) to catch real data bandwidth. Raise to 10Hz and perform some of the low-pass filtering in the software of the KCU for better data representation.
2. Change the settings on the ground station computer to avoid it going to sleep (this was done after this test with a few lines of code in the ground station application)
3. Perform some form of sensor data fusion to provide motion feedback to the ground station.
4. Test using a 10 - 30 m tether

Note, the person with the hat in Figure 6.8 is sitting on the sandbag to increase its weight. It was approximately 50kg and the person, 75kg. There were points during the flight where the person and sandbag were dragged down the beach for several meters.

(a) At launch **(b)** Effect of controller mass

Figure 6.9.: Photographs taken during test 4

Figure 6.9a shows the kite about to be released by the assistant. The controller is also being held of the ground to make the control easier. Figure 6.9b shows the degree to which the controller, as part of the tether, can sag because of its mass when the lift force of the kite is low without causing flight control issues.

6.4.5. Analysis of data from test 4

From data recorded during the first four minutes of flight, an average power consumption of 20W was measured. This average consumption, can be extrapolated to give a maximum flight time of approximately 3 hours, given that the battery stores approximately 60Wh of energy. In higher wind this average power would be higher but not by a factor of three so would still enable a flight time of greater than an hour. The maximum current measured was 11.97A, giving a power of 179W with the battery voltage dipping to 15V at the time. The GPS data shows up to 12 meters of movement which is roughly how far the sandbag was dragged.

The data-log file is included in the media in the rear cover.

7. Conclusion

The outcome of this project has been to demonstrate a functional prototype of a kite controller for airborne wind energy generation. This type of device is suited to integration with autonomous controllers for the development of small-scale generators for off-grid power supply applications. Airborne wind energy has the potential to be a significant participant in the future of sustainable energy production and result of this work is a system to aid in furthering the development thereof.

In the course of this design process, the available knowledge of airborne wind energy systems was surveyed and a concept of the system to be designed was developed with specific performance characteristics. This concept was condensed as the detailed design of a prototype which was then built and tested. This prototype kite controller was flown and shown to operate as designed, satisfying the specifications laid out. It has been flight tested on four occasions with about 20 minutes of flight time.

The modular approach to the sub-systems allowed for rapid adaptation of the separate parts of the full system while it was being integrated to ensure that they met the performance levels necessary to meet the specified requirements. This project could form a sell-able component, to be used by many different teams in their development of full AWE systems or as part of a temporary off-grid power system.

7.1. Results

The resulting prototype kite controller was shown to stably fly a kitesurfing kite on a five meter tether. These test flights enabled data recording of the kite controller dynamics and of the typical control inputs and forces required to fly a kite. The majority of the designed subsystems were included in these tests with the exclusion of the on-board power generator and the servomotor actuated safety release mechanism. The results of the control system design was a stable position controller with a maximum travel rate that allowed for full travel in three seconds and good (sub 1cm) position accuracy as detailed in Figure 6.4. The subsystems tested performed to the specified level with the possible exception of the telemetry radio link whose range was poor but functional. The prototype was mechanically and electrically robust and is able to continue to be used for testing. It was not tested to end-of-life due to insufficient test time.

7.2. Future prospects for AWE

The field of airborne wind energy is developing very quickly, there are a variety of approaches that are being investigated and much basic science and systems integration is still to be done. In the next five years, it is likely that the topologies employed will stabilize and one or two approaches will emerge as most effective. Much work is to be done in demonstrating safe and consistent performance and understanding the life-spans of the materials used as this directly affects the implementation costs. The recent price reduction in solar as a competing renewable energy source and the regulatory hurdles still to be overcome in deploying AWE solutions are seen as the biggest risks to the technology space as a whole.

7.3. Further work

The demonstration of this system, integrated with the winch and flying at up to 100m altitudes is the next step in this design process and can be performed when the winch is ready. This will also allow for proper testing of the safety release and on-board power generation capabilities of the design.

This work's scope is centred on producing a prototype that fulfils the requirements set out herein and so most of the design decisions have been optimised for this. However, as this field of study and specific application is changing so rapidly, much of what has been designed will be superseded by improvements and revisions. Thus it is likely that elements or sub-modules of the full system will be used again separately or be incorporated into new systems. This reiterates the importance of a modular implementation of the subsystems.

It would be useful to instrument the kite to measure acceleration, rotation and wind speed over the surface. This would provide more data that could be used for kite characterisation and control system feedback. This was not done here as it was deemed useful to use a kite as is. If modification of the kite is done, it would include shortening the lines between the kite and the controller and probably modifying the bridle of the kite to work with the different angle between the trailing edge lines.

The system as built can also use computer generated control inputs to fly the kite. This allows software to be developed that can provide very consistent control input to different kites to compare their responses. It also allows for the same inputs to be applied to a real kite and a kite simulation to assist in finding discrepancies in the models. This autonomous flight will also be important in the completion of an off-grid power generating system.

In the next major revision the controller could be built onto/into the wing which would remove the swinging and aerodynamic drag of the controller. A kite would be modified or possibly custom built for this use. Solid wing designs have not been considered here but their pursuit would allow for designs with significantly higher lift to drag coefficient ratios which would lead to much higher power output per unit area of the wing.

Acknowledgments

Thank-you, firstly, to Samuel Ginsberg as my supervisor for your guidance, critical insight and research support and the electrical engineering department of UCT.

To Dr. Ian de Vries for initiating this project and providing the flexibility of working environment and support needed to complete this dissertation and for the encouragement to learn to kite-surf, a worthy distraction.

And to my family and friends, especially Julia Merrett, for your unwavering love and support during the duration of this study.

Bibliography

[1] C. L. Archer and K. Caldeira, "Global assessment of high-altitude wind power." *Energies*, vol. 2, no. 2, pp. 307–319, May 2009.

[2] Makani, "Makani power, google inc." Online, Accessed Jul 2014, 2014, http://www.google.com/makani/. [Online]. Available: http://www.google.com/makani/

[3] Altaeros, "Altaeros buoyant airborne turbine," Online, 2014, http://www.altaerosenergies.com/ Acessed: Nov 2014. [Online]. Available: http://www.altaerosenergies.com/

[4] C. Jehle, "Automatic flight control of tethered kites for power generation," Ph.D. dissertation, TUM, http://mediatum.ub.tum.de/doc/1185997/file.pdf, 2012, accessed Sep 2014. [Online]. Available: http://mediatum.ub.tum.de/doc/1185997/file.pdf

[5] Ampyx, "Ampyx powerplane," Online, Acessed: Nov 2014, 2014, http://www.ampyxpower.com/OurTechnology.html. [Online]. Available: http://www.ampyxpower.com/OurTechnology.html

[6] M. Diehl, *Airborne Wind Energy*, 1st ed., ser. Green Energy and Technology. Berlin: Springer, 2013, ch. Airborne Wind Energy: Basic Concepts and Physical Foundations, pp. 3–22.

[7] R. Schmehl, "Large-scale power generation with kites," *Journal of the Society of Aerospace Engineering Students*, pp. 21–22, March 2012. [Online]. Available: http://resolver.tudelft.nl/uuid:84b37454-5790-4708-95ef-5bc2c60be790

[8] R. H. Luchsinger, "Weight matters: Tensarity kites." in *AWEC Conference Proceedings*. Airborne Wind Energy Confernce, September 2010. [Online]. Available: http://www.awec2010.com/public/presentations/luchsinger_rolf.pdf

[9] K. Marvel, B. Kravitz, and K. Caldeira, "Geophysical limits to global wind power," *Nature Climate Change*, vol. 2, no. 9, pp. 1–4, 2012.

[10] H. Neisser, J.and Steinhagen, "History of the meteorological observatory lindenberg 1905-2005," *promet*, vol. 31, no. 2-4, pp. 82–114, 2005. [Online]. Available: http://www.dmg-ev.de/gesellschaft/publikationen/pdf/promet/31_2-4.pdf

[11] I. E. Agency, "World energy investment outlook," Online, Accessed Jan 2015 2014, http://www.iea.org/publications/freepublications/publication/weio2014.pdf.

[12] EnerKite, "Enerkite ek30," Online, Accessed Dec 2014, 2014, http://www.enerkite.de/en/. [Online]. Available: http://www.enerkite.de/en/

[13] M. Barnard, "Airborne wind energy: It's all platypuses instead of cheetahs," March 2014, http://cleantechnica.com/2014/03/03/airborne-wind-energy-platypuses-instead-cheetahs/ , Online, Accessed May 2014. [Online]. Available: http://cleantechnica.com/2014/03/03/airborne-wind-energy-platypuses-instead-cheetahs/

[14] C. A. A. of South Africa, "Objects affecting airspace," http://www.caa.co.za/Pages/Obstacles/Objects-affecting-airspace.aspx, 2014, accessed: Dec 2014. [Online]. Available: http://www.caa.co.za/Pages/Obstacles/Objects-affecting-airspace.aspx

[15] EnerKite, "Notification for awes," Online, Acessed Nov 2014, 2014, http://www.energykitesystems.net/FAA/FAAfromEnerkite.pdf. [Online]. Available: http://www.energykitesystems.net/FAA/FAAfromEnerkite.pdf

[16] chinakites.org, "History of chinese kites," Online, accessed Aug 2014, 2014, http://www.chinakites.org/htm/fzls-gb.htm. [Online]. Available: http://www.chinakites.org/htm/fzls-gb.htm

[17] W. Schmidt and W. Anderson, *Airborne Wind Energy*, 1st ed., ser. Green Energy and Technology. Springer, 2013, ch. Kites: Pioneers of Atmospheric Research, pp. 95–116.

[18] P. Payne and C. McCutchen, "Self-erecting windmill," Oct. 26 1976, uS Patent 3,987,987. [Online]. Available: http://www.google.com/patents/US3987987

[19] M. L. Loyd, "Crosswind kite power," *Journal of Energy*, vol. 4, no. 3, pp. 106–111, 1980.

[20] T. Burton, D. Sharpe, N. Jenkins, and E. Bossanyi, *Wind Energy Handbook*. Chichester: John Wiley & Sons, 2001.

[21] J. R. Holton, *An Introduction to Dynamic Meteorology*, 3rd ed. San Deigo, USA: Academic Press, 1992.

[22] C. Archer, *Airborne Wind Energy*, 1st ed., ser. Green Energy and Technology. Berlin: Springer, 2013, ch. An Introduction to Meterology for Airborne Wind Energy, pp. 81–94.

[23] C. L. Archer, L. D. Monache, and D. L. Rife, "Airborne wind energy: Optimal locations and variability," *Renewable Energy*, vol. 64, no. 0, pp. 180 – 186, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0960148113005752

[24] M. Z. Jacobson and C. L. Archer, "Saturation wind power potential and its implicaitions for wind energy," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 109, no. 39, pp. 15 679–15 684, 2012.

[25] L. Fagiano, A. U. Zgraggen, and M. Khammash, "Automatic crosswind flight of tethered wings for airborne wind energy: modeling, control design and experimental results," *Submitted to IEEE Transactions on Control Technology*, 2013.

[26] L. Fagiano, M. Milanese, V. Razza, and M. Bonansone, "High-altitude wind energy for sustainable marine transportation," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 13, no. 2, pp. 781–791, June 2012.

[27] L. Fagiano, M. M., and D. Piga, "Optimization of airborne wind energy generators," *International Journal of Robust and Nonlinear Control*, vol. 22, no. 18, pp. 2055–2083, 2011.

[28] L. Fagiano, K. Huynh, B. Bamieh, and M. Khammash, "Sensor fusion for tethered wings in airborne wind energy," in *Proceedings of the 2013 American Control Conference*. Washington DC, USA: American Control Conference, 2013, pp. 2055–2083.

[29] Joby, "Joby energy and makani power merge," Online, Accessed Dec 2014, 2014, http://www.jobymotors.com/public/views/pages/company.php. [Online]. Available: http://www.jobymotors.com/public/views/pages/company.php

[30] SkySails, "Skysails functional 55 kw model," Online, Acessed May 2014, SkySails GmbH, www.skysails.info/english/power/development/1-functional-model-55-kw/. [Online]. Available: www.skysails.info/english/power/development/1-functional-model-55-kw/

[31] SkyWindPower, "Sky wind power - flying electric generators," Online, Acessed Nov 2014, 2013, http://www.skywindpower.com/. [Online]. Available: http://www.skywindpower.com/

[32] R. Schmehl, "Kite power," Online, Acessed Nov 2014, http://www.kitepower.eu/home.html. [Online]. Available: http://www.kitepower.eu/home.html

[33] TwingTec, "Twingtec tt50," Online, Acessed Nov 2014, http://twingtec.ch. [Online]. Available: http://twingtec.ch

[34] Omnidea, "Omnidea high altitude wind power using the magnus effect," Online, Accessed Nov 2014, http://omnidea.net/site/index.php.

[35] U. Zillmann and S. Hach, *Airborne Wind Energy*, 1st ed., ser. Green Energy and Technology. Springer, 2013, ch. Financing Strategies for Airborne Wind Energy, pp. 117–137.

[36] K. Geebelen and J. Gillis, "Modelling and control of rotational start-up phase of tethered aeroplanes for wind energy harvesting," Master's thesis, KU Leuven, www.kuleuven.be/optec/files/Geebelen2010.pdf, 2010, accessed November 2014. [Online]. Available: www.kuleuven.be/optec/files/Geebelen2010.pdf

[37] L. Fagiano, M. Milanese, and D. Piga, "High-altitude wind power generation," *IEEE Transactions on Energy Conversion*, vol. 25, no. 1, pp. 168–180, 2010.

[38] B. Houska and M. Diehl, "Robustness and stablility optimization of power generating kite systems in a periodic pumping mode," in *Proceedings of the IEEE Multi-Conference on Systems and Control*. Yokohama, Japan: IEEE, September 2010, pp. 2172–2177.

[39] J. Heilman, "Technical and economic potential of airborne wind energy," Master's thesis, Utrecht University, http://igitur-archive.library.uu.nl/student-thesis/2012-1211-200451/Technical2012, accessed Nov 2014. [Online]. Available: http://igitur-archive.library.uu.nl/student-thesis/2012-1211-200451/Technical%20and%20Economic%20Potential%20of%20AWE.pdf

[40] C. Haucke, A. Bormann, and C. Schmidt, "Innovative approaches for certification, permission and safety of abwe," in *Airborne Wind Energy Conference 2013*. EnerKitem Reiner Lemoin Institut, 2013. [Online]. Available: www.enerkite.com

[41] AWEC. (2014) Airborne wind energy consortium http://www.aweconsortium.org/. Airborne Wind Energy Consortium. [Online]. Available: http://www.aweconsortium.org/

[42] W. J. Ockels, "Laddermill: a novel concept to exploit the energy in the airspace," *Journal of Aircraft Design*, vol. 4, no. 2-3, pp. 81–97, 2001.

[43] R. H. Luchsinger, F. Gohl, D. Costa, and R. Verheul, "Towards the design of twing." in *Proceedings of the Airborne Wind Energy Conference*, Leuben, Belgium, 24-25 May 2011.

[44] P. Williams, B. Lansdorp, and W. J. Ockels, "Flexible tethered kite with movable attachment points, part i: Dynamics and control," in *Proceedings of the AIAA Modelling and Simulations Technology Conference and Exhibit*, no. AIAA-2007-6628. Hilton Head, SC, USA: AIAA, Aug 2007, aIAA Modelling and Simulation Technologies Conference and Exhibit.

[45] E. Bontekoe, "How to launch and retrieve a tethered aircraft," Master's thesis, Delft University of Technology, 2010. [Online]. Available: http://repository.tudelft.nl/view/ir/uuid%3A0f79480b-e447-4828-b239-9ec6931bc01f/

[46] L. Fagiano, K. Huynh, B. Bamieh, and M. Khammash, "On sensor fusion for airborne wind energy systems," *Submitted to IEEE Transactions on Control Technology*, 2012. [Online]. Available: arXiv:1211.5060

[47] P. Williams, "Optimal wind power extraction with a tethered kite," in *Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit*. Keystone, CO, USA: AIAA, August 2006.

[48] J. Gillis, J. Goos, K. Geebelen, J. Swevers, and M. Diehl, "Optimal periodic control of power harvesting tethered airplanes." in *Proceedings of the 2012 American Control Conference*, Montreal Canada, June 2012, pp. 2527–2532. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs-all.jsp?arnumber-6314924

[49] K. Geebelen, H. Ahmad, M. Vukov, S. Gros, J. Swevers, and M. Diehl, "An experimental test set-up for launch/recovery of an airborne wind

energy (awe) system," in *Proceedings of the 2012 American Control Conference*, Montreal, Canada, June 2012, pp. 5813–5818. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6315033

[50] I. Argatov and R. Silvennoinen, "Asymptotic modeling of unconstrained control of a tethered power kite moving along a given closed-loop spherical trajectory," *Journal of Engineering Mathematics*, vol. 72, no. 1, pp. 187–203, 2012.

[51] A. Ilzhofer, B. Houska, and M. Diehl, "Nonlinear mpc of kites under varying wind conditions for a new class of large-scale wind power generators," *International Journal of Robust and Nonlinear Control*, vol. 17, no. 17, pp. 1590–1599, 2007.

[52] P. Williams, B. Lansdorp, and W. J. Ockels, "Nonlinear control and estimation of a tethered kite in changing wind conditions," *AIAA Journal of Guidance, Control and Dynamics*, vol. 31, no. 3, 2008.

[53] J. Nielsen, *Usablility Engineering*, 1993, ch. Chapter 5 - Response Time: The 3 Important Limits, http://www.nngroup.com/articles/response-times-3-important-limits/.

[54] N. E. M. Association, "Degrees of protection provided by enclosures (ip code)," Online, Acessed: Jan 2015, 2004, https://www.nema.org/Standards/ComplimentaryDocuments/ANSI-IEC-60529.pdf. [Online]. Available: https://www.nema.org/Standards/ComplimentaryDocuments/ANSI-IEC-60529.pdf

[55] W. Aluminium, "Standard catalog," Online, Accessed Sep 2014, http://www.wispeco.co.za/standardcatalogue1007.pdf. [Online]. Available: http://www.wispeco.co.za/standardcatalogue1007.pdf

[56] Montar, "nrf905 radio module," Online, Accessed Jan 2014, http://www.rfdesign.co.za/Files/5645456/RFOEM[Online]. Available: http://www.rfdesign.co.za/Files/5645456/RFOEM%20manual1.5%20Montar%20LH%20pdf.pdf

[57] N. G. D. Center, "The world magnetic model," Online, Accessed Dec 2014, 2015, http://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml. [Online]. Available: http://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml

[58] FreeRTOS, "Open-source, embedded real-time operating system," Online, Accessed Dec 2014, http://www.freertos.org/.

[59] Solarplaza, "Pv solar efficiency," Online, Accessed Apr 2014, www.solarplaza.com/top10-monocrystalline-cell-efficiency. [Online]. Available: http://www.solarplaza.com/top10-monocrystalline-cell-efficiency/

[60] A. Betz, *Introduction to the Theory of Flow Machines*, T. D. G. Randall, Ed. Oxford: Pergamon Press, 1966.

[61] T. Maxino, "The effectiveness of checksums for embedded networks," Master's thesis, Carnegie Mellon University, May 2006,

http://users.ece.cmu.edu/ koopman/thesis/maxino_ms.pdf. [Online]. Available: http://users.ece.cmu.edu/~koopman/thesis/maxino_ms.pdf

[62] J. Coalson, "Crc8 implementation for libflac," Online, Accessed Dec 2013, http://libjflac-java.sourcearchive.com/documentation/1.3-1/CRC8_8java-source.html. [Online]. Available: http://libjflac-java.sourcearchive.com/documentation/1.3-1/CRC8_8java-source.html

Nomenclature

ATP	Acceptance test procedure
AWE	Airborne Wind Energy
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
GPS	Global Positioning System
GUI	Graphical User Interface
HAWT	Horizontal Axis Wind Turbine
IIC	Inter integrated circuit
IMU	Inertial Motion Unit
ISM	Industrial Scientific Medical
KCU	kite control unit
MEMS	Micro Electro-Mechanical System
MOSFET	Metal-oxide semiconductor field-effect transistor
RTOS	Real Time Operating System
UART	Universal Asynchronous Transmitter Receiver - a serial port
UCT	University of Cape Town

A. Appendix

A.1. Attached media

The DVD attached to the rear cover of this dissertation includes the following files:

- Video footage of the testing of the kite controller
- The software projects for the embedded systems and ground-station code
- The electronic CAD project for the motor drive circuit
- Datasheets for electronic parts used in this design.
- The data log file of the fourth test

A.2. Datagram Tables

A.2.1. Ground to kite message

No	Byte	Eg	Comment
0	Start Byte	0xFF	start of frame
1	Control Status	0x01	bit-flag indicators
2	Motor 1 position setpoint 0	0x7F	setpoint low byte
3	Motor 1 position setpoint 1	0xXX	setpoint high byte
4	Motor 1 speed setpoint 0	0xXX	setpoint low byte
5	Motor 1 speed setpoint 1	0xXX	setpoint high byte
6	Motor 1 Position Max	0xXX	position limit setting
7	Max Motor 1 Position Min	0xXX	position limit setting
8	Motor 1 Mode	0x01	0-stop 1-spd 2-pos
9	Motor 1 Reserved	0x00	null
10	Motor 2 position setpoint 0	0x00	setpoint low byte
11	Motor 2 position setpoint 1	0x00	setpoint high byte
12	Motor 2 speed setpoint 0	0x00	setpoint low byte
13	Motor 2 speed setpoint 1	0x00	setpoint high byte
14	Motor 2 Position Max	0x00	position limit setting
15	Max Motor 2 Position Min	0x00	position limit setting
16	Motor 2 Mode	0x01	0-stop 1-spd 2-pos
17	Motor 2 Reserved	0x00	null
18	Slave Power control 0	0x00	not used
19	Slave Power control 1	0x00	not used
20	Slave Power control 2	0x00	not used
21	Slave Power control 3	0x00	not used
22	Motor Speed/duty Limit	0x80	maximum drive duty cycle
23	Reserved	0x00	for future use,
24	Reserved	0x00	null value sent,
25	Reserved	0x00	packets are sent,
26	Reserved	0x00	faster if they,
27	Reserved	0x00	are 31 bytes long,
28	Reserved	0x00	using this module.
29	Query Number	0x01	counter
30	Checksum Byte	0xFF	8-bit CRC to verify message integrity

Table A.1.: Ground Station Transmitted Message Structure

A.2.2. Kite to ground message

A.2.2.1. Main message

No	Byte	Eg	Comment
0	Start Byte	0xFF	start of frame
1	Status/Errors	0x01	Flag bits indicating status
2	elevation stick	0x7F	0-100 (%)
3	direction stick	0xXX	0-100 (%)
4	Motor 1 position	0xXX	raw position value (50-143)
5	Motor 1 Status	0xXX	Flag bits indicating status
6	Motor 1 current hi	0xXX	Two byte motor
7	Moter 1 current lo	0xXX	current (raw ADC value)
8	Motor 2 Position	0x01	raw position value (50-143)
9	Motor 2 Status	0x00	Flag bits indicating status
10	Motor 2 current hi	0x00	Two byte motor
11	Moter 2 current lo	0x00	current (raw ADC value)
12	Battery Voltage hi	0x00	Two byte battery
13	Battery Voltage lo	0x00	pack voltage (mV)
14	Acceleration X lo	0x00	Two byte signed
15	Acceleration X hi	0x00	acceleration value
16	Acceleration Y lo	0x01	Two byte signed
17	Acceleration Y hi	0x00	acceleration value
18	Acceleration Z lo	0x00	Two byte signed
19	Acceleration Z hi	0x00	acceleration value
20	Pressure lo	0x00	Four
21	Pressure mid	0x00	byte
22	Pressure hi	0x80	pressure
23	Pressure hi hi	0x00	value
24	Temperature lo	0x00	Two byte
25	Temperature hi	0x00	temperature value
26	Reserved	0x00	null
27	left sick	0x00	0-100 (%)
28	Motor 1 setpoint	0x00	raw position value (50-143)
29	Motor 2 setpoint	0x00	raw position value (50-143)
30	Checksum Byte	0xFF	8-bit CRC to verify message integrity

Table A.2.: Kite Controller Transmitted Message 1 Structure

A.2.2.2. Secondary message

No	Byte	Eg	Comment
0	Start Byte	0xFF	Start of Frame
1	Pos Lat	0xXX	Latitude reading from
2	Pos Lat	0xXX	the GPS receiver in
3	Pos Lat	0xXX	deci-degrees (no pt)
4	Pos Lat	0xXX	Four bytes long
5	Pos Long	0xXX	Longitude reading from
6	Pos Long	0xXX	the GPS receiver in
7	Pos Long	0xXX	deci-degrees (no pt)
8	Pos Long	0xXX	Four bytes long
9	Time	0xXX	in UTC, (null)
10	Time	0xXX	Hours
11	Time	0xXX	Minutes
12	Time	0xXX	Seconds
13	Altitude	0xXX	four
14	Altitude	0xXX	byte
15	Altitude	0xXX	value in
16	Altitude	0xXX	meters
17		0x00	null
18		0x00	null
19		0x00	null
20		0x00	null
21		0x00	null
22	TELEMETRY_STATUS	0x00	Flags indicating radio status
23	IIC_DEV_STATUS	0x00	Flags indicating sensor status
24	NODE1_STATUS	0x00	Flags indicating drive 1 status
25	NODE2_STATUS	0x00	Flags indicating drive 2 status
26	RS485_STATUS	0x00	Flags indicating comms status
27	BATT_STATUS	0x00	Flags indicating battery status
28	GPS_STATUS	0x00	Flags indicating receiver status
29		0x00	null
30	Checksum Byte	0xFF	8-bit CRC to verify message integrity

Table A.3.: Kite Controller Transmitted Message 2 Structure

A.2.3. Controller to node message

No	Byte	Comment
0	Start Byte	start of frame
1	Address | Mode	Address nibble | Mode nibble
2	Duty Setpoint	Actual duty value for speed-control mode
3	Position Setpoint	Setpoint for position control mode
4	Position Maximum	Upper position limit
5	Postition Minimum	Lower position limit
6	Speed Max	A speed limit value to cap the maximum duty
7	CRC8 Checksum	8-bit CRC to check message integrity

Table A.4.: Node Received Message Structure

A.2.4. node to controller message

No	Byte	Comment
0	Start Byte	start of frame
1	Address | Status	Address nibble | Status nibble
2	Position Value	Measured value of current position of the winch
3	Current - high byte	Motor current measurement - high byte
4	Current - low byte	Motor current measurement - low byte
5	Reserved	null
6	Reserved	null
7	CRC8 Checksum	8-bit CRC to check message integrity

Table A.5.: Node Reply Message Structure

A.3. Java code for the Ground Station

The code listed below is not the complete listing of the software used in the ground station or the embedded systems. For full build-able code, refer to the media stored in the rear cover of this report.

A.3.1. Kite ground station class

```
/**
 * This Class generates the GUI for the kite ground station.
 * It controls the telemetry link and displays live data
 * @author Matteo Milandri
 */
```

```
public class KiteGroundStation extends javax.swing.JFrame {

public static final String comPort = "COM5";
public static final String gpsComPort = "";//"COM1";
LinePlot accelerationPlot;
LinePlot plotBattV;
Telemetry radio;
GPS_Receiver gndGPS;
LogCSV log;
Timer delayTimer;
Robot rob;

/* Creates new KiteGroundStation Frame
 */
public KiteGroundStation() {
    super("Ground Station");
    initComponents();

    plotBattV = new LinePlot();
    plotBattV.graphLowerLimit = 12;
    plotBattV.graphUpperLimit = 17;
    plotBattV.seriesTitle = "Batt_";
    plotBattV.graphTitle = "Battery Voltage";
    plotBattV.yLabel = "Battery Voltage (V)";
    plotBattV.init(panelGraphBattV, 1);

    accelerationPlot = new LinePlot();
    accelerationPlot.graphLowerLimit = -6000;
    accelerationPlot.graphUpperLimit = 6000;
    accelerationPlot.seriesTitle = "Acc_";
    accelerationPlot.graphTitle = "Acceleration";
    accelerationPlot.yLabel = "G's";
    accelerationPlot.init(graphMotionPanel2, 3);

    log = new LogCSV("Test Note");
    radio = new Telemetry(comPort) {
        @Override
        public void returnMessage() {
            receive();
            log.log(radio.kiteData);
            barInd3.setValue((barInd3.getValue() == 100) ? 0 : 100);
        }
        @Override
        public void getValues() {
            radio.txBuff[4] = (byte) (sliderSpeedSp2.getValue() & 0xFF);
            radio.txBuff[12] = (byte) (sliderSpeedSp1.getValue() & 0xFF);
            radio.txBuff[22] = (byte) (sliderSpeedMax.getValue() & 0xFF);
            barInd1.setValue((barInd1.getValue() == 100) ? 0 : 100);
        }
    };
    if (!gpsComPort.equals("")) {
        gndGPS = new GPS_Receiver(gpsComPort) {
```

```
            @Override
            public void returnMessage(GPSData data) {
                radio.kiteData.gndGPS = data;
            }
        };
    }
    delayTimer = new Timer();
    delayTimer.schedule(new TimerTask() {
        @Override
        public void run() {
            Point mouseLoc = MouseInfo.getPointerInfo().getLocation();
            //move the mouse to where it is to keep the screen on
            rob.mouseMove(mouseLoc.x, mouseLoc.y);
        }
    }, 30000, 30000);
}

void receive() {
    barInd2.setValue((barInd2.getValue() == 100) ? 0 : 100);//flashing indicator

    consoleText.setText(radio.kiteData.consoleString);
    barControlSetpointLeft.setValue(radio.kiteData.ctrlData.leftSetpoint);
    barControlSetpointRight.setValue(radio.kiteData.ctrlData.rightSetpoint);
    barPositionLeft.setValue(radio.kiteData.node1Data.position);
    barPositionRight.setValue(radio.kiteData.node2Data.position);
    barCurrentLeft.setValue((int) (radio.kiteData.node1Data.current * 50));
    barCurrentRight.setValue((int) (radio.kiteData.node2Data.current * 50));

    textAccelX.setText(radio.kiteData.ctrlData.acceleration[0] + "");
    barAccelX.setValue((int) radio.kiteData.ctrlData.acceleration[0]);
    textAccelY.setText(radio.kiteData.ctrlData.acceleration[1] + "");
    barAccelY.setValue((int) radio.kiteData.ctrlData.acceleration[1]);
    textAccelZ.setText(radio.kiteData.ctrlData.acceleration[2] + "");
    barAccelZ.setValue((int) radio.kiteData.ctrlData.acceleration[2]);
    accelerationPlot.addPoint(0, radio.kiteData.ctrlData.acceleration[0]);
    accelerationPlot.addPoint(1, radio.kiteData.ctrlData.acceleration[1]);
    accelerationPlot.addPoint(2, radio.kiteData.ctrlData.acceleration[2]);

    textPressure.setText(radio.kiteData.ctrlData.pressure + " hPa");
    textTemperature.setText(radio.kiteData.ctrlData.temperature + " C");
    double pressure = radio.kiteData.ctrlData.pressure;
    int altitude = radio.kiteData.ctrlData.getAltitude(pressure);
    textAltitude.setText(altitude+" M");
    plotBattV.addPoint(0, radio.kiteData.ctrlData.voltage);
    textBattV.setText(radio.kiteData.ctrlData.voltage + " V");
}

private void toggleButtonManualActionPerformed(java.awt.event.ActionEvent evt) {
    System.out.println("RC Control = " + toggleButtonManual.isSelected());
    radio.rcControl(toggleButtonManual.isSelected());
}
```

```
private void buttonSendActionPerformed(java.awt.event.ActionEvent evt) {
    radio.sending = !radio.sending;
    if (radio.sending) {
        buttonSend.setText("Stop Telemetry");
    } else {
        buttonSend.setText("Start Telemetry");
    }
}

private void formWindowClosing(java.awt.event.WindowEvent evt) {
    if (radio != null) {
        radio.sending = false;
        radio.close();
    }
    if (gndGPS != null) {
        gndGPS.close();
    }
    if (log != null) {
        log.close();
    }
    System.exit(0);
}

private void buttonLogDataActionPerformed(java.awt.event.ActionEvent evt) {
    if (buttonLogData.isSelected()) {
        log.open();
    } else {
        log.close();
    }
}

private void buttonPinPressureActionPerformed(java.awt.event.ActionEvent evt) {
    radio.kiteData.ctrlData.pressureOnGround = radio.kiteData.ctrlData.pressure;
}

public static void main(String args[]) {
    try {//look and feel
        javax.swing.UIManager.setLookAndFeel(
            javax.swing.UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.out.println(e);
    }
    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            new KiteGroundStation().setVisible(true);
        }
    });
}
}
```

A.3.2. Telemetry class

```
package kiteControl;

import java.util.Timer;
import java.util.TimerTask;
import jssc.SerialPort;
import jssc.SerialPortEvent;
import jssc.SerialPortEventListener;
import jssc.SerialPortException;

/**
 * This class opens the serial port link to the telemetry device
 * and performs the message packetisation, sending and reception.
 * @author Matteo Milandri
 */
public class Telemetry implements SerialPortEventListener {

    /* Constants */
    public static final byte motor1PosMax = (byte) (144 & 0xFF);
    public static final byte motor1PosMin = (byte) (49 & 0xFF);
    public static final byte motor2PosMax = (byte) (144 & 0xFF);
    public static final byte motor2PosMin = (byte) (49 & 0xFF);
    public static final byte defaultSpeedSetpoint = 127;
    public static final byte positionControlMode = 1;
    public static final byte speedControlMode = 0;
    public static final int messageLength = 31;

    /* Variables */
    DataStore kiteData;
    SerialPort serialPort;
    Timer serialTrigger;
    boolean start_transfer = false;
    public byte[] txBuff = new byte[messageLength];
    byte[] rxArr = new byte[messageLength - 1];
    byte[] rxBuff;
    int rxIndex = 0;
    int reply_id = 0;
    public boolean sending = false;
    public long last_millis = 0;

    public Telemetry(String comPort) {
        init(comPort);
    }

    private void init(String comPort) {
        for (int f : txBuff) {
            txBuff[f] = 0;
        }
        txBuff[0] = (byte) 0xFF;            //start byte
        txBuff[txBuff.length - 1] = 0;      //checksum
```

```
        txBuff[1] = 1;                          //control status
        txBuff[2] = (byte) (100 & 0xFF);        //position setpoint 1
        txBuff[4] = defaultSpeedSetpoint;       //motor 1
        txBuff[6] = motor1PosMax;
        txBuff[7] = motor1PosMin;
        // 0 = speed, 1 = position RC, 2 = position ground
        txBuff[8] = 1;//motor 1 mode
        txBuff[10] = (byte) (100 & 0xFF);       //position setpoint 2
        txBuff[12] = defaultSpeedSetpoint;      //motor 2
        txBuff[14] = motor2PosMax;
        txBuff[15] = motor2PosMin;
        txBuff[16] = 1;//motor 2 mode
        txBuff[29] = 0;//query number

        last_millis = System.currentTimeMillis();

        kiteData = new DataStore();

        serialPort = new SerialPort(comPort);
        try {
            serialPort.openPort();
            serialPort.purgePort(SerialPort.PURGE_RXCLEAR);
            serialPort.setParams(19200, 8, 1, 0);
            serialPort.setEventsMask(SerialPort.MASK_RXCHAR);
            serialPort.addEventListener(this);
        } catch (SerialPortException ex) {
            System.err.println("Telemetry Error " + ex);
        }

        serialTrigger = new Timer();
        serialTrigger.schedule(new TimerTask() {
            @Override
            public void run() {
                if (sending) {
                    getValues();
                    sendMessage();
                }
            }
        }, 500, 100);
    }

    void sendMessage() {
        byte checksum = 0;
        for (int i = 1; i < txBuff.length - 1; i++) {
            checksum = CRC8.update(txBuff[i], checksum);
        }
        txBuff[txBuff.length - 1] = checksum;
        try {
            if (sending) {
                serialPort.writeBytes(txBuff);
            }
        } catch (SerialPortException e) {
```

```
            System.err.println("Error␣Sending␣" + e.toString());
        }
    }

    void receiveMessage(int id) {
        if (id == 0) {
            kiteData.setValues(rxArr);
        } else if (id == 1) {
            kiteData.setAltValues(rxArr);
        }
        returnMessage();
    }

    void rcControl(boolean ctrl) {
        if (ctrl) {
            txBuff[8] = positionControlMode;
            txBuff[16] = positionControlMode;
        } else {
            txBuff[8] = speedControlMode;
            txBuff[16] = speedControlMode;
        }
    }

    /**
     * Sends a flag back to the class that has
     * created an instance of this class.
     */
    public void returnMessage() {
        System.out.println("Function␣not␣overloaded␣properly!");
    }

    /*
     * maybe a better name?
     */
    public void getValues() {
        System.out.println("getValues␣not␣overloaded");
    }

    boolean close() {
        try {
            serialPort.closePort();
        } catch (SerialPortException exception) {
            System.out.println("Error:␣" + exception.toString());
            return false;
        }
        return true;
    }

    public int millis() {
        int temp = (int) (System.currentTimeMillis() - last_millis);
        last_millis = System.currentTimeMillis();
        return temp;
```

```
    }

    public static void main(String[] args) {
        //use this to perform a simple test of this class
        System.out.println("Does␣nothing␣if␣run␣from␣here!");
    }

    @Override
    public void serialEvent(SerialPortEvent event) {
        if (event.isRXCHAR()) {
            try {
                rxBuff = serialPort.readBytes(event.getEventValue());
            } catch (SerialPortException ex) {
                System.out.println(ex);
            }
            if (rxBuff == null) {
                return;
            }
            for (byte val : rxBuff) {
                if (rxIndex == 0) {
                    if (val == (byte) 0xFF) {
                        rxIndex++;
                        reply_id = 0;
                    } else if (val == (byte) 0xFE) {
                        rxIndex++;
                        reply_id = 1;
                    }
                } else if (rxIndex > 0 && rxIndex < msgLen - 1) {
                    rxArr[rxIndex - 1] = val;
                    rxIndex++;
                } else if (rxIndex == msgLen - 1) {//message end
                    if (CRC8.calc(rxArr, rxArr.length - 1) == val) {
                        receiveMessage(reply_id);
                    } else {
                        System.out.println(millis() + "␣Checksum␣err");
                    }
                    rxIndex = 0;
                } else {
                    System.out.println(millis() + "␣state␣err");
                }
            }
        }
    }
}
```

A.3.3. Data dissemination

```
package kiteControl;

/**
 * This class holds the definition of the data sent from the kite
 * controller to  the ground
 * @author Matteo
 */
public class DataStore {
    public static final String[] consoleLabels = {"Status\t", "Cont UD\t",
        "Cont LR\t", "Pos 1\t", "Status 1", "I 1 Hi\t", "I 1 Lo\t", "Pos 2\t",
        "Status 2", "I 2 Hi\t", "I 2 Lo\t", "Batt V lo", "Batt V hi", "AccelX lo",
        "AccelX hi", "AccelY lo", "AccelY hi", "AccelZ lo", "AccelZ hi",
        "Pressure lo", "Pressure mid", "Pressure hi", "Pressure hi hi",
        "Temperature lo", "Temperature hi", "String26", "left stick",
        "M1 Setpoint", "M2 Setpoint", "Null Value"};

    String consoleString = "";
    String timeStamp = "";
    ControllerData ctrlData;
    GPSData gndGPS;
    GPSData kiteGPS;
    NodeData node1Data;
    NodeData node2Data;

    public DataStore() {
        ctrlData = new ControllerData();
        gndGPS = new GPSData();
        kiteGPS = new GPSData();
        node1Data = new NodeData();
        node2Data = new NodeData();
    }

    public void setValues(byte[] values) {
        if (values.length != 30) {
            return;
        }

        consoleString = "";
        for (int i = 0; i < values.length; i++) {
            consoleString = consoleString + DataStore.consoleLabels[i] + "\t"
                    + (int) (values[i] & 0xFF) + "\n";
        }

        ctrlData.status = (int)(0xFF & values[0]);
        ctrlData.controlUpDown = (int)(0xFF & values[1]);
        ctrlData.controlLeftRight = (int)(0xFF & values[2]);
        ctrlData.controlLeftUpDown = (int)(0xFF & values[26]);
        ctrlData.leftSetpoint = (int)(0xFF & values[27]);//Motor 1
        ctrlData.rightSetpoint = (int)(0xFF & values[28]);//Motor 2
```

```
        node1Data.position = (int)(0xFF & values[3]);
        node1Data.status = (int)(0xFF & values[4]);
        node1Data.current = ((values[5] & 0xFF) << 8) + (values[6] & 0xFF);
        node1Data.current = (node1Data.current / 1310) - 25;// 65535/50
        node1Data.setpoint = (int)(0xFF & values[27]);

        node2Data.position = (int)(0xFF & values[7]);
        node2Data.status = (int)(0xFF & values[8]);
        node2Data.current = ((values[9] & 0xFF) << 8) + (values[10] & 0xFF);
        node2Data.current = (node2Data.current / 1310) - 25;// 65535/50
        node2Data.setpoint = (int)(0xFF & values[28]);

        ctrlData.voltage = (double) (((values[12] & 0xFF) << 8)
                + (values[11] & 0xFF)) / 1000.0;
        ctrlData.acceleration[0] = (values[14] << 8) + (values[13] & 0xFF);
        ctrlData.acceleration[1] = (values[16] << 8) + (values[15] & 0xFF);
        ctrlData.acceleration[2] = (values[18] << 8) + (values[17] & 0xFF);
        ctrlData.pressure = (((values[21] & 0xFF) << 16)
                + ((values[20] & 0xFF) << 8) + (values[19] & 0xFF)) / 100.0;
        ctrlData.temperature = (((values[24] & 0xFF) << 8)
                + (values[23] & 0xFF)) / 10;
    }

    public void setAltValues(byte[] values) {
        if (values.length != 30) {
            return;
        }
        kiteGPS.latitude = catBytesGetValue(values, 1, 4) / 1000000.0;
        kiteGPS.longitude = catBytesGetValue(values, 5, 4) / 1000000.0;
        kiteGPS.time = catBytesGetValue(values, 9, 4) + "";
        if (kiteGPS.time.length() == 6) {
            kiteGPS.time = kiteGPS.time.substring(0, 2) + ":"
                    + kiteGPS.time.substring(2, 4) + ":"
                    + kiteGPS.time.substring(4, 6);
        }
        kiteGPS.altitude = catBytesGetValue(values, 13, 4);
        //System.out.println(kiteGPS.getData());
    }

    public String getDataText() {
        return (timeStamp + "," + ctrlData.getData() + ","
                + gndGPS.getData() + "," + kiteGPS.getData()
                + "," + node1Data.getData() + "," + node2Data.getData());
    }

    public static String getDataHeaders() {
        return ControllerData.getHeaders() + "," + GPSData.getHeaders()
                + "," + GPSData.getHeaders() + "," + NodeData.getHeaders()
                + "," + NodeData.getHeaders();
    }

    public int catBytesGetValue(byte[] array, int index, int length) {
```

```
        int temp = (((int) array[index + length - 1]) << ((length - 1) * 8));
        for (int i = length - 2; i >= 0; i--) {
            temp += (((int) (array[index + i] & 0xFF)) << (i * 8));
        }
        return temp;
    }
}
```

A.3.4. Data types

```
/**
 * Stores the data ascociated with the controller and its sensors
 * @author Matteo
 */
public class ControllerData {

    int status;
    double[] acceleration = {0.0, 0.0, 0.0};
    double[] rotation = {0.0, 0.0, 0.0};
    double[] magField = {0.0, 0.0, 0.0};
    double pressure = 0;
    double temperature = 0;
    double pressureOnGround = 0;
    int altitudePres = 0;
    double voltage = 0;
    int controlUpDown = 0;
    int controlLeftRight = 0;
    int controlLeftUpDown = 0;
    int leftSetpoint = 0;
    int rightSetpoint = 0;

    public ControllerData() {
    }

    /**
     * Uses the formulas specified in the datasheet to determine the altitude.
     *
     * @param pressure
     * @return altitude
     *
     */
    public int getAltitude(double pressure) {
        if (pressureOnGround == 0) {//make it 1000 if uninitialised.
            pressureOnGround = 1000;
        }
        altitudePres = (int) (44330 * (1 - Math.pow(pressure
                            / pressureOnGround, 1 / 5.255)));
        return altitudePres;
    }

    public static String getHeaders() {
        return "Voltage,Control␣Up/Down,Control␣Left/Right,"
```

```
            +"Temperature,Pressure,Alt(pres),AccX,AccY,AccZ";
    }

    public String getData() {
        return voltage + "," + controlUpDown + "," + controlLeftRight + ","
            + temperature + "," + pressure + "," + altitudePres + ","
            + acceleration[0] + "," + acceleration[1] + "," + acceleration[2];
    }
}


/**
 * Stores the data regarding each of the motor drive nodes that is sent
 * by the controller over the telemetry link
 * @author Matteo
 */
public class NodeData {
    int status;
    int setpoint;
    int position;
    double current;
    int maxPos;
    int minPos;

    public NodeData() {
        this.status = 0;
        this.current = 0.0;
        this.setpoint = 0;
        this.position = 0;
        this.maxPos = 143;
        this.minPos = 50;
    }

    public NodeData(int status, int setpoint, int position, double current,
            int maxPos, int minPos) {
        this.status = status;
        this.setpoint = setpoint;
        this.position = position;
        this.current = current;
        this.maxPos = maxPos;
        this.minPos = minPos;
    }

    public static String getHeaders() {
        return "Setpoint,Position,Current";
    }

    public String getData() {
        return setpoint + "," + position + "," + current;
    }
}
```

```
/**
 * Stores the data associated with the GPS modules.
 *
 * @author Matteo
 */
public class GPSData {

    double latitude;
    double longitude;
    int altitude;
    String time;
    int quality;
    int number_of_satellites;
    double hdop;

    public GPSData() {
        this.latitude = 0.0;
        this.longitude = 0.0;
        this.altitude = 0;
        this.time = "0:0:0";
        this.quality = 0;
        this.number_of_satellites = 0;
        this.hdop = 0.0;
    }

    public GPSData(double latitude, double longitude, int altitude,
            String time, int quality, int number_of_satellites, double hdop) {
        this.latitude = latitude;
        this.longitude = longitude;
        this.altitude = altitude;
        this.time = time;
        this.quality = quality;
        this.number_of_satellites = number_of_satellites;
        this.hdop = hdop;
    }

    public static String getHeaders() {
        return "Latitude,Longitude,Altitude,Time";
    }

    public String getData() {
        return latitude + "," + longitude + "," + altitude + "," + time;
    }
}
```

A.3.5. Data logging

```
/**
 * This Class provides the interface to perform the logging from the kite
 * control ground station Note: To use this, place a "log to file" button in the
 * program.
 *
 * TODO: make a second logfile that stores the events as seen in this.
 * For example events would be start of test, end of test, when certain buttons
 * are pushed etc.
 *
 * @author Matteo
 */
public class LogCSV extends Thread {

    FileWriter writer;
    String fileName;
    File file;
    boolean isOpen;
    String notes;
    Calendar cal;
    DateFormat timeFormat;
    String tempDat;
    ArrayBlockingQueue<String> queue;

    public LogCSV() {
        this("");
    }

    public LogCSV(String notesText) {
        queue = new ArrayBlockingQueue<>(120);
        DateFormat dateFormat = new SimpleDateFormat("yyyy_MM_dd");
        timeFormat = new SimpleDateFormat("HH-mm-ss");
        cal = Calendar.getInstance();
        fileName = "C:\\Users\\Matteo\\Dropbox\\Kites\\TestLogs\\test"
                        + dateFormat.format(cal.getTime()) + ".csv";
        notes = notesText;
    }

    //for testing
    public static void main(String[] args) throws InterruptedException {
        LogCSV log = new LogCSV("notes");
        log.open();
        log.log(new DataStore());
        Thread.sleep(1000);
        log.log(new DataStore());
        Thread.sleep(1000);
        log.log(new DataStore());
        Thread.sleep(1000);
        log.log(new DataStore());
        Thread.sleep(1000);
        log.log(new DataStore());
```

```
        log.close();
    }

    /**
     * Overrides the Thread.run()
     */
    @Override
    public void run() {
        try {
            file = new File(fileName);
            if (file.exists()) {//append to the end of the file.
                writer = new FileWriter(file, true);
            } else {
                writer = new FileWriter(file);
                createHeader();
            }
            writer.append("\n");
            isOpen = true;
        } catch (IOException ex) {
            System.err.println("Could Not Create file: " + fileName);

        }

        while (isOpen) {
            if (queue.size() > 60) {
                while (queue.size() > 0) {
                    tempDat = queue.poll();
                    if (tempDat != null) {
                        writeData(tempDat);
                    }
                }
                flushData();
            } else {
                try {
                    sleep(1);
                } catch (InterruptedException ex) {
                    ex.printStackTrace(System.err);
                }
            }
        }
        //write all the remaining queue items to the file
        while (queue.size() > 0) {
            tempDat = queue.poll();
            if (tempDat != null) {
                writeData(tempDat);
            }
        }

        //close the file
        try {
            writer.flush();
            writer.close();
```

```
        } catch (IOException ex) {
            ex.printStackTrace(System.out);
        }
    }

    private void createHeader() {
        DateFormat dateFormat = new SimpleDateFormat("HH-mm-ss");
        try {
            //Get the current date and time to generate a new data file.
            writer.append("Logfile for the Kite Control Groundstation\n");
            writer.append("Start Time," + dateFormat.format(cal.getTime())
                + "\n");
            writer.append("\nNotes:," + notes);
            writer.append("\nTimeStamp," + DataStore.getDataHeaders());

            writer.flush();
        } catch (IOException ex) {
            ex.printStackTrace(System.out);
        }
    }

    public boolean flushData() {
        if (isOpen) {
            try {
                writer.flush();
                return true;
            } catch (IOException ex) {
                ex.printStackTrace(System.out);
            }
        }
        return false;
    }

    public boolean writeData(String dataString) {
        try {
            //This timestamp must be pulled when the log function is called.
            writer.append(dataString);
            return true;
        } catch (IOException ex) {
            ex.printStackTrace(System.out);
        }
        return false;
    }

/**
 * This method can be called by another thread that has data to be logged
 * The data is put onto a queue and the function returns.
 * This function does
 * not block and wait for space on the queue
 *
 * @param data
 * @return true if the data was successfully put onto the storage queue.
```

```
 */
    public boolean log(DataStore data) {
        if (isOpen) {
            cal = Calendar.getInstance();
            data.timeStamp = timeFormat.format(cal.getTime());
            System.out.println(data.timeStamp+"\t"+data.ctrlData.voltage);
            return queue.offer("\n" + data.getDataText());
        }
        return false;
    }

    /**
     * Starts the logging thread. called from outside this thread
     */
    public void open() {
        this.start();
    }

    /**
     * Changes the open flag to false which will end the writing thread.
     */
    public void close() {
        isOpen = false;
    }
}
```

A.3.6. Ground station GPS receiver

```
/**
 * Recieves data from the ground station GPS receiver over a
 * serial port connection and presents that data to the program.
 * @author matteo
 */
public class GPS_Receiver implements SerialPortEventListener {

    /* Variables */
    SerialPort serialPort;
    byte[] rxArr = new byte[100];
    byte[] rxBuff;
    int rxIndex = 0;
    public long last_millis = 0;
    public GPSData data;

    public GPS_Receiver(String comPort) {
        data = new GPSData();
        init(comPort);
    }

    private void init(String comPort) {
        last_millis = System.currentTimeMillis();

        serialPort = new SerialPort(comPort);
```

```
        try {
            serialPort.openPort();
            serialPort.purgePort(SerialPort.PURGE_RXCLEAR);
            serialPort.setParams(9600, 8, 1, 0);
            serialPort.setEventsMask(SerialPort.MASK_RXCHAR);
            serialPort.addEventListener(this);
        } catch (SerialPortException ex) {
            System.out.println(ex);
            //NOTE: This does nothing if there is no port available for it.
        }
    }

    void receiveMessage() {
        String received = new String(rxArr, 0, rxIndex - 1);
        String[] message = received.split(",");
        switch (message[0]) {
            case "GPGGA"://global position, latitude / longitude
                for (int i = 0; i < message.length; i++) {
                    System.out.println(i + "␣" + message[i]);
                }
                if (message.length == 15) {
                    if (message[1].isEmpty() == false) {
                        data.time = message[1];
                        System.out.println("Time␣" + data.time);
                    }
                    if (message[2].isEmpty() == false) {
                        data.latitude = Double.parseDouble(message[2]);
                        data.latitude = data.latitude / 100;
                        if (message[3].equalsIgnoreCase("S")) {
                            data.latitude = -data.latitude;
                        }
                        System.out.println("Latitude␣" + data.latitude);
                    }
                    if (message[4].isEmpty() == false) {
                        data.longitude = Double.parseDouble(message[4]);
                        data.longitude = data.longitude / 100;
                        if (message[5].equalsIgnoreCase("W")) {
                            data.longitude = -data.longitude;
                        }
                        System.out.println("Longitude␣" + data.longitude);
                    }
                    if (message[6].isEmpty() == false) {
                        data.quality = Integer.parseInt(message[6]);
                    }
                    if (message[7].isEmpty() == false) {
                        data.number_of_satellites =
                            Integer.parseInt(message[7]);
                    }
                    if (message[8].isEmpty() == false) {
                        data.hdop = Double.parseDouble(message[8]);
                    }
                    if (message[9].isEmpty() == false) {
```

```
                    data.altitude =
                        (int)Double.parseDouble(message[9]);
                }
                System.out.println("quality " + data.quality);
                System.out.println("number_of_satellites "
                    + data.number_of_satellites);
                System.out.println("hdop " + data.hdop);
                System.out.println("altitude " + data.altitude);
            }
            break;
        case "GPGSA"://GPS DOP and active satellites
            break;
        case "GPRMC"://recommended minimum specific GPS/Transmit data
            break;
        case "GPGSV"://GPS sattellites in view
            break;
        case "GPVTG"://Track made good and ground speed
            break;
        case "GPGLL"://Geographic position, latitude / longigtude
            break;
        case "GPZDA"://Date and time
            break;
        /*
         $GPRMC,,V,,,,,,,,,,N*53
         $GPVTG,,,,,,,,,N*30
         $GPGGA,,,,,,0,00,99.99,,,,,,*48
         $GPGSA,A,1,,,,,,,,,,,,,99.99,99.99,99.99*30
         $GPGSV,1,1,00*79
         $GPGLL,,,,,,V,N*64
         $GPZDA,,,,,00,00*48 */
    }
    //System.out.println(message[0] + "<-- " + message.length);

    returnMessage(data);
}

/**
 * Sends data back to the class that has created an instance of this class.
 *
 * @param data GPSData received
 */
public void returnMessage(GPSData data) {
    //This should be overridden
    System.out.println("GPS Reciver: "+ data.getData());
}

boolean close() {
    try {
        serialPort.closePort();
    } catch (SerialPortException exception) {
        System.out.println("Error: " + exception.toString());
        return false;
```

```
        }
        return true;
    }

    public int millis() {
        int temp = (int) (System.currentTimeMillis() - last_millis);
        last_millis = System.currentTimeMillis();
        return temp;
    }

    public static void main(String[] args) throws InterruptedException {
        GPS_Receiver gps = new GPS_Receiver("COM6");
        Thread.sleep(5000);
        gps.close();

    }

    @Override
    public void serialEvent(SerialPortEvent event) {
        if (event.isRXCHAR()) {
            try {
                rxBuff = serialPort.readBytes(event.getEventValue());
                for (int i = 0; i < event.getEventValue(); i++) {
                    if (rxBuff[i] == '$') {
                        rxIndex = 0;
                    } else if (rxBuff[i] == '\n') {
                        receiveMessage();
                    } else if (rxIndex < 100 - 1) {
                        rxArr[rxIndex] = rxBuff[i];
                        rxIndex++;
                    }
                }
            } catch (SerialPortException ex) {
                System.out.println(ex);
            }
        }
    }
}
```

A.4. Embedded code

The code was first developed for a Freescale Coldfire V1 32bit microcontroller and then ported and further developed in a Freescale Kinetis ARM Cortex M0+ microcontroller.

A.4.1. System master controller

A.4.1.1. Main

```
/*
 * This function starts the five FreeRTOS tasks.
 * Each task has been allocated 256 bytes of stack space and have
 * equal priorities of 2 (0 being heighst and 3 being lowest priority)
 */
int main(void) {
    PE_low_level_init();

    xTaskCreate((pdTASK_CODE)(&bus_master_main_service), "BusM",256,0,2,0);
    xTaskCreate((pdTASK_CODE)(&telemetry_main_service), "Tele", 256,0,2,0);
    xTaskCreate((pdTASK_CODE)(&radio_control_main_service), "RCtl",256,0,2,0);
    xTaskCreate((pdTASK_CODE)(&run_GPS),"GPS",256,0,2,0);
    xTaskCreate((pdTASK_CODE)(&run_sensors), "Sens", 256,0,2,0);

    PEX_RTOS_START();

  for(;;){}
}
```

A.4.1.2. RS485 Bus communication master

```
/** BusMaster.c
 *  Created on: Nov 11, 2013
 *      Author: Matteo Milandri
 */
#include "Cpu.h"
#include "Events.h"
#include "Sensors.h"
#include "BusMaster.h"
#include "Telemetry.h"
#include "GPS.h"
#include "RemoteController.h"
#include "CRC8.h"
#include "quickRelease.h"

NodeControl_t node[2];
NodeData_t nodeData[2];

//RS485 Comms related variables
uint8_t last_sent_address;
uint8_t RS485_comms_state;
```

```
uint8_t rx_index;
uint8_t RS485_rx_arr[RS485_MESSAGE_LENGTH - 1];
uint8_t RS485_tx_arr[RS485_MESSAGE_LENGTH];
uint8_t rx_buff[1];
uint8_t send_to_node_1_flag;
uint8_t send_to_node_2_flag;
LDD_TDeviceData *bus_master_handle;
QueueHandle_t bus_queue;
uint8 request;

void initNode(NodeControl_t * node, NodeData_t * data) {
    node->duty_setpoint = 127;
    node->position_max = 0;
    node->position_min = 0;
    node->position_setpoint = 0;
    node->mode = 0; // 0 = speed, 1 = position RC, 2 = position ground
    node->speed_max = 200;
    node->data_access = xSemaphoreCreateBinary();
    xSemaphoreGive(node->data_access);

    data->position = 0;
    data->status = 0;
    data->current_hi = 0;
    data->current_lo = 0;
    data->data_access = xSemaphoreCreateBinary();
    xSemaphoreGive(data->data_access);
}

void bus_master_init(void) {
    bus_master_handle = Bus_UART_Init(bus_master_handle);
    bus_queue = xQueueCreate(3,1);

    initNode(&node[0], &nodeData[0]);
    initNode(&node[1], &nodeData[1]);

    last_sent_address = 0;
    RS485_comms_state = RS485_WAIT;
    rx_index = 0;
    send_to_node_1_flag = 0;
    send_to_node_2_flag = 0;

    RS485_tx_arr[0] = 0xFF; // change to a break.
    RS485_tx_arr[1] = 0;    // <address><mode>
    RS485_tx_arr[2] = 0;    // duty_sp
    RS485_tx_arr[3] = 0;    // pos_sp
    RS485_tx_arr[4] = 0;    // pos_max
    RS485_tx_arr[5] = 0;    // pos_min
    RS485_tx_arr[6] = 0;    // spd_max
    RS485_tx_arr[7] = 0;    // CRC8

    node[0].duty_setpoint = 127;
    node[0].position_max = 140;
```

```
    node[0].position_min = 52;
    node[0].position_setpoint = 80;
    node[0].mode = 0;
    node[1].duty_setpoint = 127;
    node[1].position_max = 140;
    node[1].position_min = 52;
    node[1].position_setpoint = 80;
    node[1].mode = 0;
    Bus_UART_ReceiveBlock(bus_master_handle, rx_buff, 1);
}

void bus_master_main_service(void) {
    bus_master_init();
    for (;;) {
        // Allow the motors to start moving 5 seconds after startup
        if (millis_count > 5000) {
            node[0].mode = 1;
            node[1].mode = 1;
        }
        if (xQueueReceive(bus_queue,&request,1000) == pdTRUE ) {
            switch (request) {
            case REQ_SEND_TO_NODE1:
                if (xSemaphoreTake(node[0].data_access,20) == pdTRUE ) {
                    last_sent_address = 0x10;
                    send_to_node_1_flag = 0;
                    {
                        RS485_tx_arr[1] = last_sent_address | node[0].mode;
                        RS485_tx_arr[2] = node[0].duty_setpoint;
                        RS485_tx_arr[3] = node[0].position_setpoint;
                        RS485_tx_arr[4] = node[0].position_max;
                        RS485_tx_arr[5] = node[0].position_min;
                        RS485_tx_arr[6] = node[0].speed_max;
                    }
                    xSemaphoreGive(node[0].data_access);
                    RS485_tx_arr[7] = calculateCRC8((RS485_tx_arr + 1),
                        RS485_MESSAGE_LENGTH - 2);
                    RS485_tx_en_SetVal();
                    Bus_UART_SendBlock(bus_master_handle, RS485_tx_arr,
                        RS485_MESSAGE_LENGTH);
                }
                break;

            case REQ_SEND_TO_NODE2:
                if (xSemaphoreTake(node[1].data_access,20) == pdTRUE ) {
                    last_sent_address = 0x20;
                    send_to_node_2_flag = 0;
                    {
                        RS485_tx_arr[1] = last_sent_address | node[1].mode;
                        RS485_tx_arr[2] = node[1].duty_setpoint;
                        RS485_tx_arr[3] = node[1].position_setpoint;
                        RS485_tx_arr[4] = node[1].position_max;
                        RS485_tx_arr[5] = node[1].position_min;
```

```
                    RS485_tx_arr[6] = node[1].speed_max;
                }
                xSemaphoreGive(node[1].data_access);
                RS485_tx_arr[7] = calculateCRC8((RS485_tx_arr + 1),
                    RS485_MESSAGE_LENGTH - 2);
                RS485_tx_en_SetVal();
                Bus_UART_SendBlock(bus_master_handle, RS485_tx_arr,
                    RS485_MESSAGE_LENGTH);
            }
            break;

        case REQ_MESSAGE_RECEIVED:
            if (calculateCRC8(RS485_rx_arr, 7) == 0) {
                if (last_sent_address == 0x10) {
                    if (xSemaphoreTake(nodeData[0].data_access,20)
                            == pdTRUE ) {
                        nodeData[0].status = RS485_rx_arr[0];
                        nodeData[0].position = RS485_rx_arr[1];
                        nodeData[0].current_hi = RS485_rx_arr[2];
                        nodeData[0].current_lo = RS485_rx_arr[3];
                        xSemaphoreGive(nodeData[0].data_access);
                    }
                } else if (last_sent_address == 0x20) {
                    if (xSemaphoreTake(nodeData[1].data_access,20)
                            == pdTRUE ) {
                        nodeData[1].status = RS485_rx_arr[0];
                        nodeData[1].position = RS485_rx_arr[1];
                        nodeData[1].current_hi = RS485_rx_arr[2];
                        nodeData[1].current_lo = RS485_rx_arr[3];
                        xSemaphoreGive(nodeData[1].data_access);
                    }
                }
            } else {
                //checksum error, discard message, do nothing
            }
            RS485_comms_state = RS485_WAIT;
            break;
        }
    }
}

void bus_master_timer_service(void) {
    if (last_sent_address == 0x10) {
        uint8 temp = REQ_SEND_TO_NODE2;
        xQueueSendFromISR(bus_queue, &temp, pdFALSE);
    } else if (last_sent_address == 0x20) {
        uint8 temp = REQ_SEND_TO_NODE1;
        xQueueSendFromISR(bus_queue, &temp, pdFALSE);
    } else {
        last_sent_address = 0x10;
    }
```

```
}

void Bus_UART_OnBlockReceived(LDD_TUserData *UserDataPtr) {
    //This funcion is called on a received character interrupt
    switch (RS485_comms_state) {
    case RS485_WAIT_REPLY:
        if (rx_buff[0] == 0xFF) {
            RS485_comms_state = RS485_RECEIVING;
            rx_index = 0;
        } else {
            RS485_comms_state = RS485_WAIT;
        }
        break;
    case RS485_RECEIVING:
        if (rx_index < RS485_MESSAGE_LENGTH - 1) {
            RS485_rx_arr[rx_index] = rx_buff[0];
            rx_index++;
        }
        if (rx_index == (RS485_MESSAGE_LENGTH - 1)) {
            uint8 temp = REQ_MESSAGE_RECEIVED;
            xQueueSendFromISR(bus_queue, &temp, pdFALSE);
            RS485_comms_state = RS485_RECEIVED;
        }
        break;
    case RS485_WAIT:
        break;
    }
    Bus_UART_ReceiveBlock(bus_master_handle, rx_buff, 1);
}

void Bus_UART_OnBreak(LDD_TUserData *UserDataPtr) {
    //Could be used as a start of frame character
}

void Bus_UART_OnTxComplete(LDD_TUserData *UserDataPtr) {
    RS485_tx_en_ClrVal();
    RS485_comms_state = RS485_WAIT_REPLY;
}
```

A.4.1.3. Telemetry communication

```
/*
 * Telemetry.c
 *
 *  Created on: Nov 11, 2013
 *      Author: Matteo Milandri
 */

#include "Cpu.h"
#include "Events.h"
#include "Sensors.h"
#include "BusMaster.h"
#include "Telemetry.h"
```

```
#include "GPS.h"
#include "RemoteController.h"
#include "quickRelease.h"
#include "CRC8.h"

#include "string.h"
//variables for the RF Comms
//uint8_t RF_comms_state = RF_COMMS_WAIT;
CommsState_t comm_state;
uint8_t RF_byte_timeout = 0;
uint32_t last_radio_message_time = 0;
uint8_t RF_rx_count = 0;
uint8_t RF_rx_char = 0xFF;
uint8_t RF_rx_arr[RF_DATA_LENGTH];
uint8_t RF_rx_temp[RF_DATA_LENGTH];

uint8_t RF_tx_arr[RF_MESSAGE_LENGTH];
uint8_t RF_tx_arr1sec[RF_MESSAGE_LENGTH];
SemaphoreHandle_t telemetry_rx_smphr;

uint8_t query_number = 0;
uint8_t tx_count;
uint16_t sent_count = 0; //used in the sendBlock() method
uint8_t PC_status = 0;
uint8_t rf_check_sum = 0;
uint8_t rfCommsError = 0;
uint8_t rfCommsCount = 0;

uint8_t error_char = 0;
uint8_t returnedError = 0;
uint8_t i = 0;
uint8_t reply_counter = 0;

COMMS_TDataState rfdriverState;
uint8_t rx_buff[1];

void telemetry_init(void) {

    rfdriverState.handle = RF_UART_Init(&rfdriverState);
    telemetry_rx_smphr = xSemaphoreCreateBinary();

    RF_tx_arr[0] = 0xFF;     //start
    RF_tx_arr[1] = 0x01;     //status

    comm_state = WAITING;
    query_number = 0;
    tx_count = 0;
    RF_UART_ReceiveBlock(rfdriverState.handle, &rx_buff, 1);
}

void telemetry_main_service(void) {
    telemetry_init();
```

```
for (;;) {
    if (xSemaphoreTake(telemetry_rx_smphr, 1000) == pdTRUE ) {
        if (RF_rx_char == calculateCRC8(RF_rx_temp, RF_DATA_LENGTH)) {
            uint8_t i = 0;
            for (i = 0; i < RF_MESSAGE_LENGTH; i++) {
                RF_rx_arr[i] = RF_rx_temp[i];
            }

            PC_status = RF_rx_arr[0];
            quickRelease_control(PC_status & 0x01);
            query_number = RF_rx_arr[30];
            if (xSemaphoreTake(node[0].data_access,20) == pdTRUE ) {
                node[0].duty_setpoint = RF_rx_arr[3];
                node[0].position_max = RF_rx_arr[5];
                node[0].position_min = RF_rx_arr[6];
                if (millis_count > 5000) {
                    node[0].mode = RF_rx_arr[7];
                }
                node[0].speed_max = RF_rx_arr[21];
                xSemaphoreGive(node[0].data_access);
            }
            if (xSemaphoreTake(node[1].data_access,20) == pdTRUE ) {

                node[1].duty_setpoint = RF_rx_arr[11];
                node[1].position_max = RF_rx_arr[13];
                node[1].position_min = RF_rx_arr[14];
                if (millis_count > 5000) {
                    node[1].mode = RF_rx_arr[15];
                }
                node[1].speed_max = RF_rx_arr[21];
                xSemaphoreGive(node[1].data_access);
            }
            safety_engage();
            last_radio_message_time = millis_count;
        } else { //Don't trust dodgy crc but reply.
            error_char |= ERROR_RF_RX;
        }

        //Reply
        RF_tx_arr[0] = 0xFF;            //start byte
        RF_tx_arr[1] = 0x01;            //status
        RF_tx_arr[2] = chan2.stick;
        RF_tx_arr[3] = chan3.stick;
        RF_tx_arr[27] = chan1.stick;

        //a bit pedantic but whatever
        if (xSemaphoreTake(node[0].data_access,20) == pdTRUE ) {
            RF_tx_arr[28] = node[0].position_setpoint;
            xSemaphoreGive(node[0].data_access);
        }
        if (xSemaphoreTake(node[1].data_access,20) == pdTRUE ) {
            RF_tx_arr[29] = node[1].position_setpoint;
```

```
            xSemaphoreGive(node[1].data_access);
        }

        if (xSemaphoreTake(nodeData[0].data_access,20) == pdTRUE ) {
            RF_tx_arr[4] = nodeData[0].position;
            RF_tx_arr[5] = nodeData[0].status;
            RF_tx_arr[6] = nodeData[0].current_hi;
            RF_tx_arr[7] = nodeData[0].current_lo;
            xSemaphoreGive(nodeData[0].data_access);
        }
        if (xSemaphoreTake(nodeData[1].data_access,20) == pdTRUE ) {
            RF_tx_arr[8] = nodeData[1].position;
            RF_tx_arr[9] = nodeData[1].status;
            RF_tx_arr[10] = nodeData[1].current_hi;
            RF_tx_arr[11] = nodeData[1].current_lo;
            xSemaphoreGive(nodeData[1].data_access);
        }

        if (xSemaphoreTake(sensors.data_access,20) == pdTRUE ) {
            memcpy(&RF_tx_arr[12], &sensors, 14);
            xSemaphoreGive(sensors.data_access);
        }
        RF_tx_arr[RF_MESSAGE_LENGTH - 1] = calculateCRC8((RF_tx_arr + 1),
                RF_MESSAGE_LENGTH - 2);

        RF_UART_SendBlock(rfdriverState.handle, RF_tx_arr,
            RF_MESSAGE_LENGTH);

        if (tx_count >= 10) {
            tx_count = 0;
            //while sending == true.
            vTaskDelay(30);
            //Send the 1Hz data
            RF_tx_arr[0] = 0xFE;
            RF_tx_arr[1] = 0x00;

            memcpy(&RF_tx_arr[2], &gps, 16);

            RF_tx_arr[18] = 0x01;
            RF_tx_arr[19] = 0x02;
            RF_tx_arr[20] = 0x03;

            RF_tx_arr[RF_MESSAGE_LENGTH - 1] = calculateCRC8((RF_tx_arr + 1),
                    RF_MESSAGE_LENGTH - 2);

            RF_UART_SendBlock(rfdriverState.handle,
                RF_tx_arr, RF_MESSAGE_LENGTH);
        }
        tx_count++;
        comm_state = WAITING;
        //RF_comms_state = RF_COMMS_WAIT;
    } else {
```

```
            //TIMED OUT - One Second
        }
    }
}

void RF_UART_OnBlockReceived(LDD_TUserData *UserDataPtr) {
    //Properties of the rf link: loop rate:100ms, bursts of 31 bytes, replied
    // with 31 bytes Only once the message is in and checked must the reply be
    // generated and sent. This should also handle timeouts between bytes in an
    // expected message: if the next byte in the packet is delayed by more
    // than 50ms, flag an error and restart receiving. <0xff><status>....<close>

    RF_rx_char = rx_buff[0];

    switch (comm_state) {
    case WAITING:
    case ERROR:
        if (RF_rx_char == 0xFF)
            comm_state = RECEIVING;
        RF_rx_count = 0;
        rf_check_sum = 0;
        break;
    case RECEIVING:
        if (RF_rx_count < RF_DATA_LENGTH) {
            RF_rx_temp[RF_rx_count] = RF_rx_char;
            RF_rx_count++;
        } else {
            if (telemetry_rx_smphr != NULL) {
                if (xSemaphoreGiveFromISR( telemetry_rx_smphr, pdFALSE)
                        == pdTRUE ) {
                    comm_state = RECEIVED;
                } else {
                    comm_state = ERROR;
                }
            }
        }
        break;
    case RECEIVED:
        comm_state = ERROR;
        break;
    }
    RF_byte_timeout = 0;

    RF_UART_ReceiveBlock(rfdriverState.handle, &rx_buff, 1);
}
```

A.4.1.4. Radio controller signal measurement

```
#include "Cpu.h"
#include "Events.h"
#include "Sensors.h"
#include "BusMaster.h"
#include "Telemetry.h"
```

```
#include "GPS.h"
#include "RemoteController.h"

/*
 * RemoteController.c
 *
 *  Created on: Nov 23, 2013
 *      Author: Matteo Milandri
 */

#define RIGHT_STICK_ONLY

Cap_mode capture_state;
LDD_TDeviceData *ctrl_handle;
RCData chan1; //left - not used for kite control actuation
RCData chan2; //right
RCData chan3; //delta(left-right movement of the right stick)

uint8 node1_sp;
uint8 node2_sp;

void rc_init_data(RCData * data) {
    data->value = 3400;
    data->raw = 3500;
    data->max = 0;
    data->min = 10000;
    data->stick = 0;
    data->flag = 0;
}

void radio_control_init(void) {
    ctrl_handle = RC_Control_Init(ctrl_handle);

    rc_init_data(&chan1);
    rc_init_data(&chan2);
    rc_init_data(&chan3);

    node1_sp = 0;
    node2_sp = 0;

    capture_state = WAIT;
}

void radio_control_main_service(void) {
    radio_control_init();
    for (;;) {
        if (chan2.flag) {
            if (chan2.value > chan2.max) {
                chan2.max = chan2.value;
            }
            if (chan2.value < chan2.min) {
                chan2.min = chan2.value - 1;
```

```
        }
        chan2.stick = (uint8_t) ((100 * (chan2.value - chan2.min))
            / (chan2.max - chan2.min));

        node1_sp = (uint8_t) (((uint16_t) (node[0].position_max
            - node[0].position_min) * (chan2.value - chan2.min))
            / (chan2.max - chan2.min)) + node[0].position_min;

        node2_sp = (uint8_t) (((uint16_t) (node[1].position_max
            - node[1].position_min) * (chan2.value - chan2.min)) /
            (chan2.max - chan2.min)) + node[1].position_min;

        if (chan3.stick == 50) {
        } else if (chan3.stick > 50) {
            node2_sp = node2_sp - (chan3.stick - 50);
            if (node2_sp < node[1].position_min) {
                node1_sp += (node[1].position_min - node2_sp);
                node2_sp = node[1].position_min;
            }
        } else {
            node1_sp = node1_sp - (50 - chan3.stick);
            if (node1_sp < node[0].position_min) {
                node2_sp += (node[0].position_min - node1_sp);
                node1_sp = node[0].position_min;
            }
        }

        chan2.flag = 0;
        if (millis_count > 6000) {
            if (xSemaphoreTake(node[0].data_access,20) == pdTRUE ) {
                node[0].position_setpoint = node1_sp;
                xSemaphoreGive(node[0].data_access);
            }
            if (xSemaphoreTake(node[1].data_access,20) == pdTRUE ) {
                node[1].position_setpoint = node2_sp;
                xSemaphoreGive(node[1].data_access);
            }
        }
    }

    if (chan1.flag) {
        if (chan1.value > chan1.max) {
            chan1.max = chan1.value;
        }
        if (chan1.value < chan1.min) {
            chan1.min = chan1.value - 1;
        }
        chan1.stick = (uint8_t) ((100 * (chan1.value - chan1.min))
            / (chan1.max - chan1.min));
        chan1.flag = 0;
    }
```

```
        if (chan3.flag) {
            if (chan3.value > chan3.max) {
                chan3.max = chan3.value;
            }
            if (chan3.value < chan3.min) {
                chan3.min = chan3.value - 1;
            }
            chan3.stick = (uint8_t) ((100 * (chan3.value - chan3.min))
                / (chan3.max - chan3.min));
            chan3.flag = 0;
        }
    }
}

void RC_Control_OnChannel0(LDD_TUserData *UserDataPtr) {

    /*
     * Channel 0, on either edge.
     *
     * Uses the sequence of edges, two left, two right, long delay
     * to determine the length of each of the pulses.
     * */

    RC_Control_GetCaptureValue(ctrl_handle, 0, &(chan1.raw));

    switch (capture_state) {
    case WAIT:
        capture_state = LEFT_HIGH;
        RC_Control_ResetCounter(ctrl_handle);
        break;
    case RIGHT_HIGH:
        capture_state = WAIT;
        break;
    case LEFT_HIGH:
        if (chan1.raw > 10000) {
            capture_state = WAIT;
        } else {
            chan1.value = chan1.raw;
            capture_state = WAIT_BETWEEN;
            chan1.flag = 1;
        }
        break;
    case WAIT_BETWEEN:
        capture_state = WAIT;
        break;
    }
}

/*
 * on either a rising or falling edge of timer channel 1 input
 */
void RC_Control_OnChannel1(LDD_TUserData *UserDataPtr) {
```

```
    // Channel 1, on either edge.
    RC_Control_GetCaptureValue(ctrl_handle, 1, &(chan2.raw));

    switch (capture_state) {
    case WAIT:
        capture_state = WAIT;
        break;
    case RIGHT_HIGH:
        if (chan2.raw > 15000) {
            capture_state = WAIT;
        } else {
            chan2.value = chan2.raw - chan3.raw;
            capture_state = WAIT;
            chan2.flag = 1;
        }
        break;
    case LEFT_HIGH:
        capture_state = WAIT;
        break;
    case WAIT_BETWEEN:
        if (chan1.raw > 12000) {
            capture_state = WAIT;
        } else {
            chan3.raw = chan2.raw;
            capture_state = RIGHT_HIGH;
            chan3.value = chan2.raw - chan1.raw;
            if (chan3.value < 6000) {
                chan3.flag = 1;
            }
        }
        break;
    }
}
```

A.4.1.5. GPS receiver text parsing

```
/*
 * GPS.c
 *  Created on: Apr 10, 2014
 *      Author: Matteo
 */
LDD_TDeviceData *gps_handle;

char gps_rx_buff[1];
char * token;
uint8 gps_rx_index;
char gps_rx_buffer[BUFFER_LEN];
GPS_state_t gps_state;
GPS_Data_t gps;

void init_GPS(void) {
    gps_handle = GPS_UART_Init(gps_handle);
    GPS_UART_ReceiveBlock(gps_handle, gps_rx_buff, 1);
```

```
    gps_state = GPS_RECEIVING;
}

/*
 * Accepts messages of this format
 * $GPGGA,194348.996,3357.4993,S,01827.6106,E,0,00,0.0,170.55,M,,,,*28
 * */
void run_GPS(void) {
    init_GPS();
    for (;;) {
        vTaskDelay(10);
        if (gps_state == GPS_PROCESSING) {
            token = strtok(gps_rx_buffer, ",");
            if (token != NULL)
                if (strcmp(token, "GPGGA") == 0) {
                    //Time
                    token = strtok(NULL, ",");
                    if (strlen(token) == 10) {
                        token[6] = 0;
                        gps.time = atoi(token);
                    }
                    //Latitude
                    token = strtok(NULL, ",");
                    if (strlen(token) == 9) {
                        memmove((token + 4), (token + 5), 5);
                        gps.latitude = atoi(token);
                    }
                    token = strtok(NULL, ",");
                    if (token[0] == 'S')
                        gps.latitude = -gps.latitude;
                    //Longitude
                    token = strtok(NULL, ",");
                    if (strlen(token) == 10) {
                        memmove((token + 5), (token + 6), 5);
                        gps.longitude = atoi(token);
                    }
                    token = strtok(NULL, ",");
                    if (token[0] == 'W')
                        gps.longitude = -gps.longitude;
                    //Other Stuff
                    token = strtok(NULL, ",");
                    token = strtok(NULL, ",");
                    token = strtok(NULL, ",");

                    //Altitude
                    token = strtok(NULL, ",");
                    char * ind = strchr(token,'.');
                    if (ind != NULL){
                        *ind = 0;
                        gps.altitude = atoi(token);
                    }
                }
```

```
            gps_state = GPS_IDLE;
        }
    }
}

/*
 * On receipt of each character from the GPS reciever
 */
void GPS_UART_OnBlockReceived(LDD_TUserData *UserDataPtr) {
    uint8 temp = gps_rx_buff[0];
    if (temp == '$' && gps_state == GPS_IDLE) {
        gps_state = GPS_RECEIVING;
        gps_rx_index = 0;
    } else if (gps_state == GPS_RECEIVING) {
        if (temp == '\n' || temp == '\r') {
            if (gps_rx_buffer[3] == 'G') {
                gps_rx_buffer[gps_rx_index] = 0; //terminate
                gps_state = GPS_PROCESSING;
            } else
                gps_state = GPS_IDLE; //wrong message
            gps_rx_index = 0;
        } else if (gps_rx_index < BUFFER_LEN) {
            gps_rx_buffer[gps_rx_index] = temp;
            gps_rx_index++;
        } else {
            gps_rx_index = 0;
            gps_state = GPS_IDLE; //buffer overrun
        }
    }
    GPS_UART_ReceiveBlock(gps_handle, gps_rx_buff, 1);
}
```

A.4.1.6. Sensor polling and ADC measurements

```
/*
 * ADXL345.c
 *
 *  Created on: Aug 17, 2012
 *      Author: Matteo Milandri
 */
#include "Cpu.h"
#include "Events.h"
#include "Sensors.h"
#include "BusMaster.h"
#include "I2C2.h"
#include "Telemetry.h"
#include "GPS.h"
#include "RemoteController.h"

//IIC VARIABLES
static IIC_TDataState deviceData;
uint8 err1 = 0;
uint8 err2 = 0;
```

```
uint8 err3 = 0;
uint8 last_address;

//BAROMETER
Barometer_raw_t baro_raw_data;
Barometer_Config_t baro_conf;

//ANALOG
LDD_TDeviceData * analog_handle;
LDD_ADC_TSample sample_handle;
uint16 rawValues[1];
uint8 measure_complete = 0;

//ALL SENSORS
SensorData_t sensors;

/*
 * Writes data to the specified IIC device's register
 * */
uint8 iic_write_reg(uint8 device_addr, uint8 register_addr, uint8 value) {
    uint8_t buf[2], res;
    if (last_address != device_addr) {
        last_address = device_addr;
        I2C2_SelectSlaveDevice(deviceData.handle, LDD_I2C_ADDRTYPE_7BITS,
                (LDD_I2C_TAddr) device_addr);
    }

    buf[0] = register_addr;
    buf[1] = value;
// Send OutData (3 bytes with address) on the I2C bus
// and generates a stop condition to end transmission
    res = I2C2_MasterSendBlock(deviceData.handle, &buf, 2U,
        LDD_I2C_SEND_STOP);
    if (res != ERR_OK) {
        return ERR_FAILED;
    }
    uint32 start = millis_count;
    while (!deviceData.dataTransmittedFlg) {
        vTaskDelay(1);
        if ((millis_count - start) > 50) {
            deviceData.dataTransmittedFlg = FALSE;
            return ERR_BUSOFF;
        }
    } /* Wait until date is sent */
    deviceData.dataTransmittedFlg = FALSE;
    return ERR_OK;
}

/*
 * Requests data from the specified IIC device
 * */
uint8 iic_read_reg(uint8 device_addr, uint8 register_addr, uint8_t *data,
```

```
        uint16 data_size) {
    uint8_t res;
    if (last_address != device_addr) {
        last_address = device_addr;
        I2C2_SelectSlaveDevice(deviceData.handle, LDD_I2C_ADDRTYPE_7BITS,
                (LDD_I2C_TAddr) device_addr);
    }

// Send I2C address plus register address to the I2C bus without a stop condition
    res = I2C2_MasterSendBlock(deviceData.handle, &register_addr, 1U,
        LDD_I2C_NO_SEND_STOP);
    if (res != ERR_OK) {
        return ERR_FAILED;
    }
    uint32 start = millis_count;
    while (!deviceData.dataTransmittedFlg) {
        vTaskDelay(1);
        if ((millis_count - start) > 50) {
            deviceData.dataTransmittedFlg = FALSE;
            return ERR_BUSOFF;
        }
    } /* Wait until data is sent */
    deviceData.dataTransmittedFlg = FALSE;

// Receive 1 byte from the I2C bus and generates a stop condition to end tx
    res = I2C2_MasterReceiveBlock(deviceData.handle, data, data_size,
        LDD_I2C_SEND_STOP);
    if (res != ERR_OK) {
        return ERR_FAILED;
    }
    while (!deviceData.dataReceivedFlg) {
    } /* Wait until data is received received */
    deviceData.dataReceivedFlg = FALSE;
    return ERR_OK;
}

/*
 * Initialise the Accelerometer
 * */
uint8 ADXL345_Init() {
    uint8 err;
    // F_READ: Fast read mode, data format limited to single byte
    //(auto increment counter will skip LSB)
    // ACTIVE: Full scale selection
    err = iic_write_reg(ACCEL_ADDRESS, ACCEL_DATA_FORMAT, ACCEL_DATA_FORMAT_MSG);
    vTaskDelay(5);
    err |= iic_write_reg(ACCEL_ADDRESS, ACCEL_PWR_CTL, ACCEL_PWR_CTL_MSG);
    return err;
}

/*
 * Initialise the barometer
```

```
 * */
uint8 BMP085_Init() {
    uint8 err;
    err = iic_read_reg(BMP085_I2CADDR, BMP085_CAL_AC1, (uint8_t *)
        &baro_conf, 22);
    vTaskDelay(5);
    return err;
}

uint8 BMP085_get_raw_values() {
    uint8 err;
    //uint8 oversampling = 0;   //can be 1 2 3
    err = iic_write_reg(BMP085_I2CADDR, BMP085_CONTROL, BMP085_READTEMPCMD);
    vTaskDelay(5);
    err |= iic_read_reg(BMP085_I2CADDR, BMP085_TEMPDATA, (uint8_t *)
        &baro_raw_data.temperature, 2);
    vTaskDelay(5);
    err |= iic_write_reg(BMP085_I2CADDR, BMP085_CONTROL, BMP085_READPRESSURECMD);
    vTaskDelay(10);
    err |= iic_read_reg(BMP085_I2CADDR, BMP085_PRESSUREDATA, (uint8_t *)
        &baro_raw_data.pressure, 2);
    vTaskDelay(5);

    baro_raw_data.temperature = ((baro_raw_data.temperature >> 8) & 0x00FF)
            | ((baro_raw_data.temperature << 8) & 0xFF00);
    baro_raw_data.pressure = ((baro_raw_data.pressure >> 8) & 0x00FF)
            | ((baro_raw_data.pressure << 8) & 0xFF00);
    //baro_raw_data.pressure >>= (8 - oversampling);
    return err;
}

void BMP085_process_raw_values() {
    int32 x1, x2, x3, b3, b5, b6, p, t;
    uint32 b4, b7;

    x1 = ((baro_raw_data.temperature - baro_conf.AC6) * baro_conf.AC5) >> 15;
    x2 = ((int32) baro_conf.MC << 11) / (x1 + baro_conf.MD);
    b5 = x1 + x2;
    t = (b5 + 8) >> 4;
    //temperature calculated, now for pressure
    b6 = b5 - 4000;
    x1 = (baro_conf.B2 * ((b6 * b6) >> 12)) >> 11;
    x2 = (baro_conf.AC2 * b6) >> 11;
    x3 = x1 + x2;
    b3 = ((int32) baro_conf.AC1 * 4 + x3 + 2) >> 2;
    x1 = (baro_conf.AC3 * b6) >> 13;
    x2 = (baro_conf.B1 * ((b6 * b6) >> 12)) >> 16;
    x3 = ((x1 + x2) + 2) >> 2;
    b4 = (baro_conf.AC4 * (uint32) (x3 + 32768)) >> 15;
    b7 = ((uint32) baro_raw_data.pressure - b3) * (50000);
    p = b7 < 0x80000000 ? (b7 * 2) / b4 : (b7 / b4) * 2;
```

```
    x1 = (p >> 8) * (p >> 8);
    x1 = (x1 * 3038) >> 16;
    x2 = (-7357 * p) >> 16;
    p = p + ((x1 + x2 + 3791) >> 4);

    if (xSemaphoreTake(sensors.data_access,50) == pdTRUE ) {
        sensors.barometer.temperature = (int16) t;
        sensors.barometer.pressure = p;
        xSemaphoreGive(sensors.data_access);
    }
}

void analog_init(void) {
    analog_handle = AD1_Init(analog_handle);
    AD1_CreateSampleGroup(analog_handle, &sample_handle, 1);
    measure_complete = 2;
}

void analog_50ms_trigger(void) {
    if (measure_complete == 2) {
        AD1_StartSingleMeasurement(analog_handle);
    }
}

/*
 * The task main method.
 */
void run_sensors(void) {
    sensors.data_access = xSemaphoreCreateBinary();
    xSemaphoreGive(sensors.data_access);

    vTaskDelay(100);
    deviceData.handle = I2C2_Init(&deviceData);
    err1 = ADXL345_Init();
    err2 = BMP085_Init();
    analog_init();

    for (;;) {
        vTaskDelay(60);
        if (xSemaphoreTake(sensors.data_access,50) == pdTRUE ) {
            err3 = iic_read_reg(ACCEL_ADDRESS, ACCEL_X_LSB,
                (uint8_t*) &sensors.accelerometer, 6);
            xSemaphoreGive(sensors.data_access);
        }
        err3 |= BMP085_get_raw_values();
        vTaskDelay(10);
        BMP085_process_raw_values();

        if (measure_complete == 1) {
            if (xSemaphoreTake(sensors.data_access,50) == pdTRUE ) {
                sensors.battery_voltage = rawValues[0] / 2.4856;
                xSemaphoreGive(sensors.data_access);
```

```
            }
            measure_complete = 2;
        }
    }
}

/* Once the data has been successfully sent */
void I2C2_OnMasterBlockSent(LDD_TUserData *UserDataPtr) {
    IIC_TDataState *ptr = (IIC_TDataState*) UserDataPtr;
    ptr->dataTransmittedFlg = TRUE;
}

/* Interrupt caused by data being received from an IIC device */
void I2C2_OnMasterBlockReceived(LDD_TUserData *UserDataPtr) {
    IIC_TDataState *ptr = (IIC_TDataState*) UserDataPtr;
    ptr->dataReceivedFlg = TRUE;
}

void AD1_OnMeasurementComplete(LDD_TUserData *UserDataPtr) {
    AD1_GetMeasuredValues(analog_handle, rawValues);
    measure_complete = 1;
}
```

A.4.2. Motor drive node firmware

The code was developed using the Processor Expert peripheral suite provided by Freescale. This suite generates peripheral drivers that the application invokes to write to and read from hardware registers. The generated code is not presented here but can be found in the software on the enclosed CD.

A.4.2.1. Main

```
#include "Cpu.h"
#include "Events.h"
#include "perFunctions.h"

/*
 This program is to:
 1 Communicate over RS485
 2 Output to PWM a specific duty
 3 Input position and current via ADC's
 4 Calculate duty based on some commmand and control law

 Pinning:
 1   Reset
 2   PTC0/TXD2     RS485 DI
 3   PTC1/RXD2     RS485 RO
 6   PTC4          RS485 !RE
 7   PTC5          RS485 DE
 18  PTD0/TPM1Ch0  H-U2 INB P-chan
 19  PTD1/TPM1Ch1  H-U2 INA N-chan
 20  PTD3/TPM2Ch0  H-U1 INB P-chan
 21  PTD4/TPM2Ch1  H-U1 INA N-chan
 22  PTB0/AD1P0    Current_!Fault
 23  PTB1/AD1P1    Current_Out
 24  PTB2/AD1P2    Header1
 25  PTB3/AD1P3    Header2
 42  Clock
 43  Clock
 44  BKGND
 */

void main(void) {
    PE_low_level_init();
    init();
    setDirection(0);
    for (;;) {
        mainLoop();
    }
}
```

A.4.2.2. Events

```
/*
 * events.h
 */
#define COMMS_WAIT                1
#define COMMS_MESSAGE_RECEIVING   2
#define COMMS_MESSAGE_RECEIVED    3
#define COMMS_DELAY1              4
#define COMMS_DELAY2              5
#define MODE_SPD_CTRL             0
#define MODE_POS_CTRL             1
#define MESSAGE_LENGTH            8
#define TIMEOUT_MILLIS            220
#define PULL_SLOW                 100
#define PULL_FAST                 50
#define PULL_FASTER               2
#define RELEASE_SLOW              150
#define RELEASE_FAST              200
#define RELEASE_FASTER            253
#define DONT_MOVE                 127

void init(void);
void mainLoop(void);

/*
 * Events.c
 */
#include "Cpu.h"
#include "perFunctions.h"
#include "CRC8.h"

volatile uint8_t us_cnt;
extern uint32_t msClock;
uint8_t position;
uint16_t current;
uint8_t status;
uint8_t RS485_rx_char;
uint8_t comms_state;
uint8_t comms_timeout;
uint8_t rx_byte_count;
uint8_t rx_message[MESSAGE_LENGTH - 1];
uint8_t tx_message[MESSAGE_LENGTH];
uint8_t mode;
uint8_t duty_setpoint;
uint8_t position_setpoint;
uint8_t position_max;
uint8_t position_min;
uint8_t pull_speed;
uint8_t release_speed;
uint8_t queue_control_calc;
uint8_t error_code;
```

```
uint8_t sentNo;

void init(void) {
    us_cnt = 0;
    msClock = 0;
    position = 0;
    current = 0;
    RS485_rx_char = 0;
    comms_state = COMMS_WAIT;
    comms_timeout = 0;
    status = 0;
    rx_byte_count = 0;
    mode = 0;
    duty_setpoint = 0;
    position_setpoint = 0;
    position_max = 0;
    position_min = 0;
    queue_control_calc = 0;
    error_code = 0;
    pull_speed = PULL_FAST;
    release_speed = RELEASE_FAST;
}

/**
 * Idle processes.
 */
void mainLoop(void) {
    //process message and reply
    if (comms_state == COMMS_MESSAGE_RECEIVED) {
        uint8_t calculatedCRC8 = calculateCRC8(rx_message, MESSAGE_LENGTH - 2);
        uint8_t receivedCRC8 = rx_message[MESSAGE_LENGTH - 2];
        if (calculatedCRC8 == receivedCRC8) {
            if (rx_message[0] == 0) { //this is a E-stop message.
                setDirection(DIR_SAFE);
                mode = 0;
            } else if (Address_GetVal() == (rx_message[0] & 0xF0) >> 4) {
                mode = rx_message[0] & 0x0F; //only take the first four bits
                comms_timeout = 0;

                duty_setpoint = rx_message[1];
                position_setpoint = rx_message[2];
                position_max = rx_message[3];
                position_min = rx_message[4];

                pull_speed = 127 - (rx_message[5] >> 1);
                release_speed = 127 + (rx_message[5] >> 1);

                if (mode == 0) {
                    setSpeedDirection(duty_setpoint);
                }

                tx_message[0] = 0xFF;
```

```
                tx_message[1] = (Address_GetVal() << 4) | (status & 0x0F);
                tx_message[2] = position;
                tx_message[3] = (uint8_t) (current >> 8);
                tx_message[4] = (uint8_t) current;
                tx_message[5] = duty_sp;
                tx_message[6] = 0;
                tx_message[7] = calculateCRC8((tx_message + 1), 6);

                DE_SetVal();
                nRE_SetVal();
                RS485_SendBlock(tx_message, MESSAGE_LENGTH, &sentNo);
            } //end of for me
        }
        comms_state = COMMS_WAIT; //Ignore new messages until done.
    } //end of received

    //control calculation, queued once every millisecond
    if (queue_control_calc) {
        queue_control_calc = 0;
        //is the position sensor value sensible / am I within my limits?
        if ((position > (position_max + 2)) || (position < (position_min - 2))) {
            setSpeedDirection(DONT_MOVE);
        } else if (mode == 1) {//Yes, continue...
            if (position_setpoint > position) { //wind in
                if (position < position_max) { //within limits
                    if ((position_setpoint - position) > 5) {
                        setSpeedDirection(pull_speed);
                    } else if ((position_setpoint - position) > 1) {
                        setSpeedDirection(PULL_SLOW);
                    } else {
                        setSpeedDirection(DONT_MOVE);
                    }
                }
            } else { //wind out
                if (position > position_min) { //within limits
                    if ((position - position_setpoint) > 5) {
                        setSpeedDirection(release_speed);
                    } else if ((position - position_setpoint) > 1) {
                        setSpeedDirection(RELEASE_SLOW);
                    } else {
                        setSpeedDirection(DONT_MOVE);
                    }
                }
            }
        }
    }
}

void Analog_OnEnd(void) {
    error_code = Analog_GetChanValue8(0, &position);
    error_code = Analog_GetChanValue16(1, &current);
}
```

```
void RS485_OnError(void) {
    error_1 = RS485_GetError(&error_2);
}

void RS485_OnRxChar(void) {
    error_code = RS485_RecvChar(&RS485_rx_char);
    switch (comms_state) {
    case COMMS_WAIT:
        if (RS485_rx_char == 0xFF) {
            comms_state = COMMS_MESSAGE_RECEIVING;
            rx_byte_count = 0;
        }
        break;
    case COMMS_MESSAGE_RECEIVING:
        if (rx_byte_count < MESSAGE_LENGTH - 1) {
            rx_message[rx_byte_count] = RS485_rx_char;
            rx_byte_count++;
        }
        if (rx_byte_count >= MESSAGE_LENGTH - 1) {
            comms_state = COMMS_DELAY1;
        }
        break;
    case COMMS_DELAY1:
    case COMMS_DELAY2:
    case COMMS_MESSAGE_RECEIVED:
        break;
    default:
        comms_state = COMMS_WAIT;
    }
}

void RS485_OnTxComplete(void) {
    DE_ClrVal();
    nRE_ClrVal();
}

void PWM1_OnEnd(void) {
    if (us_cnt >= 25) { // every millisecond
        us_cnt = 0; //reset counter
        error_code = Analog_Measure(0); //start another measurement
        msClock++;
        comms_timeout++;
        if (comms_timeout > TIMEOUT_MILLIS) {
            setDirection(DIR_SAFE);
            mode = 0;
            duty_setpoint = DONT_MOVE;
        }
        //delay for between 1 and 2 ms before replying.
        if (comms_state == COMMS_DELAY2) {
            comms_state = COMMS_MESSAGE_RECEIVED;
        } else if (comms_state == COMMS_DELAY1) {
```

```
            comms_state = COMMS_DELAY2;
        }
        queue_control_calc = 1;
    }
    us_cnt++;
}
```

A.4.2.3. perFunctions

```
/* perFunctions.h
 */
#define DIR_SAFE    0
#define DIR_FORWARD 1
#define DIR_REVERSE 2

extern int8_t duty_sp;

void setSpeed(unsigned char speed_value);
void setSpeedDirection(unsigned char temp_speed);
void setDirection(unsigned char dir_value);

/* perFunctions.c
 *  Created on: Aug 30, 2012
 *      Author: Matteo Milandri
 */

#include "perFunctions.h"
#include "CRC8.h"

uint8_t direction;
uint8_t returned;
int8_t duty_sp;

void setDuty(int8_t duty) {
    uint8_t temp_speed;
    duty_sp = duty;

    if (duty >= 0) {
        temp_speed = duty * 2;
        setDirection(DIR_FORWARD);
    } else {
        temp_speed = (-duty) * 2;
        setDirection(DIR_REVERSE);
    }
    setSpeed(temp_speed);
}
/*
 * Note this is where the PWM values get updated. Only here.
 * */
void setSpeed(uint8 speed_value) {
    uint8 speed = 255 - speed_value;
    if (direction == DIR_FORWARD) {
        returned = PWM1_SetRatio8(duty);
```

```
    } else if (direction == DIR_REVERSE) {
        returned = PWM2_SetRatio8(duty);
    } else {
        returned = PWM1_SetRatio8(255);
        returned = PWM2_SetRatio8(255);
    }
}

/**
 * This is a single value motor duty control value input.
 * */
void setSpeedDirection(uint8_t temp_s) {
    unsigned char temp_speed = temp_s;
    if (temp_speed > 127) {
        temp_speed = temp_speed - 128;
        temp_speed = temp_speed * 2;
        setDirection(DIR_FORWARD);
    } else {
        temp_speed = 127 - temp_speed;
        temp_speed = temp_speed * 2;
        setDirection(DIR_REVERSE);
    }
    setSpeed(temp_speed);
}

void setDirection(uint8_t dir_value) {
    direction = dir_value;

    if (dir_value == DIR_FORWARD) {
        H1_Hi_SetVal();      //turn H1 hi on - for forward
        H2_Hi_ClrVal();      //Turn H2 hi off - for forward

        returned = PWM1_SetRatio8(255);      //zero speed to start
        //zero reverse speed for forward(and to avoid shoot-through)
        returned = PWM2_SetRatio8(255);

    } else if (dir_value == DIR_REVERSE) {
        H1_Hi_ClrVal();      //turn H1 hi off - for reverse
        H2_Hi_SetVal();      //Turn H2 hi on - for reverse

        returned = PWM1_SetRatio8(255);      //zero forward speed
        returned = PWM2_SetRatio8(255);      //zero the speed
    } else {    //DIR_SAFE
        H1_Hi_ClrVal(); //turn H1 hi off - for reverse
        H2_Hi_ClrVal(); //Turn H2 hi off - for forward

        returned = PWM1_SetRatio8(255); //not equal to 255 if going forward
        returned = PWM2_SetRatio8(255); //not equal to 255 if going in reverse
    }
}
```