Development of an Autonomous, Tethered and Submersible Data Buoy

Final year project
GH7P - Artificial Intelligence and Robotics

Author:
Tom Blanchard
ttb7@aber.ac.uk

Supervisor:
Dr. Mark Neal
mjn@aber.ac.uk

Contents

Acknowledgements

I would like to thank my supervisor Dr Mark Neal who has been of immense help throughout the project. I am especially grateful for the time he has spent helping me test my project at various times this year. I would also like to thank Colin Sauzé and Michael Clarke both of whom were great sources of information, tips and general technical knowledge. Ian Izet has been a constant source of advice and help on all things electrical and mechanical, his knowledge has saved me a lot of time and hassle over the last year. Dr Alun Hubbard, Dr Jason Box and Dr Richard Bates have all been very patient in explaining what it is they wanted from the project, their advice and knowledge has been of immense help.

I would also like to thank all of the other final year students, especially Alex Kruzewski, who have spent far too much time in the DSL over the last few months. They have all helped make the countless hours spent in there a lot more enjoyable. Lastly I would like to thank my parents for their continual support throughout my degree.

Abstract

The structure and dynamics of glaciers is a very active research topic in geophysics. Scientists are currently working on understanding the processes that control glaciers and their effect on the environment. Existing techniques used to acquire water quality data near glaciers are considered too dangerous or return a limited amount of useful data. In this project the author sets out to create an autonomous tethered data buoy that can take the measurements desired by glaciologists, at multiple depths, for long periods of time. The system is designed to be as simple as possible without compromising data quality.

1 Introduction

The problem of studying glaciers has always been a complicated one. The glacier itself can move several meters a day and the underlying processes that control the glacier's movements are not yet fully understood. Of the many complex processes, the one stands out as perhaps the most interesting and complex, is the calving of glaciers. Calving[1] is the geoscientific name for the creation of icebergs from a glacier and it occurs on the leading edge that intersects a body of water[2]. Calving events can vary from a few small boulder sized pieces of ice being cast off, to pieces the size of cities[19]. Even small calving events can be very dangerous. This is part of the process that causes glaciers to advance or retreat. A year with many large calving events will cause the glacier to "retreat", a year with only small or few calving events will cause it to "advance" due to the natural forward movement of the glacier. The current theory is that the rapid draining of melt-water lakes, situated on the glacier itself, can cause a significant amount of stress to be exerted upon the glacier, which in turn causes calving events. These lakes often drain through existing englacial streams[3] which get significantly larger in the draining process leading to weak points in the glacier. One way to examine this theory is to study the draining lakes themselves, another is to study the water along the leading edge of the glacier. It is thought that by looking at various water quality measurements at the calving front it can be determined when a lake has drained. The author will be focusing on trying to collect this data by designing, building and testing an autonomous tethered and submersible data buoy.

There are currently several different methods of taking water quality measurements at the calving front of a glacier, each of which provides data which helps scientists better understand the calving process. By far the most commonly used instrument is something called a CTD, which stands for Conductivity, Temperature and Depth. This is a waterproof sensor package containing sensors for measuring, as the name suggests, conductivity, temperature and depth[4] that is turned on and lowered to the desired depth to take measurements.

[1]Also known as iceberg calving.

[2]Usually fjords, lakes or the sea.

[3]Streams that flow through and under the glacier.

[4]Modern CTDs often have other sensors fitted to allow them to measure things like dissolved O_2 and Chlorophyll content.

Figure 1.1: Lille glacier in Greenland is one of the glaciers being studied by glaciologists and is a possible deployment location for the system described in this project.

1.1 Current CTD Deployment Methods

There are several ways of deploying a CTD, one is to take a boat to the desired location, manoeuvre it into position and lower the CTD down to the required depth before pulling it back onto the boat. This requires long cables (often several hundred meters) and an on-board winch. Getting close enough to the glacier to take useful measurements is often very dangerous, calving events are currently impossible to predict and can occur very quickly. The chances of surviving a medium to large event even if only 100m away are slim. On top of this, data can only be collected while there is someone present to do so and renting or running a boat for long periods of time can be very expensive.

The second method is to attach the CTD to a tethered buoy and leave it sampling, this is an approach often used in other parts of the world. Providing that the buoy can relay this data (using Satellite, GSM networks or an Acoustic Modem) it is possible to get real-time data without a person being present. These buoys can be designed to run for several years and so have a high longevity. The downside is that being on the surface of the water, near a calving glacier, is as dangerous a place to be for a buoy as for a human. There is a possibility that the glacier could calve directly onto the buoy, most likely destroying it completely and during the winter the surface of the sea freezes solid which would crush the buoy.

The last method is to fit the CTD to an autonomous craft, like an underwater glider which holds station at the leading edge of the glacier, taking measurements and ascending to transmit data. Whilst this allows for surveying multiple locations along the leading edge (which is very useful) gliders are notoriously difficult to control and usually do not have sufficient power to last more than a few months at best.

Ideally, Glaciologists would like a device that could take measurements along the leading edge of a glacier at a variety of depths for long periods of time and transmit that data back in real time. Being slightly more realistic, several static devices could be deployed along the front of the glacier to perform these measurements. There already exists a type of submersible buoy called an Argo float which can submerge to take measurements. However, the buoys cost around $15,000 and once running costs are factored in this figure almost doubles. They are also not designed to be deployed in an environment containing large amounts of ice. This project aims to make an autonomous submersible data buoy that is relatively similar in cost, simple and inexpensive to run. It will be tethered[5] in position and will control its buoyancy to ascend and descend to certain depths to take measurements with a CTD. It will also attempt to transmit the data it has recorded in as near to real-time as possible, which it will do by periodically rising to the surface. It will be important to check if the surface is actually clear of debris before surfacing. The buoy is intended to be deployed in front of glaciers where there will be icebergs, brash ice and solid ice in the winter which could damage the buoy should it attempt to surface. During winter it is expected that the buoy will not surface, and therefore not transmit any data, for 4 to 6 months.

[5]To keep it in one position.

2 Background

2.1 Argo Float Network

The Argo Float network is a global array of free drifting floats. Deployment started in 2000 and by 2007 had fully been completed, with over 3000 floats deployed (see fig 2.1)[16]. The floats measure temperature, salinity and depth as well as the velocity of the upper ocean currents and relay the data back through the Systéme Argos location and data transmission system. The data, some 100,000 profiles a year, is then made publicly available. The

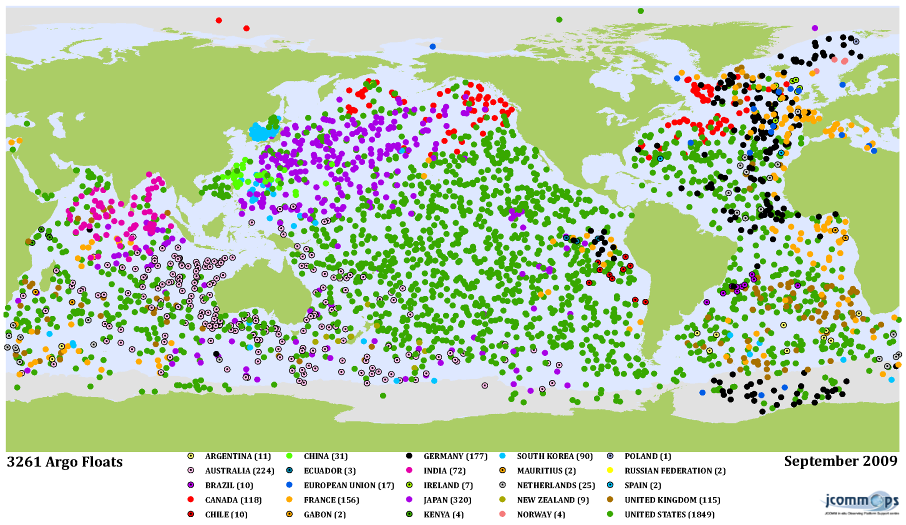


Figure 2.1: The Argo network as of September 2009, now comprised of around 3200 floats from many different countries. Also worth noting is how few floats there are near either polar region.

floats have two different profiling modes; 'simple' and 'park and profile'. In the simple mode the float descends to 2000m, where it then starts sampling the water. It stays at this depth for approximately 9 days before starting to ascend slowly over the course of 6hrs to the surface. Once at the surface it transmits the data it has collected and submerges again. When configured for the park and profile mode, the float descends to a recommended depth of 1000m where it 'cruises' for approximately 9 days. It then descends further

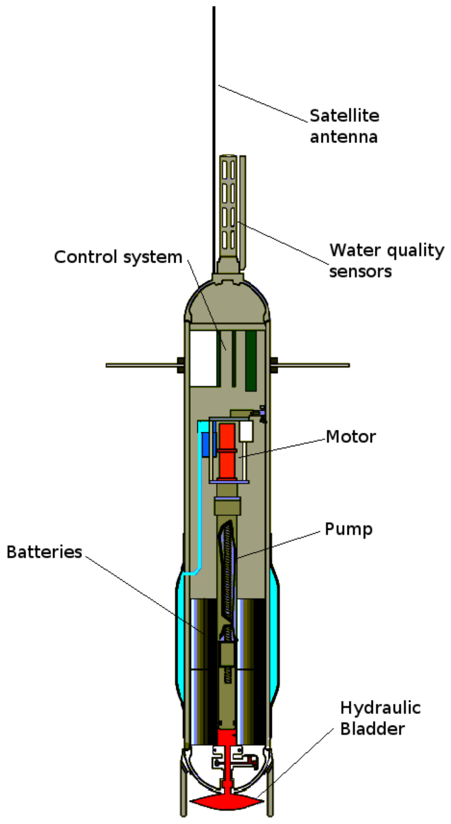


Figure 2.2: A cross-section of an Argo Float.

to 2000m where it starts to record data and slowly rise over the course of 6hrs to the surface to transmit the data. Of the 3000 floats in service 70% sample from depths greater than 1500m and a further 20% between 1000m and 1500m. Deployment is currently still ongoing in an effort to achieve global ocean coverage.

The floats themselves are battery powered, autonomous, are neutrally buoyant at their 'cruising' depth and have expected lifetimes of 3-4 years. A cross-sectional diagram is shown in fig 2.2. They are also unguided, drifting with the ocean currents. These drifting patterns are considered useful data as the float's position (accurate to 100m) can be tracked by the Argo Network's satellite based tracking and communication system. For an accurate position information and error free transmission of data the floats need to spend 6-12hrs on the surface. Buoyancy control is achieved by pumping hydraulic fluid into, or out of, an external bladder to change the floats volume thus creating a change in buoyancy.[10]

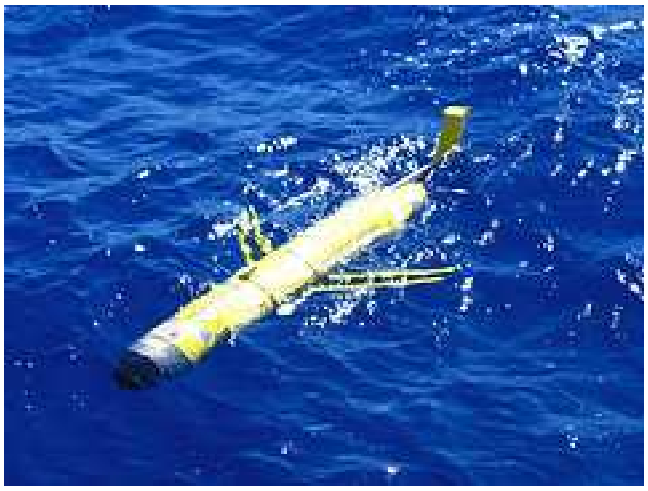


Figure 2.3: The Slocam glider during a test in the Sargasso Sea, the "wings" can be seen in a neutral position keeping the glider surfaced and level.(Image courtesy of Rutgers)

2.2 Gliders

Gliders, in this field, are a kind of autonomous submersible that 'glide' underwater. Their method of propulsion comes from changing their buoyancy and are designed so that when ascending or descending some of the vertical motion is converted into forwards motion which is accomplished using small wings. This results in a slow but very low power movement forward, which while slower than conventional AUVs (autonomous underwater vehicle) offers much better range and mission duration, often extending into months and thousands of kilometres of range. They typically operate at depths of around 1000m-2000m, rise over the course of several hours and can travel significant distances horizontally in the process. Like the Argo Floats they sample things like depth, temperature and conductivity (which allows salinity to be calculated) and when surfaced they transmit their data via satellite. One benefit of using a glider over a float is that they can be guided to target locations. This coupled with satellite connectivity and their ability to go to specific locations, means that their mission can be changed without having to recover them.

2.2.1 Slocum Glider

The Slocum glider[1] was first fabricated and tested in January 1991 and this initial development showed that such a device was viable. Initially the change in buoyancy was achieved by electromagnet released weights and

[1]Named after the American sailor Joshua Slocum, the first to sail around the world single-handedly.

later by using a hydraulic pump to inflate an external bladder in the tail section. The goal was to exploit the temperature difference between the upper and lower layers of the ocean to create a thermocline-driven buoyancy engine. Some of the outcomes of the testing were that glide speeds of $0.28 m/sec$ horizontally were achievable. This was the figure originally deemed necessary for the glider to be able to meaningfully station-keep. The increases to this speed were mainly due to reduced drag and increased drive force (buoyancy change). Another outcome was the knowledge that to maximize the glider's horizontal velocity, the ideal dive angle and a constant drive force over the course of the dive had to be maintained. This meant that passive buoyancy compensators would be needed to match the gliders density variation to that of the ocean over the 1800m operating range. They also concluded that suitable control over the glider had been achieved in terms of holding headings and turning in a tight circle.

The controlling factors for long-term success were considered to be low power two-way satellite communications and the thermocline-driven buoyancy change engine. Gliders are now used widely and in fact 480 Slocum gliders occupy, on a monthly basis, the 48 hydro-graphic sections that took the World Hydrographic Program 12 years to survey just once using a ship. They surface around six times a day to report to Mission Control and transmit data via satellite.[5]

2.3 Data Buoys

Data Buoys can generally be divided into two separate categories, tethered and untethered. They are usually used for sea-state and weather monitoring and record data such as air temperature, water temperature, wave height, wave period, air pressure, wind speed and wind direction.

2.3.1 Drifting Data Buoys

These survive for about 18 months and use some form of drogue[2] to prevent them from moving out of the current they are monitoring. They are also quite cheap which is important because of their relatively short lifespans and are often used in the Lagrangian analysis of ocean currents[3].[1]

2.3.2 Tethered Data Buoys

Often serving National Forecasting needs these buoys are usually large, some up to 12m across, and expensive. Some of these record more data than they transmit, in order to lower costs, meaning that periodic retrieval of the data is needed. This means that most of the large buoys are also regularly

[2]A parachute-like sea anchor often suspended at a specific depth to keep the buoy in a particular ocean current.

[3]Lagrangian analysis is where the individual parcels of fluid are tracked in space and time to give an understanding of flow field

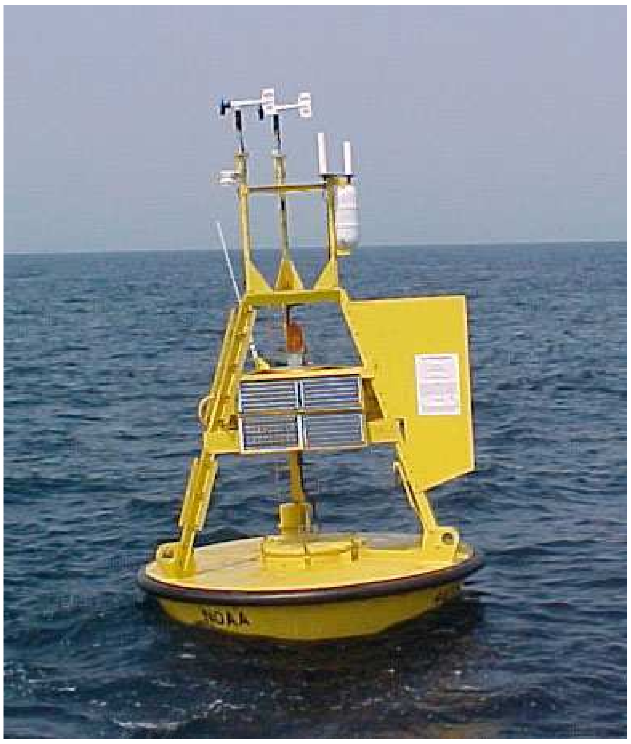


Figure 2.4: A medium sized weather buoy deployed by the National Oceanic and Atmospheric Administration(NOAA).

maintained, serviced and upgraded. Live data is used in research work and forecasting, non-live data is processed and analysed to provide more accurate historical records. They are tethered at many different depths, from relatively shallow waters to depths of 6000m. Due to their static nature vandalism and theft is an issue especially when they are tethered in close proximity to fishing grounds, where their tethers may get caught in nets.

2.4 Water Quality Sensors

Most of the systems described above contain complex water quality sensors called Sondes[4] or CTDs[5], for measuring parameters such as depth, temperature, turbidity, conductivity, salinity, pH levels, concentrations of nitrogen/oxygen and many more. Many of the systems described above use pre-made units, as developing and engineering these sensors to work at depths of several thousand meters is expensive and complex. These vary in capability from very small and portable pieces of equipment that measure only a few parameters to larger systems, measuring 10's of parameters and

[4]Which means sensor/probe in several European languages.
[5]Conductivity, Temperature, Depth.

rated for depths of over 6000m. Nearly all contain a basic control system and batteries allowing them to log selected parameters to internal memory for a small period of time or to output NMEA[6] formatted strings usually over a serial connection. Due to the extremely high pressures that they are subjected to, the vast majority of the sensors are optical as there are few other technologies capable of withstanding such pressures for long periods of time. This means that calibration is vital and that there is potential for fouling of the sensors due accumulated dirt or debris.

2.5 Satellite Communications

There are many different companies offering satellite communication services, some have their own satellite networks and some resell services from other providers. Different companies provide different hardware and software to make use of their services which often differ in reliability, performance and coverage. The author has focused on Iridium due to the department's experience with them, although research into other companies and services was undertaken it is not presented here.

2.5.1 Iridium

Iridium owns one of the largest satellite constellations currently in orbit around earth, consisting of 66 cross-linked satellites in a LEO (Low Earth Orbit) of 780km above the Earth. They also operate 7 in-orbit spares. The constellation is made up of six orbital planes, each made up of 11 satellites, that intersect over the poles. The satellites travel at around $17,000mph$, complete an orbit in 100min and take approximately 8 minutes to travel from horizon to horizon. Each satellite projects 48 spot beams toward Earth, each beam covers an area with a diameter of 250 miles and in total one satellite covers an area with a diameter of 2800 miles (spot beams overlap). This allows Iridium to provide truly global coverage. Calls and data are relayed through the constellation then downlinked to an Iridium Gateway on the ground where it is patched into the PSTN (Public Switched Telephone Network).

The constellation and infrastructure design means that satellite outages will be localised to a particular region and that inter-satellite links can be routed around the damaged satellite. The in-orbit spares also mean that a damaged satellite can be replaced quickly and backup "Earth Terminals" provide redundancy for the downlink element of the system. Iridium also sell a service called SBD (Short Burst Data), a text message like system that allows the transmission of 340 byte messages and receipt of 270 byte messages. There is a fixed cost per byte, a monthly subscription fee and purchasable SBD modems that can be interfaced with to send and receive data.[4]

[6]National Marine Electronics Association 0183 data specification.

2.6 Buoyancy Control

Archimedes said:

> "Any object, wholly or partially immersed in a fluid, is buoyed up by a force equal to the weight of the fluid displaced by the object."[11]

For an object to float in a fluid it must be less dense than the fluid, i.e. for a certain volume of both the object and the fluid, the object must weight less. In the case of water, the weight of water displaced is directly proportional to the volume displaced, assuming that the density remains uniform. To consider it from another angle, if there are two objects of equal weights but different volumes immersed in a fluid, the object with a greater volume is more buoyant.

This means to submerge a floating object it must either decrease it's volume or increase it's weight. To make a submerged object surface the volume must be increased or the weight decreased. It is generally easier to change your volume than your weight and there are various techniques that are currently used. Submarines pump water into tanks to take on weight and pump it out to lose weight. Divers often carry inflatable packs that they can fill with air from their tanks, increasing their volume. Argo Floats fill an external bladder with hydraulic fluid to increase their volume and many gliders do the same. There has been research into buoyancy control using wax that has a very low melting temperature. The difference in water temperature at the surface and at depth is enough to melt or solidify the wax, which results in a change in volume that could potentially require no power.

It is also important to be neutrally buoyant at a certain depth, this means that an object is as dense at the water around it and won't sink or rise. This allows you a "rest" depth which requires no energy to maintain. The effects of drag and the viscosity of the water affect the rate at which you move through water, but it was felt that this isn't worth exploring in too much detail for my project. Another issue is that of stability, floating objects are vertically stable, but not rotationally stable. This may cause problems with the tether but should be relatively simple to solve.[27]

2.7 Sonar

Sonar (SOund NAvigation and Ranging) is the use of sound propagation for navigation, communications and object detection. As a technology it can be divided into passive and active sonar. Active sonar produces pulses of sound and listens for the echoes, whereas passive sonar just listens. The word sonar often refers to the equipment used to generate the pulses and listen to responses. The frequency of sound used ranges from infrasonic to ultrasonic depending on the application. Lewis Nixon invented the very first passive sonar “listening” device in 1906. The technology was later demonstrated in 1914 where Reginald Fessenden showed that it could be used for depth sounding, underwater communications (using Morse code) and echo ranging. In 1915 Paul Langévin used sonar for detecting submarines and during the first and second world wars the technology was improved significantly[3].

There are several factors that affect the speed of sound in water, such as salinity, pressure and temperature. These should all be taken into account when trying to get accurate distance information. Also, as the ocean temperature varies with depth, sound can be refracted in unexpected ways. This is especially true of the thermocline, a layer of water from about 30-100m that “separates” the warm upper layer and cold lower layers of the ocean. Shallow waters can also provide problems as sounds are reflected off the bottom and surface, this could cause problems but tests will be conducted when a working sonar module is acquired.

3 Requirements

3.1 Objectives

The objectives listed here are the general goals of the whole system. It is not expected that they will all be met within this project, but serve as something to aim for. Some are not quantifiable and thus hard to measure or test, regardless of this the design should take these objectives into consideration. Any decisions that result in negatively impacting progress towards a goal will need to be strongly justified.

- Create a tethered data buoy that can autonomously take water quality samples at a variety of depths at the calving front of a glacier.
- Transmit the data in real-time, enabling faster examination and dissemination.
- Keep the costs as low as possible including the initial cost, maintenance and running costs.
- Make the system easy to use for a people with low/no programming or electro-mechanical skills.
- Record at least conductivity, temperature and depth measurements and provide the facilities for more types of measurements.
- Create a robust system that can function as intended, completely unattended for long periods of time.
- Meet all requirements detailed in the following section.
- Produce a system that can be further developed to add extra or more complex functionality.

3.2 Requirements

These are the requirements that have determined after several conversations between the author and their supervisor and with various geographers and glaciologists who were interested in the project.

1. Sample water continuously at a rate of 1Hz.

2. Transmit a reading every 10m of ascent, store the rest of the data internally.
3. Water samples should include the following data: depth, temperature, conductivity. Other measurements such as dissolved oxygen and chlorophyll content would also be of use if available.
4. The buoy should be completely autonomous and should reliably recover from faults.
5. The data should be transmitted in as close to real time as possible.
6. All data should be timestamped with accurate GMT time so that it can be matched to any events that occur.
7. The buoy should perform a profile every 6 hours.
8. The buoy should be able to detect ice above it and stop its ascent, to prevent damaging itself.
9. If the buoy cannot transmit the data collected it should be stored and an attempt should be made to send it again as soon as possible.
10. The buoy should be tethered.
11. The buoy should be able to operate at depths of up to 200m.
12. The buoy should be easy and quick to deploy and retrieve.
13. The buoy should be of a comparable price to any similar products.
14. The buoy should have enough battery power to function for 1 year.
15. The buoy should be re-deployable after recharging the battery and retrieving data.

3.3 Desirable, non-essential Features

There are several features that have been determined to be outside the scope of this project but would be considered beneficial. Completing any of these features will only be considered if all other requirements have been met. Designing the system so that these features could later be implemented is also desirable.

- Two way communication.
- Measuring the thickness of any ice above.
- Sending status messages with information such as battery voltage and number of profiles done.

The main goals of the project are to make a robust and reliable system that can operate for long periods of time autonomously. While, as a computer scientist, the scope of the project is focused on software engineering and not the engineering and design of the actual buoy the author feels that considering these areas in depth is critical. To this end it is the intention of the author to build the complete system; software, electronics and other hardware with the goal of having a complete and usable system by the end of the project. The end users of the system consider a 'long period' to be approximately 1 year, so the system will be designed to have enough power to last 400 days. This should allow the author a small margin for error. The requirement that the system be able to operate at a depth of 200m is challenging from an engineering perspective, for testing 20m will suffice but the system will be designed with depths of 200m in mind.

4 Methods and Tools

4.1 Methodology

Although this is the largest project the author has undertaken, as projects go it is still a small one. This coupled with the fact that the author is working alone, rather than with a team of people, leads him to feel that there isn't any one Process Model that would be entirely suitable. The temptation is therefore to fall back on the "Hacker Methodology" or absence of any coherent structure; however, the author feels that this is not appropriate. Despite being a small project the author feels that it can benefit from many aspects of other methodologies and has therefore come up with their own model. In fact it is often argued that projects involving only one or two people require no methodology at all; "The specific methodology used in performing any of the tasks does not need to be coordinated when only one or two people are involved."[26]. Many lone developers claim to use eXtreme Programming (XP) to cover the fact that they don't adhere to a methodology and while there are certainly elements of XP that can be considered in single developer projects, generally XP has a heavy focus on being part of a team. There have been discussions on adapting XP for one person[2] but the author is still not convinced of it's suitability in this instance.

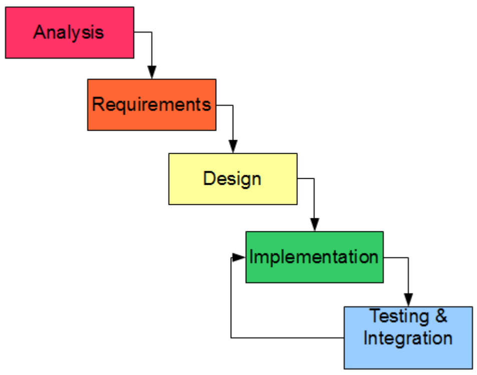


Figure 4.1: The variation of the waterfall method used in the project. The main difference is the approach to implementation and testing.

The methodology that the author has based his own methodology on is the Waterfall Methodology[24], from it is taken the over-all "structure" of the project; first analysing the problem then specifying requirements, designing, integrating and then testing. The last two "phases": deployment and maintenance do not really fit into the scope of this project as the author expects the result of the project to be a tested and working control system but the physical device to still be a prototype. The author will be following the linear path through the phases, not looping back, this is due to unavoidable constraints such as hardware limitations and access to equipment preventing all but minor changes to the requirements. While this could be considered restrictive the author believes that by keeping in contact with several glaciologists at the University he can make sure that the requirements will be adequate before moving on to the design phase.

The approach used for implementing, integrating and testing the software and hardware is taken from a different methodology. The author considered a risk based approach but decided that there was a chance of spending too much time trying to get the more complicated things working perfectly first and not progressing satisfactorily. Instead this phase of the project will utilise a form of RAD[1][12] which is an incremental iterative approach. Although it is not expect that requirements will change as a part of the implementation phase, it is felt that this style of incrementally building up functionality and continually iterating, suits this type of project. The project will be split into "sections", a section will then be developed, integrated into the whole system and the process repeated until all sections have been completed, at which point iterations will cover the whole project. At this point the system should meet all of the requirements and full system and acceptance tests can be carried out.

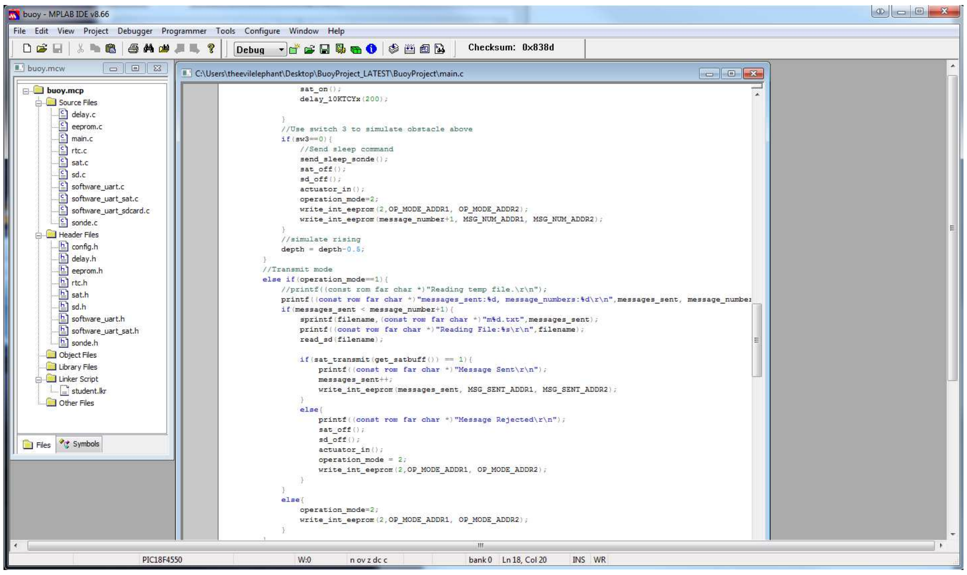
4.2 Development Tools

4.2.1 Programming Language and IDE

For ease of development the author has chosen to use the MPLAB IDE and C18[6], which is an ANSI compliant C compiler for PIC18 microcontrollers. The IDE is very basic but offers features the like estimated memory usage for EEPROM, ROM and RAM the compiled program and support for different memory models. MPLAB and C18 are nicely integrated with each other and while MPLAB isn't a fantastic IDE, lacking integrated version control or any advanced editing capabilities, it is acceptable for a project of this size. This choice of development software also means that the author will primarily be working on systems running the Windows XP Operating System although it is possible to get MPLAB almost completely functional on Linux using wine[2].

[1]Rapid Application Development.

[2]A compatibility layer for running Windows programs on Unix systems



```
                sat_on();
                delay_10KTCYx(200);

            }
            //Use switch 3 to simulate obstacle above
            if(sw3==0){
                //Send sleep command
                send_sleep_sonde();
                sat_off();
                sd_off();
                actuator_in();
                operation_mode=2;
                write_int_eeprom(2,OP_MODE_ADDR1, OP_MODE_ADDR2);
                write_int_eeprom(message_number+1, MSG_NUM_ADDR1, MSG_NUM_ADDR2);
            }
            //simulate rising
            depth = depth-0.5;
        }
        //Transmit mode
        else if(operation_mode==1){
            //printf((const rom far char *)"Reading temp file.\r\n");
            printf((const rom far char *)"messages_sent:%d, message_numbers:%d\r\n",messages_sent, message_numbe
            if(messages_sent < message_number+1){
                sprintf(filename,(const rom far char *)"m%d.txt",messages_sent);
                printf((const rom far char *)"Reading File:%s\r\n",filename);
                read_sd(filename);

                if(sat_transmit(get_satbuff()) == 1){
                    printf((const rom far char *)"Message Sent\r\n");
                    messages_sent++;
                    write_int_eeprom(messages_sent, MSG_SENT_ADDR1, MSG_SENT_ADDR2);
                }
                else{
                    printf((const rom far char *)"Message Rejected\r\n");
                    sat_off();
                    sd_off();
                    actuator_in();
                    operation_mode = 2;
                    write_int_eeprom(2,OP_MODE_ADDR1, OP_MODE_ADDR2);
                }
            }
            else{
                operation_mode=2;
                write_int_eeprom(2,OP_MODE_ADDR1, OP_MODE_ADDR2);
            }
```

Figure 4.2: MPLAB IDE.

C18 is almost identical to ANSI C99, the primary differences being that some functions are only partially implemented and a few others not implemented all. Two examples of this are the printf function which is implemented in it's entirety except for the ability to print floats or longs and the sscanf function which is not implemented. Unfortunately these exceptions are not particularly well documented and rarely throw any errors during compilation, thus trying to use printf to print a float will compile and run but not work as expected.

Version Control and Coding Standards

Although facilities for SVN3[3] were made available, the author chose not to use them because there was no integration with the IDE. A program like TortoiseSVN could be used to apply version control to the project folder; however, the author has had some bad experiences with SVN. Instead to backup project files the project directory will be compressed and stored on the central university filestore. This only takes 30 seconds and the university filestore is, in turn, suitably backed up by the universities I.T. staff.

The author had originally intended to follow an embedded C coding standard. However, the author was unable to procure a copy of Netrino's book detailing this standard and so no official embedded standard will be followed while programming.

[3]Subversion software versioning system.

4.2.2 Programming Hardware and Debugging

Programming the microcontroller will be done via USB using the FS USB programming software supplied by the company, Microchip, that makes them. Debug information will be sent via RS232 Serial to a host computer running a communications terminal which will allow the data to be either viewed or recorded to a file.

5 Design

This chapter first covers the different hardware components that have been selected for the project, the reasons for selecting them and how the decisions effect other aspects of the project. The commands needed to generate the necessary responses will also be shown and any additional circuitry required will be detailed. The software design is then presented including the control system architecture, any algorithms developed, files required/created and flow diagrams of complex code sections. Lastly plans for mechanical construction of the buoy and any materials needed is discussed.

5.1 Electronic Hardware

The block diagram (Fig. 5.1) below gives an overview of the different hardware elements of the buoy and the connections between them.

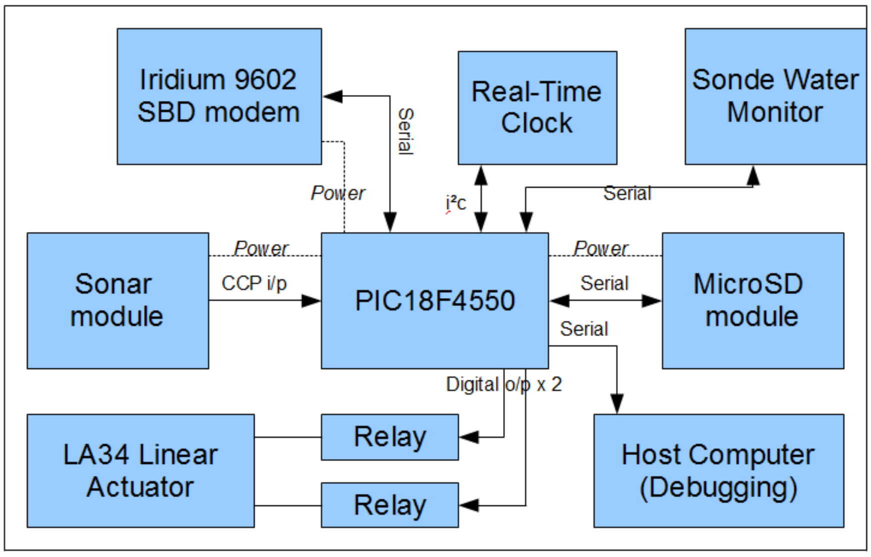


Figure 5.1: Simplified block diagram showing the hardware components and the connections between them. The diagram shows that the system uses 4 serial connections, one i^2c connection, one CCP enabled input and five digital outputs for power and actuator control.

5.1.1 PIC Microcontroller

The first decision made was to use a microcontroller as the only piece of control hardware. There was an option to use a Gumstix[1] computer instead of, or in parallel, with the PIC. The Gusmtix is far more powerful in terms of processing power[2], it has a file system and storage is easy to add. It also has pre-made add-on boards that offer increased functionality such as networking and I/O. There are several reasons for not choosing to use the Gumstix; firstly it is felt that the system doesn't require the amount of processing power such a device would offer, secondly many of it's other features are not of any use in the project, and lastly the power requirement is much higher than that of a microcontroller.

The next decision was the choice of which type of microcontroller to use, there are many kinds such as ATMEL AVRs or Parallax Propellers and this is where the department's experience was very helpful. They had done a lot of work with PICs[3] and so a PIC 18F4550[9] was chosen, a microcontroller that they had a lot of experience with. Another reason for this choice was so to save the author from designing all of the hardware himself, as there is a pre-existing prototyping board called a PICDEM FS-USB Demo-Board. This is a pre-assembled circuit for this microcontroller that has all of the necessary electronics needed to power and use the PIC. This includes power regulation, oscillators, reset buttons, USB programming support and easy access to pins. The downside of using the PICDEM board is that it is fairly large and requires more power than a custom made circuit. However, it was felt that designing a circuit to take it's place would require too much time for it to be worth it for the project.

The PIC has 32KB of Program Memory, 2KB of RAM and 256B of EEPROM[4], although these are very small amounts only the amount of RAM is considered to be a potential problem as the project will involve some fairly large data structures. There are also 16 i/o pins, 3 timers, a 13 channel 10bit ADC[5] and runs at 48Mhz. One fairly important fact is that the PIC has only one serial port, whereas the design calls for 4. This means that three will need to be implemented in software which limits their speed to 4800bps[6]. This was not expected to be a problem given as most hardware supports this speed.

5.1.2 Satellite Communications Modem

This was another area where both the author's experience and the experience of his supervisor proved to be very useful. Previous experience with Iridium

[1]A small "gumstick" sized computer capable of running Linux.

[2]500Mhz clock speed compared to the 48Mhz of the PIC.

[3]Programmable Integrated Circuit

[4]Electrically Erasable Programmable Read-Only Memory.

[5]Analogue to Digital Converter.

[6]Bits per second.

SBD modems, in particular the 9601 SBD[7] modem simplified the choice. The choice of the Iridium 9602 modem was due to it's more compact size, lower power requirements and it's backwards compatibility with commands for the 9601. Another important factor in this decision was that Iridium has very good coverage over the entire globe including the poles, whereas other satellite networks coverage often deteriorates near the poles. As the buoy is designed to be used in places like Greenland and should stay surfaced for as short a time as possible, this is a crucial design decision.

The modem is very small (40mm square, 13mm deep), weighs only 3g and has a peak power consumption of 10W while transmitting. The power consumption is especially impressive and considering that a message can be sent in 1 second it is perfect for this project. Communication is achieved via a 0v-3v serial connection and AT commands, although the PIC uses 0v-5v on it's i/o pins from previous experience the two should be compatible. The commands needed to send an SBD message are as follows:

1. AT+SBDWT - The command to write data to the modems output buffer. Response: READY
2. Now send the message data followed by a Carriage Return. Response: 0 or 1 (succeeded or failed)
3. AT*R1 - Turns the radio on. Response: 0 or 1 (succeeded or failed)
4. AT+SBDI - Initiate the transfer. Response: 0, 1 or 2 (No message to send, succeeded or failed)

There are other commands for functions such as flushing the output buffer but the documentation says that they are not essential for correct operation, so for the sake of simplicity they have been omitted. If this causes a problem it will be reinvestigated during the implementation phase of the project.

5.1.3 Water Sampling Hardware

This is an area where there is really very little to differentiate two products. The measurements taken by different units are often equally accurate and precise, most offer the same sensors and depth rating is the main factor in terms of cost. The choice to use the YSI 6600V2 Sonde was due to the fact that the department already owned one and because the glaciologists who are interested in the project consider it adequate. It is can withstand depths of approximately 200m and contains sufficient battery to power itself for a full year.

It uses an RS232 level serial connection for communication and outputs NMEA formatted data. Some configuration is needed to set the sample rate, but once set up all settings are stored permanently. To get a constant stream of data requires the following commands:

[7]Short Burst Data

Environmental Monitoring Systems
YSI
6600 V2
Multiparameter Water Quality Sonde

Figure 5.2: The YSI 6600V2 Sonde used in this project.

1. Send Escape character several times to wake up the Sonde.
2. Send "nmea" and a Carriage Return.

After it has performed its cleaning routine (which takes approximately 45 seconds) it will continuously output data until commanded to stop. It is worth noting that the Sonde will perform this self cleaning routine whenever it feels it needs to, there is no control over this aspect. To power off the Sonde:

1. Send Escape character several times to stop the output.
2. Send "sleep" and a Carriage Return.

5.1.4 Storage

The use of EEPROM modules for storage was seriously considered as there are several EEPROM chips that utilise an 'i^2c'[8] bus for communication. The particular EEPROM chip considered was a 24FC512 from Microchip, which could store 512Kbits and had the additional feature of being able to work with 7 other EEPROM chips connected to the same i^2c bus as one 4Mbit EEPROM. It was also low power needing only 5mA to write and 400uA to read. The main problem with using these devices was that 4Mbit, or 500Kbytes, split over 400 days is 1.25Kbytes a day of space. If every reading is only 25bytes, which is rather optimistic, then that means that there is only space to store 50 readings a day, split over the 4 profiles required a day means there is only space to store 12 measurements per profile. This is nowhere near enough, a far better amount would be one measurement per meter, which for a 200m buoy would result in 200 per profile or 800 a day. Using a more realistic 75 bytes per measurement this equates to 24MBytes in 400 days. Due to this it was decided that a micro SD card module would be a better choice, this would allow the use of micro SD Cards up to 2GBytes in size. It will also make retrieving the data much

[8]Inter-Integrated Circuit bus

easier as the SD card can simply be removed and a new one inserted in its place.

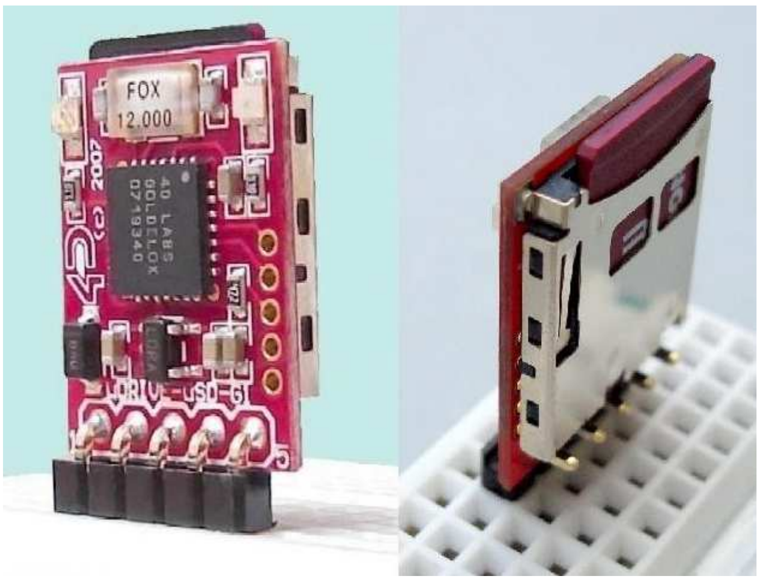


Figure 5.3: The 4D Systems Micro SD Card module used in the project.

A 4D-Systems GOLDELOX micro SD Card[25] module was chosen as recommended by the author's supervisor who had previous experience with them. It requires a 5v power supply, uses 250mA at peak and uses an autobauding 0v-5v serial connection for communications. It also has the benefit of integrated FAT16 support which means that the data will be correctly formatted into files making it much easier to retrieve, as opposed to copying the data byte by byte. In order to help keep things simple parameters for commands have been chosen to prevent handshaking which, for small amounts of data (less that 512bytes), is considered acceptable. Commands are in hexadecimal byte form, the following describes how to autobaud the connection, initialise the card, write a file and read a file (hexadecimal values are shown as 0xValue).

Autobaud

1. Send 0x55 - Autobaud command (May require several attempts). Response: 0x06 or 0x15 (Accepted or Not Accepted.)

Initialise Card

1. Send 0x40 - Indicates next command is a FAT command.
2. Send 0x69 - Initialise command. Response: 0x06 or 0x15 (Accepted or Not Accepted.)

Write File

1. Send 0x40 - Indicates next command is a FAT command.
2. Send 0x74 - Write file command.
3. Send 0x80 - Append mode with no hand-shaking.
4. Send Filename followed by 0x00 to null terminate.
5. Send size of data to be transmitted split into 4 bytes in big-endian format. Response: 0x06 or 0x15 (Accepted or Not Accepted.)
6. If received 0x06 then transmit data. Response: 0x06 or 0x15 (Accepted or Not Accepted.)

Read File

1. Send 0x40 - Indicates next command is a FAT command.
2. Send 0x61 - Read file command.
3. Send 0x00 - No hand-shaking.
4. Send Filename followed by 0x00 to null terminate. Response filesize split into 4 bytes in big-endian format.
5. Send 0x06 to accept or 0x15 to reject the file. Response data: or 0x15 (Not Accepted)
6. Once all the data has been sent 0x06 is received.

5.1.5 Buoyancy Control

Several different possibilities were considered for controlling the buoys buoyancy, the first was to use a linear actuator and piston to displace water in a tube, the second was to use compressed air to do the same, the last was to pump oil from an internal bladder to an external bladder. Using compressed air would be easier and require less engineering than the other two solutions. However, it is simply not possible to fit enough compressed air into the buoy to be able to operate for the 400 days necessary. The amount of air needed to provide a fixed increase in buoyancy changes as air is consumed, due to the fact that the weight of the tank changes. This need to compensate for the weight of the air in the tank effectively rules it out as a possible solution. The following formula was used to determine how much air would be left in the tank when trying to change the buoys buoyancy:

$$n + 1 = n(b * p + \frac{(n * w * p)}{1000}) \tag{5.1}$$

- n= Previous result (Starting at 2400, the amount in litres of air in the full tank).

- $n + 1$ = Air left in the tank.
- b = Desired change in buoyancy (in litres).
- p = Pressure (in atmospheres, 1atmos per 10m down).
- w = Weight of 1 g of air.

Using this formula a program was created that would iterate until the amount left in the tank became less than 0 indicating an empty tank. Using an 8Ltr tank at 300bar (2400 litres of air), ascending from a depth of 200m and trying to change volume by 125ml it was calculated that the tank would only last 127 ascents or 31.75 days, as shown in the Appendices, Fig D.2. Further calculations showed that even with 10 million litres of air it would only last 113 days!

This left two possibilities; pumping oil between internal and external bladders and an actuator displacing water from a tube. The limiting factor in the pump design is the pump itself in particular its pumping pressure. The limiting factor in the actuator design is how much force the actuator can produce. It was decided that, due to the less complex engineering required, the actuator design would initially be used. Using an actuator and piston set-up is, by no means, without its own drawbacks; mainly the amount of force requires to move the piston at great depths. But at 200m it is certainly possible using an actuator with around 4000N of force to displace 200ml of water, a graph showing actuator force vs depth is shown in the Appendices, Fig D.1. Controlling the actuator is very simple, as they are powered by a DC[9] motor. Two relays allow power and ground to be connected across the motor in both directions. The circuit being used is shown in the Power section, the motor is expected to draw 15A at full load which will take up the majority of the system's power budget even though it is only run for a few minutes a day.

5.1.6 Sonar

So that a custom made circuit does not have to be made a pre-made sonar kit will be used, which is readily available from many online retailers. More specifically a Velleman ultrasonic car parking sensor will be used. The transducers will need to be replaced with waterproof versions and it will need to be integrated with the rest of the system. The circuit outputs variable width pulses that change with the range of a detected object. Using the PICs Capture functionality it will be possible to measure the length of pulses and therefore the distance of a detected object. Although the kit only has a range of 1.5 meter, sound travels around 4.3 times faster in water than in air which effectively increases the range to approximately 6.5 meters.

Since the hardware is responsible for creating and detecting the sonar pulses the remaining work lies in making sense of the output from the sonar

[9]Direct Current.

hardware. It was decided that an interrupt based approach should be used as it removes the need to poll the sonar for the entire duration of the ascent. The PIC has a CCP[10] module that is ideal for this kind of application. The maximum pulse length needs to be determined, this is so that a timer pre-scaler can be chosen allowing use of the timer without it overflowing. This will need to be done by looking at the output on an oscilloscope and measuring the pulse length. The CCP module is configured to interrupt on a rising edge and peripheral interrupts are enabled. When the interrupt occurs the timer value is recorded and the CCP module reconfigured to interrupt on a falling edge. When this interrupt occurs the timer value is again stored and the first timer value subtracted from it.

$$timer\_value * pre - scaler * \left(\frac{1}{clock\_speed}\right) \tag{5.2}$$

The PIC being used for the project runs at 48Mhz, using the biggest pre-scaler available (256) and an example timer value of 50 would indicate a pulse length of:

$$50 * 256 * \left(\frac{1}{48Mhz}\right) = 6.4 \cdot 10^{-4} seconds \tag{5.3}$$

5.1.7 Real-Time Clock

So that data can be timestamped with an accurate time it will be necessary to use a Real-Time Clock module, this is an accurate electronic clock that has its own battery so that the internal oscillator is constantly running should its external power supply turn off. The Sonde also has an internal clock but it tends to drift and so does not keep time particularly well during long periods of inactivity. The particular RTC module has been chosen is based on a DS1307[23] chip, it requires a 5v power supply and uses 1.5mA when in use or 500nA when using it's battery. It uses i^2c for communications on address D0 and uses seven registers to store seconds, minutes, hours, day of the week, date, month and year. The format of the registers is BCD[11], with values being split into 10's and digits, thus 45 is stored as the combination of two nibbles; 0100 (4 in binary) and 0101 (5 in binary) equalling 01000101. The reason for this is not explained but it does make setting and decoding the time slightly more complex.

5.1.8 Power Consumption and Control

Something else that is vital for the design is how much electrical power the system will need to operate for the requisite 400 days. The Sonde contains its own battery capable of powering it for the full extended year. Table 5.1 shows voltage and current requirements, how long each hardware component

[10]Capture Compare and PWM
[11]Binary Coded Decimal.

Item	Peak Current	Voltage(V)	Duty Cycle(s)	Joules a day
Sonde	n/a	n/a	n/a	n/a
Iridium	1500	5	120	3600
PIC	60mA	12	900	2592
Storage	250mA	5	15	75
Actuator	15A	12	20	14400
Sonar	30mA	12	900	1296
RTC	1.5mA	5	900	27
Total				21990

Table 5.1: The power requirements of the different components

is on for and how many joules per day each piece of hardware uses. The estimate of joules used is an extreme one using the maximum current and the pessimistic value of how long each must remain on. This pessimistic view of the system's power consumption shows that the system uses 21963 joules a day. This means that a battery that can store 400x 21990 = 8.796MJ is needed. In reality after testing this number should go down, also during the winter the surface is likely to be frozen so the buoy will not be able to use the Iridium SBD modem which is responsible for a sixth of the power consumption. The actuator also only draws 15A when it is under its maximum rated load which may be the case at 200m, but at the surface it won't be under much load in comparison so the current usage should drop substantially.

The system will therefore require batteries with a capacity of at least 8.796MJ, the energy density of batteries varies with the technology used, a break down of the properties of different battery technologies is shown in Appendices Table B.1. It is clear that only batteries with a very high energy density are suitable, such as Lithium Polymer or Lithium Ion. Using either of these technologies would require 12.21Kg of batteries which is acceptable as the batteries can double as ballast to help make the buoy neutrally buoyant. There are other technologies that have a higher energy density, such as Lithium Thionyl Chloride (2.5MJ/Kg) which would mean less weight (3.5Kg) but they are more expensive and harder to source. Due to the high cost of large quantities of any type of battery, a small quantity of Lithium Polymer batteries will be used for testing, one 3-cell 11.1V pack should be sufficient for this purpose.

As previously mentioned certain pieces of hardware need to be turned on or off to save power, this will be achieved by using a combination of relays and transistors. A table showing the power state of hardware modules during each system state is shown in the Appendices, Table A.1. The sonar, SD Card module and the satellite modem all use relatively small amounts of power so can be switched on small logic level relays, the motor for the actuator will need transistors driving the coils of bigger relays. The circuit that will be used is shown in Figure5.4, the transistors and relays used for the motor will be rated for higher current but otherwise the circuit will essentially be the same. This can be built on vero-board with components available in the workshop. Five outputs from the PIC will be needed to control the transistors and relays

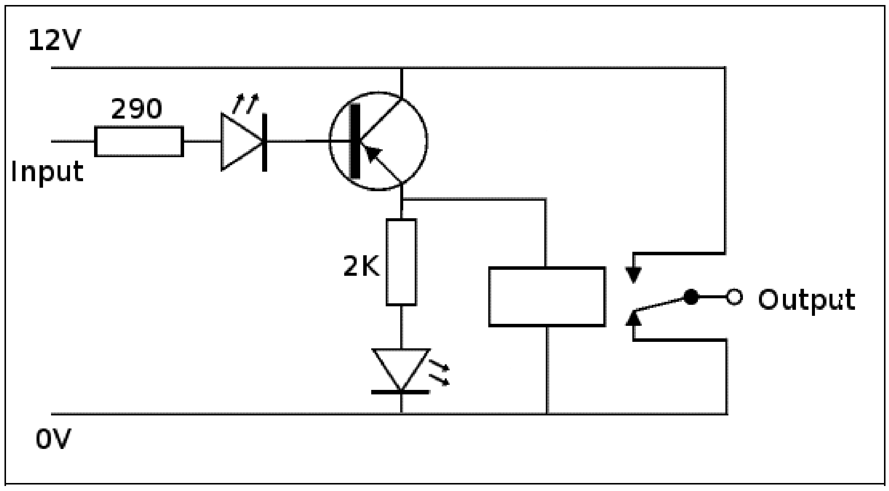


Figure 5.4: Circuit for driving the motor or power relays from a PIC output, two will be needed to control the motor and three for power control to various pieces of hardware.

5.2 Software

5.2.1 State Diagram

The overall system architecture that will be used is based on the state model shown in Fig. 5.5, a change in states occurs when certain parameters are met or after a certain amount of time has elapsed. The four states are as follows; SAMPLE, TRANSMIT, DIVE and SLEEP. In the SAMPLE state the buoy will turn on the water monitoring hardware, increase its volume to rise to the surface, start taking and storing measurements and lastly use the sonar module to detect if there is an obstacle above. The TRANSMIT state handles sending the data stored for transmission via the satellite modem. The DIVE state decreases the buoys volume and monitors the depth until the target resting depth has been reached. Lastly the SLEEP state keeps the buoy at the resting depth with all of the hardware turned off- until the next sample should take place. The reason that a state based approach is

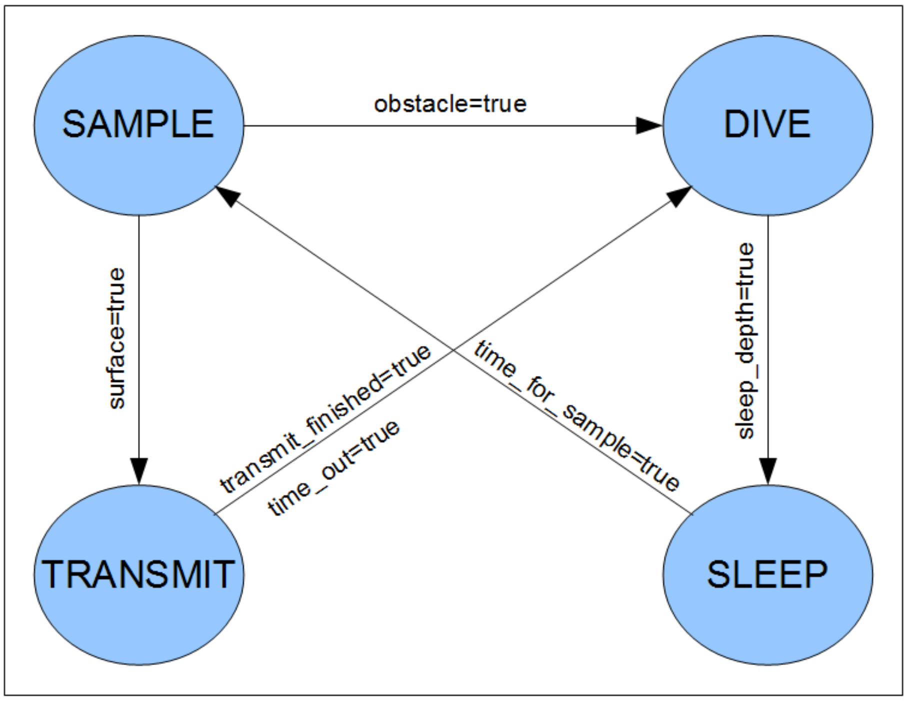


Figure 5.5: The state diagram describing the control system's behaviour. The diagram shows all of the states and the transitions between them with arrows indicating the direction of the state change. The labels next to the transition arrows indicate the condition that must be passed for the transition to occur.

being used is for simplicity. It translates into code very well and during development allows the code to be split up into easily testable sections. These can then be implemented one state at a time, tested and integrated into the rest of the system. Also possible is the enabling or disabling of states to more easily debug any issues. Lastly it provides a simple fault recovery mechanism by tracking the current state which allows it to be resumed should something go wrong.

Sample State

In this state the Sonde is sent commands to wake up and start outputting data, the buoyancy is altered to make the buoy positively buoyant and the system then waits until the Sonde starts sending data. Once a NMEA string is received it is parsed, extracting just the data to reduce the amount of information to process, and the current depth is also extracted for use elsewhere. The parsed string is then stored in the main data log on the SD Card and a check is performed to determine if this reading should also be transmitted via satellite. If it should, then it is written to a message file along with a time-stamp from the RTC. The algorithm for determining this described is below.

```
count ← Current resting depth / desired measurement interval;
upper_bound ← (count * desired measurement interval) + deadband;
lower_bound ← (count * desired measurement interval) - deadband;
if (depth ≤ upper_bound) && (depth ≥ lower_bound) then
    count−−;
    Write parsed NMEA string to message file;
    Write RTC to message file;
    string_count ← string_count + 1;
    if string_count < 2 then
        string_count++;
    else
        string_count ← 0;
        message_number + 1;
    end
end
```

Algorithm 1: Algorithm to calculate if the NMEA string at the current depth should be stored in a message file.

Three readings with time-stamps can fit into a 340 byte SBD message and the number of the of message files is tracked. This is done by storing the number of message files and the number of the last sent file in EEPROM and adding this value onto the filename parameter passed when writing to the SD Card. To use an example; the buoy rises from 200m storing a parsed NMEA string every 10 meters, 3 of these strings fit into one message file so there will be a total of 7 message files. The depth is then checked to determine if the buoy has surfaced, if it has then it switches state to TRANSMIT. Lastly the buoy checks for any obstacles above using the sonar, if an obstacle is detected then it changes to state to DIVE.

Transmit State

The SAMPLE state will ensure that the Satellite Modem is powered on before changing to the TRANSMIT state, once in this state the current message number is compared to the number of sent messages. This provides a method of determining how many and which messages are still queued to be transmitted. If the current message number was 8 but the last message sent was 4 then 5-8 still need to be transmitted. If there are no messages to send then move to the DIVE state. If there are messages to send then construct the filename (m+message_number+.txt) and read the message file from the SD Card. Attempt to send the message, if the modem reports that the message has been sent then increment the number of sent messages and write the value to EEPROM. Now loop to check for more messages to send. If the modem rejects the message (due to inability to transmit) then change to DIVE state. Before transitioning to the DIVE state the SD Card and Satellite Modem are both powered off to minimise power consumption.

```
if messages_sent < total_messages then
    filename ← "m" + messages_sent + ".txt";
    Read message file;
    Try to transmit message;
    if message was sent then
        messages_sent++; Write messages_sent to EEPROM
    else
        Turn Satellite Modem Off;
        Turn SD Card Module Off;
        Move Actuator;
        Write new state value to EEPROM;
        Change state;
    end
end
```

Algorithm 2: Algorithm for queueing and sending messages.

Dive State

This is a very simple state, monitoring the current depth using the Sonde and waiting for it to equal the resting depth. Once resting depth is achieved then move to SLEEP state.

Sleep State

This state is also relatively simple, it stops the actuator to ensure it is not left powered whilst sleeping and then commands the microcontroller to sleep for a pre-configured amount of time. As the length of the sleep interval is known to approximately 1 second there is no need to check the time RTC time. A sleep period of 2 minutes and 20 seconds has been chosen (for reasons described in the Fault Handling section), which requires the sleep function to be run 154 times to achieve just under 6 hours of sleep.

5.2.2 Fault Handling

The author considers fault handling and the ability to recover from errors/-faults to be critical to the project. It should be noted that the type of errors dealt with are ones that still allow the buoy to operate. For example, faults caused by communications error, the SD card, satellite modem, or any other hardware component failing to respond should be detected and recovered from by the system. However, if the buoyancy control stops working then there is no way of recovering, the buoy is stuck at whatever depth it is currently at. This also raises the point that there is no "safe" state for the buoy to retreat to, rising to the surface and staying there risks freezing or crushing due to ice, staying at depth makes it irretrievable. This means that even if it were possible to detect a "fatal" error there is no reliable safe procedure to carry out. One possibility would be to attempt to rise to the surface and transmit a help message, the effectiveness of which is debatable as it extremely unlikely that there would be anyone close enough to fix or retrieve it.

The method used to recover from smaller faults utilizes the watchdog timer and the system desired state. The watchdog timer is a hardware timer that, when enabled, counts down to 0 at which point it resets the microcontroller. During normal operation the timer is prevented from reaching 0 by resetting it every time around the control loop; however, when a hardware component stops responding the program effectively "hangs" and the watchdog timer will count down and reset the system. Configuring the watchdog timer is done by setting a pre-scaler during programming, which divides the clock frequency by the pre-scaler value to increase or decrease how long it takes the watchdog to reach 0. As the watchdog is also used for sleeping there is a trade off between a pre-scaler value that allows the system to respond to faults quickly and a pre-scaler value that doesn't require the microcontroller to wake up excessively(which uses power) during its sleep loop. It was felt that the time taken to reset the system due to a fault should be no more than 5 minutes and that the PIC should not sleep for less than 1 minute. A pre-scaler value that resulted in sleep/reset period of 2 minutes and 20 seconds was chosen.

To re-enter the correct state upon restarting the current state must be tracked. To ensure this happens every time a state transition occurs the value of the state being transitioned to will be saved to EEPROM. At the start of the program this value will be read from EEPROM and allowing the program to jump back to that state. To use an example; the satellite modem stops responding, several minutes later the watchdog resets the microcontroller, upon restarting it is determined that TRANSMIT was the last state and the program returns to the TRANSMIT state. This, of course, relies on being able to solve the problem by powering devices on or off, more

Filename	.c file	.h file
main	√	
eeprom	√	√
sd	√	√
sonde	√	√
sat	√	√
uart	√	√
uart_sd	√	√
uart_sat	√	√

Table 5.2: Files.

complex diagnostic systems would be very difficult to fit onto such a small microcontroller.

5.2.3 User Interaction

So as to make the system as simple as possible user interaction has been minimised as much as possible. The only remaining interaction is turning on the buoy and retrieving data, the first involves flicking a switch and putting the lid on the second requires removing the lid and removing an SD Card to copy the data files. Data sent by Satellite Modem arrives by email to a specified address. It is felt that this will allow the system to be used by less technically minded people.

5.2.4 Files

As a Procedural language is being used rather than an Object Orientated language, to give some indication as to the intended structure of the code, table 5.2 shows the code files (file type) that will be in the project. There will be two types of file created by the PIC on the SD Card. The first will be the main datafile, called "DATAFILE.TXT", of which there is only one containing every Sonde reading. The second type are message files, these are small files under 340 bytes in size that contain the data to be transmitted via the satellite modem. They are named sequentially "m0.txt", "m1.txt", "m2.txt"... etc and contain up to 3 Sonde readings, each timestamped with an RTC value.

5.3 Mechanical Hardware

5.3.1 Body Design

The body of the buoy needs to be able to withstand significant pressures and so it must be made from a strong material ideally in the form of a tube. The choices of material available are acrylic, steel or aluminium with the second two being preferable due their high strength. A tube 500mm long,

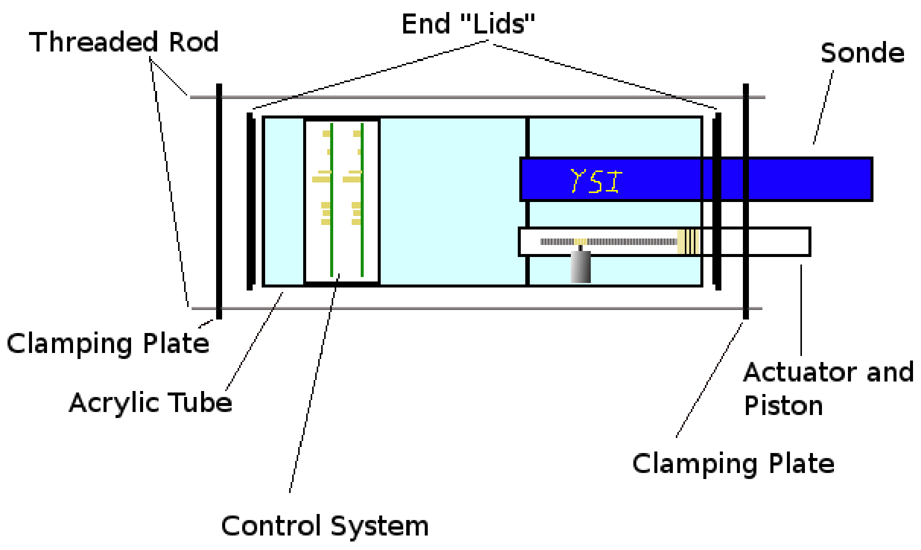


Figure 5.6: Side view of the buoy. The clamping plates are compressed by tightening nuts at either end of the threaded rod. The plates help create a good seal around the top lid and compress the o-ring around the Sonde.

with a 200mm diameter and 5mm thick wall is physically large enough to fit everything needed inside. It is obvious that this tube will weight significantly more if made from steel than acrylic, in fact it would weigh approximately 26Kg; however, this also means that it is negatively buoyant as it would only displace 15.7 litres of water. Even if the wall thickness is reduced, it would still be very heavy. This would give a smaller margin for error and it is much easier to add weight to the buoy than to increase its volume. It was decided that acrylic would be the better choice as the cost of aluminium tubing is quite high and it is easier to work with acrylic. While acrylic is not as strong, it should be capable of withstanding depths of 30-40 meters.

The top and bottom of the tube will be sealed with two “lids”, machined to be a tight fit. The bottom lid will be glued into place with an acrylic solvent which will make a strong and water-tight seal. The top lid will be compressed downwards by a clamping plate and will be sealed using either a rubber o-ring or a removable sealing agent such as silicone. The bottom lid will also have two holes cut out, one for the sonde and the other for the actuator-piston tube, the first will be sealed with an o-ring the second will be glued in place. The main reason for keeping part of the sonde inside the buoy is that it makes the structure easier to handle and also removes the need to waterproof cables and connectors.

5.3.2 Actuator Design

The actuator mechanism will be a sealed piston design using several greased rubber o-rings to create the seal. The tube will be 50mm diameter, 3mm wall acrylic and the actuator will have a stroke length of 100mm. Using the formula below the volume of the resulting displacement can be calculated.

$$\pi \cdot r^2 \cdot h = v$$
$$\pi \cdot 2.5^2 \cdot 10 = 196.35ml$$

The the amount of force required to move the piston is dictated by the pressure acting on it, at 200m this is 20 atmospheres or 2MPascals. The graph shown in the Appendix D.1 shows that 4000N of force is sufficient at 200m. The displacement could be increased without needing to get a higher power actuator by increasing the linear travel of the actuator. Due to the size of pre-made actuators it is a possibility that a "home-made" actuator will be built and used for the project.

5.3.3 Tether

There are many possibilities for the tether system, some complicated like powered or sprung reeling mechanisms and some simple, like a piece of string and a rock. It comes, yet again, down to simplicity. In order to keep the tether tangle free it will be heavily weighted at one end and be attached to a small float at the other. To prevent this float from being destroyed by ice, the tether will be roughly 1m too short which will keep the tether's float submerged by 1m. The buoy will be attached to the tether via a 1.5m stainless steel cable thus allowing run up and down the tether and to surface as shown in Figure 5.7

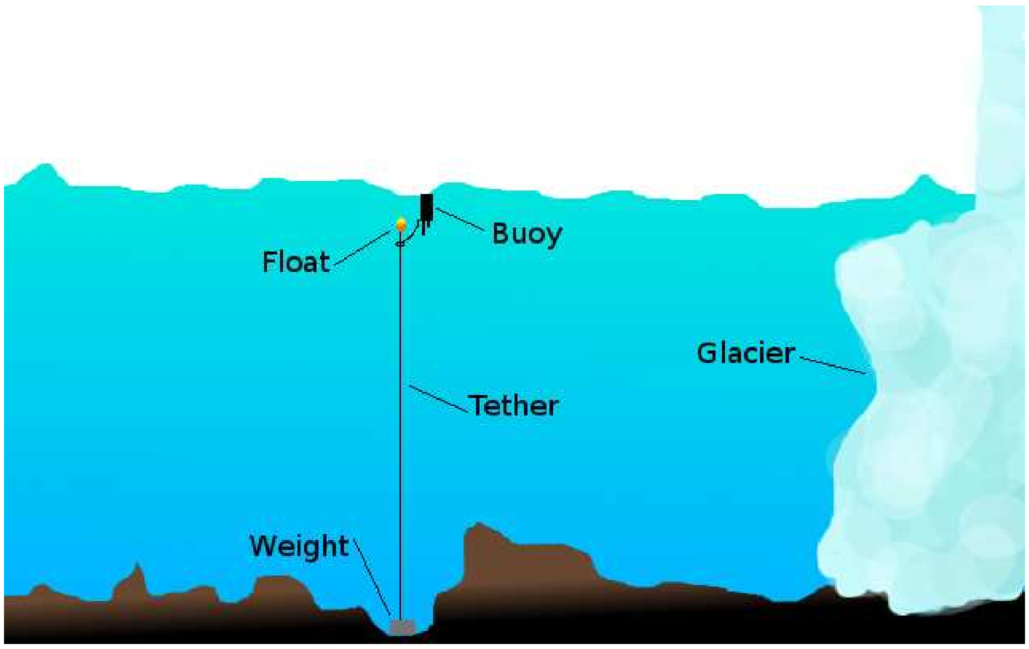


Figure 5.7: The method used to tether the buoy in place.

6 Implementation

6.1 Iteration 1 - Sonar and PIC

Having had previous experience working with PICs, it only took an hour to set up a project in MPLAB and compile a quick test program. Once this had been done the author spent a few days re-familiarising himself with the PIC's features. A few days later the sonar kit arrived and had to be assembled which involved soldering roughly one hundred components onto the supplied PCBs[1]. Once this was done the circuit was connected to a power supply and tested to determine its performance. The module buzzed when it detected an object within a certain distance, which could be varied between 5cm and 150cm. The sonar transducers that came with the kit were replaced with sealed waterproof versions, so that the unit could more easily be tested in water.

Once it was determined that the module was working it had to be modified so that it would interface with the PIC. The first change made was to remove the buzzer and look at the output on an oscilloscope. It showed that the pulses output from the circuit were at 12V, the PIC's inputs are rated 0v-5v so it would need a voltage divider to bring the level down. The next task was to configure the CCP module, which was technically challenging

[1]Printed Circuit Board

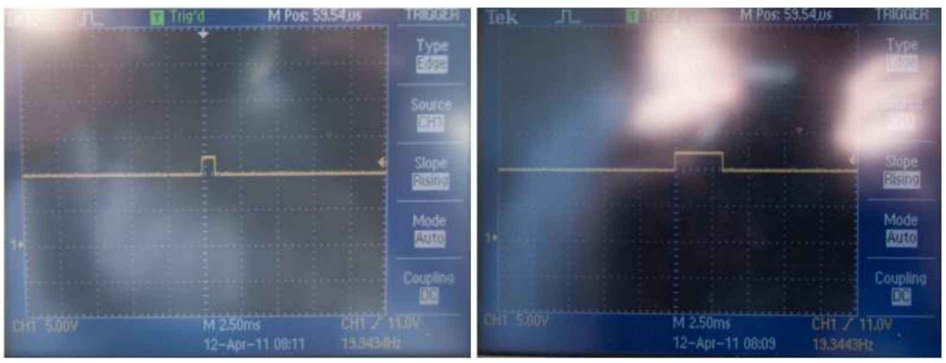


Figure 6.1: The shortest and longest pulses produced by the sonar module. The shorter the pulse the closer the detected object is.

but eventually worked sufficiently well that the PIC could be connected to a frequency generator and calculate the pulse lengths.

After completing the work described above it was decided that, due to time constraints, it would be wise should move on to the next iteration as a significant amount of time had been spent working on the sonar. The only remaining hardware change was to level shift the output of the circuit down to 0-5V so as not to damage the PIC ADC input, once that has been done it can all be tested together.

The author intended to come back to this work in another iteration but after several discussions with his supervisor it was agreed that sonar wasn't really appropriate. This was mainly due to the difficulties in detecting the difference between the surface and an obstacle and the potential failure modes if the module should malfunction. If the sonar became fouled then the buoy would never surface. Other possibilities were discussed but it was agreed that at this point in the project it was best to focus on other parts of the project. The functionality remains in the code and an obstacle detection can be simulated be pressing a button.

6.2 Iteration 2 - Sonde

The hardware serial port on the PIC was initially used to communicate with the Sonde. This had benefit of having the correct voltage shifting hardware built in and the correct physical connector. However, it was decided the hardware port should be used for debugging and the Sonde was moved to a software serial port. Code for this software port had already been written by the author's supervisor for previous projects and so this code was used. The main benefit of this is that the code has been tested for long periods of time in many different devices and has benefited from several bug fixes. To use the software serial port with the Sonde a logic level to RS232 level serial converter had to be built using a MAX232 converter chip. This was the source of much confusion as, despite functioning perfectly well for a few weeks, it suddenly stopped working. Further investigation showed that the circuit had been incorrectly built and once fixed communication with the Sonde worked flawlessly.

When communicating with the Sonde there is the option to use a menu system, which offers more features like calibration and the ability to change parameters such as sample interval. Despite the benefits, autonomously navigating menu systems designed for humans is fairly hard, especially as serial connections are subject to small levels of data corruption when surrounded by other electrical items. This could lead to becoming completely lost in the menu system and accidentally changing the wrong parameter, such as baud rate, which could stop the sonde from working. Instead the author opted to use the simplest commands possible; “nmea” and “sleep”.

Some simple code was written to read the incoming NMEA strings, buffer them and print them out over the debug serial port. The start of a string was detected by waiting for the character sequence "$YS" at which point characters were buffered until a newline character was received.

6.3 Iteration 3 - SD Card

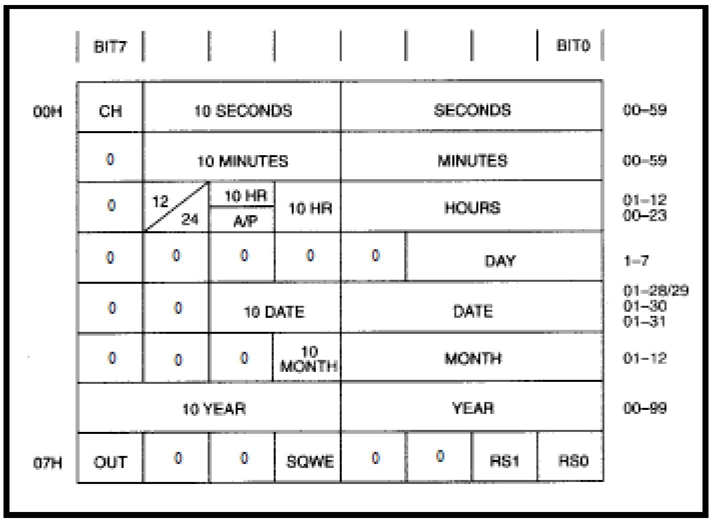
Now that data could be retrieved from the sonde, it needed to be stored. The SD Card module was simple to wire up requiring only communication and power lines; however, a voltage regulator was needed to step down the 12V to 5v. The SD Card module is autobauding so to set the correct baud rate the host (the PIC) sends the autobaud command 0x55. Initially the device would respond with 0x15 which is a NAK (Not Acknowledged) byte, but it was determined that the module needed a longer delay between power-up and sending the command. The module occasionally still responds with a NAK but sending the autobaud command again returns an ACK. The next command required an SD card to be present so one was formatted to the FAT16 filesystem. The initialise disk command worked first time, returning an ACK.

The next task was to try writing to a file, it is at this point that things started going wrong. As well as the write file command certain other parameters need to sent; these include handshaking size, append mode enable/disable, the filename and the size of the data to be written. After carefully consulting the datasheet the commands were sent. The module responded with a NAK before the commands had finished being sent, there was nothing in the data sheet to indicate what this meant. After triple checking the data it was eventually discovered that the data sheet was incorrect in places; however after determining what the correct commands were, the module still didn't work. Eventually one of the support staff on manufacturer's forums suggested that the module may be a pre-FAT support version and after reflashing its firmware the module worked. It was now possible to read data from the sonde and log it to the SD Card; however, the SD Card would intermittently stop working.

It was decided that in an effort to track down the bug, all of the code would be reworked. However, neither the author nor several other people who offered to help were able to determine the problem. Eventually after examining the transmit and receive lines on an oscilloscope and it was noticed that they were occasionally being pulled low, although there was nothing in the code that would cause this behaviour. After several more days of investigation it was discovered that the pins were used as voltage reference pins for a hardware feature not used in the project. By explicitly turning off this module the problem was fixed and the SD Card worked reliably.

6.4 Iteration 4 - Real-Time Clock

This iteration was all about data formatting. Firstly, the data from the sonde is constructed from code/value pairs so that almost 50% of the data is not actually of any use. For example "22, 0.021" means that the depth in meters is 0.021, since the order of these pairs doesn't (and can't) change there is no need to keep this data. A parser was written that stripped out the codes and left just the values separated by commas. The next problem was that the Sonde's internal clock tends to be rather unreliable. This is what the RTC module is for, to provide accurate time so that the sonde readings can be correctly referenced temporally.



	BIT7							BIT0	
00H	CH	10 SECONDS			SECONDS				00–59
	0	10 MINUTES			MINUTES				00–59
	0	12/24	10 HR A/P	10 HR	HOURS				01–12 00–23
	0	0	0	0	0	DAY			1–7
	0	0	10 DATE		DATE				01–28/29 01–30 01–31
	0	0	0	10 MONTH	MONTH				01–12
	10 YEAR				YEAR				00–99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	

Figure 6.2: The register layout for the RTC module, values are stored in BCD form which seems to be an unnecessary complication.

The module uses i^2c to communicate and so there was a choice of using either the hardware or software based i^2c on the PIC. As the pins were not needed for anything else the hardware based i^2c was selected for its easier set up.

The RTC value is stored at the following times; at the start of an ascent the value is stored in the main datafile and after each Sonde reading it is stored in the appropriate message file. Reading the data and time from the RTC module is simple but the values returned are in BCD format and certain bits are control bits. To convert into text a byte is read, half masked off and the bit shift operator used to shift the byte 4 bits to the right. This value is stored as an integer and then combined with a different masked original byte. Once the RTC module has had its oscillator started it will continue to operate until it is disconnected from power and its backup battery is removed. It was therefore decided that just functions to get the RTC module would be written.

6.5 Iteration 5 - States & Iridium Satellite Modem

By this point there was enough working hardware to start work on implementing the state diagram. The implementation went well and took far less time than the author had anticipated, within a day it was working with the Iridium modem being simulated using two buttons. When in the transmit state one button was used to simulate the message being sent and another to simulate the message failing to send. To be able to test the states the depth value needed to change. To achieve this an artificial depth value was incremented and decremented during the sampling and dive states. Testing was carried out to ensure the states and transitions were working as intended.

The Iridium modem is by far the most complex hardware component in the project and so it was initially connected to a computer so that commands could be sent manually. The first problem was that the modem would default to a baud rate of 19200 which the software serial port used on the PIC couldn't handle. While looking for the command to change the default baud rate, commands to turn off echoing (previously the modem echoed everything typed at it) and turn on numeric responses were found. These were then saved to a new profile which was set as the default. There are two ways of sending the message text to the modem. The first is as part of the command: AT+SBDWT="message text", the second is after the command: AT+SBDWT CR (Carriage Return) "message text" CR. The first option limited the length of the message to 100bytes but the second was limited only by the maximum size of the message, 340 bytes. Before a message could be sent a buffer large enough to store it in was needed, unfortunately the linker fails to link files that are "too big" which makes declaring large arrays almost impossible. The linker file had to be modified to allocate a specific portion of memory 340bytes in size to the buffer which was then referred to in the code. An attempt to send a message was made, which despite being inside the lab, succeeded. The modem was then connected to the rest of the hardware and an attempt made to send a message using the PIC. This time it didn't work. Further investigation showed that the modem didn't like being sent the newline character, although no reason could be found in the documentation, and would stop responding. The fix was simple, just to change the line delimiter in the message files to a '#' after which the modem sent the message.

6.6 Iteration 6 - Power Electronics and Actuator

The next task was to control the actuator and power on/off the SD Card and Satellite Modem. As the SD Card and the Satellite Modem use very little power, mini relays were used which can be run at 5v. At first it seemed to work but the PIC would occasionally stop working, which was due to the amount of current being sourced by the relays. The PIC can only "source" so much current (30mA) from its inputs and, by exceeding that limit, its

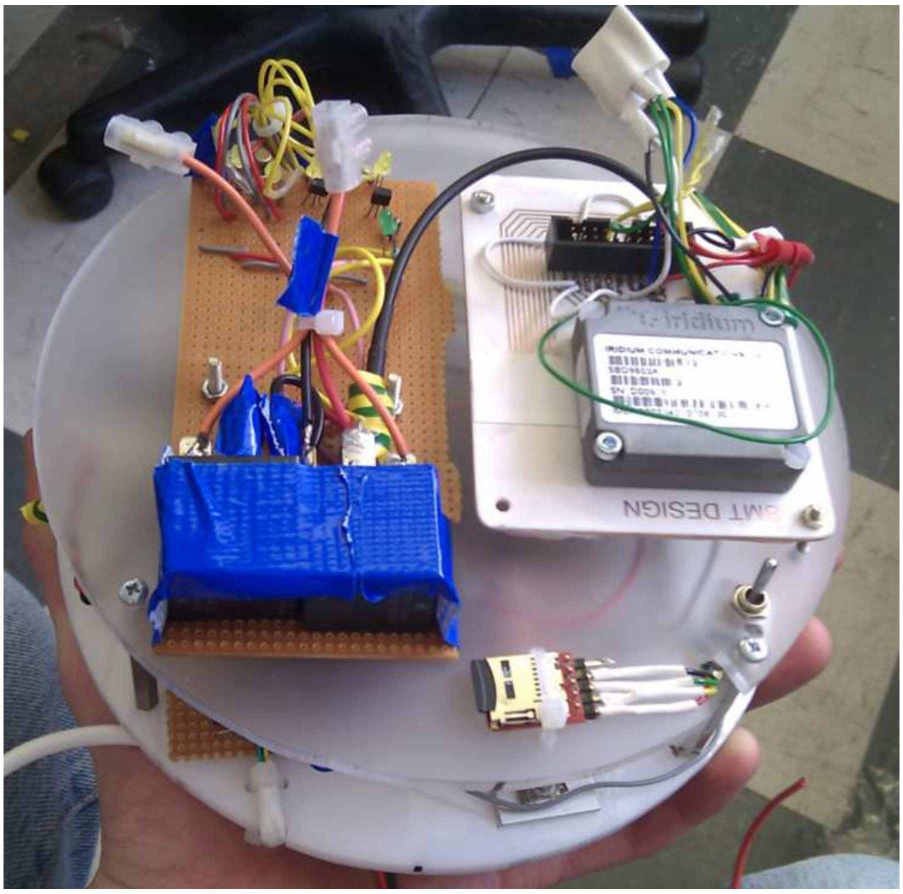


Figure 6.3: The top layer of the control system board which contains the power control circuitry, the satellite modem and the SD Card module.

behaviour became unpredictable. The solution was to add a transistor to switch the coils of the mini relays which dropped the sourced current from 16mA to 5mA per relay. For the actuator relays the circuit was very similar, the main difference being that the relays were rated for higher switching currents and so required a much higher coil current (150mA) which in turn required a slightly different transistor.

After several consulting with his supervisor the author decided to try to use a pump to control buoyancy as it offered a much higher change in buoyancy and could handle greater depths. However, after several weeks of work the pump wasn't performing as well as expected. The problem was that the pump was designed for moving hydraulic actuators and relied on the fluid going into the pump being pressurised. The decision was made to return to the actuated piston method of buoyancy control replacing the bulky pre-made actuator with a geared electric motor. Once the actuator mechanism was built, a motor sufficiently powerful was needed, all of the motors that were readily available had plastic gears and would break quickly under the loads involved. A suitable motor was found online and ordered. Due to stock issues it took around a month to finally arrive which delayed the project significantly as it was the only remaining item needed to construct the buoy.

6.7 Iteration 7 - Body Construction

While waiting for the actuator motor to arrive all of the machining and construction work, that could be done without needing the motor, was done. Rounds of 15mm thick acrylic were cut out and mounted to a lathe. A "step" was cut into them so that they would be a very tight fit in the acrylic tube, doing this to the lids took several hours per lid. The bottom lid was then mounted on a milling machine and two holes cut, one for the actuator tube and the other for the Sonde. The actuator hole had to be exactly the right diameter so that the acrylic glue would make a strong and watertight seal.

All of the electronics were disassembled and re-mounted on two stacked disks which fit into the acrylic tube. This enabled the easy removal of all of the electronics if required. Several brackets were glued to the inside of the tube to support the internal bulkhead, the battery and the electronics board. When the actuator motor finally arrived it was mounted to the actuator tube and the bottom lid glued into the body. Two aluminium clamping plates were then cut out, the top plate with a hole for the Iridium modem and the bottom plate with two holes for the Sonde and actuator tube.

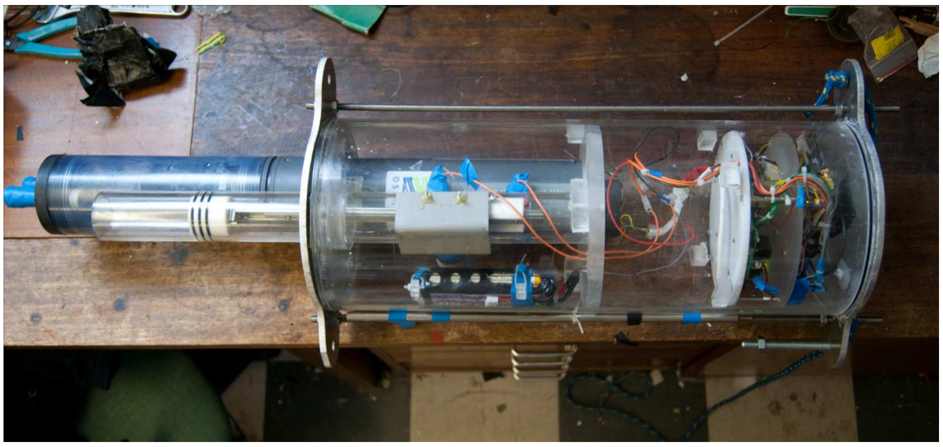


Figure 6.4: The finished buoy. On the left is the Sonde and the tube/piston used to change buoyancy. Inside is the test battery and the control electronics.

6.8 Remaining Issues or Known Bugs

There are a few remaining issues that the author acknowledges exist but are mainly limitations he doesn't have the time or expertise to work around or fix. The first is the inability to detect obstacles above the buoy, which was a requirement. Most of the code and hardware is in place to use sonar to do this but as previously mentioned it was decided that sonar was not the best way of doing this. Ideally something low power, with no moving parts, that isn't effected by dirt accumulation or algae growth would be used but there wasn't time to reinvestigate the issue.

The second known remaining issue is that of debug output, it was decided early on during the project that the only debug output would be via a serial port on the PIC. Although it would have been possible to log this information to a debug file on the SD Card it was felt that this could have adverse effects on the control system and could be a potential source of bugs or other problems. The use of a hardware serial port using a library written by Microchip is less likely to contain bugs or stop working than using code to write to an SD Card. The downside of this decision is that debug output cannot be viewed while the buoy is operating in the field. Construction of a serial port logger device that would fit into the buoy was considered but the author felt that there wasn't enough time to finish and test such a device.

7 Testing

In this chapter several kinds of test are presented; hardware tests, in-lab system tests and field tests. The first are designed to test the functionality of individual pieces of hardware and were used as validation that the hardware was functioning correctly. The system tests are tests of the whole system working in the laboratory, which provided the opportunity to test the control system in a setting where debug output could be viewed and changes could easily be made. It also afforded the opportunity to test the system's error handling ability, by disconnecting various pieces of hardware in-turn and examining the control system's response. After these tests were performed field testing was carried out, for which the whole system was tested in a lake. These tests were run unattended, with no debug output or possible intervention other than recovering the whole system. During all of the tests described above, the depth value was simulated. For hardware and lab tests this was due to not being able to get a varying depth value any other way. It would have been possible to use the real depth value during the field tests but given that the water was only 5-12m deep the simulated depth was used so that it could be eliminated as the source of any potential failures. Acceptance tests were planned but the members of the glaciology group who are interested in the project were on a research trip in Greenland.

7.1 Hardware Tests

Hardware tests were performed in the lab, running just the code necessary for the test. Where possible simulated data was not used so that the tests were more representative of the final system. Every test had to be passed for the system to function as intended and these tests were frequently used during development to determine when a piece of hardware was functioning correctly. A simplified list of these tests is shown below.

1. Sonde - Wake up, send data and sleep.
2. SD Card - Set baud rate, initialise card, write file and read file.
3. RTC - Read time.
4. Satellite Modem - Write to buffer and transmit message.
5. Actuator - Move in, move out and stop.

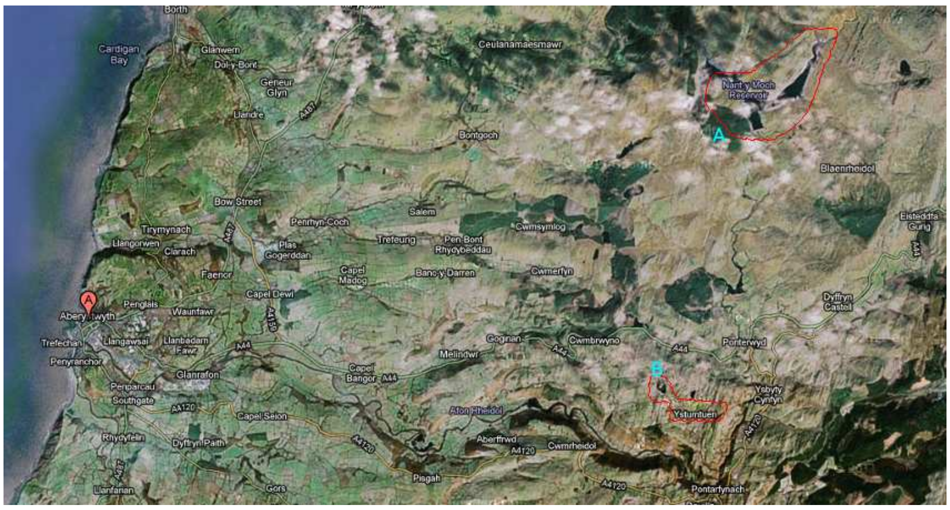


Figure 7.1: The location of the two field testing locations, Nant-y-Moch(A) and Ystumtuen(B).

7.2 In-lab System Tests

Several sets of tests were performed by running the whole system in the lab to verify that the control system and hardware were working correctly. This provided opportunities to find and fix bugs and once they had been competed the final system tests could be carried out. During these tests the system was powered from a power supply rather than a battery so that there would be no need to repeatedly recharge batteries. The Iridium modem would occasionally be able to transmit but usually could not get an adequate signal.

7.3 Field Tests

7.3.1 Location and Test Procedure

The field testing was carried out in two locations; Nant-y-Moch Reservoir and a lake near Ystumtuen. The water the buoy was tested in, was between 5-12 meters deep in Nant-y-Moch and clear enough to see the buoy while it was underwater. In Ystumtuen the water was quite shallow, only 3-4 meters deep. The reason that field tests were not conducted in the sea was due to the complications such locations would add to the testing procedure.

To make the buoy neutrally buoyant extra weight needed to be added. To simplify testing, a the majority of the extra weight needed to make the buoy neutrally buoyant was tied onto the bottom of the buoy. Any additional weight was in the form of stainless bolts that were added one by one until the correct buoyancy had been achieved.

The procedure for testing was as follows:

1. Clear the SD Card of all Files and insert into SD Card module.
2. Connect the battery positive and ground to the system, connect the two actuator wires, the Sonde and the Iridium modem's antenna.
3. Apply acrylic sealing agent around the rim of the lid and power the system on.
4. As the system turns on, hold switch 3 to reset the EEPROM variables.
5. Put the electronics into the body and face the Iridium antenna upwards. Let the system do at least one full cycle.
6. Place the lid on and tighten the clamping bars on both sides to compress the lid.
7. Place the buoy in the water while in its transmit state and add weight until it is only slightly buoyant and still on the surface.
8. Let the buoy rise and fall for as long as possible, occasionally checking for leaks.
9. Remove buoy from the water, undo the lid and turn off the electronics.
10. Copy the files on the SD Card to a laptop.

7.3.2 Results

The first field test was carried out near the dam in the Nant-y-Moch reservoir (Appendix E.1 point A). The water was just deep enough to allow the buoy to submerge to a depth of around 20cm. The tests on the shore revealed a bug with the actuator that resulted in it not moving every time it should. The actuator would extend without fail but would not always retract. At the time it was suspected that to be a wiring issue and so the buoy was left in the water for 10 minutes to test the water-proofing, the weight distribution and other aspects of the control system. While looking at the data after the test it was noticed that the date/time returned by the RTC clock was "RTC:01/1/00-00:00:00" which meant that it had become reset at some point. This was believed to have been caused by accidentally connecting the battery the wrong way around, something that had happened before. By looking at a message file from one of the lab tests, an offset of +1hrs 58min 16secs was calculated to correct the Sonde's time to UTC (Timezone used by Iridium).

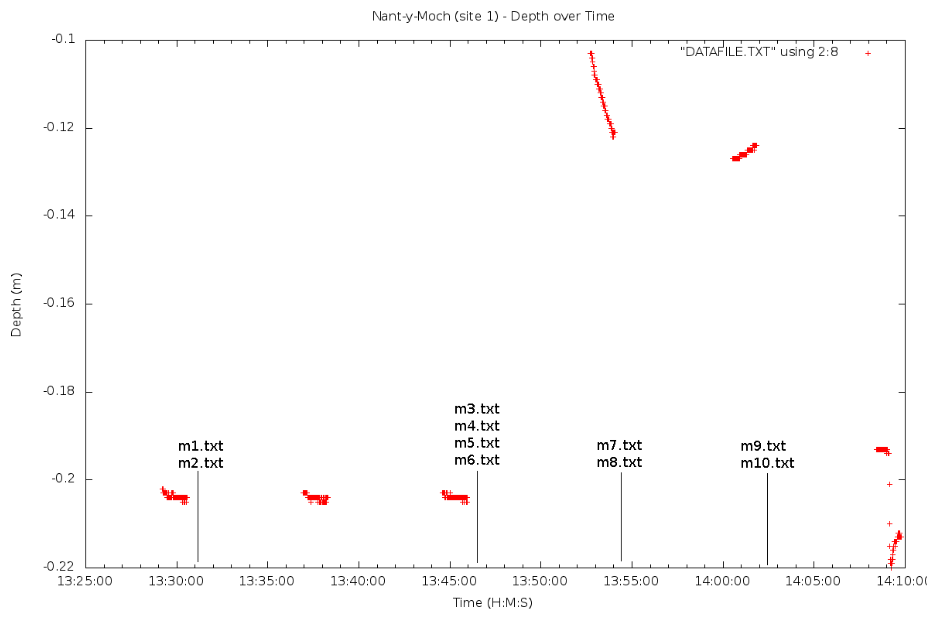


Figure 7.2: Depth of the buoy during it's sampling state as measured by the Sonde. The first 3 sets of points are relatively flat which indicates that the buoy was not yet sealed and in the water. For the next two sets the buoy was in water and the last set was during the recovery of the buoy. The times that satellite messages were sent are also marked.

All but the last two messages were sent successfully and the queueing behaviour can be seen when messages 3-6 are sent due to lack of reception during the previous attempt. The reason that the last two messages were not transmitted is that this was while the buoy was being disassembled on the shore and the antenna was facing the ground. Despite the actuator not working the rest of the system can be seen to be working (Figure 7.2)with readings being taken and messages being sent.

The second test was performed further along the reservoir(Appendix E.1) where the water was 5-12 meters deep. The RTC clock displayed the same problems as it did during the previous test, due to the lack of equipment to reset it. The actuator initially displayed the same problems as before but started functioning as expected once the buoy was in the water. The buoy was left in the water for approximately 25 minutes, during which it ascended and descended four times before it was removed and turned off. During this time everything seemed to work, the only problem was that the buoy was slightly too heavy which meant that when "surfaced" the top of the buoy was just underwater by approximately 2cm.

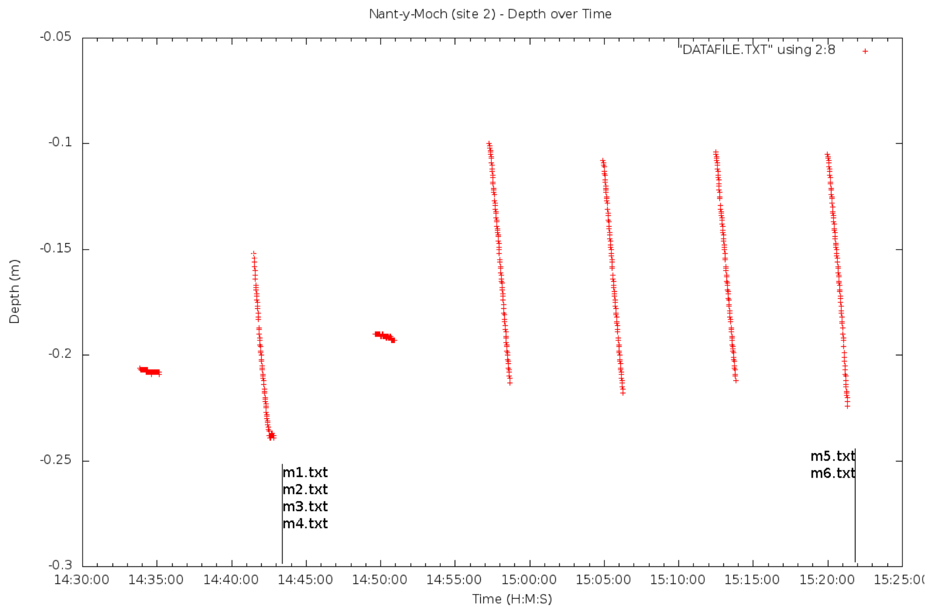


Figure 7.3: Depth of the buoy during it's sampling state as measured by the Sonde. The first three sets of points are recorded during assembly on the shore, the rest while the buoy was in the water. The times that satellite messages were sent are also marked.

It can be seen in Figure 7.3 that the buoy almost failed every time when trying to send messages while it was in the water, this was most likely due to the signal attenuation caused by being just below the surface when trying to transmit. Another possibility is that there weren't many satellites overhead at the time, something that can occur with a non-geostationary constellation of satellites.

The graphs in Figures 7.2 and 7.3 show the depth of the buoy plotted against time and appear to show a seemingly perfect change in depth. In fact it is too perfect and it wasn't plausible that the buoy had behaved as shown in the graphs, after some careful consideration it was determined that the depth sensor may be on a section of the sonde that was inside the buoy. After examining the Sonde this was found to be true. Every time the actuator moved in or out the pressure inside the buoy would increase/decrease which was read as a change in depth by the Sonde. For the next test the Sonde was moved so that the depth sensor was outside of the buoy. See Appendix C.1 for more details.

Before the third test in Ystumtuen lake (Appendix E.2), the RTC clock was reset and started again and the actuator wiring checked. During the initial on shore test the actuator displayed the same problems as in the first two tests, this time further investigation revealed a bug linked to sending messages. The section of code responsible for transmitting messages contains an 'if statement' to check if there are more messages to send, the else statement lacked the function call to move the actuator. The poor signal quality in the lab meant that very few messages were sent and therefore this section of code was never run. Since there was no way of reprogramming the buoy at the lake, the solution was to disconnect the Satellite Modem's antenna so that the modem would always fail to transmit but the actuator would move. The results shown in Figure 7.4 show that the depth was no longer being misread, however after 10 minutes of operation the buoy started to take on water and the test had to be halted.

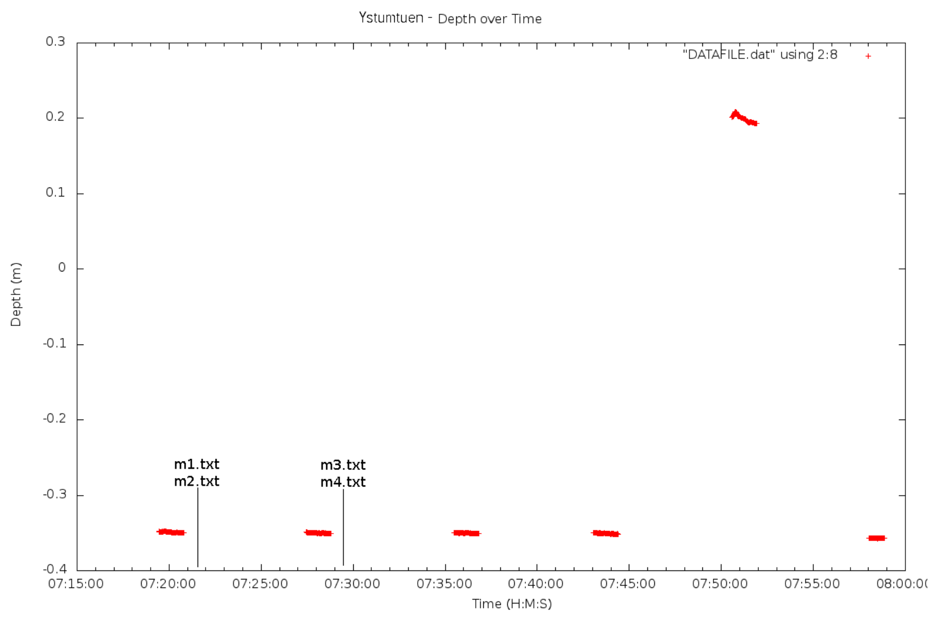


Figure 7.4: Depth of the buoy during it's sampling state as measured by the Sonde. For the first four and the last sets of points the buoy was on the shore, for the fifth it was in the water. The times that satellite messages were sent are also marked.

The graph in Figure 7.4 shows that the buoy was approximately 0.5m underwater which is consistent with observations during testing. Also visible is the intake of water which can be seen as a small increase in depth at the start of the ascent. Enough water was taken in that the buoy wasn't able to surface. The author believes the water leaked in from the seal around the Sonde which relied on compressing a rubber o-ring, if another student hadn't needed the Sonde it would have been possible to use acrylic sealant to form a better seal.

Graphs of NMEA data, examples of data files, message files, received message files and debug output are contained in the appendices.

7.3.3 Acceptance Tests

Although no formal acceptance tests were performed, the project supervisor knew enough about the intended use of the buoy that he could act as an end-user during testing. Although this isn't a substitute for real acceptance tests it was felt that his overall satisfaction with the system was a good indication that the Glaciologists would also be satisfied. However, it should be clear from the results presented that this is not a finished system ready for deployment. Before that could happen the buoy's body would most likely have to be custom made by an engineering company and more testing would need to be carried out for longer periods.

The tables below show the requirements as set out in Chapter 3 and whether the current system meets them.

Requirement	Pass/Fail	Comments
Sample water continuously at a rate of 1Hz.	Pass	When recording data is stored at this rate.
Transmit a reading of every 10m of ascent, store the rest of the data internally.	Pass	Occasionally one line of the transmitted data may be corrupt.
Water samples should include the following data: depth, temperature, conductivity. Other measurements such as dissolved oxygen and chlorophyll content would also be of use if available.	Pass	As well as the three required measurements, the sonde measures dissolved oxygen, chlorophyll, salinity and turbidity .
The buoy should be completely autonomous and should reliably recover from faults.	Pass	
The data should be transmitted in as close to real time as possible.	Pass	An attempt to send data is made after every ascent.

All data should be timestamped with accurate GMT time so that it can be matched to any events that occur.	Pass	The accuracy of the RTC clock is dependant on initially setting it to the right value.
The buoy should perform a profile every 6 hours.	Pass	Although not tested, the length the buoy sleeps can be varied to meet this value.
The buoy should be able to detect ice above it and stop its ascent, to prevent damaging itself.	Fail	Although most of the code and hardware is in place, the buoy cannot currently meet this requirement.
If the buoy cannot transmit the data collected, it should be stored and attempt to send it again as soon as possible.	Pass	
The buoy should be tethered.	Fail	Although considered during design, the buoy was never tested while attached to a tether.
The buoy should be able to operate at depth of up to 200m.	Fail	The current hardware would not be able to survive such depths.
The buoy should be easy and quick to deploy and retrieve.	Pass	The ease of deployment is subjective, but very few steps are involved and little technical knowledge required.
The buoy should be of a comparable price to any similar products.	Pass	In its present form the buoy is substantially cheaper, a professionally engineered version would be roughly equal to current systems (Appendix F.1).
The buoy should have enough battery power to function for 1 year.	Pass	
The buoy should be re-deployable after recharging the battery and retrieving data.	Pass	As long as the buoy is undamaged.

8 Evaluation

8.1 Evaluation of the Final System

The final system, in my opinion, successfully meets most of the requirements set out in Chapter 3. Tables shown in section 7.3.3 show that the system meets 12/15 of the requirements which is 80%. The buoy is able to ascend and descend, take water quality measurements, send satellite messages and recover from faults all to the specifications provided. Despite the high percentage of requirements met, there are several requirements that were not met, that were considered to be important. The buoy's ability to detect objects above itself was an integral part of the project and was one of the main features that would have set it apart from other existing products. It is also a crucial feature if it were to be deployed in its intended environment. The choice to use sonar was one made early on in the project and was agreed on by myself, my supervisor and the Glaciologists interested in the system. It was only until much later on in the project that it was decided that this technology was not really suitable. For this reason there was no time to come up with an alternative. The system is still designed to work with some kind of obstacle detection device, the control system currently uses a switch to simulate a detected obstacle, but there is no hardware in place to sense real obstacles.

The two remaining failed requirements; that the buoy should be tethered and that it should be able to survive 200m depths, which are not considered to be major failures. This is because both of these requirements were considered during the design phase and the reason the final system does not meet them is mainly due to a lack of suitable engineering facilities and equipment. To survive such depths the buoy's casing and actuator would need to be remade by engineers with experience in depth proofing, the author simply does not have the necessary skills or resources. In the case of the tether, the buoy was designed to work with a tether but due to limited testing time one was not used.

There are several other issues that are not in the requirements but still merit discussion. The choice of a PIC microcontroller seemed to be correct at the start of the project; however, by the end of the project its limitations had become apparent. Nearly all of the i/o pins were in use, to communicate with all of the hardware three software serial ports were needed and failings of programming language used had become apparent. It is the author's opinion that going with a “bigger” PIC or a different type of microcontroller would not be the right choice, the PIC used does work and using almost all of its features would be considered a good thing by many software engineers. The use of multiple software serial ports is not ideal as the maximum baud rate they are capable of is 4800bps, which is much slower than most of the other hardware was capable of but sufficient for the purposes of this project. The longer the author used the C18 compiler and the version of C used to program the PIC, the more poorly documented/undocumented bugs or “features” were found. The fact that Microchip’s implementation of the printf function couldn’t print floats or longs took the author days to figure out.

The chosen method of buoyancy control is something that the author is quite proud of; instead of using an off the shelf linear actuator a “home-made” actuator was built. This meant that the actuator used 200mA rather than the 15A of current the alternative actuator used, which is 75 times less. This is a phenomenal reduction, the effect of which is even more dramatic as in my estimated power usage (Table 5.1) the actuator consumed more power than anything else by a large margin. This would allow much fewer batteries to be used if the buoy were to be deployed.

Something that was noticed when looking through the results of the tests was that the buoy didn’t send messages at every opportunity. As mentioned in the testing section this may have been due to a poor signal, but it is felt that it would have been a good idea to mount the antenna higher up and retry sending a message a few times before giving up. This may have resulted in more messages being transmitted. Something else that was considered to improve the amount of data sent and reduce costs was data compression. Currently a maximum of three NMEA strings and three RTC times can fit into a message. By compressing this data and using the full 340 bytes available, between 5 and 8 NMEA strings and RTC times could fit into one message. The PIC’s limited processing power and RAM limits the types of compression algorithms to relatively simple ones but it is something that is definitely possible.

Most of the issues described above could be resolved if the author had had a little more time to work on the system. If the project were to be started again, the author would not change many things. The requirement to sense obstacles above would be considered in more detail and more time would be set aside for testing. As it stands the system should be considered a “final prototype” and after some more testing and bug fixing it could be deployed for real in Greenland.

8.2 Personal Evaluation

For me, the project has been very enjoyable, challenging and incredibly interesting. I have learnt a huge amount from it, about many different aspects of hardware and software. I have not struggled with keeping myself motivated, despite some infuriating and complicated problems, at any point during the year.

For the duration of the project the I have made steady progress. Initially due to the number of other modules the I was taking, the amount of time available to work on the project was relatively small. During the second semester, however, I had a lot more free time and so dedicated the majority of it to the project. This approach paid off and I made significant progress during my second semester, the only stumbling block was the delay in getting the motor required for the actuator. This took a month longer than expected and while I strived to fill this intervening time with work on other aspects of the project I quickly found myself with nothing to do but wait for the motor to arrive. I feel that this highlights some of my weaknesses, which are not planning well ahead and sometimes not being quite as proactive as I could be. Had I considered the need for such a motor at the start of the year I would not have been delayed for so long. I also could have been a more proactive in finding out what the delay was and trying to resolve it. I really would have liked to have had more time to test the system, although I do not think I could have worked any harder. I feel that my expectation of having a fully working and deployable system may have been a bit ambitious.

In summary I feel I have learnt a great deal over the course of the project and have greatly developed my hardware and software, design and implementation skills. I also think that I have achieved a lot, in terms technical achievement, in the project. I started the project thinking that connecting all of the hardware together and getting it working, would simply be a case of putting wires in the right place and reading the manual. I have learnt, however, that this can actually be the hardest part of the development process and that there is always an unpredicted problem. I am very proud that I single-handedly managed to write the software, design and build the electronics and most of the mechanical structure of the system. I think my ability to self motivate is what enabled me to progress as far as I did and resulted in meeting most of the system requirements.

Bibliography

[1] "Drifting buoys." [Online]. Available: http://www.pacificgyre.com

> The website of a company that makes drifting data buoys, provided good information on why such buoys are used and what technologies they use.

[2] "Extreme Programming For One," Internet, http://xp.c2.com/ExtremeProgrammingForOne.html.

> An attempt to make eXtreme Programming work for 1-2 person projects. Although interesting it is not particularly convincing.

[3] "History of sonar." [Online]. Available: http://inventors.about.com/od/sstartinventions/a/sonar_history.htm

> A very brief overview of the history of sonar, leaves out the role of Fessenden and the Fessenden oscillator.

[4] "Iridium constellation." [Online]. Available: http://www.iridium.com/About/IridiumGlobalNetwork/SatelliteConstellation.aspx

> An article detailing how the Iridium constellation works and various technical details of the satellites.

[5] "Slocum glider: Design and 1991 field trials," 1991. [Online]. Available: http://www.webbresearch.com/pdf/SlocumGlider.pdf

> The outcomes of testing and initial designs mentioned were criticial in understanding how and why the glider was developed and how it worked.

[6] "MPLAB C18 C Compiler Libraries," 2005.

> The C18 library reference manual. It contains descriptions of functions and some small sections of example code.

[7] *Seaweb Acoustic Communication and Navigation Networks*, 2005.

A fascinating article on the construction and design of a network of underwater equipment, including gliders, AUVs data buoys and relay buoys. Communication between different elements is by acoustic modem and data is relayed out of the network using Iridium satellite modems on dedicated gateway nodes. The article provides an insight into technologies and strategies used and was very useful in making certain design descisions.

[8] *IEEE Journal of Oceanic Engineering*, 2006-2010.

This journal contained countless interesting papers on emerging technologies and current research work in the field of oceanic engineering. Nothing was directly taken from the journal but it provided a lot of food for thought and gave a good overview of the research area.

[9] "PIC18F2455/2550/4455/4550 Data Sheet," 2009.

A very useful document detailing every aspect of the 18F4550 PIC. Contains very detailed information on how to use each feature.

[10] "Argo homepage," Internet, November 2010, http://www.argo.ucsd.edu/.

The homepage for the Argo project. The whole site is very interesting and contains a lot of information on the Argo project.

[11] Archimedes, *Works of Archimedes.* Cambridge University Press, 1897.

Material on the fundamentals of buoyancy is quite hard to come by but this book provides a good overview of the subject. Some elements are mathematically heavy and can be tricky to understand, however diagrams are used throughout and help a lot.

[12] P. Beynon-Davies, C. Carne, H. Mackay, and D. Tudhope, "Rapid application development (RAD): an empirical review," *European Journal of Information Systems*, vol. 8, pp. 211–223, 1999.

The paper gives an overview of the RAD development approach and details several case studies of its use. Very useful in helping determine the effectiveness of RAD.

[13] A. Clements, *Microprocessor Systems Design*, 3rd ed. PWS-Kent Publishing Company, 1997.

Provides a fascinating insight into what actually is inside a microcontroller and the benefits and disadvantages of using a micontroller. It also was helpful when deciding if programming in C would be the right option, the added complexity of implementing high level features in assembly described in the book made me see how much extra work it would be.

[14] R. D. Y. D. Dean Roemmich, Stephen Riser, "Autonomous profiling floats: Workhorse for broad-scale ocean observations," *Marine Technology Society Journal*, vol. 38, pp. 31–39, 2004.

A paper published about midway through the deployment phase of the Argo Float network, provides a good background to why the project was conceived and how the deployment is going. Also describes and explains the technologies used in the floats.

[15] M. Fowler, *Refactoring: Improving the Design of Existing Code.* Addison Wesley, 2005.

The first few chapters provide a good explanation as to what refactoring is and why it should be done. It also explains, in later chapters, methods of refactoring. While reading it, it became apparant that tradeoffs between refactoring and following an embedded coding stadards would have to be made.

[16] R. D. e. a. Gould, J., "Argo profiling floats bring new era of in situ ocean observations," *Eos*, vol. 85, pp. 179,190–191, 2004.

An overview of the Argo project including deployment strategy, progress and brief results analysis.

[17] J. Guold, "From swallow floats to argoo - the development of neutrally buoyant floats," *Deep-Sea Research*, vol. 2, pp. 529–543, 2005.

The history of neutrally buoyant floats. A fascinating insite into the history of such devices and how they developed into the Argo floats that are now so prevalent.

[18] O. T. N. T. Hosoda, S., "A monthly mean dataset of global oceanic temperature and salinity derived from argo float observations," *JAMSTEC*, vol. 8, pp. 47–59, 2008.

A study of temperature and salinity data obtained from the Argo float network.

[19] A. Hubbard, J. E. Box, R. Bates, F. Nick, A. J. Luckman, R. van de Wal, and S. H. Doyle, "The kinematic response of petermann glacier, greenland to ice shelf perturbation," in *American Geophysical Union*, 2010.

A paper detailing the possible causes for the detatchment of a 275 square km area of the Petermann Glacier shelf.

[20] B. O. Klatt, O. and E. Fahrbach, “A profiling floats sense of ice,” *American Meteorological Society*, vol. 24, pp. 1301–1308, 2007.

The development of an algorithm to enable Argo floats to detect ice above using water column temperature measurements.

[21] T. J. P. A. Krishfield, R. and M.-L. Timmermans, “Automated ice-tethered profilers for seawater observations under pack ice in all seasons,” *American Meteorological Society*, vol. 25, pp. 2091–2105, 2008.

A paper detailing the construction and testing of a modified Argo float which is tethered to moving sea-ice.

[22] P. Prinz and T. Crawford, *C in a Nutshell.* O’Reilly Media, 2005.

A very useful reference book on the C programming language. Some chapters are worth reading just to refresh the memory on certain topics, the rest is mostly useful to just dip into when a particular problem arises.

[23] D. Semiconductor, “Ds1307 serial real-time clock data sheet,” Internet.

The data sheet for the RTC module used in the project. This was very helpful in initially setting up the device.

[24] I. Sommerville, *Software Engineering 9.* Pearson Education, 2010.

Good reference book for software engineering practices and process models.

[25] D. Systems, “udrive-usd-g1 data sheet,” Internet, April 2009.

The data sheet for the SD Card Module, useful but contains mistakes.

[26] F. F. Tsui and O. Karam, *Essentials of Software Engineering.* Jones & Bartlett Publishers, 2006.

[27] J. S. Turner, *Buoyancy Effects in Fluids.* Cambridge University Press, 1980.

An incredibly complex book on buoyancy effects, 99 percent of which I couldn’t understand, but there was the occasional page that was very useful in understanding buoyancy.

Appendices

A Hardware On/Off State Guide

Hardware	Transmit	Dive	Sleep	Sample
SD Card	On	Off	Off	On
Sonde	Off	On	Off	On
Sat Modem	On	Off	Off	Off
PIC	On	On	Sleep	On
Sonar	Off	Off	Off	On
Actuator	Off	On	Off	On

Table A.1: The power state for every hardware module in each control system state.

B Battery Technologies

Technology	Energy Density(MJ/Kg)	Power(W/Kg)	Efficiency	Discharge	Life
Lead-acid	0.11-0.14	180	70-90%	3-4%	20
Alkaline	0.31	50	99.9%	<0.3%	<5%
NiMH	0.11-0.29	250-1000	66%	30%	?
Li-ion	0.46-0.72	1800	80-90%	5-10%	2-3
Li-Pol	0.47-0.72	3000+	?	?	2-3
Ni-Cad	0.14-0.22	150	70-90%	10%	?

Table B.1: Various attributes for different types of battery technology

C Plotted Sonde Data

C.1 Pre/Post Sonde Depth Problem

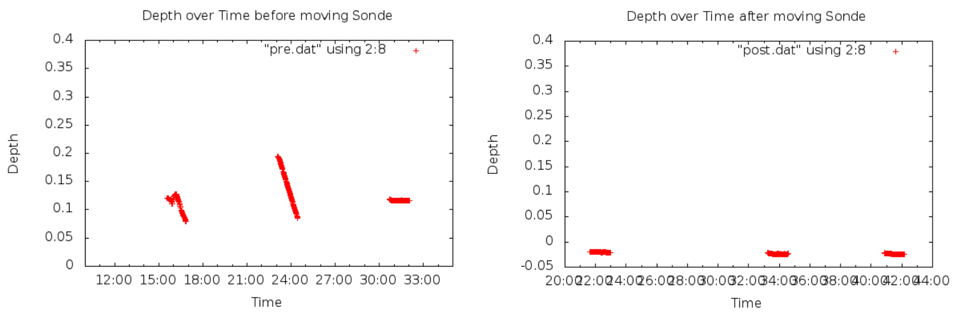


Figure C.1: The depth value as measured by the Sonde while in the lab, the effect of moving the Sonde can be seen in the second graph where the depth value remains relatively constant while the actuator moves.

C.2 Coductivity Data

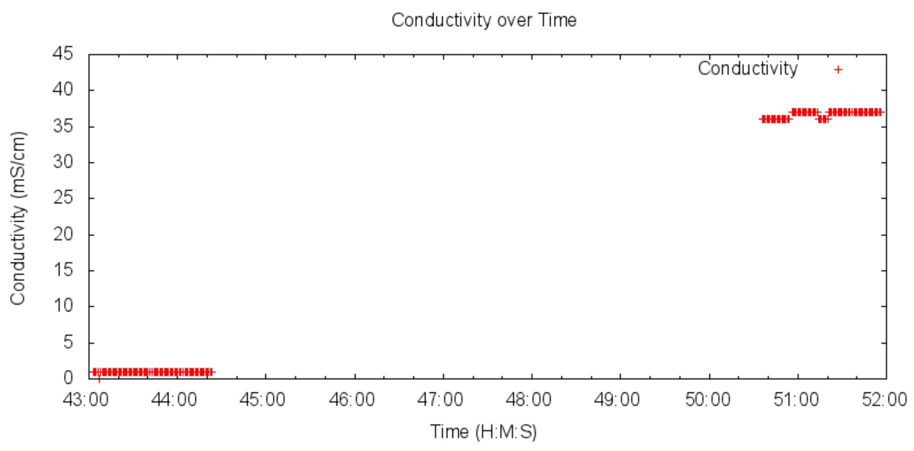


Figure C.2: A sample of conductivity data taken while sampling. The first set of data was taken while the buoy was out of the water, the second while in the water.

C.3 Chlorophyll and Oxygen Data

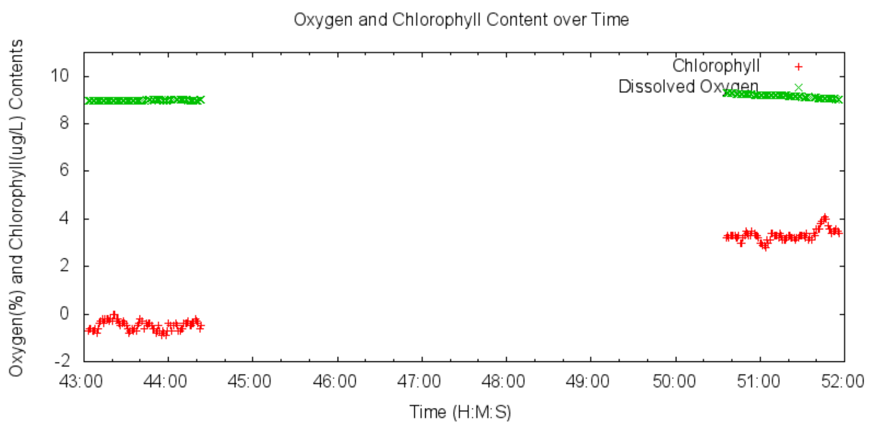


Figure C.3: A sample of Chlorophyll and Oxygen concentrations as recorded by the buoy. The first set of data was taken while the buoy was out of the water, the second while in the water.

C.4 Temperature Data

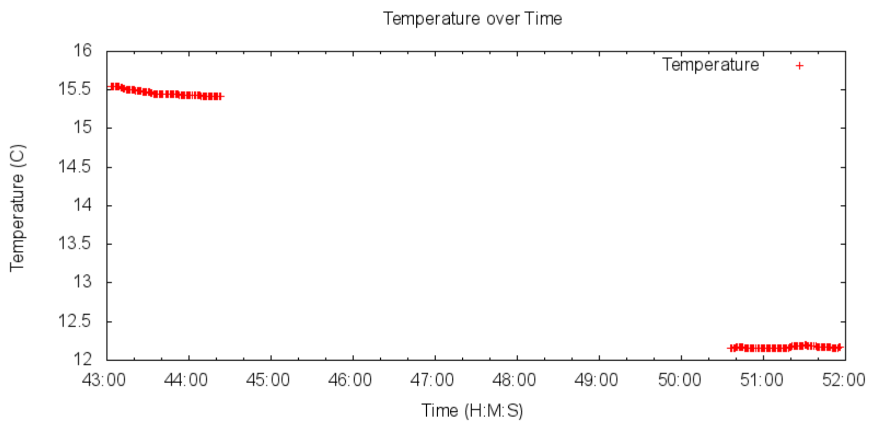


Figure C.4: Example of temperature variation during sampling. The first set of data was taken while the buoy was out of the water, the second while in the water.

D Buoyancy Control Graphs

D.1 Actuator Force Graph

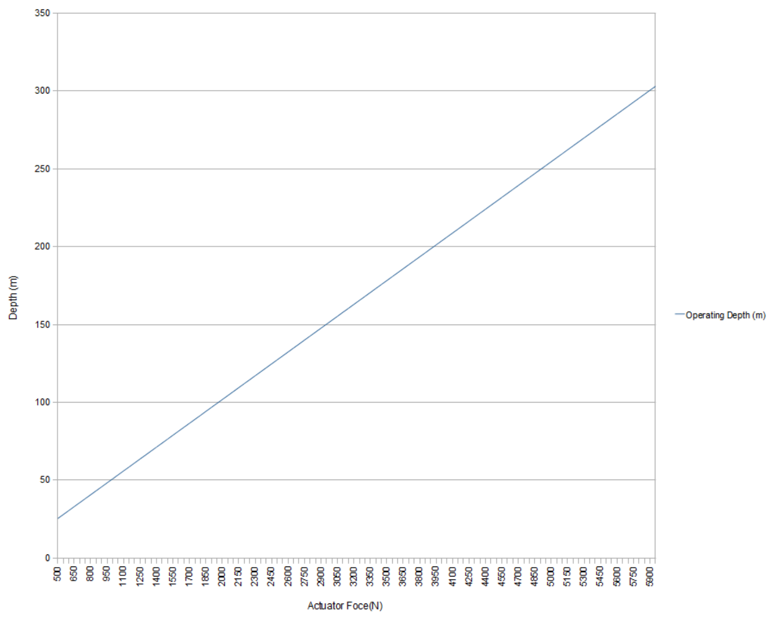


Figure D.1: The maximum depth that actuators with different push strengths(in Newtons) can move a 5cm diameter piston.

D.2 Air Tank Duration Graph

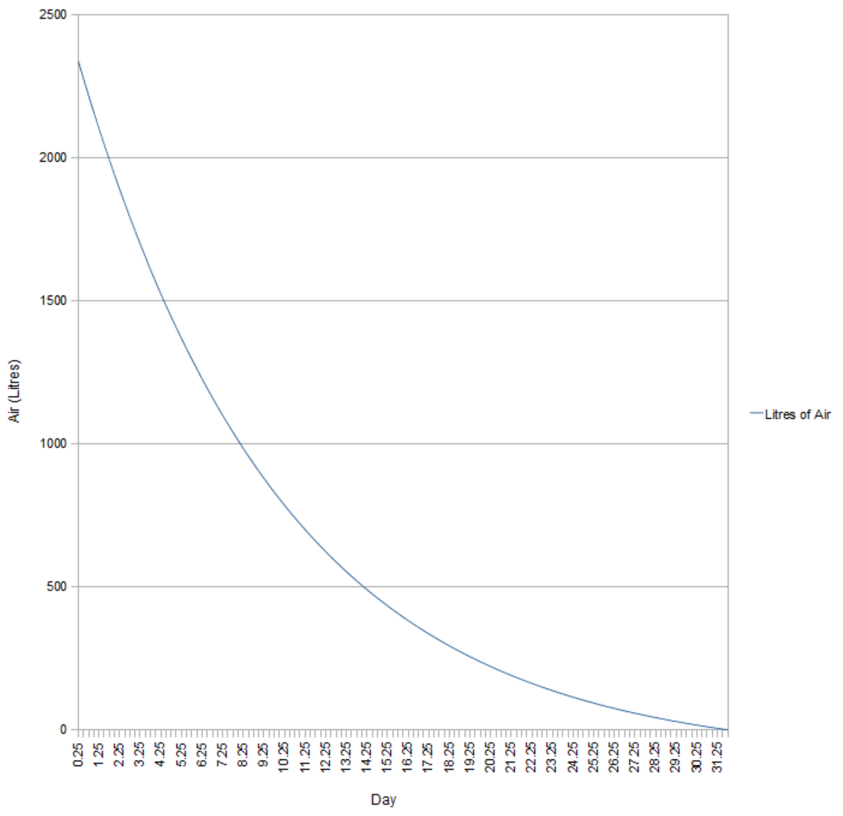


Figure D.2: The amount of air left in a tank that initially has 2400 litres (8 litres at 300bar) of air, after each ascent. For each ascent 125ml of positive buoyancy is being created.

E Maps

E.1 Nant-y-Moch

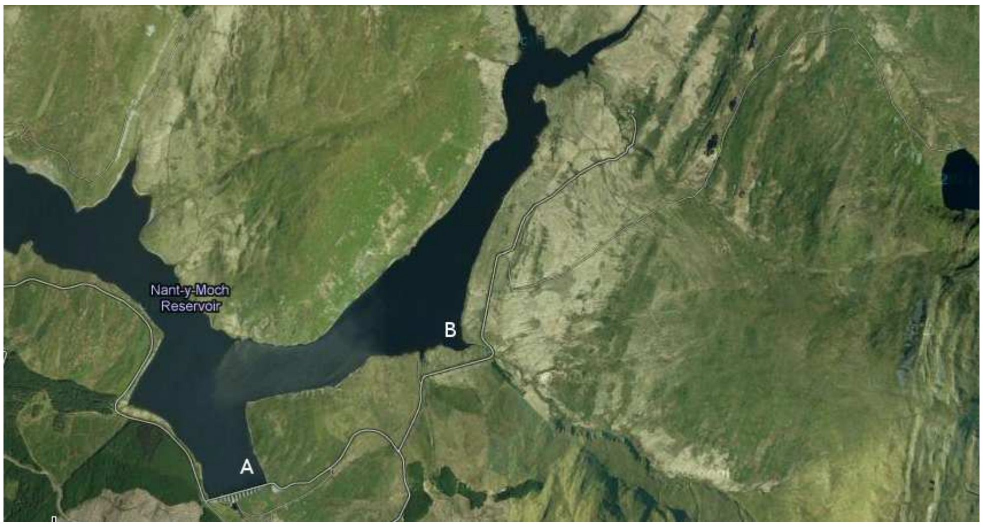


Figure E.1: Map of the Nant-y-Moch reservoir used for the first two field tests. Location A was the site of the first test and, B the site of the second test.

E.2 Ystumtuen

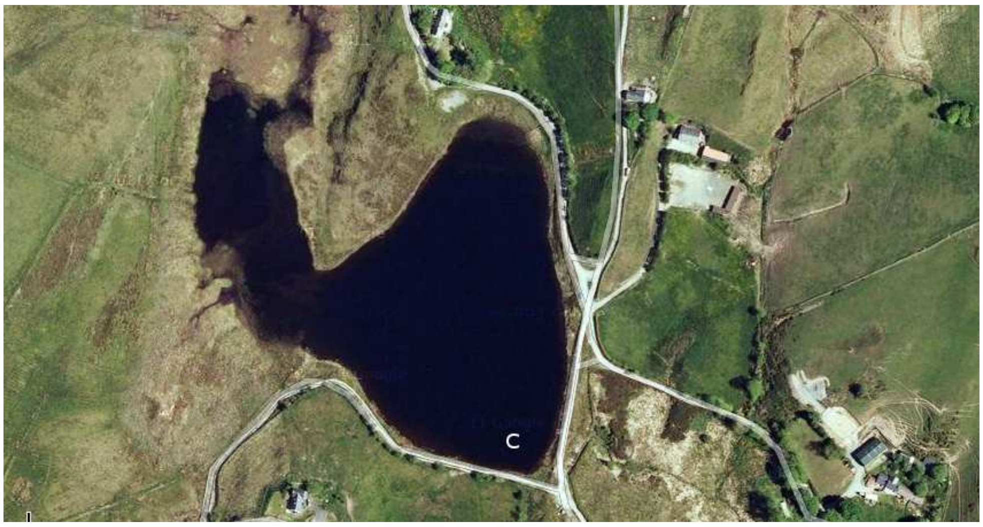


Figure E.2: Map of the Ystumtuen lake used for the third field test, the location of the test is marked ‘C’.

F Equipment Cost

Hardware Component	Cost (£)
PIC FS-USB Board	40
Iridium Modem	450
Micro SD Card module	18
2Gb Micro SD Card	5
Real-Time Clock module	9
Misc. Electronics	10
Misc. Materials	150
Total	682

Table F.1: The cost of all of the hardware components and materials used.

The Sonde (already owned, so not shown in table) is by far the most expensive piece of equipment required for the project. The estimated cost is between 6-11,000, this brings to total cost in-line with the cost of an Argo float.

The yearly Iridium data costs for the project would be quite substantial. Profiling 200m of water four times a day would, using the current system, require 27 messages per day. Each message contains approximately 240 bytes, each byte costs 0.1 cent, so one day would total \$6.48 and a year \$2365.2 (£1418.5).

G Example Data

G.1 DATAFILE.TXT

```
########################################
Example contents of a DATAFILE.txt file.
########################################
RTC:19/4/11-10:24:21
19/04/11, 07:27:27, 17.19, 0.001, 0, 0, 0, -0.349, -2.6, 0.3, 89.6, 8.62, 12.2
19/04/11, 07:27:29, 17.17, 0.001, 0, 0, 0, -0.349, -2.6, 0.1, 89.8, 8.65, 12.2
19/04/11, 07:27:30, 17.16, 0.001, 0, 0, 0, -0.35, -2.6, 0.1, 89.8, 8.65, 12.2
19/04/11, 07:27:31, 17.15, 0.001, 0, 0, 0, -0.35, -2.6, 0.2, 89.8, 8.65, 12.3
19/04/11, 07:27:32, 17.15, 0.001, 0, 0, 0, -0.35, -2.6, 0.3, 89.7, 8.64, 12.3
19/04/11, 07:27:33, 17.14, 0.001, 0, 0, 0, -0.35, -2.6, 0.5, 89.6, 8.64, 12.2
19/04/11, 07:27:34, 17.13, 0.001, 0, 0, 0, -0.35, -2.6, 0.4, 89.6, 8.63, 12.2
19/04/11, 07:27:35, 17.12, 0.001, 0, 0, 0, -0.35, -2.6, 0.6, 89.5, 8.63, 12.3
19/04/11, 07:27:36, 17.11, 0.001, 0, 0, 0, -0.35, -2.6, 0.5, 89.6, 8.64, 12.3
19/04/11, 07:27:37, 17.11, 0.001, 0, 0, 0, -0.35, -2.6, 0.4, 89.6, 8.64, 12.2
19/04/11, 07:27:38, 17.1, 0.001, 0, 0, 0, -0.35, -2.6, 0.3, 89.7, 8.65, 12.2
19/04/11, 07:27:39, 17.1, 0.001, 0, 0, 0, -0.35, -2.6, 0.3, 89.6, 8.65, 12.3
19/04/11, 07:27:40, 17.1, 0.001, 0, 0, 0, -0.35, -2.6, 0.2, 89.5, 8.64, 12.2
```

G.2 M0.TXT

```
########################################
Example contents of a message file.
########################################
19/04/11,07:27:27,17.19,0.001,0,0.000,0.00,-0.349,-2.6,0.3,89.6,8.62,12.2,
#RTC:19/4/11-10:25:43#
19/04/11,07:27:47,17.08,0.001,0,0.000,0.00,-0.350,-2.5,0.4,89.7,8.65,12.2,
#RTC:19/4/11-10:26:03#
19/04/11,07:28:07,16.97,0.001,0,0.000,0.00,-0.350,-2.4,0.2,89.7,8.67,12.3,
#RTC:19/4/11-10:26:23#
```

G.3 Debug Output

```
######################################
The following is the debug output for one complete cycle of the control sys-
tem.
######################################
Initialised UARTS
Powering on SD Card Module.
Sending SD Card Module Auto baud
Received: 6
Received ACK
SD Card module baud set
Sending SD Card Initialise Command
Received: 6
SD Card Detected and Initialised
Requesting NMEA strings (May take up to a minute)
Request Sent
RTC Initialised,RTC:19/4/11-10:16:23
Press Switch 3 to reset EEPROM variables
EEPROM Variables Reset.
Read EEPROM Message Number: 1, Sent Messages:1
Writing RTC to datafile
Iridium Off.
Recovery state: 3
Recovered from sleep failure.
Stored for Sat, Depth:40.0
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Stored for Sat, Depth:30.500
```

```
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Stored for Sat, Depth:20.500
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Stored for Sat, Depth:10.500
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Writing data to SD.
Stored for Sat, Depth:0.500
Writing data to SD.
Sending sleep command to sonde
Surfaced.
Iridium On.
messages_sent:1, message_numbers:2
Reading File:m1.txt
Sending AT+CIER=0
0
Sending AT+SBDWT
R
241
Sending AT*R1
0
Sending AT+SBDI
1
Message Sent
messages_sent:2, message_numbers:2
Reading File:m2.txt
Sending AT+CIER=0
0
Sending AT+SBDWT
R
194
Sending AT*R1
0
Sending AT+SBDI
1
Message Sent
messages_sent:3, message_numbers:2
```

```
Diving.
Depth:0.0
Diving.
Depth:1.0
Diving.
Depth:2.0
Diving.
Depth:3.0
# Depth continues to increase but has
# been removed to save space.
Diving.
Depth:38.0
Diving.
Depth:39.0
Diving.
Depth:40.0
Reached Resting Depth
Preparing to sleep.
Powering off SD Card Module.
Going to sleep.
I am awake!
```