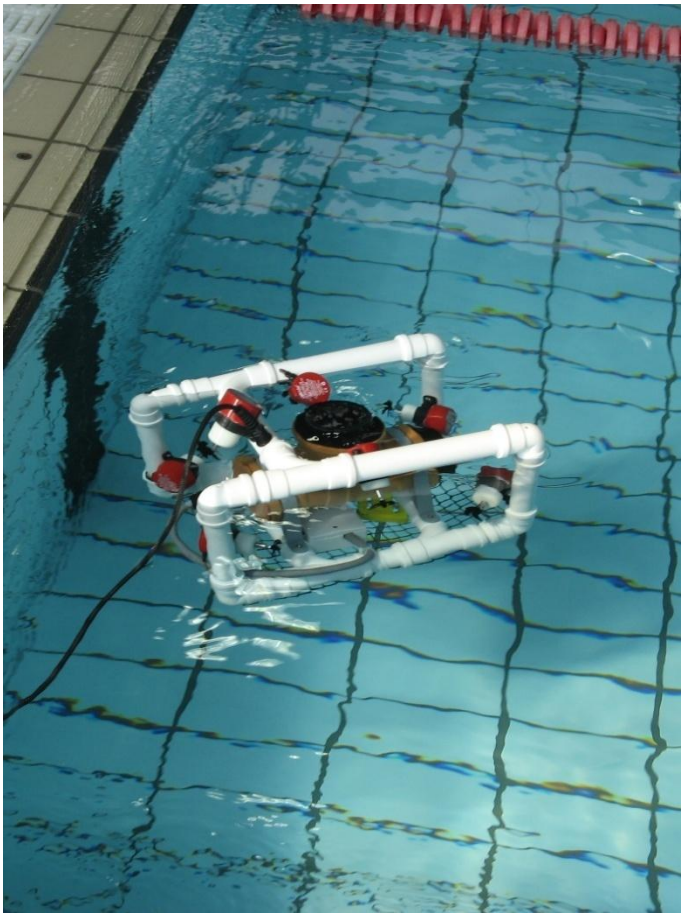
cover page goes here

Title:	Underwater Remotely-Operated Vehicle
Author:	Scott O'Brien
Supervisor:	Dr. Andrzej Tarczynski
Date:	May 2012
Course:	B.Eng. Electronic Engineering



SCOTT ROV

Abstract

The aim of this project is to design and construct an underwater, remotely-operated vehicle (ROV), fitted with a 4-axis positioning system, a video transmission system, and a 2-axis (pitch and roll) control system to assist imaging and positional stability.

The project includes the design and development of the PVC frame and water-proof enclosure, design and development of an ARM Cortex-M3 microcontroller-based electronic circuit for the operator interface, and another ARM Cortex-M3 microcontroller-based electronic circuit that controls the seven DC motors fitted with propellers.

A fully operational vehicle has been constructed, though the 2-axis control system and video transmission system remain incomplete.

Contents

Acknowledgements

I would like to take this opportunity to thank my partner Anne Donald, my Project Supervisor at the University of Westminster, Dr. Andrzej Tarczynski, and Jack Bowles and Colin Pullen from the Lodge Scuba Diving Club for all their support and help.

In addition, the following people helped considerably:

- Dr Mohammed Al-Janabi , Dr Viv Bartlett, John McNamara, Professor Izzet Kale, Karl Kowalczy and Mukesh Popat at the University of Westminster.
- Jane Clegg, Hazel Fraser, Fion Gunn, Kay Trotter, Carol Walker and Stephen Wiltshire from the Stanthorpe Triangle Residents Association.
- Paul Hoy, Simon Lodge, Sarah Morgan, Mitch Pluck-Wyatt, and John Porterhouse from the Lodge Scuba Diving Club
- Martin Evans
- Chris Styles and Simon Ford at mbed.com
- Robert Forsyth, Silogini Gnanasundaram, Raji Guhanesan, and Robert Karpinski at the University of Westminster
- Tim Marvin

And the following companies for their support:

- AbPlas
- ARM Holdings Ltd
- Elektron Technology
- Exar Corporation
- Keller UK Ltd
- Northern Connectors
- RelChron Limited
- Saab Seaeye
- SMD Ltd
- VideoRay LLC
- Woods Hole Oceanographic Institution

Glossary

Ballast:	weights used to offset buoyancy
Buoyancy:	the tendency to float
Centroid:	the mid-point between the centre-of-mass and centre-of-buoyancy
DOF:	degrees of freedom, the number of independent axes
Heave:	to move up and down
IC:	integrated circuit
IMU:	inertial measurement unit
IP68-rated:	an industry “standard” for rating environment-proofed components
kbps:	a data rate of kilobits per second
Manipulator:	a robotic claw or tool
Pitch:	as in nodding your head up and down
PWM:	pulse width modulation, used for proportional speed control
Roll:	as in tilting your head to one side
ROV:	remotely-operated vehicle
Surge:	to move forwards and backwards
Sway:	to move from side-to-side
Thruster:	motor fitted with a propeller, used for propulsion and positioning
Trimming:	positioning of ballast and buoyancy elements to level the ROV
Weight-in-air:	the weight of the ROV out of the water
Yaw:	as in turning your head to the side

1. Introduction

1.1 Why Build an ROV?

The purpose of this project is to design and construct a remotely-operated underwater vehicle (ROV) fitted with a 4-axis positioning system, and a self-stabilising 2-axis control system. The many varied aspects of this project present quite a number of challenges and learning opportunities:

- design and construction of an electronic operator interface
- design and construction of an electronic DC motor drive system
- design and construction of a water-proof enclosure mounted on an appropriate frame
- research, design and implementation of a level measurement system
- research, design and implementation of a self-stabilising control system

This ROV project ultimately represents an overlap of interests: control systems, underwater exploration, and electronics.

1.2 Why Include a Self-Stabilising Control System?

All objects in water, including underwater vehicles, will find their natural disposition in water due to gravity acting upon their mass and relative buoyancy. This position however may not be level, which in the case of an underwater vehicle makes positioning and imaging problematic. This issue is usually minimised by careful trimming i.e. locating of ballast and buoyancy elements on the vehicle, but this is not an ideal solution.

There are two main problems with relying solely on accurate trimming:

- Firstly, it cannot compensate for unpredictable external influences such as strong currents. It may not be obvious from the surface but undersea currents are typically not parallel to the water surface and commonly have a significant vertical component.

- Secondly, trimming at the surface prior to commencing a voyage does not allow for any load variation. If the ROV collects for example a soil sample, the variation in the mass (and the corresponding variation in the position of the centre-of-mass) will cause a tilt and so negate the careful trimming previously performed.

Most simple ROV's do not include any automated self-stabilising ability, due to the added cost and complexity, and simply accept the problems that occur. For larger, commercial ROV's however, particularly those that have manipulators where the mass distribution will vary during a voyage, a self-stabilising system would seem to be a requirement. Somewhat surprisingly, they are usually only found on the most expensive, most robust vehicles available. The "Seaeye" range from SAAB for example, currently has 9 models ranging in size from 60 kg through to 1,500 kg (weight-in-air) and it is only the biggest, most expensive model, the Jaguar, which is fitted with an automatic pitch / roll stabilising function [1].

Accordingly, an investigation into the design and application of a control system to augment the natural positioning of underwater vehicles, in particular the smaller vehicles, seems to be a worthwhile and practical undertaking.

1.3 Report Structure

The remainder of this first Chapter gives an introduction to the world of underwater vehicles, their classifications and their commercial uses.

Chapter Two gives a brief overview of the ROV system architecture that was developed.

Chapters Three, Four and Five detail the design and construction of the hardware and electronic elements.

Chapter Six follows with details of the self-stabilising control system implemented and Chapter Seven summarises the testing and tuning processes once the ROV was actually in the water.

The report conclusions are contained in Chapter Eight, followed by the official Project

Specification, References and Bibliography sections, and finally the Appendices.

1.4 An Introduction to Underwater Vehicles

Underwater vehicles can be broadly classified as either:

- Manned,
- Remotely-operated, or
- Autonomously-operated

1.4.1 Manned Underwater Vehicles

A manned underwater vehicle is one that contains a waterproof enclosure, pressurised to 1 atmosphere, suitable for human occupation. Examples include submarines for defence purposes, and ALVIN, a scientific research vehicle operated by the Woods Hole Oceanographic Institution (WHOI) shown in Figure 1.1.

Figure 1.1. ALVIN, a manned underwater vehicle, operated by WHOI [2].

1.4.2 Remotely-Operated Underwater Vehicles

A remotely-operated underwater vehicle is controlled by an operator who remains out of the water. The operator typically uses a joystick to manipulate the position of the vehicle in the water, and a video display to see the environment it is operating in.

This project focuses solely on this unmanned remotely-operated class of vehicle.

1.4.3 Autonomously Operated Underwater Vehicles

An autonomously-operated underwater vehicle is designed to work without an operator and without a direct connection to the surface. They are usually designed for a specific application and are pre-programmed to perform certain specific tasks such as sea-floor mapping and imaging, temperature and salinity measurement etc. Figure 1.2 shows the WHOI SEABED vehicle designed for optical and acoustic sea-floor imagining. Upon completion of the assigned tasks the vehicles typically surface, broadcast their location (and often their captured data set) via satellite communications, and await recovery.

Figure 1.2. SEABED, an autonomous survey vehicle, operated by WHOI [3].

1.4.4 Typical ROV Configurations

A typical commercial ROV will have some or all of the following attributes:

- Surface-based operation
- Video cameras and lights for observation
- Manipulators for environmental intervention
- Cables (commonly known as tethers) for communication to and from the operator on the surface, and power from the surface
- Motors fitted with propellers for propulsion and positioning (known as thrusters)
- Sensors for depth and orientation, and environmental monitoring

1.4.5 ROV Classification

The commercial sub-sea industry applies some broad categorisation to ROV's. These can be summarised as:

- Micro Observational Class
- Mini Observational Class
- Light & Medium Work Class
- Heavy Work Class
- Seabed Working Class

1.4.6 Micro Observational Class

These are typically light-weight construction, optimised for portability, designed exclusively for observation in shallow waters (less than 100 m). Typical uses include ship, pier and pipe inspections. Figure 1.3 shows an example from VideoRay LLC.

Figure 1.3. The P4 CD 300 is a micro observational class ROV developed by VideoRay LLC [4].

1.4.7 Mini Observational Class

These vehicles perform a similar role to the micro observational class vehicles but are designed with heavier duty construction techniques that make them more suited to the greater depths they operate at (typically down to 1,000 m).

1.4.8 Light & Medium Work Class

A step up in size and durability of construction, these light-to-medium-weight ROV's are typically fitted with a single small manipulator giving them a rudimentary ability to handle objects. Compared to the Heavy Work Class, they have relatively low thruster power and therefore a lower payload lift capacity. An example is shown in Figure 1.4.

Figure 1.4. The Saab Seaeye Lynx is light-medium work class ROV [5].

1.4.9 Heavy Work Class

Designed for extreme depths (up to 6,000 m is not unusual), and situations where size and weight are not considered primary considerations, these heavy duty ROV's perform the most challenging underwater tasks. They are usually fitted with at least two manipulators, specialised tooling, multiple cameras and lights, and as we have seen, pitch and roll self-stabilisation systems. These vehicles can weigh in excess of 1.5 tonnes. Fitted with the most powerful thrusters available, they can carry and lift the largest payloads, sometimes in excess of 300 kg.

Figure 1.5 shows an example from Saab Seaeye.

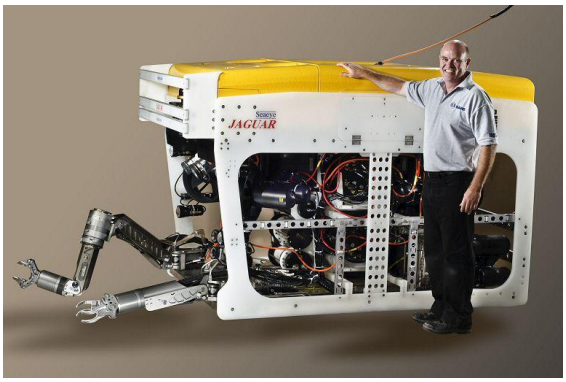


Figure 1.5. The Saab Seaeye Jaguar is a heavy work class ROV [6].

1.4.10 Seabed Working Class

These highly specialised vehicles are designed to lay undersea pipes and cables. Some are capable of cutting or blasting a channel in the seabed, laying the cable, and then burying it, all in a single pass. An example from the Newcastle-based company SMD Ltd is shown in Figure 1.6.

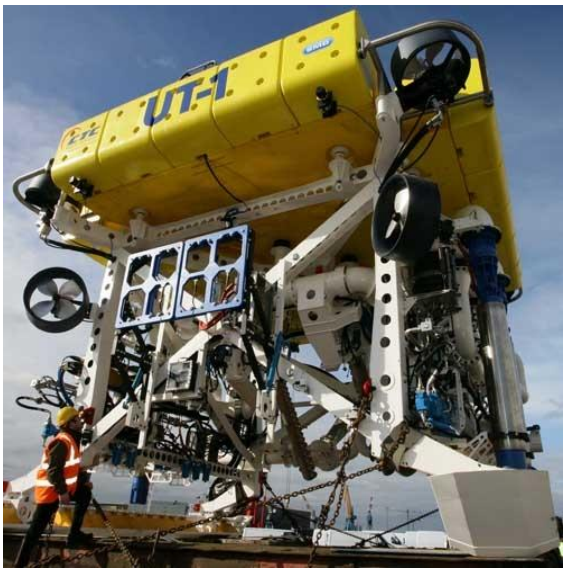


Figure 1.6. The SMD UT-1 jet trencher [7].

1.4.11 Commercial ROV Use

As can be seen by the large variety of ROV's available, there are an accordingly large number of applications they are used for. Here is a selection:

- Port and pier inspection
- Trenching and ploughing
- Cable laying & maintenance
- Pipe laying & maintenance
- Nuclear plant inspection
- Water tank inspection
- Environmental monitoring
- Sea floor surveying
- Wreck discovery
- Archaeology
- Acoustic positioning
- Harbour and coastal defence
- Mine counter-measures
- Sub-sea construction
- Well-head maintenance
- Mining
- Search-and-rescue
- Biological research

Major events where ROV's have been used extensively include:

- Japanese tsunami search-and-recue
- BP Macondo well-head repair
- The search for the Titanic
- The investigation into the sinking of the Costa Concordia cruise ship

2. System Requirements and Architecture

From an analysis of commercial ROV's that can be found operating in the field, and with consideration to the ROV's applications and operating environments, the design of this ROV system has closely followed these primary design guidelines:

- ease-of-use
- portability
- cost effective construction methods and materials

Working from these guidelines, the following requirements were specified:

- operator controller with joysticks, switches, status display screen and LED's, and video screen
- 4 thrusters for horizontal positioning
- 3 thrusters for vertical positioning and the self-stabilisation system
- on-board power
- 30 m tether for data and video transfer between operator controller and ROV
- lightweight frame construction

The system can be most easily visualised by examining the major system blocks as shown in Figure 2.1.

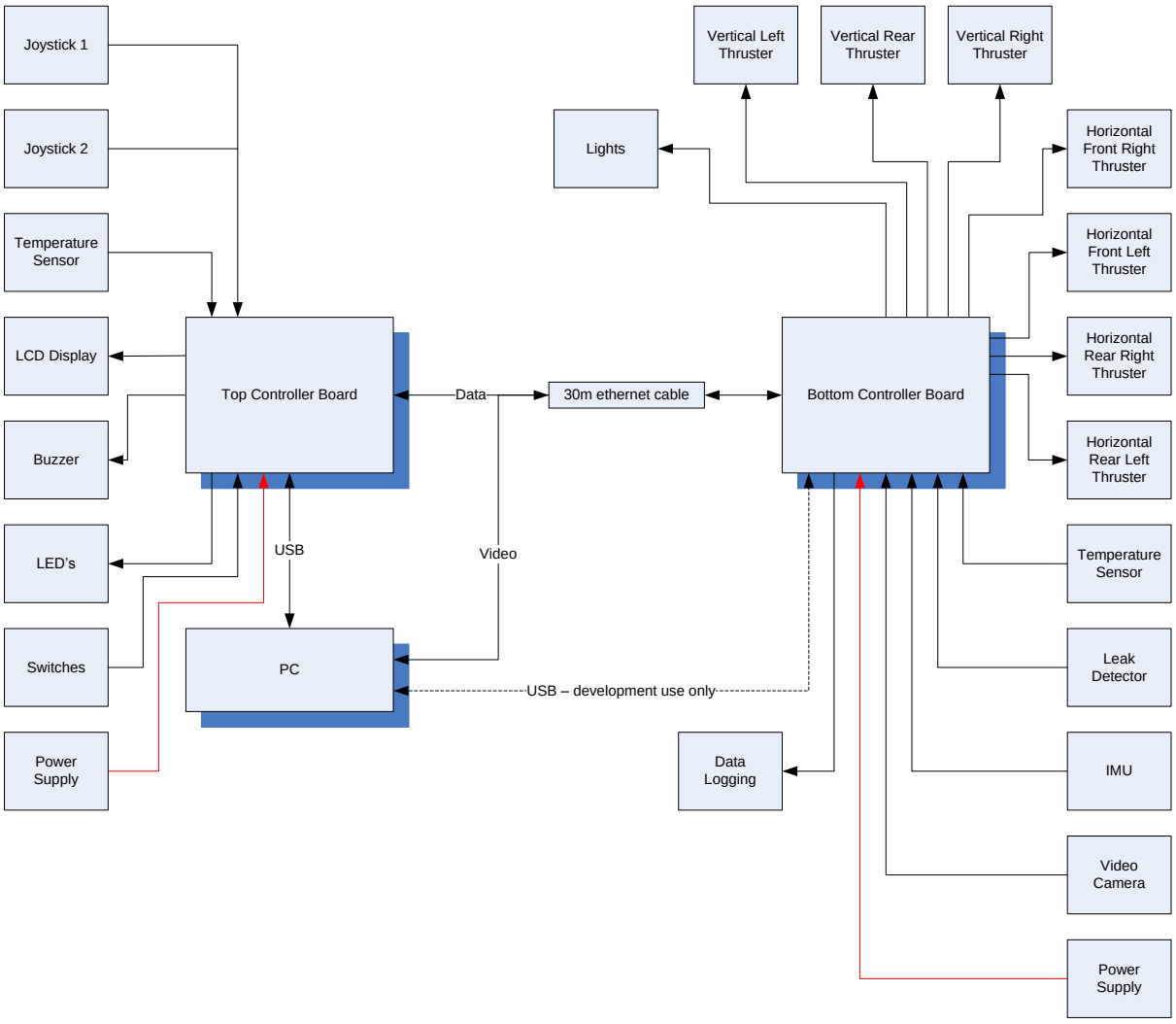


Figure 2.1. ROV system block diagram.

3. Mechanical Design and Construction

This Chapter details the many facets of designing and physically constructing an ROV.

3.1 Frame Design

The purpose of the frame is to support the water-proof enclosure, the thruster motors, and any trimming weights. Ideas for the frame design were initially considered and assessed using pencil and paper, and simple wire models, as shown in Figure 3.1. The principle design goal for the frame, taking into account thruster and enclosure positioning and support, was to ensure there was maximum water flow through the open frame, to therefore minimise drag.

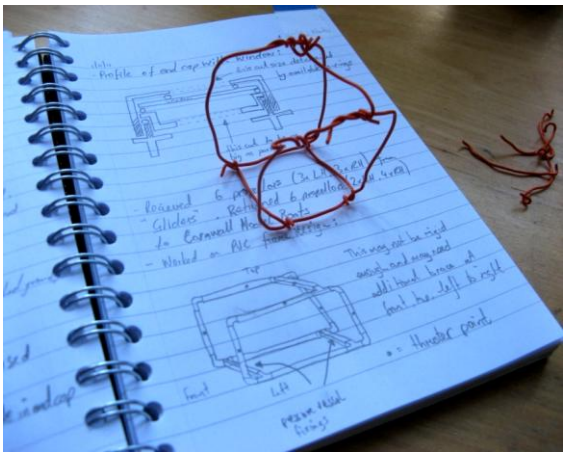


Figure 3.1. Designing the frame.

The chosen design consists of 3 sections placed horizontally perpendicular to the forward direction of travel, two along the bottom to support the enclosure, and a single upper section towards the rear. They are used for cross bracing and vertical thruster support, and join the two rectangular side sub-frames that support the horizontal thrusters.

The hollow frame is designed to fill with water during operation to assist with buoyancy and ballast trimming, and has a number of holes drilled for this purpose. In addition, there is a mesh attached to the bottom of the frame which serves as a surface on which to mount the trimming weights.

3.2 Frame Construction

The frame is constructed of plastic tubing, and the associated connecting elements, with a nominal 32 mm outer diameter. This material is lightweight yet rigid and strong, readily available, easy to work, and cost effective.

Images of the construction of the frame are shown in Figure 3.2.

Figure 3.2. Construction of the frame.

All the wiring to and from the motors, and the tether to the surface controller, have been routed through the inside of the plastic tubing to minimise drag, to avoid damage to wiring, and to avoid fouling of the propellers.

Figure 3.3. Routing and sealing the wiring.

After routing all the wires, they have been concentrated inside a box and connected to an 18-core cable using screw-terminals. This box was then flooded with a polyurethane resin to seal the connections from the water (Figure 3.3).

3.3 Water-Proof Enclosure Design and Construction

The water-proof enclosure is where all the underwater electronics and the video camera are located. Due to the high cost of these components, (approximately £300), keeping water out of this area is absolutely essential, so given the limited budget available, considerable research was done into finding suitable low-cost off-the-shelf components that would meet the requirements of:

- water-proof
- accessible
- readily available
- usable
- modifiable
- uncompressible

PVC drainage pipe fittings were found to fit these requirements as:

- they are designed with seals for blocking water
- there are a range of fittings that will allow access, via screw-thread and push-fit end-caps
- the components are readily available through local suppliers
- handy mounting hardware is available
- they are easily machined for modifications
- they are purpose-designed for underground installation so can handle external pressure, at least to some degree

It should be noted that the components are actually designed to keep water in, rather than water out, so it was initially unclear whether this system will prove to be completely water-proof.

In order to find out, a T-section with two screw end-caps, and a single push-fit end-cap (as can be seen in Figure 3.2) were trialled in a local swimming pool down to a maximum depth of 5 metres for 30 minutes. This testing met with success as no water was detected inside the enclosure, so it was decided to proceed with using these components.

3.3.2 Adding a Window to the Water-Proof Enclosure

Following the successful first test, the next step was to add a window for the video camera. A number of designs were considered for the window construction. Cross sections of the various designs are shown in Figure 3.4 with the red circles indicating the location of sealing o-rings.

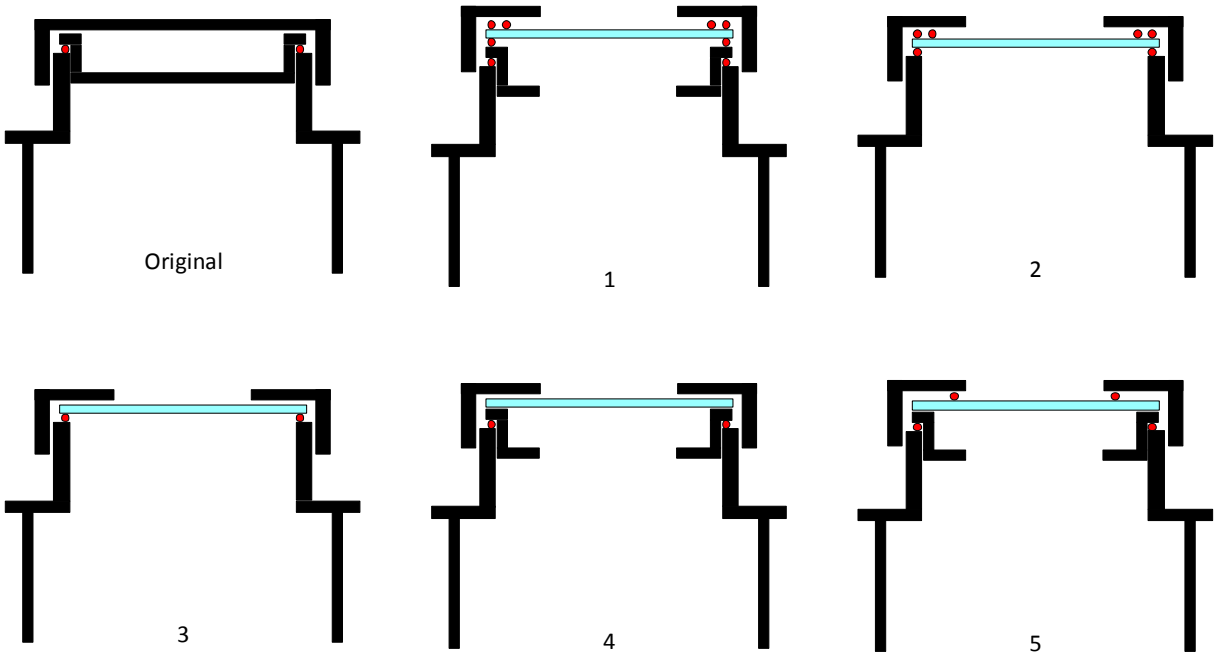


Figure 3.4. Cross-sections of designs for the camera window.

After some experimentation with the different options, it was found that using more than a single o-ring did not allow the cap to be screwed down enough to ensure a tight seal. It was also observed that if the o-ring was not held in place, the act of compression would cause the o-ring to move and thus give an unreliable seal. For those reasons option 4 was chosen. The window was then machined out and installed as can be seen in Figure 3.5.

Further water-proofing trials were then undertaken and they also proved to be a success.

Figure 3.5. The camera window (with the protective masking still in place).

3.3.3 Further Modification to the Water-Proof Enclosure

At this point confidence was high, but the biggest problem was yet to be faced. How to get all the motor and signal wires through the hull of the water-proof enclosure, and still keep the enclosure dry? With 2 wires for each of the 7 motors and 1 light, and 8 wires inside the Ethernet cable, this meant there are 24 individual wires. These 24 wires mean, in theory, there could be up to 24 individual holes in the hull, and of course 24 holes = 24 possible leak sources.

After much consideration, the obvious solution, though certainly not the cheapest solution, but the one that was finally opted for, was to concentrate all the wires into one single multi-core cable with a single multi-pole connector. This ensured there would be only a single hull penetration, and therefore a much reduced chance of leakage occurring.

By judiciously connecting all the common 12 V wires in the concentrating box, and reducing the number of usable Ethernet wires from 8 down to 6, the number of required connections was reduced to 18. An 18-core cable was sourced, and a suitable IP68-rated plug and socket acquired. Figure 3.6 shows the construction of this cable and connectors pairing.

Figure 3.6. Construction of the 18-way cable and connectors.

Initial water-proof testing of the enclosure, now fitted with the multi-pole connecter plug followed and was thought to be successful. Subsequent operation showed this was not the case, and small amounts of water (approximately 4 tablespoons per hour were observed) were entering the enclosure through the hole that was created for the connector.

Upon inspection, it became obvious that the hole had not been made sufficiently accurately and did not meet the connector manufacturers' specifications. An additional end-cap was purchased and another hole was machined, this time much more accurately using more appropriate and accurate tools.

3.4 Propulsion by Thrusters

Propulsion of the vehicle through the water, both vertically and horizontally, under control of the operator, is the job of the thrusters.

3.4.1 Thruster Construction

The thrusters used for this project are sealed DC motors, derived from bilge pumps, and fitted with propellers. Figure 3.7 shows the pumps before, during and after modification.
To turn the bilge pumps into ROV thrusters, the following steps were undertaken:

- Unclip the blue protective shroud
- Machine away the white outer shell to expose the sealed body and impeller.
- Remove the impeller
- Attach a propeller-to-shaft adapter to the motor shaft
- Attach a propeller to the adapter
- Repeat six more times

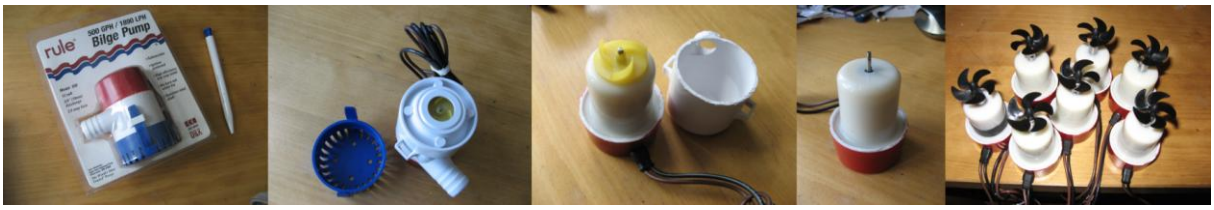


Figure 3.7. Conversion from bilge pump to thruster.

3.4.2 Horizontal Thruster Configuration

An analysis of the commercial ROV market shows two common configurations are used for horizontal thruster placement:

- the "H" layout as shown in Figure 3.8
- the "vectored" layout as shown in Figure 3.9

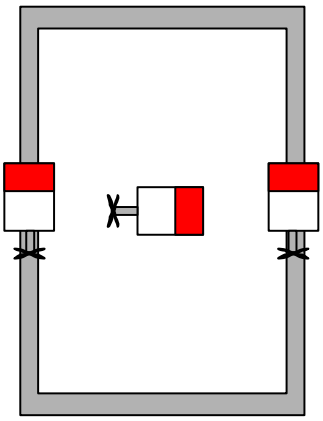


Figure 3.8. "H" horizontal thruster layout.

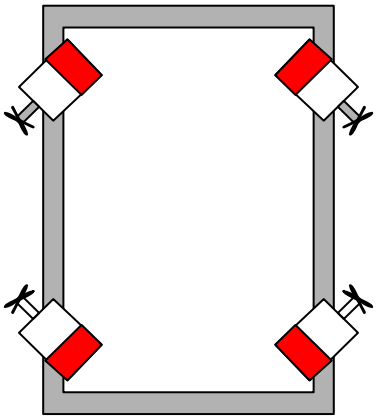


Figure 3.9. "Vectored" horizontal thruster layout.

Both configurations are used commercially: the P4 CD 300 from VideoRay LLC, (Figure 1.3) uses the H layout, whereas the Saab Seaeye Lynx and Jaguar, (Figures 1.4 and 1.5), use the vectored layout, as does the SMD UT-1 (Figure 1.6).

The H layout requires only three thrusters instead of four which gives a small cost advantage, but there are also a number of disadvantages:

- All three thrusters require bi-directional control (i.e. they must be able to be driven forward or backward), which incurs significant additional cost and complexity in the control electronics
- 6 wires have to be passed through the water-proof pressure hull
- Sideways motion is relatively slow as there is only a single thruster
- The single sideways-acting thruster should be mounted exactly at the centroid of the ROV otherwise it will impart a rotational force on the ROV

The vectored layout on the other hand requires only uni-directional thruster control, giving simpler, cheaper electronics, and needs only 5 connections through the hull (as one wire is common to all thrusters and can be connected together externally). The major disadvantage of this layout is that the maximum thrust for forward, reverse and sideways operations is reduced by 30% due to the 45° offset of each thruster from the direction of travel.

Clearly, both layouts offer unique advantages and disadvantages, but after careful consideration of these, the vectored layout was chosen. The thrust reduction due to the angled thruster mounting was deemed to be an acceptable trade-off for simpler electronics.

Figure 3.10 shows how, using the vectored layout, a combination of any two thrusters gives 6 possible directions of movement.

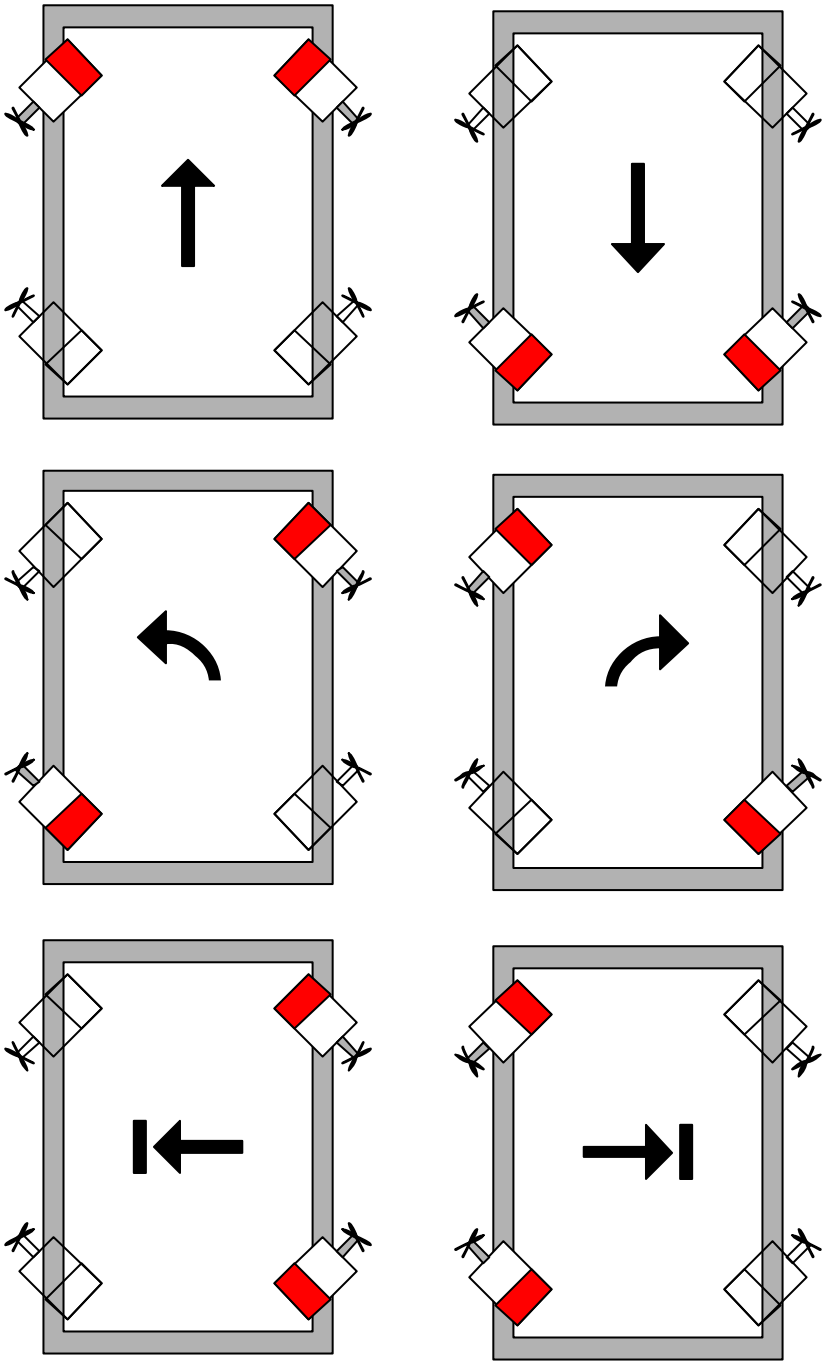


Figure 3.10. Directional control using a "vectored" horizontal thruster layout.

3.4.3 Vertical Thruster Configuration

For simple vertical positioning control, a single bi-directional thruster, centrally located, and mounted vertically is sufficient. However, to support the self-stabilisation system (see Chapter 6), additional bi-directional thrusters are necessary to give the ability to correct disturbances along the pitch and roll axes.

For the roll axis correction, a thruster is mounted along the top of the left and right side frames at the midpoint. These thrusters are set at an angle offset from the vertical to assist in roll correction. For normal vertical positioning they are driven equally, but for roll correction they are driven at different rates, and different directions if necessary.

For the pitch axis, a single thruster is mounted at the top rear of the frame. The offset to the vertical is one of the factors considered during the testing phase and discussed in Chapter 7. It is not used for normal vertical

positioning of the ROV as its non-symmetric positioning on the frame may induce pitch.

3.4.4 Propellers

A number of different propellers (Figure 3.11) have been tested to ascertain the relative thrust measurements and the current draw by the thruster motors, to find the most suitable propeller:

- Unmodified bilge pump with impeller
- Robbe R1473 3-blade 35 mm propeller
- Robbe R1471 3-blade 50 mm propeller
- Robbe R1465 3-blade 60 mm propeller
- Graupner G2308.50 3-blade 50 mm propeller
- Graupner G2298.40 5-blade 40 mm propeller

Figure 3.11. Five different propellers were tested.

The testing apparatus is shown in Figure 3.12 and consists of a rotating cross-shaped structure with the thruster attached to one end. When power is applied to the thruster, the structure pivots and applies a downward force to the scales. That force, measured as weight by the scales, is directly proportional to the thruster force output. Due to the nature of the testing apparatus, the measurements taken are not considered highly accurate but serve to provide relative indications of the differences between the propellers. The results, averaged over a number of tests, are shown in Table 3.1.

Figure 3.12. Thrust and current draw measurements.

Propeller	Current Draw (A)	Thrust (g)	g / A
Bilge pump impeller	1.60	75	46.88
3-blade Robbe 35 mm	2.05	125	60.98
3-blade Robbe 50 mm	2.80	190	67.86
3-blade Robbe 60 mm	3.50	190	54.29
3-blade Graupner 50 mm	2.20	250	113.64
5-blade Graupner 40 mm	2.10	170	80.95

Table 3.1. Thrust and current draw measurement results.

On a gram per Amp basis, the Graupner 50 mm propeller seems the logical choice, however it was noted during testing that the 5-bladed 40 mm Graupner propeller generated significantly less turbulence. It also had a much higher reverse thrust compared to any of the 3-blade propellers. Minimising turbulence is important for good visibility during operation of the ROV and though reverse thrust is not important for the horizontal thrusters, it is critical for the vertical thrusters. For these reasons, 5-blade 40 mm Graupner propellers were chosen for all seven thrusters.

3.4.5 Propeller Rotation

As a single propeller rotates, it imparts a rotational force to the vehicle, and this has the effect of steering the vehicle to the right or left. To minimise this force, two thrusters are used simultaneously, with each thruster having a propeller that rotates in the opposite direction to the other. Table 3.2 shows the allocation of rotation direction to the individual thrusters.

Thruster	Rotation Direction
Horizontal Front Right	Counter-Clockwise
Horizontal Front Left	Clockwise
Horizontal Rear Right	Clockwise
Horizontal Rear Left	Counter-Clockwise
Vertical Right	Counter-Clockwise
Vertical Left	Clockwise
Vertical Rear	Counter-Clockwise

Table 3.2. Thruster propeller rotation direction.

To ensure an equivalent level of thrust from each of the two thrusters, propellers designed for opposite rotation are used. Figures 3.13 and 3.14 show the two different 5-bladed propellers used, each designed for rotation in a specific direction.

Figure 3.13. Counter-clockwise rotating propeller.

Figure 3.14. Clockwise rotating propeller.

3.5 Tether

The tether is the physical and communications link between the top controller, located out of the water, and the ROV in the water. It has 3 responsibilities:

- deliver instructions down to the ROV from the top controller
- deliver sensor data from the ROV up to the top controller
- deliver video up to the operator

The tether in this case consists of a single 30 m Ethernet cable (Cat. 6), with two of the four sets of twisted pairs being used. This length will give a maximum depth of 30 metres straight down, and 21 metres out at a 45° angle. 30 metres is the typical maximum depth of recreational scuba divers, and therefore the maximum depth for easily testing the ROV, allowing for an emergency recovery.

Two wires (one set of the four twisted pairs, wires 1 and 2) are used for serial communications between the top controller and bottom controller. A third wire (wire 3) is used to carry the analogue video signal, and a fourth wire (wire 4) is the common ground connection between the top and bottom controllers. A third pair (wires 5 and 6) are unused, but available for future use. The fourth pair (wires 7 and 8) are not connected at all due to the lack of spare conductors in the 18-core cable running into the water-proof enclosure.

The primary physical requirements for the tether are that it must be as light and small as

possible to minimise drag through the water, and ideally should be neutrally buoyant. The ethernet cable is not neutrally buoyant so a number of floatation devices are attached to the tether at regular intervals to ensure neutral buoyancy. To avoid fouling of the tether with the thrusters, the tether is made positively buoyant for the metre closest to the ROV.

For physical protection, at the ROV end a cable strain relief grommet is used, and the first 5 metres of the tether are enclosed in a black braided sheath.

3.5.1 Wireless Communications

It has been suggested that a completely wireless system may be possible and this has been investigated but despite being a very interesting subject has been found unsuitable for this specific project. Briefly, there are a number of advantages to a wireless system including:

- less drag
- fewer buoyancy issues
- no tangles, snags or propeller fouling

There are however, two major technical problems with underwater wireless communications:

- limited bandwidth
- latency

The bandwidth available with long range acoustic modems is approximately 10 kbps [8]. This would be sufficient for control and sensor data but is not sufficient for video transmission and therefore makes it unsuitable for this project. There are considerable efforts currently underway to apply modern communications techniques such as OFDM and MIMO to underwater communications [9], so this may be less of an issue going forward.

The second problem is that sound propagates through water at approximately 1,500 metres per second [10] which is vastly slower than an electrical signal in a cable that travels at close to the speed of light. Therefore a considerable level of latency occurs and this obviously increases as the distance increases. This may not be a significant factor for a distance of 30 metres, but for commercial ROV's operating in real-time at 6,000 metres it is an unworkable solution as round trip times would be in the order of 8 seconds.

The examination of the propagation of sound through water, sea water in particular, is not a trivial exercise and well outside the scope of this project. The reader is encouraged to look at the US Navy publication "Physics of Sound in the Sea" resulting from research done during World War II for further details on the physics of communications through sea water.

4. Top Controller

The primary purpose of the top controller is to act as the operator control interface. From the operator's point-of-view, they need to be able to control the position of the ROV in the water on any one of four axes, as easily as possible, and receive timely feedback from the ROV of its position and the nature of the environment it is in.

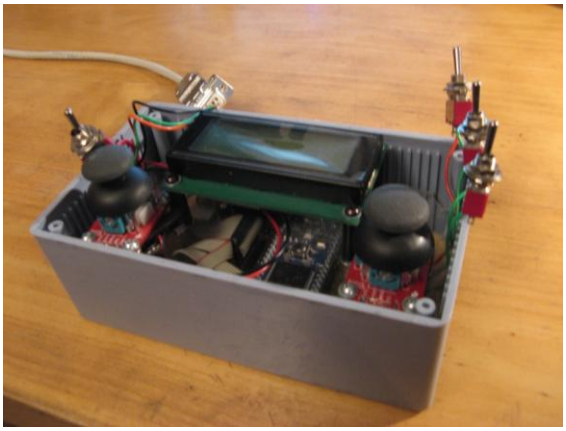


Figure 4.1. The completed top controller.

The following sections of this Chapter detail the design, specification and construction of the operator control interface.

4.1 Top Controller Design

From the requirements it was determined that the 4-degree-of-freedom (DOF) positioning and control performed by the operator (forward/backwards, up/down, move left/move right, and rotate left/rotate right) would be best achieved using two joysticks and a number of switches. Feedback to the operator would come via a video screen, an LCD screen for sensor data and warnings, and a number of LED's to indicate a particular state or warning.

The electronic design of the top controller centres around a microcontroller (in this case an NXP LPC1768 on an mbed microcontroller development board) interfaced to:

- three switches
- an LCD screen
- two joysticks
- six LED's
- a temperature sensor
- a buzzer
- the serial connection to the bottom controller

The circuit diagram and strip-board layout can be found in Appendix B, and the final operational microcontroller C programming code in Appendix D.

4.2 Construction and Testing

For a system as complex as the top controller, where almost every available I/O pin is being used on the mbed microcontroller, connecting every element simultaneously and then attempting to test and debugging from there, from experience, seemed likely to be inefficient and ineffective.

Instead, an alternative and ultimately successful approach was used, where each element of the top controller was first tested in isolation from the other elements of the system to ensure they performed as required (see Figure 4.2). Only once they were proven to work in isolation were they integrated into the evolving system built on a breadboard. As each new element was added, it along with all the existing elements already in place, were then fully tested to ensure they worked with each other.

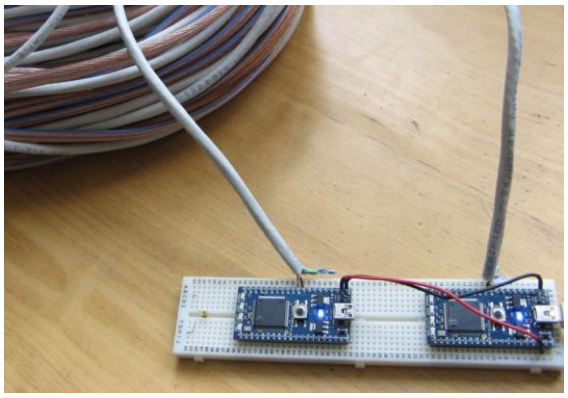


Figure 4.2. Two mbed modules mounted on a breadboard during serial communications testing.

This gradual addition, element by element, to the working system, ensured that any problems that did occur were easily identified. Figure 4.3 shows the development in progress.

The order of implementation was:

1. mbed microcontroller development board
2. voltage regulator
3. LCD display
4. joysticks
5. temperature sensor
6. leak buzzer
7. switches
8. LED's
9. RS232

Only once the top controller was working entirely as required on the breadboard, was it moved onto the strip-board, as shown in Figure 4.3.

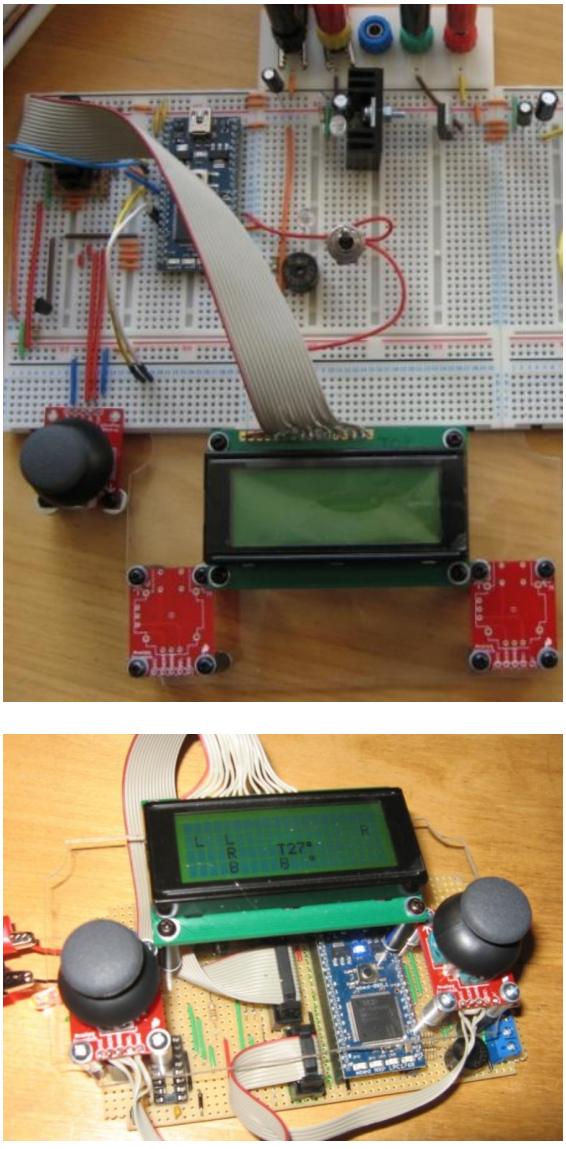


Figure 4.3. The top controller during development, on breadboard (top) and later on strip-board (bottom).

Two primary tools were used during the testing and construction phase. Firstly, a digital multi-meter was used for checking connections and voltage levels to ensure every component was correctly placed and wired. The second tool was the LCD display which proved to be very useful. It was used extensively during the development of the top controller (and also the bottom controller) due to the flexible nature of the data it can display. This is primarily due to the ease-of-use offered by the mbed rapid prototyping development system.

4.3 mbed Rapid Prototyping Development System

The mbed development system is the combination of an NXP LPC1768 microcontroller mounted on a user-friendly 0.1"-spaced DIP package, an online web-based development environment, and a collection of official and community-developed standard libraries. This combination is specifically designed to facilitate quick and easy prototyping as can be seen in Figure 4.2.

The NXP microcontroller has an ARM Cortex-M3-based core running at 96 MHz It contains 512 kB of flash memory, 32 kB RAM along with I^2C, SPI, CAN and serial I/O, 6 PWM output channels, 6 ADC input channels, and a single DAC output channel. It also offers USB and Ethernet connectivity and a 3.3V regulated power output [11].

4.4 Power Supply

Operation of the top controller requires a number of different voltage levels. The mbed microcontroller can accept an input from 4.5 V to 14.0 V, the LCD display operates logically at 3.3 V, though its backlight requires 4.2 V, and the RS232 IC requires 3.3 V for operation.

Power was supplied during development by a standard PP3 9 V battery. A 5 V linear voltage regulator, TS7805CZ, is used to cut the 9 V down to 5 V for both the LCD backlight and the mbed. The 4.2 V for the backlight is generated from the 5 V using a voltage divider circuit.

The 3.3 V required by the LCD display and RS232 IC is generated by the mbed's onboard voltage regulator.

4.5 LCD Display

The LCD display chosen is a 20 x 4 character black-on-green display fitted with an LED backlight. The display uses a 4-bit parallel data connection to the microcontroller (there is also the option to use an 8-bit connection) and requires 2 additional control lines: 'RS' and 'E' [12]. It is powered from the 3.3 V generated by the mbed's onboard regulator. The backlight is powered separately by a 4.2 V source.

During ROV operation, the display is used to indicate some or all of:

- the state of the joysticks
- the temperatures from the sensors in the top and bottom controllers
- the pitch and roll tilt angles
- the PID control settings
- the status of the data logging function, the lights and the control system
- leak warnings
- H-bridge over-temperature warning

4.6 Analog Joysticks

The top controller uses two analog joysticks to allow the operator to control the position of the ROV in the water along any of 4 different axes. Figure 4.4 shows how the eight possible position instructions are mapped to the eight joystick positions.

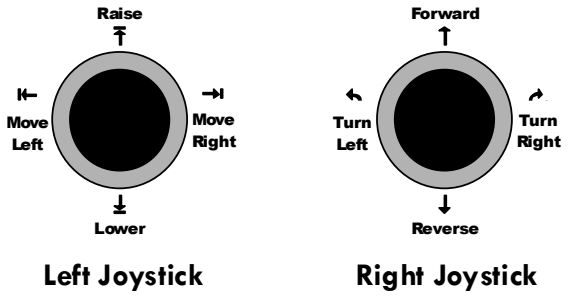


Figure 4.4. Joystick control direction assignment.

Each joystick is essentially a structure holding two linear potentiometers aligned along two perpendicular axes. The output from each axis of each joystick is connected to an ADC input on the mbed microcontroller where the value of the resistance in the potentiometers, and therefore the position of the joystick, is directly proportional to the voltage appearing at the ADC input, referenced to 3.3 V.

Though the joysticks are analog devices, they are effectively used as digital inputs with one of six possible states:

- up
- down
- left
- right
- horizontally centred
- vertically centred

The software determines the current state of each joystick based on the value returned from the ADC input, and maps that to the appropriate instruction for the ROV.

The sampling of the joysticks occurs 20 times per second. Subjective testing indicated an update rate any slower induced a perceptible lag between the operator input and the ROV response.

4.7 Temperature Sensor

The top controller is fitted with a temperature sensor to measure the temperature inside the enclosure. This information is sampled every few seconds and displayed on the LCD display.

The sensor is an LM335 from National Semiconductor, supplied in an easy-to-use plastic TO-92 package. It was chosen as it offers a good balance between price and accuracy. It has an operating range from -40° to +100° C and is typically accurate to ±2° C which was considered acceptable for this application, however accuracy can be increased by calibration if necessary.

The sensor works like a Zener diode with a breakdown voltage output directly proportional to the temperature at 10 mV / K [13]. Due to the linearity of its output, by connecting to an ADC input on the microcontroller, the ambient temperature in degrees Celsius can be

calculated using the following formula, where the sampled value is in the range of [0, 1]:

$$temperature\ (\circ C) = sampled\ value \times reference\ voltage \times 100 - 273 \quad (1)$$

4.8 Water Leak Detection Warnings

If the bottom controller detects a water leak into the water-proof enclosure, it will communicate this to the top controller. The top controller will respond to this by sounding a small buzzer, displaying a warning on the LCD screen, and illuminating a red LED.

4.9 Switches and LED's

The top controller has 3 switches and 6 status LED's. The three switches are used for:

- lights on / off
- control system on / off
- data logging on / off

The LED's are used for:

- power (green)
- leak detected (red)
- data link to bottom controller (green)
- state of light switch (green)
- state of control system switch (green)
- state of data logging switch (green)

4.10 Data Communications

The top controller has two separate data communications channels.

4.10.1 Communication with the Bottom Controller

Communications between the top controller and bottom controller, via the tether, are by RS232 serial. An EXAR SP3232ECP-L driver chip is used in both controllers for this as the inbuilt serial UART's in the mbed microcontroller are not designed for transmitting over 30 m cable lengths.

A relatively simple communications protocol was established for the exchange of control and sensor data between the top and bottom controllers.

Tables 4.1, 4.2 and 4.3 detail the 14 bytes sent from the top controller to the bottom controller:

Byte	Name	Purpose
1	controlData[0]	8 bits for control data
2	controlData[1]	8 bits for control data
3	kP8[0]	first byte of 4 for kP float
4	kP8[1]	second " "
5	kP8[2]	third " "
6	kP8[3]	fourth " "
7	kI8[0]	first byte of 4 for kI float
8	kI8[1]	second " "
9	kI8[2]	third " "
10	kI8[3]	fourth " "
11	kD8[0]	first byte of 4 for kD float
12	kD8[1]	second " "
13	kD8[2]	third " "
14	kD8[3]	fourth " "

Table 4.1. Data sent from the top controller to the bottom controller.

Bit Number	Purpose
7	'1' = first control data byte
6	'1' = data logging on
5	'1' = control system on
4	'1' = lights on
3	'1' = rear left thruster on
2	'1' = rear right thruster on
1	'1' = front left thruster on
0	'1' = front right thruster on

Table 4.2. controlData[0] byte.

Bit Number	Purpose
7	'0' = second control data byte
6	unused
5	unused
4	unused
3	unused
2	unused
1	'1' = drive down
0	'1' = drive up

Table 4.3. controlData[1] byte.

Transmission of floating point values for the P, I, and D constants for the control system is achieved by breaking the 32-bit floating point value into four 8-bit values and sending those, as shown in Table 4.1. Each set of four bytes is then reconstructed back into a 32-bit floating point value at the receiving end by the bottom controller.

There are also 4 bytes sent back from the bottom controller to the top controller:

Byte	Name	Purpose
1	txData[0]	8 bits for warning flags
2	txData[1]	bottom temperature
3	txData[2]	pitch tilt angle
4	txData[3]	roll tilt angle

Table 4.4. Data sent from the bottom controller to the top controller.

Bit Number	Purpose
7	'1' = first data byte
6	unused
5	'1' = thermal overload detected
4	'1' = thermal overload detected
3	'1' = thermal overload detected
2	'1' = leak detected
1	'1' = leak detected
0	'1' = leak detected

Table 4.5. txData[0] byte.

4.10.2 Communications with the Development PC

The mbed development board includes a USB connection to the development PC. This connection was used, in addition to downloading program code, to give a serial terminal interface to the top controller and to the bottom controller, via the top controller. This serial terminal interface was used for debugging purposes, and for PID control tuning on-the-fly.

Figure 4.5 shows a terminal session in operation during development, between the development PC and the top controller.

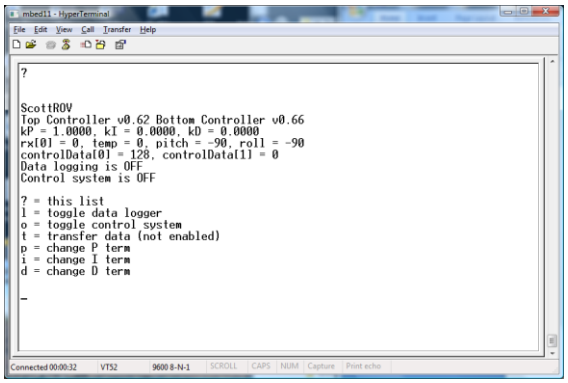


Figure 4.5. Serial terminal interface, via USB, from the development PC to the top controller.

4.11 Enclosure Design

The front panel design, to fit the chosen enclosure, is shown in Figure 4.6.

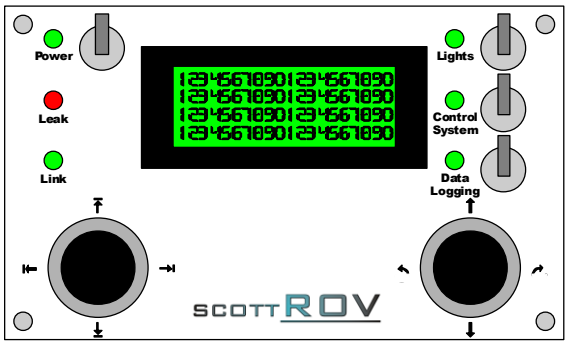


Figure 4.6. Front panel design.

The rear of the enclosure is where the tether input and video output connections are made. This is shown in Figure 4.7.

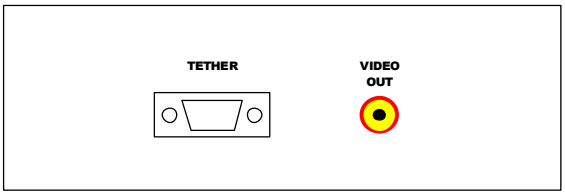


Figure 4.7. Rear panel design.

4.12 Software

The design of the top controller software running on the ARM microcontroller follows an "interrupt-driven" model. Periodic tasks were set up to run at regular intervals using a timer function that generates interrupts. These tasks include:

- read the temperature sensor once every 3 seconds
- read the joystick positions 20 times per second i.e. every 0.05 seconds
- update the LCD display 20 times per second
- send and receive data to and from the bottom controller 20 times per second

In addition, asynchronous changes by the operator of any of the three switches on the control panel are also handled by the interrupt mechanism.

Figure 4.8 shows a representation of the system.

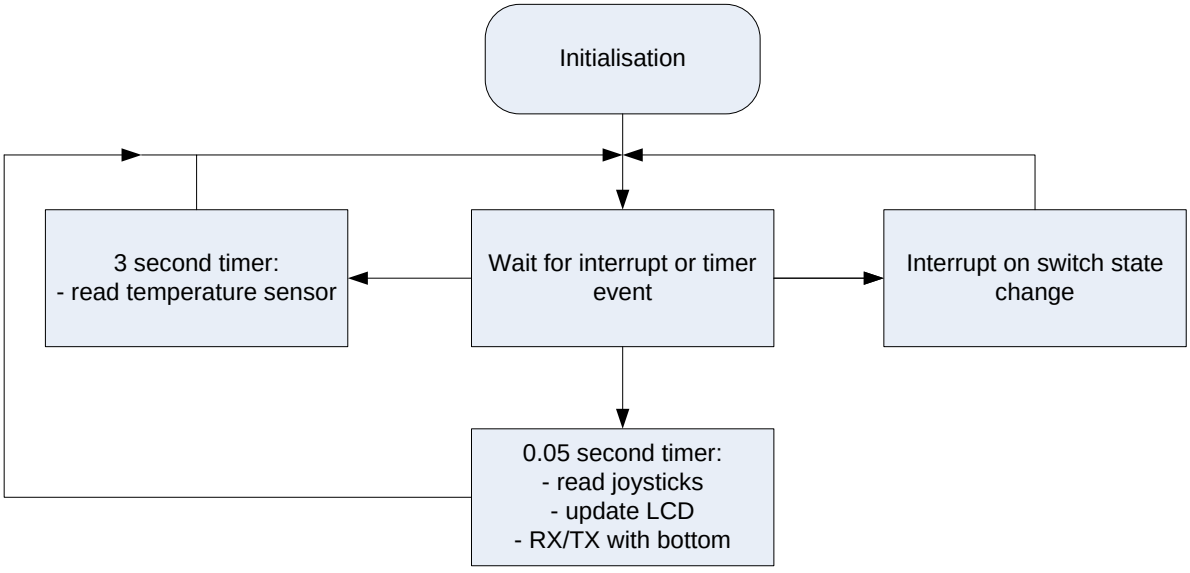


Figure 4.8. Top controller system block diagram.

5. Bottom Controller

The primary purpose of the bottom controller is operation of the thrusters and lights, as instructed by the operator using the top controller. There are 7 thrusters: 4 requiring uni-directional control, and 3 requiring bi-directional control. There is also an integrated inertial measurement unit (IMU) for the self-stabilising control system.

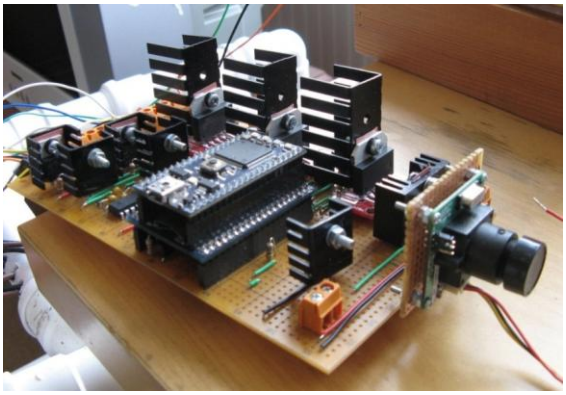


Figure 5.1. The completed bottom controller.

The following sections of this Chapter detail the design, specification and construction of the bottom controller.

5.1 Bottom Controller Design

Given the requirements of supporting four uni-directional, and three bi-directional thrusters, and one light switch, it was determined that five MOSFET's, acting as switches, and three H-bridge motor drive IC's would be required. In addition a range of sensors would be required:

- a 2-axis tilt sensor for the control system
- a leak detector
- a video camera to allow the operator to see where the ROV was heading
- a temperature sensor

The electronic design of the bottom controller centres around a microcontroller (once again an NXP LPC1768 on an mbed microcontroller development board) interfaced to:

- five MOSFET's
- three H-bridge IC's
- an IMU daughterboard
- a temperature sensor
- a leak detector
- the serial connection to the top controller

The circuit diagram and strip-board layout can be found in Appendix C, and the microcontroller C programming code in Appendix E.

5.2 Construction and Testing

As occurred for the top controller, the bottom controller system uses almost every available I/O pin on the mbed microcontroller, so testing by connecting every element simultaneously and debugging from there, once again seemed likely to be an inefficient and ineffective approach.

The alternative and ultimately successful approach was used again, where each element of the bottom controller was first tested in isolation from the other elements of the system to ensure they performed as required. Only once they were proven to work in isolation were they integrated into the evolving system built on a breadboard. As each new element was added, it along with all the existing elements already in place, were then fully tested to ensure they worked with each other.

This gradual addition, element by element, to the working system, ensured that any problems that did occur were easily identified. Figure 5.2 shows the development in progress. The order of implementation for the bottom controller was:

1. mbed microcontroller development board
2. IMU
3. Leak detector
4. temperature sensor
5. voltage regulator
6. MOSFET's
7. H-Bridges
8. RS232
9. Data logging
10. Lights
11. Camera

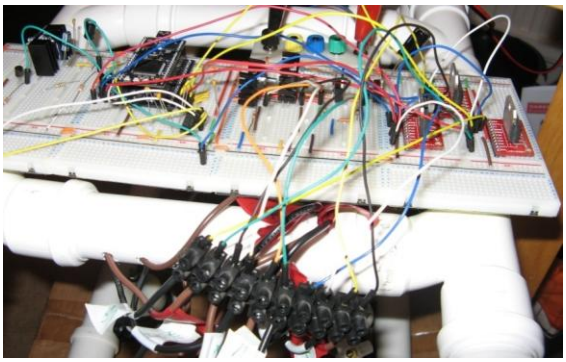


Figure 5.2. The bottom controller during development, on breadboard, wired to the seven thrusters.

Again, only once the entire bottom controller was working reliably on the breadboard, was it moved onto the strip-board (see figure 5.1).

5.3 mbed Rapid Prototyping Development System

The same mbed development system as used in the top controller is once again the central core of the bottom controller system. Three of the six available PWM output channels are used to drive, at variable speed and direction, the H-bridges for the vertical thrusters that require bidirectional control.

5.4 Inertial Measurement Unit

A 2-axis accelerometer is used to perform the tilt measurements for the roll and pitch axes used with the self-stabilising control system. The accelerometer used in this project, an LIS331DLH from ST Microelectronics, is part of a complete inertial measurement unit (IMU) that was designed by Tim Marvin specifically to work with the mbed development board. As can be seen in Figure 5.1, it has been cleverly designed so that the mbed board plugs in to the top.

Originally designed for a model aircraft autopilot system, it contains, in addition to the 2 accelerometer axes used, a third accelerometer axis, a 3-axis gyroscope, and a 3-axis magnetometer.

Each axis of the accelerometer outputs a floating point value in the range of -1 to +1 corresponding to -90° to +90° [14]. The conversion from floating point value to degrees is found by:

$$Tilt\ (degrees) = \frac{arcsin(Tilt\ (float)) \times 180}{\pi} \quad (2)$$

During development of the control system, certain spurious values were noticed to be coming from the accelerometer. An analysis and resolution of this issue can be found in Appendix I.

The IMU board also conveniently contains a micro SD slot connected through to the mbed. This was used for the data-logging and is described in section 5.13. The schematic and PCB layout of the IMU board as designed by Tim Marvin can also be found in Appendix G.

5.5 Leak Detector

To warn of any leaks that may have occurred inside the water-proof enclosure, a leak detector system has been developed. As can be seen in Figure 5.3, it consists of a small section of copper-coated strip-board with the connecting wires attached to adjoining tracks.

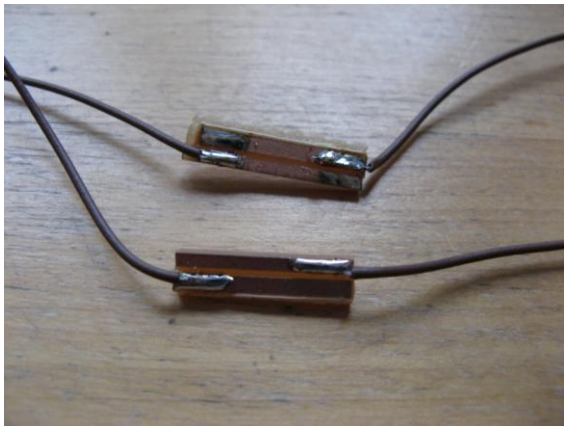


Figure 5.3. The leak detectors.

If any water is present and causes a bridge across the copper tracks, a positive voltage appears on pin 11 of the mbed. The software is configured to react to this pin going high and transmits a warning signal to the top controller, where an LED is lit, a buzzer sounds and a message appears on the LCD display.

5.6 Temperature Sensor

Like the top controller, the bottom controller is also fitted with an LM335 temperature sensor to measure the temperature inside the water-proof enclosure. This information is sampled every few seconds and communicated to the top controller for display on the LCD display.

5.7 Power Supply

Two methods of supplying power to the bottom controller were considered:

- Power from the surface, via the tether
- Power from batteries on board the ROV

5.7.1 Power from the Surface

Powering the ROV from the surface was initially investigated, but a major problem using the surface power option became evident quickly: the voltage drop along the wire caused by the inherent resistance of the wire. It can be overcome to some degree by using a larger gauge wire but this adds weight to the tether and therefore increases drag and buoyancy requirements. Furthermore, as the current drawn through the wire will vary depending upon which motors are being used, so does the voltage drop vary, giving an unpredictable voltage at the ROV end of the tether. A simple example is illustrated in Table 5.1:

Surface Voltage	Current Draw	Voltage Drop	Available Voltage
12 V	1 Amp	0.425 V	11.575 V
12 V	3 Amps	1.276 V	10.724 V
12 V	6 Amps	2.552 V	9.448 V

Table 5.1. Voltage drop example.

These calculations are based on a 30 m length of 2.5 mm^2 copper conductor.

As the H-bridges used (see section 5.9) have a low voltage cut-out somewhere in the range of 9 V to 11 V, just the possibility of the voltage dropping down into this range suggested this was not the optimal solution, so the alternative of an on-board power source was investigated.

5.7.2 On-Board Power

A number of factors were considered when looking at the different onboard power source options:

- cost
- recharge ability
- size and weight
- voltage
- power capacity and energy density

Given those factors, a number are different battery technologies were investigated:

- Lead acid
- Nickel Cadmium
- Nickel Metal Hydride
- Lithium Polymer

Lead acid batteries, as used in automotives, were deemed too heavy, too big, and prone to leakage. Nickel Cadmium and Nickel Metal Hydride batteries were also looked at closely but the decision was made to use the Lithium Polymer (LIPO) batteries as they offered:

- greatest power-to-weight ratio and energy density
- ready availability in large capacity configurations
- suitable cell voltage
- suitable form-factor
- lowest self-discharge rate when not used
- reasonable cost
- no memory effect

The specific LIPO battery used during development is the Turnigy 2.2 Ah 3S 30C battery (see Figure 5.4) that offers a nominal 11.1 V (3 cells @ 3.7 V, each in series). In practice it can be charged up to 12.6 V and is generally considered to have only 10% capacity remaining at 11.1 V.

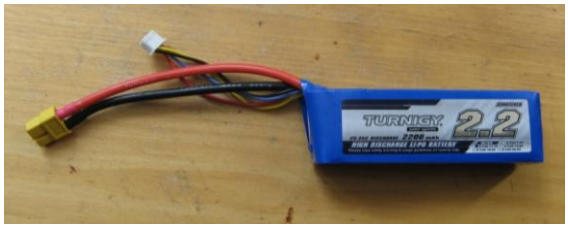


Figure 5.4. Turnigy 2.2 Ah 3S 30C LIPO battery.

The "30C" rating of the battery indicates the maximum continuous discharge current the battery can deliver without overheating and damaging the battery:

$$Maximum\ continuous\ discharge\ current = capacity\ (Amp\ hours) \times C\ rating \quad (3)$$

$$= 2.2 \times 30 = 66\ Amps$$

From this it can be seen that the 30C rating for this 2.2 Ah battery is more than suitable for the ROV bottom controller system which does not draw much more than 5 - 6 Amps at any time.

It is estimated that this single LIPO battery will give a typical run-time of around 15 minutes. This can easily be extended by simply adding more batteries in parallel.

LIPO batteries are not without their limitations though and need to be carefully handled. The first battery used was destroyed by accidently shorting the positive and negative terminals when attaching a connector, and a second destroyed by inadvertently not disconnecting it after use and letting it discharge too low: if the cells are discharged below 9 V they cannot be recharged. They are also prone to exploding if punctured or charged incorrectly, and a special LIPO "balancing" charger must be used to ensure each of the three internal cells is charged at and to the same voltage, without over-charging.

The motors and lights are driven directly off the LIPO's nominal 11.1 V supply. A 5 V linear voltage regulator, TS7805CZ, is used to reduce the 12 V down to 5V for the mbed. The 3.3 V required by the other logic devices and the RS232 IC is generated by the mbed's onboard voltage regulator.

5.8 MOSFET's

The lights, and the four uni-directional horizontal thrusters, operating at 12 V, are simply switched on or off as required by the microcontroller. Unfortunately the microcontroller cannot handle the 12 V switching task and so a MOSFET is employed as a switch.

The major factors considered when selecting the MOSFET were:

- cost
- availability
- usability
- suitability

Based on these factors, the RFP30N06LE was chosen as it was readily available from a trusted supplier, costs only £0.89, was suitable for through-hole construction, had a voltage and current capacity of 60 V and 30A and was compatible with the 3.3 V operation of the microcontroller [15].

To use these devices as simple logical switches, it was found necessary to add a 10 KΩ resistor from the gate to ground, as without the resistor the MOSFET would not turn off due to the device capacitance. With the resistor fitted, the capacitance now has a path to dissipate and the device turns off.

5.9 H-Bridges

The three vertical thrusters, operating at or around 12 V, require bi-directional variable speed control to enable the ROV to ascend and descend. For this reason, a simple MOSFET acting as an on / off switch will not work. A configuration of switching relays could be used to change direction, but they do not switch fast enough to be used with a pulse-width-modulated variable-speed system (see section 5.9.1).

The solution is to use a circuit arrangement known as an H-bridge as they allow for high-speed-switching bidirectional motor control. An H-bridge can be built from discrete components but many IC versions are readily available so it was decided to use one of these, if a suitable one could be found.

Once again, the major factors considered when selecting an H-bridge IC were:

- cost
- availability
- usability
- suitability

Unfortunately the choice is not as easy as when choosing a MOSFET: most modern H-bridge motor drive IC's are only supplied in surface mount format which is not compatible with the through-hole construction techniques used in this project. This greatly limits the choice of components available but there is one component found that meets some of the criteria: the LMD18200 from National Semiconductor (now a part of Texas Instruments).

The chip is readily available, and is suitable for the application, being able to deliver up to 3 A continuously [16]. It is supplied in a cumbersome 11 pin dual inline TO-220 package with an awkward pin spacing of 0.67" but fortunately a break-out board to convert to the more common 0.1" pin spacing is available. The major issue with this IC is the cost: £18.40 each (plus £1.28 for the breakout board). Figure 5.5 shows the three LMD18200's mounted on their breakout boards, and installed on the strip-board. The chip has a number of connections:

Pin Number	Connected To	Use
1	no connection	-
2	motor +	motor connection +
3	mbed pin 18	motor direction
4	brake	not used, tied to ground
5	mbed pin 21	motor PWM
6	12 V	power supply
7	ground	circuit ground
8	current sense	not used
9	mbed pin	thermal overload detection
10	motor -	motor connection -
11	no connection	-

Table 5.2. LMD18200 pin assignment.

As per the recommendation of National Semiconductor, 300 µF bypass capacitance was used to absorb any back EMF caused by switching the inductive load, and a heatsink was attached.

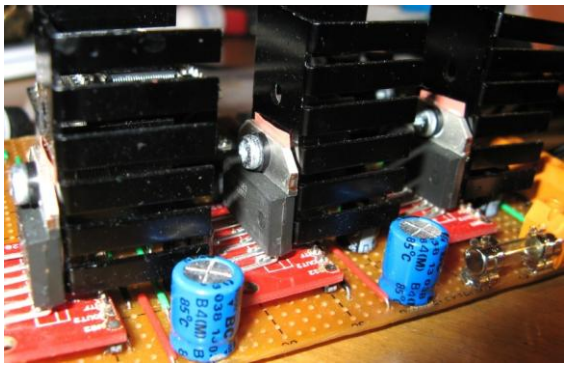


Figure 5.5. Three LMD18200 H-bridge IC's mounted on their break-out boards.

5.9.1 Pulse Width Modulation

Pulse Width Modulation (PWM) is a method commonly used for digital control of certain analogue devices such as DC motors and LED lights. It involves sending a pulse train of high and low levels, at varying ratios (the duty cycle), to effectively mimic a varying voltage level. A digital PWM system offers a very flexible and fine-grained control of the speed of a motor by being able to control programmatically the time of the high level output compared to the low level output.

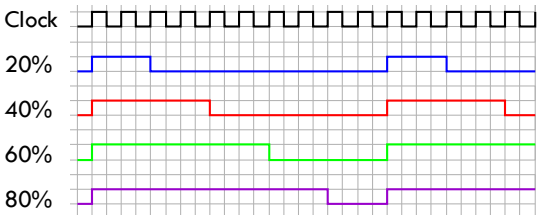


Figure 5.6. PWM duty cycle operation.

As can be seen in Figure 5.6, when there is a high level 20% of the time, and a low level 80% of the time (i.e. a 20% duty cycle), the effective average voltage over the total period is 20% of the maximum output voltage. If that maximum output voltage is 12 V, then: 20% of 12 V = 2.40 V. Accordingly, when there is a high level 80% of the time, and a low level 20% of the time, the effective average voltage over the total period is 80% of 12 V = 9.60 V. Therefore by varying the ratio of high-level time to low-level time, many different average voltage values can be obtained between 0 V and 12 V.

The total time period for each PWM cycle, and therefore the switching frequency, can be varied according to the components used. It is common for DC motor PWM operation to have an operating frequency at a level outside the range of human hearing, though that is not of concern when the motors are submerged. A period of 100 microseconds, giving an operating frequency of 10 kHz was found to be perfectly acceptable.

5.10 Data Communications

The bottom controller has two separate data communications channels.

5.10.1 Communication with the Top Controller

Communications between the top controller and bottom controller, via the tether, are by RS232 serial. An EXAR SP3232ECP-L driver chip is used in both controllers for this.

5.10.2 Communications with the Development PC

The mbed development board includes a USB connection to the development PC. This connection was used, in addition to downloading program code, to give a serial terminal interface direct to the bottom controller for debugging purposes.

5.11 Video Camera

A number of considerations were made when examining the options for generating the video image sent up the tether to the operator:

- cost
- ease of application
- image type and quality

Very expensive diving cameras, in very expensive water-proof housings, are readily available but the aim was to keep the cost of the video camera to a minimum. The solution was to find a small camera that would operate from within the existing water-proof enclosure (with the obvious addition of a window). This would allow the use of the existing power supply, remove the need for an additional water-proof housing, and minimise external wiring junctions.

The camera chosen, as can be seen mounted on the front of the bottom controller in Figure 5.1, is the colour CM-26N/P CMOS board video camera. It is powered from the 5 V supply of the bottom controller. The video signal output is analogue TV quality (PAL or NTSC) via a 2-wire composite format (signal and ground) [17].

The video signal is transmitted to the operator via one of the wires inside the Ethernet cable. It is then available as output from the top controller for connection to a tv, monitor or PC fitted with suitable analog video capture capabilities.

Despite operating perfectly well for a few months during testing, in February the camera module developed a fault and has been returned to the supplier for replacement. Unfortunately the replacement has not yet been received as they are currently out of stock.

5.12 Lights

As the development and testing of the ROV has been conducted in relatively shallow (maximum depth 5 metres) suburban swimming pools, the development of the lights has not been absolutely necessary, and has currently not progressed beyond some basic investigations into driving and mounting LED’s.

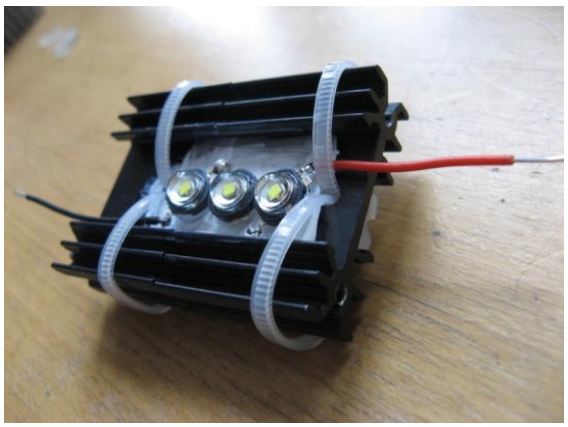


Figure 5.7. LED light experiment.

Figure 5.7 shows an example of three white LED’s connected in series and “mounted” to a

heatsink that was at hand at the time. Considerable further work is necessary here on mounting, driving, and ultimately sealing the lights, and it may end up simply easier to purchase pre-made submersible lights. For the short term, a diving torch is available if required.

Full support for the addition of lights has been made in the bottom controller circuitry, control software, top controller switch, and ROV wiring, and the lights simply need to be connected to the in-place wiring and physically mounted to the ROV frame.

5.13 Data Logger

The IMU daughterboard used for this project conveniently includes a micro SD card slot and this is used for logging data as required.

A typical sample of the data obtained during one of the data logging exercises is shown in Table 5.3.

Time stamp	Pitch angle	PitchAdj	VB motor power
1.034826	0.031372	-0.015686	0.007500
1.176539	0.015686	-0.007843	0.015417
1.234058	0.015686	-0.007843	0.007500
1.291520	0.015686	-0.007843	0.007500
1.349056	0.015686	-0.007843	0.007500
1.480808	0.015686	-0.007843	0.007500

Table 5.3. Example of logged data.

Any data logged is loaded into Microsoft Excel and MATLAB for further analysis.

5.14 Depth Sensor

A depth sensor was initially included in the ROV design specifications, but has not been implemented as yet, as the search for a suitable sensor, at a "reasonable" price has so far been unsuccessful.

Within the last few days, a "slightly out-of-spec" PAA-6L pressure transducer from Keller (Figure 5.8) was donated to the project, but it has not yet been fitted to the ROV. It operates on the principle of an increased output voltage proportional to the pressure on the outer diaphragm [18].

Figure 5.8. Keller PAA-6L pressure transducer.

5.15 Software

The design of the bottom controller software running on the ARM microcontroller follows a modified "interrupt-driven" model. Periodic tasks have been set up to run at regular intervals using a timer function that generates interrupts. These tasks include:

- read the temperature sensor once every 3 seconds
- read the tilt sensors 4 times per second ie every 0.25 seconds

The reception and subsequent transmission of serial data to and from the top controller is handled as an asynchronous interrupt-generating event. In addition, other asynchronous events such as the detection of a leak, or a thermal overload on an H-bridge are also handled by the interrupt mechanism.

Figure 5.9 shows a representation of the system.

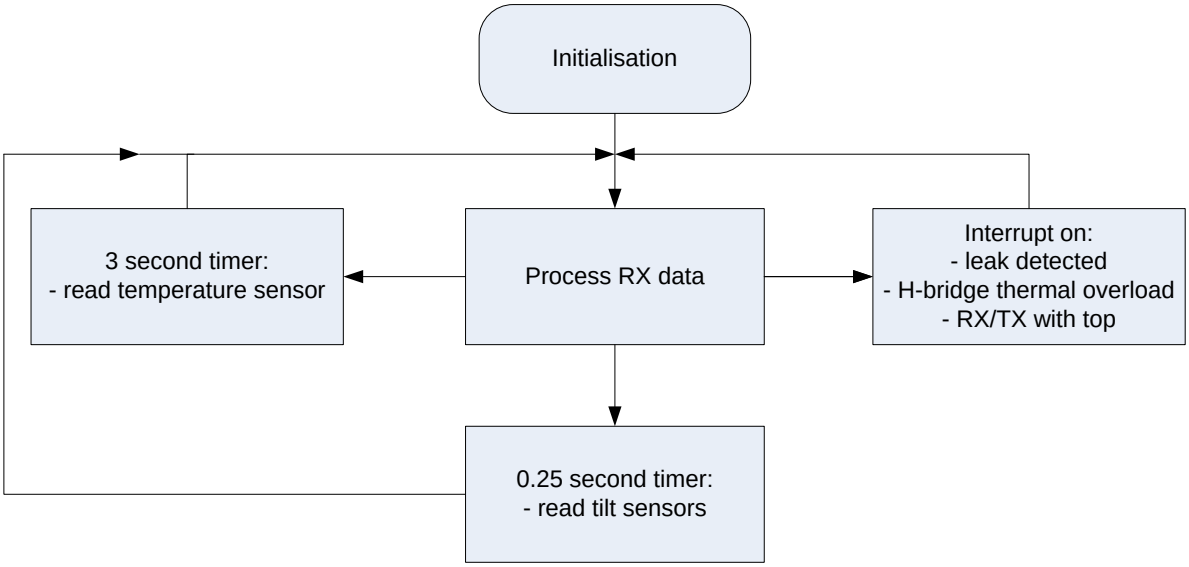


Figure 5.9. Bottom controller system block diagram.

6. Self-Stabilising Control System

As was discussed in Chapter 1, there are certain operational problems associated with underwater vehicles, primarily to do with imaging and manipulators, when they are not sitting level in the water. To counter this, in this ROV project a control system has been developed to attempt to correct any major variations in the pitch and roll axes (see Appendix H for an explanation of motion around these axes). Before examining this system further however, an overview of control system theory is necessary.

6.1 Control System Theory

In general, any system that can adjust itself, based on measuring itself, is thought of as a control system [19]. These control systems are commonly known as closed-loop controllers.

As can be seen in Figure 6.1, the closed-loop controller adjusts its output based on a comparison of the desired output with a measurement of the actual output. The outcome of this comparison (a subtraction) is known as the "system error", and this process is known as "negative feedback".

An open-loop controller, on the other hand, is one whose desired output is set to a certain value. It simply does not have the ability to adjust itself based on any variation in the output as there is no feedback process. This is shown in Figure 6.2. Open-loop controllers are subject to, and cannot react to, external influences and are unsuitable for use in this project, therefore a closed-loop system has been implemented.

6.2 Self-Stabilising System Overview

The self-stabilising control system, configured as a closed-loop controller, is designed to react to measured tilts on the pitch and roll axes by applying power to the vertical thrusters, to induce an inverse rotation around the relevant axis, to correct the measured tilt. Accordingly, any tilt that is measured is considered the "system error" and it is fed back into the correction mechanism.

Figure 6.3 shows a block level overview of the control system where the desired tilt is set to zero and the current tilt level is fed back as the error.

To examine this control system in further detail, only the pitch axis (with only a single thruster for control) will be considered. The thruster is controlled by a PWM system (see section 5.9.1) where an input value to the PWM system gives a certain output level. For example, an input value of 0% equals no power applied to the motor, and an input value of 100% equals maximum power applied to the motor.

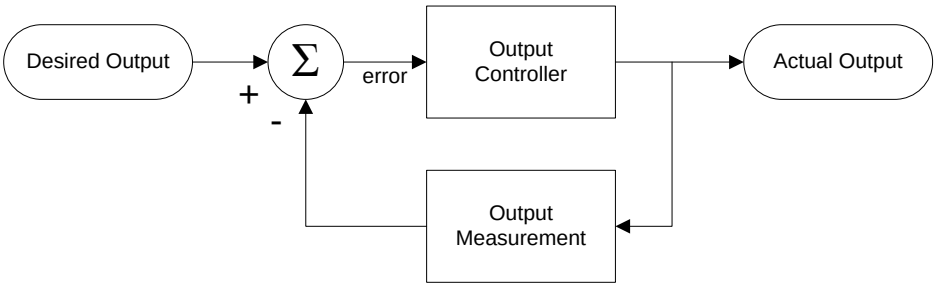


Figure 6.1. Closed-loop control system.

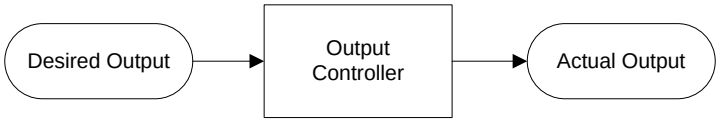


Figure 6.2. Open-loop control system.

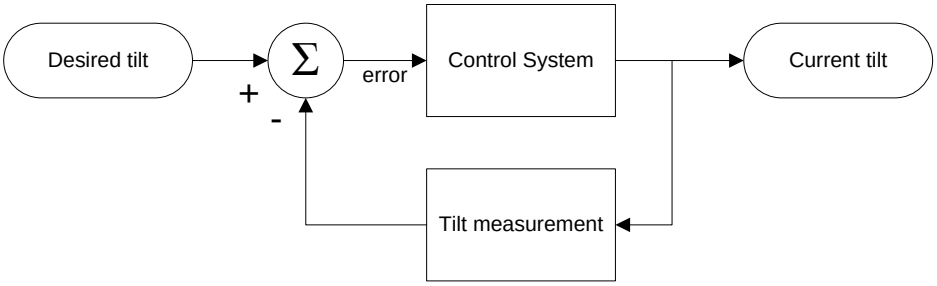


Figure 6.3. Control system overview.

6.3 P Controllers

There are many different ways to implement a closed loop control system. Perhaps the simplest method of correcting any tilt error is to use what is known as a 'P controller', where 'P' refers to a proportional multiplier constant [20]. The calculation is relatively simple: take the measured tilt value, multiply it by some constant (P_k) and apply that to the motor as a PWM percentage.

For example, say the tilt measurement value obtained from the accelerometer is 0.34202 (≈20 degrees), and the P_k constant is 40:

$$PWM\ output\ = tilt\ value\ \times P_k \qquad (4)$$

$$= 0.34202\ \times 40 = 13.68\%$$

Mathematically, the P controller can be stated as:

$$y_P(t) = P_k e(t) \qquad (5)$$

Where:

- $y_P(t)$ is the P controller output
- e(t) is the error input (tilt)
- P_k is the proportional constant

Unfortunately a P controller it is not ideal: if the P_k constant is set too low it may never actually reach the desired level (Figure 6.4), and if it does, it may simply take too long. Conversely, if the P_k constant is set too high, the response of the system will overshoot and possibly oscillate and never reach a steady state at the desired level (Figure 6.5).

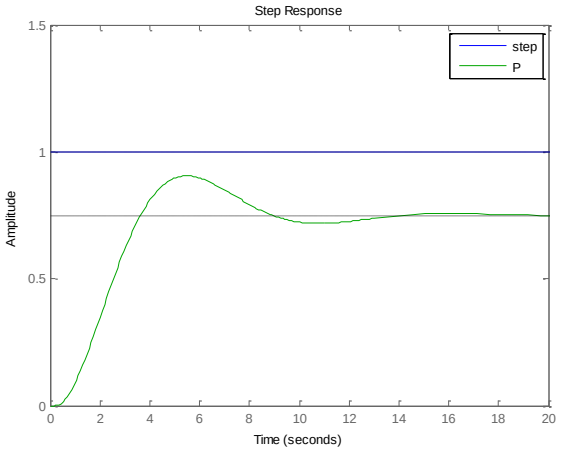


Figure 6.4. P controller output where P_k is too low.

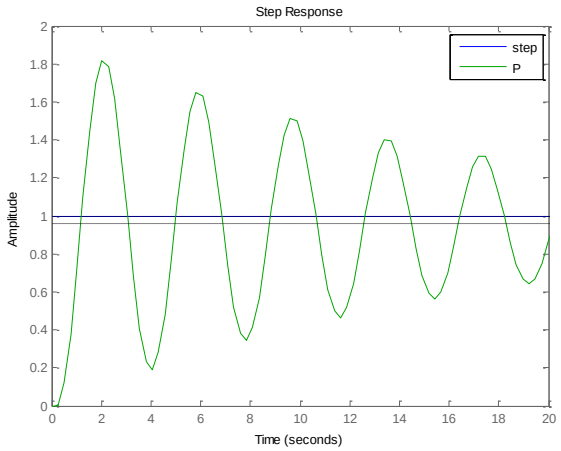


Figure 6.5. P controller output where P_k is too high.

Even with careful adjustment, error, overshoot, and oscillation are inherent problems with P controllers, and clearly something that achieves a result closer to the ideal is required.

6.4 PID Controllers

As Bennett (1993) has said:

"... it has been shown repeatedly that in the absence of any knowledge (in terms of a dynamical model) of the process to be controlled, the PID controller is the best form of controller." [21]

This statement has led directly to the investigation and implementation of a PID-type controller for this project as a dynamical model is not readily available for the pitch tilt correction system.

6.4.1 An Overview of PID Controllers

A PID controller is one that incorporates the P controller previously examined, along with a differential element and an integral element, hence the P, I and D's in the title.

Mathematically it can be stated as:

$$y_{PID}(t) = P_k e(t) + I_k \int_0^t e(\tau)d\tau + D_k \frac{d}{dt} e(t) \quad (6)$$

Where:

- $y_{PID}(t)$ is the PID controller output
- $e(t)$ is the error input (tilt)
- P_k is the proportional constant
- I_k is the integral constant
- D_k is the derivative constant

The block diagram shown in Figure 6.6 may help visualise the concept.

The addition of the integral and derivative terms have significant effect on the output of the controller and, with careful tuning, can often overcome all the problems associated with the P controller on its own. These extra integral and derivative terms come at a cost however, in complexity and performance, and some systems do not require or justify those costly additions. Fortunately simpler variations can work just as well, depending upon the system requirements.

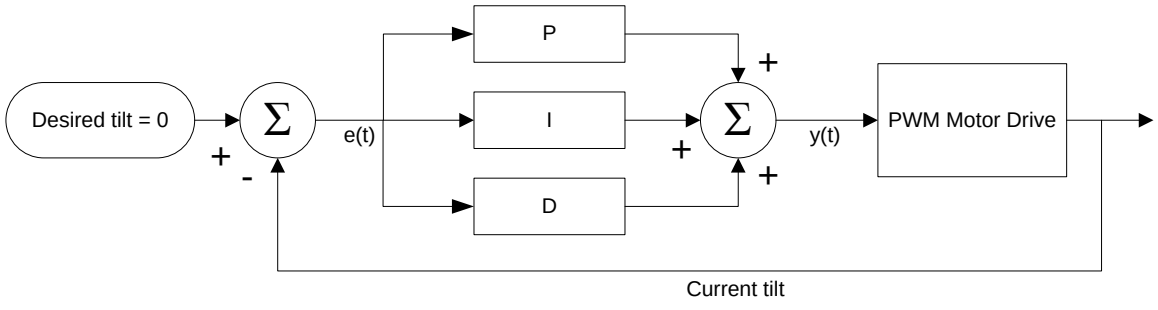


Figure 6.6. PID block diagram.

6.4.2 The PI Controller

The PI controller is one of the variations of the PID controller in that it omits the derivative element. It is stated mathematically as:

$$y_{PI}(t) = P_k e(t) + I_k \int_0^t e(\tau) d\tau \qquad (7)$$

The integral term, by accumulating error over time helps to reduce the steady state error. Figure 6.7 shows the result when an integral constant I_k is added to the P controller. If this constant is set too high overshoot will result.

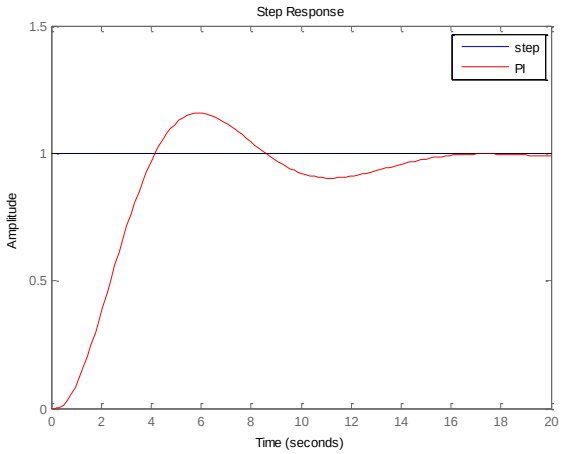


Figure 6.7. PI controller output.

6.4.3 The PD Controller

The PD controller is another of the variations of the PID controller in that it omits the integral element. Mathematically it is stated as:

$$y_{PD}(t) = P_k e(t) + D_k \frac{d}{dt} e(t) \qquad (8)$$

The derivative term, by reducing the rate of change of the output over time, helps to reduce any overshoot and can therefore improve stability. Figure 6.8 shows the result when a derivative constant D_k is added to the P controller. Compare this with the output from the P controller only in Figure 6.4 and you can see the effect it has. If the derivative constant is set too high, increased sensitivity to noise results which can lead to instability.

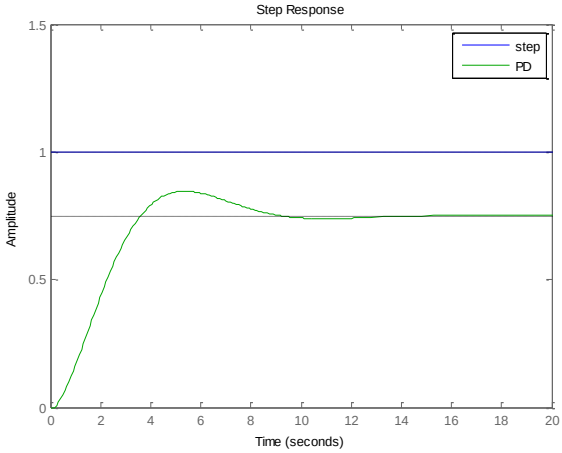


Figure 6.8. PD controller output.

6.4.4 The PID Controller

The PID controller is the variant that includes both the integral element and the derivative element. Using all three elements can result in a response that:

- has a steady state at the desired level
- quickly reaches a steady state
- does not over- or under- shoot excessively
- is stable in operation

Figure 6.9 shows the example with all three PID terms in operation and a comparison with the previous examples. It can be seen there is less overshoot than the PI controller gives and the desired steady-state level is achieved, unlike the P controller and PD controller.

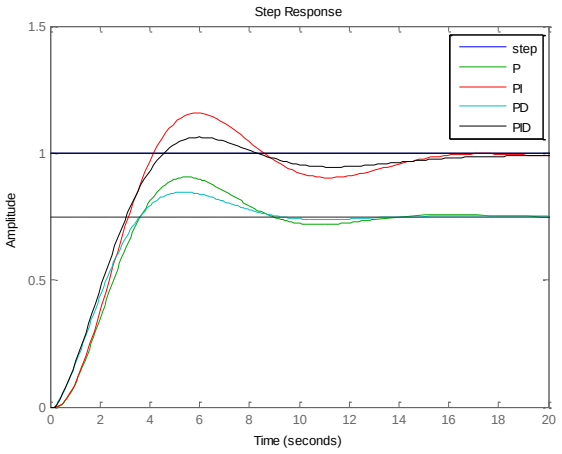


Figure 6.9. PID controller variant outputs.

6.4.5 Tuning a PID-type Controller

The process of finding the optimal constants P_k, I_k and D_k, for a control system is known as 'tuning'. The tuning process is often done using mathematical modelling tools. An example is the MATLAB PID Tuner shown in Figure 6.10 which will automatically adjust the P_k, I_k and D_k constants to give the optimum response based on a number of adjustable parameters such as the response time (via the slider at the bottom) and the type of controller to be used.

In addition to the P_k, I_k and D_k constants, the user is also presented with information on the rise and settling times, the overshoot percentage and peak value, and if the system is considered to be closed-loop stable.

Of course, all the auto-tuning tools require that you have a model of the dynamic behaviour of the system (i.e. a transfer function representing your system), and, as will be shown in Chapter 7, without a model the tuning process is simply guess work or tedious experimentation.

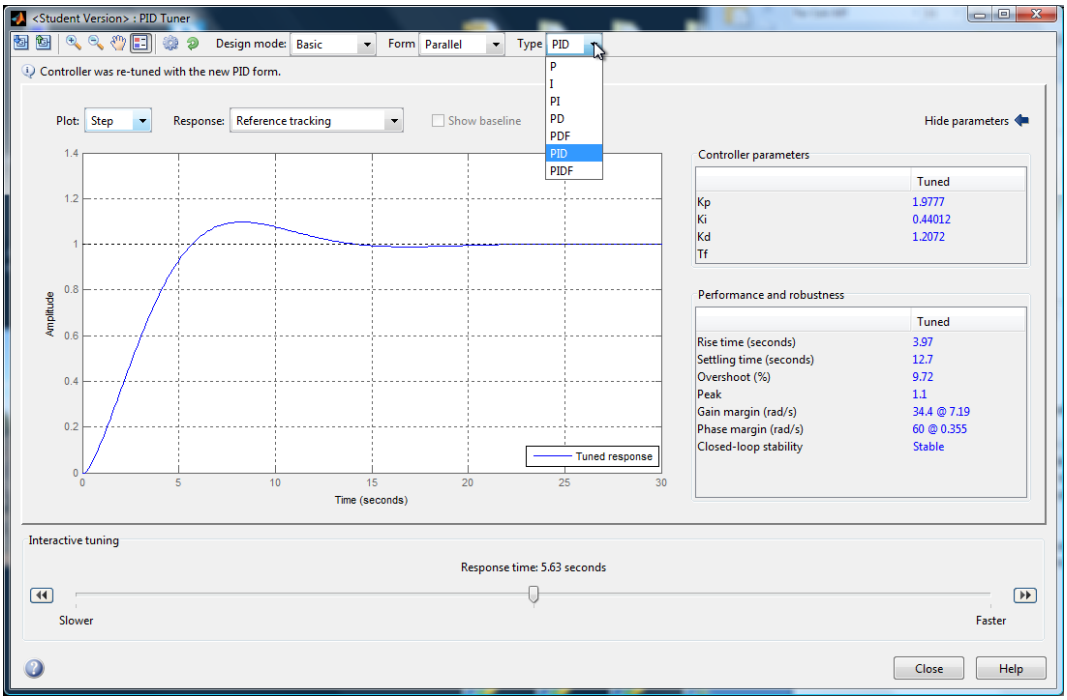


Figure 6.10. MATLAB's PID Tuner.

6.5 Implementing a PID Controller

Having thus determined that a PID controller, or variant thereof, may be appropriate for this project, implementation and testing was the next step.

An incredibly useful element of the mbed development platform is the large range of official ARM-developed, and community-developed library functions available, and the PID controller library is an excellent example. Following significant research and evaluation, and then successful testing, it was decided to utilise this excellent free-to-use library on the simple principles of open-source software and community code reuse.

Actual use of the controller library is trivial: once the controller object has been instantiated and initialised, actual operation is simply two lines of C code:

```
// send current pitch tilt to PID
controller as current error
pitchCon.setProcessValue(pitchTilt);

// let PID controller do its thing
pitchAdj = pitchCon.compute();
```

The output of the controller is set to be a value in the range [-1.0 , 1.0] which is equivalent to the PWM output level in the range [-100% , 100%] that is applied to the thruster.

This controller was shown to work well out of the water: the greater the ROV is tilted, the faster the thruster motors rotate, and varying the P_k, I_k and D_k constants showed obvious variation in the response.

The obvious next step was to determine the P_k, I_k and D_k constants by in-water experimentation. This is detailed in Chapter 7.

6.6 Building a Model of the Pitch Control System

After the initial testing to determine the P_k, I_k and D_k constants was performed, an effort was made to try to build a mathematical model of the pitch tilt correction system. The block diagram of the control system is shown in Figure 6.11.

The system has three cascading 'gain' blacks, each with their own transfer function:

- PID Controller block
- PWM → Motor block
- Physical Response block

These three transfer functions, in the Laplace domain, together give an overall transfer function for the pitch tilt correction system of:

$$G_O(s) = \frac{feedforward}{1 - feedback} \tag{9}$$

$$= \frac{G_{PID}\ (s) \times G_{PWM}(s) \times G_{PR}(s)}{1 - G_{PID}\ (s) \times G_{PWM}(s) \times G_{PR}(s)} \tag{10}$$

Where the transfer function for the PID Controller block is:

$$G_{PID}(s) = P_k + \frac{I_k}{s} + sD_k \tag{11}$$

The transfer function for the PWM → Motor block is effectively unity as no modification to the magnitude of the signal occurs, except for removing the sign for direction control:

$$G_{PWM}(s) = 1 \tag{12}$$

Despite the efforts detailed in Chapter 7, the transfer function for the Physical Response block has not yet been determined:

$$G_{PR}(s) = ?$$

Therefore:

$$G_O(s) = \frac{(P_k + \frac{I_k}{s} + sD_{k)} \times G_{PR}(s)}{1 - (P_k + \frac{I_k}{s} + sD_{k)} \times G_{PR}(s)} \tag{13}$$

Only once the transfer function of the pitch tilt correction system has been determined, can the stability and frequency response of the system be examined and determined. Tuning of the P_k, I_k and D_k constants can then also occur so as to ensure an appropriate response time and behaviour.

6.7 Additional Aspects of the Self - Stabilising Control System as Implemented

The pitch tilt correction system as actually implemented includes a directional element of pitch control to allow for positive and negative pitch angles. The magnitude of the response of the control system is identical regardless of tilt direction, but the rotation direction of the thruster propeller varies depending upon the sign of the tilt measurement.

The roll tilt correction system operates in a very similar manner to the pitch tilt correction system but with three important differences:

- includes support for two thrusters operating at half the required power each
- allows for the opposite rotation of each thruster required for the necessary opposing action
- allows for simultaneous vertical ROV positioning by the operator, and tilt correction by the control system

Currently these simultaneous vertical positioning operations have 50% of the power allocated to each thruster, with the other 50% assigned to the control system. It may be possible to reduce the weighting applied to the control system and therefore increase the power available to the operator for vertical positioning if desired.

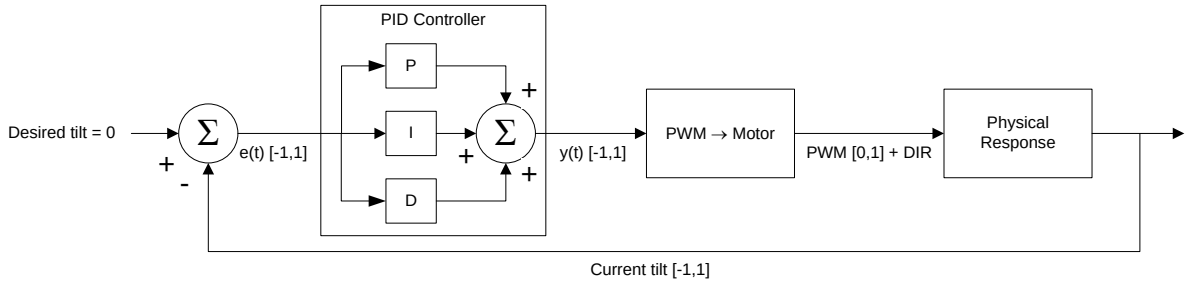


Figure 6.11. Block diagram of the Pitch Tilt Correction Control System.

7. Testing and Measurement

Testing of the many unique elements of this project was an essential and ever-present requirement, as inadequate testing could easily result in catastrophic (i.e. expensive) results. The general approach was to systematically test each and every change as thoroughly as possible before making another change, and to minimise the times where multiple changes were made simultaneously. Experience has shown that with multiple simultaneous changes, trouble-shooting becomes exponentially more difficult.

Though testing effectively never ceases, once the ROV was shown to be effectively operational the measurement phase commenced.

The following sections detail the specific aspects of the testing and measurements undertaken.

7.1 Testing the Electronic Circuits

Following this principle of making and testing individual changes one at a time, the circuits were built-up element-by-element. The approach was to first test each element, both electrically and programmatically, in isolation from the other elements of the system to ensure they performed as required. Only once they were proven to work in isolation were they integrated into the evolving system software and circuit built on a breadboard, and tested with the other elements already put in place.

This gradual addition, element by element, to the working system, with complete testing of the software and circuit as each new element was added, meant that any problems that did occur were relatively easily identified, and only once the entire circuit and its software were working entirely as required on the breadboard was it moved onto the more permanent strip-board.

The other related factor that ensured the development proceeded relatively smoothly was the careful consideration taken to order the implementation of the elements. Again, this systematic approach ensured that when any problems did occur they were relatively easily identified. For example, during the development of the top controller, the required voltage regulator elements were determined and put in place and ascertained to work as required before the LCD display (and its associated backlight with its different power requirements) was added. In this case, trying to add an additional power source for the backlight to the circuit at a later stage could easily have caused issues.

7.2 Testing the Tether and the Wiring

The communication between the top and bottom controllers via the tether is performed using the RS-232 standard. This communication system was initially developed and tested using short (less than 10 cm) lengths of wire, before any attempt was made to use 30 m of Ethernet cable. This systematic approach ensured that when the Ethernet cable was finally used, it worked as planned immediately.

The concentration of the motor and light wires, and the Ethernet cable into a sealed, single 18-core cable and plug and socket combination, is a significant possible point-of-failure, so considerable thought and attention went into it before any work commenced. Firstly the many individual wires were concentrated inside a plastic box and attached to two sections of screw-terminals. Each connection was individually tested and a full test of the operation of the positioning and communications systems was performed to ensure there were no problems.

Construction of the multi-core cable and plug and socket then occurred, with each and every solder joint fully tested with a digital multi-meter (DMM). Again, a full test of the operation of the positioning and communications systems was performed to ensure there were no problems.

Only once these tests were completed successfully was the concentrator box filled with a water-proof resin to seal the screw terminal connections. These steps can be seen in Figure 3.3.

This systematic approach to construction and testing has ensured that, at least so far, there have been no problems associated with the tether or the motor wiring.

Due to the failure of the camera module in February, it has not been possible to fully test the video sub-system in the finished hardware, but during the time it was working, the camera was tested and performed perfectly well.

7.3 Water-Proof Testing of the Enclosure

Again, following the principle of making and testing individual changes one at a time, the construction and validation of the water-proof enclosure proceeded step-by-step:

1. Test empty enclosure with three unmodified end-caps
2. Test enclosure with prototype camera window
3. Test enclosure with second prototype camera window
4. Test enclosure with multi-core connector installed
5. Test enclosure fitted to the ROV for trimming
6. Test enclosure fitted to the ROV for powered testing
7. Test enclosure fitted to the ROV for PID measurements
8. Test enclosure fitted to the ROV for leak location determination
9. Test enclosure to ensure leak has been prevented

All these tests were performed in local suburban swimming pools, at varying depths up to 5 m, in conjunction with the Lodge Scuba Diving Club.

An early design for a camera window was one of the first tests performed. Unfortunately, even before making it into the water, it was found that the window which had been secured onto the face of an end-cap using epoxy glue was not sufficiently well attached. It is understood now that that particular glue does not work well with the type of plastic the end-cap is formed from. An improved window construction method was designed (detailed in Chapter 3) and subsequently successfully tested.

Following the window tests, all tests up to and including the trimming stage were successful with no leaks detected, however during the following powered tests a small leak developed at the multi-core cable entry-point. It was determined that this was due to poor machining of the end-cap (a consequence of using inappropriate tools in this case) and it was replaced, this time with a more accurately machined hole. Further testing has shown this particular issue has now been resolved.

7.4 Trim Testing

The effects of positioning ballast elements (lead weights, and closed-cell foam) follow the Archimedes Principle that objects that displace more water than they weigh (e.g. foam) create a positive buoyancy, and objects that displace less water than they weigh (e.g. lead) create a negative buoyancy. The ideal buoyancy for the ROV is to be as close to neutrally buoyant as possible, and as level as possible. In addition, for stability, it is optimal to have the positively buoyant elements physically higher than the negatively buoyant elements to ensure the centre-of-buoyancy is above the centre-of-mass.

To that end, two pieces of closed-cell foam were attached to the top of the frame and four lead weights of varying sizes were attached to mesh on the bottom of the frame.

Figure 7.1. A friendly SCUBA diver is useful when fine-tuning buoyancy.

The weights were distributed as listed in Table 7.1. The differential required between the front and back, and also between the left and right, is primarily caused by the weight of the off-centre-mounted multi-core cable.

Location	Weight
Front left	520 g
Front right	450 g
Rear left	430 g
Rear right	420 g
Total	**1,820 g**

Table 7.1. Lead weight distribution.

The positions of each of the four lead weights were adjusted in each corner to ensure the ROV was sitting level in the water. Two large pieces of the closed-cell foam were initially used to achieve positive buoyancy and they were then slowly reduced in size until neutral buoyancy was achieved.

The two remaining pieces of foam each measure approximately 310 mm × 25 mm × 25 mm. With a volume of approximately 0.2 litres each, they therefore each give approximately 200 g of buoyancy (based on the assumption that 1 litre of water weighs 1000 grams).

The total ballast offered by these six elements can be calculated as:

$$\begin{aligned} &total\ ballast \\ &= weight\ of\ lead \\ &- equivalent\ weight\ of\ foam \end{aligned} \tag{14}$$

$$= 1820 - 400 = 1420\ grams$$

For operation in sea-water, which has a greater density, some additional trim weights will be necessary.

A short video taken during the trimming process is available to watch at:
www.youtube.com/watch?v=-OokiggFwLU

7.5 Powered Testing

Following the successful trim testing, the next phase of testing the ROV was to ensure it would actually move through the water satisfactorily.

The first test of the operational capabilities found that the horizontal positioning worked exactly as expected, however the vertical positioning would not work: a software bug resulted in an incorrect rotation direction of one of the vertical thrusters. This meant that one thruster was pulling up and the other was simultaneously pulling down. This was corrected and testing the following week showed both vertical and horizontal positioning worked as required (see Figure 7.2).

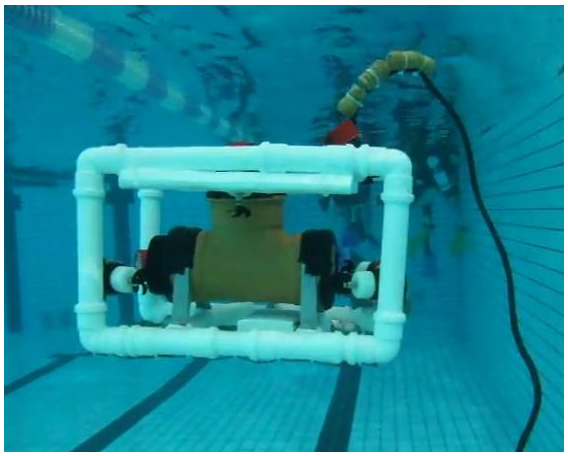


Figure 7.2. Powered testing in the pool.

A short video of this is available at
www.youtube.com/watch?v=x2vWEfYG3sE

At this point the ROV was deemed to be effectively operational and the focus of the testing and measurement then moved to the control system.

7.6 Measurements of Dynamics Characteristics for Control System Development

As detailed in Chapter 6, successful operation of the PID-based tilt correction control system of the ROV requires the determination of the appropriate P_k, I_k, and D_k constants, and a number of attempts were made to measure the dynamics characteristics of the system in various configurations.

7.6.1 Varying the PID Constants

The initial attempts at determining the optimum P_k, I_k, and D_k constants involved operating the ROV in the water, with a deliberate tilt applied, and cycling through varying values of the constants and measuring the dynamics characteristics, i.e. the effective response of the control system, by looking at the resulting tilt level. It was thought the data recorded may give at least an indication of suitable values for the PID constants, but this proved in practice to be too difficult to achieve and was ultimately an ineffective method.

Figure 7.3, for example, shows a plot of the data obtained when the Pk constant was slowly increased from 1 to 50. Before every increase, Pk is reset to 1 and the system, at least in theory, is allowed to settle.

What actually happened was that the ROV was simply driven down to the bottom of the pool by the single operating thruster, and stayed there, and rather disappointingly, no effective tilt correction was observed or recorded for any value of Pk. Another approach was clearly needed.

7.6.2 Varying the PWM Level

Having been unsuccessful in the numerous attempts to manually and exhaustively determine the Pk, Ik, and Dk constants, another approach was tried, where an effort was made to determine the response of the system to the operation of the single vertical rear thruster. Again, it was thought that any data recorded may give at least an indication of the system response, which may then lead to building a model of the system (and from there to determining the Pk, Ik, and Dk constants), but once again this proved in practice to be too difficult to achieve in the available time frame and so far has been ultimately ineffective.

The tests undertaken involved applying a varying level of power, from 0% to 100%, to the thruster, in both rotational directions, and recording the resulting tilt levels. Any tilt recorded should represent a response by the system to the application of thrust from the thruster by a rotation around the pitch axis.

Figure 7.4 shows the results of the measured data, and they confirm the physical observations made at the time that, rather than sitting "mid-water" and rotating as desired, the ROV was repeatedly driven straight down to and up from the bottom of the pool by the single operating thruster. Rather disappointingly again, no effective change in tilt was observed or recorded for any PWM level.

It can be observed at approximately 300 and 700 seconds into the test, when the motor is at 100% full power, that there is a change in the tilt level which could indicate a desirable response, however it was observed that the ROV was sitting on the pool bottom at the time and it is therefore not considered a reliable observation of the system response by pitch rotation.

It can also be observed that at approximately 780 seconds into the test a change has occurred: the rear vertical thruster was manually rotated around the pitch axis so that instead of being oriented vertically, it was set at a 45° tilt. This rotation of the thruster was done as it was observed from the poolside that there was no rotation of the ROV about the pitch axis occurring, and it was thought at the time that changing the angle may make a difference. In fact it imparted a not insignificant horizontal force and the ROV moved forwards and backwards in the water in addition to vertically up and down. No additional rotation about the pitch axis was observed at the pool side, and this particular experiment was deemed a failure.

Finally, a cursory glance at the data post-780 seconds seems to show a positive and then negative tilt correction occurring, but that was

in fact due to the ROV sliding down a steep slope in the pool floor.

A video of the pool session can be seen at: www.youtube.com/watch?v=ZuQB57rVeSg

After an examination of this data, and a careful re-assessment of the physical design, a likely cause of the lack of rotation about the pitch axis has been discovered. It was always assumed that due to the near-symmetrical frame design the centroid, the mid-point between the centre-of-mass and centre-of-buoyancy and the point about which rotation of the ROV would occur, was essentially in the middle of the frame. It now appears that assumption was incorrect as the single rear vertical thruster plus the multi-core cable and connectors do actually significantly change the relative position of the centre-of-mass, but were never considered in this light. This has the effect of moving the centroid closer to the rear vertical thruster, and therefore much lower (effectively zero) angular momentum is imparted by the thruster.

A number of possible solutions to this problem present themselves:

- redesign the frame so that the thrusters are further away from the centre
- add additional weights (and correspondingly offsetting foam) to move the centre-of-mass closer to the centre

Unfortunately, due to time constraints neither have been investigated further and the problem is now listed as work for further consideration.

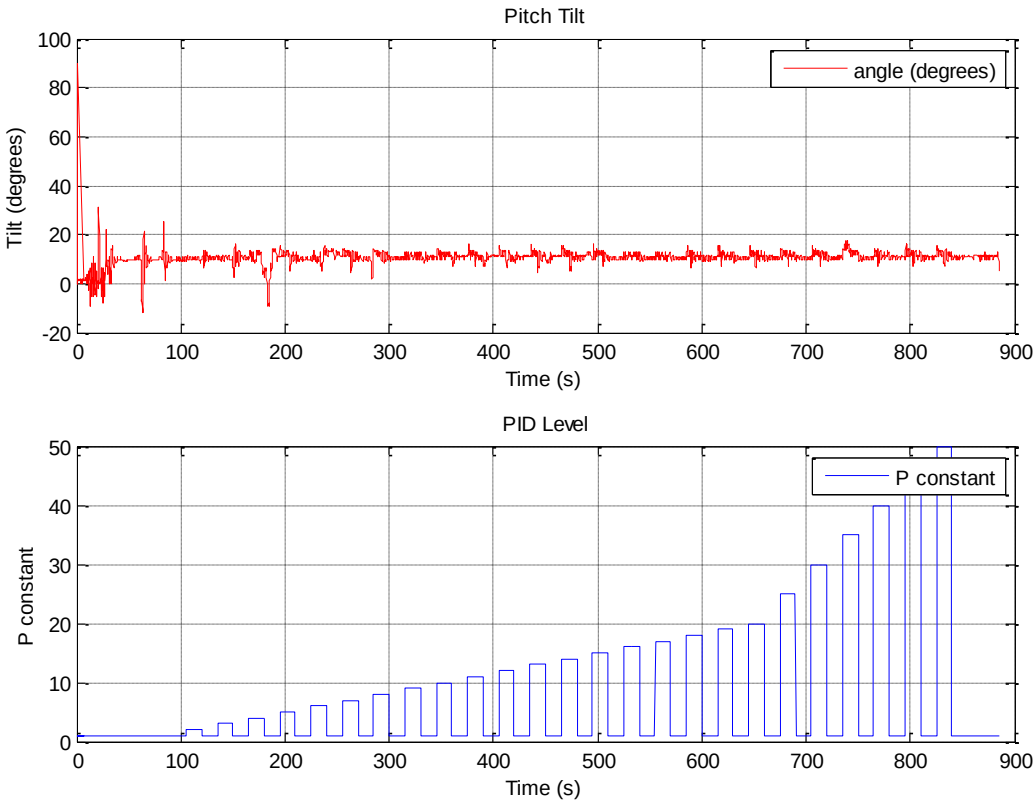


Figure 7.3. Results from varying the P_k constant.

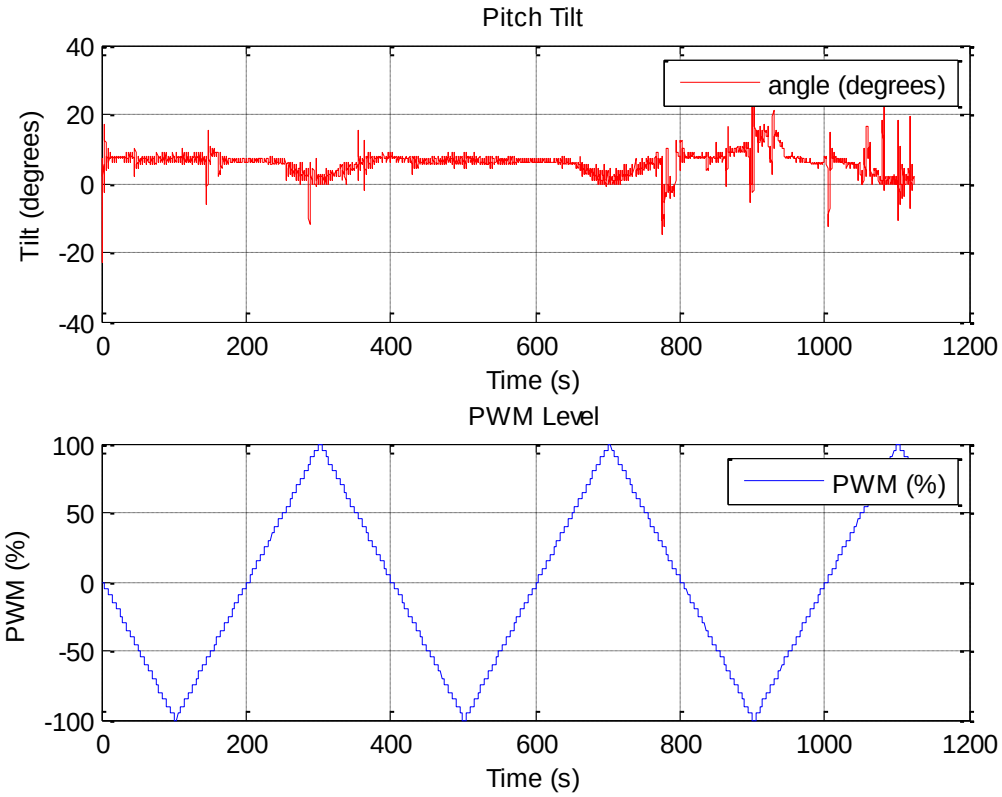


Figure 7.4. Results from varying the PWM level.

8. Conclusions

This project started with the overall goal of combining personal interests and newly developed skills, to produce a potentially useful and interesting device, whilst presenting as many realistic challenges as possible.

The project has now ended with that goal completely satisfied: an extremely interesting, fully working underwater vehicle has been designed and constructed, from scratch, utilising and learning many new skills, and overcoming many, many challenges along the way.

8.1 Results

It would be fair to say that the defined goals of the project, as set many months ago, may not be fully met as yet, however this is really more a function of the time available, and to a lesser degree the finances available, rather than any inability to overcome the challenges presented.

Looking closer at the specific primary and secondary goals:

- **Primary Goal 1: Build an underwater vehicle fitted with thrusters for horizontal and vertical positioning**

  This has been successfully completed.

- **Primary Goal 2: Build microcontroller-based electronic circuitry for operating the thrusters**

  This has been successfully completed.

- **Primary Goal 3: Build remote operator providing basic interface to the vehicle**

  This has been successfully completed.

- **Primary Goal 4: Test the vehicle and identify its dynamics characteristics**

  This has been partially achieved. In- and out-of-water testing of the vehicle has been an ongoing process throughout the entire development process. In-water measurements to identify the vehicles' dynamic characteristics are still currently underway and have not been completed.

- **Primary Goal 5: Design and implement an automatic control system for pitch and roll of the vehicle**

  This has been partially achieved. A working PID controller system has been implemented for both the pitch and roll axes for tilt correction, however, tuning of the control system parameters is still underway.

- **Secondary Goal 1: Test robustness of the control system**

  This has been partially completed through the process of measuring the dynamics characteristics, but until the control system parameters have been tuned (Primary Goal 5), this cannot be completed.

- **Secondary Goal 2: Consider (if needed) any improvements to the control system**

  This has been completed. Section 8.3 contains this information.

- **Secondary Goal 3: Install a video camera and lights on the vehicle**

  This has been partially completed. A video camera system was implemented and was working successfully, but it has since failed, and is currently awaiting replacement. The addition of lights to the vehicle has been fully supported but only partially achieved physically.

8.2 Analysis

A great deal of thought has gone into every aspect of this project in an attempt to minimise the number of problems that might occur, and given the success of the project so far, it has to be said that this considered approach has

worked to a large degree, but it was always known that there would be some major challenges to be faced, particularly in the physical construction aspect.

The design and construction of the water-proof enclosure, was always expected to present many difficulties, and so it has proven, but it is fair to say that the difficulty level was grossly underestimated. A limited budget, and the inability to accurately machine certain components, using inappropriate tools, caused many unnecessary headaches, and wasted a considerable amount of time. It can be simply summarised as: on a limited budget, water-proofing is difficult.

Convenient access to a suitable in-water testing environment also proved to be greatly overestimated. Though excellent support was received from the SCUBA diving club, the practical logistics of taking the ROV to the pool via public transport (in some cases a one hour trip requiring 2 busses and a train), just to run a single 10 minute test, was inefficient to say the least. In addition, local swimming pools are not actually ideal test and development environments as they are inconveniently placed, not readily accessible outside SCUBA club sessions, have no pool-side power supplies to run laptops, do not allow cameras (except under very special circumstances), and offer limited testing space. In addition there are considerable health and safety issues to deal with as well. A number of alternatives were considered, and some even tried, including a number of home bath tubs, neighbour's fish ponds, and a duck pond at a local common. All were unsuitable for one reason or another (size and water quality being the main issues faced).

When looking at the attempts made to try to measure the system's dynamics characteristics, it is now obvious that measuring and characterising a real world (ie non-linear) device, operating in the real world (a non-linear environment) is actually a difficult and complex task and the difficulty level of this task was also greatly underestimated.

Other real-world issues that were faced include the fact that everything costs more than is expected, everything takes longer to arrive than it should, everything takes longer to build than it appears it should, and everything fails at some time. Having reliable and trustworthy suppliers was key to minimising the problems these issue cause.

Despite all these difficulties, there were many positives that came out of this project:

- Many satisfying challenges were faced and overcome that presented excellent learning opportunities
- The subject matter was very varied and very interesting, and this made it actually enjoyable, and gave the opportunity to converse with many interested and interesting people from many fields
- The valuable confirmation, from the successful outcome of the project, of the design and testing methodologies employed, that reaffirms the problem solving skills refined over the last 3 years
- Many valuable skills were developed or refined including:
  - embedded C programming
  - circuit design and board layout
  - assembly and fabrication
- Many new and unfamiliar components and techniques were used such as:
  - LIPO batteries
  - ARM-based micro-controllers on the mbed development boards
  - H-bridges, voltage regulators and accelerometers
  - LCD displays
  - RS232 communications
  - PID controllers and PWM systems

8.3 Future Work

Throughout the entire project, many possible improvements and additions were envisioned, and suggested by others. Many were taken on board, however many more were considered to be simply beyond the scope of this project. Following the completion of this phase of the project, it is hoped that many of the following ideas will be examined for implementation:

- Depth sensor
- Environmental sensors eg water salinity and water temperature sensors
- Digital compass
- Pan-and-tilt for camera

- On-screen data display
- Manipulator
- On-board battery monitoring
- Longer tether
- Additional cameras and lights
- Fibre optic data link
- Wireless data link
- PC control
- PCB construction and design-for-manufacture
- Open-sourcing the design, plans, and software
- Analog control of horizontal thrusters for finer control
- Obstacle detection
- Autonomous operation
- Investigations into alternative control systems

8.4 Alternative Approaches

Throughout the project there have been many decisions made that are, with hindsight, to be regretted. Most of those have been relatively minor and the consequences not severe: routing a motor wire through a certain channel for example. Some modifications and improvements that would be seriously considered, given the opportunity, however are noteworthy:

- Re-route the 18-core cable to minimise weight distribution offset
- Use multiple 6-conductor connectors instead of a single 18-conductor version
- Add a USB-to-wireless data link to the development PC to allow programming and debugging without having to remove the bottom controller from the enclosure
- Conduct more research and testing to determine the optimum angle for operation of the vertical thrusters

Given an unlimited budget, and unlimited time and space, the following would've greatly assisted in producing a better outcome:

- Design and construct a purpose-built pressure enclosure. This is a relatively expensive exercise (≈£200) but will allow greater depths, greater reliability, and greater confidence
- Construct a home-based testing tank to allow for much more convenient in-water testing. The average home bath tub is simply not large enough. Research into this is ongoing.
- Invest in better tools, or find conveniently local ones: personal access to a laser cutter or CNC mill would be incredibly useful

8.5 Summary

In summary, this underwater vehicle project, though somewhat daunting at the start, was extremely enjoyable, and yet incredibly challenging at the same time. It was interesting and varied, included many opportunities to research and apply new and exciting techniques and technologies, and allowed for the development and refinement of new and existing skills. It gave the opportunity for interesting discussions with interesting people, and finally, it gave a real insight into actually developing a product.

9. References

1. Saab Seaeye Ltd, (2009). *Seaeye ROV Comparison Chart*. [online]
Available from: < http://www.seaeye.com/comparerovs.html>
[Accessed: 18 February 2012]

2. Woods Hole Oceanographic Institution, (2007). *Human Occupied Vehicle Alvin*. [online image] Available from: < http://www.whoi.edu/cms/images/v44n1-alvin-intro1en_12472_36562_82629.jpg > [Accessed 9 February 2012] Used with permission

3. Woods Hole Oceanographic Institution, (2007). *Seabed*. [online image]
Available from: < http://www.whoi.edu/cms/images/hanu2-en_28930_42353.jpg >
[Accessed 9 February 2012] Used with permission

4. VideoRay LLC, (2011). *P4 Product Packaging*. [online image]
Available from: < http://www.videoray.com/images/picture/P4PACKAGING_web.jpg >
[Accessed 9 February 2012] Used with permission

5. Saab Seaeye Ltd, (2009). *Saab Seaeye Lynx* [online image]
Available from: < http://www.seaeye.com/pop.html?image=images/compare-rovs/lynx.jpg >
[Accessed 20 February 2012] Used with permission

6. Saab Seaeye Ltd, (2009). *Saab Seaeye Jaguar* [online image]
Available from: <http://www.seaeye.com/pop.html?image=images/compare-rovs/jaguar.jpg>
[Accessed: 20 February 2012] Used with permission

7. SMD Ltd, (2011). *SMD UT-1 trencher*. [online image]
Available from: < http://www.smd.co.uk/resources/page/123_118_128_47_image1_l.jpg >
[Accessed: 18 February 2012] Used with permission

8. Sayers, C. , (1999). *Remote Control Robots*. New York: Springer.

9. Pelekanakis, K. and Baggeroer, A., (2011). Exploiting Space-Time-Frequency Diversity With MIMO-OFDM for Underwater Acoustic Communications. *IEEE Journal of Oceanic Engineering.* October 2011 **36** (4), 502-513.

10. National Defence Research Committee, (1969). *Physics of Sound in the Sea*. Washington: US Dept of the Navy.

11. NXP B.V., (2009). *mbed NXP LPC 1768 prototyping board*. [online]
Available from: < http://www.nxp.com/documents/leaflet/LPC1768.pdf >
[Accessed: 18 February 2012]

12. Powertip Tech. Corp., (2006). *PC1602ARU-HWB-G-Q*. [online]
Available from: < http://www.farnell.com/datasheets/40247.pdf >
[Accessed: 18 February 2012]

13. National Semiconductor, (2000). *LM135/LM235/LM335, LM135A/LM235A/LM335A Precision Temperature Sensors*. [online]
Available from: < http://www.farnell.com/datasheets/36878.pdf >
[Accessed: 18 February 2012]

14. STMicroelectronics, (2009).*LIS331DLH*. [online]
Available from: < http://www.st.com/internet/com/TECHNICAL_RESOURCES/
TECHNICAL_LITERATURE/DATASHEET/CD00213470.pdf >
[Accessed: 18 February 2012]

15. Fairchild Semiconductor, (2004). *RFP30N06LE, RF1S30N06LESM Data Sheet*. [online]
Available from: < http://www.farnell.com/datasheets/65413.pdf >
[Accessed: 18 February 2012]

16. National Semiconductor, (2011). *LMD18200*. [online]
Available from: < http://www.farnell.com/datasheets/78246.pdf >
[Accessed: 18 February 2012]

17. Sparkfun Electronics, (2011). *CM-26N/P C-MOS Color Camera Module*. [online]
Available from: < http://www.sparkfun.com/datasheets/Sensors/Imaging/CM-26N.pdf >
[Accessed: 18 February 2012]

18. Keller Ag., (2009). *Keller Piezoresistive OEM Pressure Transducers*. [online]
Available from: < http://www.keller-druck.com/picts/pdf/engl/3L_10L_e.pdf >
[Accessed: 18 February 2012]

19. Dickinson, M., (2011). *Introduction to Control Engineering*. UK: Elektor International Media.

20. Lewis, P. and Yang, C., (1997). *Basic Control Systems Engineering*. Upper Saddle River: Prentice-Hall Inc.

21. Bennett, S., (1993). *A history of control engineering 1930 - 1955*. London: Peter Perigrinus Ltd.

22. STMicroelectronics, (2010). *AN3182 Application note, Tilt measurement using a low-g 3-axis accelerometer*. [online]
Available from: < http://www.st.com/internet/com/TECHNICAL_RESOURCES/
TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00268887.pdf>
[Accessed: 18 February 2012]

10. Bibliography

- Bohm, H. and Jensen, V., (2010). *Build your own underwater robot and other wet projects*. Vancouver: Westcoast Words.

- van Dam, B., (2010). *ARM Microcontrollers, Part 1: 35 projects for beginners*. Susteren: Elektor International Media.

- Roberts, G. and Sutton, R., (2006). *Advances in Unmanned Marine Vehicles*. Stevenage: The Institution of Electrical Engineers.

- Antonelli, G., (2003). *Underwater Robots, Motion and Force Control of Vehicle-Manipulator Systems*. Berlin: Springer-Verlag.

- N-Nagy, F. and Siegler, A., (1987). *Engineering Foundations of Robotics*. Englewood Cliffs: Prentice-Hall Inc.

- Horowitz, P. and Hill, W., (2008). *The Art of Electronics*. Cambridge: Cambridge University Press.

- Ly, U., (1997). *Stability and Control of Flight Vehicle*. [online] Seattle: University of Washington. Available from: < http://metalab.uniten.edu.my/~farrukh/mywork/FLIGHT~1.PDF > [Accessed 25 February 2012]

11. Appendices

Appendix A - DES Project Specification Form

Title: Underwater remotely operated vehicle
Student Name: Scott O'Brien
Year: 2011/12
Supervisor: Dr. Andrzej Tarczynski
Assessor:
Moderator:

Aims and Description:
To design and build an underwater vehicle with pitch and roll control system

Primary Goals:
1. To build an underwater vehicle fitted with thrusters for horizontal and vertical positioning
2. To build microcontroller-based electronic circuitry for operating the thrusters
3. To build remote operator providing basic interface to the vehicle
4. To test the vehicle and identify its dynamics characteristics
5. To design and implement automatic control system for pitch and roll of the vehicle

Secondary Goals:
1. To test robustness of the control system
2. To consider (if needed) any improvements to the control system
3. To install a video camera and lights on the vehicle

Resources Needed:
- The student will fund the project
- Access to the ECS workshop
- Eagle-CAD (freeware)
- Matlab

Health and Safety Assessment and Arrangements:
- Follow the normal health and safety precautions appropriate to the work performed on the project.

Supervisor signature:
Date: 10 October 2011

Student signature:
Date: 10 October 2011

Appendix B - Top Controller Schematics

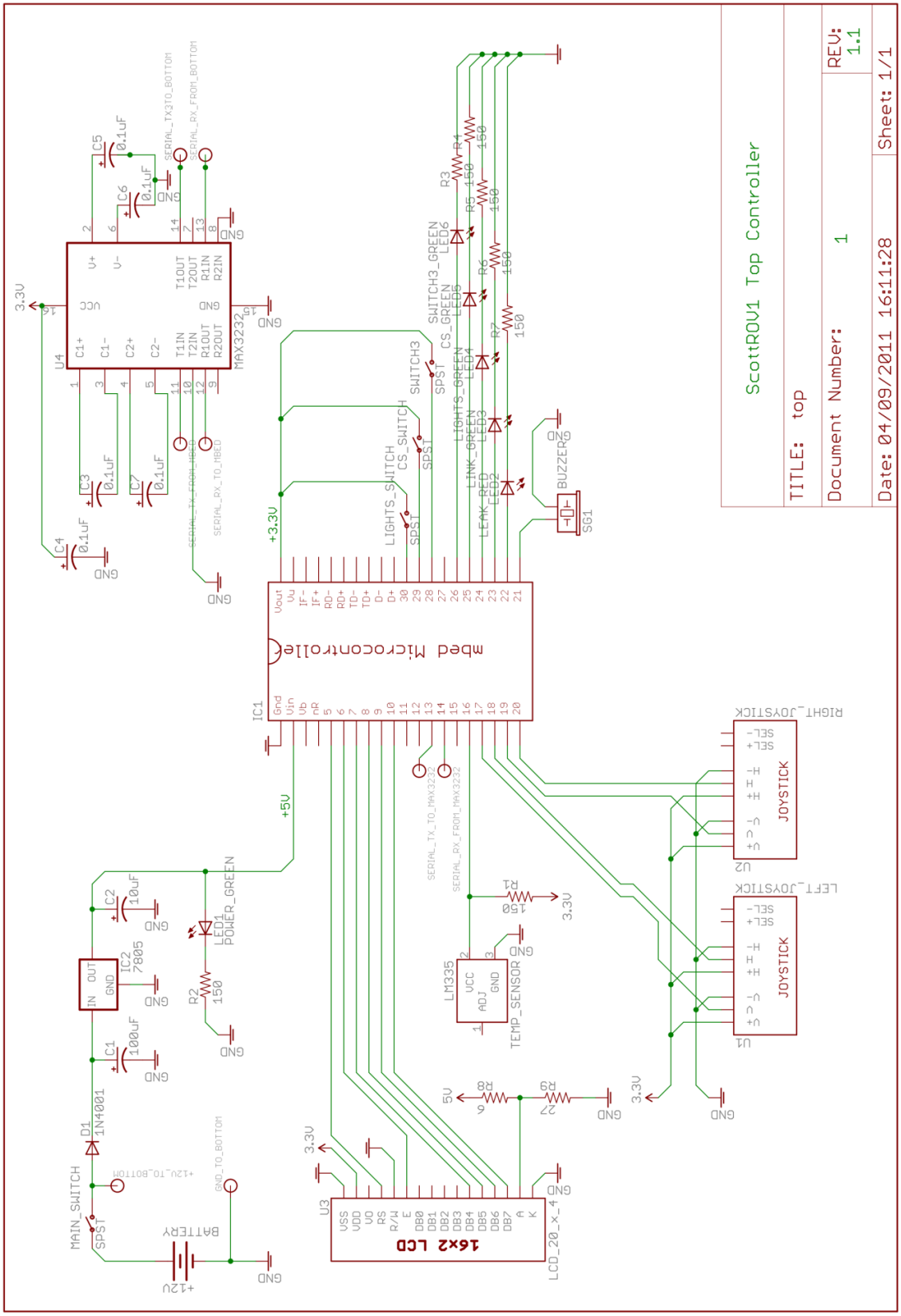


Figure B.1. Top controller schematic.

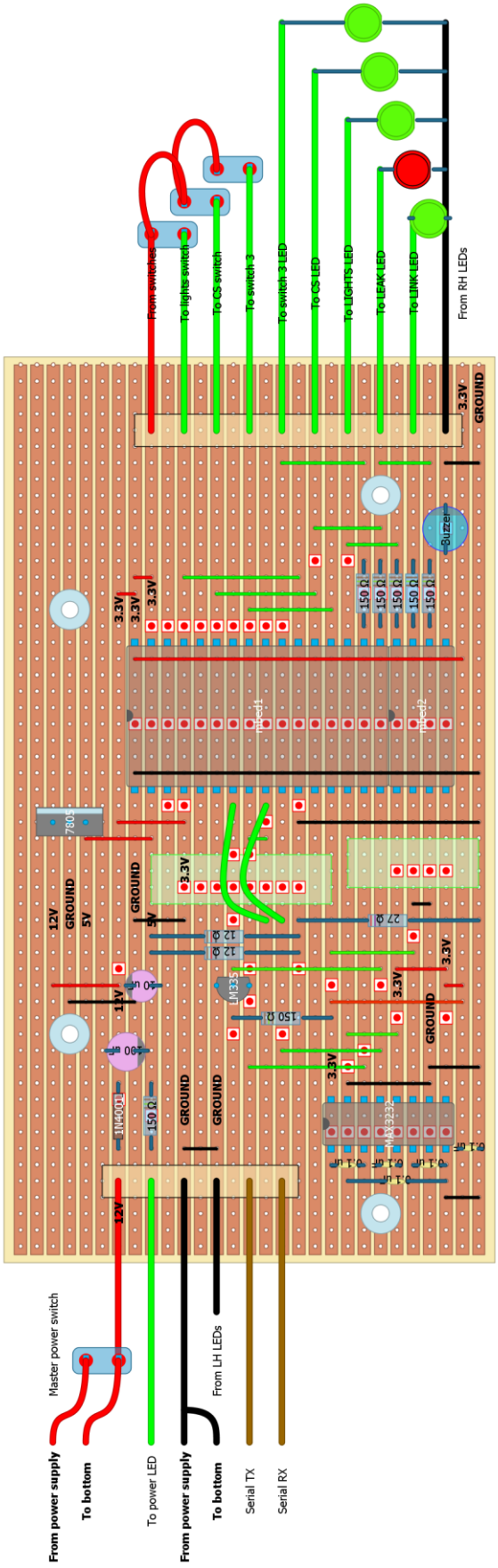


Figure B.2. Top controller strip-board layout.

Appendix C - Bottom Controller Schematics

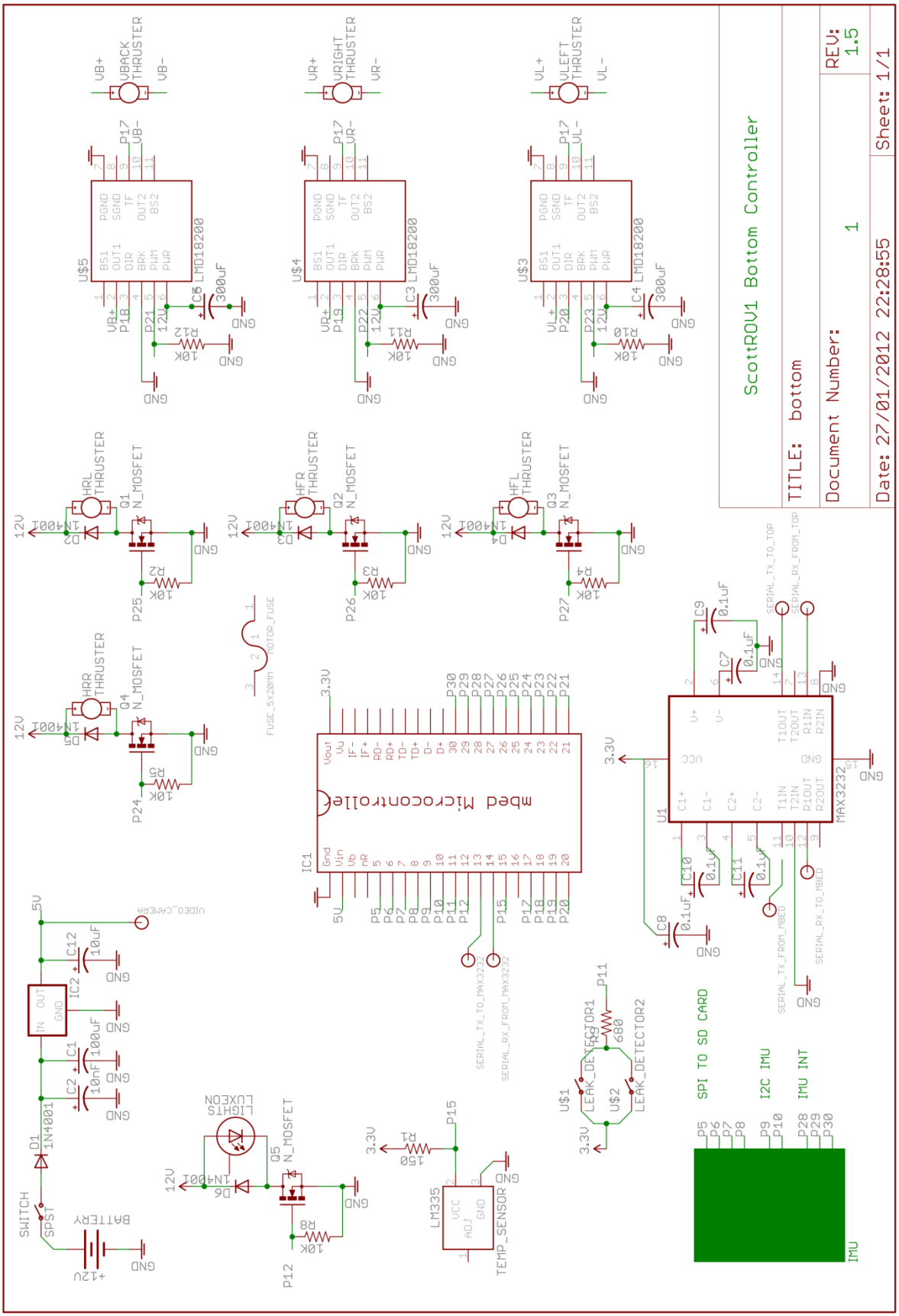


Figure C.1. Bottom controller schematic.

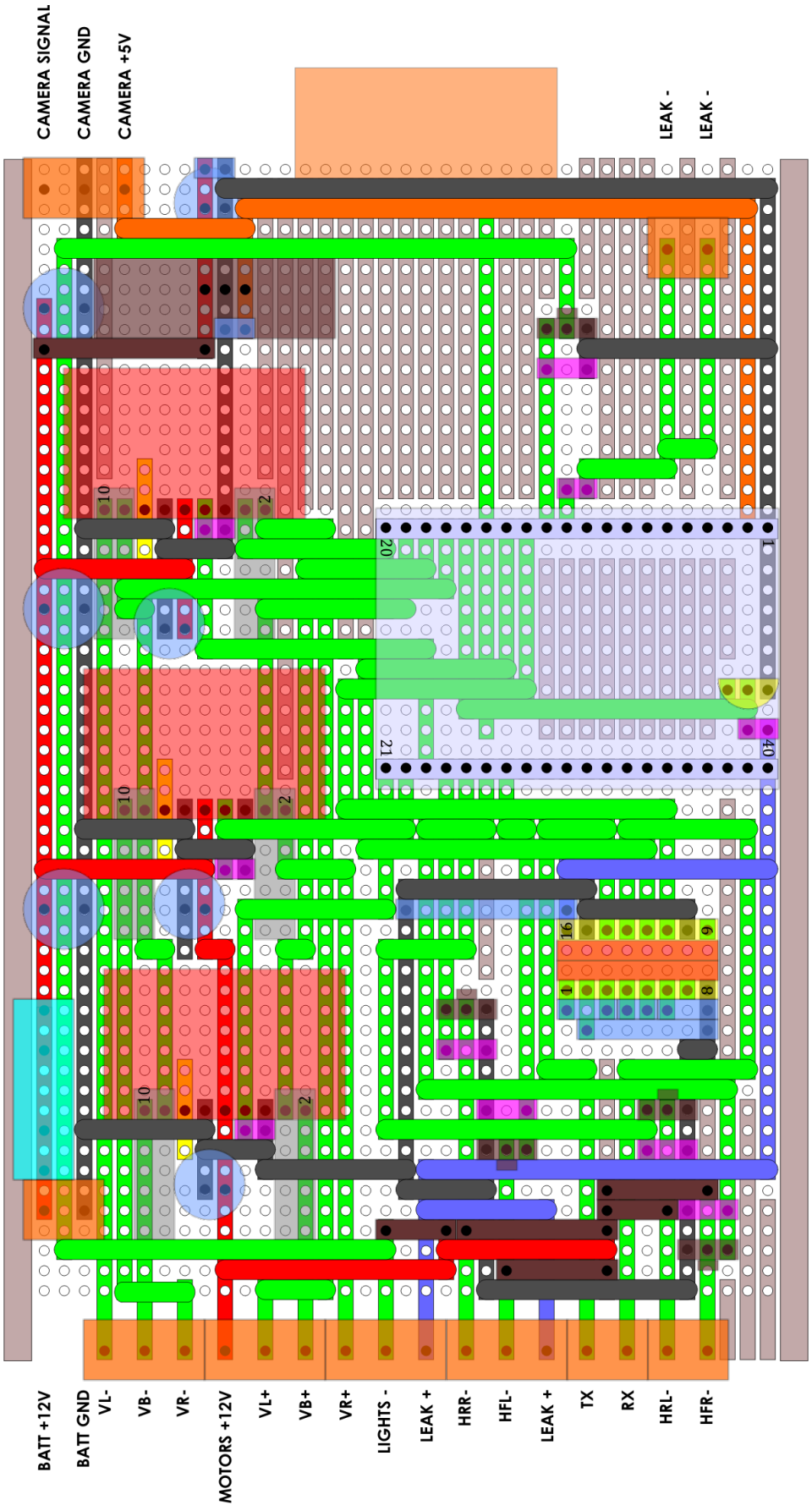


Figure C.2. Top controller strip-board layout.

Appendix D - Top Controller Code

```
// TOP controller for ROV
//  v0.70  24 April 2012
//  by Scott O'Brien

// LIBRARIES
            #include "mbed.h"
            #include "TextLCD.h"

// PIN DEFINITIONS
            Serial pc(USBTX, USBRX);  // diagnostic connction to PC via USB
            Serial topSerial(p13,p14);// tx, rx to bottom cont. via MAX3232

            DigitalOut mbedLED1(LED1);          // onboard led's
            DigitalOut mbedLED2(LED2);
            DigitalOut mbedLED3(LED3);
            DigitalOut mbedLED4(LED4);

            DigitalOut leakLED(p22);            // red
            DigitalOut linkLED(p23);            // green
            DigitalOut lightsLED(p24);          // green
            DigitalOut controlSystemLED(p25);   // green
            DigitalOut switch3LED(p26);         // green

            TextLCD lcd(p5,p6,p7,p8,p9,p10,TextLCD::LCD20x4); // 20x4 LCD

            AnalogIn leftJoystickUpDownAnalog(p17);
            AnalogIn leftJoystickLeftRightAnalog(p18);
            AnalogIn rightJoystickUpDownAnalog(p19);
            AnalogIn rightJoystickLeftRightAnalog(p20);

            AnalogIn tempTop(p16);              // temperature sensor

            InterruptIn lightsSwitch(p30);
            InterruptIn controlSystemSwitch(p29);
            InterruptIn switch3(p28);

            PwmOut buzzer(p21);                 // leak alarm buzzer

// TICKERS AND TIMERS
            Ticker majorEvent;                  // major event ticker
            Ticker tempSensor;                  // read temp sensor ticker

// VARIABLES AND CONSTANTS
            char  raise           = 0;
            char  lower           = 0;
            char  moveLeft        = 0;
            char  moveRight       = 0;
            char  forward         = 0;
            char  reverse         = 0;
            char  turnLeft        = 0;
            char  turnRight       = 0;
            float topTemp         = 0;
            char  lightsOn        = 0;
            char  controlSystemOn = 0;
            char  switch3On       = 0;
            char  controlData[3]  = {0,0,0};        // data to tx to bottom
            char  rxData[5]       = {0,0,0,0,0};    // data rx from bottom
            char  rxPCData        = 0;
            char  dataLoggingOn   = 0;
```

```
            float kP                = 1.0;              // PID "constants"
            uint8_t *kP8            = (uint8_t*)&kP;
            float kI                = 0.0;
            uint8_t *kI8            = (uint8_t*)&kI;
            float kD                = 0.0;
            uint8_t *kD8            = (uint8_t*)&kD;

// FUNCTIONS

// send and recieve data from PC via USB
void SendAndReceiveDataFromPC() {
            if(pc.readable()) {
                rxPCData = pc.getc();
                pc.printf("%c",rxPCData);

                if (rxPCData == '?') {
                    pc.printf("\r\n\n\nScottROV\r\nTop Controller v0.70
                        Bottom Controller v0.70");
                    pc.printf("\r\nkP = %.4f, kI = %.4f, kD =
                        %.4f",kP,kI,kD);
                    pc.printf("\r\nrx[0] = %i, temp = %i, pitch = %i, roll
                        = %i", rxData[0],  rxData[1], rxData[2] - 90,
                        rxData[3] - 90);
                    pc.printf("\r\ncontrolData[0] = %i, controlData[1] =
                        %i",controlData[0],controlData[1]);
                    if (dataLoggingOn) {
                        pc.printf("\r\nData logging is ON");
                    }
                    else {
                        pc.printf("\r\nData logging is OFF");
                    }
                    if (controlSystemOn) {
                        pc.printf("\r\nControl system is ON");
                    }
                    else {
                        pc.printf("\r\nControl system is OFF");
                    }
                    pc.printf("\r\n\n? = this list\r\nl = toggle data
                        logger\r\no = toggle control system\r\nt = transfer
                        data (not enabled)\r\np = change P term\r\ni =
                        change I term\r\nd = change D term\r\n\n");
                }

                else if (rxPCData == 'l') {

                    dataLoggingOn = !dataLoggingOn;
                    if (dataLoggingOn) {
                        pc.printf("\r\n\ndata logger turned on\r\n\n");
                    }
                    else {
                        pc.printf("\r\n\ndata logger turned off\r\n\n");
                    }
                }

                else if (rxPCData == 'o') {
                    controlSystemOn = !controlSystemOn;
                    if (controlSystemOn) {
                        pc.printf("\r\n\ncontrol system turned on\r\n\n");
                    }
                    else {
                        pc.printf("\r\n\ncontrol system turned off\r\n\n");
```

```
                    }
                }

                else if (rxPCData == 't') {
                    pc.printf("\r\nThis feature is not yet enabled, but
                        data logger has been turned off\r\n\n");
                    dataLoggingOn = 0;
                }

                else if (rxPCData == 'p') {
                    pc.printf("\r\nEnter kP value: ");
                    pc.scanf ("%f",&kP);
                    pc.printf("You entered: %.4f\r\n",kP);
                }

                else if (rxPCData == 'i') {
                    pc.printf("\r\nEnter kI value: ");
                    pc.scanf ("%f",&kI);
                    pc.printf("You entered: %.4f\r\n",kI);
                }

                else if (rxPCData == 'd') {
                    pc.printf("\r\nEnter kD value: ");
                    pc.scanf ("%f",&kD);
                    pc.printf("You entered: %.4f\r\n",kD);
                }

                else {
                    pc.printf("\r\n\ncommand not understood - try
                        ?\r\n\n");
                }
            }
            return;
}

// respond to light switch flipped on
void turnLightsOn() {
            wait_ms(25);
            if (lightsSwitch == 1) {
                lightsLED = 1;
                lightsOn = 1;
            }
            return;
}

// respond to light switch flipped off
void turnLightsOff() {
            wait_ms(25);
            if (lightsSwitch == 0) {
                lightsLED = 0;
                lightsOn = 0;
            }
            return;
}
```

```
// respond to control system switch flipped on
void turnControlSystemOn() {
            wait_ms(25);
            if (controlSystemSwitch == 1) {
                controlSystemLED = 1;
                controlSystemOn = 1;
            }
            return;
}

// respond to control system switch flipped off
void turnControlSystemOff() {
            wait_ms(25);
            if (controlSystemSwitch == 0) {
                controlSystemLED = 0;
                controlSystemOn = 0;
            }
            return;
}

// respond to switch 3 flipped on
void turnSwitch3On() {
            wait_ms(25);
            if (switch3 == 1) {
                switch3LED = 1;
                switch3On = 1;
                dataLoggingOn = 1;
            }
            return;
}

// respond to switch 3 flipped off
void turnSwitch3Off() {
            wait_ms(25);
            if(switch3 == 0) {
                switch3LED = 0;
                switch3On = 0;
                dataLoggingOn = 0;
            }
            return;
}

// read joysticks
void read_joysticks(){
            // each joystick axis is averaged over 3 samples to eliminate
            // any ADC glitches.
            if ((leftJoystickUpDownAnalog +
                  leftJoystickUpDownAnalog +
                  leftJoystickUpDownAnalog) / 3 > 0.75) {
                    lower = 32;       // 32 = space char, represents not on
                    raise = 94;       // 94 = ^ char, represents on
            }
            else if ((leftJoystickUpDownAnalog +
                        leftJoystickUpDownAnalog +
                        leftJoystickUpDownAnalog) / 3 < 0.25) {
                              raise = 32;
                              lower = 118;   // 118 = v char, represents on
            }
            else {
                    raise = 32;
                    lower = 32;
```

```
            }

            if ((leftJoystickLeftRightAnalog +
                  leftJoystickLeftRightAnalog +
                  leftJoystickLeftRightAnalog )/ 3 < 0.25) {
                    moveLeft = 32;
                    moveRight = 126;// 126 = right arrow char represents on
            }
            else if ((leftJoystickLeftRightAnalog +
                  leftJoystickLeftRightAnalog +
                  leftJoystickLeftRightAnalog ) / 3 > 0.75) {
                    moveRight= 32;
                    moveLeft = 127;  // 127 = left arrow char represents on
            }
            else {
                    moveLeft = 32;
                    moveRight = 32;
            }

            if ((rightJoystickUpDownAnalog +
                  rightJoystickUpDownAnalog +
                  rightJoystickUpDownAnalog) / 3 > 0.75) {
                    reverse = 32;   // 32 = space char, represents not on
                    forward = 94;   // 94 = ^ char, represents on
            }
            else if ((rightJoystickUpDownAnalog +
                        rightJoystickUpDownAnalog +
                        rightJoystickUpDownAnalog) / 3  < 0.25) {
                              forward = 32;
                              reverse = 118; // 118 = v char, represents on
            }
            else {
                    forward = 32;
                    reverse = 32;
            }

            if ((rightJoystickLeftRightAnalog +
                  rightJoystickLeftRightAnalog +
                  rightJoystickLeftRightAnalog) / 3 < 0.25) {
                    turnLeft = 32;
                    turnRight = 126;// 126 = right arrow char represents on
            }
            else if ((rightJoystickLeftRightAnalog +
                  rightJoystickLeftRightAnalog +
                  rightJoystickLeftRightAnalog) / 3  > 0.75) {
                    turnRight= 32;
                    turnLeft = 127; // 127 = left arrow char, represents on
            }
            else {
                    turnLeft = 32;
                    turnRight = 32;
            }
            return;
}

// output data to lcd screen
void display_data() {

            lcd.locate(1,1); lcd.printf("L");       // left joystick
            lcd.locate(0,1); lcd.putc(moveLeft);
            lcd.locate(2,1); lcd.putc(moveRight);
```

```
            lcd.locate(1,0); lcd.putc(raise);
            lcd.locate(1,2); lcd.putc(lower);

            lcd.locate(4,1); lcd.printf("R");       // right joystick
            lcd.locate(3,1); lcd.putc(turnLeft);
            lcd.locate(5,1); lcd.putc(turnRight);
            lcd.locate(4,0); lcd.putc(forward);
            lcd.locate(4,2); lcd.putc(reverse);

            lcd.locate(6,1); lcd.printf("P");       // PID settings
            lcd.locate(7,1); lcd.printf("%2.1f",kP);
            lcd.locate(6,2); lcd.printf("I");
            lcd.locate(7,2); lcd.printf("%2.1f",kI);
            lcd.locate(6,3); lcd.printf("D");
            lcd.locate(7,3); lcd.printf("%2.1f",kD);

            lcd.locate(11,2); lcd.printf("P");     // pitch tilt level
            lcd.locate(12,2); lcd.printf("%3i",rxData[2] - 90);
            lcd.locate(11,3); lcd.printf("R");     // roll tilt level
            lcd.locate(12,3); lcd.printf("%3i",rxData[3] - 90);

            lcd.locate(16,2); lcd.printf("T");     // top temp
            lcd.locate(17,2); lcd.printf("%2.0f",topTemp);
            lcd.locate(19,2); lcd.putc(223);       // degree symbol

            lcd.locate(16,3); lcd.printf("B");     // bottom temp
            lcd.locate(17,3); lcd.printf("%2.0i",rxData[1]);
            lcd.locate(19,3); lcd.putc(223);

            lcd.locate(6,0); if ((rxData[0] & 135) == 135) {
                                 lcd.printf("LEAK");
                                 buzzer.pulsewidth_us(250);
                                 leakLED = 1;
                             }

                             else {
                                 lcd.printf("    ");
                                 buzzer.pulsewidth_us(0);
                                 leakLED = 0;
                             }

            lcd.locate(11,0); if ((rxData[0] & 184) == 184) {
                                 lcd.printf("TEMP");
                              }

                              else {
                                 lcd.printf("    ");
                              }

            lcd.locate(0,3); if(lightsOn) {
                                 lcd.putc('L');
                             }
                             else {
                                 lcd.putc(32);
                             }

            lcd.locate(1,3); if(controlSystemOn) {
                                 lcd.putc('C');
                             }
                             else {
                                 lcd.putc(32);
```

```
                                }
            lcd.locate(2,3); if(dataLoggingOn) {
                                     lcd.putc('D');
                                }
                                else {
                                     lcd.putc(32);
                                }
            return;
}

void send_and_receive_data(){
            linkLED = !linkLED;

            // build tx data byte controlData[0]
            // bit 0: front right thruster, on = 1
            //      1: front left thruster, on = 1
            //      2: rear right thruster, on = 1
            //      3: rear left thruster, on = 1
            //      4: lights, on = 1
            //      5: control system, on = 1
            //      6: data logging, on = 1
            //      7: reserved, set to 1, signifies byte controlData[0]

            controlData[0] = 128;                          // set bit 7

            if (forward == 94) {
                    controlData[0] = controlData[0] + 3; // set bits 0 & 1
            }

            if (reverse == 118) {
                    controlData[0] = controlData[0] + 12;// set bits 2 & 3
            }

            if (turnRight == 126) {
                    controlData[0] = controlData[0] + 6; // set bits 1 & 2
            }

            if (turnLeft == 127) {
                    controlData[0] = controlData[0] + 9; // set bits 0 & 3
            }

            if (moveRight == 126) {
                    controlData[0] = controlData[0] + 10;// set bits 1 & 3
            }

            if (moveLeft == 127) {
                    controlData[0] = controlData[0] + 5; // set bits 0 & 2
            }

            if (lightsOn == 1) {
                    controlData[0] = controlData[0] + 16;// set bit 4
            }

            if (controlSystemOn == 1) {
                    controlData[0] = controlData[0] + 32;// set bit 5
            }

            if (dataLoggingOn ==1) {
                    controlData[0] = controlData[0] + 64;// set bit 6
            }
```

```
            // build tx data byte controlData[1]
            // bit 0: go up = 1
            //     1: go down = 1
            //     2: unused, set to 0
            //     3: unused, set to 0
            //     4: unused, set to 0
            //     5: unused, set to 0
            //     6: unused, set to 0
            //     7: reserved, set to 0, signifies byte controlData[1]

            controlData[1] = 0;

            if (raise == 94) {
                    controlData[1] = 1;      // this sets bit 0 only
            }

            if (lower == 118) {
                    controlData[1] = 2;      // this sets bit 1 only
            }


            // send data via serial connection
            if(topSerial.writeable()) {
                mbedLED1=!mbedLED1;
                topSerial.putc(controlData[0]);
                topSerial.putc(controlData[1]);

                // send PID constant float values as 4 x 8 bit chars
                topSerial.putc(kP8[0]);
                topSerial.putc(kP8[1]);
                topSerial.putc(kP8[2]);
                topSerial.putc(kP8[3]);

                topSerial.putc(kI8[0]);
                topSerial.putc(kI8[1]);
                topSerial.putc(kI8[2]);
                topSerial.putc(kI8[3]);

                topSerial.putc(kD8[0]);
                topSerial.putc(kD8[1]);
                topSerial.putc(kD8[2]);
                topSerial.putc(kD8[3]);
            }

            // receive data via serial connection
            if(topSerial.readable()) {
                rxData[0] = topSerial.getc();
                rxData[1] = topSerial.getc();
                rxData[2] = topSerial.getc();
                rxData[3] = topSerial.getc();

            }
            return;
}

// read temp sensor called by ticker
void read_temp_sensor(){
            // average of 3 readings: single reading * 3.3V * 100
            topTemp = ((tempTop + tempTop + tempTop) * 110) - 273;
            return;
}
```

```
// major events called by ticker
void majorEventFunctions () {
            read_joysticks();
            display_data();
            send_and_receive_data();
            return;
}

// MAIN
int main() {
            // startup display
            leakLED = 1;
            lcd.cls();
            lcd.printf("     SCOTTROV \n\n");
            lcd.printf("Top: v0.70  24.4.12\n");
            lcd.printf("Bot: v0.70");
            wait(3);
            lcd.cls();
            leakLED = 0;

            // set up serial comms to bottom
            topSerial.baud(38400);

            // set up comms with pc to trigger by interrupt on data arrival
            pc.attach(&SendAndReceiveDataFromPC, Serial::RxIrq);

            // set up buzzer PWM
            buzzer.period_us(500);                  // 500uS for 2kHz tone

            // set up ticker events
            tempSensor.attach(&read_temp_sensor, 3.0); // read every 3s
            majorEvent.attach(&majorEventFunctions, 0.05); // 20 times / s

            // set up interrupts for control panel switches
            lightsSwitch.rise(&turnLightsOn);
            lightsSwitch.fall(&turnLightsOff);

            controlSystemSwitch.rise(&turnControlSystemOn);
            controlSystemSwitch.fall(&turnControlSystemOff);

            switch3.rise(&turnSwitch3On);
            switch3.fall(&turnSwitch3Off);

            // determine initial switch state
            if (lightsSwitch == 1) {
                lightsLED = 1;
                lightsOn = 1;
            }
            else {
                lightsLED = 0;
                lightsOn = 0;
            }

            if (controlSystemSwitch == 1) {
                controlSystemLED = 1;
                controlSystemOn = 1;
            }
            else {
                controlSystemLED = 0;
                controlSystemOn = 0;
            }
```

```
            if (switch3 == 1) {
                switch3LED = 1;
                switch3On = 1;
            }
            else {
                switch3LED = 0;
                switch3On = 0;
            }

            while(1) {

            }// end while

}// end of MAIN
```

Appendix E - Bottom Controller Code

```
// BOTTOM controller for ROV
//  v0.70  26 April 2012
//  by Scott O'Brien

// LIBRARIES
            #include "mbed.h"
            #include "LIS331.h"          // accelerometer library
            #include "SDFileSystem.h"    // SD card library for data logging
            #include "PID.h"             // PID controller library

// COMPILER DEFINITIONS
            #define PI 3.14159265

// PIN DEFINITIONS
            Serial pc(USBTX, USBRX);      // diagnostic conn. to PC via USB
            Serial bottomSerial(p13,p14);// tx, rx to top cont. via MAX3232

            DigitalOut mbedLED1(LED1);    // onboard led's
            DigitalOut mbedLED2(LED2);
            DigitalOut mbedLED3(LED3);
            DigitalOut mbedLED4(LED4);
            DigitalOut lights(p12);       // external lights

            SDFileSystem sd(p5, p6, p7, p8, "sd"); // data logging
            LIS331 accel(p9, p10);        // I2C conn. to accelerometer

            DigitalOut motorHFL(p27);     // front left horizontal thruster
            DigitalOut motorHFR(p26);     // front right horizontal thruster
            DigitalOut motorHRL(p25);     // rear left horizontal thruster
            DigitalOut motorHRR(p24);     // rear right horizontal thruster
            PwmOut motorVL(p23);          // left vertical thruster
            DigitalOut motorVLdir(p20);   // 0 = down, 1 = up
            PwmOut motorVR(p22);          // right vertical thruster
            DigitalOut motorVRdir(p19);   // 0 = down, 1 = up
            PwmOut motorVB(p21);          // back vertical thruster
            DigitalOut motorVBdir(p18);   // 0 = down, 1 = up

            InterruptIn thermalOverload(p17);   // thermal overload flag
            InterruptIn leakDetector(p11);      // leak detectors
            AnalogIn tempBottom(p15);           // temperature sensor

            PID pitchCon(1.0, 0.0, 0.0, 0.1);   // Kp, Ti, Td, interval
            PID  rollCon(1.0, 0.0, 0.0, 0.1);   // Kp, Ti, Td, interval

// TICKERS AND TIMERS
            Ticker readSensors;                 // enable sensor timer
            Ticker readTiltSensors;             // enable tilt sensor timer
            Timer  loggingTimer;                // timer for data logging

// VARIABLES AND CONSTANTS
            char  controlData[3]      = {0,0,0};// control data rx from top
            char  txData[5]           = {0,0,0,0}; // data tx to top
            char  controlSystemOn     = 0;
            char  thermalOverloadFlag = 0;
            float bottomTemp          = 0;
            float pitchTilt           = 0;
            float rollTilt            = 0;
            char  dataLoggingOn       = 0;
            float kP                  = 1.0;            // PID "constants"
```

```
            uint8_t kP8[5]              = {0,0,0,0,0};
            float kI                    = 0.0;
            uint8_t kI8[5]              = {0,0,0,0,0};
            float kD                    = 0.0;
            uint8_t kD8[5]              = {0,0,0,0,0};
            float kPID                  = 0;   // change in PID settings?

            // level of VB thruster to control pitch, from PID controller
            float pitchAdj              = 0;
            // level of VL/R thrusters to control roll, from PID controller
            float rollAdj               = 0;

// FUNCTIONS

// interrupt function for serial comms
void sendAndReceiveData() {
            mbedLED2 = !mbedLED2;
            if(bottomSerial.readable()) {
                controlData[0] = bottomSerial.getc();
                controlData[1] = bottomSerial.getc();

                kP8[0] = bottomSerial.getc();   // PID constants (float) as
                kP8[1] = bottomSerial.getc();   // 4 x 8 bit chars
                kP8[2] = bottomSerial.getc();
                kP8[3] = bottomSerial.getc();

                kI8[0] = bottomSerial.getc();
                kI8[1] = bottomSerial.getc();
                kI8[2] = bottomSerial.getc();
                kI8[3] = bottomSerial.getc();

                kD8[0] = bottomSerial.getc();
                kD8[1] = bottomSerial.getc();
                kD8[2] = bottomSerial.getc();
                kD8[3] = bottomSerial.getc();

                kP = *((float *)kP8);  // 4 x 8 chars back to single floats
                kI = *((float *)kI8);
                kD = *((float *)kD8);

                if ((kP + kD + kI) != kPID) {     // check for any changes
                    pitchCon.setTunings(kP,kI,kD);
                    rollCon.setTunings(kP,kI,kD);
                    kPID = kP + kD + kI;
                }
            }

            if(bottomSerial.writeable()) {
                bottomSerial.putc(txData[0]);
                bottomSerial.putc(txData[1]);
                bottomSerial.putc(txData[2]);
                bottomSerial.putc(txData[3]);
            }
            return;
}
```

```
//  interrupt function to respond to leak detector
void leakDetected() {
            txData[0] = (txData[0] | 135);        // set bits 0-2, 7
            return;
}

//  interrupt function to respond to thermal overload on h-bridges
void thermalOverloaded() {
            txData[0] = (txData[0] | 184);        // set bits 3-5, 7
            thermalOverloadFlag = 1;
            return;
}

// read temp & voltage sensors
void read_sensors(){
            bottomTemp = ((tempBottom + tempBottom + tempBottom) * 110.0) -
            273; // take average of 3 readings
            txData[1] = ((char)bottomTemp) & 127;
            return;
}

// read tilt sensors
void read_tilt_sensors(){
            pitchTilt = (accel.getAccelX() + accel.getAccelX() +
            accel.getAccelX() + accel.getAccelX() + accel.getAccelX()) / 5;

            rollTilt  = (accel.getAccelY() + accel.getAccelY() +
            accel.getAccelY() + accel.getAccelY() + accel.getAccelY()) / 5;

            if (controlSystemOn == 1) {

            // send current pitch tilt to PID controller as current error
                    pitchCon.setProcessValue(pitchTilt);
            // let PID controller do its thing
                    pitchAdj = pitchCon.compute();

            // send current roll tilt to PID controller as current error
                    rollCon.setProcessValue(rollTilt);
            // let PID controller do its thing
                    rollAdj = rollCon.compute();
            }

            else {
                    pitchAdj = 0;
                    rollAdj  = 0;
            }
            // conv to degs, +90 to ensure +ve number to enable TX as char
            txData[2] = (asin(pitchTilt) * 180 / PI) + 90;
            txData[3] = (asin(rollTilt) * 180 / PI) + 90;
            return;
}
```

```
// log data
void log_data(){
            FILE *fp = fopen("/sd/data.txt", "a");
            mbedLED3 = !mbedLED3;
            fprintf(fp, "\n\r%f,%i,%f,%f,%f,%f,%f,%f,%f,%f,%i,%f,%i,%f,%i",
            loggingTimer.read(), controlSystemOn, kP, kI, kD, pitchTilt,
            pitchAdj, rollTilt, rollAdj, motorVL.read(), motorVLdir.read(),
            motorVR.read(), motorVRdir.read(), motorVB.read(),
            motorVBdir.read());
            fclose(fp);
            return;
}

// MAIN
int main() {

        // initialise timer for data logging
        loggingTimer.start();

        // initialise all motors
        motorHFL   = 0;
        motorHFR   = 0;
        motorHRL   = 0;
        motorHRR   = 0;
        motorVL    = 0.0;
        motorVLdir = 0;
        motorVR    = 0.0;
        motorVRdir = 0;
        motorVB    = 0.0;
        motorVBdir = 0;

        // init comms to top cont. to trigger by interrupt on data arrival
        bottomSerial.baud(38400);
        bottomSerial.attach(&sendAndReceiveData, Serial::RxIrq);

        // init interrupt for leak detector & thermal overload on h-bridges

        // attach address of the leakDetected function to the rising edge
        leakDetector.rise(&leakDetected);

        // attach address of thermalOverloaded function to the falling edge
        thermalOverload.fall(&thermalOverloaded);

        // set up ticker event for temp and voltage sensors
        readSensors.attach(&read_sensors, 3.0);     // read every 3 seconds

        // set up accelerometer
        accel.setFullScaleRange2g();          // 2g range on accel
        accel.setPowerMode(47);               // 100 Hz ODR
        readTiltSensors.attach(&read_tilt_sensors, 0.250); // every 0.25s

        // set up PWM for vertical thrusters
        int pwmPeriod = 100;                  // 100 microseconds = 10 kHz
        motorVL.period_us(pwmPeriod);
        motorVR.period_us(pwmPeriod);
        motorVB.period_us(pwmPeriod);

        // set up PID controller for pitch
        pitchCon.setInputLimits(-1.0, 1.0);   // Input from accelerometer
        pitchCon.setOutputLimits(-1.0, 1.0);  // PWM output limits
        pitchCon.setBias(0.0);          // Only if there needs to be a bias.
```

```
        pitchCon.setMode(AUTO_MODE);
        pitchCon.setSetPoint(0.0);      // Target is to be zero degrees

        // set up PID controller for roll
        rollCon.setInputLimits(-1.0, 1.0);    // Input from accelerometer
        rollCon.setOutputLimits(-0.5, 0.5);   // PWM output limits
        rollCon.setBias(0.0);           // Only if there needs to be a bias.
        rollCon.setMode(AUTO_MODE);
        rollCon.setSetPoint(0.0);       // Target is to be zero degrees

        while(1) {

                // rx data byte controlData[0]
                // bit 0: front right thruster, on = 1
                //     1: front left thruster, on = 1
                //     2: rear right thruster, on = 1
                //     3: rear left thruster, on = 1
                //     4: lights, on = 1
                //     5: control system, on = 1
                //     6: data logging, on = 1
                //     7: reserved, set to 1, signifies byte controlData[0]

                // rx data byte controlData[1]
                // bit 0: go up = 1
                //     1: go down = 1
                //     2: unused, set to 0
                //     3: unused, set to 0
                //     4: unused, set to 0
                //     5: unused, set to 0
                //     6: unused, set to 0
                //     7: reserved, set to 0, signifies byte controlData[1]

                if ((controlData[0] & 131) == 131) {          // go forward
                    motorHFL = 1; motorHFR = 1;
                    motorHRL = 0; motorHRR = 0;
                }

                else if ((controlData[0] & 140) == 140) {    // go backwards
                    motorHFL = 0; motorHFR = 0;
                    motorHRL = 1; motorHRR = 1;
                }

                else if ((controlData[0] & 134) == 134) {    // turn right
                    motorHFL = 1; motorHFR = 0;
                    motorHRL = 0; motorHRR = 1;
                }

                else if ((controlData[0] & 137) == 137) {    // turn left
                    motorHFL = 0; motorHFR = 1;
                    motorHRL = 1; motorHRR = 0;
                }

                else if ((controlData[0] & 138) == 138) {    // move right
                    motorHFL = 1; motorHFR = 0;
                    motorHRL = 1; motorHRR = 0;
                }

                else if ((controlData[0] & 133) == 133) {    // move left
                    motorHFL = 0; motorHFR = 1;
                    motorHRL = 0; motorHRR = 1;
                }
```

```
else {                                  // do nothing
    motorHFL = 0; motorHFR = 0;
    motorHRL = 0; motorHRR = 0;
}

if ((controlData[0] & 144) == 144) {  // lights on
    lights = 1;
}
    else {                              // lights off
    lights = 0;
}

if ((controlData[0] & 160) == 160) {  // control system on
    controlSystemOn = 1;
}
else {
    controlSystemOn = 0;                // control system off
    pitchAdj = 0;
    rollAdj=0;
}

if((controlData[0] & 64) == 64) {     // data logging on
    dataLoggingOn = 1;
    log_data();
}
else {
    dataLoggingOn = 0;                  // data logging off
}

if (((controlData[1] & 1) == 1) &&
        (thermalOverloadFlag == 0)) {       // go up
            // turn on 50%, dir: 0 = down, 1 = up
            motorVL = 0.5 + (rollAdj/2); motorVLdir = 1;
            // turn on 50%, dir: 1 = down, 0 = up
        motorVR = 0.5 - (rollAdj/2); motorVRdir = 0;
}

else if (((controlData[1] & 2) == 2) &&
        (thermalOverloadFlag == 0)) {       // go down
            // turn on 50%, dir: 0 = down, 1 = up
            motorVL = 0.5 + (rollAdj/2); motorVLdir = 0;
            // turn on 50%, dir: 1 = down, 0 = up
            motorVR = 0.5 - (rollAdj/2); motorVRdir = 1;
}

else {  // don't drive up or down, turn off all vertical
        // thrusters except for control system adjustments
    if (rollAdj < 0) {
        motorVL = abs(rollAdj)/2; motorVLdir = 0;
        motorVR = abs(rollAdj)/2; motorVRdir = 0;
    }
    else {
        motorVL = rollAdj/2; motorVLdir = 1;
        motorVR = rollAdj/2; motorVRdir = 1;
    }

    motorVB = abs(pitchAdj);    // control system adjustment
    if (pitchAdj < 0) {         // determine rotation
        motorVBdir = 1;
    }
```

```
                    else {
                        motorVBdir = 0;
                    }

                }

        }// end of while
} // end of main
```

Appendix F - Data Logging Code

Bottom Pitch Data Logger

```
// BOTTOM controller PITCH DATA LOGGER for ROV
//  v0.30
//  18 April 2012
//  by Scott O'Brien

// LIBRARIES
            #include "mbed.h"
            #include "LIS331.h"          // accelerometer library
            #include "SDFileSystem.h"    // SD card library for data logging

// COMPILER DEFINITIONS
            #define PI 3.14159265

// PIN DEFINITIONS
            DigitalOut mbedLED1(LED1);  // onboard led's
            DigitalOut mbedLED2(LED2);
            DigitalOut mbedLED3(LED3);
            DigitalOut mbedLED4(LED4);

            SDFileSystem sd(p5, p6, p7, p8, "sd"); // for data logging
            LIS331 accel(p9, p10);       // I2C connection to accelerometer

            DigitalOut motorHFL(p27);    // front left horizontal thruster
            DigitalOut motorHFR(p26);    // front right horizontal thruster
            DigitalOut motorHRL(p25);    // rear left horizontal thruster
            DigitalOut motorHRR(p24);    // rear right horizontal thruster
            PwmOut motorVL(p23);         // left vertical thruster
            DigitalOut motorVLdir(p20); // 0 = down, 1 = up
            PwmOut motorVR(p22);         // right vertical thruster
            DigitalOut motorVRdir(p19); // 0 = down, 1 = up
            PwmOut motorVB(p21);         // back vertical thruster
            DigitalOut motorVBdir(p18); // 0 = down, 1 = up

// TICKERS AND TIMERS
            Timer  loggingTimer;         // timer for data logging
            Ticker change_VB_PWM_ticker;// change PID settings every 15s

// VARIABLES AND CONSTANTS
            float pitchTilt           = 0;
            float VBLevel             = 0;// amount to apply to VB thruster
            char  VB_incrementer_dir  = 0;// increment (0) or decrement (1)
            char  passes              = 0;// number of logging cycles
            FILE  *fp = fopen("/sd/data.txt", "a");

// FUNCTIONS

// read tilt sensors
void read_tilt_sensors(){

            // get samples and take the MODE
            int n = 7;
            float sample[7] = {0,0,0,0,0,0,0};
            for (int i = 0 ; i < n ; i++) {         // get samples
                    sample[i] = accel.getAccelX();
            }
```

```
            for(int i = 0 ; i < n; i++) {           // sort the samples
                    for(int j = 0 ; j < n - 1 ; j++) {
                            if(sample[j] > sample[j + 1]) {
                                    float temp = sample[j + 1];
                                    sample[j + 1] = sample[j];
                                    sample[j] = temp;
                            }
                    }
            }

            pitchTilt = sample[3];// use value from the middle of the array

            return;
}

// log data
void log_data(){

            if (fp != NULL) {
                FILE *fp = fopen("/sd/data.txt", "a");
            }
            mbedLED3 = !mbedLED3;
            fprintf(fp, "\n\r%f,%f,%f,%i",  loggingTimer.read(), pitchTilt,
motorVB.read(), motorVBdir.read());
            fclose(fp);
            return;

}

// change VB PWM levels every 5 seconds
void change_VB_PWM(){

            if (VBLevel >= 1.0) {       // time to head down
                VB_incrementer_dir = 1;
            }

            if (VBLevel <= -1.0) {       // time to head up
                VB_incrementer_dir = 0;
                passes = passes + 1;
            }

            if (VB_incrementer_dir == 0) {
                VBLevel = VBLevel + 0.05; // increment PWM by 5% every step
            }
            else {
                VBLevel = VBLevel - 0.05; // decrement PWM by 5% every step
            }

            mbedLED2 = !mbedLED2;
            return;
}
```

```
// MAIN
int main() {

            wait(30);  // time to get in the water

            // initialise timer for data logging
            loggingTimer.start();

            // set up ticker event to change PWM settings every 5 seconds
            change_VB_PWM_ticker.attach(&change_VB_PWM, 5.0);

            // initialise all motors
            motorHFL   = 0;
            motorHFR   = 0;
            motorHRL   = 0;
            motorHRR   = 0;
            motorVL    = 0.0;
            motorVLdir = 0;
            motorVR    = 0.0;
            motorVRdir = 0;
            motorVB    = 0.0;
            motorVBdir = 0;

            // set up accelerometer
            accel.setFullScaleRange2g();        // 2g range on accel
            accel.setPowerMode(47);             // 100 Hz ODR : 0d47 = 0x2f

            // set up PWM for vertical thrusters
            int pwmPeriod = 100;                // 1 millisecond = 1000 Hz
            motorVL.period_us(pwmPeriod);       // 1 microsecond = 1 MHz
            motorVR.period_us(pwmPeriod);       // 100 microseconds = 10 kHz
            motorVB.period_us(pwmPeriod);


            mbedLED4 = 0;     // LED4 turns on when finished data logging

            while(1) {
                    read_tilt_sensors();

                    if (passes == 5) {           // logged enough data yet?
                        change_VB_PWM_ticker.detach();// disable PWM
                        mbedLED4 = 1;            // turn on "Done" LED4
                        VBLevel = 0;             // turn off VB thruster
                    }
                    else {
                        log_data();
                    }

                    motorVB = abs(VBLevel);      // apply variable
                    if (VBLevel < 0) {           // determine rotation
                        motorVBdir = 1;
                    }
                    else {
                        motorVBdir = 0;
                    }

            }// end of while
} // end of main
```

MATLAB

```
%%  PWM pitch data analysis
%
%   By Scott O'Brien
%   #12747212
%

%% Initialise
clear all; close all; clc;

%% Import data
rawData = importdata('data_pwm_pitch_logger_pool_22_april.txt');

%% Prep data
timeData         = rawData(:,1);
pitchTiltData    = rawData(:,2);
VBMotorLevelData = rawData(:,3);
VBMotorDirData   = rawData(:,4);

% convert accelerometer readings to degrees
pitchTiltAngle   = (asin(pitchTiltData) * 180 / pi);

% adjust for direction of rotation
for n = 1:length(VBMotorLevelData);
        if VBMotorDirData(n) == 0
                VBMotorLevelData(n) = VBMotorLevelData(n) * -1;
        end
end

%% Plot tilt and PWM level
figure(1);
hold on;

subplot(2,1,1), plot(timeData,pitchTiltAngle,'r');
legend('angle (degrees)'); title('Pitch Tilt');
xlabel('Time (s)'); ylabel('Tilt (degrees)');
grid on;

subplot(2,1,2),plot(timeData,VBMotorLevelData*100,'b');
legend('PWM (%)'); title('PWM Level');
xlabel('Time (s)'); ylabel('PWM (%)');
grid on;
```

```
%%  PID pitch data analysis
%
%   By Scott O'Brien
%   #12747212
%

%% Initialise
clear all; close all; clc;

%% Import data
rawData = importdata('data_pid_pool_12_April_2012.txt');

%% Prep data
timeData = rawData(:,1);
```

```
p_constant     = rawData(:,2);
i_constant     = rawData(:,3);
d_constant     = rawData(:,4);
pitchTiltData  = rawData(:,5);

% convert accelerometer readings to degrees
pitchTiltAngle = (asin(pitchTiltData) * 180 / pi);

%% Plot tilt and PID constants
figure(1);
hold on;

subplot(2,1,1), plot(timeData,pitchTiltAngle,'r');
legend('angle (degrees)'); title('Pitch Tilt');
xlabel('Time (s)'); ylabel('Tilt (degrees)');
grid on;

subplot(2,1,2),plot(timeData,p_constant,'b');
legend('P constant'); title('PID Level');
xlabel('Time (s)'); ylabel('P constant');
grid on;
```

Appendix G - IMU Schematic and PCB Design

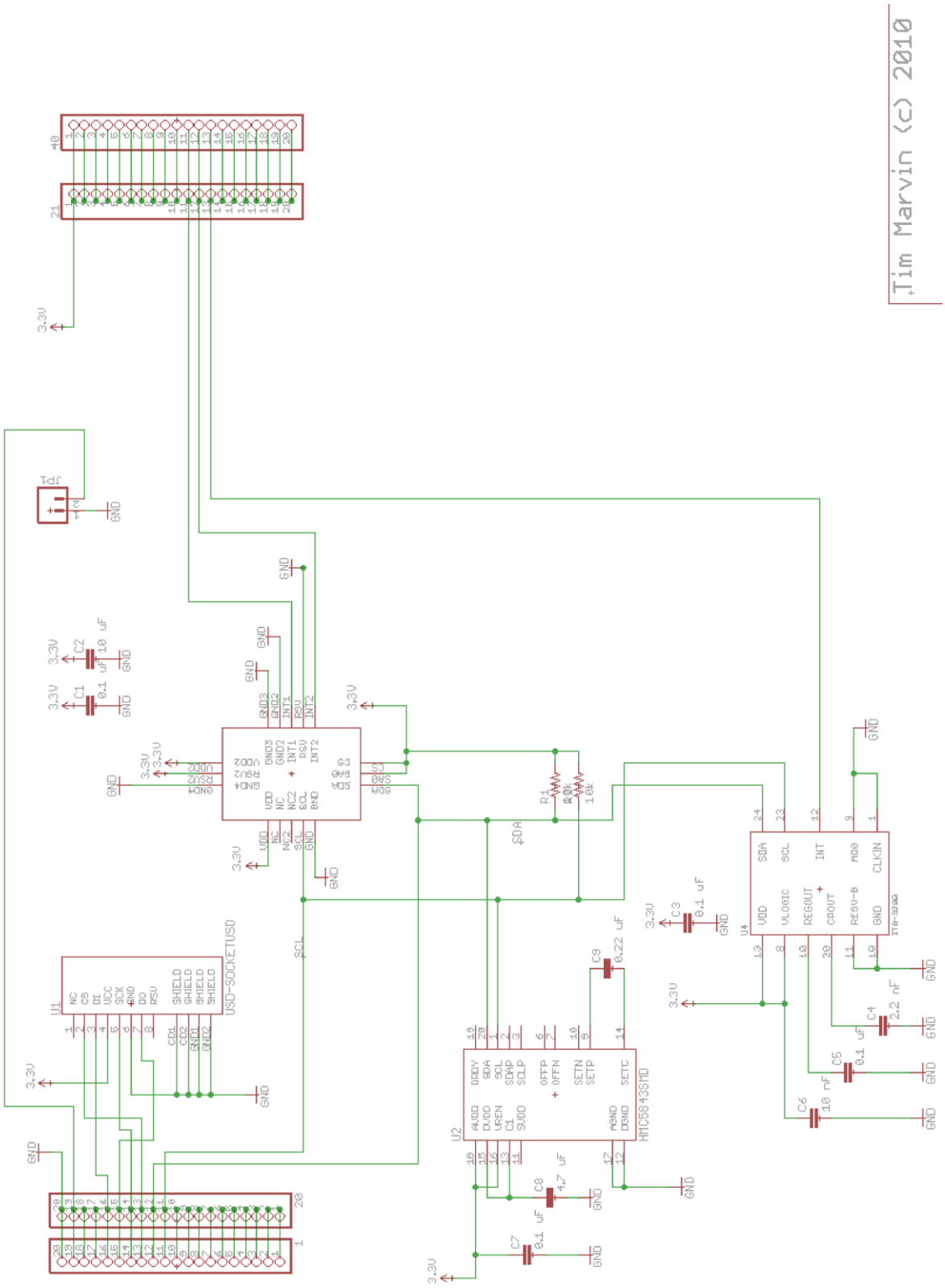


Figure G.1. Circuit schematic for Tim Marvin's IMU board.

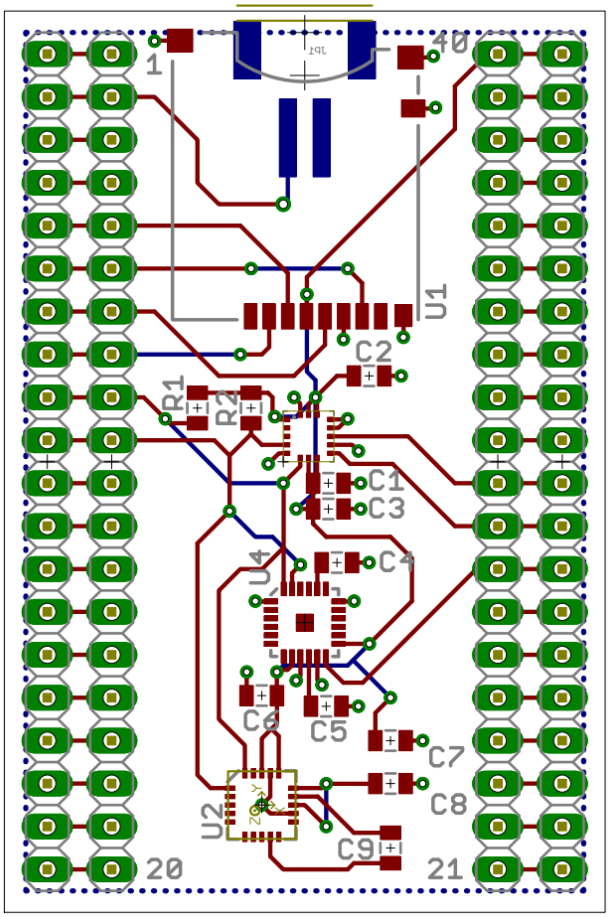


Figure G.2. PCB design for Tim Marvin's IMU board.

Appendix H - Definitions of Motion

There are six different motions a vessel in or on water can experience:

- Yaw
- Pitch
- Roll
- Sway
- Surge
- Heave

These are shown in Figure H.1. Sway, surge and heave are positional movements ***along*** the three axes, whereas yaw, pitch and roll are rotational movements ***around*** those three axes.

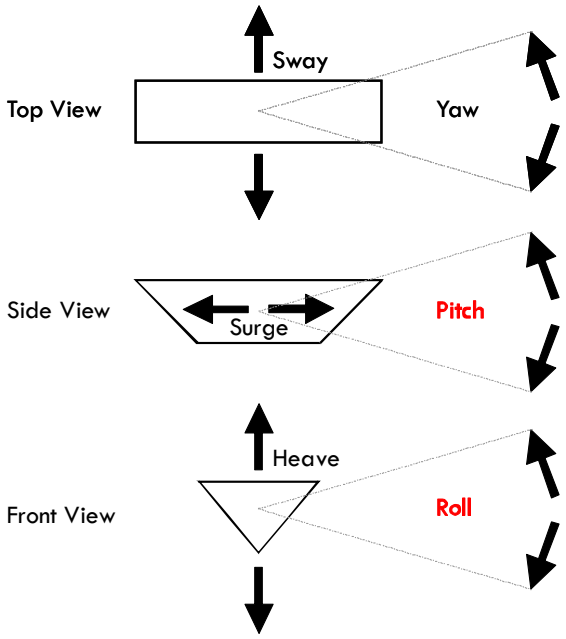


Figure H.1. Water-based vehicular motions.

Appendix I - Accelerometer Noise Analysis

During initial testing of the LIS331DLH accelerometer, some unexpected glitches were observed in the raw output data.

15 seconds of data readings were logged whilst the accelerometer was sitting motionless on a relatively level table. As can be seen in Figure I.1, most of the time the reading is -0.000061 which is near enough to level, but every now and then a value of -0.015747 occurs. It was determined that it was not anything mechanical on or around the table causing this, as the number never varies from these two exact figures – external mechanical disturbances show up as varying values.

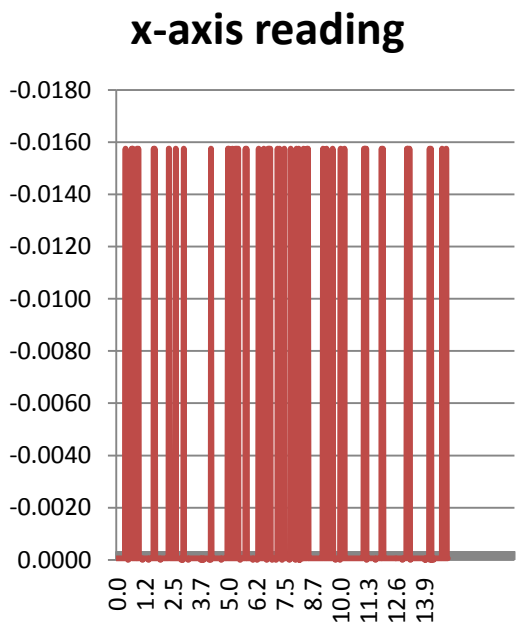


Figure I.1. Logged data generated at an output data rate (ODR) of 1000 Hz.

These glitches were some sort of noise, representing 0.9 degrees of tilt [22]:

$$Tilt\ (degrees)$$

$$= \frac{arcsin(accelerometer\ value)\ \times\ 180}{\pi}$$

$$= \frac{\arcsin(0.015747)\times 180}{\pi} = 0.9°$$

Further investigation seemed warranted. The power supply and the USB communications were considered and eliminated as possible sources of this noise, and a low pass filter, using an averaging over 10 samples, was used to try to eliminate it, but this did not eliminate the problem.

A frequency analysis of the data in Matlab was performed to look for clues (Figure I.2). It was observed that the unwanted noise bears a strong visual resemblance to the MATLAB uniform noise function. Figure I.3 shows the plot of data generated by the ***rand()*** function and the similarity is obvious.

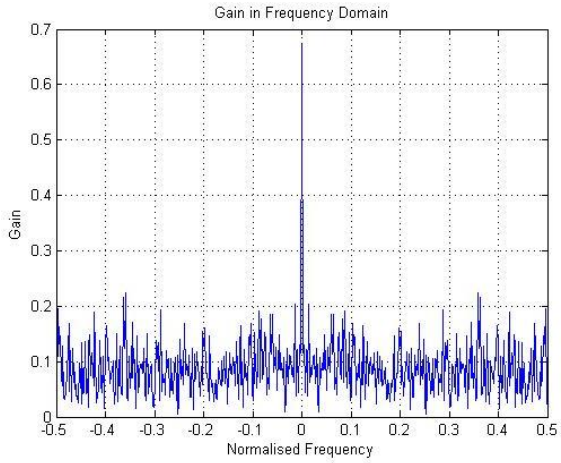


Figure I.2. Signal and noise generated at an output data rate (ODR) of 1000 Hz.

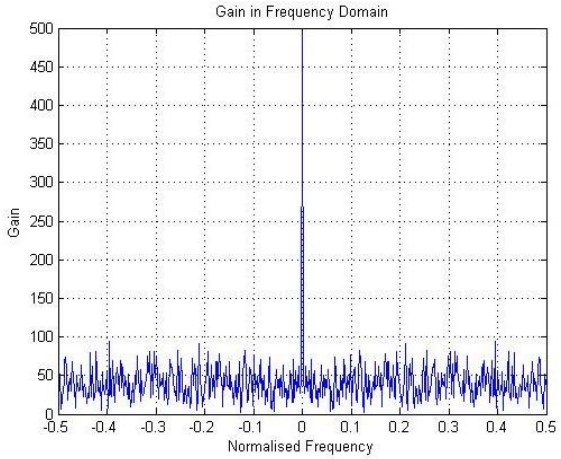


Figure I.3. Uniform noise generated by the rand() function.

After further research, it was found that noise is common and inherent in these forms of incredibly sensitive MEM's based devices. The manufacturers recommended solution is to lower the default output data rate (ODR) from 1000 Hz to 100 Hz.

Further testing, including logging more data and performing frequency analyses shows that reducing the ODR does in fact lower the noise floor and reduces the number of spurious samples received. This is shown in Figures I.4 and I.5. By comparing the vertical axes of Figures I.2 and I.5 it can be seen there is a reduction in the average noise levels from approximately 0.1 down to approximately 0.03.

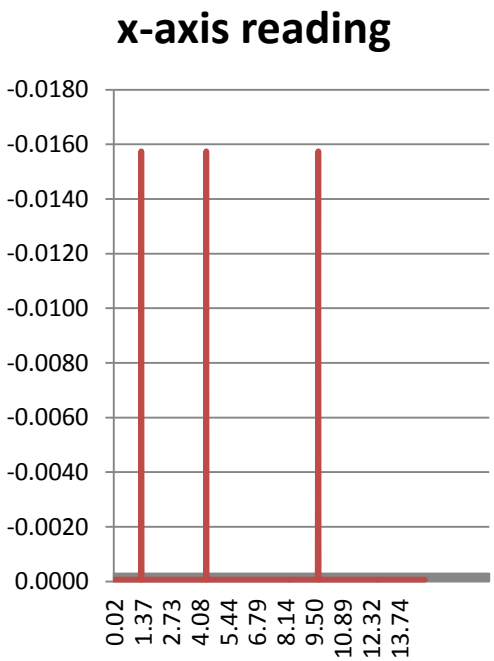


Figure I.4. Logged data generated at an output data rate (ODR) of 100 Hz.

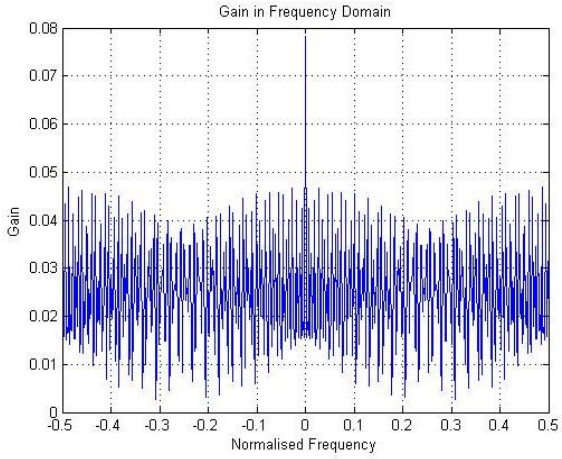


Figure I.5. Signal and noise generated at an output data rate (ODR) of 100 Hz.

As the spurious samples are now received very infrequently, simply using the mode of 5 or 7 samples should be enough, in practice, to remove any problem they cause.

The raw data, Excel and MATLAB files can be found on the accompanying CD.

Appendix J - Bill of Materials

Bill of Materials: Top Controller			All prices include VAT	
Item	**Quantity**	**Price each**	**Total**	**Supplier**
mbed	1	£ 48.88	£ 48.88	Donated by ARM
joystick	2	£ 2.45	£ 4.90	Proto-Pic
joystick breakout board	2	£ 1.50	£ 3.00	Proto-Pic
LCD screen	1	£ 15.60	£ 15.60	Farnell
large stand-offs	8	£ 0.36	£ 2.88	Proto-Pic
small standoffs	8	£ 0.15	£ 1.20	Proto-Pic
bolts	32	£ 0.03	£ 0.96	Proto-Pic
washers	32	£ 0.02	£ 0.64	Proto-Pic
3 junction screw terminal	2	£ 0.25	£ 0.50	Farnell
MAX3232 IC	1	£ 1.84	£ 1.84	Donated by Exar
2x5 way IDC plug	1	£ 1.09	£ 1.09	Maplin
2x5 way IDC socket	1	£ 1.14	£ 1.14	Maplin
2x8 way IDC plug	1	£ 1.19	£ 1.19	Maplin
2x8 way IDC socket	1	£ 1.29	£ 1.29	Maplin
20 pin header strip	2	£ 0.59	£ 1.18	Farnell
16 pin DIL socket	1	£ 0.15	£ 0.15	Farnell
switch	4	£ 1.20	£ 4.80	Farnell
green LED	5	£ 0.20	£ 1.00	Farnell
red LED	1	£ 0.19	£ 0.19	Farnell
enclosure	1	£ 3.79	£ 3.79	Maplin
DB9 female connector	1	£ 1.49	£ 1.49	Maplin
piezo buzzer	1	£ 0.95	£ 0.95	Proto-Pic
7805 voltage regulator	1	£ 0.99	£ 0.99	Maplin
strip-board	1	£ 3.40	£ 3.40	Farnell
9 V battery connector	1	£ 1.09	£ 1.09	Maplin
9 V battery	1	£ 3.00	£ 3.00	Morrisons
1N4001 diode	1	£ 0.15	£ 0.15	Proto-Pic
100 □F capacitor	1	£ 0.24	£ 0.24	Farnell
10 □F capacitor	1	£ 0.20	£ 0.20	Farnell
0.1 □F capacitor	5	£ 0.15	£ 0.75	Farnell
150 □ resistor	7	£ 0.10	£ 0.70	Farnell
27 □ resistor	1	£ 0.10	£ 0.10	Farnell
6 □ resistor	1	£ 0.10	£ 0.10	Farnell
LM335 temperature sensor	1	£ 0.85	£ 0.85	Farnell
small heatsink	1	£ 0.48	£ 0.48	Farnell
heatsink mounting kit	1	£ 0.15	£ 0.15	Farnell
rca jack	1	£ 0.59	£ 0.59	Proto-Pic
acrylic	1 piece	£ -	£ -	Donated by AbPlas
wire	various	£ -	£ -	Maplin
ribbon cable	150 mm	£ -	£ -	Maplin
		Total	**£ 111.45**	

Bill of Materials: Bottom Controller				All prices include VAT
Item	**Quantity**	**Price each**	**Total**	**Supplier**
mbed	1	£ 48.88	£ 48.88	Donated by ARM
IMU board	1	£ 90.00	£ 90.00	Tim Marvin
bolts	4	£ 0.07	£ 0.28	Proto-Pic
nuts	12	£ 0.03	£ 0.36	Proto-Pic
washers	4	£ 0.02	£ 0.08	Proto-Pic
3 junction screw terminal	5	£ 0.25	£ 1.25	Farnell
2 junction screw terminal	4	£ 0.19	£ 0.76	Farnell
2.2 Ah Turnigy LIPO battery	1	£ 10.00	£ 10.00	HobbyKing
MAX3232 IC	1	£ 1.84	£ 1.84	Donated by Exar
20 pin header strip	4	£ 0.59	£ 2.36	Farnell
7805 voltage regulator	1	£ 0.99	£ 0.99	Maplin
strip-board	1	£ 3.40	£ 3.40	Farnell
fuse	1	£ 0.19	£ 0.19	Farnell
fuse holder	1	£ 0.13	£ 0.13	Farnell
200 □F capacitor	3	£ 0.26	£ 0.78	Farnell
100 □F capacitor	4	£ 0.24	£ 0.96	Farnell
10 □F capacitor	1	£ 0.20	£ 0.20	Farnell
0.1 □F capacitor	5	£ 0.15	£ 0.75	Farnell
10 nF capacitor	1	£ 0.09	£ 0.09	Farnell
10 K□ resistor	8	£ 0.10	£ 0.80	Farnell
680 □ resistor	1	£ 0.10	£ 0.10	Farnell
150 □ resistor	1	£ 0.10	£ 0.10	Farnell
LM335 temperature sensor	1	£ 0.85	£ 0.85	Farnell
1N4001 diode	6	£ 0.15	£ 0.90	Proto-Pic
RFP30N06LE MOSFET	5	£ 1.12	£ 5.60	Proto-Pic
LMD18200 H-bridge IC	3	£ 17.95	£ 53.85	Farnell
LMD18200 breakout board	3	£ 1.20	£ 3.60	Proto-Pic
small heatsink	6	£ 0.48	£ 2.88	Farnell
large heatsink	3	£ 0.60	£ 1.80	Farnell
heatsink mounting kits	9	£ 0.15	£ 1.35	Farnell
heatsink compound	1	£ 3.69	£ 3.69	Maplin
switch	1	£ 1.79	£ 1.79	Maplin
battery connector	1	£ 0.40	£ 0.40	HobbyKing
terminal block	1	£ 1.28	£ 1.28	B & Q
video camera	1	£ 30.06	£ 30.06	Proto-Pic
microSD card	1	£ 15.99	£ 15.99	Amazon
cable ties	numerous	£ -	£ -	B & Q
heatshrink	various	£ -	£ -	Farnell
wire	various	£ -	£ -	Maplin
		Total	**£ 288.34**	

Bill of Materials: Hardware				All prices include VAT
Item	**Quantity**	**Price each**	**Total**	**Supplier**
piping	4	£ 1.14	£ 4.56	B & Q
corner junction pipe	8	£ 1.09	£ 8.72	B & Q
small T-junction pipe	6	£ 1.30	£ 7.80	B & Q
small c-clamp	7	£ 0.34	£ 2.38	B & Q
large c-clamp	2	£ 0.51	£ 1.02	B & Q
large T-junction	1	£ 12.00	£ 12.00	B & Q
screw end-cap	2	£ 8.99	£ 17.98	B & Q
push-fit end-cap	1	£ 3.99	£ 3.99	B & Q
junction box	1	£ 4.56	£ 4.56	B & Q
screw terminal block	2	£ 1.28	£ 2.56	B & Q
Rule 500 bilge pump	7	£ 12.50	£ 87.50	Borough Bridge Marina
Graupner 5-blade propeller	7	£ 4.77	£ 33.39	gliders.uk.com
screws	14	£ 0.03	£ 0.42	B & Q
nuts	4	£ 0.03	£ 0.12	B & Q
bolts	4	£ 0.07	£ 0.28	B & Q
30 m Ethernet cable	1	£ 10.68	£ 10.68	scan.co.uk
DB9 male connector	1	£ 4.10	£ 4.10	Maplin
cable braid	2	£ 1.91	£ 3.82	Farnell
18-core cable	0.5 m	£ 1.62	£ 1.62	Farnell
18 conductor plug and socket	1	£ 36.77	£ 36.77	Northern Connectors
cable grommet	1	£ 3.49	£ 3.49	Maplin
cable holder	1	£ 3.98	£ 3.98	B & Q
polyurethane resin	1	£ 3.15	£ 3.15	Farnell
silicone grease	1	£ 4.49	£ 4.49	Maplin
propeller shaft adapter	7	£ -	£ -	Donated by U.o.W
acrylic	1 piece	£ -	£ -	Donated by AbPlas
plastic coated wire mesh	400 mm x 300 mm	£ -	£ -	Donated by Jack Bowles
wood-based flotation device	various	£ -	£ -	Donated by Jack Bowles
lead weights	4	£ -	£ -	Donated by Colin Pullen
closed-cell foam	various	£ -	£ -	Donated by Colin Pullen
wire	various	£ -	£ -	Maplin
electrical tape	various	£ -	£ -	B & Q
cable ties	numerous	£ -	£ -	B & Q
		Total	**£ 259.38**	

Appendix K - Permissions

Images from Woods Hole Oceanographic Institution:
Email received 9 February 2012:

Hi Scott,

Thank you for your email. You are welcome to use some images from our site in your project report. Please credit them to Woods Hole Oceanographic Institution.

Best regards,
Erin

Media Relations Office
93 Water Street, MS #16
Woods Hole Oceanographic Institution
(508) 289-3340
media@whoi.edu

Images from VideoRay LLC:
Email received 9 February 2012:

Hi Scott,

You may. Please credit VideoRay LLC in any of the images you use.

Regards,

Brian Luzzi
Marketing Manager
VideoRay LLC

(P) +1 610 458 3015
(C) +1 610 937 6151
(F) +1 610 458 3010
VideoRay LLC
580 Wall Street, Phoenixville, Pennsylvania 19460
www.videoray.com

Schematic and PCB layout from Tim Marvin:
email received 10 April 2012

Scott,

Amazing work man! Of course you can use whatever you need in any way you need to. Attached are the original .sch and .brd files for Eagle. If you need PDFs they should be on my project page on the mbed site. If you need anything else just let me know. I'm happy to help in whatever way I can. Keep up for great work...I'm always interested in seeing how it's going.

Images from Saab Seaeye:
email received 20 February 2012

Hi Scott,

No problem at all. Please go ahead and use the images.

Best regards

James Douglas
Sales Manager

Mob.Tl: +44 (0) 7766 207 384

Saab Seaeye Ltd
20 Brunel Way
Segensworth East
Fareham
Hampshire
PO15 5SD
United Kingdom
Tel: +44 (0) 1489 898 000
Fax: +44(0) 1489 898 001
Web: www.seaeye.com

Images from SMD Ltd:
email received 20 February 2012

Scott
Thank you for your request.

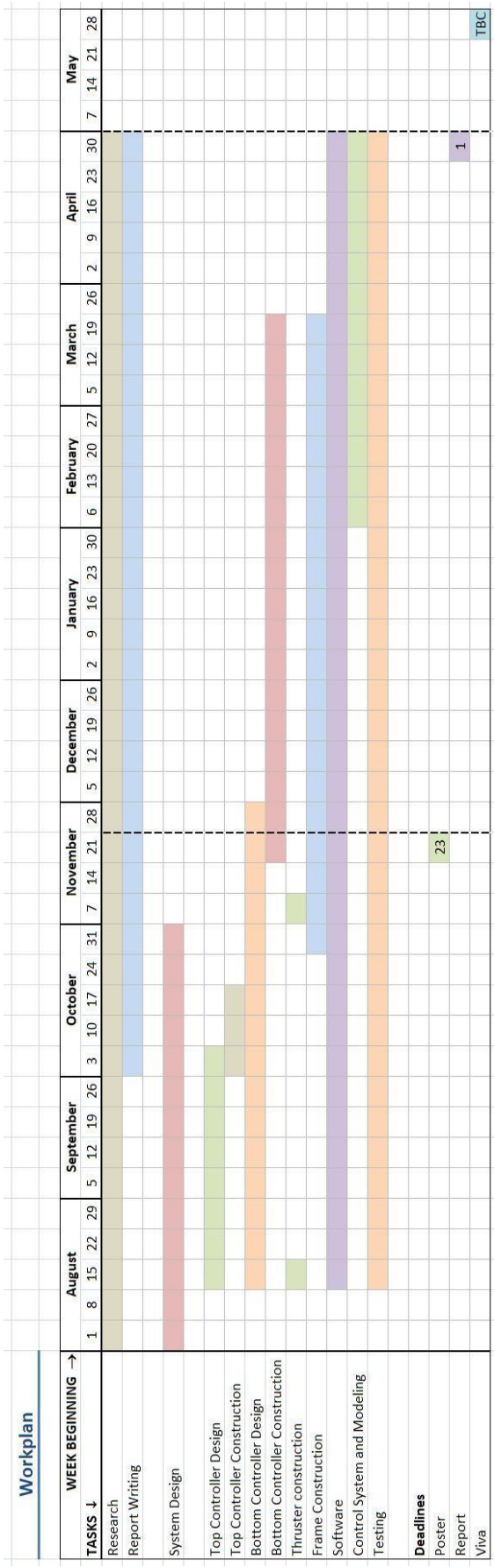
Please feel free to use our images, however I would appreciate SMD being credited.

Many thanks

Graeme Walker
Sales Manager Trenching & Special Projects

SMD Ltd
Mobile +44(0)7891 260324
www.smd.co.uk

Appendix L - Workplan



Appendix M - Disk Contents

Report	- pdf format - docx format
Code	- top controller - bottom controller - data logger code - MATLAB PID controllers plot file
Pitch and Roll data logging	- MATLAB plot files - logged data files
Schematics and diagrams	- top controller schematic - top controller strip-board - bottom controller schematic - bottom controller strip-board - Tim Marvins IMU schematic - Tim Marvins IMU PCB layout
Photos	
Videos	
Noise Analysis	- raw data text files - MATLAB analysis file - Excel spreadsheets
Bill of materials	- Excel spreadsheet
Workplan	- Excel spreadsheet

end of report