Title: Control software for a sailing robot

Module code: SEM9060

Author:Colin Sauzé

Supervisor: Mark Neal

Subitted in partial fulfilment of an MEng degree in software engineering at the University of Wales, Aberystwyth.

Rhif y modiwl Module number	
Dyddiad Date	

Nifer y tudalennau a gyflwynwyd Number of pages handed in	
Rhif Cyfair y Myfyriwr/ **Student Reference Number**	

Cyfenw/Surname .....

Enwau Cyntaf/ First Names.....

.....

Llofnod/Signature.....

Cyfarwyddiadau

1. Lluniwyd y ffurflen hon i sicrhau y bydd eich gwaith cwrs yn cael ei farcio'n ddi-enw.

2. Rhowch eich enw yn y blwch yng nghornel uchaf ochr dde y dudalen hon. Plygwch y gornel uchaf ochr dde a'i selio (defnyddiwch y papur gludiog a roddir ichi yn y dderbynfa) er mwyn cuddio eich enw.

3. Darllenwch lawlyfr y myfyrwyr ar y we i sicrhau eich bod yn ymwybodol beth yw llên-ladrad. Darllenwch y Datganiad Gwaith Gwreiddiol isod a'i arwyddo yng nghornel uchaf ochr dde y dudalen i ddweud mai eich gwaith chi yw hwn.
4. Cysylltwch y dudalen yma â'ch gwaith at ei gilydd trwy ddefnyddio'r styffylwr o'r dderbynfa.
5. Cyflwynwch eich aseiniad yn ôl cyfarwyddyd y darlithydd.

Datganiad Gwaith Gwreiddiol

Trwy arwyddo'r uchod, yr wyf yn cadarnhau mai:

Fy ngwaith gwreiddiol i yw hwn, oni bai ei fod wedi nodi yn eglur fel arall.

Deallaf fod y gosb am lên-ladrad ac unrhyw ymddygiad annheg arall yn llym, ac y gallai arwain at golli marciau neu hyd yn oed beidio â dyfarnu gradd.

Yr wyf wedi darllen yr adrannau am ymddygiad annheg yn Llawlyfr Arholiadau'r Myfyrwyr a'r adrannau perthnasol yn rhifyn cyfredol Llawlyfr Myfyrwyr yr Adran Cyfrifiadureg.

Yr wyf yn deall rheoliadau'r Brifysgol ar y materion hyn a chytunaf i gadw atynt.

Instructions

1. This form is designed to ensure that your coursework is marked anonymously.

2. Please enter your name in the space provided at the top right of this page. Fold over and seal down the top right-hand corner of the page (using the gummed slip provided at reception) in order to conceal your name.

3. Please read the student handbook on the web to ensure that you are aware of the plagiarism issue. Read the Declaration of Originality below and sign in the space provided at the top right of this page to say that the work is your own.
4. Fasten this page to your work using the stapler available from reception.
5. Please submit your assignment as directed by your lecturer.

Declaration of Originality

In signing above, I confirm that:

This submission is my own work, except where clearly indicated.

I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.

I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.

I understand and agree to abide by the University's regulations governing these issues.

Acknowledgements

I would like to thank Dr. Mark Neal for his supervision of this project, help with hardware issues and the time he took to help with testing. I would also like to thank Dr. Fred Labrosse for his co-supervision and help with compass issues. I am also very grateful to Mr. Tero Kuusela for his help with the simulator and to all the members of the Psion-Linux and Jornada-Linux mailing lists who were of excellent assistance on a variety of hardware and Linux problems. Finally I would like to thank Claire Griffiths for her loving support throughout this project.

Abstract

This project implements both PID and fuzzy logic controllers to allow a small sailing robot to sail a pre-determined course based upon compass headings. A simulator was adapted from an open-source project in order to aid testing of these algorithms, actual testing on a lake proved the ability to sail a reasonably accurate course. Given this ability a sailing robot could be used to maintain a given position within a few hundred metres, making it a suitable alternative to data buoys for long-term oceanographic monitoring. Such robots may also be suited to coastal oceanographic monitoring, tracking sea life, surveillance and border patrol operations. Additional work is still required in the areas of concurrency, failure monitoring and adaption, collision avoidance, remote operation and accurate navigation.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Project Background

This project aims to produce a software suitable for controlling a small robotic sailing boat, so that it is able to sail a pre-determined course. It is hoped that a sailing robot could be uesd as a platform to gather oceanographic data. Currently manned survey ships, moored or drifting data buoys and satellites are used to obtain oceanographic information such as monitoring ocean currents, sea temperature and salinity, air temperature, air pressure, humidity, fish stock monitoring, sea bed mapping, wave height and period etc. A sailing robot could potentially be used to replace (or complement) these methods and allow for far greater flexibility and lower costs.

1.1 Overview of existing systems

1.1.1 Moored data buoys

At present there exists a number of moored data buoy networks, some of the most notable of these are the EGOS (European Group on Ocean Stations) network in the North Atlantic, TOA (Tropical Atmosphere Ocean project)/TRITON (Triangle Trans-Ocean buoy Network) network in the South Pacific and the PIRATA (Pilot Research Moored Array in the Tropical Atlantic) network in the South Atlantic. These are able to monitor a number of conditions including sea and air temperature, salinity, air pressure and rainfall. They then report their findings daily by satellite. These buoys require a vast number of man hours to keep them in working order, the TOA and TRITON network consists of over 400 buoys and each of these are visited twice annually, requiring a total of 300 days of ship time per year [4].

1.1.2 Drifting data buoys

In addition to moored networks there are also floating buoys which drift with ocean currents. These tend gather similar information to the moored buoys and must be recovered by ship, although many are washed ashore or lost [5].

1.1.3 Ships

Measurements are also taken from dedicated survey ships and ordinary commercial shipping, known as "Ships of Opportunity" [6]. Although these can provide a valuable source of information, the geographical area covered tends to be limited to normal shipping lanes.

1.1.4 Satellites

Satellite systems tend to be used for monitoring of large geographic areas such as the whole of the North Atlantic. They are often used to monitor surface temperatures, winds, sea state and cloud cover. Satellites have problems measuring under water data and also suffer from accuracy problems given their distances from the seas they are measuring. A bigger problem of satellites is the amount of time and money required to launch one, it is not uncommon for a satellite launch to be delayed for years at a time, this problem is increased at present due to the US space shuttle being out of service.

1.2 Project Justification

The long term justification for this project is to create a system which can complement and perhaps even replace some of the other methods for gaining oceanographic data and in the process reduce the costs, improve the quality and quantity of data collected and improve the reliability and safety of these systems.

1.3 Project Hardware

A small prototype sailing robot has been completed by Dr. Mark Neal, the supervisor of this project. This boat currently contains servos to control the sail and rudder, a wind direction sensor, digital compass and a Basic Stamp small micro controller which communicates with these devices and can provide access to them over an RS-232 serial link. This system also has the potential to interface with other sensors such as a GPS receiver, thermometer, barometer etc. A photograph of the boat is shown in figure 1.1.

Figure 1.1: A photo of the sailing robot.

Chapter 2

Requirements specification

2.1 Requirements Overview

This project aims to develop software that is capable of controlling a small robotic sailing boat so that it is able to sail a pre-specified course. The software will be able to manipulate the rudder and sail of the boat by controlling attached servos and be able to read the current heading of the boat and the direction of the wind from onboard sensors. This software must be capable of running on a micro-controller, PDA or some other kind of portable computer which will fit inside the existing boat and have the ability to run for several hours on its own batteries or those held within the boat.

The software must be capable of reading information from the on-board sensors to determine an appropriate setting for the rudder and sail in order to achieve the desired course. It should also be capable of logging its decisions to a file for later recovery and analysis. In addition to logging the decisions it should also be possible to record the states of all the sensors and servos over time. As well as performing control and logging functions the software should be able to support the recording of some additional environmental variables such as the sea temperature, wave height, salinity etc.

2.2 Hardware Requirements

Its is important that a suitable micro-controller or PDA is selected to run the control software. This device will be responsible for running the software which will control all the servos, read from the sensors, take decisions on setting the sails/rudder, recording the requested parameters and possibly communicating via a radio link. The hardware system will need to be both physically small enough and consume a small enough amount of electricity to fit within the boat and run off a battery internal to the device or the boat's own battery. It should also not generate an excessive amount of heat as it will be placed within a small sealed compartment. Moving parts (e.g. Hard drives) would also best be avoided as vibration from the general movement of the boat could cause problems. As long as these requirements are not violated it should be possible to use several micro-controllers or PDAs to run the software, this could include several controllers of the same type or several different ones. The devices used should be housed to protect them from water damage and should be cheap and expendable as they may be lost at sea.

2.3 Software Requirements

The roles the software must perform can be broken down into two key areas, the first is the actual control of the servos and sensors the other is the software which is responsible for making the decisions as to how to set the servos based upon the information received by the sensors.

2.3.1 Servo and Sensor control software

The low level software sensor and control software must be capable of reading the current values of the sensors and setting the states of the servos based on the values given to it by the high level control software. It will be required to perform conversion of the values between real world values such as degrees and values understood by the sensor/servos such as voltage levels or Pulse Width Modulation timings. In all the sensor and servo control software must be able to read from a digital compass and wind sensor as well as any environmental sensors such as a temperature sensor for measuring sea temperature, there is also the possibility that it may need to be able to read from a GPS receiver and the design should be able to cope with this need. It must also be able to control the rudder, sail and possibly be able to transmit data over a radio link back to shore.

2.3.2 Navigation Requirements

The system needs some means by which navigation can be performed in order to gather information which can be used to keep the boat on course. There are two obvious forms this can take, the first being GPS and the second being an onboard compass. Ideally any such system should not be subject to variation resulting from the boat leaning and it should be quick to update to allow rapid feedback between measurements and the maneuvers which keep the boat on course. Given the relatively slow speed at which GPS updates and its inaccuracy in determining headings when moving at slow speeds a compass may prove more suitable. However a compass (unless placed on gimbals) is prone to a loss of accuracy as the boat leans and cannot detect if the boat is drifting side ways or going backwards as a GPS can.

2.3.3 Simulator Requirements

As testing on a real robot in the water will be limited by weather conditions and time it will be necessary to develop or identify a suitable simulator system which can (from the artificial intelligence code's point of view) take the place of the lower level code and pretend that it is the real robot. Such a simulator needs to provide the same basic information (wind direction and boat heading) to the artificial intelligence algorithms while at the same time allowing the algorithms to control the simulated rudder and sail. The simulator should be reasonably realistic and not allow things which a real life boat clearly couldn't do, such as being able to sail directly into the wind. However it cannot be expected that a simulator will be fully realistic and it cannot be expected to accurately simulate all possible conditions as to do so could require immense amounts of time and computing power. Ideally the simulator should give output in a graphical form and to a data file from which graphs can later be generated is also acceptable. It would also be beneficial if the simulator were able to vary the wind strength, wind direction and boat heading in order to simulate changes in the environment, if this is not done then it is possible that errors which only occur when small corrections are required do not get discovered until the robot is tested in real life. Using a simulator also helps to reduce the risk to the project, should the robot sink or become irreparably damaged in an early test run then it will be possible to continue development using the simulator alone.

2.3.4 Artificial Intelligence Algorithms

The artificial intelligence level control algorithms are responsible for actually making the sailing decisions and issuing appropriate commands to the low level interface, which will in turn send them to the rudder/sail or return data from the sensors. The decisions taken by these algorithms need to follow the basic rules of sailing as defined in many sailing manuals such as the Tech Dinghy guide[2]. These algorithms will also be required to record their decisions so that analysis can be performed later on to see why a given decision was taken.

As testing will require the use of a simulator, these algorithms should be able to interface with either the simulator or the real robot without noticing the difference and requiring virtually no reconfiguration. It is possible that minor reconfiguration could be required to overcome differences between simulator behaviour and robot behaviour.

2.3.5 Portability Requirements

Where-ever practical all software should be written in a manner which allows it to be ported to another CPU architecture and/or another operating system. This will allow the possibility of changing any hardware which fails to perform as required with more appropriate hardware without the need for a major re-write of any software. As a result of this requirement all software should be written in portable languages such as C or Java where ever possible. There may still be some situations where a specific piece of hardware requires a specialised development environment to be used and thus results in non-portable code. The use of such hardware environments is permitted, but the amount of code written for them should be kept at a minimum and alternatives should be considered.

2.3.6 Data Logging Requirements

It is necessary to log a number of parameters at regular intervals during the running time of the control programs in order to be able to decipher what happened and to understand the decisions taken by the code at a later point in time. The code should log the current time or time elapsed since the program started, the current heading and if possible position, the current wind direction and the current desired heading. The data should be logged to a file with each of these parameters separated by a space or comma in order for easy analysis with a spreadsheet, graphing application or custom written analysis tool.

2.3.7 Configuration Requirements

It will be necessary to vary the parameters of the control software in order to perform testing. Such parameters include the target position or heading, the amount of time to spend sailing before the program exits and possibly parameters to control the exact behaviour of the artificial intelligence algorithms.

2.4 What is not required

This project is not attempting to develop the actual hardware for a sailing robot, it is only developing software to control one in order to sail a pre-determined cause. As this project is to be implemented part time in under 9 months it cannot hope to implement every software system required for a sailing robot. There is not be sufficient time to attempt to implement systems

for reliable bi-directional communications, collision avoidance, management of fleets of robots or control of complex sensors such as rain and underwater sensors. However the software should be built in such a manner that adding these features would not require any major changes to the existing code.

Chapter 3

Market Analysis

3.1 Why is the system needed? What is the problem it will solve?

There are a number of possible applications for a sailing robot. These include long term off-shore oceanographic monitoring, coastal or inshore monitoring, surveying the sea, monitoring marine life, providing surveillance to port entrances and remote un-patrolled stretches of coastline or searching for mines. Of these the most obvious application is for off-shore oceanographic monitoring, which is currently performed via a combination of satellites, data buoys and survey vessels. There are a number of major shortcomings with current oceanographic monitoring systems which a sailing robot may be able to address.

Satellites tend to be very expensive to build and maintain, they also take years to construct and can spend years waiting to be launched especially with the current lack of US space flights. Satellites also have problems with the resolution and accuracy of their sensors as a result of the distance between the satellite and the environment it is monitoring, whereas a sailing robot would be on the water surface and therefore would not be subject to these problems, it is also far easier and cheaper to deploy than a satellite.

The use of survey ships tends to be a lot cheaper than a satellite, but coverage area is usually a lot smaller and there is still the need to feed and supply the crew on-board which tends to limit the length of time a survey can be carried out for. Where as a sailing robot can sail anywhere it needs to go and remain there indefinitely providing that it does not break down, has sufficient power to maneuver itself and power the onboard equipment. Sailing robots could be instructed to sail a pre-determined course (e.g. across an entire ocean), monitor ocean conditions as they move and then report this information back to shore via a satellite link. This would remove the limitations of following normal shipping lanes and avoiding hazardous waters (e.g. the Antarctic in winter).

Data buoys (both moored and drifting) solve some of these problems as these can stay on station for years, large networks (e.g. TOGA and GOOS) can be laid to give wide area coverage and at relatively low cost. However data buoys still require a ship(s) to launch them, repair them and in the case of drifting buoys recover them. Deploying several hundred buoys into a moored network such as TOGA is a very labour intensive process and takes several ships, several years to achieve and once a network such as TOGA is deployed it is fixed at pre-determined locations. Additional problems occur with drifting data buoys when they wash ashore as they may become damaged in the process and it is unlikely that they will be returned to their owners. A sailing robot offers the potential to build a large network of monitoring stations but to have them re-deployable to

a specified location. This allows for the benefit of a data buoy network without the need to use a fleet of ships to deploy each buoy and anchor it or the need to recover it. A sailing robot could potentially take on the role of both a moored buoy by attempting to hold a given position or that of a drifting buoy by simply drifting with the currents but able to sail away from the coastline should it drift too close. If a robot needs to be recovered it could be tasked to sail back to a port or rendezvous point.

3.2 What kind of related system are there in production and research at present?

There are no commercially available sailing robots at this point in time, however a number of systems for performing similar tasks currently exist and a number of others are currently being researched. Existing systems include data buoys, survey ships and satellites. Systems under research include autonomous powered boats, autonomous submarines and biomimetic vehicles as well as other attempts at building a sailing robot.

3.2.1 Systems presently available

At present there are several systems in existence for monitoring the worlds oceans. These include GOOS (Global Ocean Observing System) and GCOS (Global Climate Observing System) which in turn are made up of many regionally organised networks such as EGOS. These systems integrate data from a number of sensor networks including moored data buoys, floating data buoys, survey ships, ships of opportunity (commercial ships equipped with sensors), satellites, tidal gauges and coastal observing stations. Between them these systems are able to gather a wide range of data, however a number of problems exist.

Moored data buoys must be anchored to the sea bed at considerable expense and require regular maintenance, as a result ships must visit them in order to carry out repairs. According to EGOS [5] it costs €400 to deploy each data buoy, hopefully a sailing robot would be able to cut this to nearly nothing. The TOA/TRITON network requires some 300 days of ship time each year to recover and maintain moored buoys [4]. A sailing robot could be made to hold a given position while gathering data and the return to port after several months for maintenance, this could result in a dramatic reduction in deployment and maintenance costs. Data buoys have also proven to be a problem for fisherman [7], fish are actually attracted to the moorings and as a result so are fisherman who often get their nets tangled in the buoys causing damage to them. This is a problem for all involved, it means fisherman loose their nets and may expose themselves to dangers in recovering them, the buoys become damaged in the process causing problems for oceanographers and meteorologists and as the fish are concentrated in a smaller space this will have an impact on the surrounding environment.

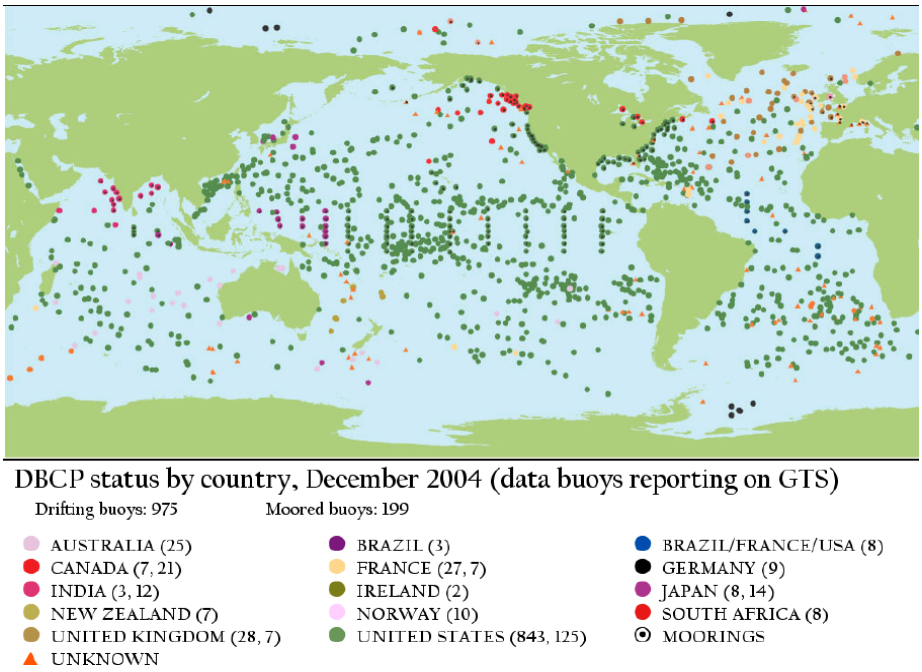

Figure 3.1: Map showing locations and owners of all moored and drifting data buoys. Values in brackets indicate the number of drifting buoys followed by the number of moored buoys. Source Operations and Achievements of the DBCP [1]

As with moored buoys floating buoys suffer from problems of deployment and recovery. It is not uncommon for them to be lost at sea or to run ashore after only a few months in the water. A sailing robot could potentially be made to drift either by pointing towards the wind or somehow lowering its sail and/or mast while at the same time avoiding land and being able to automatically deploy and recover themselves. Sailing robots could also be used in place of or in addition to survey ships and ships of opportunity to perform surveys of conditions across a large area. Although survey ships have no major restrictions as to where they can travel, they are expensive to deploy and only a limited number exist. Ships of opportunity help to complement this as they are far greater in numbers, according to the Met Office "At the end of 2000 the UK Voluntary Observing Fleet (VOF) numbered 552 vessels" [8].

Satellite systems are currently used for a vast number of monitoring tasks including measuring the surface temperature of the sea, wind speeds, cloud cover etc. Satellites are very good at monitoring wide areas, but are not so well suited to take measurements under the surface and it may still be cheaper to deploy a series of sailing robots or data buoys to detect things like sea temperature than to launch a single satellite. To give an indication of this, data buoys cost around €10,000 to build and about € 400 to deploy[5] each, a satellite launch (excluding manufacturing costs) is at least $ 8,00,000 [9] or € 6,250,000 the cost of manufacturing 625 data buoys. In addition to this, satellite launches are often delayed by months or even years and its is not uncommon for satellites to be lost or at the very least placed in the wrong orbit.

3.3 Systems currently being researched

3.3.1 Other sailing robots

No sailing robots currently exist as off the shelf products. The nearest available products are remote control sailing boats, but these tend to be small and unsuitable for use in open waters.

There is no off the shelf software for the control of sailing robots, although there have been two previous attempts by academic researchers to build robotic sailing boats. These are Abril, Salom, Calvo, 1997[10] and Ross, 1998 [11]. Unfortunately neither of these seem to have followed up their research into a system actually capable of performing oceanographic monitoring.

3.3.2 Powered Autonomous Surface Craft

Several attempts have been made to produce autonomous surface craft (ASCs) that are driven by electric motors, combustion engines or turbines. The obvious limitation of these is that they must either enough carry fuel and/or batteries to power themselves or gather enough power from the sun, wind or waves. These factors severely limit the amount of time a boat can operate for and/or its maximum speed in comparison to that of a sailing robot. However powered ASCs also offer the major advantage over a sailing robot of being able to operate without sufficient winds and at relatively high speeds even if only for a few hours at a time. A number of research projects have been carried out in this field, perhaps the most notable is Vaneck (1997) [12] who built a 1.4 metre long autonomous boat guided by GPS and controlled using a fuzzy logic controller and Goudey et al (1998) [13] who attempted to use a small kayak to autonomously track fish in a river. Both of these projects were part of the wider MIT Sea Grant project and were conducted by the MIT AUV Laboratory [14]. Additional powered ASCs have been built by Ross (1998) [11] who built a small sailing robot and a larger electrically powered boat and Rocca (1999) [15] who built a small GPS guided boat controlled by a simple assembly language program.

Additionally remotely controlled surface craft have been used in environmental monitoring with the Texas Coastal Observation Network (TCON) [16, 17] which is currently experimenting with such craft to obtain water quality information in shallow estuaries. The motivation behind using such craft is because of their exceptionally shallow draft and small size which allows them to travel with a minimum of disruption through the exceptionally shallow waters found along the Texas coastline. An autonomous sailing robot could be used for similar applications and could be equipped with a similar sensor package to that of the TCON boat. The only possible problem with using the robot used in this project is that it has a 70cm deep keel which may strike the bottom in exceptionally shallow waters.

3.3.3 Underwater vehicles

3.3.3.1 Robotic Fish and other biomimetic systems

It has been suggested that attempting to mimic the actions of a fish and other marine creatures is an efficient way to power and steer man made vessels, mimicking nature in this way is generally known as biomimetics. A number of researchers have built robotic fish these include Triantafyllou and Triantafyllou (1995) [18] who created the RoboTuna one of the first robot fish. As mentioned by Tzeranis, Papadopoulos and Triantafyllou [19] robot fish are currently only capable of operating for a few hours at best and are therefore not yet suitable for performing oceanographic functions given this limitation. Despite the efficiency of swimming in comparison to a propeller a robotic fish still requires constant movement of its fins and is unlikely to be able to match the power efficiency a sailing robot can potentially deliver, however it should theoretically be able to exceed that of a conventionally powered surface craft. As identified by Tzeranis, Papadopoulos and Triantafyllou [19] the possibility exists to improve the efficiency of robotic fish by propelling them using Shape Metal Alloys and Electroactive Polymers both of which are capable of changing shape from electrical stimulation, such technologies remove the need to the translate rotary motion of traditional electric motors into the flapping motion of a fish however such technologies are not yet mature enough to achieve this.

3.3.3.2 Autonomous Submarines

A number of researches have attempted to build autonomous submarines (often called AUVs or Autonomous Underwater Vehicles) for a variety of tasks including ocean monitoring, undersea cable laying and mine clearance. Two notable examples of these are Southampton University's AutoSub-1 [20]and the United States Naval Postgraduate School's ARIES AUV [21]. Both of these AUVs are powered by onboard batteries, limiting their range to a few days operation at best. When compared to a sailing robot AUVs are somewhat limited due to this factor, however they have the advantage of being able to sense what is deep beneath the surface rather than simply the top few metres of water as a sailing robot can.

3.3.3.3 Argo Floats

Argo floats [22] are a form of drifting data buoy which are able to sink and gather information underwater and then resurface to transmit the information they have found. They achieve this by increasing and decreasing their volume through the use of an inflatable bladder into which oil is pumped when the float wishes to surface or from which oil is withdrawn if the float wishes to sink. Argo floats have no control over their course, only their depth, they usually drift under water for several days before surfacing.

3.3.3.4 Ocean Gliders

Ocean gliders [23] work on a similar concept to the argo float, however they are equipped with wings and are able to glide upon temperature differentials caused ocean currents controlling their direction with aircraft like control surfaces. As with the argo float the depth of the glider can be controlled by using an inflatable bladder, ocean gliders also tend to surface every few days both to transmit their data and to take a GPS reading in order to correct their course. They are capable of running for several months, although they are still dependant on battery power to change the size of their bladder and to operate their control surfaces. Both ocean gliders and argo floats are capable of deep sea monitoring, which a sailing robot is not. However they are both reliant on being underwater and are not able to hold a position and are therefore not capable of covering roles similar to a moored data buoy (or a sailing robot attempting to hold a given position). It is possible that sailing robots and argo floats/ocean gliders could be used together, a large sailing robot might be able to deploy or even retrieve argo floats and ocean gliders reducing the need for manned ships to deploy them, thus reducing their operating costs and/or allowing them to operate in more remote locations.

3.3.4 Autonomous Ocean Sampling Networks

Autonomous Ocean Sampling Networks (AOSNs)[24] are an attempt to integrate information from a number of sources such as Ocean gliders, AUVs, survey vessels, satellites, data buoys and aircraft to produce a wider picture of what is going on within the ocean. Currently research into this area is being pioneered by MBARI, the Monterey Bay Aquarium Research Institute [25] in conjunction with several universities in the USA. It is quite possible that sailing robots could be used as an additional component within an AOSN to either add another data source or to replace (or at least reduce) the role undertaken by survey vessels.

3.4 Are there any existing systems that can be adapted to do this job?

At present there are no off the shelf systems designed for the complete autonomous control of a sailing boat, however there are auto helm systems intended to help sailors maintain a steady course. Such systems are often used by solo yachtsmen to allow them to continue sailing while sleeping or to advise the yachtsmen on how to adjust their course and sails in order to sail more efficiently. One such system is the RoboSail [26] project, which uses AI techniques to make course corrections and wind sensors, a compass and GPS to gather data. Other systems within the boat such as navigation, data logging, communication and steering systems are likely to be very similar to those already found in data buoys, autonomous boats and autonomous underwater vehicles. It is therefore likely that some of the equipment and algorithms used maybe similar.

From a wider perspective there are existing systems which can perform the oceanographic applications of a sailing robot. However, as previously mentioned these systems have a number of shortcomings and sailing robots maybe able to overcome some of these. This may result in lower operational costs, a more versatile oceanography platform and a system which is less likely to become lost at sea.

There have been several research efforts and efforts by private individuals to produce autonomous powered boats. Some of the control algorithms used in these could be used to control navigation of a sailing robot, but they are not suitable to control the actual sailing logic. One such system that may be suitable is the Roboat system by Rocca (1999) [15]. This system was developed to allow a small electrically powered model boat to navigate around a lake for a competition run by electronics magazine, Circuit Cellar. Another similar system is the Robotic boat for Autonomous Fish Tracking by Goudey et al (1998) [13]. This followed a similar strategy as Rocca's system by taking GPS readings at regular intervals and steering appropriately upon each reading.

3.5 Is there any wider demand for this product?

The main demand for a sailing robot appears to be from the oceanographic community at this point in time. However there may also be potential uses in performing environmental monitoring of river estuaries and inshore waters, as is being done by the remote control boats used in the TCON network [16, 17]. There may also be applications in patrolling coastlines for smugglers or terrorists, as well as military applications such as reconnaissance in enemy ports and mine detection,. Such applications have already been proposed for powered boats [27], however a sailing robot would provide an obvious platform for these applications given its ability to run for long periods of time without refuelling or needing new batteries. Another possibility is for performing marine biology tasks such as tracking whales and fish via passive hydrophones.

3.6 What are the risks involved in producing, delivering, supporting and using this system?

3.6.1 Risks during development

The immediate risks in developing this project are mostly related to the potential loss of the robot during testing. Should this happen it is unlikely that another robot will be available before the end of the project and without an appropriate backup this would leave all of the software

un-testable. To reduce this possibility a simulator is to be used to test the high level sailing control algorithms, should the robot become unavailable the simulator will be able to act as an appropriate alternative to develop these algorithms. There is also a danger that software which works perfectly in the simulator may not work in real life due to the differences between the behaviour of the simulator and the real robot. Such differences could occur in a number of areas such as the robot not sailing properly at all, the wind being less predictable and consistent in the real world and the action of currents, waves and tides upon on the boat.

3.6.2 Risks to end-users and the environment

There are also a number of risks for an end user who deploys robotic boats. The biggest financial risks are most likely to concern the potential loss of a boat or several boats, this could due to adverse weather, hardware failure, hitting land, colliding with another vessel, colliding with or being swallowed by an animal, software errors or any combination of these. Some of these issues are also a problem with data buoys and satellites. A large number of data buoys are lost or damaged either by the weather or by humans, as is shown by the European Group on Ocean Stations Report on the EGOS Management Committee Meeting [5] and by TRTION Data Buoy Network's web site [28]. Problems with data buoys are increased as fish are attracted to mooring cables [7] and as a result fisherman being attracted towards the data buoys.

A potential liability issue also exists if a robotic boat causes damage to another vessel, a buoy, fishing nets/lines/farms, wildlife etc. This would presumably follow similar liability problems which drifting data buoys face. Liability could potentially rest with both the owner/user of the boat as well as the programmers, hardware designers and manufacturers should a failure which cause damage to somebody else be the result of negligence. Several such problems have already arisen from data buoys, there have been cases of lead acid batteries producing hydrogen gas and subsequently exploding due to electrical faults. Unfortunately in 2001 a crew member of a ship performing buoy maintenance for the Indian National Institute for Ocean Technology was killed when such an explosion took place [1]. Although this fault does not appear to have been caused by software, the accident might have been prevented by the use of software which could power down the electronics when a hydrogen leak was detected or at least send a warning message so that the recovering crew are aware of the issue. It is likely that a sailing robot could suffer from the same problem of hydrogen leaks from a lead-acid battery and that safety systems to detect and/or prevent such a problem be added to any sailing robot to be used in the real world.

Similar issues have been highlighted with regards to Autonomous Underwater Vehicles and Autonomous Vehicles in general. According to Showalter (2004) [29] the International Regulations for Preventing Collisions at Sea only apply to vessels and defines a vessel as "includes every description of watercraft or other artificial contrivance used, or capable of being used, as a means of transportation on water", therefore it is not certain that most autonomous vehicles will be classified as such as they are not capable of acting as a means of transportation. Should an autonomous vehicle be classified as a vessel then a number of implementation issues will arise as a vessel is supposed to provide appropriate lighting (e.g. port and starboard lights), pass other vessels with the port side of the vessels facing each other and respond to audible signals. As these regulations where written in the 1970s autonomous vehicles were not considered. However, if a sailing robot does not classify as a vessel then it does not need to adhere to these regulations and given the size of the robot being built for this project it is unlikely that a collision with a vessel of any size would actually cause the other vessel any major damage. As Showalter points out problems also apply with regards to harm caused to animals, this especially applies to the operation of autonomous vehicles operating within marine reserves. However, as a sailing robot is unlikely to move at any great speed, has no propeller and does not cause any major noise

(unless sonars or acoustic modems are fitted to it) it is unlikely to cause any harm to animals and the greater risk is that the animals will damage the boat.

3.6.3 Support Issues

Support problems potentially exist if a software update needs to be applied to a robot while it is at sea. It would be theoretically possible to perform updates of software remotely, but failure in doing this (or any other software failure resulting in a loss of communication) could cause the loss of a robot, and thus financial loss to its owner. Most support issues with commercial users of sailing robots are more likely to regard hardware problems than software problems.

3.6.4 What issues concerning intellectual property rights need to be considered in relation to this project?

There are currently no known intellectual property issues with this project. There are no known patents covering this technology and the only 3rd party software currently being used is available under Open Source licenses.

Chapter 4

Design

4.1 Hardware Architecture

The choice of hardware available to this project is rather restricted due to the limited budget available. As a result 3 suitable computer platforms are available. These are the Basic Stamp 2sx micro controller, a Palm M100 PDA and a Psion Series 5mx PDA.

4.1.0.1 The Basic Stamp

Basic Stamp 2sx[30] is a small micro controller featuring a simplistic micro processor, 16 general purpose I/O lines, I2C bus support, 16 kilobytes of EEPROM split into 2 kilobyte pages, 32 bytes of RAM and an RS232 interface for connection to a PC and/or other micro controllers. The Basic Stamp uses a variant of the BASIC programming language for writing code, this makes it simple to program in comparison to many other micro controllers. Its I/O facilities make it ideal for connecting to a number of control lines and sensors, however its limited processing power and memory make it difficult to implement complex control algorithms.

4.1.1 The Psion Series 5mx

The Psion 5mx[31] is a PDA (Personal Digital Assistant) designed to be a miniature replacement for a laptop PC. It features a 33 mhz ARM processor, 16 megabytes of memory, 640x200 pixel LCD screen, an RS-232 serial port and infra-red adapter for connecting to other devices and a compact flash slot allowing for storage to non-volatile memory. By default the Psion 5mx runs the EPOC operating system from its internal ROM, however it is also possible to boot other operating systems such as Linux or NetBSD from a compact flash card. The Psion's small size, (relatively) fast processor/large memory and support for non-volatile compact flash make it an ideal choice for implementing higher level control algorithms which require more memory and computing power than a micro controller can offer. However the Psion alone does not make a suitable controller as it only has two I/O devices, a serial port and an infra-red port. This makes it difficult to connect the Psion to more than two other pieces of hardware, thus it needs to be connected to some other device (such as a Basic Stamp) in order to be useful for controlling a sailing robot.

4.1.2 The Palm M100

The Palm M100[32] is a PDA designed to act as a powerful pocket organiser and features a 16mhz Motorola Dragonball processor, 2 megabytes of RAM, a 160x160 pixel LCD screen, an RS232

serial interface, an infra-red port and a ROM containing the PalmOS operating system. At this point in time the Palm is not capable of running alternative operating systems, however PalmOS programming tools are readily available as is programmer documentation. Like the Psion 5, the Palm alone is not suitable for controlling an entire robot as it only has a serial port and an infra-red port with which to interface to other hardware. Unlike the Psion, the Palm features no way of storing non-volatile data as it has no support for flash memory cards and relies solely on keeping power running through its RAM to store data.

4.1.3 Choice of Hardware

The final choice of hardware is to use a Psion 5mx in conjunction with a Basic Stamp 2sx. This combination allows for the Psion's I/O limitations to be overcome by passing requests over an RS232 cable to the Basic Stamp. It also allows for the memory and processing limitations of the Basic Stamp to be overcome by using the Psion to perform any CPU and memory intensive operations. The Psion's ability to use compact flash cards allows for significant amounts data to be logged and should the power fail no data will be lost. Compact flash cards also allow for the use of Linux or NetBSD operating systems instead of the default EPOC, the advantages of this will be discussed in the next section.

One additional possibility for the Psion is that its infra-red communications facilities can be used to allow reconfiguration of the sailing algorithms, without having to physically access the Psion. As the deck of the robot is clear perspex it should be possible to use infra-red communication between the Psion inside the robot and another device supporting infra-red communication such as another PDA or a Laptop. This will speed up testing time as there is no need to disassemble the robot in order to change parameters on the Psion. The most obvious way to implement this system is to attach a console to the infra-red port under Linux, this will allow a remote user to work as if they were typing directly on the Psion, allowing them to edit configuration parameters, reboot the system and restart the control programs. Unfortunately infra-red communication will only work at very short range (probably no more than 1 metre) and only with line-of-sight to the client system. Ideally such a system could be replaced by a longer range radio link such as Bluetooth or 802.11 wireless link, however the Psion has no built-in support for such systems.

If sufficient funds were/are made available to this project then a more suitable microcontroller could be used. Ideally such a microcontroller would combine the processing power and storage capabilities of the Psion with the diverse I/O facilities of the Basic Stamp. One micro-controller system which fulfills this potential is the Strong ARM 1100, this actually a successor to the ARM7 processor used in the Psion. The Strong ARM 1100 includes a 200MHZ ARM processor as well as controllers for a number of common peripherals including LCD screens, serial ports, USB ports and PCMCIA cards. The Strong ARM 1100 is commonly used in a number of microcontroller boards such as the LART or Linux Advanced Radio Terminal, it has also been used in several PDAs including the Compaq IPAQ, HP Jornada 7xx Series and Sharp Zaurus. At the other end of the scale, simpler microcontrollers such as the Motorola 68HC11 may also be suitable, however these are not capable of running an operating system as sophisticated as Linux and would require some changes to the current code.

4.2 Software Environment Choices

4.2.1 Available Operating Systems

Three possible operating systems can be used on the Psion Series 5mx. These are Symbian EPOC which is included in the ROM of the Psion, NetBSD [33] a highly portable Unix system

based around the BSD core and Linux[34] a popular Unix clone.

EPOC is mainly intended for the development of graphically driven applications and development must be done via either an extensive SDK, which requires the use of Microsoft Visual C++. Development can also be done using a BASIC like language called OPL which comes bundled with the Psion, however OPL is rather limited in its features. Development via the EPOC SDK or via OPL will also restrict the portability of any code, as the resulting code will only run on EPOC machines which are currently limited to Psion PDAs and a few mobile phones.

NetBSD [33] is intended to be a highly portable general purpose operating system and has been ported to 18 different microprocessor architectures. It is possible to cram a NetBSD installation into a few megabytes, making it ideal for use on a small scale device such as a Psion 5mx. Given this information NetBSD sounds like an ideal choice, unfortunately the current port to the Psion 5mx receives little attention and a number of issues still remain, preventing it being a usable operating system.

Linux was originally intended to only run on desktop PCs with Intel 80386 compatible CPUs, however since the mid 1990s it has been ported to a variety of architectures including the ARM CPUs which the Psion 5mx uses. As with NetBSD it is possible to reduce a Linux installation to a few megabytes, making it ideal for use on the Psion 5mx. There currently exists an active community supporting Linux on Psion PDAs, this centres around the OpenPsion project[35] which hosts kernels specific to Psion systems. Programming with Linux allows for the control software to be ported to any other system which can run a POSIX compliant operating system (such as NetBSD or any other Unix, QNX or VxWorks) by simply recompiling the source code for that platform.

4.2.2 Cross-compilation Environments

In order to compile code for a Psion 5mx it is easiest to compile it using a desktop PC equipped with a specialist compiler to produce executables for another platform, these are known as cross compilers. The popular and freely available GCC C compiler can be rebuilt to cross-compile, however configuring this from scratch is not an easy task. A number of utilities exist to ease this configuration, these include CrossTool [36], a script for setting up a cross-compile environment. Initial experiments with CrossTool proved unpromising and a number of errors were encountered. An alternative system, called uClibc Buildroot [37] was found. uClibc buildroot is capable of building a full embedded linux system, allowing the user to choose what components are included. The uClibc buildroot includes the uClibc C library, a cut down version of the full GNU Standard C Library and Busybox, a single executable which replaces all the standard Unix commands. uClibc is exceptionally easy to configure, simply downloading the source and typing "make" presents a configuration menu and then builds all required tools (including the cross compiler) and a complete image file of a Linux system. The resulting image file is usually no more than two megabytes, which is ideal for installation to a compact flash card on a Psion 5mx. Given this ease of use and small size of resulting executables uClibc is the most obvious choice for a cross compile environment.

4.3 Overall Software design

The system needs to be split into 4 main areas. The first is the micro-controller code which runs on the Basic Stamp and interacts directly with the sensors and servos of the robot. It

also communicates with the next level via the Basic Stamp's serial port. The second area is the medium level code running on the Psion, this interacts directly with the stamp and creates an API for higher level components to use to read from sensors and control servos. It is also responsible for performing conversion of values, so that the voltage or time values given to/by the servos and sensors are in suitable units (e.g. That a compass heading is in degrees). The third area is the high level control algorithm which is actually responsible for making decisions for steering the boat and setting the sail appropriately. It should perform these operations by communicating with the low level code. The fourth area is a simulator which can take the place of the low level code and allow the high level control algorithms to think that they are controlling the real boat.

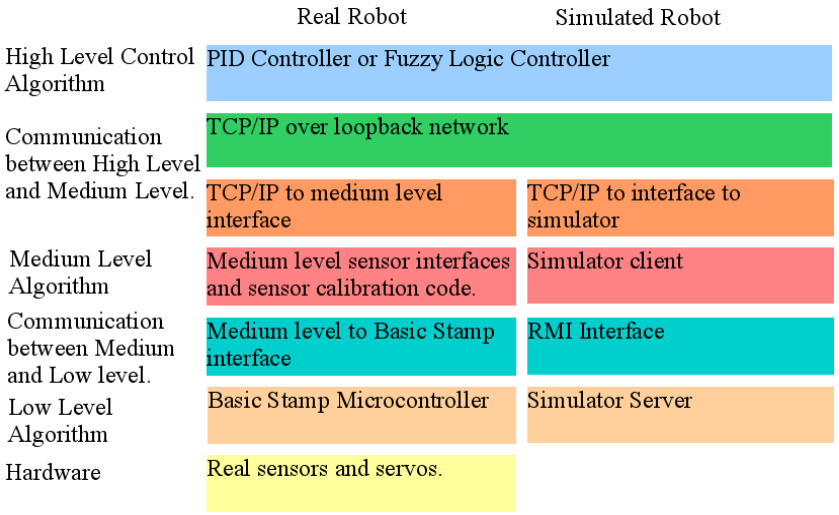

Figure 4.1: The different layers of the software architecture. The high level code may interface either to the simulator or to the real robot.

4.3.1 Detailed design of the micro-controller code

The lowest level of the code will run on a Basic Stamp micro-controller and be able to directly interface with the sensors and servos using the Basic Stamp's I/O facilities. This code is to be largely based upon code supplied by Dr. Mark Neal. Apart from actually sending data to/from the sensors and servos this code must also interface with the Psion via the Basic Stamp's serial port. In order to do this a common protocol must be used on both devices, given the Basic Stamp's limited memory and processing power this protocol needs to incredibly simple to implement.

The code on the Basic Stamp must be able to handle requests to read values from the compass and wind sensor. It is also desirable for it to be able to read the sail position, although this isn't absolutely required as the previous value the sail was set to can be used. It must also be possible to set the position of the sail and rudder. Table 1 shows the value range. As a result of differing value ranges, different amounts of memory will be required for some values and this will require more bytes to be transmitted over the serial line. The medium level code must also be able to handle this.

Sensor/Servo Name	Set or Get	Value Range
Rudder	both	0-255
Sail	both	0-160
Compass	get	0-220
Wind sensor	get	0-65535

Table 4.1: The value range for each of the variables handled by the Basic Stamp.

In addition to the compass, wind sensor, sail and rudder the Basic Stamp should have the flexibility to handle other pieces of hardware. The two devices most likely to be added to the boat are a GPS receiver and a radio link. GPS receivers typically work by generating 10 or more character NMEA strings, which state the current longitude/latitude and optionally the time and altitude. As a result of this there is a requirement for the protocol that communicates with the Psion to handle message parameters of at least 10 characters.

Calibration of any values sent to/from the hardware is done in the middle layer code running on the Psion rather than in low level code on the Basic Stamp. However there is scope for the Basic Stamp to perform some simple tasks with the data such as taking an average over several sensor readings rather than supplying a single reading to the Psion when gathering data from sensors.

4.3.2 Implementation Platform

As most of the development tools and example code for the Basic Stamp centre around the custom version of the BASIC programming language used in the Basic Stamp this seems an obvious choice for implementing all Basic Stamp code in. The resulting programs are likely to be no more than 200 lines so the shortcomings of the BASIC language (e.g. Goto statements, poor parameter handling in subroutines etc) should not be a major problem.

4.3.3 Serial Protocol Design

The serial protocol used between the Psion and the Basic Stamp must be capable of getting/setting all the variables stated in Table 4.1. The communications protocol shall work by sending 3 bytes to make up each command, followed by up to another byte to specify any operands. The first character of each command sent to the stamp will be either "G" for get commands, "S" for set commands. The next 2 characters determine what device is being requested, these values will be "CH" for compass heading, "WS" for wind direction, "RD" for rudder position and "SL" for sail position. The return data will then be up to 10 bytes in length, set commands will simply reply with the string "!!!!!!!!!!" and get commands will reply with the value read from the sensor, replies will be encoded as raw bytes rather than character strings (e.g. a value of zero sends a value of zero not the ASCII character for zero.) Full details are shown in table 4.2.

Command	Description	Parameters	Returns
GCH	Gets Compass heading	None	Result of PWM measurements from the compass.
SSL	Sets sail position	Sail position , as a potentiometer value to match.	Nothing
GSL	Gets sail position	None	The sail position as a potentiometer value.
GWS	Gets wind direction	None	Returns the wind direction as a potentiometer value.
SRD	Sets rudder direction	Rudder direction as a potentiometer value.	Nothing
GRD	Gets rudder direction	None	Rudder direction as a potentiometer value.
SUW	Turns the sail counter-clockwise to unwind the wire to the wind sensor should it become wrapped around the mast.	None	None

Table 4.2: The protocol used for communication between the Psion and the Basic Stamp. This is based upon a protocol originally designed for this system by Dr. Mark Neal, see original specification in Appendix C.

4.4 Detailed design of medium level code

The medium layer code runs on the Psion and is responsible for providing a TCP/IP socket interface to allow the high level code access to the servos and sensors, making use of calibration data to translate between real world measurements such as degrees and measurements understood by the low level code such as Pulse Width Modulation timings and Potentiometer values and finally it must interface with the Basic Stamp via an RS-232 serial line and forward on commands which it receives over TCP/IP to the Basic Stamp. The use of a TCP/IP interface to the high level code allows for the high level code to potentially run on a different computer should a distributed architecture be required and it also allows the high level code to run as a separate process and even be written in a different language to the medium level code. This is especially useful as languages often suited to the development of lower level I/O routines (e.g. C) are not as well suited to implementing very high level algorithms and languages suited to very high level development (e.g. Prolog, Java, Lisp) are often not very suited to lower level development.

4.4.1 Implementation Platform

The medium level code is to be implemented in ANSI and POSIX compliant C code. This allows for portability to any POSIX compliant operating system including most versions of Unix, Linux,

QNX and VxWorks simply by recompiling the code for the relevant platform. Dependencies on specialist libraries are to be kept to a minimum, the only external calls needed should be to perform serial I/O and TCP/IP socket operations.

4.4.2 The TCP/IP Interface

The commands used over the TCP/IP bear a very similar resemblance to those used over the serial line to the Basic Stamp. However there are a few differences, firstly all commands work in real world units such as degrees, secondly the command format is more verbose using strings like "get compass" rather than "gch" and finally there are no update commands, calling the get command will result in the appropriate update commands being issued to the basic stamp as well as the get command. Table 4.3 shows a full description of commands available over the TCP/IP interface.

Command	Description	Parameters	Returns
Get compass	Gets the compass heading.	None	The compass heading in degrees.
Get wind	Gets the wind direction.	None	The wind direction in degrees, relative to the boat.
Get sail	Gets the sail direction.	None	The sail direction in degrees.
get rudder	Gets the rudder direction.	None	The rudder direction in Degrees.
get northing	Gets the number of degrees north or Y coordinate for simulator.	None	The number of degrees north or a negative value if south of the equator. If using the simulator the Y co-ordinate, which will be positive if above the origin and negative if below.
get easting	Gets the number of degrees east or the X coordiante for the simulator.	None	The number of degrees east or a negative value if west of the Greenwich meridian. If using the simulator the X co-ordinate, which will be positive if right of the origin and negative if left of it.
Set rudder	Sets the rudder direction.	The rudder direction in degrees.	None
Set sail	Sets the sail direction.	The sail direction in degrees.	None
Set unwind	Turns the sail counter-clockwise several times in order to un-wind the wire connecting to the wind sensor.	None	None

Table 4.3: The commands available over the TCP/IP interface.

It should be noted a simplistic TCP/IP interface has been chosen instead of using an existing RPC mechanism such as CORBA/IIOP due to the simplicity of implementing such a protocol from scratch rather than using an existing one and the reduced overheads of a custom design rather than trying to fit into an existing complex RPC mechanism. It should also be noted that although the protocol shown in table 4.3 includes support for a GPS there is no support for this in the protocol shown in table 4.2 as there is no GPS receiver connected to the Basic Stamp. Support is only included in the TCP/IP protocol to allow for logging of X and Y co-ordinates in the simulator and for future support of GPS hardware.

4.4.2.1 Error Handling in the TCP/IP Interface.

Unlike the low level code running on the basic stamp, the medium level code performs validation of the commands it is sent and will return a response beginning with "ERROR" if an error occurs. The word "ERROR" will be followed by an error code and a textual description of the error, the error codes are there to ease error parsing by the high level code. A full list of error codes is shown in table 4.4.

Error String	Meaning
"ERROR-1: Value out of range"	The requested value is too high or too low for this command (set commands only).
"ERROR-2: Invalid Mode"	The command did not begin with get or set
"ERROR-3: Communication Error"	The basic stamp is not responding.
"ERROR-4: Missing or invalid command"	Get or Set were not followed with a valid command.
"ERROR-5: Missing parameter"	This command required a parameter and none was sent.
"ERROR-6: Invalid parameter"	The supplied parameter is invalid, this might be because its not a number, is too long or is too short.

Table 4.4: Error codes and their meanings.

4.4.3 Using the calibration data

As the medium level code must translate between real world data and the values understood by the sensors and servos a conversion of some form must take place. As each sensor/servo has been calibrated five times by recording its real world value against the value understood by the sensor or servo, this data can be used to perform a conversion from one data type to another. There are two possible strategies that could be used to perform these conversions, the first is to generate a mathematical function based upon the calibration data using a statistical analysis technique such as a least squares regression or a reduction to linear form followed by a least squares regression. It is possible to perform such analysis using most modern spreadsheets and there are also plenty of freely available libraries which do this. The other method is to simply store all the data in a lookup table and have the code use the nearest viable value. This method does not allow for any extrapolation of data between data points, and may limit the accuracy and resolution to which values can be specified. However given that most of the hardware components in the robot are not accurate to less than 5 degrees this isn't likely to be noticeable. The advantages of this method are that the data file can be changed and thus the behaviour of the program can be changed without the need to recompile, using the mathematical function approach it is likely that this function would be hard-coded and a recompile necessary if the function changed. The other advantage is that the algorithms to perform a lookup are exceptionally simple and easy to implement. For these reasons this method has been chosen over the mathematical function method, the data tables used are to be based upon the median values from the calibration data. One major problem with this method is that it is unlikely to be suitable on its own for using the compass data, as calibrating the compass requires moving the boat around to each heading and then reading the value from a real compass it is unlikely that intervals of less than 45 degrees are practical, thus it may be necessary to produce a data table for the compass which involves

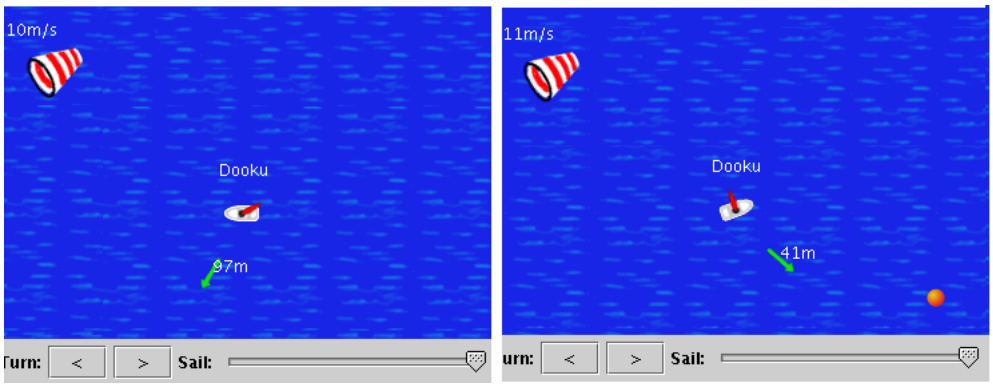

Figure 4.2: Two screenshots of tracksail, on the left the boat is sailing almost into the wind and the sail is only able to go out a small amount despite the slider being set to full. On the right the wind is almost directly behind the boat and the sail is out as far as possible.

the use of interpolated data, as it would be useful to be able to read the compass at a resolution of five degrees rather than 45 degrees.

4.5 The Simulator

As there is a high chance of the robot failing (e.g. Sinking) at an early stage in this project and testing time will be very limited, a simulator is required to allow for testing of the control algorithms. The only suitable application which could be found was an open source multi-player game written in Java called Tracksail [38]. Tracksail works using a client-server architecture and it should be possible to replace the client with code which implements the same TCP/IP interface as the medium level robot code does. This would make the simulator appear to be the same as the real robot to the high level control algorithms.

There are bound to be some areas where the behaviour of a simulator will differ from real life (as is the case with all simulators) and in many cases there will be little that can be done about this. However there are some areas where tracksail's behaviour clearly differs significantly from that of the robot. One such problem is that tracksail automatically limits the maximum angle of the sail depending on the wind direction relative to the boat whereas the robot allows the sail to be placed at any position regardless of wind direction. Tracksail also moves the sail whenever the wind direction and/or course of the boat changes, this is not an unreasonable action as this is what happens to a normal sailing boat. The sail is also set to a value between 0 and 100 using a graphical slider and this is then mapped to a sail angle which is dependent on the wind direction relative to the boat, figure 4.2 illustrates this. In the actual robot the sail behaves rather differently, as it is a solid metal sail being moved by a servo (traditionally sails have to be pushed into place by the wind) it is possible to move the sail to almost any setting regardless of the wind direction, although doing so may cause the boat to alter its course. Mapping this behaviour onto tracksail is not possible without a major rewrite of the code, therefore a compromise has been taken. Tracksail will take a desired angle for the sail from the client code, if this angle is within a range that it could currently meet then that angle is used. For example if the boat has the wind behind and slightly to the left of it then the sail could be anywhere from 180 to 90 degrees in relation to the boat, so tracksail will set the sail to any angle requested in the 90-180 range. However if an angle is specified outside this range the nearest possible angle (180 or 90 in this case) will be taken. Should the boat be sailing closer to the wind then this range will be much smaller.

Algorithm 1 Pseudo code for the sail setting algorithm supplied in tracksail.

```
FUNCTION sail_angle(VAR wind_direction)
    /*wind_direction is the absolute direction of the wind
    direction is the current heading of the boat
    sail_value represents the sail setting between 0 and 100*/

    /*calculate the angle of the sail if the sail is let out fully*/
    sail_dir = (wind_direction-direction+360) MOD 360
    //convert the current 0-100 setting to an angle between 0 and 90
    restrict_angle = 90 * sail_value / 100
    //calculate positions for the sail on left and right tacs
    k1 = (180 - restrict_angle + 360 ) MOD 360
    k2 = (180 + restrict_angle) MOD 360
    //choose between k1 and k2
    IF sail_dir < k1 THEN
        sail_dir = k1
    ENDIF
    IF sail_dir < k2 THEN
        sail_dir = k2
    ENDIF
//return the sail setting
RETURN sail_dir + direction
```

4.5.1 Simulator Modes

Two different modes of operation have been created for the simulator. The first of these is known as "Perfect Mode", in this mode the wind is consistent and the boat will not deviate from its course without an explicit command telling it to do so. The other mode known as "realistic mode" which attempts to simulate more realistic conditions, in this mode the wind will vary in both direction and strength at random and the boat's heading will change at random from time to time. The reason for having these two separate modes is to first develop an algorithm which will behave properly under perfect mode without having to deal with any noise from wind or course variations. This helps to show that the algorithm is behaving correctly and that any unexplained actions are not the result of random events which have been introduced to add realism. Once an algorithm is behaving correctly under perfect mode it can then be tested under realistic mode to show that it is capable of coping with variations in the environment. Hopefully any algorithm which behaves correctly under realistic mode will also behave correctly in the real world.

4.5.1.1 Simulator Sail Setting Algorithm Design

As previously mentioned in section 4.5 tracksail sets the sail via a slider which produces values between 0 and 100. The exact mapping of these values to angles depends on the current wind direction and heading of the simulated boat. In order to allow the sail position to be specified as an angle (as is done in the actual robot) this algorithm must be reversed. Algorithm 1 shows the pseudo code for the algorithm which translates slider values into angles and algorithm 2 shows the reversed algorithm used to convert angles supplied over the TCP/IP interface to values which can be used by tracksail.

Algorithm 2 Pseudo code for the sail setting algorithm used to take the value supplied over TCP/IP and apply it to tracksail.

```
/*desired_direcction is the angle we wish to place the sail at, it may not be
possible to achieve this angle*/
/*limit the requested angle to between 270, 0 and 90*/
IF desired_direction > 90 AND desired_direction <= 180 THEN
    desired_direction = 90
ELSEIF desired_direction > 180 AND desired_direction < 270 THEN
    desired_direction = 270
ENDIF
/*convert from tracksail style angles*/
desired_direction = convertCoords(desired_direction)
wind = getWindDirection()
heading = getHeading()
/*get the maximum angle the sail can be set to*/
sail_dir = (wind - heading + 360 ) MOD 360
new_sail_dir = sail_dir
IF sail_dir < 90 THEN
    new_sail_dir = 90
ENDIF
IF sail_dir < 270 THEN
    new_sail_dir = 270
ENDIF
/*we have now determined the maximum angle the sail can take*/
max = new_sail_dir
new_sail_dir = sail_dir
IF sail_dir < 180 THEN
    new_sail_dir = 180
ENDIF
min = new_sail_dir
/*max should always be more than min select the appropriate one depending on
the desired direction*/
IF max<min THEN
    IF desired_direction < max THEN
       desired_direction = max
    ELSEIF desired_direction > min THEN
       desired_direction = min
    ENDIF
ELSE
    IF desired_direction > max THEN
       desired_direction = max
    ELSEIF desired_direction < min THEN
       desired_direction = min
    ENDIF
ENDIF
/*now we must scale our chosen angle to between 0 and 100 and set the sail to
this as this is system used internally by tracksail*/
range = max - min
value = (desired_direction - min)/range*100
setSail (value)
```

Algorithm 3 Pseudo code for the modified simulator rudder algorithm.

```
/*global variable to represent the rudder angle*/
INTEGER rudder_angle

/*Takes a rudder angle between 270,0 and 90.
0 being centred, 270 being fully left and 90 being fully right*/

FUNCTION setRudder(INTEGER rotate_angle)
    IF rotate_angle >= 270 THEN
       //change the angle to something between 0 and -90
       rudder_angle = (360 - rotate_angle ) * -1
    ELSE IF rotate_angle >=0 AND rotate_angle <= 90 THEN
       rudder_angle = rotate_angle
    END IF
END FUNCTION

FUNCTION rotate(INTEGER rotate_angle)
    /*modify the heading variable (part of tracksail) by rotate_angle
    this  will be either a positive or negative value
    add 360 to prevent problems with values below 0, mod by 360
    to limit output to between 0 and 360*/
    HEADING = (360 + HEADING + rotate_angle) MOD 360
END FUNCTION

/*thread to constantly monitor the rudder_angle variable and perform
appropriate rotation*/

THREAD performRotation
    WHILE true
       rotate(rudder_angle/100)
       /*wait 1 second*/
       sleep(1000)
    END WHILE
END THREAD
```

4.5.1.2 Simulator Rudder Setting Algorithm Design

Tracksail by default allows the user to steer the boat by pressing on a turn left or turn right button, these will add or subtract 10 degrees from the heading. This behaviour is rather dissimilar to the robot where the rudder is placed in a position and held there until the desired heading is achieved at which point it is centred again. Therefore the simulator must be adapted to behave in this manner. In order to achieve this the simulated boat must continue to rotate as long as the rudder is turned, this can be simulated by constantly making small turns with a small wait in between each turn. There needs to be some kind of delay in between each rotation and the magnitude of this rotation needs to be in proportion to the rudder setting.As this rotation needs to constantly occur it must run in a separate thread of execution or be called regularly as part of the main program loop. The pseudo code for this algorithm is shown in algorithm 3.

4.6 Design of the high level code

There are a number of different algorithms which could be used for the high level control system, possibilities include fuzzy logic controllers, PID controllers, neural networks and genetic algorithms just to name a few. Fuzzy logic controllers have been used successfully by Abril, Salom and Calvo (1997) [10] in their sailing robot and by Vaneck (1997) [12] in his powered autonomous boat and therefore would seem a logical choice given this success in similar applications. Abril, Salom and Calvo also conclude that their fuzzy logic controller offers similar performance to that of a PID controller therefore implying that a PID controller is also a worthy system for such a tasks. Given this it was decided to implement both a PID and fuzzy logic controller and to compare their effectiveness in controlling a sailing robot. This presents the scientific question "Which is more effective in controlling a sailing robot, Fuzzy logic controllers or PID controllers?".

4.6.1 Implementation Platform

The implementation platform for the high level algorithms needs to be one which is suitable for writing artificial intelligence code and at representing data structures and mathematical operations. It also needs to support TCP/IP sockets for communication with the lower level code. C could be said to fulfil these requirements although handling data structures in C can at times be confusing in comparison to other languages with dynamic memory allocation and garbage collection (such as Java). These problems could lead to memory leaks, ultimately crashing the system during pro-longed use. The problem with using Java is that as it relies on a byte code interpreter, which will impact performance significantly especially on the slow processor featured in the Psion 5mx. In order to use Java there must also be a working byte code interpreter for the hardware platform/operating system, an attempt was made to compile the Kaffe Java VM to run on the Psion, unfortunately this was not successful and it was decided to implement the artificial intelligence code in C.

4.6.2 Sailing Algorithms

4.6.2.1 Sailing Theory

In order to design algorithms for controlling a sailing robot it is first necessary to look at the general rules behind sailing a conventional sailing vessel. Figure 4.3 shows what are known as points of sail, these are the positions taken by the sail given a certain wind direction. In order to sail with the wind directly behind the boat, the sail is allowed to run out as far as it can so that the sail's angle is perpendicular to the wind direction, this is known as running. The next point of sail is known as a broad reach and occurs when the wind comes from about halfway between the back of the boat and the side (angles of approximately 135 or 225 degrees), on this point of sail the angle between the centre of the boat and the sail will be in the region of 70 to 80 degrees. Should the wind direction be perpendicular to the boat's course the sail should be approximately half way out or at an angle of 45 degrees between it and the centre of the boat, this is known as a beam reach and is generally the most efficient point of sail to use. If the wind is coming over the front of the boat at an angle of up to 45/315 degrees then the sail should be pulled in as far as possible so that it is nearly in line with the centre of the boat, this is known as sailing close hauled or beating. In order to sail towards the direction of the wind the boat must zig-zag (also known as tacking) across the wind with no more than 45 [1] degrees between the direction of travel and the wind direction. Should the difference be less than 45 degrees then

[1] The value of 45 degrees is generally used in sailing manuals, although the reality can be different depending on the boat itself. The software needs allow this value to be selected by the user.

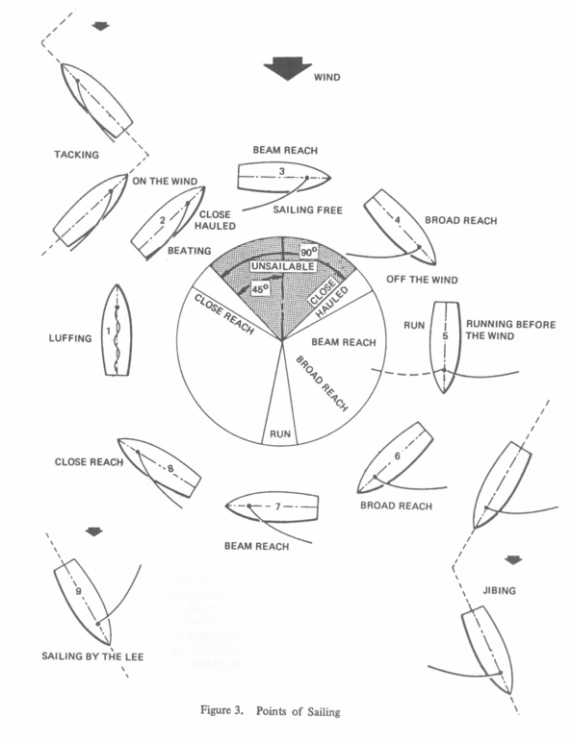

Figure 4.3: Diagram showing the different points of sail. Source: Sailing and the Tech Dinghy [2]

the boat will stop sailing properly and is said to be "in irons" or "luffing", this is sometimes done deliberately as a means of stopping the boat. For all points of sail other than a run the sail needs to be out on the opposite side of the boat to the direction the wind is coming from.

4.6.2.2 The Tacking Algorithm

Both the PID and fuzzy logic algorithms are based around the same basic concept. The algorithm will be responsible for positioning the boat to its desired heading by setting the rudder appropriately until that heading is achieved, once the heading is achieved the rudder must be straightened and the sails set according to the direction the wind is coming from. Should the desired heading be within 45 degrees of the wind direction then the boat needs to sail at 45 degrees +/- the wind direction, as it is not possible for it to sail any closer to the wind. As this means the boat is no longer travelling towards its desired heading it must alternate between the wind direction +/- 45 degrees, this process is known as tacking. This can be achieved by comparing the desired heading with the wind direction, if tacking is necessary then the desired heading must be adjusted to either +/- 45 degrees and this value must alternate on a regular basis to ensure the boat still travels in the desired direction. The algorithm must also take account of how long should be spent on each tack, if the desired heading is exactly the direction the wind is coming from then an equal amount of time should be spent on each tack however if the desired heading happens to be 40 degrees beyond the wind direction then more time must be spent on one tack than the other. A pseudo-code algorithm for doing this is outlined in algorithm 4 and a diagram illustrating the problem is shown in figure 4.4.

Algorithm 4 Pseudo code algorithm for handling tacking.

```
/*function takes the desired course and the current wind direction
as parameters, returns the course which should be sailed*/
function(integer desired_heading , integer wind_dir)
/*variables to represent the range of headings where we must tack*/
lower_limit = ((wind_dir - 45) + 360) % 360
upper_limit = ((wind_dir + 45) + 360) % 360
/*test if the desired course is in that range*/
IF upper_limit > lower_limit
   IF desired_heading > lower_limit and desired_heading < upper_limit
      tacking=true
   ENDIF
ELSE
   /*the upper limit is less than the lower limit
   e.g. if upper limit = 5 and lower limit = 275*/
   IF desired_heading > lower_limit or desired_heading < upper_limit
       tacking=true
       time_on_tack = current_time
   ENDIF
ENDIF
IF tacking = true
   /*calculate how long to spend on each tack*/
   difference = desired_heading - lower_limit

   upper_time = difference/45 * tack_length
   lower_time = (90-difference)/45 * tack_length
   /*calculate elapsed time on this tack*/
   time_on_tack = current_time - start_time
   /*test if its time to switch tack*/
   IF using_upper_tack
      IF time_on_tack >= upper_time
          time_on_tack = 0
          start_time = current_time
          using_upper_tack = false
       ENDIF
      return upper_limit
   ELSE
     IF time_on_tack >= lower_time
       time_on_tack = 0
       start_time = current_time
       using_upper_tack = true
     ENDIF
     return lower_limit
   ENDIF
ENDIF
```

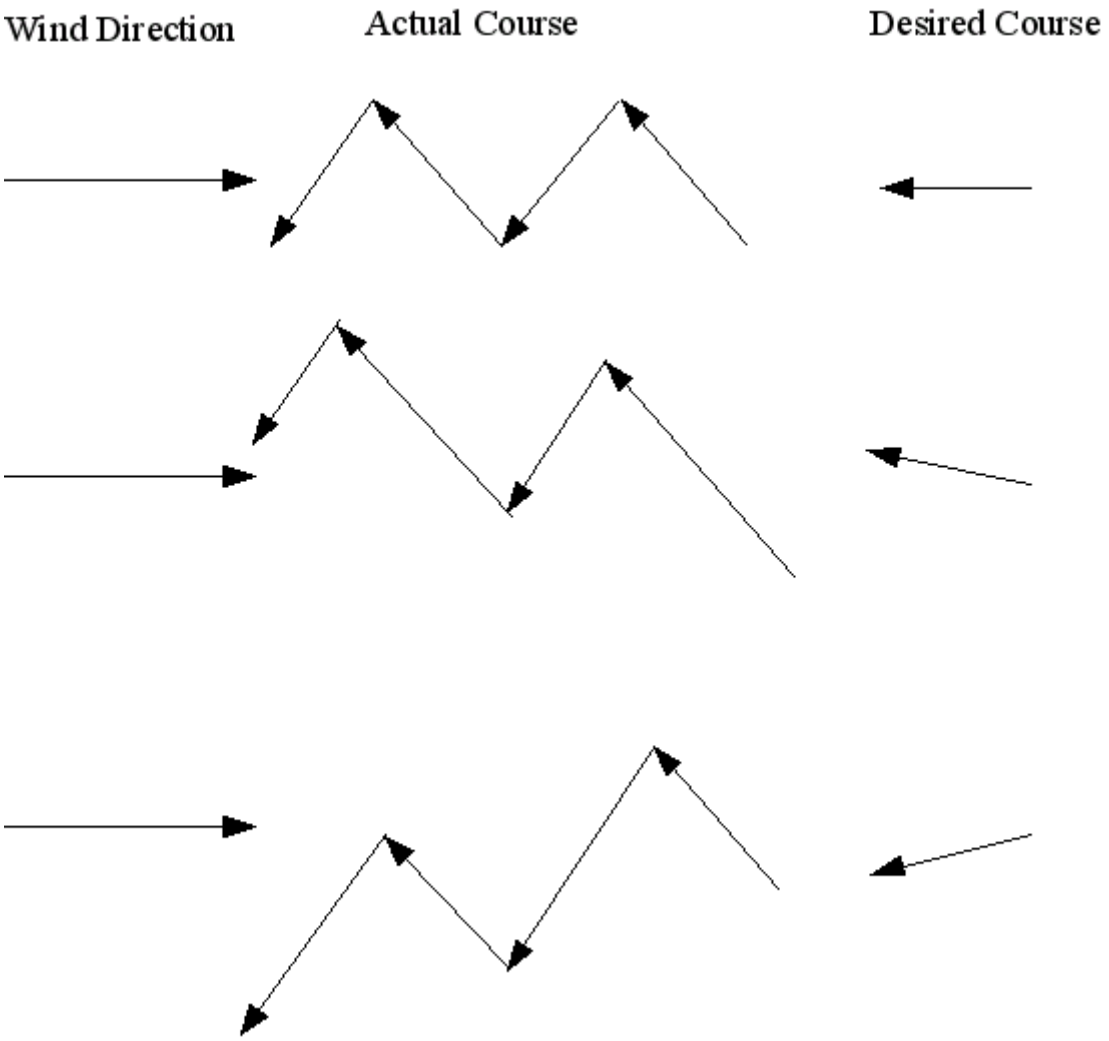

Figure 4.4: Diagram illustrating the situations that the tacking algorithm must deal with.

4.6.3 The PID Controller

A PID controller is a commonly used method in control systems and makes use of the errors observed in a system in order to perform corrections, in the case of the sailing robot this will be in terms of the boat's desired heading and desired sail position. The error is determined by the difference between the current heading and the desired heading or the difference between the current sail position and the desired sail position. A PID controller is made up of three parts known as bands, these are known as the Proportional, Derivative and Integral bands. The Proportional band, as its name suggests makes corrections in proportion to the size of the error, this alone is enough to control many systems however it may encounter problems that as it nears its destination that the force applied is so low that the desired course is never quite achieved (known as the steady state problem) or it may oscillate around the desired level. Integral control takes the sum of all errors and multiplies them by the amount of time elapsed, in doing this the steady state problem can be overcome as eventually the integral control will cause enough of a correcting force to be applied to hit the desired course. Finally derivative control acts as a form of break to prevent the rate of change occurring too quickly and thus overshooting the desired course, this is determined by measuring the rate of change in the error. The final value of a PID controller is calculated by multiplying the output of each band by a pre-determined constant

known as the gain and then taking the sum of all of the three bands. The gain constants are dependant upon the system and a PID controller must be tuned in order to find optimal settings for the gain constants. A general equation for a PID controller is shown in figure 4.5 and Pseudo code is shown in algorithm5.

$Output = KpE + Ki\Sigma(E\Lambda t) + Kd\frac{\Lambda E}{\Lambda T}$

Kp is the proportional gain
Ki is the integral gain
Kd is the derivative gain
E is the current error

$\Sigma(E\Delta t)$is the sum of all past errors (the integral of the error)

$\frac{\Delta E}{\Delta t}$ is the rate at which the error is changing (the differential of the error)

Figure 4.5: General equation for a PID controller source Modern Control Technology page 381 [3].

Algorithm 5 Pseudo code algorithm for a PID controller.

```
/*setup the timer for the integral band*/
previous_time = current_time
/*setup previous error*/
previous_error = 0

WHILE true
  /*calculate the proportional band*/
  proportional =  error * prop_gain

  /*integral band*/
  time_diff = current_time - previous_time
  previous_time = current_time
  integral = (error_sum + (time_diff * error)) * integral_gain


  /*derivative band*/
  derivative = (error - previous_error) * derivative_gain

  /*increase the error_sum*/
  error_sum = error_sum + error

  output = proportional + integral + derivative

  set_course(output)

END WHILE
```

4.6.4 Fuzzy Logic Design

4.6.4.1 Fuzzy Logic Controller background

Fuzzy logic is based on the principle of approximate reasoning and attempts to follow a system similar to that followed by humans where every decision is based upon things being approximately correct given the action performed. This is the antithesis of many computer based systems which work by very exact rules with clear-cut boundaries. Fuzzy logic controllers are based on the principle of fuzzy sets which are similar to a traditional sets aside from one major difference, each member or potential member of the set is given a probability to express the extent to which they belong to the set, this probability is defined by a membership function for each set. An example of such a set might be to determine if a given heading is to the left of another heading, if there is only a few degrees difference then the probability of membership will be quite low whereas if the difference is 90 degrees then the probability of membership will be very high. In this example the membership function might be a linear equation where 90 degrees corresponds to a probability of 1 and zero degrees corresponds to a probability of 0.

In order to use fuzzy sets to produce a fuzzy logic controller, inference rules are devised which link membership of one set to another. These rules are written using natural language and contain phrases such as "if desired heading is to the left then turn the rudder left". In order to give an output which can be used in a control system (such as the number of degrees to turn a rudder) the probability of membership for the first set is calculated from an input value (in this example the number of degrees left the desired heading is), this process is known as fuzzification. Usually there are several input sets and the probability of membership for each of these will be calculated. Once the probabilities of membership for the input sets are determined the input rules are matched to a corresponding output set, in the example this will be that membership of the heading to the left set implies membership of the turn rudder left set. The probability of membership for the input sets is then applied to the corresponding output set and from this the value which has that probability of membership can be identified, this process is known as defuzzification. Going back to the rudder example, if the probability of membership for the input set were 0.5 then the value who's probability is also 0.5 on the output set is located, this maybe 70 degrees therefore linking being 45 degrees off heading to needing to turn the rudder to 70 degrees.

4.6.4.2 Fuzzy logic code design

In order to implement a fuzzy logic controller in a computer some system must be designed for representing set membership information and performing inferences. One obvious approach is to determine the membership functions for each set and then implement these in code however this presents the problem that the code must be altered and recompiled every time a change is required to be made, this maybe somewhat difficult to perform when actually testing a sailing robot. Another approach would be to represent the membership functions in a file which could be edited thus preventing the need to recompile in order to reprogram the fuzzy logic controller. The membership function could be represented either as a data table or as some kind of equation description which must then be interpreted by the code, if it could be guaranteed that only linear equations would be used then this could simply be the gradient and intercept of a linear equation. It was decided the most appropriate system to use was the data table method as this presented the fewest coding challenges, is similar to the calibration system specified in section 4.4.3 and finally as it allows for the set specification to be changed without recompilation and at the same time does not restrict sit membership functions to being linear. The most obvious format for these tables are to list probabilities in one column and the corresponding values in another.

Usually probabilities are specified between 0 and 1, however as floating point calculations add complexity and must be performed in software on many microcontrollers it makes more sense to represent probabilities as integers between 0 and 100.

4.6.4.3 Fuzzy Logic for rudder control

The rudder control logic has three possible outputs: turning left, going straight ahead or turning right. There are also three corresponding inputs which represent the current heading deviation, they are: too far left, on course and too far right. The inference rules are shown in algorithm 6.

Algorithm 6 Fuzzy inference rules for rudder control.

```
IF too far left THEN turn right
IF on course THEN straighten rudder
IF too far right THEN turn left
```

4.6.4.4 Fuzzy Logic for sail control

The sail setting algorithm needs to follow rules which link wind directions to points of sail similar to those shown in figure 4.3. The algorithm needs to take the wind direction as an input and the sail position as an output. The input sets can be divided up into different wind directions and the output sets into different sail positions. The input sets are therefore: wind across front, wind at side and wind behind. The output sets are: all the way in, half out and fully out. It was decided not to introduce a 4th set to represent a broad reach, where the sail is three quarters of the way out as the robot's sail and wind sensor are not accurate enough to justify this. The inference rules are shown in algorithm 7. As there are two different angles where each rule is true the sets must be duplicated for the wind coming over the left hand side of the boat and the sail being on the right hand side as well as the opposite situation, this gives rise to sets with the words left and right appended to their names. Full graphical definitions of the fuzzy sets can be found in appendix K.

Algorithm 7 Fuzzy inference rules for sail control.

```
IF wind across front right THEN sail is all the way in on left
IF wind across right side THEN sail half way out on left
IF wind behind on the right side THEN sail is all the way out on left side
IF wind across front left THEN sail is all the way in on right
IF wind across left side THEN sail half way out on right
IF wind behind on the left side THEN sail is all the way out on right side
```

4.7 Data logging

In both the simulator and real world environments it is necessary to log data about the course taken by the boat as defined in the requirements specification section 2.3.6. The data needs to be recorded often enough to show all of the moves taken by the boat and to effectively allow them to be replayed for later analysis. The most obvious way to produce a log file is using a plain text format where each line represents a given point in time and each column a different variable. The variables which are required are the elapsed time since the log began, heading,

heading error, observed wind direction and X/Y (or Northing/Easting) coordinates. As there is no support to produce a Northing or Easting from the robot these will simply be written as zeros.

4.8 Other Design Issues

At this point in time there is no intention to implement a full scale bi-directional communications system, data logging system, collision avoidance, fleet management, fault diagnostics/compensation or remote software upload facilities. This is due to the limited time available on this project. Any "full" implementation of a sailing robot should include these features.

A problem has been identified with the wire connecting the wind sensor on top of the mast to the Basic Stamp, the problem very simply is that with enough turns of the mast in a single direction the wire will eventually become wrapped around the mast and could eventually break causing wind information to be lost. The software needs to keep track of how many turns the mast has made and if the wire is becoming tight it needs to unravel the wire by turning the mast around several times in the opposite direction. Doing this is likely to set the boat off course and the software will have to recover the boat to its previous state once the operation is complete. The code to handle this could potentially be implemented at any level in the software, however if the high level code were to implement it, then the effects would be seen in both the simulator and on the robot.

Chapter 5

Calibration methods and results

The calibration of the sail servo, rudder servo, wind sensor and compass aims to produce tables of data allowing for the conversion between the everyday units associated with these devices (in all cases this is in degrees) and the values actually understood by the electronics. Before these "official" measurements were taken, a series of practice runs were made to establish the range of values used by the sensors and servos, these values were then used as the basis for a specialist calibration program which cycles the servos through all their possible values and for the sensors requests that the user place them in given positions.

For the sail servo, rudder servo and wind sensor a small program has been written to place the sail/rudder into one of several positions and then prompt the user to enter the actual angle of the sail or rudder. Wind measurements are made by rotating the sail through 360 degrees, while keeping the wind direction constant, this actually gives the wind values for different directions as the wind sensor is rotating rather than the wind. The actual code will have to take this into account to determine the wind direction relative to the whole boat rather than to the sail. The compass must be calibrated by constantly turning the boat and entering readings from an ordinary compass against those taken from the on-board compass. To reduce the chance of anomalies causing errors in the calibration, each measurement will be performed a total of five times. The software to perform the calibration works by running on a laptop connected to the basic stamp, it records all necessary data into a plain text data file. The plain text data file can later be read by a spreadsheet in order to calculate medians for each data set which in turn can be used to produce the calibration tables used by the actual control programs.

5.1 Rudder Calibration

The rudder was calibrated by placing a piece of paper containing a printout of a protractor (which can be found in appendix B) with lines drawn every five degrees underneath the rudder and setting the rudder to a number of different positions and recording the angle at each position. The zero degree line of this piece of paper was aligned to point towards the front of the boat. A total of eleven different values were sent to the rudder servo and each measurement was retaken a total of five times. The program which set the rudder position would prompt the user to enter the current angle of the rudder before it moved onto the next position, these angles were recorded in a data file against the values which had been sent to the servo, a full copy of this data is available in Appendix F. Before the angle measurement was taken, but after each movement of the rudder had taken place, the rudder was manually pushed to either the left or right (to the left when on the right side of the boat and vice-versa) as it was decided that the action of the water flowing against it would do similar. Typically this moved the rudder about two or three degrees.

5.2 Compass Calibration

Compass calibration took place by placing the boat on top of a box on which it could be rotated, turning the boat to a given heading, taking several readings from the compass. This was repeated with the boat at different headings of45 degree intervals. A photograph of this setup can be found in figure 5.2. As this data only gives a 45 degree accuracy and the relationship between the values read from the compass and the heading is linear, a least-squares regression was performed upon the recorded data in order too produce a general function describing the relationship, this function is shown in figure 5.1. This function was then used to derive a data table with 5 degree accuracy, 5 degree accuracy was chosen as the data sheet for the compass [39] specifies an accuracy level of approximately 10 degrees.

if $v < 97$
$c = 1.64v - 158$
if $v >= 97$
$c = 1.64v + 202$
Where c is the number of degrees and v is the value returned by the Basic Stamp.

Figure 5.1: Formula to convert compass values.

5.3 Sail and Wind Sensor Calibration

Sail and wind sensor calibration was undertaken by having a small program loop through all possible positions for the sail and prompting the user for the angle of the sail at each of these and recording these responses as well as the wind sensor value. The angle of the sail was measured by placing a paper protractor (see appendix B) on the deck of the boat, upon this piece of paper there were a series of lines at five degree intervals, a piece of copper wire was also attached to the base of the mast to allow an angle to be read for each position, figure shows this setup. A simulated wind was provided by a fan placed at the height of the wind sensor and at a 45 degree angle to the boat as illustrated in figures 5.4 and 5.5 .

Figure 5.2: The compass calibration setup. By co-incidence the right side of the room shown is almost exactly magnetic north.

Figure 5.3: The setup for measuring the angle of the sail. A piece of copper wire is attached to the mast below the sail and facing the same direction as the sail, although its a bit difficult to see in this image. The piece of paper on the deck has angles marked out every five degrees, with zero degrees facing towards the front of the boat.

Figure 5.4: The setup for wind testing. The fan was later moved to be at 45 degrees to the boat.

Figure 5.5: A more detailed view of the wind sensor on top of the mast.

Chapter 6

Implementation Discussion

6.1 Overview of the implementation

6.1.1 Basic Stamp Implementation

The Basic Stamp code is responsible for processing commands sent over the RS232 line and then performing the appropriate action with the sensor or servo and finally returning any data or indicating that the command has completed. A fully working program was already supplied by Dr. Mark Neal at the start of the project, however it has undergone significant changes as shown in appendix D. The original code worked upon the principle of allowing concurrency by using a series of update commands which requested that a sensor updated itself and store the value in memory for later collection via a get command, this principle also applied to setting servos and the set commands would instantly reply before the servo had finished moving. The key problems with this approach are that there is no way for the code running on the Psion (which communications with the Basic Stamp) to know when a set command has completed or when a sensor has finished obtaining its reading, this approach also requires a lot of extra commands to be sent over relatively slow serial lines. If these two problems didn't exist then this approach would work very well to facilitate concurrent operation of different sensors and servos. In order to solve these problems the original code was heavily edited so that no set commands would return any data until they had completed the movement of the servo and get commands would no longer require updates to have first been requested. This removed the problem of having to guess when the Basic stamp had finished a task and allowed much quicker response times to events. Previously the Psion code was forced to wait the maximum possible time for a command to complete before issuing the next command, this was especially problematic with sail setting code as moving the sail 180 degrees takes several seconds while moving it one degree takes takes less than one second. It was found that if the one command was accidentally issued before another had finished then the second command would never be executed by the Basic Stamp and the Psion would then wait indefinitely for a response effectively crashing it.

Once the Basic Stamp code had been changed so that commands did not signal their completion until it had actually taken place, performance dramatically improved. However performance was still poor as it was found that a delay of approximately one second was still required between commands and setting the sail could take up to 15 seconds depending on how far the sail had to move. The supplied code rotates the sail in a three-point control system, by moving it a small amount then testing if it has reached the desired point and then moving on again, if it overshoots the desired position it will then move in the opposite direction, once the desired position is achieved it stops. The size of the motion is the same regardless of how far the sail current is from its desired position. It was decided that this problem could be solved using a PID

	Normal Method	PID Controller	Normal method with inter-turn delays reduced	PID Controller using small turns.
Time to return from 180 degrees to 0	9.87 seconds	10.98 seconds	8.35 seconds	7.35 seconds
Time to move from 0 to 270 degrees	4.53 seconds	3.05 seconds	3.11 seconds	4.14 seconds
Time to move from 270 to 90 degrees	8.32 seconds	5.80 seconds	6.24 seconds	9.74 seconds
Time to move from 90 to 180 degrees	7.32 seconds	5.18 seconds	10.5 seconds	5.32 seconds
Total	30.04 seconds	25.01 seconds	28.2 seconds	26.55 seconds

Table 6.1: Table showing the affects of using a PID controller to rotate the sail.

controller running on the Basic Stamp as this would produce large moves when a large distance needed to be covered and smaller moves when smaller distances had to be covered. The code was based around the original sail movement code and still used exactly the same parameters allowing the higher level code to remain unchanged. This could have theoretically been implemented at a higher level by having the higher level code constantly monitor the sail position and request each individual move itself, however given that each of these moves requires an additional delay of one second this is impractical. The general result of the PID controller was a slight improvement in sail turning times however problems where encountered with overshooting and despite plenty of tweaking of constants it was not possible to completely remove this. A slight modification to the PID algorithm did manage to reduce this problem, by reducing the size of the proportional constant when the required distance was small the overshoot problem was dramatically reduced. It is possible that the same affect could be achieved through proper tuning of the system, however given the poor repeatability of the system and the time it takes to tune this approach was found to work better.

A small experiment was conducted to test the performance of the PID controller in comparison to the original algorithm. This test was conducted by requesting a turn and measuring the amount of time until the command was signalled as having been completed. Four tests where performed, the first moves the sail from a position of 180 degree to 0 degrees, the second from 0 to 270 degrees, the third from 270 to 90 degrees and the fourth from 90 to 180 degrees. The test was performed a total of 5 times and the mean result taken. The results of these tests are shown in table 6.1. As the results show using a PID controller the turn time was reduced in most cases and overall by 5 seconds or 17%. However the PID controller seems to have had trouble positioning the sail back to zero (it kept overshooting) and took longer than the original algorithm to do this. By using the reduced turn size when nearing the desired point this problem was reduced but the PID controller still performed worse than the three-point controller in this case.

6.1.2 Server Implementation

The server code runs on the Psion and is responsible for taking commands via a TCP/IP socket using the protocol outlined in table 4.3, translating the values specified in these commands to values understood by the Basic Stamp, sending a command to the Basic Stamp and if appropriate returning the result to the client. The server is written entirely in C and consists of four files. The first of these is sail.c which is responsible for serial port communications. The second is

calibrate_data.c which is responsible for converting values using a series of lookup tables based upon the calibration data, commands.c which is responsible for determining which command has been requested, requesting that any values be converted between real world measurements and values understood by the Basic Stamp and finally to call the appropriate function in sail.c to send commands to the Basic Stamp. The final file is server.c which contains the main function for the server program and is responsible for listening on the TCP/IP socket and passing any input onto calibrate_data.c.

6.1.2.1 Inter-Command delays and Caching Rudder and Sail values

Two major deviations from the original design were made in the server code these are the introduction of caching on the sail and rudder commands and the addition of a 1 second minimum delay between issuing each command to the Basic Stamp. Each time the sail or rudder is set the value which it is set to is cached, so if a later call is made to request these values rather than sending a command to the Basic Stamp the value cached in memory is sent. In addition to this if a request is made to set the sail or rudder to the value held in the cache then no command is sent as it can be assumed that the sail or rudder is already in this position. The inter command delay was introduced because a problem where commands arrived over the RS-232 connection too quickly for the Basic Stamp to react. The Basic Stamp would actually claim it was ready to receive new commands, but would crash upon receipt of a new command unless approximately one second had elapsed. The true cause of this problem was never established (the debug code is suspected of slowing down the Basic Stamp), however a work around was created by enforcing a mandatory one second delay between two commands being sent over the RS-232 line (or more specifically the results of one command being received and the next command being sent). Unfortunately this had an adverse affect on the time taken for the control algorithms to respond to changes in the environment. For example the code would first read a compass heading and wind direction, then set the rudder and finally set the sails. This meant that 4 seconds would be spent simply waiting on the Basic Stamp in performing regular tasks, when added to the time it takes to set the sail (up to 15 seconds for large moves) the code ran into problems of not being able to react to changes such as course deviations quickly enough. The result of these actions significantly improved the response times of the control algorithms, however the only full solution is to use a single microcontroller capable of offering the necessary I/O facilities and processing power in a single unit or offering high speed inter-device communication. This problem is also exacerbated by the slow speed at which the sail servo moves and the inaccuracy of the sail position sensor.

6.1.2.2 Automatic Wind Sensor Recalibration

In order to transport the robot without damaging the wind sensor it was necessary to remove the plastic wind vane from the potentiometer and reconnect it after transportation, in doing so the vane would be placed with the potentiometer rotated to a different position making the calibration data invalid. This problem was solved by using the calibration data to produce a general linear equation to describe the relationship between the potentiometer's output value and its angle. A small program was then produced which asked the user to place the wind sensor at an angle of 0 degrees, it then took a reading from the potentiometer and recomputed the calibration tables based upon the derived equation, taking the current position as 0 degrees and recalculating the intercepts of the linear equation. This did have the slight side effect of causing a loss of precision with certain wind directions as the potentiometer gives a few values which are not part of the linear relationship as it approaches its maximum value. The calibration data had been able to cope with this, however modelling this in a linear equation (or even two linear

equations) proved difficult and the result was a small region in which the wind sensor was less accurate.

6.1.3 Simulator Implementation

The simulator implementation required the production of an alternative TCP/IP server written in Java, this manipulates a simulated boat based upon command received over the TCP/IP socket. In order to implement this a new class had to be added to the simulator to provide similar features socket listening and command parsing features similar to those of server.c and commands.c in the server implementation. Once a command was processed some of the internal variables of tracksail would either be modified or returned depending upon the nature of the command. The get functions proved relatively easy to implement as all that was required was for the result of a get method to be returned to the client. However both the set sail and set rudder commands proved much harder to implement. As shown in section 4.5.1.1 and 4.5.1.2 these algorithms are relatively complex and took considerable time and effort to implement. The sail algorithm remained very much similar to that specified in the design however the rudder algorithm required some changes as the rotation was not very smooth when movement was only undertaken once per second. Instead a system was devised that would vary the amount of time between each turn and then make turns of one degree. This gave a much smoother and more realistic feel to the rotation of the boat which and aided in producing of high level algorithms.

One problem encountered in adapting tracksail was the way in which it interpreted the angle's for headings and wind directions. The default behaviour was to consider 0 degrees as being to the right, 270 degrees as being upwards, 90 degrees as downwards and 180 degrees as being to the left. Although having this all the wrong way round doesn't actually impede the ability to test algorithms it is very confusing when trying to observe behaviour and work out if an algorithm is behaving as expected. As a result of this problem a method was added to convert from tracksail angles to "logical angles" where 0 degrees points upwards, 90 degrees points left, 270 degrees right and 180 degrees down.

Another problem which proved to consume large amounts of time during testing was that tracksail requires the user to start both a client and server, then using the client enter a player name, choose a track to sail and click start game. These restrictions prevented automated testing if the client was to be restarted in between runs, which was desirable in order to reset the boat to the same conditions every time. To remove these restrictions the startup screen code was reconfigured to automatically enter the username, select the track and start the simulation. To further aid automated testing a shell script was produced to launch both the tracksail client and server and the unwind command was adapted to signal that tracksail should exit.

As defined in section 2.3.3 the simulator needs to implement both a perfect mode and realistic mode. Perfect mode is where the wind strength and direction do not vary and the boat will remain on course once its initial position is set. Realistic mode is where the wind will vary in strength and direction and the boat will deviate from its initial course as a real sailing boat would. Tracksail works on a basis of defining a number of tracks which contain both the points which players are meant to race around and a definition of the winds strength, direction and variability. The track definition is of little importance to the simulation as no attempt is being made to follow the defined tracks. The wind definitions are of great importance, in perfect mode the wind must be set to a totally constant setting with no variability. In realistic mode these settings need to be changed to be highly variable, making this change is simply a matter of changing a few numbers in a text file. Making the boat change heading is implemented as part

of the rudder setting algorithm, a random variable is used to decide if any change is needed, the direction this change and its size if a change is made the heading is modified by the desired amount. A command line argument to the tracksail server is used to determine if this code is to be active or not.

6.1.4 General Client Implementation

Despite there being two separate implementations (fuzzy logic and PID) of a client there is a significant amount of common code between them. These common features can be split into two areas, firstly there is the code which deals with communication to the server via a TCP/IP socket and secondly there is some code which determines the difference between the current and desired heading, determines the heading to follow if sailing into the wind, processes command line arguments and performs logging to a file.

The communication code is responsible for sending commands in the format specified in table 4.3 and provides a series of general functions such as getRudder, getSail, getHeading, getWindDirection, setSail and setRudder which provide an easy to use interface to this code. These are all placed within a file called client.c which is linked with both the fuzzy logic and PID code at compile time. The other common code is placed in a file called control_common.c and is responsible for determining the error between the current heading and the desired heading and to deciding which heading to follow when tacking. Between them these functions implement the algorithm specified in algorithm 4. There are also two other functions for processing command line arguments and performing logging. One slight issue with the command line argument processing is that the parameters differ slightly between the PID controller and the fuzzy logic as the PID controller must be supplied with gain constants, while the fuzzy logic does not require them. This causes a slight issue if the user forgets to give these arguments as no checks are made for them, checks are made the common arguments which include the amount of time to spend on each tack, the angle between the heading and the wind at which tacking will be performed, the number of seconds to run the program for, the desired heading and whether or not the sail should be unwound when the program finishes.

6.1.5 PID Implementation

The PID implementation was split into two separate implementations, the first only implemented proportional control and the second proportional, integral and derivative control. It was decided that PID control of the sail could not be performed as it would be too dependant upon communication between the Basic Stamp and Psion which do not perform fast enough to make this practical, as discussed in section 6.1.1 the PID implementation for sail control was moved to run entirely within the Basic Stamp. However this code still required the Psion to supply a sail setting, initially this was performed with a series of "if" statements which analysed the wind direction and set the sail accordingly however this proved to be difficult to program and produced messy code. A more elegant solution was undertaken in the form of a lookup table which specified the sail position for different wind directions and used the calibration system to calculate a sail position for a given wind direction the contents of this table can be found in Appendix J. Rather ironically this solution is very similar to the implementation of the fuzzy logic controller except that it is based on crisp sets where there is only one possible output for each wind direction.

$K_P = 0.6K_P$
$K_I = 2/T_C$
$K_D = T_C/8$

Figure 6.1: The equation for tuning a PID controller using the Zeigler-Nichols Ultimate Cycle Method, source Modern Control Technology page 381 [3].

K_P	0.6
K_I	1
K_D	0.25

Table 6.2: The results of performing the Zeigler-Nichols Ultimate Cycle PID Tuning with the simulator.

6.1.5.1 Simulator PID Tuning

PID tuning involves finding the ideal settings for the proportional, integral and derivative gain constants. The ideal setting will converge quickly to the desired state and once at the desired state will not oscillate from it. In terms of positioning a sailing robot this means converging to the desired heading and staying on that heading. In order to discover this in the simulator a test was setup whereby the simulated boat had to turn to a heading of zero degrees from an initial heading of 70 degrees and once on course it must hold the course without deviation.

An attempt was made to tune the PID controller using the Zeigler-Nichols Ultimate Cycle method (also known as the Continuous-Cycle method) as specified by Kilan [3]. This method involves initially setting the proportional constant to 1 and the other two constants to 0 and gradually increasing the proportional constant until a constant amplitude is seen in the oscillations. Once this is complete the variable T_C is calculated as the time (wavelength) between two oscillations and is used to calculate the three constants using the formula show in figure 6.1, the system can now be considered tuned.

It was found that a proportional constant of one gave a regular oscillation every two seconds (figure 6.2 shows the graph for these settings). The results of the tuning are shown below in table 6.2. Based upon general observations these values did not seem to be particularly appropriate as a large amount of oscillation was still occurring and although the oscillation reduced as time went on it still took nearly a minute to eliminate itself, attempts to alter these values slightly to counteract this did not see any great improvement. An additional test was performed to investigate what happened when the proportional constant was increased to values of two and three (and the integral and derivative constants left at zero). A proportional gain constant of two showed a quicker convergence rate than a constant of one and also showed that the controller converged on the desired course but continued to oscillate with a similar frequency and amplitude as a constant of one had done. Increasing the constant further to a value of three yielded an interesting result, the system entered a stable state and converged quickly and correctly to the target heading. The results of these tests are shown in figure 6.2. It was therefore decided to use the values shown in table 6.3 for all further testing.

K_P	3
K_I	0
K_D	0

Table 6.3: The results of performing a trial and error based tuning with the simulator.

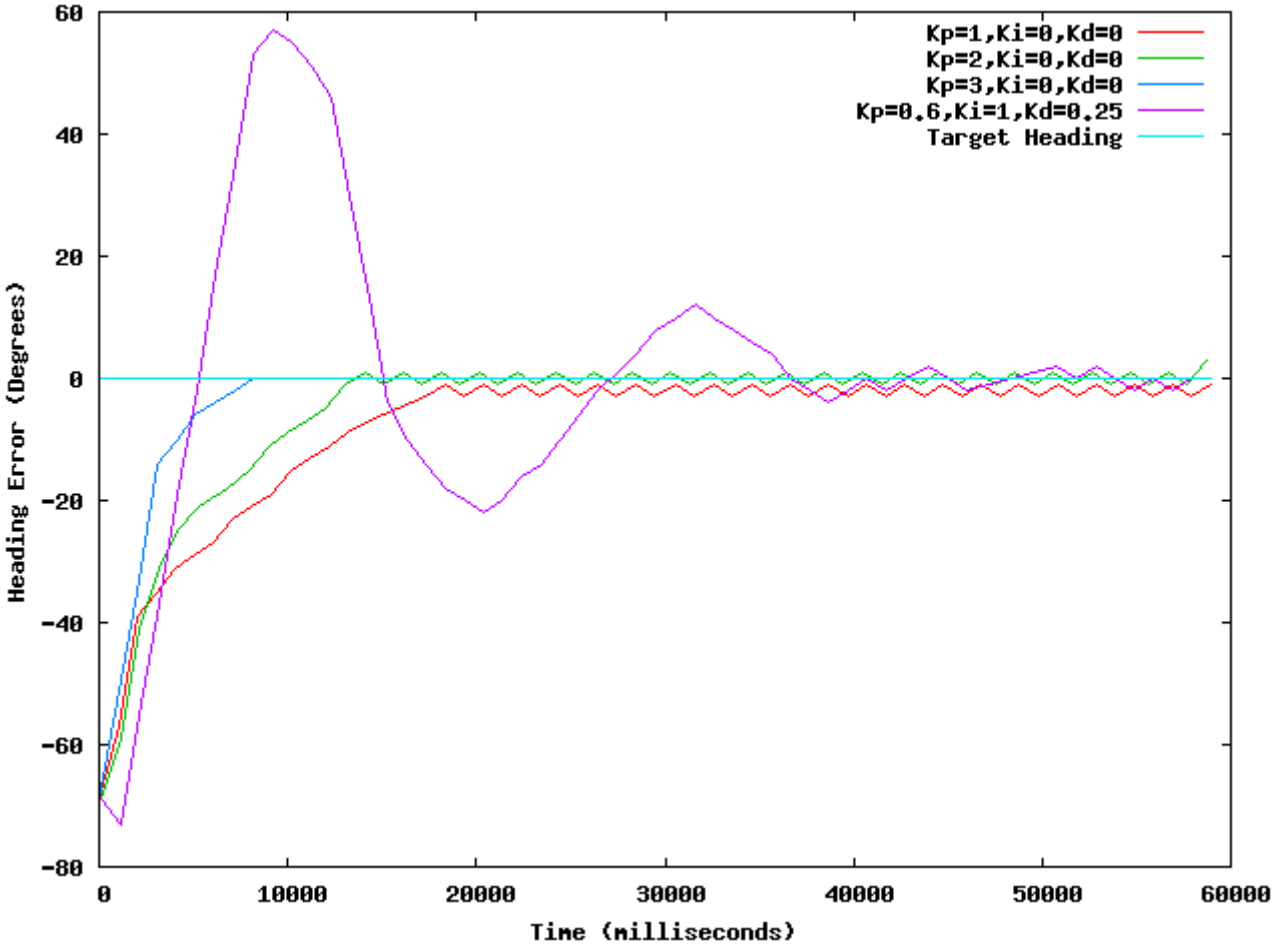

Figure 6.2: A graph showing the current heading error against time for the PID tuning attempts. The target heading was zero degrees and the start heading 70 degrees.

6.1.6 Fuzzy Logic Implementation

The fuzzy logic implementation consists of only a single file called fuzzy.c. This file contains code which implements the fuzzy rules specified in section 4.6.4 for controlling both the rudder and sail as well as performing lookups in a series of data tables which define the fuzzy sets. As specified in the design each fuzzy set is defined by a data table giving a probability between 0 and 100 along with an associated value. These tables are implemented simply by writing the probability followed by a space and then the value onto a line within a plain text file. Each fuzzy set which is defined in the design has its own associated table file, these are loaded into an array when the fuzzy logic controller starts up. Two functions allow lookups to take place within the fuzzy sets, one returns the probability that a given value is in the specified set and the other defuzzifies a value to probability for a given set. The inference rules for both the rudder controller and the sail controller are written into two other functions which rely upon these lookups in order to make their decisions. Rudder decisions are made upon the basis of the current heading error using the same code in control_common.c which the PID controller makes use of. Sail decisions are made upon the basis of the current wind direction.

6.1.7 Linux Configuration

The Psion was configured with a small Linux distribution based around a pre-compiled kernel from the OpenPsion project [35] and the executables from the uClibc/uClinux distribution. uClinux is able to use a program known as busybox which provides a single executable file which can replace most common unix commands with cut down versions, this allowed a sufficiently capable Linux distribution to be build in approximately a megabyte. This was then placed on a compact flash card along with cross compiled versions of the program code and the Arlo boot loader that allows a Psion to bootup Linux.

It was found that during real world operation and simulator testing that it was necessary to vary the heading parameter of the control algorithms in order to allow the boat to sail back to where it started and that this would vary depending upon the wind direction. Rather than writing extra code into the control algorithms it proved easier to write a simple shell script which invoked the control algorithms several times with varying headings. The shell scripts where also used to automate the startup of the simulator and server code. In doing this configuration of the boat's course was greatly simplified and the need for specialist code was replaced by a few simple lines of shell script. The success of doing this reinforces the case for using Linux to control the robot, had a simpler operating system (or no operating system) been used then there may have been a need for large amounts of custom code to perform the same tasks. The infra-red support was implemented by first compiling the IRDA-utils package which provides userspace utilities to initialise the infra-red subsystems, an init script was added to allow the infra-red system to be automatically started at boot (a copy of which is available in appendix L), an instance of the getty program was also bound to the infra-red port this caused a login dialog to be displayed when an infra-red client first connected and upon login presented the user with a standard command shell interface from which they could perform any task which would have been possible had they been using the Psion's keyboard and screen directly. If the infra-red communication was broken and the re-established the user would be taken back to the login prompt and all processes which they had launched would be stopped as soon as communication was lost. This proved to be problematic as it caused the control programs to exit as soon as the infra-red client was moved, to solve this the control programs were started via a shell script which spawned a new virtual console for them to run on and thus prevented them from stopping upon a loss of communications. Although infra-red communication reduced the difficulties encountered in altering parameters once the Psion had been sealed inside the boat, it was by no means a perfect solution as the remote system could be no more than a few centimetres away from the boat and it was highly sensitive to the angle, communication was made harder by putty (which was used as a sealant) obscuring the view through the perspex decking. Ideally some kind of radio based system such as 802.11 or even Bluetooth is needed, unfortunately the Psion cannot support either of these without hardware modifications.

A number of problems were encountered in using the Psion as a result of its poor quality serial port connector, it was therefore decided to replace it with an HP Jornada 720 PDA (for more details on this see section 7.5.2.3). The Jornada was able to offer 802.11 wireless support and thus reduced the problems of remotely configuring the software. However it was not possible to do this using the same Linux distribution which had been used on the Psion as there was no wireless support built-in to uClinux and compiling wireless support for it was unsuccessful. Instead a distribution known as Familiar Linux [40] was installed, this included full support for wireless networks and included DropBear a small SSH (Secure SHell) server which allowed remote logins' to the Jornada. Familiar Linux is not based around uClibc, but rather glibc (the full version of the standard C library) as a result of this the uClibc cross-compiler would not produce executables for Familiar Linux and a new cross-compiler aimed at Familiar Linux developers had to be used to recompile all of the control software.

Chapter 7

Testing

7.1 Software Component Testing

7.1.1 Testing the Basic Stamp Code

The Basic Stamp code was tested via a simple menu program which allowed the user to perform any of 7 functions, these were set the sail position, set the rudder position, get the sail position, get the rudder position, get the wind direction, get the compass heading and unwind the sail. When the user requested one of these options it would send the corresponding command over the serial cable to the Basic Stamp and then display it's response on the screen. This could be used to test the Basic Stamp code independently of the rest of the program.

7.1.2 Testing the Server code

The server code was tested by using the telnet program and a special version of the menu program used with the Basic Stamp (known as netmenu) to issue commands over a TCP/IP socket and monitor the responses. By sending commands over the TCP/IP socket additional data was generated on the serial line which connects to the Basic Stamp and no return data would be sent over the TCP/IP socket until a response was read from the serial line. This presented two possible methods for testing, either connect the Basic Stamp and make use of the values it received or connect the serial port to a terminal on a PC and simply type in random data in response to each command. In the end both methods were used, the PC method was more convenient for regular tests but didn't give realistic values or response times as the Basic Stamp did. The server code is not only responsible for sending what is received on the TCP/IP socket to the Basic Stamp and vice-versa but also for converting any values between those understood by the hardware and real world values such as degrees which can be easily understood by humans. In order to test that sensible values were being sent by the server code the translation code was tested separately by creating a main function to the C file responsible for this and using that to pass in values and see the corresponding values on screen, manual lookups in the data tables were then performed in order to check these values correctness.

7.1.3 Testing the Common Client code

The client code was perhaps the most complex part of the system to test as it consists of a two different client implementations in addition to a number of common functions for interfacing to the TCP/IP socket and providing other common functions such as determining heading errors. The

TCP/IP interfacing code was tested using the netmenu program which as described in the previous sections allowed the user to select any one of seven common functions and send commands to the TCP/IP server, the code for performing this was linked to the same socket interfacing code as the AI algorithms in order to test it. The common code found in control_common.c which determines heading errors and tacking decisions was not testable on its own as it could not be run in a manner that was self standing, therefore it had to be tested by testing the PID and fuzzy logic code and adding extra code to show the workings of these functions in order to verify their behaviour. As both these functions performed relatively simple mathematical functions it was possible to place the system into a state where the outputs of these functions could be calculated using either a calculator or a spreadsheet and then verify the results, a copy of a spreadsheet used to verify the heading error results can be found in appendix M.

7.1.4 Testing the PID Controller

The PID controller proved to be one of the hardest parts of the system to test. This was due in part to its heavy interaction with other code and that in order to verify its output the other code had to be running and working correctly. The PID controller testing was split into two areas, testing the rudder algorithm and testing the sail algorithm. Testing was performed by using a series of extra debug statements in the code in order to display the outputs from the controller and the simulator was used for visual verification of these outputs. In order to test only the rudder algorithm the sail algorithm was simply made to set the sail to zero at all times and vice-versa for testing the sail algorithm.

7.1.5 Testing the Fuzzy Logic Controller

The fuzzy logic controller was tested by observing its choices of rules of final decisions by manually recalculating those decision based upon the data tables available to the fuzzy logic controller. Initial tests where performed by removing the fuzzy logic controller from other code and only having it display its results as numbers on screen. This allowed for a complete verification of the results before any testing took place with the simulator or robot. Later tests then placed the fuzzy logic controller back into the full environment to test its interactions with other components and to visually verify its output using the simulator. The fuzzy logic controller actually proved to by very simple to test in comparison to the PID controller, this was due to the fact that most of the intelligence actually came from the fuzzy sets which were produced using data tables rather than computer code. This advantage was somewhat offset by the time spent designing and checking the fuzzy sets where correct.

7.1.6 Testing the Simulator

The simulator was tested via a number of stages, these were the TCP/IP protocol code, the sail setting code and the rudder setting code. Before the sail and rudder code could be properly tested it was necessary to prove the TCP/IP code was working correctly. In order to do this commands where issued both via a telnet client and from the netmenu program. The 7 standard commands were tested and additional test harness code was added to the server portion of the simulator in order to establish that the parser code was correctly identifying the commands and was able to return something, additionally malformed commands were sent to ensure that the correct errors were returned. Once it was established that the TCP/IP code responded correctly the rudder and sail code was tested in a similar manner, by using a combination of debug output from the simulator code and observing the actions performed on screen it was possible to establish if the sail or rudder actions were correct given inputs from the netmenu program. A number

of difficulties were encountered with the sail algorithm as there were several different situations which altered the algorithms behaviour, the main test being that the sail would set correctly when the tack was altered. A common bug was that requesting a sail setting of 90 degrees worked on both tacks, when it should have worked on one tack and a value of 270 should have been required for the opposite tack.

7.1.7 Test Coverage

When testing any software system it is desirable to ensure that every single line of code is exercised in the process of testing. Given the relatively small size of the code in this project the tests described in the previous sections where capable of reaching every single line of code in the system. As there were few conditions that occur exceptionally rarely this was relatively easy to achieve. The main area in which such conditions occurred was in the string parsing code of the server where a number of if statements are used to look for mis-formed strings and obscure error conditions such as sockets terminating in the middle of ongoing communications. It was found that testing these was possible by deliberately stopping either the TCP/IP client or server in the middle of normal program operations and by placing malformed commands into the socket by deliberately mis-typing them over the a telnet session and observing the resulting error messages.

7.2 Algorithm Performance Testing

In order to effectively test the performance of algorithms and produce a comparison between the fuzzy logic and PID algorithms as well as determining optimal gain constants for the PID controller a method for testing the performance of each algorithm was required. The aim of this test was to establish which algorithm could make the boat sail most efficiently. The ideal algorithm needs to be able to quickly position the boat on the correct heading, maintain the heading with minimal deviation, set the sails correctly so that the boat is able to sail as quickly as possible and be able to adjust the sails efficiently as the wind and/or course changes. An ideal algorithm also needs to be able to perform well on all points of sail.

In order to perform such an evaluation it was necessary to the boats position, heading and desired heading. Within the simulator this can be done simply by logging the X and Y co-ordinates which are stored internally by the program, for the actual robot this required a GPS to be placed on board and used to log data which could be matched up to the corresponding log data for the heading and desired heading at a later point in time.

7.2.1 Algorithm Evaluation Methods

7.2.1.1 Distance Covered vs Straight Line Distance from start to finish

By measuring the straight line distance from where the boat began to where it finished up and then dividing this by the distance of the journey it actually took a ratio for how effectively a course was followed can be generated. This can be calculated by taking the X and Y (or Latitude/Longitude) co-ordinates of both the start and stop point and treating them as a triangle and using Pythagoras's theorem to determine the length of the line. The distance actually travelled can be calculated in a similar manner where the distance between each recorded point and the next recorded point is calculated using Pythagoras and the total of all these distances taken as the total distance covered. Figure 7.1 shows an example plot with both the actual course and the straight line course shown. One shortcoming of this method is that if the boat

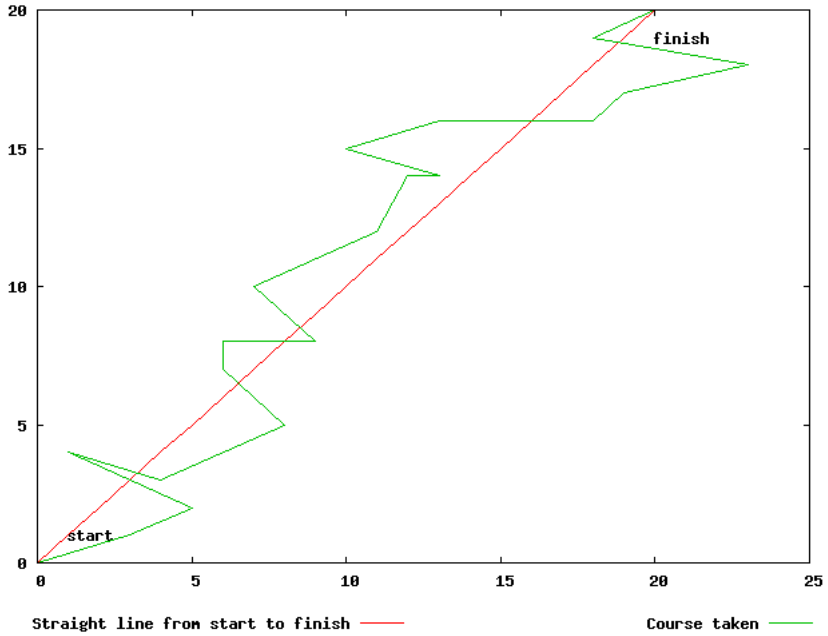

Figure 7.1: Example plot showing how course efficiency can be measured.

goes in the wrong direction but still sails efficiently then it will score highly, another is that if it doesn't move (or goes round in very small circles) then it may also score highly.

7.2.1.2 Distance from goal

This method sets a goal point to which the boat needs to try and reach, the nearer it gets in a constant amount of time the better. This method addresses the problems of the boat sailing in the wrong direction or not moving at all. The difficulty is establishing a good goal point as such a point needs to beyond a distance which can be practically reached, the desired point needs to be further than the boat could possibly travel in the allocated amount of time in order to prevent it from passing the point and skewing the results. It is also somewhat difficult to make either the simulated boat or the real robot travel towards a goal point as neither allows the user to specify this, they only allow a desired heading to be specified. However this method can still be used by specifying a goal some distance from the start point which is also on the correct heading from the start point. It is possible to calculate the distance from the goal by taking the last point the boat reached before the allowed time expired and measuring the distance from this point to the goal, again using Pythagoras.

7.3 Full System Testing

In addition to testing individual components it was necessary to verify the behaviour of the system as a whole to ensure that all levels of communication worked correctly and that the system was capable of performing the tasks required of in both the simulator and robot environments.

7.3.1 Simulator Tests

Once component testing of the simulator was complete and it was confirmed that the simulator responded correctly to commands sent to it either via a telnet session or the netmenu program,

control was switched to the normal PID and Fuzzy Logic code. The resulting motion was observed visually via the simulators GUI interface and also logged as part of the PID and fuzzy logic code. The position data or the time/heading data from the log files was then used to draw a data plot to show the course taken over the entire test, an example of such a plot is shown in figure 7.2. Two tests were conducted for each algorithm, the first was to sail a beam reach on a specified heading, then perform a 180 degree turn and sail back to the start point and the second to sail a triangular course again returning to the start point.

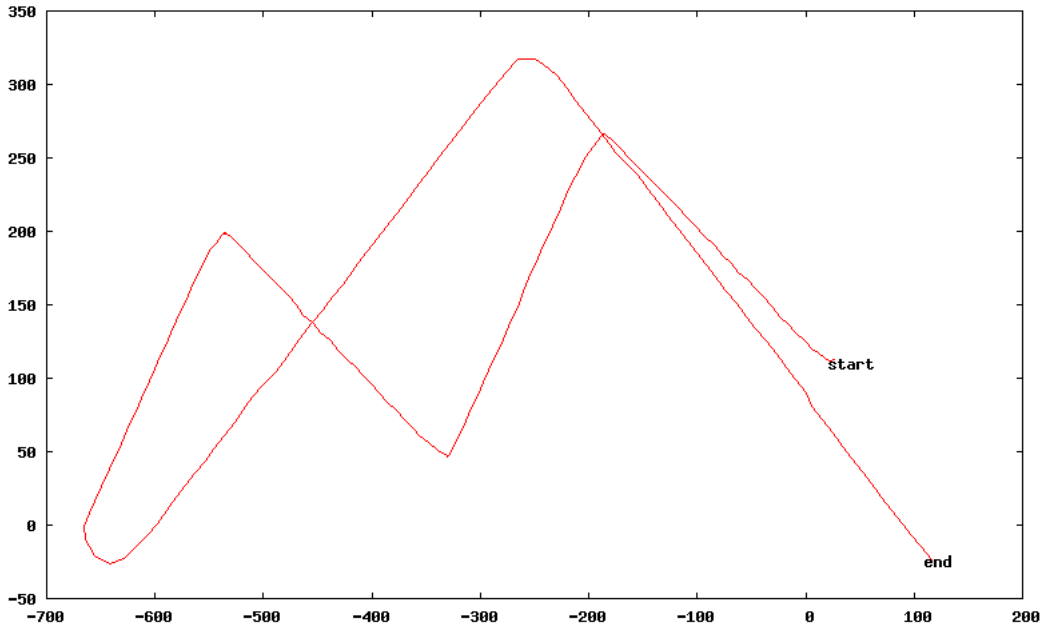

Figure 7.2: Example plot showing the path taken by the boat during a simulator run. It is also possible to plot the heading against time instead of the explicit X-Y co-ordinates.

7.3.2 Lab Testing

Lab testing took place by placing the robot in a position where it could be manually rotated, as shown in figure 5.2 and the wind "simulated" as shown in figure 5.4 and 5.5. The control algorithm was then started up, whenever the rudder was rotated the boat was turned to show a corresponding turn and when the rudder was straightened the turn stopped. This offered a chance to test the performance of the algorithms on the actual boat and check that reasonably correct behaviour was being applied. It also allowed for the entire system to be tested as a whole and for prolonged amounts of time in order to uncover any problems that might result from communication errors, memory leaks etc. This testing also made it possible to observe the typical response times of the robot to changes in its environment. It was still not possible to simulate a number of real world conditions in this testing the most important of these being the speed at which the robot was able to turn on the water and the effects of movement and tilting upon the hardware and sensor readings. Ideally an additional testing step would have been to test on some kind of swimming pool or test tank in order to discover these problems without having to deal with the difficulties of variable weather in the real world, unfortunately no such setup was available.

7.3.3 Real world Testing

Real world testing was undertaken on a small lake near Aberystwyth, this reduced the chance of the boat being lost as should the code crash the boat would most likely drift towards shore within a relatively short amount of time and it would often be possible to recover by throwing a rope over it. The intention was to test the algorithms by first trying to cross the lake by sailing along a fixed heading, then to attempt to sail across the lake and back on a beam reach and finally to sail a triangular course using three different points of sail. Before running such tests the wind direction was determined through a combination of visual observation and the use of a hiking compass, this allowed for an appropriate course to be determined for each test. During each test a GPS receiver was placed onboard the boat in order to allow the course taken to be recorded using the built-in logging functions of the GPS receiver. Each set of tests was to be repeated with differing gain constants using the PID controller and also with the fuzzy logic controller, if possible each test was to be performed multiple times as differing wind conditions would not make the comparisons fair and there was a desire to test repeatability.

7.4 Using the evaluation methods effectively

The methods specified in the previous section can be combined with some common sailing tasks in order to produce a comprehensive testing system. The first to have it sail away from its start point on a beam reach and then turn around and return to the start and second to have it sail in a complete triangle again returning to the same start point.

7.4.1 The Beam Reach test

A beam reach involves sailing perpendicular to the wind and on this point of sail it should be possible for the boat to sail in one direction for a given amount of time, turn 180 degrees and then sail back on the opposite tack (with the wind now at 270 degrees in relation to the boat, instead of 90 degrees or vice-versa). The result of this should be that the boat is able to return approximately to its start point. In order to measure how well the course is sailed the course needs to be considered in two parts, the part before the turn and the part after. For each of these the course efficiency needs to be taken as well as the distance from a goal, which needs to be along the course the boat is travelling but beyond a the distance it can possibly reach. It is likely that a significant amount of sideways movement will occur during testing of the robot on this course given that the boat has a very narrow keel and so little very little to prevent this happening when the wind is blowing across the side in this manner. Figure 7.3 shows an example plot of a boat attempting to sail in this pattern.

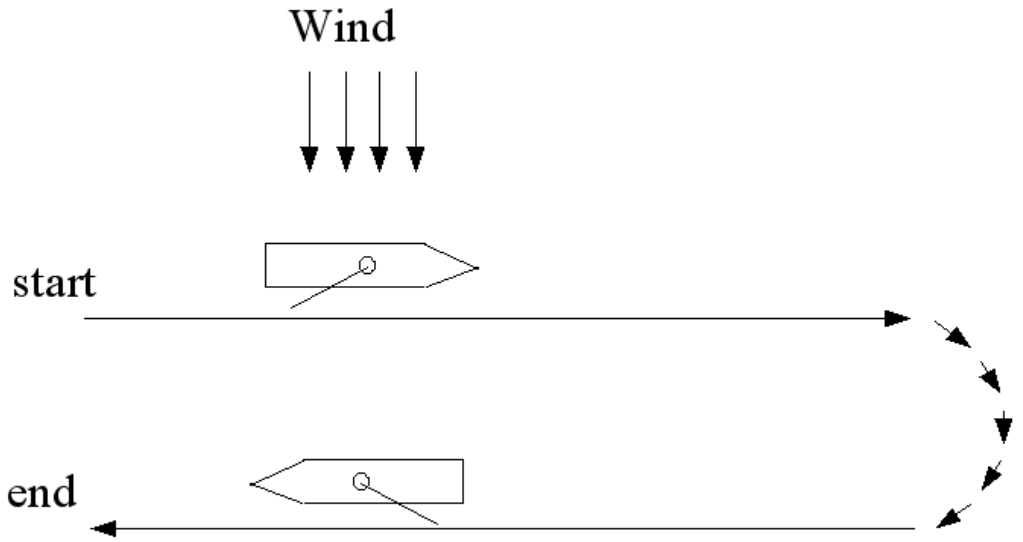

Figure 7.3: An example plot showing the course taken during the beam reach test.

7.4.2 The Triangular Course test

A common task when teaching humans sailing is to ask them to sail a triangular course, this is done as it usually requires the use of 3 points of sail and for sailing into the wind to be performed. It therefore seems logical that an autonomous sailing boat should be capable of performing this task and that asking it to do so is a good test of the artificial intelligence algorithms. Any algorithm which is able to sail a triangular course reasonably well should be able to sail the boat to any position its operator desires. The triangular course can be specified as three different headings with 60 degrees between each. In order to form a triangle and actually return to the starting point, the boat will have to spend more time on some parts of the triangle than others as it will move fastest when sailing on a beam reach and slowest when sailing close hauled, therefore the time spent on each part of the triangular course must compensate for this in order to produce equal lengths to each side of the triangle and these timings must be determined manually through a process of trial and error they will also differ between the algorithms. It is also likely that some sideways movement will occur on the actual boat and that given the lack of GPS support this will not be detected or compensated for and as a result the robot will not produce a perfect triangle. As with the beam reach an evaluation needs to be undertaken for each side of the triangle, when comparing two algorithms each portion of the triangle is compared against the same portion for other algorithms. As the goal is to return to the start point it may also be useful to see how far from this point the boat ends up, although evaluating against this alone is prone to problems as the boat will also be close to this point if it never moves. Figure 7.4 shows an example plot of a boat attempting to sail in a triangle.

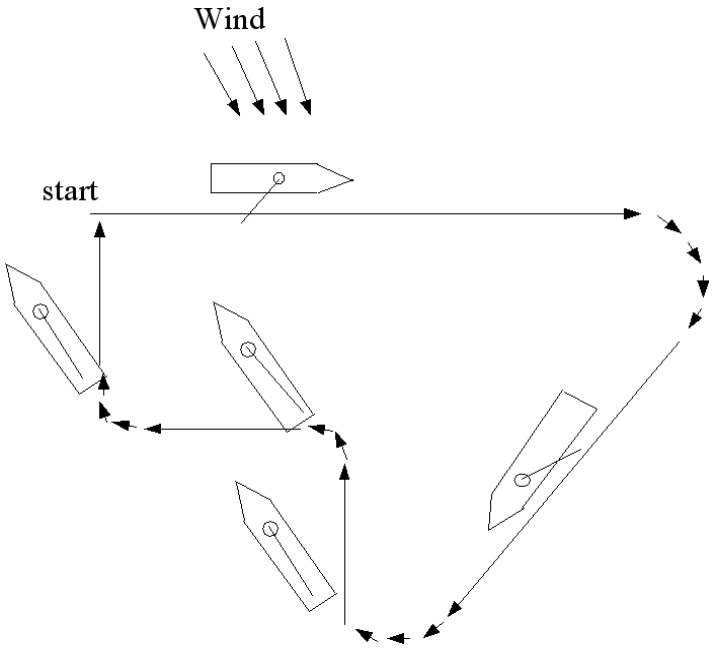

Figure 7.4: An example plot showing how a triangular course is performed. It should be noted that a triangular course does not always require sailing into the wind and can sometimes be performed without repeatedly tacking across the wind.

7.5 Testing Results

7.5.1 Simulator Test Results

7.5.1.1 PID Controller Results

The PID controller was given the optimal constants defined in section 6.1.5.1 which were a proportional gain of 3, integral and derivative gain of 0. The beam ream test was conducted by configuring the PID controller to sail a beam reach for one minute on a heading of 180 degrees after which it sailed for one minute on a heading of 0 degrees. The triangle test was conducted by configuring the PID controller to sail on a heading of 180 degrees for 60 seconds, then on a heading of 315 degrees for 60 seconds and finally a heading of 45 degrees for 30 seconds. The timings were decided as they were found to return the boat to its start point, a trial and error process was used to determine these values. In both tests the PID controller performed very well as the results in tables 7.2 and 7.1 show. Graphs showing the exact paths taken can be found in appendix P. It seems the key to success with the PID controller is the ability to move very quickly to a new course and to hold that course accurately once it has been reached.

Journey	Total Distance	Straight Line Distance	Efficiency Ratio	X End	Y End	X Goal	Y Goal	Distance from Goal
Outward	619.310	579.346	93.5%	137	-552	0	-550	137.0145
Return	688.556	587.020	85.2%	246	2	137	-552	564.621

Table 7.1: The results of the simulated PID controller beam reach test.

Triangle Portion	Total Distance	Straight Line Distance	Efficiency Ratio	X End	Y End	X Goal	Y Goal	Distance from Goal
1	629.726	573.900	91.13%	161	-540	0	-550	161.310
2	522.990	477.322	91.26%	-178	-231	-270	-270	99.924
3	412.768	401.110	97.17%	82	79	40	21	71.610

Table 7.2: The results of the simulated PID controller triangle test.

7.5.1.2 Fuzzy Logic Controller Results

The fuzzy logic tests were performed in the same manner to the PID tests, the same timings of one minute each way were used in the beam reach test and a new set of timings determined for the triangle test in order to allow the triangle to actually be completed. As the plots in appendix P show the fuzzy logic controller encountered difficulties in reaching its desired heading and appears to have settled on a heading somewhat short of the specified one, it also takes significantly longer than the PID controller to achieve this. Upon closer examination it was found that a bug in the defuzzification routines would cause an incorrect output to be produced thus resulting in the wrong heading being followed. It was also realised that the current set definitions are comparable with a proportional gain constant of one in a PID controller as one degree of error equates to one degree of movement, it may therefore be possible to improve the convergence speed of the fuzzy logic controller by altering the set boundaries so that the probability that the boat is off-course reaches 100% far sooner. Despite these problems the fuzzy logic controller is able to control the boat in a reasonably sensible manner and it is likely that with some additional tuning it could be made to match the performance of the PID controller.

Journey	Total Distance	Straight Line Distance	Efficiency Ratio	X End	Y End	X Goal	Y Goal	Distance from Goal
Outward	620.605	547.891	88.3%	456	-329	0	-550	506.731
Return	589.951	478.682	81.1%	823	-36	456	-329	469.614

Table 7.3: The results of the simulated fuzzy logic beam reach test.

Triangle Portion	Total Distance	Straight Line Distance	Efficiency Ratio	X End	Y End	X Goal	Y Goal	Distance from Goal
1	643.950	570.411	88.57%	467	-346	0	-550	509.613
2	616.912	517.549	83.89%	-1	-177	-173	-173	172.047
3	204.865	180.050	87.88%	19	8	27	17	12.046

Table 7.4: The results of the simulated fuzzy logic triangular test.

7.5.2 Real World Test Results

7.5.2.1 First Test - April 8th 2005

During this test the boat was never actually launched into the water. This was in part due to snow and cold weather and in part due to some Psion specific bugs in the code which prevented correct parsing of commands in the TCP/IP server. An attempt was made to fix these bugs on site but due to the poor weather the decision was made to take the boat back to the lab and continue debugging there.

7.5.2.2 Second Test - April 27th 2005

During this test an initial run was made using the proportional controller code in an attempt to sail a beam reach (perpendicular to the wind direction) and then turn around after 100 seconds and sail back. Unfortunately the boat made some stranger maneuvers in the first few seconds of operation which tangled the cable linking the wind sensor to the Basic Stamp this prevented the sail from turning and thus left the Basic Stamp's sail algorithm constantly trying to move the sail and caused the Psion based code to block. At the same time the rudder was fully turned and the combination of this and the resulting sail position actually caused the boat to enter a state known as "heaving too" in which the sail and rudder contradict each other to leave the boat in a near stationary position. The result of this was that the boat occasionally turned in a circle or sailed backwards while gradually drifting further from the shore. The map recorded by the on-board GPS can be seen in figure. Matters were made worse by changing winds which due to local topography shifted between the southwest and southeast. Later it was identified that the compass appeared to be giving the same reading at all times this was attributed to a hardware fault. In order to work around this problem it was decided to use the wind sensor to as a direction sensor and to keep the boat with the wind at the same angle, this would in theory cause the boat to go in a straight line providing the wind did not change. Unfortunately this did not work properly either as the wind sensor began to return consistent values regardless of its position, after confirming that the wind results where correct via the Basic Stamp it was concluded that the rather loose fitting RS232 connector on the Psion was at fault.

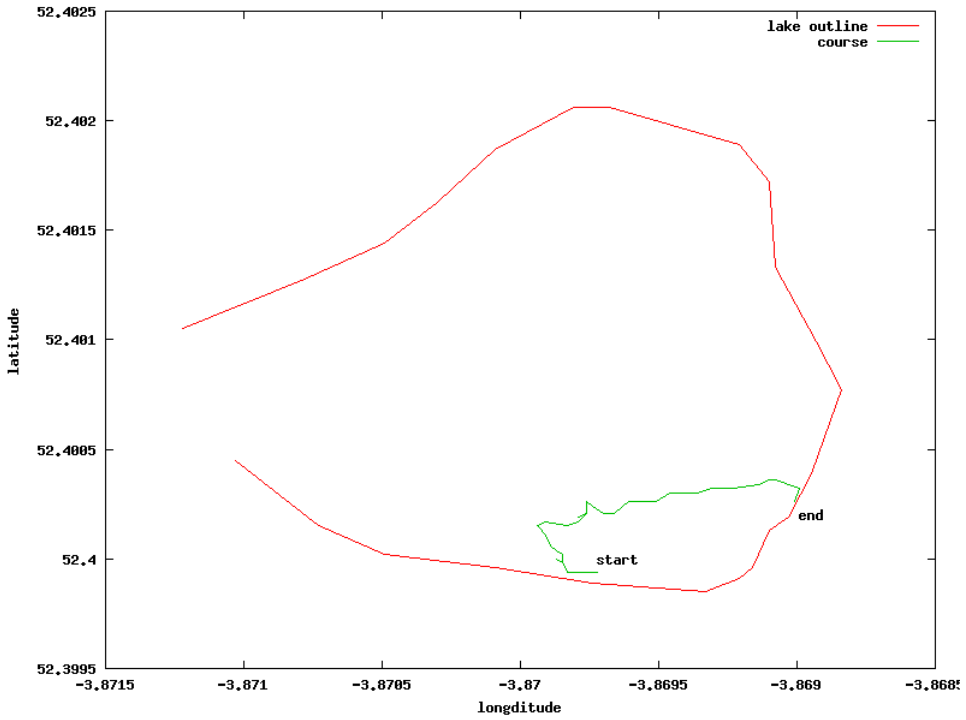

Figure 7.5: GPS Plot of the first test in which the sail jammed and the boat was eventually pushed onto shore.

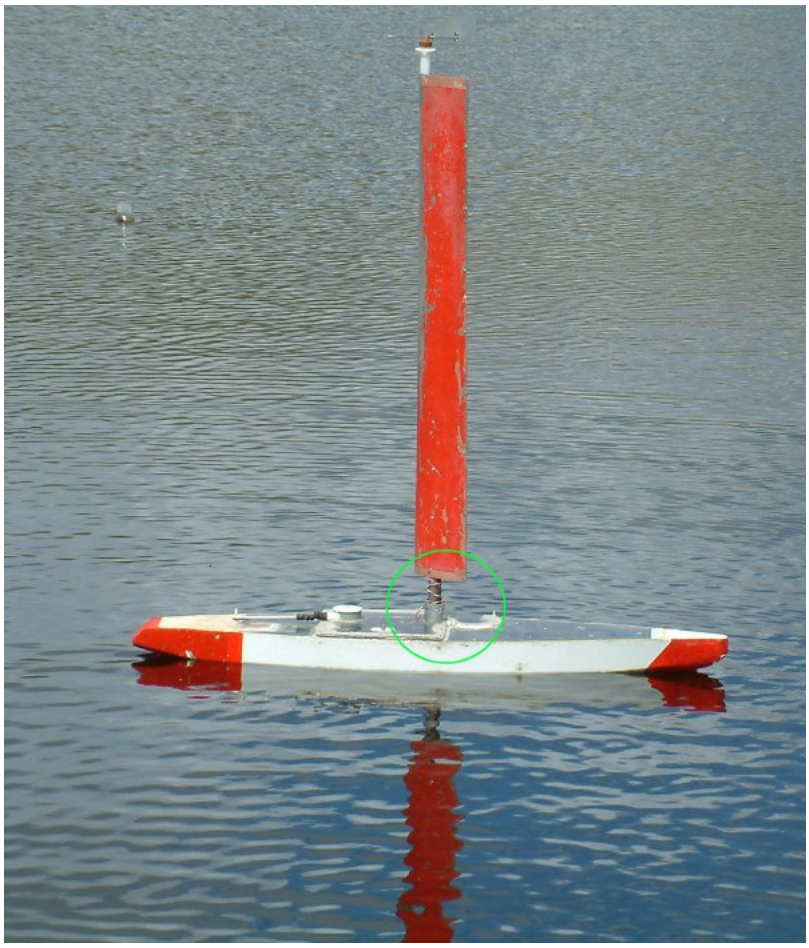

Figure 7.6: Photo illustrating the wire wrapped around the mast, this prevented the sail from turning and in turn prevented any further commands from being issued as the sail code continuously attempted to correct this.

7.5.2.3 Third Test - April 28th 2005

For this test the hardware platform was changed from a Psion 5mx to a Jornada 720, another PDA or similar size and shape which is also based around an ARM processor and capable of running the same Linux operating system and control software. The main reason for doing this was that the Psion's serial connector was moving out of place as the boat turned and causing communications with the Basic Stamp to stop and the Jornada features a much more robust serial port connector, additionally the Jornada features a PCMCIA slot and allowed for a wireless network card to be placed on-board allowing for remote access to the boat from a much greater distance than that permitted by the Psion's infra-red port. Despite this a problem occurred when the boat sailed out of radio range from the shore, due to the manner in which peer to peer wireless networks operate when the boat appeared back in range of the laptop they were unable to re-establish communications as each card formed its own independent cell once the signal was lost and would not automatically merge back into a single cell once signal was regained.

Given previous experience with the control of the sail it was decided to manually set the sail's position and then use the code only to control the rudder in order to keep a given compass heading. The wind was blowing from the north west allowing for a beam reach to be sailed from

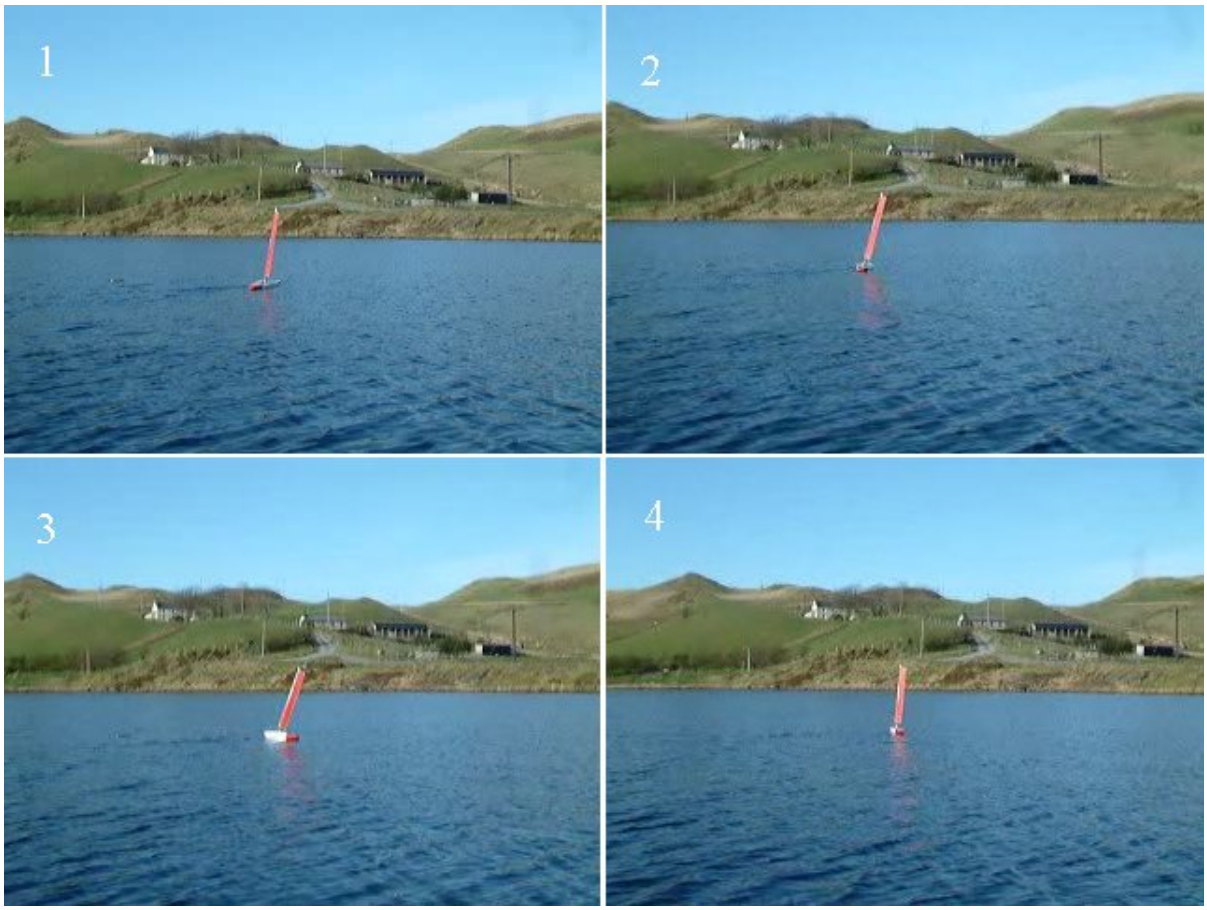

Figure 7.7: Four stills extracted from video of the second test showing the extent to which oscillation was occurring.

the southern shore to the eastern shore of the lake on a heading of approximately 45 degrees. In order to achieve this the sail was manually positioned to approximately 315 degrees.

This test was generally more successful than either of the previous two tests, a total of four test runs were performed. In the first an attempt was made to sail in a straight line from the southern shore of the lake to the eastern shore on a heading of approximately 40 degrees. The code was started manually from the laptop via the wireless connection and upon loosing signal approximately 100 metres from shore the program stopped as the login shell from which it had been launched timed out and terminated. Despite this the boat continued without software control on its existing course with little deviation and was able to sail across the lake.

For the second test the same course was attempted but the program was made to spawn off processes onto a separate virtual console (this approach had been taken on the Psion) to prevent them from terminating when the process which launched them exited (as it does upon loosing its network connection). During this test the boat was able to again sail across the lake in a reasonably straight line, however a large amount of oscillation was observed suggesting that the proportional gain constant was set too high. Figure 7.7 shows four stills from a video of this test illustrating the level of oscillation which was occurring, each of the frames shown were taken with only a few seconds between them. In order to test the repeatability of this a total of three separate voyages were made, the code was left running between each of these as the boat was carried back to the start point. A GPS plot of this test is shown in figure N.2.

For the third test it was decided to reduce the proportional gain constant from two to one and to conduct the full beam reach test, having the boat turn around and sail back to its start point. This was achieved by writing a small shell script which would sail for three minutes on a course of 45 degrees and then exit, the netmenu program was then used with a redirected standard input to change the sail position from 315 to 45 degrees (as the sail setting had been disabled in the normal code) and finally another instance of the proportional controller code was launched this time sailing on a heading of 315 degrees for an infinite amount of time. The result was generally encouraging with the returning to within 30 metres of its launch point. It was also found that reducing the gain constant reduced the oscillation significantly, this can be seen in the heading/time plots in appendix O.

For the fourth test it was decided to try the fuzzy logic controller and to have the boat sail the same course as the third test but to make it perform this three times. Unfortunately after making its first run the wind strength began to drop significantly and after rotating the sail it took the boat approximately 2 minutes to get onto the correct heading, after the next turn was made a minute later the wind had almost disappeared and the boat then drifted for a further 15 minutes at which point the code exited leaving the boat without any control code. Finally after another 12 minutes it was eventually blown back onto shore by the then virtually non-existent wind. One observation of the fuzzy logic controller was that it was waiting far longer between rudder movements than the proportional controller, this caused it to oscillate far more than the proportional controller had done although it showed that there could be several seconds of delay between each rudder movement without adversely affecting the boat's ability to keep itself going in the correct direction.

7.5.3 Known Bugs

7.5.3.1 Jornada leading byte upon server restart

A problem was identified when running the code with the Jornada 720 (and only the Jornada 720) that once the server application had been started and had issued some commands to the Basic Stamp then if it were restarted a leading byte would then appear on the Basic Stamp upon the receipt of each command. This caused the Basic Stamp to mis-recognise each command it received and thus stop responding. The true cause of this issue was never discovered and the commands produced by the Jornada were found to be correct when it was connected to other terminal equipment for diagnosis.

7.5.3.2 Stability of the Basic Stamp code.

A number of bugs exist in the Basic Stamp code which cause it to crash at times thus preventing the robot from continuing its normal operation. One such bug is that there appears there needs to be a minimum of at least one second between issuing two commands over the serial line, if not the Basic Stamp will crash. However to make matters worse this condition does not appear everytime less than one second elapses between commands and also appears to be more common after the set sail command than any other. It is unclear if this bug is connected with the problems seen with the Jornada issuing extra bytes.

7.5.3.3 Sail updates too frequent.

Given the slow rate at which the sail turns and the inaccuracy of the sensor, it typically takes at least three seconds to move the sail during which time no other commands can be issued. This

Run Number	1	2	3	4
Controller Type	Proportional	Proportional	Proportional	Fuzzy
Constants	2	2	1	N/A
Observations	Sailed across lake despite loss of wireless signal causing code to terminate halfway.	Sailed across lake with code running the whole time. Observed to oscillate quite considerably. Repeated this three times, but kept the code running while bringing the boat back to the launch point.oscillate	Sailed across lake and back. Observed to oscillate far less than with a gain constant of 2.	Sailed to middle of lake before wind dropped off leaving it to drift to shore. Observed considerable oscillation.
Efficiency Ratio	95%	97.5%, 99.5% and 99.6%	95.85% out, 86.98% coming back (This is reduced by the turn which aims the boat back at the shore)	52% overall, 99.4% for initial journey before wind dropped.

Table 7.5: Table describing each journey made during the test runs. It has been observed that the GPS failed to take enough samples to show the true course taken by the boat and as a result the efficiency ratios are higher than they should be. [2]

is sufficient time for the boat to move off course by at least 45 degrees, as the code currently sets the sail everytime it goes round its main loop this causes immense problems in controlling the boat effectively. During real world testing the sail setting code had to be completely disabled and the sail set manually in order to avoid this problem.

7.5.3.4 Fuzzy Logic Controller fails to centre properly

A bug in the fuzzy logic code prevents the fuzzy logic controller from centering on the desired heading and consistently leaves the boat off course. This is due to a problem defuzzifying to a set which has two sloping boundaries (the centre rudder output set is the only set in the system to feature this). It should be possible to solve this by splitting this set into two, one for when the boat is centred but still to the right and one for centred and to the left.

Chapter 8

Project History

8.1 A month by month account of the project

This section gives an account as to what was achieved in each month of the project. A copy of the original project timetable can be found in Appendix Q.

8.1.1 October 2004

During October the initial background research to the project was performed and the initial version of the Market Analysis was written. A basic version of the serial protocol was implemented in C and tested on a Linux laptop, this allowed the laptop to control the basic features of the boat. Research was also undertaken to find a suitable simulator, after assessing many projects Tracksail[38] was found to be the only suitable program available. Additional research was undertaken into cross-compilers, several cross-compilation strategies including manually recompiling GCC, pre-built cross compilers, crosstool and finally uClibc Buildroot where tested. uClibc was chosen as not only did it include a cross compiler, but it also prepared a complete Linux system image which the Psion was able to boot successfully.

8.1.2 November 2004

During November the bulk of the low level implementation and calibration was completed. Calibration had originally been tasked with taking only one week, but as initial results were not satisfactory some calibration had to redone. Problems also emerged with the compass at this point, it was found that the compass was always returning the same value, this was eventually traced to a hardware fault and as a result compass calibration was not completed until early February. An initial implementation of the server was also undertaken during November, however problems were encountered with the sail setting algorithm. This implementation succeeded in offering all other simulator facilities. Towards the end of November additional changes were made to the documentation, with an initial production of the design and calibration sections.

8.1.3 December 2004

During December the TCP/IP system was implemented and an initial proof of concept program was written using this. Initially this program would align the sail with the wind sensor, if the wind sensor moved the sail would then move to the same angle. This algorithm then evolved into the beginnings of a proportional controller.

8.1.4 January 2005

As a result of the Christmas break, other coursework and exams very little work was undertaken during January. This caused some slippage against the original timetable specification. A small amount of work was carried out in relation to compass calibration and fixing bugs within the TCP/IP and serial handling code.

8.1.5 February 2005

During February a number of outstanding issues where solved, these included the simulator and compass calibration, the Basic Stamp code was also completely restructured with some minor changes to the serial protocol emerging. Once this was complete an implementation of a PID controller began, this was based upon the proportional controller written during December. Development of the PID controller was also coupled with the development of methods and specialist programs to aid in algorithm development as well as the addition of logging features to the high level code, this allowed for various variables such as the course taken, heading error and boat positions to be graphed and analysed. It was hoped that some initial testing could be carried out on the water during February, unfortunately bitterly cold weather prevented this.

8.1.6 March 2005

During March further development of the PID controller was undertaken and a series of simulator tests were conducted. This information was then analysed using additional scripts which implemented the algorithm evaluation strategies, selection of optimal values from this process is still being undertaken. Towards the middle of March the documentation was converted from OpenOffice to LaTeX and finally to LyX as OpenOffice was causing problems with layouts, bibliographies and sectioning. The design section was also expanded, and the beginnings of the testing and conclusion sections were produced.

8.1.7 April 2005

During April implementation of the fuzzy logic controller was undertaken. Large amounts of testing were performed in the lab by rotating the boat around and using a fan to simulate wind, this testing helped to identify a number of issues with the control code and several Psion specific bugs which had not been found during earlier tests using a laptop to control the robot. An initial attempt to perform a test on the water was thwarted by a combination of bad weather and buggy code. A later attempt proved somewhat more successful with the boat actually entering the water, however a series of hardware faults prevented it from sailing correctly. A final attempt saw a far greater level of success with four test trips taking place, the hardware was also changed from a Psion 5MX to a Jornada 720 a more powerful PDA with a similar (but faster) Strong ARM processor and the ability to run a wireless network.

Chapter 9

Evaluation and Conclusion

9.1 PID vs Fuzzy Logic, Which worked best?

Generally it was found that the PID controller yielded better results than the fuzzy logic controller in both the simulator and on the robot. However far more time and effort was spent tuning the PID controller and if the fuzzy logic controller had seen similar tuning then the final result may show less of a difference. The fuzzy logic controller also suffered from poor winds during the robot testing. The simulator tests showed that the fuzzy logic controller was far slower at turning towards the correct heading and never actually managed to achieve the correct heading, this may in part be due to a bug in the implementation of the fuzzy logic controller and due to a lack of tuning with the set boundaries. However it is believed that the current set definitions for rudder control should offer similar performance to the PID controller running with a proportional gain constant of one, yet performance still seems to be worse than this. In order to truly decide which is the better strategy for controlling a sailing robot further investigation needs to be undertaken. It is quite possible, especially given the nature of the hardware in this project that a fuzzy logic controller could yield better results. As the accuracy of the sail positioning differs depending on the desired direction it maybe possible to improve this by altering the fuzzy set definitions in place, doing this with a PID controller is much harder given that it forms a general rule which applies to all possible positions and basically assumes hardware behaves consistently regardless of its position.

9.2 Evaluation

Overall this project has managed to fulfil its aims in producing software which is capable of controlling a sailing robot to allow it to follow a pre-determined course, but only within a simulator. In the real world the software still requires additional development in order to do this fully and to a large extent the lack of ability to do this is more a result of unreliable hardware than poor software. Despite its shortcomings the resulting software is able to produce the correct moves in many situations and has been produced in a highly portable manner allowing it to theoretically work with any POSIX compliant operating system that includes a C compiler. This has been illustrated through its use on both laptop and two PDA Linux systems based around two very different Standard C Libraries and three different distributions. The simulator is capable of providing a reasonably accurate simulation of a sailing boat and is able to show its output both in a realtime graphical form and by saving data to a log file containing headings, position, wind direction etc. It is also capable of running under the same control algorithms the boat does and performing somewhat similar actions.

9.2.1 The final product compared with requirements specification

9.2.1.1 Hardware Requirements

The requirements specification states the need for an appropriate hardware platform to be identified and that this platform needed to be capable of both interfacing with the sensors and servos as well as having enough computing power to perform the AI functions and enough storage space to log any data which is generated. It was believed that this requirement was satisfied by the combination of the Basic Stamp microcontroller and the Psion 5mx with the Basic Stamp supplying the I/O facilities and the Psion supplying the computing power and storage facilities. Unfortunately linking these two proved to be problematic as it increased response times between sensing an event such as a wind change and responding to them. The physical cable linking the two devices also proved to be a source of communications problems. These problems were addressed by the Jornada 720 which also satisfied all the other requirements which the Psion 5MX had passed. If a suitable budget had been available then a more appropriate microcontroller which combined the I/O facilities of the Basic Stamp and the computing power of the Psion could have been used, such microcontrollers include the Gumstix[41] or LART [42]microcontrollers all of which are based around ARM processors similar to that of the Psion and are capable of running Linux while at the same time providing at least 10 general purpose I/O lines.

9.2.1.2 Sensor and Servo control software

The requirements stated the need for the software to be able to interact directly with the sensors and servos and relay information from them onto higher level code which was responsible for decision making. Such software was implemented in the form of the Basic Stamp code and the TCP/IP server code which interacts with it, this code was also able to provide a translation from device specific measurements such as PWM timings into human understandable ones such as degrees. The requirements stated the need to possibly control GPS or radio link hardware, no low level support was written for GPS although higher level support was implemented, no radio link support was attempted at this level, however a simple system was implemented via the use infra-red and wireless network communications to the Psion/Jornada in order to reduce the number of times the robot needed to be opened to update settings.

9.2.1.3 Artificial Intelligence Algorithms

The artificial intelligence algorithms were required to be able to control both the robot and the simulator in a reasonably intelligent manner. This included following the basic rules of sailing such as avoiding sailing directly into the wind. Both the PID and fuzzy logic implementations when run under the simulator are capable of sailing intelligently and will allow the simulated boat to sail in any direction it is told and will avoid sailing directly into the wind when required. However when run on the robot the code performs far from perfectly, this is mainly due to performance issues, the code is actually making reasonably intelligent movements but is unable to perform these quickly enough in order to make the boat appear to be doing anything remotely intelligent or correct.

9.2.1.4 Simulator Requirements

The original requirements called for a simulator which was capable of acting a reasonably realistic manner and producing either graphical output or logging data to a file. The chosen simulator was based upon an open source project called Tracksail [38], it was able to fulfill the requirements

of producing a reasonably realistic simulation and presenting the output graphically in realtime. Additional changes were made to allow the same code which ran the robot to interact with it and to allow logging to a datafile.

9.2.1.5 Navigation Requirements

The requirements specified that there should be some means by which the system can navigate in order to follow a pre-determined course and that ideally such a navigation system should not be prone to errors as a result of tilting and that it should be able to update quickly enough to keep the boat on a straight line. In the final system the only navigation equipment available was a compass which unfortunately did not have any means by which it could be kept level however it did prove to be able to update quickly enough to keep the boat on a reasonably straight course (once the sail setting code was removed to allow the compass to be read regularly enough).

9.2.1.6 Portability Requirements

The requirements stated that the code should be as portable as possible and be capable of running on a variety of CPU architectures and operating systems. So far the code has been successfully proven to work under Mandrake Liunx 10.1 running version 2.6 of the Linux kernel using a Pentium 3 processor, on a Psion running uClinux with a version 2.4 of the Linux Kernel running on an ARM7 processor and on a Jornada 720 running Familiar Linux 0.82 using version 2.4 of the Linux kernel and a StrongARM-1100 processor. An attempt has been made to produce all code in a portable manner and no operating system or CPU architecture specific features have been used in the C code. The Java code has been tested using Sun Java version 1.5 on Linux, Windows and Solaris. Additional testing with other POSIX compliant operating systems such as BSD Unix or QNX should be undertaken to establish exactly how portable the current code is. One desirable feature is that the code can be ported to an operating system-less microcontroller which has support for the C language, in order to do this some changes would be required, such as the serial port and socket code as these features would not be supported (at least in their current form) without an operating system.

9.2.1.7 Data Logging Requirements

The requirements specified that the program should be able to log data to a file for recovery and analysis at a later point in time. The program writes the current elapsed time, heading, heading error, position (simulator only, 0s are written on the robot) and wind direction at regular intervals to a data file. As the graphs in the testing section show it is possible to use this data in order to perform later analysis. Additional logging capabilities for the robot are obtained by using an independent GPS receiver which stores data to its own internal log which can then be downloaded to a PC at a later point in time.

9.2.1.8 Configuration Requirements

The requirements specify that the end user must be able to configure the control algorithms in order to specify the desired heading, amount of time to run the program for and any parameters which the AI algorithms may require. The PID controller allows the user to specify the desired heading, amount of time to run, tacking angle and time to spend on each tack, whether or not to unwind the sail when finished and the proportional, integral and derivative gain constants. The fuzzy logic controller allows all of the latter except for the gain constants as these are not relevant to a fuzzy logic controller. It is possible to alter the fuzzy sets (which are effectively

the only parameters to the fuzzy logic controller) by editing a series of files which contain data tables.

9.2.2 Limitations of the current design

A number of limitations and problems exist within the current hardware and software designs. Many of these centre around the difficulties of using two different computing platforms connected together via an RS232 port. Many are also as a consequence of the quality of the hardware being used and reflect the small budget available for this project as well as the fact that the boat is a hand built prototype.

9.2.2.1 Stamp delays and communication problems

Many problems with the current design stem from the joint use of the Psion and Basic Stamp. The first of these problems is the inter command delay problem where the Basic Stamp appears to stop accepting new commands if it is issued two commands without a gap of at least a second between them. This problem is exacerbated by the poor quality of the Psion's serial port connector which can be knocked out of place very easily. Given the size and shape of the Psion, its connector and the boat, the connector must be plugged into the Psion once it has been placed inside the boat as it will not fit through the hole while connected. Once placed inside the boat it is difficult to connect and very easy to leave half connected. The use of a Jornada 720 was able to address this problem, however the Jornada was an extremely tight fight and ideally a smaller PDA needs to be used.

9.2.2.2 Sail setting time

Another limitation of the current hardware and software designs is the amount of time taken to set the sail. This can be as much as 10 seconds and is typically at least 5 seconds. This can be considered both a limitation of hardware as the hardware is not capable of turning the sail particularly fast, but it is also a limitation of software as the software attempts to turn the sail far too often and could be improved by reducing this frequency and only moving it when large moves are required.

9.2.2.3 Accuracy and repeatability of sensors and servos

The general accuracy and repeatability of the sensors and servos onboard the boat are present is quite poor. This is especially true of the wind sensor and sail servo/position sensor, this results in wide variations from second to second as the code is running which can cause the boat to move without there being any visible reason for it to do so. In addition to this the compass heading varies immensely as the boat tilts, the result being that as the boat turns the compass heading is distorted causing either the desired heading to be overshot or undershot.

9.2.2.4 Wire wrapping around the mast

On the current hardware design a cable runs up the mast in order to connect the wind sensor with the electronics inside the boat, unfortunately as the sail turns it wraps this wire around the mast and can end up preventing the mast from turning. Although an unwind command was implemented to counteract this, no code was implemented to keep track of how far the sail had turned and whether or not the unwind command should be called. One possible strategy for

implementation might be for the basic stamp program to detect when the mast fails to turn and to then issue an unwind command, another might be to keep track of the total distance turned by the sail and when it reaches a given value to unwind it.

9.2.2.5 Lack of GPS

The current design does not fully support the use of GPS, as a result it is difficult for the code to tell if the boat is moving sideways or even backwards due to the force of the wind, tides, currents etc. In order to implement GPS support within the current design support needs to be added to the Basic Stamp and to the high level algorithms. One possible strategy is to actually program a series of waypoints into the GPS receiver and then have the boat attempt to reach each waypoint in turn with the GPS receiver only supplying the Basic Stamp with information about the current bearing to the next waypoint and then using this information to determine the desired heading.

9.2.2.6 Unintelligent tacking and jibing

When sailing the problem of turning onto the opposite tack so that the wind is now on the opposite side of the boat requires the crew to either turn the boat towards the direction of the wind and temporarily point the front of the boat directly towards the wind as they turn (known as tacking) or turn the boat so that the rear of the boat points directly towards the wind during the turn (known as a jibe). It is common practice only to jibe when the wind was already coming from directly behind the boat, the jibe tends to be a far more violent maneuver as the boom on a conventional boat will suddenly move itself by up to 180 degrees (often hitting crew members on the head in the process!). The current code implementation does not take this into account and will always take the maneuver requiring the smallest turn regardless as to whether this is a tack or jibe, as the sail is fixed and cannot move itself (and there is no crew to be knocked out by a rapidly moving boom) the violence of a jibe is reduced. An investigation needs to be undertaken as to what if any advantages are gained for the sailing robot by always performing a tacking maneuver unless the wind is coming from directly behind the boat. Should it be favourable to do this, then the course setting code must consider the wind direction and whether the boat will be turning through the wind when deciding which way to turn.

9.2.2.7 Simulator

The simulator's physics model is far from realistic, although it covers the basic principles of sailing and will not allow the boat to sail directly into the wind it does a poor job at simulating momentum or the efficiency of different sailing positions. An attempt has been made during this project to simulate variable winds and the boat changing heading without rudder action, however these are still quite basic models. Ideally the simulator could be based upon high quality physics model which is based around the exact shape of the sailing robot, this is a similar approach to that being taken by a number of modern flight simulators which actually simulate airflow over a wing in order to determine flight characteristics. Alternatively it could be based upon a series of measurements regarding performance of the boat, these could be placed into data tables which can then be used by the simulator to provide an accurate simulation, the problem with this approach is accurately measuring the boat's performance and any behaviour which isn't modelled will be undefined. If the accuracy of the simulator can be improved then it will greatly aid the development of algorithms as it is far easier to test algorithms under the simulator than on the robot itself. An additional limitation of the simulator is its unrealistic response times, when setting a sail or rudder in the simulator the results are virtually instantaneous there is no

inter command delay and the sail will turn to any angle within a few milliseconds. Ideally this needs to be slowed down to a rate comparable with the boat so that the algorithms will have to deal with similar latencies in issuing commands.

9.2.2.8 Additional algorithm evaluation methods

As mentioned in section 7.2 the current algorithm evaluation methods fail to fully test if the boat actually travelled in the desired direction, they only check that it travelled the course it took efficiently and examine how close it got to a goal position. If such methods are to be used in automatic evaluations and systems which are able to self tune or even appear to learn then more complex evaluation methods need to be devised.

9.3 If I was going to do it all again, what would I do differently?

9.3.1 Run all code on a single hardware platform

If this project were to be attempted again there are a few things which could have been done differently. The first might be to strip down the code to its bare minimum and make it run entirely inside the Basic Stamp thus eliminating all of the problems that inter-device communication brought with it. However this may remove the ability to make use of calibration data tables and require working directly in units which the sensors and servos understand, however the calibration data would still be required in order to know what values to be using. It would also remove the logging and simulation capabilities. Another possibility would be to use another low cost microcontroller which featured slightly more memory than the Basic Stamp and allowed most of the desired functions to be implemented and using a portable language such as C. A third possibility is to use a high end microcontroller which can run Linux (or another POSIX compliant operating system) and also interface with all of the hardware directly.

9.3.2 Serial Port interface to the simulator

The simulator suffered from being too good in comparison to the actual robot, it responded much quicker to commands and would generally do as it was told. If the simulator had been moved to take commands via a serial port instead of a TCP/IP socket it would have provided a more realistic interface in terms of latency and testing for serial communication issues thus giving the simulator an equivalent role to the Basic Stamp. This may also remove the need to use a socket at all within the higher level code thus reducing its size, complexity and improving portability to operating-system less platforms.

9.3.3 Find a more suitable testing environment

All the real world testing was performed on a lake with full exposure to the elements, ideally testing should have first been carried out in a less harsh environment. Although the code was tested in the lab by using a fan and manually moving the boat around this was not sufficient to find many bugs. What would have been more useful would either be a swimming pool or small indoor tank with some fans (or even an open door on a windy day!) to simulate wind, this way far more time could have been dedicated to performing a much more realistic level of testing. Unfortunately no such setup was available!

9.3.4 Better GPS Logging

By relying on the logging features of the GPS accuracy is lost as some compression is applied in order to reduce the amount of data stored, essentially the receiver only logs when it sees a significant change in the course. This reduces the sampling rate to a point which is far too low to show many of the small movements which the boat performed, for instance the GPS plots (see appendix N) show the course followed was virtually a straight line yet the heading/time plots (see appendix O) show there was significant oscillation. Unfortunately this caused many small moves which the GPS is actually capable of logging from being shown by the GPS logs, for example during the first test in the water the boat spun around several times yet this is not visible on the GPS logs as the distance moved was too small to log. This could have been improved by connecting the GPS to the Basic Stamp and recording its position at regular intervals into the logfile for each journey. However this may have had a side affect of increasing the amount of time between being able to alter the rudder or sail setting as additional load would be placed on the Basic Stamp. If a better microcontroller could have been used then this might not be such a problem.

9.3.5 Better Wireless Control

Initially wireless control was performed via an infra-red port, this only allowed for access at very small distances although it did reduce the number of times the boat had to be taken apart. Later tests were performed using an 802.11b wireless link running in peer to peer mode, unfortunately when two systems running in peer to peer mode loose sight of each other and then regain sight they often fail to re-establish a communications link as each system forms its own cell and once a cell is formed other systems which have already formed their own cells do not change cells, the only devices which will join such a network are new ones which have just booted up (or restarted their drivers). The effect of this was to break communication between the boat and the laptop on the shore once the boat sailed out of range. Had a wireless access point been used this problem would not have occurred and communications would have re-established themselves when the boat came within range of the laptop again, this could also be achieved without using a physical access point as many wireless cards can emulate an access point using the Hostap software [43] under Linux. Unfortunately the wireless cards available to this project (a Lucent Orinoco and Buffalo Airstaion) were not compatible with hostap and no hardware access point was available. It is also likely that a hardware access point would have been capable of transmitting further as most feature external antennas rather than the limited internal antennas of network cards.

9.3.6 Change the sail setting algorithm

The sail setting algorithm currently attempts to set the sail based upon the current wind direction and heading every time the main loop is executed. This proved to be exceptionally time consuming as the slightest change in either heading or wind could cause the sail to move and given the poor quality of the sail setting hardware this required several seconds. A better approach would have been to only attempt to set the sail every 20 or 30 seconds and when doing this to set the sail for the desired heading not the current heading. As the boat maybe oscillating from its desired course constantly moving the sail to a setting appropriate for the current course will only help to worsen the oscillation as it both increases the amount of time taken to compensate (as the rudder and sail cannot be moved simultaneously) and as the action of turning the sail actually causes the boat to turn. Doing this may add a small amount of complexity to the sail setting algorithms as the wind direction is currently measured in terms of the boat's present heading and in order to set the sail according to the desired heading the wind direction must also be expressed in terms of the desired heading and not the current heading.

9.4 Possible directions for future work

9.4.1 Hardware Improvements

A number of limitations were found with the current boat and its on-board hardware, several of these caused difficulties in creating effective control software. This section outlines some of the hardware problems which have been encountered and possible solutions to them.

9.4.1.1 Improved Power Source

The boat's power supply currently consists of two lead acid batteries, these are estimated to be able to provide approximately 12-36 hours of continuous operation[44]. While more than ample for simple in-shore testing, this is not suitable use in a system for long term operation. Any such system would most likely need to employ solar panels coupled with on board batteries. Another possibility is to use some kind of energy recovery strategy similar to regenerative breaking in hybrid cars or kinetic watches, such a system could take advantage of movements caused by waves.

9.4.1.2 Improved Sensor and Servo Accuracy

A major problem throughout the project was the accuracy of the sensors and servos, in particular the sail servo/position sensor and wind sensor. Such inaccuracies made it very difficult to choose an appropriate course or to set the sail correctly, this resulted in the boat either sailing inefficiently, not sailing at all or being pushed off course by the wind. If these problems could be prevented then course accuracy could be greatly improved and the boat would be able to sail more efficiently and go where it had been told to go.

9.4.1.3 Larger Sail size

The sail used on this boat is not particularly efficient as a result of its small width, this size was chosen in order to prevent the boat getting out of control and capsizing. Any production robot would need to have a reasonable size sail, however at the same time the software would need to understand the need to deliberately reduce sailing efficiency during high winds in order to reduce strain on the sail and prevent the boat from capsizing. This is especially important when sailing close to the wind as sailing vessels tend to lean over when sailing in this manner.

9.4.1.4 Larger Boat size and shape more suited to sea conditions.

Given the current boats small size it is unlikely to be able to sail effectively in storm conditions at sea, making it unusable for use as a replacement to data buoys. In order to solve this problem the entire boat would need to be scaled to a suitable size, however this should not require any major changes to the software.

9.4.1.5 Fault Detection and Redundancy

A key need when operating a sailing robot in open waters is that of survivability. This is both in terms of surviving faults and surviving the elements (which may cause faults in the onboard hardware). In order to counter any hardware faults they must first be detected. Fault detection can take place through a number of different techniques, these include sensors that will detect

faults such as irregular outputs from electrical circuits, leaks and non-moving servos. Faults detection can also work on the principle of checking for responses from hardware, for instance if one CPU is unable to get a response from another then it can be assumed that a fault has occurred either in the other CPU or the circuits linking them.

A system commonly adopted for fault tolerance in safety critical systems such as aircraft and industrial control systems is that of redundancy. In such systems odd numbers of CPUs (often of different architectures) are tasked with the same role and their outputs are connected to some kind of voting mechanism which then carries the majority decision. If a CPU fails in such a system then it is still possible to operate the system using the remaining processors. It is also common to implement multiple actuators especially in aircraft systems, in this case each processor may be connected to several actuators each of which is sufficiently capable of performing the role on its own. Within the context of a sailing robot such systems could be implemented as multiple servos, multiple microcontrollers and possibly even multiple masts/sails and multiple rudders. There would also be the need for sensors to verify which of these was working, in the event of failure it maybe possible for the boat to alert some kind of control centre via a long distance radio or satellite link so that pickup (or having the boat sail home) and repair can be arranged.

9.4.1.6 Scientific Instrumentation and access to data

The current robot includes no scientific instrumentation such thermometers, barometers, sonars etc which can be used to perform some kind of useful data gathering. In order to be of any use to oceanographers, meteorologists, marine biologists, environmentalists etc such data must be recorded and both the hardware and software must support it and ideally be able to transmit data at regular intervals. Possible instruments might include thermometers, barometers, fish finders, hydrophones and water analysis equipment. In order to improve the usability of data from hydrophones it would be useful to be able to classify noises in order to determine the type of sound and if made by an animal the species of animal and number of animals, this could then be combined with data from other boats in order to triangulate the position of any animals detected. Preliminary work in using neural networks to classify such sounds has already been undertaken by Howell , Wood and Koksal (2003) [45]. Ideally the data being received from a series of sailing robots could be placed on the Internet in an open and portable format such as XML. This approach is currently being taken by the United States National Oceanic and Atmospheric Administration (NOAA) for weather data [46]. In doing this scientists from a variety of disciplines would be able to access and make use of the data from a network of sailing robots helping to reduce the number of observation systems required and increasing the amount of science which can be performed.

9.4.1.7 Navigation

At present the robot's only navigational aid is a compass, although this allows it to follow a specified heading it does nothing to detect lateral forces such as ocean currents which may move the boat without altering its heading. In order to either keep a fixed position or sail a long distance course some form of positional information will be required, this most obvious way to obtain such information is via GPS. Given the present hardware and software it would not be difficult to add GPS support, all that is required is for a GPS receiver to be connected to the Basic Stamp and for code to be written to obtain an NMEA string from it via the Basic Stamp and process this string on the Psion. Given this data it would be possible to determine how far the robot is from a designated point and to generate a bearing to that point and feed this into the existing heading error detection code.

9.4.1.8 Communications

In order to prove of use to any kind of remote monitoring or multi-robot based application it will be necessary to place some form of communications equipment on-board. Perhaps the most obvious choice is a satellite modem to allow for communications regardless of location, for coastal operations it maybe possible to substitute this for terrestrial systems such a GSM Mobile Phone using a GPRS packet radio link. For inter-robot communication some form of high bandwidth, point-to-point, low latency and low cost system would be desirable depending on the exact distance required wireless networking systems such as 802.11 (WI-FI) or 802.16 (WIMAX) might be practical. Such systems would also be highly useful for testing scenarios as large amounts of telemetry data can be sent in realtime and instructions can be issued to the robot in order to recover it from problem situations such as it loosing its tether and sailing in the wrong direction!

9.4.2 Software Improvements

9.4.2.1 Improved concurrency and modular design

The current software implementation makes no attempt to implement any concurrency, all commands are issued in series with each command waiting for the previous one to finish before taking place. If a concurrent system were implemented then it would be possible to be performing a turn while positioning the sail or taking a wind reading, this would greatly reduce the amount of time taken to perform a maneuver and would most likely increase the accuracy at which a course could be sailed. It would also be vital for allowing for communications, navigation and scientific readings to be taken without disrupting the ability to sail during the same time. However implementing concurrency greatly increases the complexity of the software and would require a complete rewrite of the Basic Stamp code if a Basic Stamp were being used, it would be far simpler to implement concurrency on a single microcontroller system that had both the processing power of the Psion and the I/O facilities of the Basic Stamp. Given a concurrent system it would be possible to split the system into a number of different software modules which could run concurrently, these could include keeping the boat on its desired heading, setting the sail correctly, logging scientific data, communications and performing navigation (and feeding the results of this into the steering algorithm).

9.4.2.2 Self Tuning Algorithms

Perhaps the biggest limitation of the current implementation is that the algorithms must be manually tuned. A number of techniques exist which can be used to tune and re-tune both PID and fuzzy logic controllers. It is possible to automatically apply the Zeigler and Nichols ultimate cycle method or to use a genetic algorithm [47] to tune a PID controller where each of the gain constants is evolved using a genetic algorithm with the fitness function giving the best weighting to the values which cause the least oscillation in the system. Similar techniques also exist for fuzzy logic controllers [48, 49] and these work by altering the set boundaries as and when they are required. In order for such systems to work on a sailing robot it is necessary to constantly measure the course (or perhaps speed) of the boat either through GPS or a compass in order to detect if the algorithm has correctly tuned itself or not and to feed this information back into the tuning process.

The algorithm evaluation methods developed for this project could also be used to form the basis of a fitness function for a genetic algorithm. Genetic algorithm based tuning could also be

undertaken by beginning the training phase within a simulator and then moving onto the robot after the initial training phase is complete, a similar system has successfully been used by Walker and Wilson (2002) [50] using a wheeled Kephera robot. However in order to achieve this with a sailing robot the simulator accuracy needs to be greatly improved. Such techniques may allow the system to adapt to changes in its sailing characteristics which may result either from damage or stormy weather. For example if the rate of turn was reduced by strong currents then the proportional constant could be increased resulting in larger rudder movements to compensate for the strong currents.

9.4.2.3 Improving adaptability and survivability with Biologically inspired techniques.

In addition to self-tuning algorithms it maybe desirable to introduce biologically inspired survival instincts into the system to allow it to cope better with extreme situations and in particular situations which were not conceived by the system designer. A possible application for this could be in the conservation of battery power, when power is low the boat can react in a similar way to animals rationing their food as supplies run low, as power levels drop non-vital systems such as communications and navigation are shutdown and power to other systems is reduced (possibly by making steering movements less frequently or making smaller moves until power levels increase). In addition to survivability there is the potential to use biologically inspired techniques to improve adaptability, it is possible to treat the boat like the body of an animal and when it is damaged (the equivalent of harm, injury or infection in biological terms) then an appropriate anti-body is released these anti-bodies take the form of corrective actions such as changing tack so that water does not spill in through a leak on a particular side of the boat, deploying pumps to remove water or shutting down replicated systems in order to prevent damaged electronics interfering with the rest of the system. Additional possibilities such as swarm behaviours maybe useful in co-ordinating a number of boats to perform a single mission between them. Despite the possibilities of such biologically inspired techniques their benefit must be considered against the benefits of simpler more traditional control systems which are likely to be cheaper and easier to implement as well as more predictable in their responses.

9.4.2.4 Weather Awareness

If being operated on an open sea it maybe desirable depending on the exact mission requirements to avoid storms. In order to achieve this the boat must know where the storm is in relation to itself and which direction to move in. Such information could either be gathered directly by the robot itself, perhaps through RADAR or visual systems or it can be supplied remotely to the robot from weather satellites. There may also be the need to reverse this process and have the boat deliberately sail into storms in order to study them.

9.4.2.5 Use of new cross compilation systems

Since the start of this project several new systems have emerged for cross-compiling code for a Linux environments. The most notable being embedded Gentoo [51] and the T2 project [52]. Both of these are systems which aim to cross-compile a complete installable Linux distribution for a given host architecture and allow the user to select any of thousands of available programs to be placed into that distribution. This is a logical step on from the uClibc/uClinux build system which only features a very narrow set of core applications by default and requires the user to manually cross-compile any additional applications that they may require. The use of such systems would have greatly eased the creation of a suitable Linux distribution given that

a great amount of time was spent cross-compiling applications which were not available from uClibc's build system (or which refused to compile using it, e.g. cardmgr the PCMCIA card manager and depmod the kernel module dependencies calculator).

9.4.3 Fleet Management and Telecommunications

Many of the possible applications for a sailing robot involve the use of multiple robots. In such scenarios it will be necessary to manage the fleet either through a centralised system or by having the robots communicate and co-ordinate amongst themselves. This would obviously require long distance communication systems and ideally GPS navigation on each robot as discussed in section 9.4.1.8. Currently most data buoy networks such as TOA and TRITON are organised in grid patterns, therefore it would seem logical that the fleet management system would be able to co-ordinate a group of sailing robots to form themselves into a grid pattern over a large area such as the South Pacific. Such a system should also be able to reorganise the grid pattern should a single robot fail, it would be possible to reorganise the grid to avoid a single large gap and increase the distance between each robot slightly allowing for a graceful degradation of the group of robots rather than leaving significant gaps as currently occurs in fixed data buoy networks. It would also be useful for such fleet management systems to be able to increase or decrease the distance between robots, move the entire fleet to another area or have individual members of the fleet move around or return home.

9.4.4 Collision Avoidance

Any sailing robot which is going to be moving upon the open sea or more importantly in coastal waters will need to avoid collisions with other vessels, animals, the shore, the sea bed and submerged hazards such as wrecks. The current implementation makes no attempt at all to avoid such collisions. There are several ways in which the risk of collision can be reduced, onboard sensors can be used to detect nearby objects and alter the boat's desired course to avoid them or the boat can be equipped with "knowledge" about the world which it can then reference against its current GPS position. Such knowledge might include databases from ocean surveys showing sea depth, the location of the coastline and the location of exclusion areas such as military test ranges, wildlife reserves, shipping lanes etc.

9.4.4.1 Detecting imminent collisions via World Models and Geographical Information Systems

As previously stated collisions can be prevented by storing information about hazardous areas which must be avoided. This could possibly be done by using a Geographical Information System which contains different layers of information, such layers might include the location of the coastline and no-go areas like shipping lanes. By using such information, if the robot senses it is nearing such an area then it can adjust its course to avoid the collision. This can be compared to so called softwall systems being developed for aircraft which prevent them from passing into certain restricted areas such as the centre's of major cities.

9.4.4.2 Detecting Collisions via realtime sensors

Collision detection via Geographical Information Systems will never be capable of avoiding collisions with dynamic parts of the environment such as ships, animals and even shifting sand banks. In order to avoid collisions with such objects some form of onboard sensing system will be needed, such sensors could take a number of forms. The most obvious method for avoiding

collisions with surface objects is to use RADAR, however RADARs tend to be expensive, bulky and may suffer problems in heavy seas as all the radar will be able to see is the few metres around the boat before a wave obscures its view. A possible alternative might be some form of visual sensor using onboard cameras and possibly night-vision equipment, however visual systems also suffer in heavy seas, at night and in heavy rain or fog. In addition to detecting surface hazards it may be necessary to detect underwater hazards, especially if the robot is being used in shallow waters. The most obvious technique for detecting underwater hazards is active SONAR, however visual systems may also be appropriate. Passive SONAR, also known as hydrophones may also be helpful especially for detecting nearby animals by listening for their sounds and using multiple hydrophones to detect the direction. Hydrophones may also aid the detection of nearby ships or any other hazard which creates noise in the water, they may also be helpful in fulfilling an oceanography role as they may help to detect underwater seismic events.

If many robots are being used in close proximity to each other then it is possible that they may be able to inform each other of the locations of hazards or at least provide information which may lead to accurately determining the location of a hazard. For example if there is a ship in between two sailing robots and they both report the direction at which they sense it, by knowing each other's location (via GPS) then it should be possible for both robots to estimate the location of the ship. One problem with this approach is that the information requires constant updating, if the same situation is detected 10 minutes later then it is difficult to determine if its the same ship or a new one, additionally the robots need to be able to tell the difference between a ship and an island or some other static object. Ultimately a good collision avoidance system will bring together knowledge from a variety of sensors, other robots and static knowledge about the world.

9.5 Conclusion

This project has shown that it is possible to achieve some level of control over a sailing robot using only low cost off the shelf PDA and simple microcontroller, however it has also identified the complexity of this task and shown that achieving a usable system will take a vast amount of time and fine tuning. Two different control strategies, a PID controller and a fuzzy logic controller have been proven to work effectively at controlling a simulated boat. This project has also succeeded in creating a framework for implementing more complex control strategies and a simulator in which they can be tested. Further work is still required to improve the accuracy of the simulator, the response times of the control algorithms and the robustness of the hardware.

Glossary

802.11

A series of standards for wireless networking operating in the unlicensed 2.4ghz frequency band. Typically it is possible to achieve distances of several hundred metres or more when specialist antennas are used.

ARM

A common type of CPU often (but not exclusively) used in embedded applications including the Psion and Jornada PDAs.

ASC

Autonomous Surface Craft, a floating vessel which is able to control itself without human intervention.

AUV

Autonomous Underwater Vehicle, an underwater vehicle such as a submarine which is capable of controlling itself without human intervention.

Bluetooth

A short distance wireless communication system intended to replace data cables for devices such as printers and mobile phones. It is possible to run a TCP/IP network over bluetooth and to use it over ranges of several hundred metres making it a possibility for configuring and controlling a robot remotely.

Compact Flash

A small memory storage card measuring about 2cm x 3cm. Able to function as an alternative to a hard disk drive, capacities range from 1 megabyte to 8 gigabytes.

CORBA

Common Object Request Broker Architecture, an object-oriented system for communicating objects between different programming languages.

DBCP

Data Buoy Co-operation Panel. A multi-national group co-operating on a number of data buoy related issues including safety and design.

EEPROM

Electronic Erasable Programmable Read Only Memory. A type of memory chip who's contents can be reprogrammed electronically. Such memory is used on the Basic Stamp to store the current program.

EGOS

European Group on Ocean Stations.

EPOC

The operating system supplied on Psion PDAs, a newer version known as Symbian is also found in many modern mobile phones.

GCC

Gnu Cross Compiler, a suite of open source compilers with the ability to build executables for other platforms. For example building executables for an ARM processor from a system with an Intel processor inside it.

GOOS

Global Ocean Observing System, a global network of sensors monitoring the state of the worlds ocean's.

GPS

Global Positioning System, a network of satellites which transmit time signals allowing a receiver to detect its position within a few metres.

Hydrophones

A system for listening to noises in the water using microphones, sometimes known as passive sonar. If multiple directional sensors are used then it is possible to determine the direction of a given noise, if sensors are placed in multiple locations then an approximate location of a noise can be detected via triangulation.

IIOP

Internet Iter ORB Protocol, A system for transmitting CORBA objects via the internet.

IrDA

Infra-Red Data Association, a standard for allowing computer equipment to communicate via infra-red.

Linux

An open source Unix like operating system kernel. Started in 1991 by Linus Torvalds and now available for most computing platforms.

NetBSD

Another open source Unix like operating system. Based upon the original BSD Unix, NetBSD aims to offer a highly portable architecture allowing it to run on virtually any computing platform.

PCMCIA

Personal Computer Memory Card International Association, a standard for credit-card sized memory cards for handheld and laptop PCs. Often used for expansion cards such as network cards, modems etc.

NMEA

National Marine Electronics Association, A standard interface for GPS receivers which produces a human readable string representing the current location.

NOAA

The United States National Oceanic and Atmospheric Administration. An organisation responsible for oceanography, meteorology and similar sciences in the USA. NOAA are responsible for a large number of data buoys worldwide.

PDA

Personal Digital Assistant, a general term for small handheld computers such as the Palm Pilot, Psion and Jornada.

PIRATA

Pilot Research Moored Array in the Tropical Atlantic, a network of data buoys in the centre of the Atlantic ocean.

POSIX

Portable Operating System Interface, a set of common functions for operating systems to implement. Theoretically a program written to POSIX standards can be compiled and run on any POSIX compliant operating system without modification.

PWM

Pulse Width Modulation, a technique to encode data to/from hardware devices via the amount of time a signal is high. PWM is used by many of the hardware components such as compass in the sailing robot.

RPC

Remote Procedure Call, a generic term for calling a computer program running on a remote system or using techniques which would allow remote communication but on the same system.

RS-232

A standard for serial communications, most PCs, laptops, PDAs and some microcontrollers including the Basic Stamp.

SDK

Software Development Kit, a term for a suite of programs to aid developers for a given system.

TCP/IP

Transmission Control Protocol/Internet Protocol. The standard protocol for communicating data via the internet, TCP/IP can also be used for two programs running on the same system to communicate with each other via a system known as a loopback network.

TOA

Tropical Atmosphere Ocean project, a system to deliver real time data from the pacific ocean in order to monitor the El Niño and La Niña.

TOGA

Tropical Ocean Global Atmosphere, part of the World Climate Research Program. Aimed at studying rapid changes in the ocean conditions such as El Niño and La Niña.

TRITON

Triangle Trans-Ocean buoy Network, a network of data buoys operating in Western pacific by the Japan Marine Science and Technology Center (JAMSTEC).

uClibc

A highly optimised (in terms of size) standard C library for use with the uClinux distribution. It is targeted at embedded applications and offers a cross-compiler to recompile any programs for the target system. Programs compiled for uClibc will not run on systems using the standard glibc and vice-versa.

uClinux

A linux distribution based around uClibc, it mostly consists of a program known as busybox which provides virtually all common unix command line utilities compiled into a single executable. It offers a few extra packages for programs which are not part of busybox such as wireless tools and cardmgr the PCMCIA configuration utility.

Userspace

A term to describe normal user programs running on a Unix or Linux system. The reverse of this is kernel space which refers to programs which run as part of the operating system kernel.

Bibliography

[1] Operation and achievements of the dbcp. Technical report, Data Buoy Co-operation Panel, 2005.

[2] Halloran P Bisbee, J and N. Larkin. *Sailing and the Tech Dingy - Instruction Manual*. Massachusetts Institue of Technology, Cambridge, MA, USA, 1995.

[3] C. Kilian. *Modern Control Technology: Components and Systems*, chapter 11, pages 361–409. Number ISBN: 076682358X. Delmar Thomson Learning, 2002.

[4] H. Diamond. The united states detailed national report on systematic observations for climate: United states global climate observing system (us-gcos) program. Technical report, National Oceanic and Atmospheric Administration, 1335 East-West Highway, Room 7214, Silver Spring, Maryland, USA, 20910, 2001.

[5] A. Hageberg. Report on the egos management committee meeting. Technical report, European Group on Ocean Stations, Christian Michelsen Research AS, P.O. Box 6031 Postterminalen. N-5892 Bergen, Norway, 2001.

[6] The ship of opportunity program. http://www.brest.ird.fr/soopip (Accessed on May 2nd 2005).

[7] M et al McPhaden. Effects of fishing activity on tropical moored buoy arrays. http://www.dbcp.noaa.gov/dbcp/fishing_effects.zip (Accessed on May 2nd 2005).

[8] United kingdom report on systematic observations for climate for the global climate observation system. Technical report, Department for Environment, Food and Rural Affairs, Nobel House, 17 Smith Square, London, SW1P 3JR, UK, 2001.

[9] Us launch vehicle data - cost estimating web site. http://www.jsc.nasa.gov/bu2/ELV_US.html (Accessed on May 2nd 2005), 2004.

[10] J. Abril, J. Salmon and O Calvo. Fuzzy control of a sailboat. *International Journal of Approximate Reasoning*, 16(3-4):359–375, April - May 1997.

[11] B Ross. Robot boat project. http://www-2.cs.cmu.edu/~br/CbotWeb/rb98.html (Accessed on May 2nd 2005), 1998.

[12] T.W. Vaneck. Fuzzy guidance controller for an autonomous boat. *IEEE Control Systems Magazine*, 17(2):43–51, 1997.

[13] C. et al Goudey. A robotic boat for autonomous fish tracking. *MTS Journal*, 32(1):47–53, 1998.

[14] Massachusetts Institue of Technology. Auv laboratory at mit sea grant. http://auvlab.mit.edu/ (Accessed on May 2nd 2005).

[15] R. Rocca. Roboat. *Circuit Cellar*, Issue 115:32, Feb 2000.

[16] R. Steidley, C. Bachnak and W. Lohatchit. Developing a remote controlled vehicle for environmental studies. 2004.

[17] A. Steidley, C. Sadovski and R. Backnak. Intelligent systems integration for data acquisition and modeling of coastal ecosystems. *Innovations in Applied Artificial Intelligence*, 11b(3):1112–1122, 2004.

[18] M. S. Triantafyllou and G.S. Triantafyllou. A efficient swimming machine. *Scientific American*, March 1995:40–48.

[19] Papadopoulos E. Tzeranis, D. and G. Triantafyllou. On the design of an autonomous robot fish. In *Proc. 11th IEEE Mediterranean Conference on Control and Automation, (MED '03)*, 2003.

[20] G. et al Griffiths. Open ocean operation experience with the autosub-1 autonomous underwater vehicle. In *Proceedings of the Unmaned Untethered Submersible Technology Symposium, New Hampshire, USA*, August 1999.

[21] A. J. Healey and D. B Marco. Command, control and navigation: Experimental results with the nps aries auv. *IEEE Journal of Oceanic Engineering*, 26(4):466–477, 2001.

[22] J. Turton. Argo 2002: Progress towards a global array of profiling floats. *Sea Technology*, 44(10):33–36, 2003.

[23] R. et al Bachmayer. Underwater gliders: Recent developments and future applications. *Proceedings of the IEEE International Symposium on Underwater Technology (UT'04), Tapei, Taiwan*, 2004.

[24] Bellingham J Catipovic J. Curtin, T.B and D. Webb. Autonomous oceanographic sampling networks. *Oceanography*, 16(3), 1993.

[25] Monterey Bay Aquarium Research Institute. Mbari - autonomous ocean sampling networks. http://www.mbari.org/aosn/ (Accessed on May 2nd 2005).

[26] C. Aartijk, M. Taliola and P. Adriaans. The robosail project. In *Proceedings of European Conference on Artificial Intelligence*, pages 653–657, RoboSail Systems BV, Waterlandlaan 120, 1441 RW Pu, 2002. Robosail Systems.

[27] Smart boat corporation. http://www.smartboat.com/ (Accessed on May 2nd 2005).

[28] Artificially damaged triton buoy - photo record. http://www.jamstec.go.jp/jamstec/TRITON/ (Accessed on May 2nd 2005), 1999/2000.

[29] S. Showalter. The legal status of autonomous underwater vehicles. *The Marine Technology Society Journal*, 38(1):80–83, 2004.

[30] Basic stamp 2sx module. http://www.parallax.com/detail.asp?product_id=BS2SX-IC (Accessed on May 2nd 2005).

[31] *Psion Series 5mx Users's Manual.* Psion Inc, 150 Baker Avenue, Concord, MA 01742, US.

[32] Palm One, 400 N. McCarthy Blvd. Milpitas, CA 95035, USA. *Handbook for the Palm m100 Handheld.*

[33] The netbsd project. http://www.netbsd.org (Accessed on May 2nd 2005).

[34] The linux kernel archives. http://www.kernel.org/ (Accessed on May 2nd 2005).

[35] The openpsion project linux for psion computers. http://www.openpsion.org (Accessed on May 2nd 2005).

[36] D Kegel. Building and testing gcc/glibc cross toolchains. http://kegel.com/crosstool/ (Accessed on May 2nd 2005).

[37] uclibc - a c library for embedded linux. http://www.uclibc.org (Accessed on May 2nd 2005).

[38] T. Kuusela and Brockmann S. The tracksail project. http://tracksail.sourceforge.net (Accessed on May 2nd 2005), 2002.

[39] Daventech Ltd. Cmps03 - robot compass module. http://www.robot-electronics.co.uk/htm/cmps3doc.shtml (accessed on May 2nd 2005).

[40] The familiar project. http://familiar.handhelds.org/ (accessed on May 1st 2005).

[41] Gumstix - finally a pack of very small linux machines. http://www.gumstix.org/ (Accessed on May 2nd 2005).

[42] The lart pages. http://www.lart.tudelft.nl/ (Accessed on May 2nd 2005).

[43] J. Malinen. Host ap driver for intersil prism 2/2.25/3, hostapd, and wpa supplicant. http://hostap.epitest.fi/ (accessed on May 2nd 2005).

[44] M. Neal. A hardware proof of concept for a free sailing robot for oceanic observation. (unpublished). Journal of Oceanic Engineering, 2005.

[45] Wood S. Howell, B.P. and S. Koksal. Passive sonar recogition and analysis using hybrid neural networks. *Proceedings of IEEE Oceans 2003*, 2(22-26):1917–1924, 2003.

[46] National Oceanic and Atmospheric Administration. Experimental national digital forecast database xml web service. http://weather.gov/xml/ (Accessed on May 2nd 2005).

[47] I. Griffin. On-line pid controller tuning using genetic algorithms. Master's thesis, Dublin City University, 2003.

[48] G. Tan and H. Xiheng. More on designing fuzzy controllers using genetic algorithms: Guided constrained optimisation. 1997.

[49] J. Foran. Optimisation of a fuzzy logic controller using genetic algorithms. Master's thesis, Dublin City University, 2002.

[50] J. Walker and M. Wilson. How useful is lifelong evolution for robots? In *In: From Animals to Animats 7, Proceedings of the 7th International Conference on the Simulation of Adaptive Behaviour (SAB-02)*, 2002.

[51] The Gentoo Foundation Inc. Gentoo linux projects - embedded gentoo. http://www.gentoo.org/proj/en/base/embedded/index.xml (accessed on April 30th 2005).

[52] The t2 project. http://www.t2-project.org/index.html (accessed on April 30th 2005).

[53] C.P. Summerhays. Goos moves into implementation. *Sea Technology*, August 1997.

[54] Flemming. Making the case for goos. *Sea Technology*, January 1995.

[55] S. Hansen. Sea watch - connecting to goos. *Sea Technology*, February 1995.

[56] S.R. Piotrowicz. Toga observing system and goos. *Sea Technology*, August 1997.

Appendix A

Diagrams and Photos showing a detailed design of the robot.

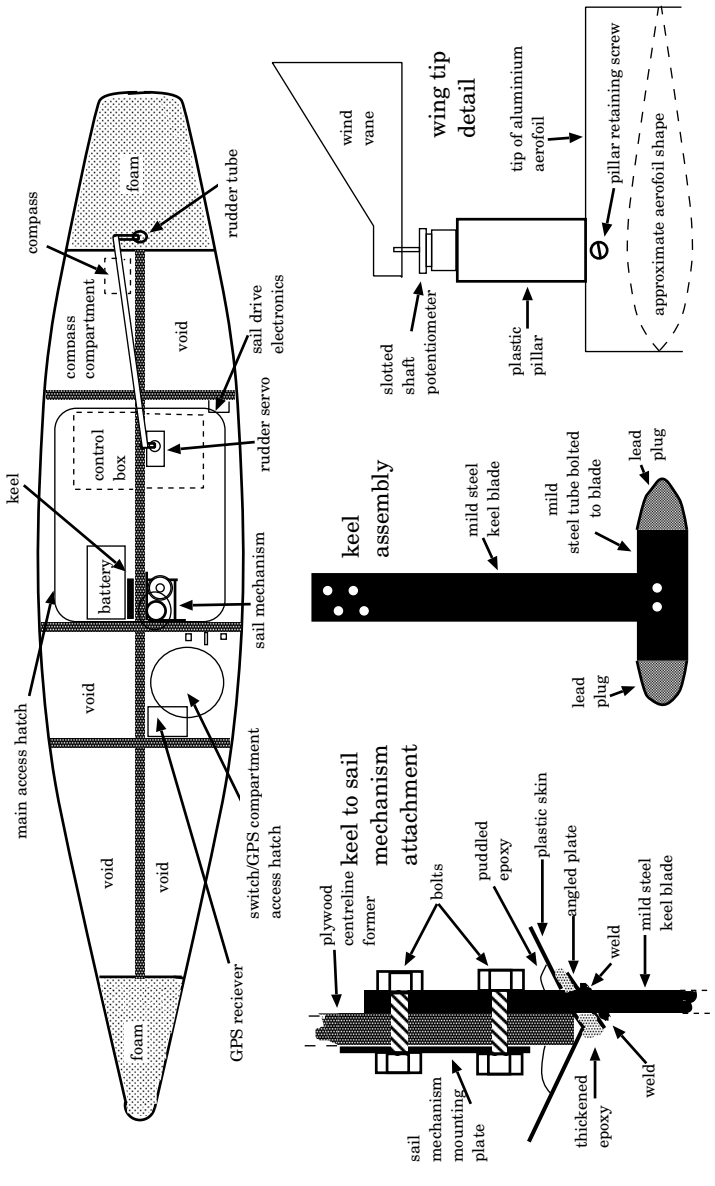

Figure A.1: Diagram showing the parts of the boat in detail. Courtesy of Dr. Mark Neal.

Appendix B

Paper protractor used for calibration measurements

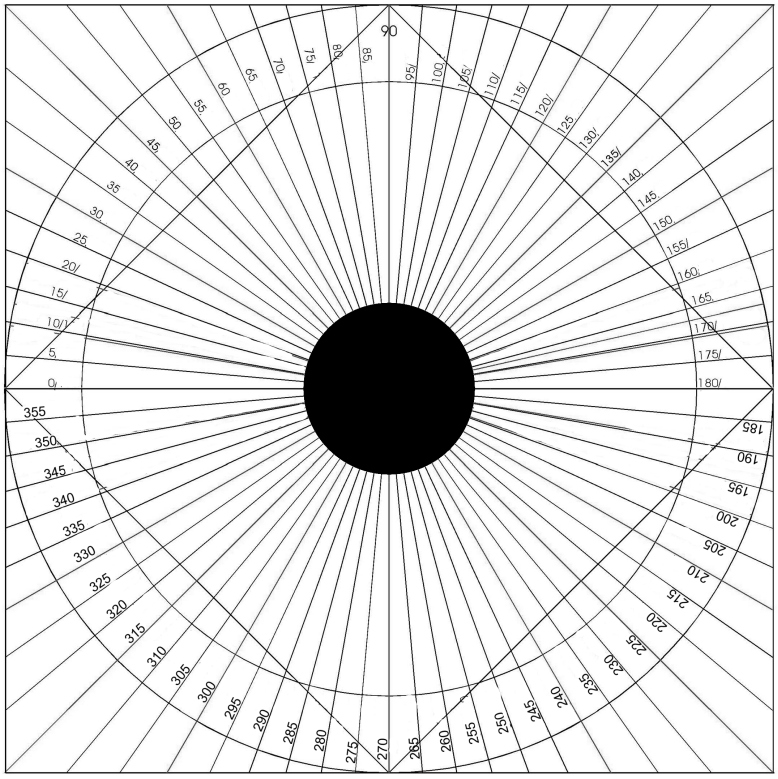

Figure B.1: The protractor which is placed around the mast in order to calibrate the sail.

Appendix C

Original protocol design as supplied by Dr. Mark Neal.

Commands:

SRD x : set rudder position reply: null

SSL x : set sail position reply: null

SLP x : go to sleep for x seconds reply: null

Updates:

UGP : switch on and update GPS position reply: null

UWS : switch on and update wind sensor values reply: null

UCH : switch on and update compass reading reply: null

Transmit Requests:

GWS : get wind sensors reply: a b : sensor readings for the two strain gauges

GCH : get compass heading reply: a : compass heading

GPN : get GPS Northing reply: complex_string

GPW : get GPS Westing reply: complex_string

GSL : get sail position reply: a : sail position

GRD : get rudder position reply: a : rudder position

GBT : get battery voltage reply: a : battery voltage

GCX : get custom sensor reading reply: a : unknown (to be logged and inspected later)

All packets are the same length. All will be formatted text for readability. Packet out from Palm will be three characters specifying command and one character of data. All replies will be ten characters. Interpretation to be performed at each end.

Examples from Palm:

SRD

Will tell the Stamp to set rudder to position 34 (ASCII for)

UGP!

Will tell Stamp to start-up the GPS and store it's current reading ready for transmission to the Palm. ! will be ignored.

GBT!

Will cause Stamp to send back the battery voltage. ! will be ignored.

Examples from Stamp:

Upon receiving GBT! form Palm the Stamp will transmit a ten character packet:

t!!!!!!!!!

t is the value 116. This will signify a particular battery voltage.

Upon receiving GPN! from Palm the Stamp will transmit:

N05223.848

which is a valid Northing from the GPS. This requires all 10 characters.

Appendix D

Differences between original Basic Stamp code and final version.

! indicates a changed line

- indicates a removed line

+ indicates a new line

```
*** newcontrol.bs2.orig 2005-04-22 19:48:05.000000000 +0100
--- newcontrol.bs2 2005-04-22 20:02:24.000000000 +0100
***************
*** 1,14 ****
'{$STAMP BS2sx}
- winda CON 0 ' Constants defining locations in scratchpad for state information
- windb CON 1
compass CON 2
- northing CON 3
- westing CON 13
rudder CON 23
sail CON 24
- battery CON 25
- custom CON 26
n96n CON $40F0 ' Constant defining 9600 no parity serial comms
--- 1,8 ----
***************
*** 19,60 ****
outpacket VAR Byte(10)
elapsed VAR Word
avg VAR Word
! tmp VAR Word
DEBUG "firing up", CR
start:
SERIN 0, n96n, [STR inpacket\4]
DEBUG STR inpacket\4,CR
! IF (inpacket(0) = "S") THEN SETACTS
! IF (inpacket(0) = "G") THEN SENDDATA
! IF (inpacket(0) = "U" AND inpacket(1) = "G" AND inpacket(2) = "P") THEN RUNGPS
! IF (inpacket(0) = "U" AND inpacket(1) = "W" AND inpacket(2) = "S") THEN RUNWIND
! IF (inpacket(0) = "U" AND inpacket(1) = "C" AND inpacket(2) = "H") THEN RUNCOMPASS
! GOTO start
! SETACTS:
- ' DEAL WITH SETTING SERVOS AND SLEEP
- SEROUT 1, n96n, ["!!!!!!!!!!"]
! IF (inpacket(1) <> "R") THEN NOTRUD
SEROUT 10, n96n, [255, 0, inpacket(3)]
PUT rudder, inpacket(3)
! DEBUG "set rudder servo: ", DEC inpacket(3), CR
! GOTO start
!
! NOTRUD:
! IF (inpacket(1) <> "S") THEN NOTSAIL
```

```
HIGH 4
PAUSE 1
--- 13,60 ----
outpacket VAR Byte(10)
elapsed VAR Word
avg VAR Word
! max_elapsed VAR Word
! min_elapsed VAR Word
DEBUG "firing up", CR
start:
+ DEBUG "back at start", CR
+
SERIN 0, n96n, [STR inpacket\4]
DEBUG STR inpacket\4,CR
! IF (inpacket(0) = "S" AND inpacket(1) = "S" AND inpacket(2) = "L") THEN SETSAIL
! IF (inpacket(0) = "S" AND inpacket(1) = "R" AND inpacket(2) = "D") THEN SETRUD
! IF (inpacket(0) = "S" AND inpacket(1) = "U" AND inpacket(2) = "W" ) THEN SETUNWIND
! IF (inpacket(0) = "G" AND inpacket(1) = "W" AND inpacket(2) = "S") THEN GETWIND
! IF (inpacket(0) = "G" AND inpacket(1) = "C" AND inpacket(2) = "H") THEN GETCOMPASS
! IF (inpacket(0) = "G" AND inpacket(1) = "S" AND inpacket(2) = "L") THEN GETSAIL
! IF (inpacket(0) = "G" AND inpacket(1) = "R" AND inpacket(2) = "D") THEN GETRUD
! GOTO start
!
!
! 'set the rudder
! SETRUD:
!
SEROUT 10, n96n, [255, 0, inpacket(3)]
PUT rudder, inpacket(3)
! DEBUG "set rudder servo: ", DEC inpacket(3), CR
! GOSUB SENDRESPONSE
! GOTO start
! 'set the sail
! SETSAIL:
HIGH 4
PAUSE 1
***************
*** 94,290 ****
PUT sail, inpacket(3)
DEBUG "set sail: ", DEC inpacket(3), CR
! GOTO start
!
! NOTSAIL:
!
! IF (inpacket(1) <> "L") THEN NOTSLEEP
! DEBUG "going to sleep for: ", DEC inpacket(3), " seconds...", CR
! SLEEP inpacket(3)
! GOTO start
!
! NOTSLEEP:
!
! GOTO start
!
! SENDDATA:
!
! ' DEAL WITH TRANSMITTING DATA TO PALM
!
! IF (inpacket(1) <> "W") THEN NOTWIND
! GET winda, temp
! outpacket(0) = temp
! GET windb, temp
! outpacket(1) = temp
! SEROUT 1, n96n, [STR outpacket\10]
! DEBUG "Sent (wind) data: ", DEC outpacket(0), " ", DEC outpacket(1), CR
! GOTO start
!
! NOTWIND:
!
! IF (inpacket(1) <> "C" AND inpacket(2) <> "H") THEN NOTCOMP
! GET compass, temp
! outpacket(0) = temp
! SEROUT 1, n96n, [STR outpacket\10]
! DEBUG "Sent (compass) data: ", DEC outpacket(0), CR
```

```
! GOTO start
! NOTCOMP:
! IF (inpacket(1) <> "P" AND inpacket(2) <> "N") THEN NOTNORTHING
! FOR i = 0 TO 9
! GET (i + northing), temp
! outpacket(i) = temp
! NEXT
! SEROUT 1, n96n, [STR outpacket\10]
! DEBUG "Sent (northing) data: ", STR outpacket, CR
! GOTO start
! NOTNORTHING:
- IF (inpacket(1) <> "P" AND inpacket(2) <> "W") THEN NOTWESTING
- FOR i = 0 TO 9
- GET (i + westing), temp
- outpacket(i) = temp
- NEXT
- SEROUT 1, n96n, [STR outpacket\10]
- DEBUG "Sent (westing) data: ", STR outpacket, CR
- GOTO start
! NOTWESTING:
! IF (inpacket(1) <> "R" AND inpacket(2) <> "D") THEN NOTRUDDERSEND
GET rudder, temp
outpacket(0) = temp
SEROUT 1, n96n, [STR outpacket\10]
DEBUG "Sent (rudder) data: ", DEC outpacket(0), CR
GOTO start
! NOTRUDDERSEND:
! IF (inpacket(1) <> "S" AND inpacket(2) <> "L") THEN NOTSAILSEND
GET sail, temp
outpacket(0) = temp
SEROUT 1, n96n, [STR outpacket\10]
DEBUG "Sent (sail) data: ", DEC outpacket(0), CR
GOTO start
! NOTSAILSEND:
!
! IF (inpacket(1) <> "B" AND inpacket(2) <> "T") THEN NOTBAT
! GET battery, temp
! outpacket(0) = temp
! SEROUT 1, n96n, [STR outpacket\10]
! DEBUG "Sent (battery) data: ", DEC outpacket(0), CR
! GOTO start
!
! NOTBAT:
!
! IF (inpacket(1) <> "C" AND inpacket(2) <> "X") THEN NOTCUSTOM
! GET custom, temp
! outpacket(0) = temp
! SEROUT 1, n96n, [STR outpacket\10]
! DEBUG "Sent (custom) data: ", DEC outpacket(0), CR
! GOTO start
!
! NOTCUSTOM:
!
!
! GOTO start
!
! RUNGPS:
!
! ' DEAL WITH GPS
!
! SEROUT 1, n96n, ["!!!!!!!!!!"]
!
! DEBUG "Updating GPS readings", CR
!
! ' POWER UP AND WAIT FOR A FIX (pin 5 to switch it on, pin 6 for NMEA communication)
!
! ' run 1
!
! GOTO start
!
!
! RUNWIND:
! ' DEAL WITH WIND SENSORS
```

```
!
! 'SEROUT 1, n96n, ["!!!!!!!!!!"]
!
! ' DEBUG "Updating wind sensor values", CR
!
! ' DEAL WITH WIND SENSORS
!
! 'SEROUT 1, n96n, ["!!!!!!!!!!"]
!
! ' DEBUG "Updating wind sensor values", CR
HIGH 2
PAUSE 100
RCTIME 2, 1, elapsed
- 'SEROUT 1, n96n, [DEC elapsed]
- 'SEROUT 1, n96n, [CR]
-
outpacket(0) = elapsed/255
! outpacket(1) = elapsed - outpacket(0)
SEROUT 1, n96n, [STR outpacket\10]
-
DEBUG DEC elapsed, CR
-
GOTO start
! RUNCOMPASS:
! SEROUT 1, n96n, ["!!!!!!!!!!"]
! ' DEAL WITH COMPASS on pin 5
avg = 0
! compwt:
! IF IN3 = 0 THEN compwt
! RCTIME 3,1,elapsed
! elapsed = (elapsed - 900)/100
! IF (elapsed < 450) THEN compcool
! elapsed = 449
! compcool:
!
! avg = avg + elapsed
!
! 'debug "prelim ", ? elapsed, ? avg, CR
!
! FOR i = 1 TO 10
! wtl:
! IF IN3 = 0 THEN wtl
! RCTIME 3,1,elapsed
! elapsed = (elapsed - 900)/100
! IF (elapsed < 450) THEN compcooll
! elapsed = 449
! compcooll:
! avg = avg + elapsed
! tmp = avg/(i + 1)
! DEBUG ?tmp, ?avg, ?elapsed
! IF ((tmp + 100) - elapsed) < 150 THEN ok
! DEBUG "oops...", CR
! avg = elapsed * (i + 1)
! ok:
NEXT
! avg = avg/11
! avg = (avg * 255)/449
! PUT compass, avg
- ' run 3
! GOTO start
\ No newline at end of file
--- 94,227 ----
PUT sail, inpacket(3)
DEBUG "set sail: ", DEC inpacket(3), CR
! GOSUB SENDRESPONSE
! GOTO start
! SETUNWIND:
! HIGH 7
! HIGH 6
! PAUSE 6000
! LOW 6
! GOSUB SENDRESPONSE
! GOTO start
```

```
+ SENDRESPONSE:
! SEROUT 1, n96n, ["!!!!!!!!!!"]
! RETURN
! GETRUD:
! GOSUB CLEARDATA
GET rudder, temp
outpacket(0) = temp
SEROUT 1, n96n, [STR outpacket\10]
DEBUG "Sent (rudder) data: ", DEC outpacket(0), CR
GOTO start
! GETSAIL:
! GOSUB CLEARDATA
GET sail, temp
outpacket(0) = temp
SEROUT 1, n96n, [STR outpacket\10]
DEBUG "Sent (sail) data: ", DEC outpacket(0), CR
GOTO start
! GETWIND:
! GOSUB CLEARDATA
HIGH 2
PAUSE 100
RCTIME 2, 1, elapsed
outpacket(0) = elapsed/255
! outpacket(1) = elapsed - (outpacket(0)* 255)
SEROUT 1, n96n, [STR outpacket\10]
DEBUG DEC elapsed, CR
GOTO start
! GETCOMPASS:
! DEBUG "in RUNCOMPASS", CR
! GOSUB CLEARDATA
avg = 0
! 'compwt:
! 'DEBUG "waiting for IN11 to go nonzero", CR
! 'IF IN11 = 0 THEN compwt
! 'RCTIME 11,1,elapsed
! 'elapsed = (elapsed-1250)/125
! 'DEBUG "got data from compass", CR
! 'DEBUG DEC elapsed, CR
!
! 'min_elapsed = elapsed
! 'max_elapsed = elapsed
!
! 'FOR i=1 TO 5
! ' compwait:
! 'DEBUG "waiting for IN11 to go nonzero", CR
! 'IF IN11 = 0 THEN compwait
! 'RCTIME 11,1,elapsed
! 'elapsed = (elapsed-1250)/125
! 'DEBUG "got data from compass:", CR
! 'DEBUG DEC elapsed, CR
! 'IF elapsed < min_elapsed THEN NEWMIN
! 'IF elapsed > max_elapsed THEN NEWMAX
! 'endif:
! 'NEXT
!
! 'take the median value (half the range + min)
! 'avg = ((max_elapsed - min_elapsed) /2) + min_elapsed
!
! FOR i = 1 TO 30
! wt:
! IF IN11 = 1 THEN wt
! RCTIME 11,0,elapsed
! IF elapsed < avg THEN NOUPD
! avg = elapsed
! NOUPD:
NEXT
! avg = avg/200
!
! outpacket(0) = avg/255
! outpacket(1) = avg - (outpacket(0) * 255)
! SEROUT 1, n96n, [STR outpacket\10]
! DEBUG "AVG = ", DEC avg, CR
! DEBUG "Sent (compass) data: ", DEC outpacket(0), CR
```

```
! DEBUG "Sent (compass) data: ", DEC outpacket(1), CR
!
! GOTO start
!
! 'NEWMIN:
! ' min_elapsed = elapsed
! ' GOTO endif
! 'NEWMAX:
! ' max_elapsed = elapsed
! ' GOTO endif
! CLEARDATA:
!
! FOR i = 1 TO 9
! outpacket(i) = 0
! NEXT
!
! FOR i = 1 TO 9
! inpacket(i) = 0
! NEXT
!
! RETURN
\ No newline at end of file
```

Appendix E

Sail Servo and Wind Sensor Calibration Data

	Set 1		Set 2		Set 3	
Sail value	Sail Angle	Wind Reading	Sail Angle	Wind Reading	Sail Angle	Wind Reading
0	24	4462	37	3742	35	3983
2	54	3776	61	3308	42	3746
4	57	3725	63	2962	52	3556
6	52	3904	47	3690	53	3569
8	56	3798	57	3335	60	3212
10	58	3351	58	3393	52	3806
12	63	3448	64	3081	67	3239
14	70	3122	71	3030	62	2990
16	72	3234	76	2691	73	2768
18	74	2915	66	3307	76	2697
20	73	3166	76	2810	69	3064
22	74	3243	77	2763	84	2786
24	86	2763	80	2617	83	2427
26	82	2770	90	2319	77	2657
28	90	2443	88	2295	87	2483
30	95	2513	91	2251	90	2393
32	101	2119	95	2040	96	1901
34	101	2177	100	2011	103	1803
36	100	2176	102	2027	103	1722
38	104	2047	108	1804	105	1841
40	109	2284	105	1647	110	1580
42	110	1879	108	2037	112	1605
44	117	1950	118	1429	117	1446
46	123	1590	124	1115	122	1283
48	125	1471	122	1413	115	1784
50	136	1102	128	1065	123	1240
52	136	1113	137	819	125	1119

	Set 1		Set 2		Set 3	
Sail value	Sail Angle	Wind Reading	Sail Angle	Wind Reading	Sail Angle	Wind Reading
54	142	1041	137	803	126	1100
56	146	1000	145	546	141	636
58	152	911	150	435	144	619
60	148	977	152	227	155	183
62	155	735	156	190	157	227
64	159	376	158	111	167	1
66	164	324	160	73	172	1
68	168	276	167	1	175	1
70	170	217	173	1	172	1
72	182	1	184	1	173	1
74	191	1	185	60448	189	59071
76	193	1	190	59587	193	9187
78	197	9862	197	58650	196	10154
80	199	12886	202	8983	198	9063
82	202	9653	207	9233	204	9209
84	206	11181	208	9097	207	9681
86	213	9407	215	9133	210	9628
88	217	8764	215	9205	215	8864
90	223	6183	219	8538	225	8938
92	225	5917	227	8852	226	8611
94	230	5703	229	8571	230	8873
96	235	5415	235	8016	232	8489
98	239	5505	238	8074	237	8218
100	244	5368	242	8295	241	8240
102	245	5220	246	8414	242	8044
104	252	5310	235	8473	251	7915
106	253	7523	252	8277	241	8089
108	259	7581	257	7936	240	8660
110	262	7838	257	7797	257	7880
112	265	7465	265	7566	251	7949
114	270	7586	270	7167	258	8014
116	271	7418	275	7164	277	7400
118	267	7150	280	7234	282	7388
120	280	7304	274	7252	267	7783
122	283	7370	284	7270	275	7239
124	290	7342	289	7051	292	6831
126	296	6777	297	6494	297	6945
128	301	6425	302	6849	284	6821
130	309	6479	307	6417	305	7106
132	316	6308	314	6393	307	6223
134	318	5808	302	6525	307	6484

	Set 1		Set 2		Set 3	
Sail value	Sail Angle	Wind Reading	Sail Angle	Wind Reading	Sail Angle	Wind Reading
136	324	6090	310	6608	320	5966
138	327	5774	322	6143	309	6439
140	332	5806	325	6128	316	6045
142	338	5674	332	5520	333	5622
144	341	5513	340	5842	325	5721
146	340	5672	335	5963	341	5147
148	343	5561	345	5442	337	5863
150	347	5367	347	5251	342	5112
152	352	5191	354	5301	345	5553
154	356	4917	356	5213	353	4923
156	359	4765	2	5081	356	4993
158	2	4884	356	4937	5	4662
160	7	4479	4	4641	9	4700

	Set 4		Set 5	
Sail value	Sail Angle	Wind Reading	Sail Angle	Wind Reading
0	37	3683	33	4323
2	62	3185	57	3432
4	62	3194	52	3450
6	53	3483	55	3559
8	53	3328	57	3347
10	62	3123	62	3034
12	64	3017	66	2966
14	68	2961	73	2988
16	70	2901	75	2744
18	77	2823	78	2519
20	75	2838	74	2781
22	82	2669	77	2763
24	81	2656	80	2513
26	88	2247	90	2303
28	90	2538	82	2699
30	95	2138	87	2361
32	99	2063	97	2076
34	100	2019	101	1978
36	104	1912	99	1891
38	107	1712	105	1883
40	105	1944	107	1881
42	107	1770	114	1496
44	112	1705	119	1421
46	122	1332	120	1297
48	124	1125	127	1099
50	127	1190	125	1103
52	131	1043	1133	982
54	136	940	133	1047
56	142	781	140	661
58	144	638	145	460
60	158	261	147	408
62	162	201	155	220
64	164	47	157	106
66	157	318	168	1
68	163	99	161	209
70	172	1	161	29
72	182	1	180	1
74	187	58330	187	60128
76	193	10701	190	58791
78	195	8947	195	10573
80	199	9208	197	9685

	Set 4		Set 5	
Sail value	Sail Angle	Wind Reading	Sail Angle	Wind Reading
82	204	9406	201	9367
84	205	9182	205	9258
86	208	8796	210	8811
88	215	8769	216	9040
90	217	8725	218	8928
92	221	8820	220	8653
94	226	8578	224	8542
96	228	8475	231	8700
98	232	8183	234	8465
100	240	8363	225	8588
102	242	8296	240	8223
104	244	8143	247	7896
106	245	8229	250	7558
108	257	7779	256	8048
110	258	7445	257	7662
112	251	7734	262	8020
114	259	7966	267	7717
116	272	7445	272	7530
118	265	7517	276	7008
120	274	7640	265	7868
122	276	7068	280	7081
124	290	7156	284	7599
126	294	6811	290	6754
128	299	6882	297	7126
130	300	6278	301	6611
132	307	6415	306	6522
134	315	6519	306	6714
136	314	6293	304	6255
138	319	6259	315	6073
140	330	5771	322	5761
142	331	6102	326	6102
144	330	5954	334	5807
146	337	5684	337	5535
148	343	5478	342	5609
150	347	5605	349	5248
152	350	5574	349	4918
154	355	5201	349	5141
156	357	5080	350	5339
158	2	4727	352	4714
160	3	4965	5	4716

Appendix F

Rudder Calibration Data

Servo Value	Rudder Angles				
	Set 1	Set 2	Set 3	Set 4	Set 5
0	312	312	312	312	312
25	312	312	312	312	312
50	313	314	313	313	313
75	327	329	327	329	329
100	347	347	347	346	346
125	4	3	4	4	4
150	18	18	19	17	18
175	32	33	34	32	32
200	42	42	39	40	40
225	42	42	39	40	40
250	42	42	39	40	40

Appendix G

Compass Calibration Data

Heading	Compass Output Value	Heading	Compass Output Value	Heading	Compass Output Value
0	109	45	126	90	154
0	112	45	128	90	153
0	108	45	129	90	154
0	113	45	129	90	158
0	110	45	127	90	143
0	100	45	127	90	152
0	104	45	128	90	145
0	102	45	129	90	144
0	107	45	126	90	156
0	105	45	128	90	156
0	98	45	126	90	152
0	109	45	128	90	146
0	111	45	129	90	145
0	104	45	124	90	157
0	107	45	121	90	147
0	108	45	128	90	157
0	108	45	127	90	145
0	111	45	127	90	146
0	109	45	123	90	142
0	109	45	127	90	139
0	109	45	126	90	147
0	111	45	120	90	149
0	108	45	116	90	150
0	108	45	127	90	157
0	111	45	127	90	157

Heading	Compass Output Value	Heading	Compass Output Value	Heading	Compass Output Value
135	178	180	210	225	20
135	175	180	204	225	22
135	178	180	208	225	20
135	174	180	210	225	21
135	179	180	208	225	19
135	168	180	208	225	20
135	180	180	210	225	21
135	180	180	210	225	18
135	179	180	204	225	20
135	176	180	209	225	20
135	178	180	207	225	19
135	180	180	205	225	19
135	178	180	207	225	18
135	173	180	208	225	18
135	178	180	208	225	19
135	180	180	211	225	19
135	175	180	211	225	20
135	180	180	207	225	19
135	177	180	207	225	19
135	179	180	211	225	20
135	180	180	206	225	20
135	163	180	210	225	19
135	176	180	209	225	19
135	180	180	208	225	19
135	178	180	212	225	19

Heading	Compass Output Value	Heading	Compass Output Value
270	48	315	1129
270	52	315	2142
270	50	315	3222
270	54	315	4261
270	52	315	5313
270	52	315	6434
270	54	315	7470
270	53	315	8514
270	53	315	9522
270	53	315	10566
270	53	315	11574
270	52	315	12595
270	51	315	13632
270	52	315	14660
270	52	315	15667
270	54	315	16718
270	53	315	17730
270	53	315	18737
270	52	315	19753
270	50	315	20803
270	52	315	21810
270	50	315	22865
270	52	315	23889
270	53	315	24896
270	51	315	25911

Appendix H

Sample logging output (Simulator Generated)

Milliseconds elapsed	heading	X coordinate	Y Coordinate	Heading Error
1129	339	24	16	-134
2142	334	38	22	-129
3222	316	51	30	-111
4261	297	59	40	-92
5313	278	62	49	-74
6434	259	63	54	-55
7470	240	59	66	-35
8514	232	55	74	-27
9522	228	48	83	-23
10566	224	42	90	-19
11574	220	36	96	-15
12595	218	31	101	-13
13632	214	26	106	-9
14660	212	16	113	-7
15667	210	7	119	-5
16718	208	0	124	-93
17730	194	-8	128	-79
18737	176	-13	129	-61
19753	158	-13	129	-43
20803	144	-15	128	-29
21810	138	-17	125	-23
22865	134	-21	122	-19
23889	132	-25	117	-17
24896	128	-29	112	-13
25911	126	-33	105	-11
26918	122	-37	98	-7

Appendix I

Sample logging output (Boat Generated)

As the GPS Receiver is not connected with the Basic Stamp or Psion it logs data independently. Although it is theoretically possible to merge the data from the GPS and the Psion no attempt was made to do this given the complexity of the task. As a result all position information is shown as zeros.

Milliseconds elapsed	heading	X coordinate	Y Coordinate	Heading Error
1129	339	0	0	-134
2142	334	0	0	-129
3222	316	0	0	-111
4261	297	0	0	-92
5313	278	0	0	-74
6434	259	0	0	-55
7470	240	0	0	-35
8514	232	0	0	-27
9522	228	0	0	-23
10566	224	0	0	-19
11574	220	0	0	-15
12595	218	0	0	-13
13632	214	0	0	-9
14660	212	0	0	-7
15667	210	0	0	-5
16718	208	0	0	-93
17730	194	0	0	-79
18737	176	0	0	-61
19753	158	0	0	-43
20803	144	0	0	-29

Appendix J

Sail setting lookup table for the PID controller.

Wind Angle	Sail Angle
0	0
45	0
68	22
90	45
135	68
179	90
180	270
225	208
270	325
293	342
315	359
360	359

Appendix K

Fuzzy Logic Set Definition Tables

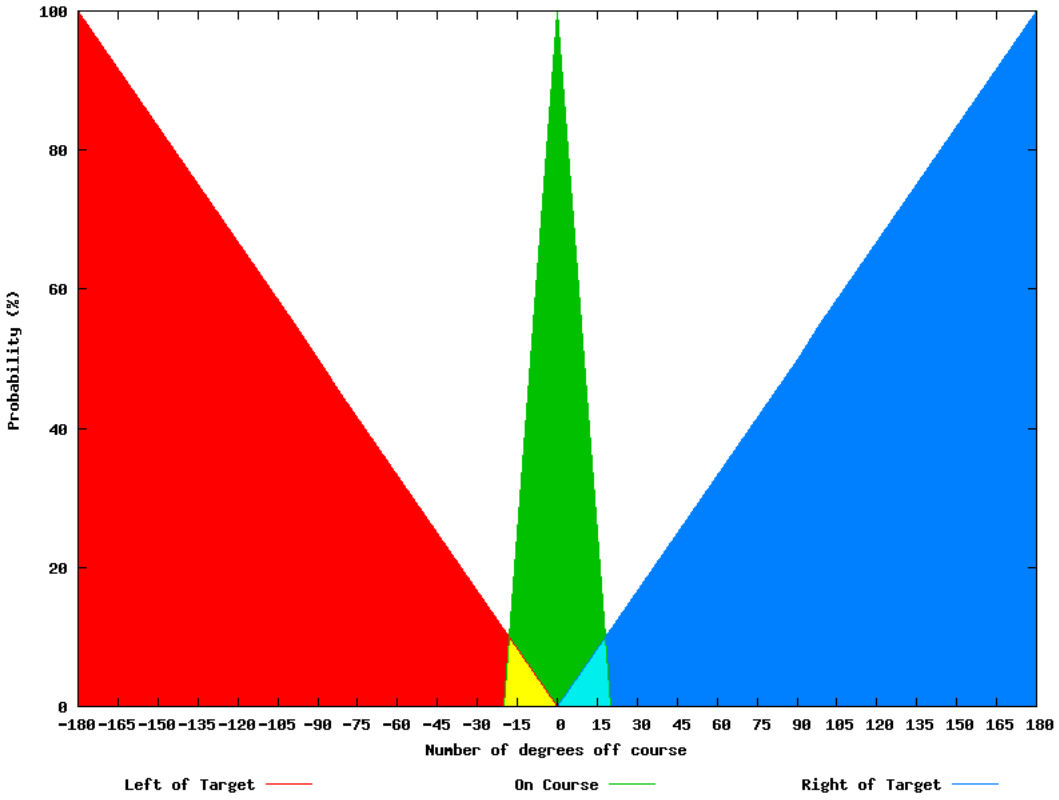

Figure K.1: Fuzzy logic set definitions for classifying heading error.

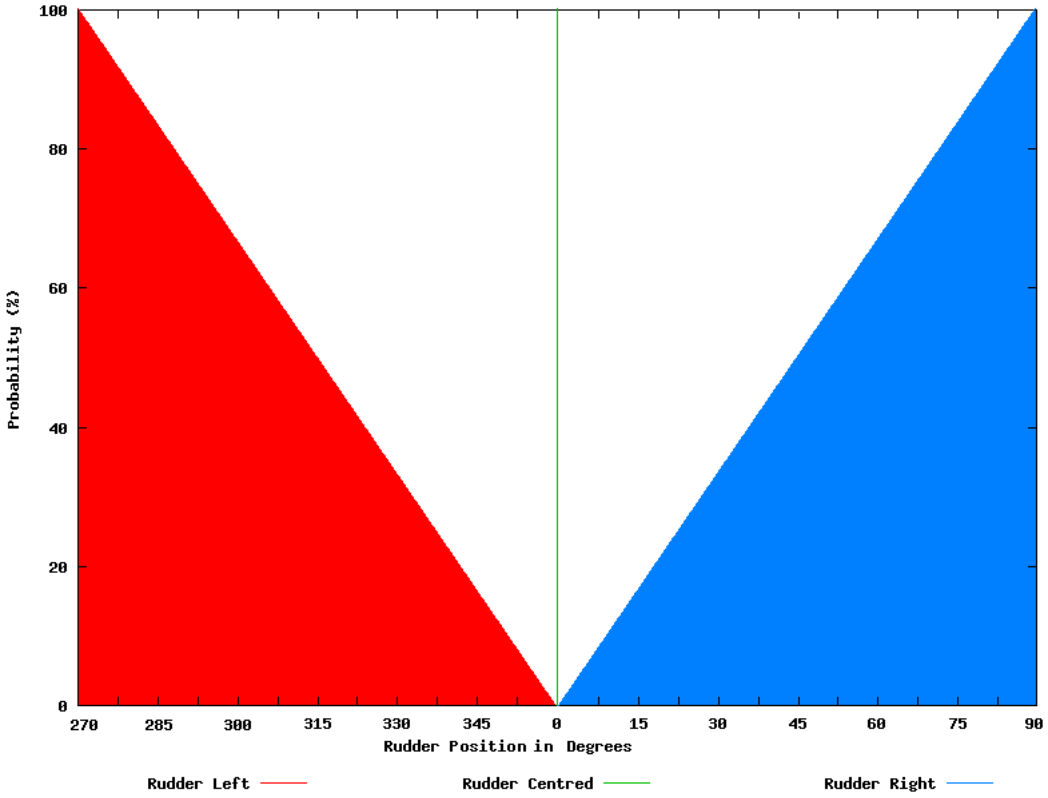

Figure K.2: Fuzzy logic set definitions for setting the rudder.

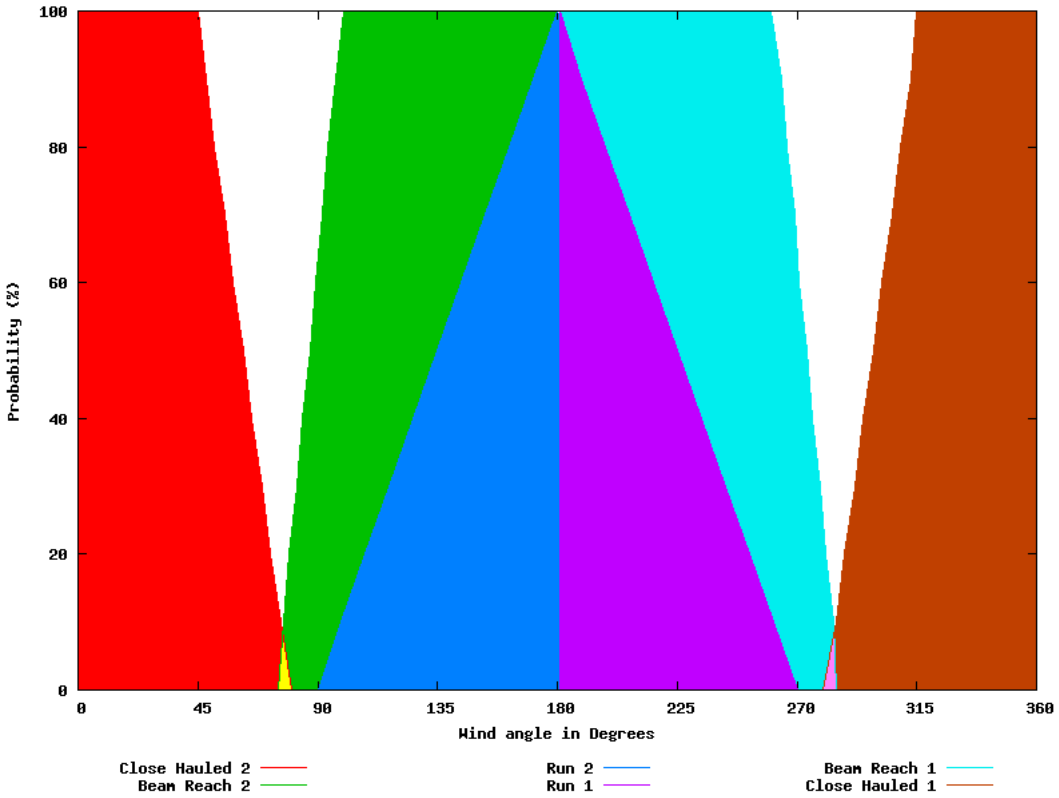

Figure K.3: Fuzzy logic set definitions for classifying wind direction.

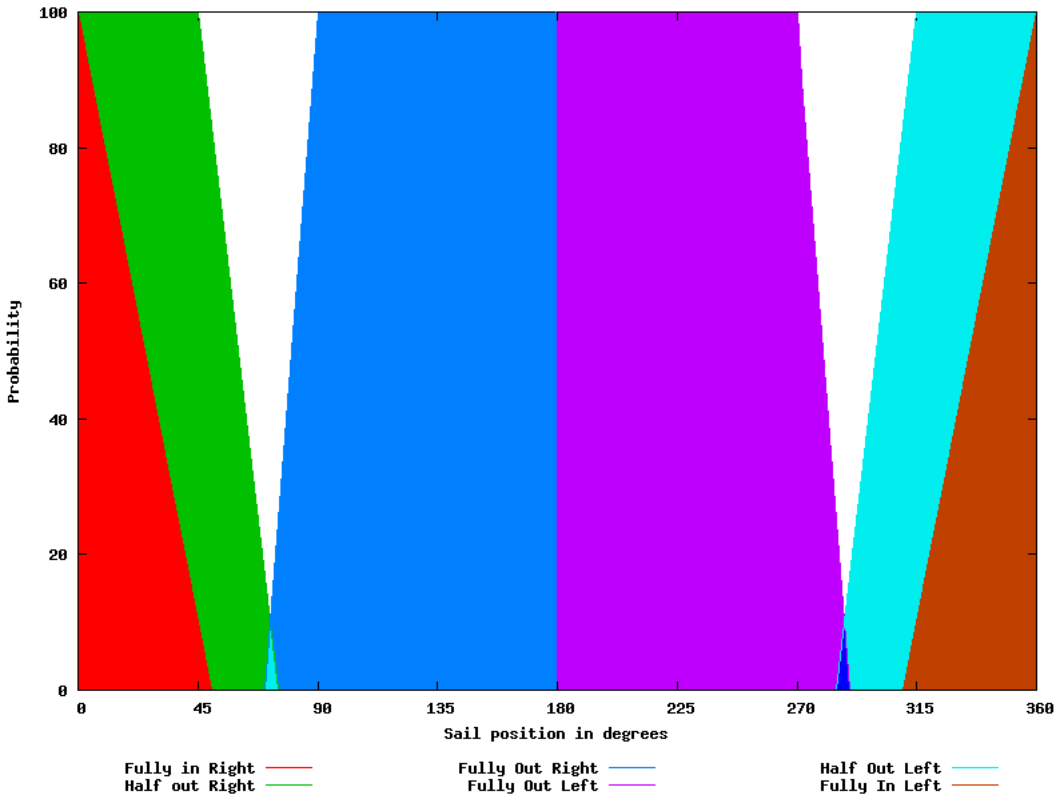

Figure K.4: Fuzzy logic set definitions for setting the sail position.

Appendix L

Init scripts for use on the Psion

```
#!/bin/sh
#attach /dev/ttyAM0 to /dev/ircomm0
irattach /dev/ttyAM0
```

Figure L.1: The /etc/init.d/S46irda script used to start the infra-red device upon boot on the Psion.

```
# /etc/inittab
# Copyright (C) 2001 Erik Andersen <andersen@codepoet.org>
# Note: BusyBox init doesn't support runlevels.  The runlevels field is
# completely ignored by BusyBox init.
# Format for each entry: <id>:<runlevels>:<action>:<process>
# id        == tty to run on, or empty for /dev/console
# runlevels == ignored
# action    == one of sysinit, respawn, askfirst, wait, and once
# process   == program to run
# Startup the system
null::sysinit:/bin/mount -o remount,rw /
null::sysinit:/bin/mount -t proc proc /proc
null::sysinit:/bin/mount -a
null::sysinit:/bin/hostname -F /etc/hostname
null::sysinit:/sbin/ifconfig lo 127.0.0.1 up
null::sysinit:/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo
# now run any rc scripts
::sysinit:/etc/init.d/rcS
# Set up a couple of getty'
::respawn:/bin/login
# Put a getty on the infra-red port
ircomm0::respawn:/sbin/getty -h -L ircomm0 9600 vt100
# Logging junk
null::sysinit:/bin/touch /var/log/messages
null::respawn:/sbin/syslogd -n -m 0
null::respawn:/sbin/klogd -n tty3
::respawn:/usr/bin/tail -f /var/log/messages
# Stuff to do for the 3-finger salute
::ctrlaltdel:/sbin/reboot
# Stuff to do before rebooting
null::shutdown:/usr/bin/killall klogd
null::shutdown:/usr/bin/killall syslogd
null::shutdown:/bin/umount -a -r
null::shutdown:/sbin/swapoff -a
```

Figure L.2: The modified /etc/initab script used to automatically start a login console on the infra-red port. This also causes the login program (called getty) to respawn everytime it exits.

Appendix M

Spreadsheet used to verify heading error outputs

Current Heading	Desired Heading	Actual Error	Current Heading	Desired Heading	Actual Error
0	1	1	45	0	-45
0	90	90	45	44	-1
0	180	180 or -180	45	46	1
0	270	-90	45	90	45
0	359	-1	45	180	135
90	89	-1	45	270	-135
90	91	1	135	0	-135
90	180	90	135	90	-45
90	270	180 or -180	135	134	-1
90	0	-90	135	136	1
180	0	180 or -180	135	180	45
180	90	-90	135	270	135
180	179	-1	225	0	-135
180	181	1	225	90	135
180	270	90	225	180	-45
270	0	90	225	224	-1
270	90	180 or -180	225	226	1
270	180	-90	225	270	45
270	269	-1	315	0	-45
270	271	1	315	90	135
			315	180	-135
			315	270	-45
			315	314	-1
			315	316	1

Table M.1: Example of the verification spread sheets used to check the heading errors determined by the code. The heading error code was tested against all combinations shown in this table in to show it was working correctly. If the output differed from what was expected then the test fails.

Appendix N

GPS Plots of the test runs

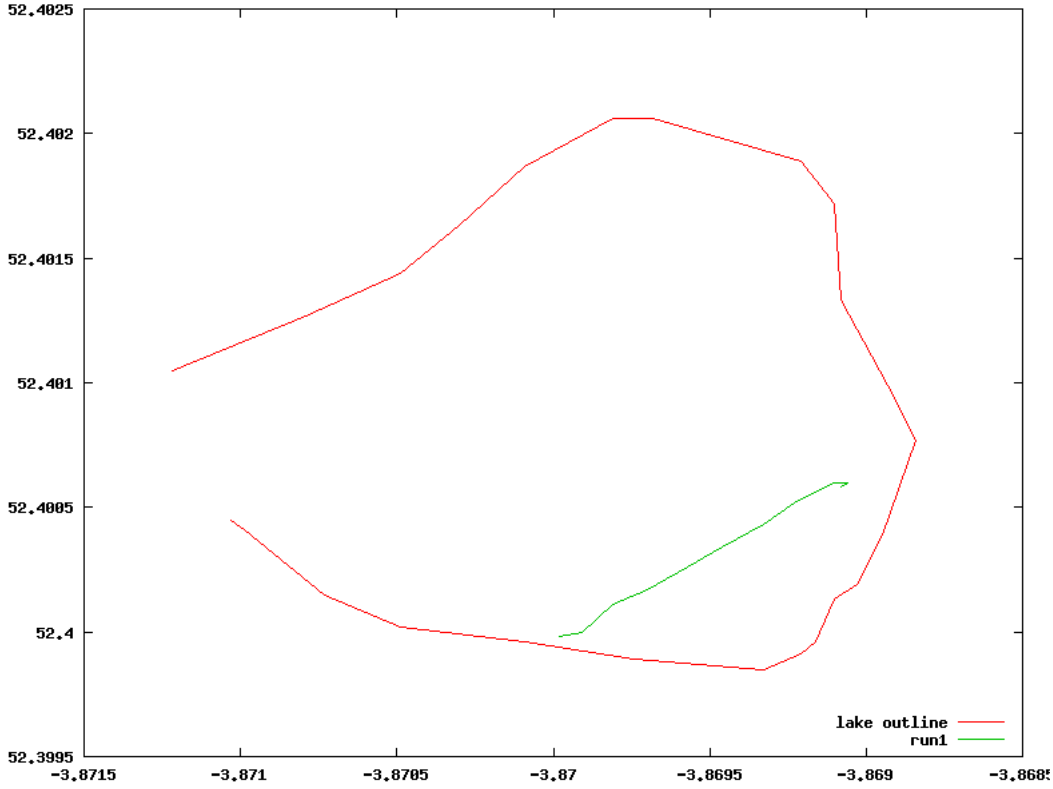

Figure N.1: The GPS plot for the first journey.

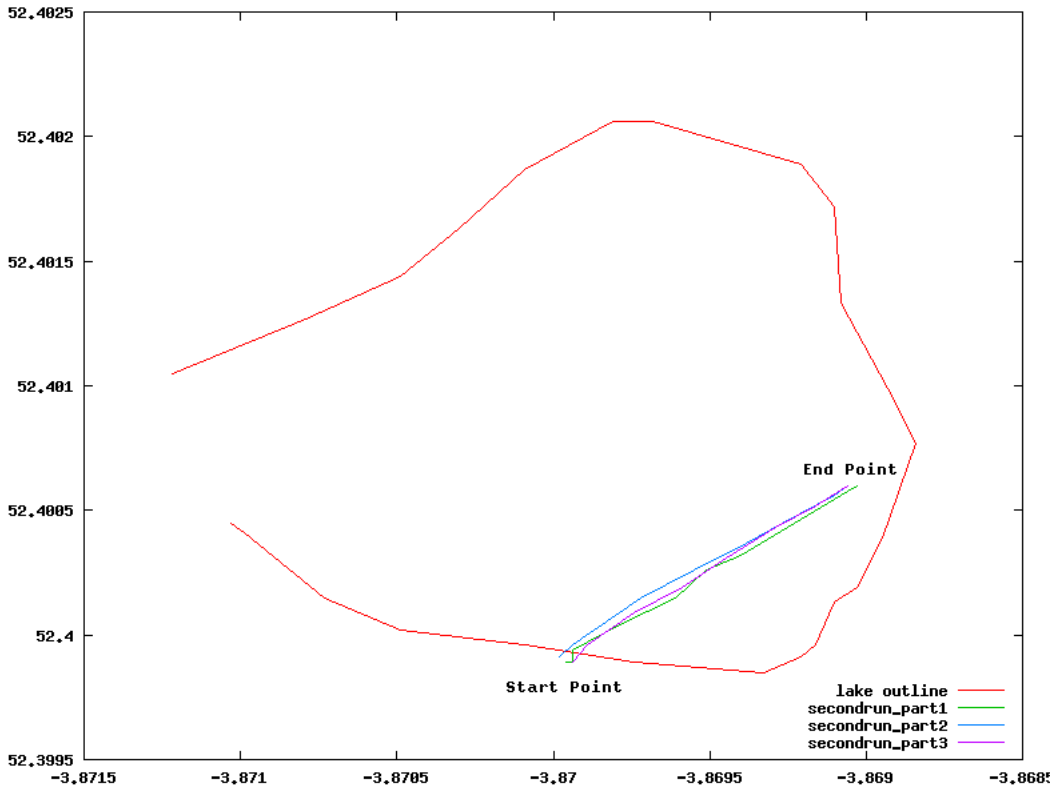

Figure N.2: The GPS plot showing the three journeys made during the second test run. For all three of these the proportional constant was set to a value of two. The trips to return the boat to its start point have been removed.

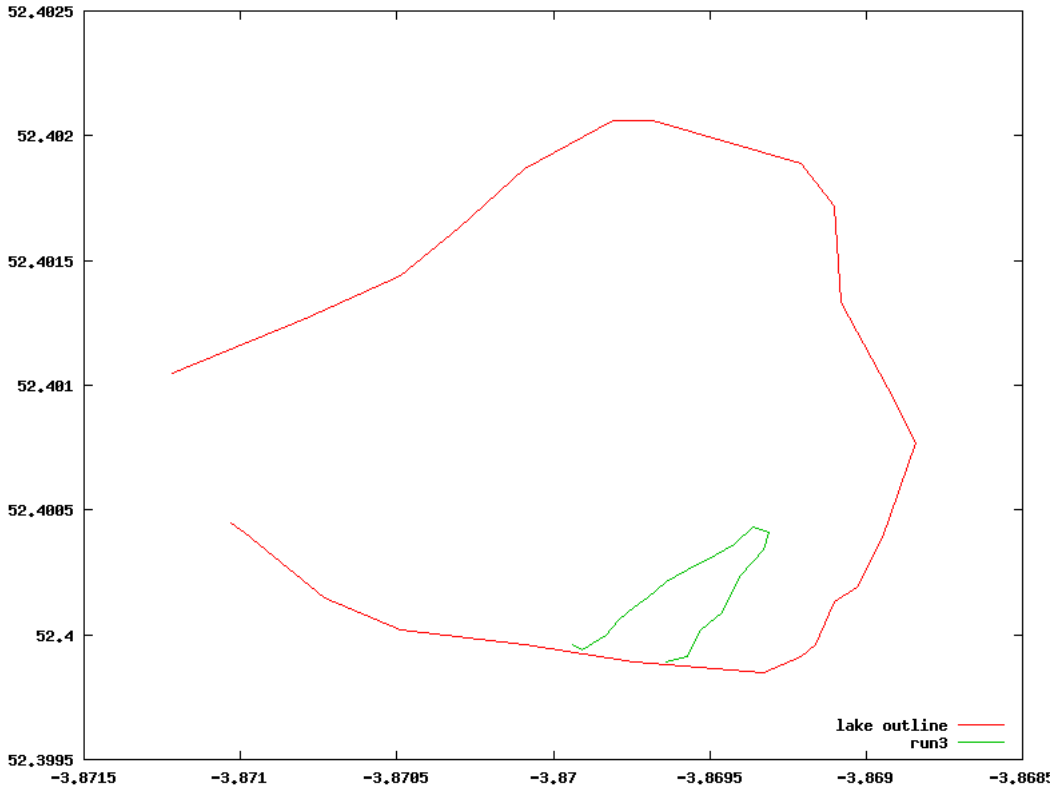

Figure N.3: GPS plot showing both the outbound and return journeys made during the third test run. For this test the proportional constant was set to one.

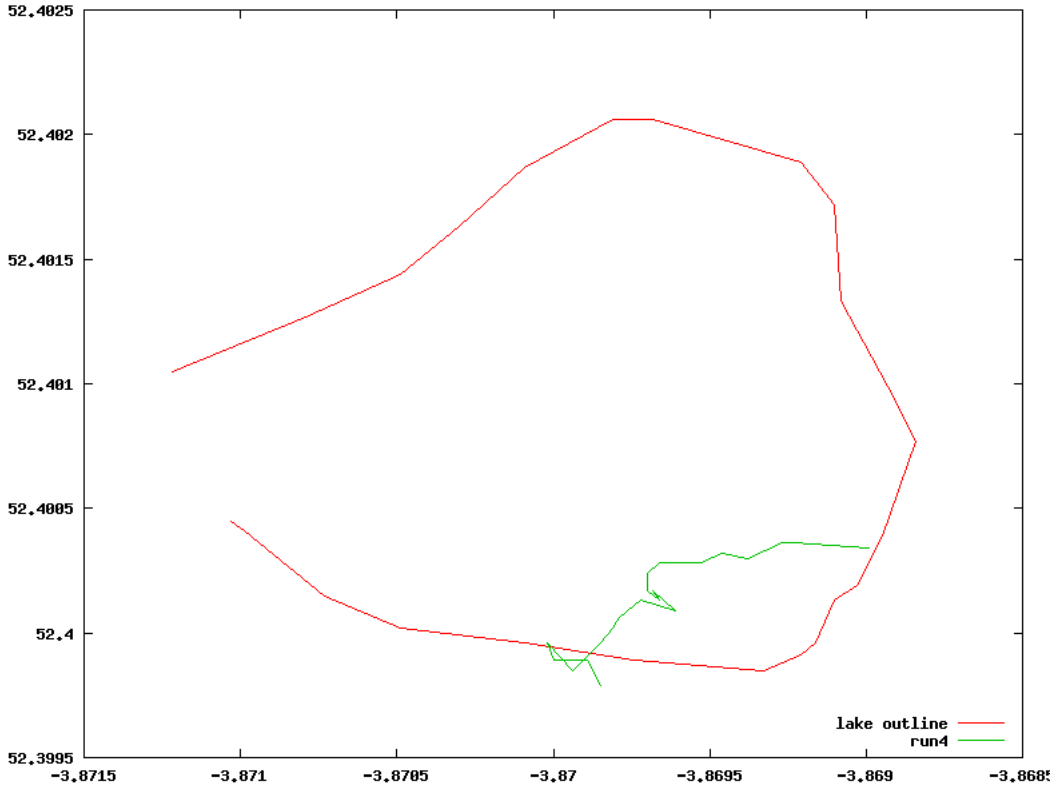

Figure N.4: GPS Plot of the fourth test. It was intended that the boat would sail to the same point at which it turned in the previous test, then turn around sail back towards the shore and repeat this three times. Unfortunately the wind dropped soon after the boat reached the turning point.

Appendix O

Heading/Time Plots of the test runs

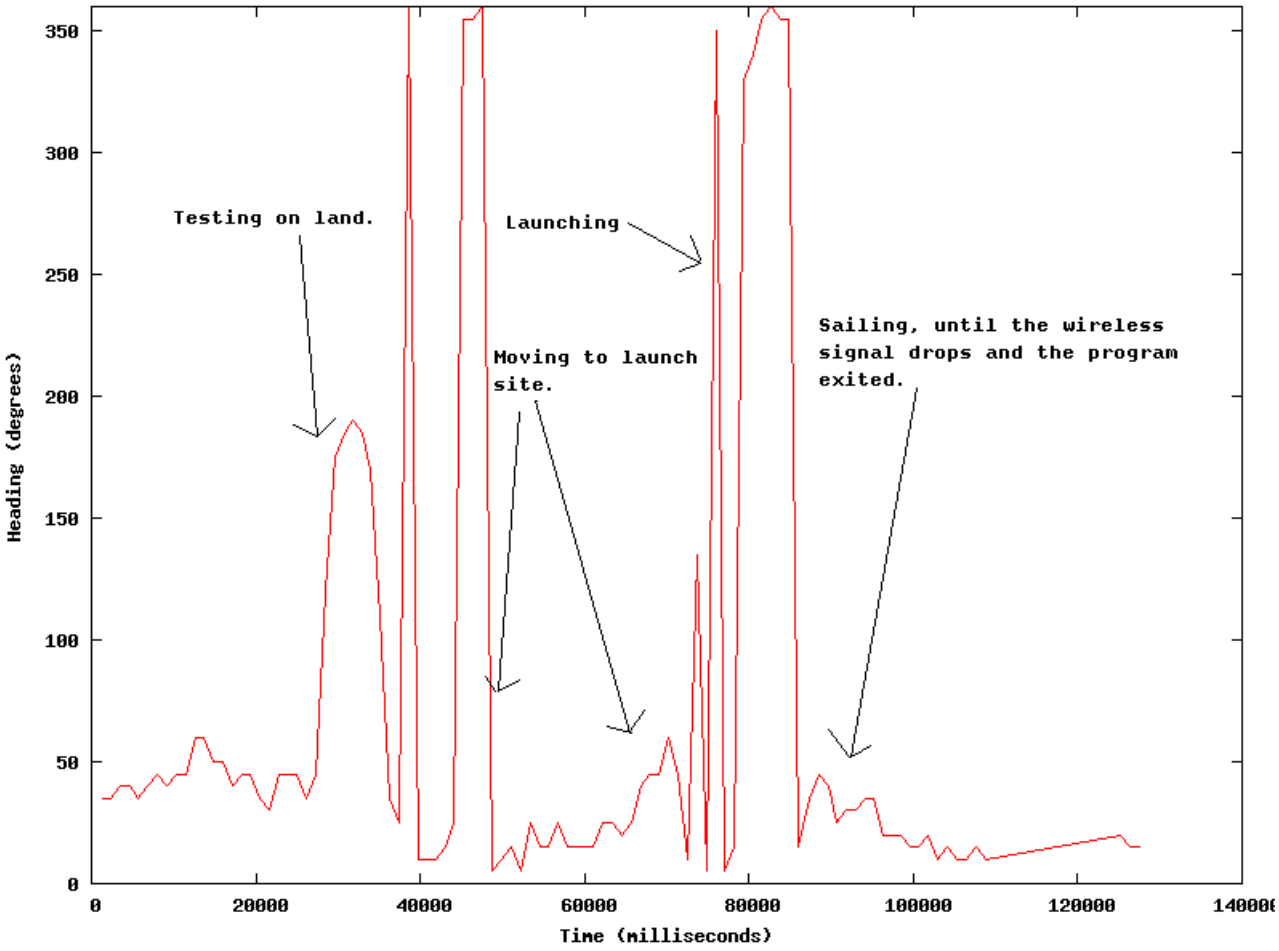

Figure O.1: Heading versus time plot for the first test run. As the code stopped logging early in this test it does not cover the full journey shown in the GPS plot.

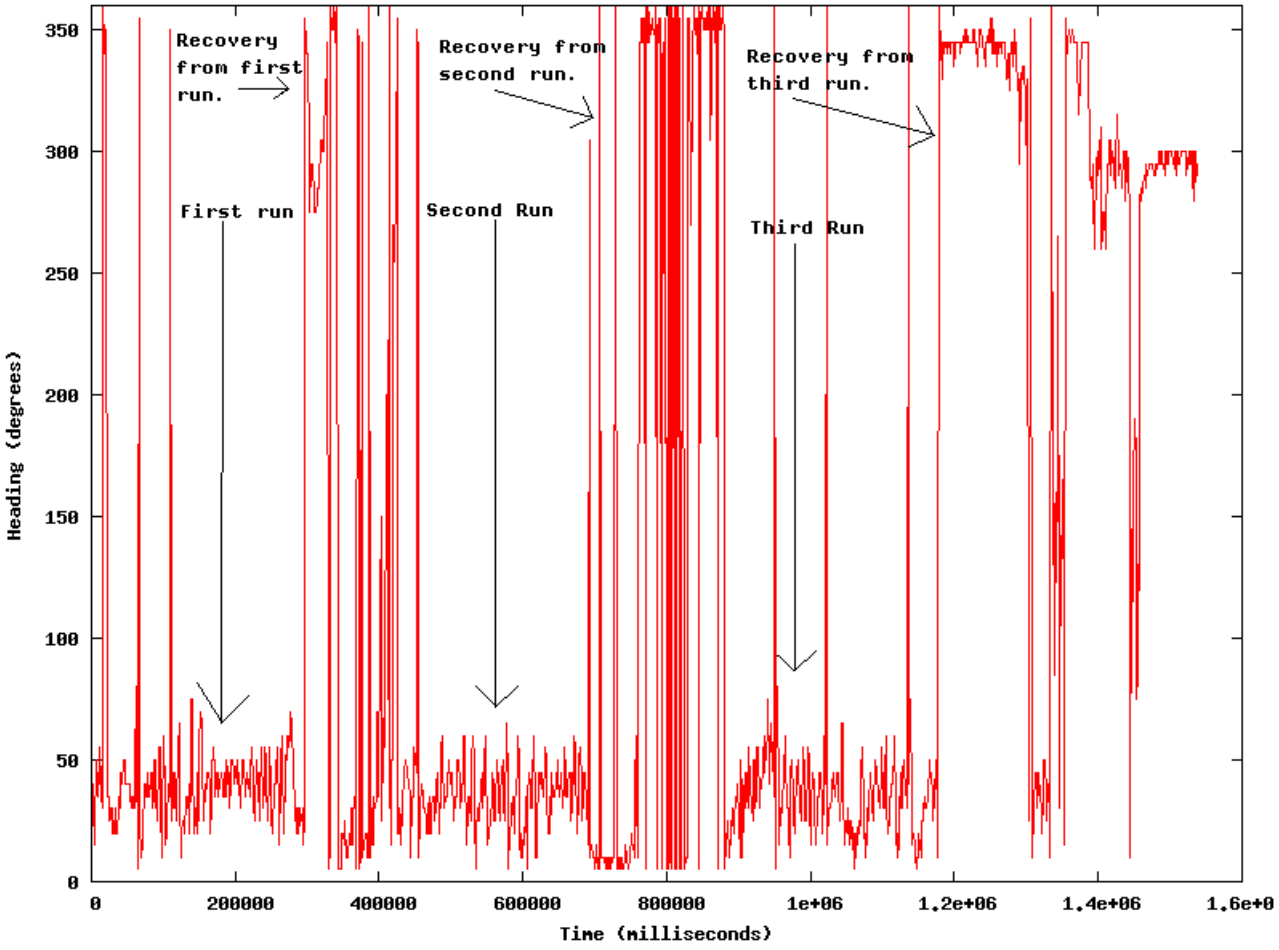

Figure O.2: Heading versus time plot for the second run. All three runs are shown as is the time spent carrying the boat back to the original start position.

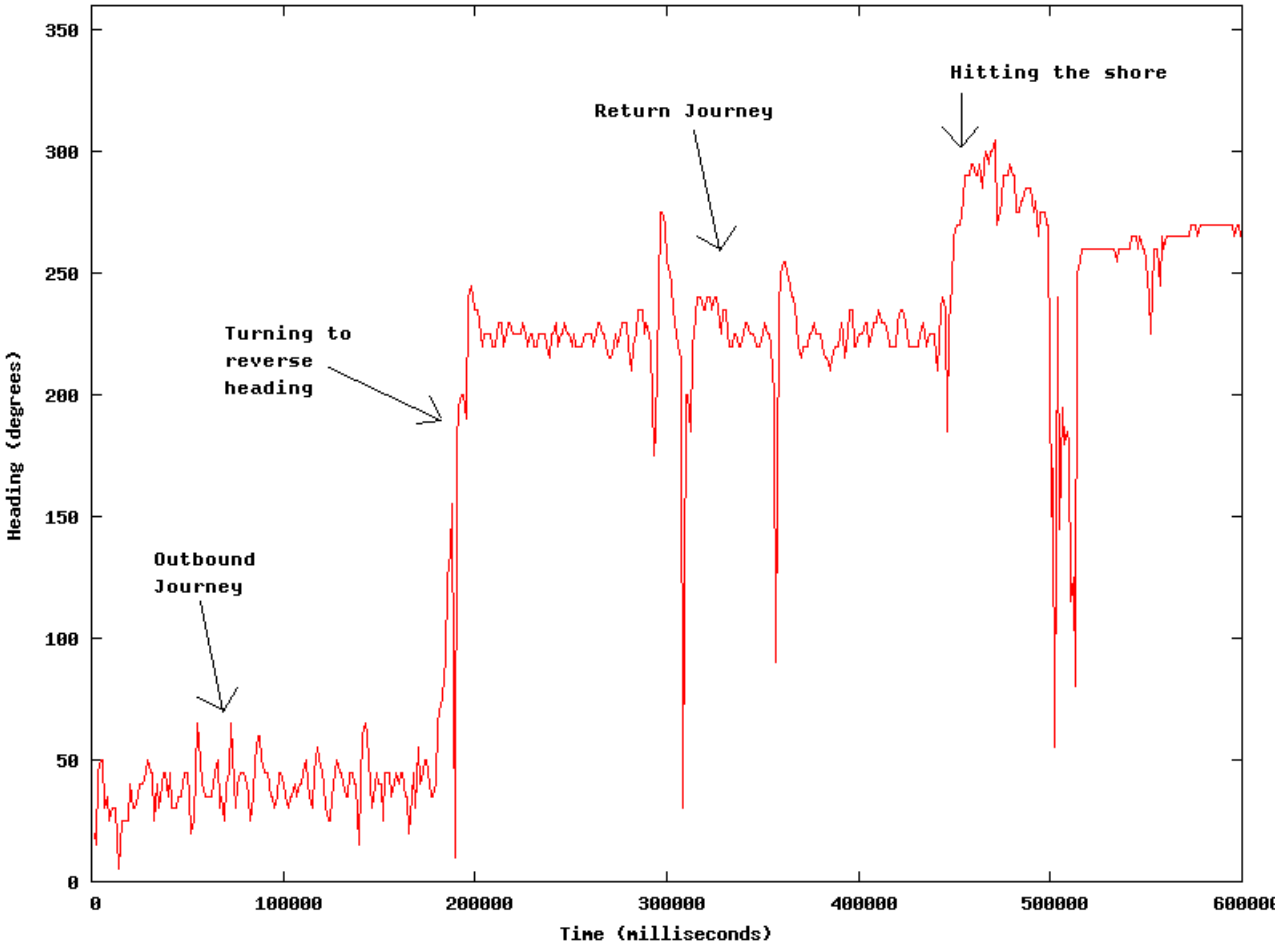

Figure O.3: Heading versus time plot for the third run. Both the outward and return journeys are shown. There is a small gap during which time no logging took place as the sail was moving.

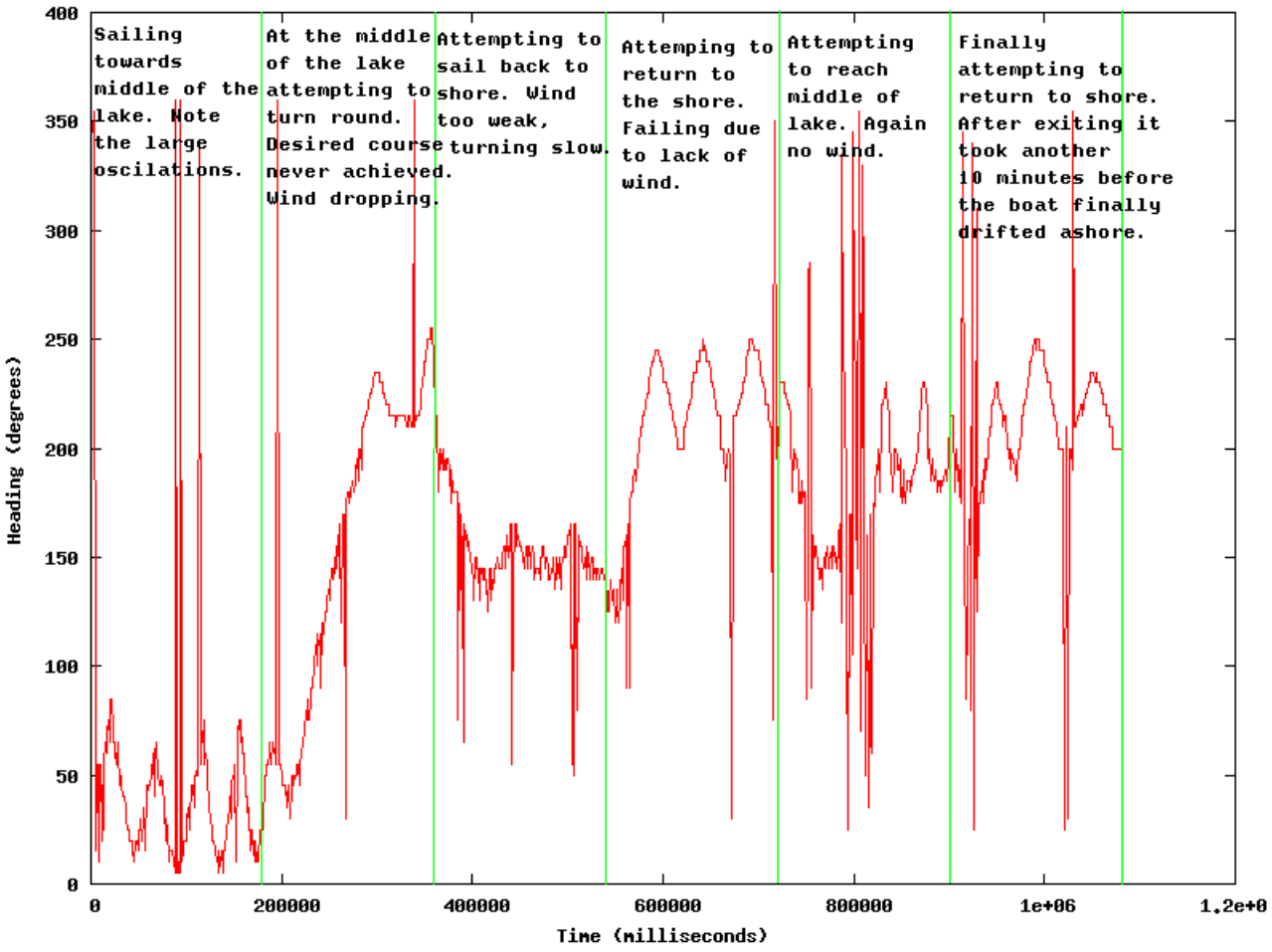

Figure O.4: Heading versus time plot for the fourth run. The green lines show the division between each three minute run of the fuzzy logic controller.

Appendix P

Simulator Testing Plots

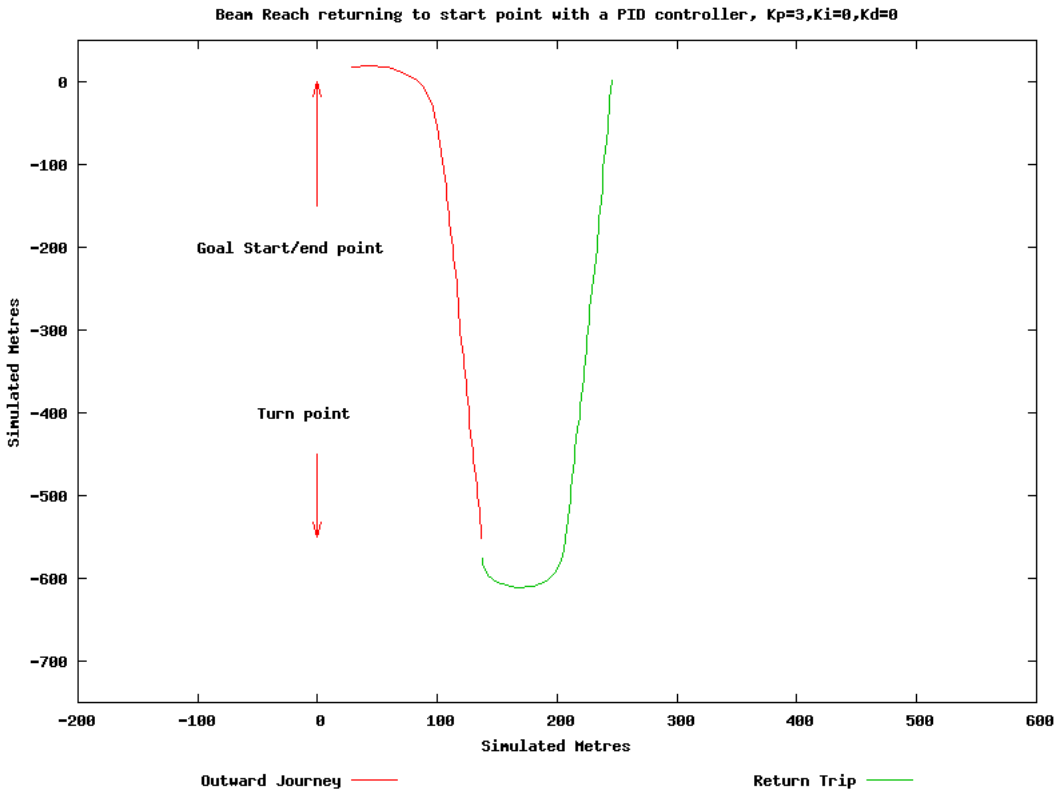

Figure P.1: Plot showing the course taken during the simulated PID controller beam reach test.

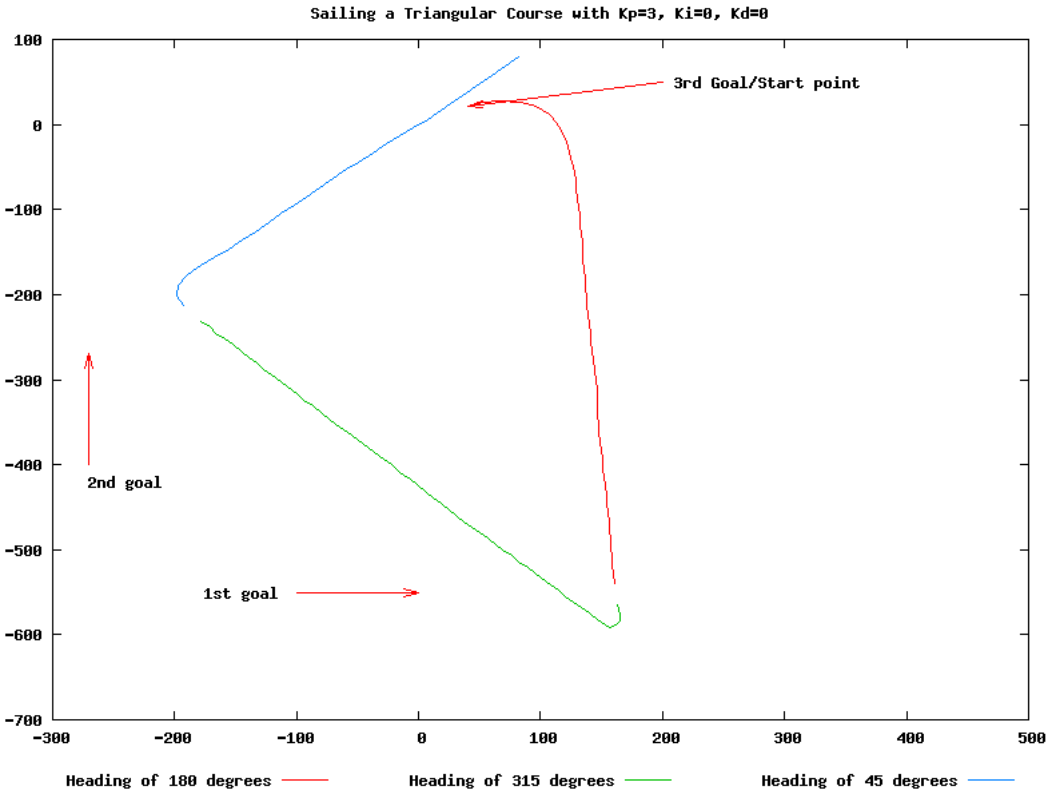

Figure P.2: Plot showing the course taken during the simulated PID controller triangle test.

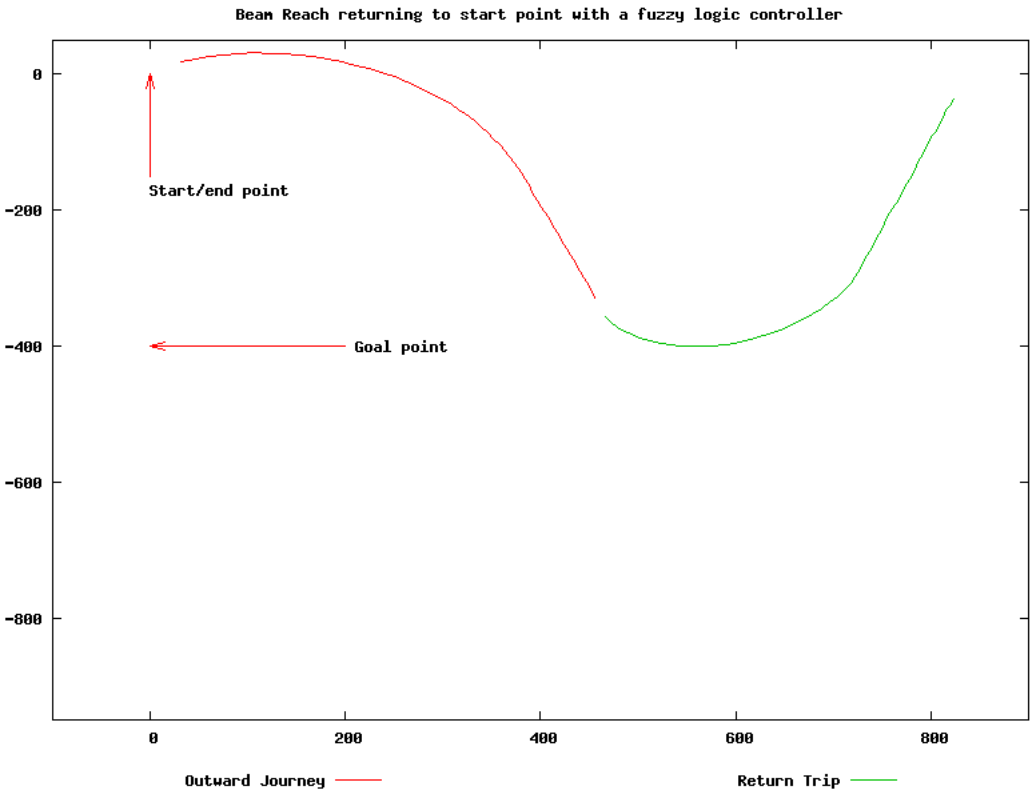

Figure P.3: Plot showing the course taken during the simulated Fuzzy logic controller beam reach test.

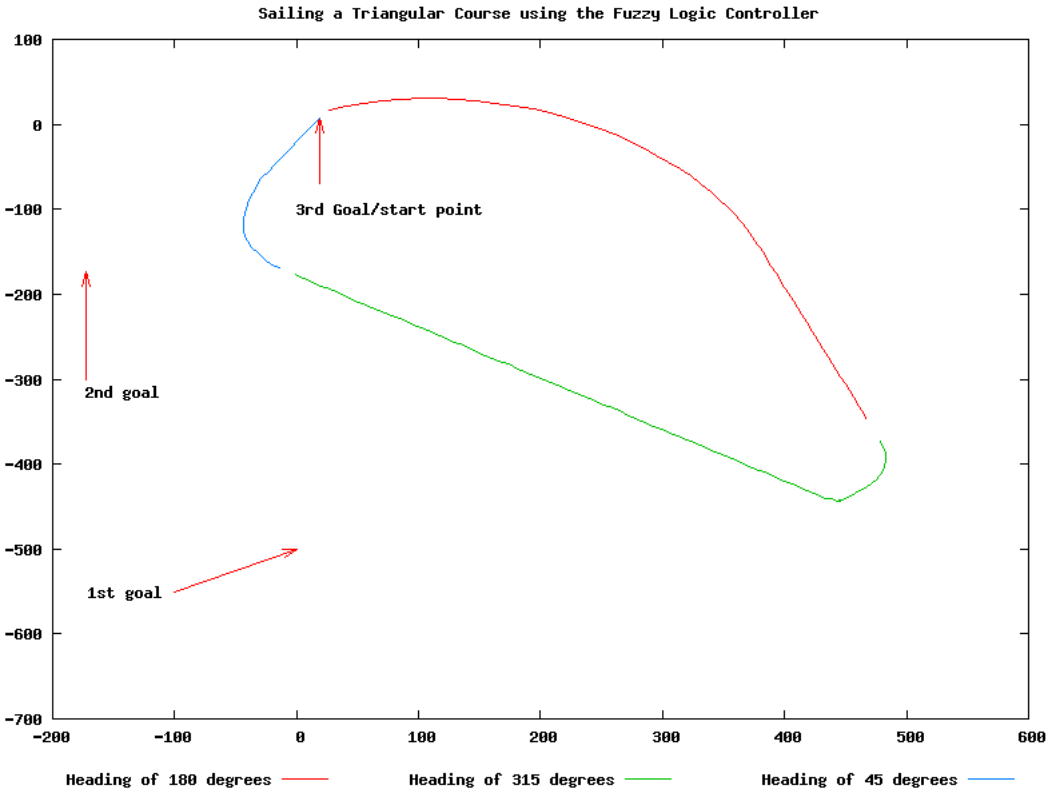

Figure P.4: Plot showing the course taken during the simulated fuzzy logic controller triangle test.

Appendix Q

Original Project Timetable

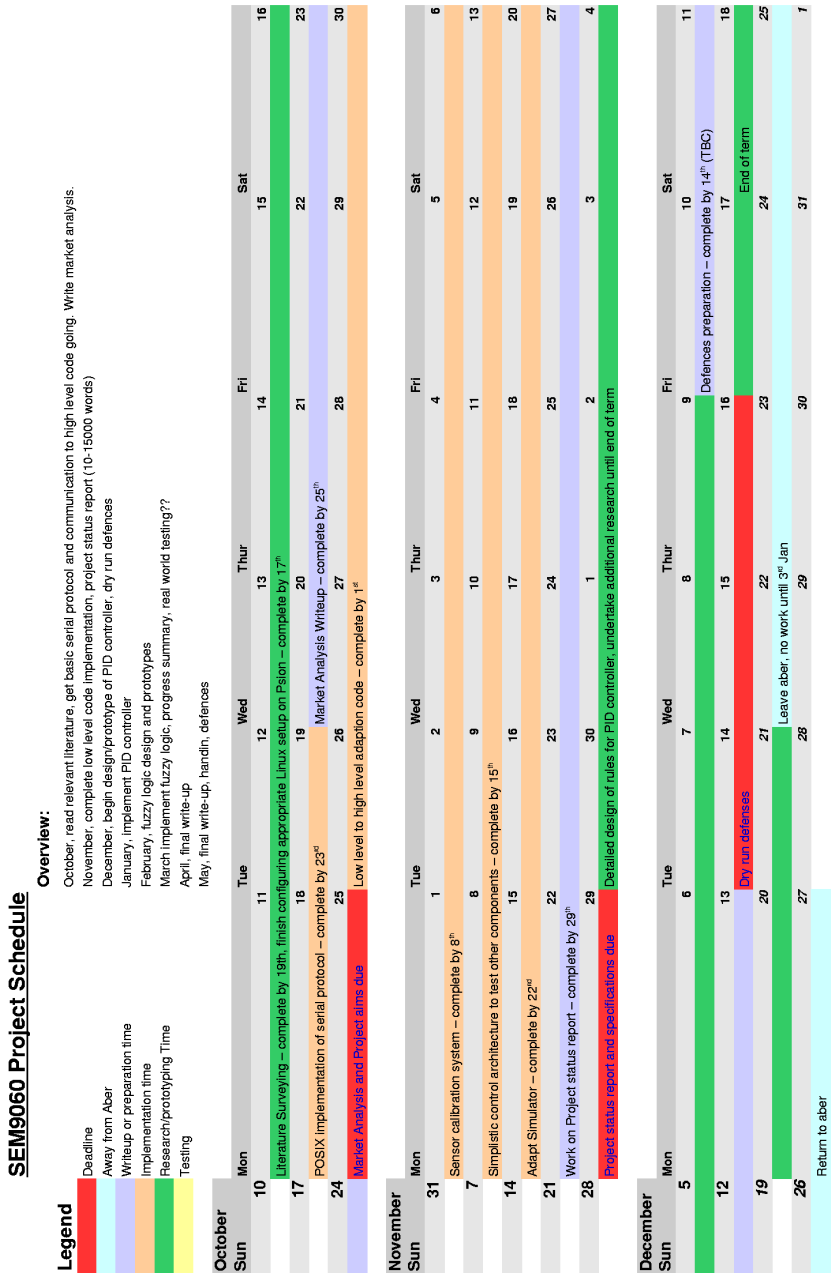

Figure Q.1: Original timetable - October to December.

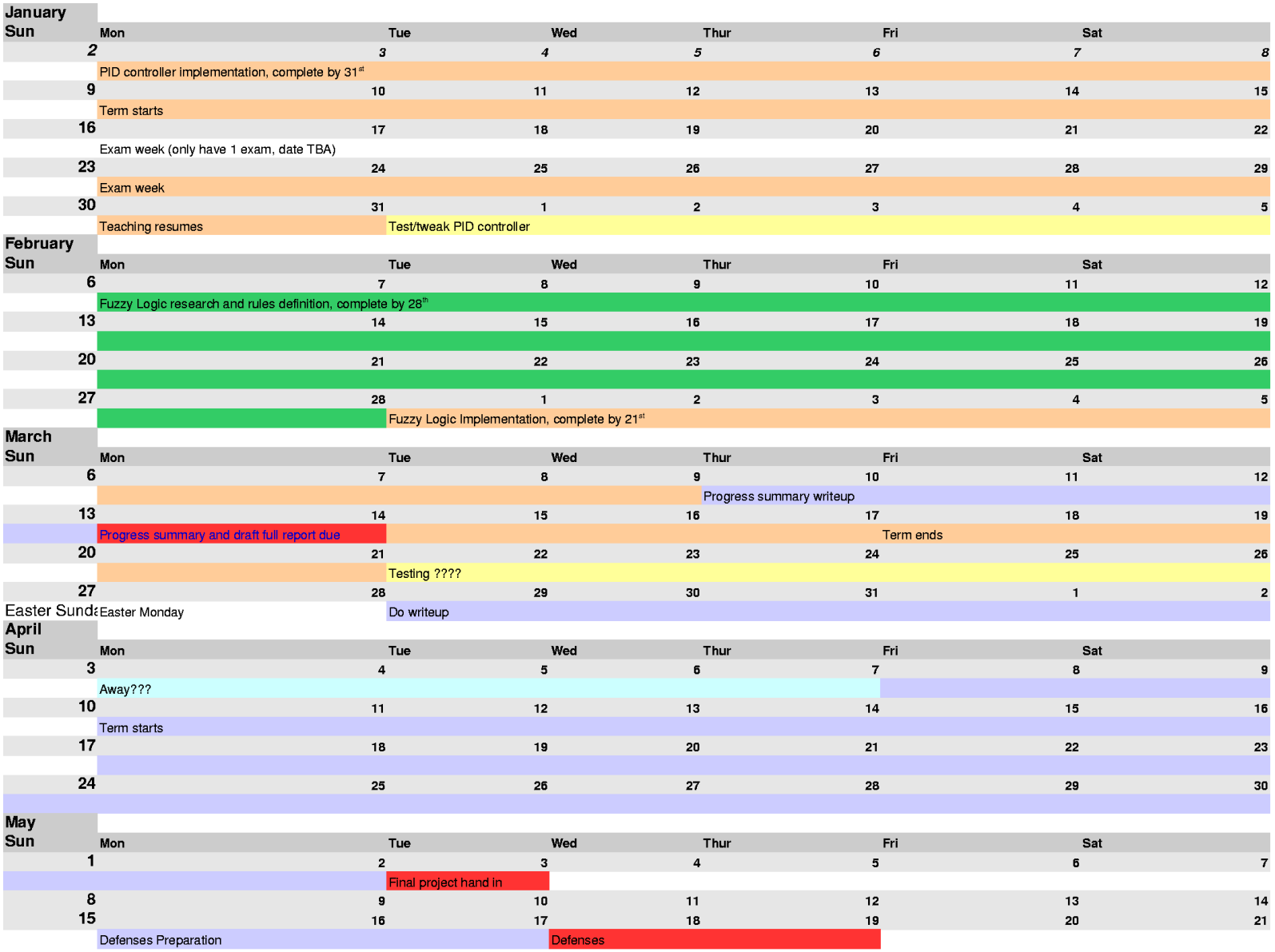

Figure Q.2: Original timetable, January to May.