Area Scanning Control System for Backpackable Robot Boats

Report for CS39440 Major Project

Author: Elizabeth Stone (eas12)
Supervisor: Dr. Mark Neal (mjn)

Version: 1.9 - Draft
Friday 5th May, 2017

This report was submitted as partial fulfilment
of a BSc degree in Space Science and Robotics (FH56)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name .....Elizabeth.Stone.....

Date .....30/04/2017.....

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name .....Elizabeth.Stone.....

Date .....30/04/2017.....

Acknowledgements

I thank my dissertation supervisor Dr. Mark Neal for all his help with this project, and in previous work that led me to be able to complete this project.

I also thank the department for providing space for me to work and the resources I needed.

I also thank those in the department who helped me, most notably Dr. Pete Todd who was always there to lend a hand. PhD students Tom Blanchard and Sam Nicholls, and fellow BSc students Sam Claxton and Martin Handy, I thank for their time helping me move equipment to and from tests; I would never have been able to complete the tests without this help.

To AberSailbot I owe my love of boats, and my experience of working with robotic boats. I thank all involved during my time with AberSailbot, and for giving me experience in such a unique field.

And finally, to my friends and family who have supported me throughout university and made this project bearable.

Abstract

this project...

Contents

Chapter 1

Background & Objectives

1.1 Background

1.1.1 Motivation

Academics at Aberystwyth University Geography department studied Scottish lochs two and a half years ago using a lightweight remote control boat. The study required the boat to scan the lochs using sonar sensors to build up an image of the bottom of the lochs and to locate logs on the loch floor (in order to investigate climate change using dendrochronology)[1]. In that study, the boat was controlled by RC. This project aims to make the boat fully autonomous in order to increase efficiency. Other projects since have used this boat, again controlling the boat under RC.

With these projects it was found that it was hard to properly control the boat as it is difficult to tell from shore exactly where the boat is; large flat lakes with a small boat at many tens of meters away make it very hard to see the movement of the boat. This makes it impossible to tell if the boat is travelling in a straight line. This is not good, as large areas of lake can be missed and so data from these areas is not collected, leading to incomplete data sets for analysis. Often it is not realised during the experiments that large areas have been missed until the results are processed (which may not be instantly if scanning lakes in remote locations), and so a better way of scanning water is needed.One solution is automating the boat so that it can be known for sure that every part of the lake has been scanned; a boat with a GPS will know exactly where it is and if it is going off course.

A platform like this could be useful in other areas of research, for example, for marine monitoring. The boat itself is a catamaran and is therefore extremely stable. Additional sensors could easily added to the existing hardware, or equipment (such as sondes [2]) could be easily carried to be released in remote locations.

Another use for vessel of this type would be for sending into areas too dangerous for a manned vessel to travel into (e.g. close to glaciers) or into areas where manned vessels are unable to travel to (e.g. lochs on top of remote mountains).

1.1.2 Similar Projects

Other boats have been automated in similar ways. There are teams across the world that automate robotic sailing boats and compete in yearly competitions such as the World Robotic Sailing Championships (accompanied by the Internal Robotic Sailing Conference)[3], which has run for nearly a decade, and the International Robotic Sailing Regatta [4]. Sailing boats are much more complex to control than motor powered boats (strongly relying on the angle of the wind to the boat and sails), but the basic concepts are similar; creating a system to navigate across the water demonstrating specific behaviours.

Different strategies for area scanning have been demonstrated at robotic sailing competitions, such as the World Robotic Sailing Championships and the International Robotic Sailing Regatta, both of which have an

area scanning challenge where there are generally mixed results [5]. Many different control and navigation systems were discussed at these conferences [6].

A project by AberSailbot [7] has similar aims to this project, so their system was looked into. There is the use of several different systems (e.g. arduino code, boatd [8], behaviours) to make their boat function. This in ways is very useful as parts are easily swappable , but also inconvenient when there are too many layers of system to understand. The overall structure is good, and a particularly nice feature is the use of a Raspberry Pi to control high level logic and an Arduino to control all input and output; this distributed design means there is not too much processing being done on either board.

[9]

Similar to AberSailbots system, a general control system non-specific to sonar would be useful. A system that can be easily used across similar platforms.

1.2 Analysis

1.2.1 Project Description

The AqASS (Aquatic Area Scanning System) project will produce a control system for a small, lightweight, motor-powered robotic boat to enable the boat to autonomously scan any given body of water, with selectable parameters.

The most basic aim for this project is to develop code to make the boat travel in a straight line towards a selected location on a body of water. This can then be built upon to make the boat travel to a series of way-points.

Building on this, the project will then develop an algorithm for automatically selecting the most efficient course for boat to take to scan a selected area (as defined by a series of coordinates).

It will then investigate telemetry and the design of a user interface that allows the user to communicate with the boat while it is on the water, and easily select the route the user wishes the robot to take.

It may also be useful to investigate semi-autonomous functions that can be used on the boat before and after starting the area scanning, such as heading holding, station keeping or return home functions, to aid in deployment of the system.

Suitable development methodologies will need to be investigated early on in this project.

1.2.2 Proposed Tasks

The following tasks are proposed:

- **Investigate Development Methodologies**
- **RC Lake Tests and environment research.** Investigate the environment encountered by the boats and the factors that affect their motion on the water (e.g. current, wind, waves), and what can be done to detect and account for them.
- **Investigate OS and programming languages.** Research which languages are most suitable for the hardware given, and which OS would be best to use on the Pi (if any).
- **Control System Development**
  - **Investigate types of navigation systems.** Research types of system and see which type is most suitable for this project (vector field systems, bearing systems etc).

  - **Investigate Control System Architectures** See which design would be best for this type of project. It may be that a modular approach will be necessary, which would allow easy swapping of hardware.
  - **Write code to get boat to travel in a straight line to a way-point/series of way-points.** This will involve accounting for factors discovered during rc testing.
  - **Investigate ways of representing and specifying body of water to be scanned and develop an algorithm to determine the most efficient route to scan the given area.** Specifying the body may be as simple as manually entering the data points, or as advanced as having a GUI where the user may draw the outline of the water on a map.The algorithm will likely depend on the shape of the body of water, current direction and other factors. It should also consider parameters such as desired resolution of scan (which would depend upon the sonar sample rate, speed of boat, and spacing of consecutive traversals of the area).
  - **Research and Develop Telemetry System** Develop easy to use interface to pass information to and from boat. Investigate if long distance communications would be useful and viable.
  - **Investigate plausibility of (and implement) collision avoidance system.** A system to prevent the boat from travelling into shallow waters, or crashing into objects in its path (e.g. other boats, buoys). This may involve processing the data from the sonar scanner(s), thus research will need to be done into how to process this data. It may also be necessary to develop a modular system so that sonar device may be swapped easily for another similar device.

- **Test control system** The methodology used will define the frequency of testing, but tests on one or more large bodies of water would be ideal nearer the end of the project. On land tests will be carried out throughout the rest of the project.
- **Project Meetings and Project Diary** Weekly project meetings will be held with the supervisor, and a project diary will be kept to keep track of all parts of the project. The diary will be written in markdown and backups will be kept in a git repository.
- **Demonstrations**

1.2.3 Deliverables

Deliverables expected from this project:

- Mid-project demonstration
- Control system software
- Usable user interface and telemetry software
- Collision avoidance software
- Final report
- Final demonstration

1.3 Process

This project used Feature Driven Development (FDD) methodology for orginization. Upfront a desgin for the hardware and a software flow diagram was created, as can be seen in section A.4. The main features were identified (Pi code, Arduino Code and Comms), and a list of tasks to be completed was then kept to work through these features. Sub-features were identified within the main features, as reflected in the sections of this report. The list of tasks were ordered by dependency (the first tasks on the list needed to be completed before the next tasks). The task list was kept on GitHub, removing the tasks from the to-do

list after completion, and adding new ones as new tasks were identified (mostly when the next feature was being designed).

To adapt FDD to a project with a single person, it was decided that the different roles normally used in FDD were not needed.

Chapter 2

Design

2.1 Overall Design

The hardware available for this project include rudder servos, servo motors, compass, gps and sonar scanner. The sonar scanner is not used in the final design. The final plan for the hardware is shown in Figure 2.1, with the GPS on the Pi and all other hardware on the arduino.

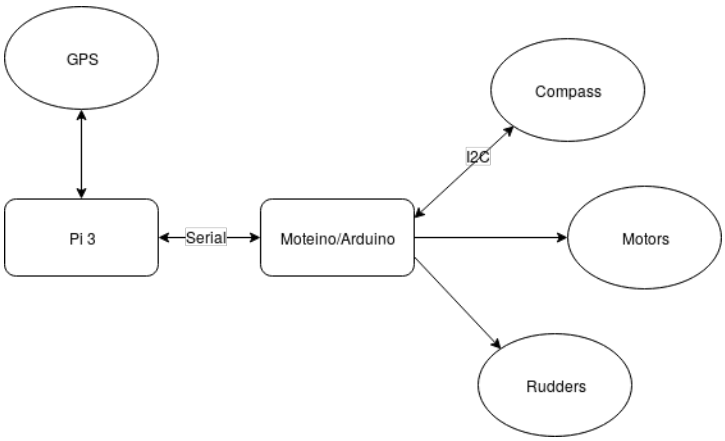


Figure 2.1: Final hardware design.

The control system has two separate parts; one for high level computations for global navigation (deciding where to go overall) and one for low level control for local navigation (keeping the boat travelling along a given heading using pid). If the two systems were not separate, the high level logic may delay the lower level logic and prevent the motors and rudders being updated in a short enough time for pid to work properly (this can result in oscillation of the boat about the heading it wants to go on, rather than going straight on the heading, which ultimately could lead to the boat going far off-course).

The lower level controller controls all input and output necessary to keep the boat steering on a desired heading (*e.g.* compass, motors and rudders).

The higher level controller communicates a heading to the lower level controller, and this heading is decided using the current GPS location, and given GPS co-ordinates to build a map of where to go.

The design is also be easily adaptable to different hardware, so that hardware can be easily updated or swapped easily. For this reason, for each piece of hardware there is a driver, separate from the main code, that the main code can use to talk to the hardware. For each new, or different, piece of hardware a new driver can be implemented.

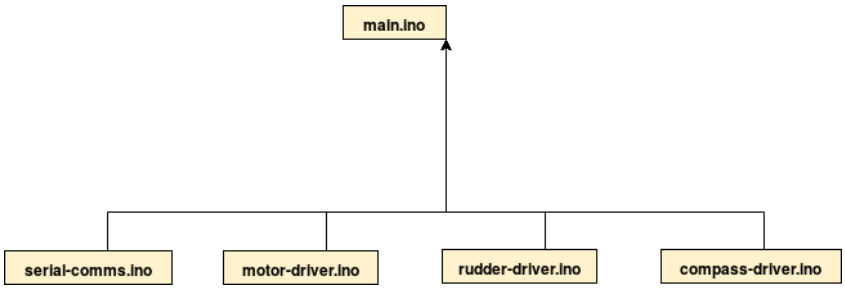


Figure 2.2: Design of the low level code.

The python code is similarly modular with interchangeable sections:

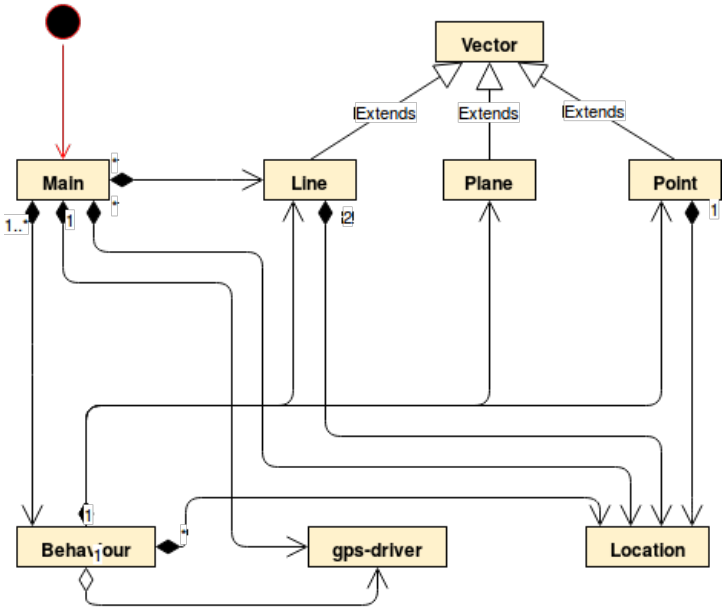


Figure 2.3: Design of the low level code.

As this project is open-source under the GPL3 ***license, having drivers for each separate piece of hardware is important as anyone who wishes to use this software will almost definitely have different hardware to what was used here.

The initial design can be found in section A.4.

*** insert flow diagram below

See the final design.

*** insert flow diagram below

The input and output from and to the hardware is handled using drivers to allow easy swapping of hardware with this code. On the Moteino there are three pieces of hardware attached; the rudder servos, motor servos and compass. There is also a serial connection to the Raspberry Pi, but the driver for this is more complex, as will be explained in subsection 2.2.3.

Use of the Robotic Operating System (ROS)*** was considered to organise the system and handle messaging, but having looked into using ROS it was found to be hard to work with, with many layers of dependencies

and very much over-engineering a solution for this project.

2.2 Detailed Design

2.2.1 Arduino Code

Arduino C was used as the Arduino is a cheap and easily obtainable micro-controller ***, and Arduino C can be used on similar micro-controllers such as the moteino,*** so is useful for anyone wanting to use this software on their own boats. In this implementation, a Moteino was used (though some code was also tested on an Arduino Uno and worked the same).

2.2.1.1 PID Controller

The local navigation system needs to have a pid control loop to control the rudders to adjust the boats' heading.

The motors could also be PID controlled, as this would mean that the boat will go at a steady speed despite its environment (e.g. strong wave, wind) which, if used for sonar scanning of a lake (or sensor readings of any type), readings would be at approximately the same distance apart (as it is most likely that the readings will have to be taken at every given time interval***). This would be useful, but complex to implement. The feedback to say how fast the boat is going is only measurable by the GPS, and this only gives the *** average speed between co-ordinates, which is only updated once a second. There are two limits with this: firstly, when not travelling in a perfectly straight line the measured speed will be less than the actual speed, which posses problems when the boat turns a tight corner and thus appears to have moved no distance. Secondly, the readings for speed being given only once a second is not fast enough for PID to properly work; PID works best with as small as possible time steps*** and a second is too large. Thus it was decided to not control the speed by PID.

With speed, there is an added complication that, when turning a tight corner, it is best to not travel at full speed as this increases the turning circle and means the boat must travel further to make it bake on course. It is better to travel slower while turning tight corners.

2.2.1.2 Hardware Drivers

It would have been desirable to have an interface that the each driver could implement, to make it easier for alternative drivers to be easily implemented, but Arduino C does not support this feature. It would also have been ideal if the separate .ino files in a folder were not automatically included in the main code as having this means that if there are multiple drivers with the same functions (e.g. a compass driver which controls hardware and a compass driver that simulates a compass) in the same file it will not be possible to tell when a function is called which driver it will use. There is no way of specifying this in the main code, so the only way to change this is by removing the drivers not currently wanted from the folder (placing them in a different folder).

Due to these limits on the language the drivers are not as useful as they could have been, but this design is still better than having the harware code in the main code which would mean the hardware could not be easily swapped.

To find how this specific hardware had been controlled before, code was obtained from the previous system used on this boat (see section B.2). It was noted that the rudder servos were controlled in the same way as any other (with 90 being the central point) using the arduino Servo library. For the motor servos it was noted that an arming value was needed to be set for a set amount of time during the setup for the motors to be able to work, and that a value greater than 90 will turn the motors in the direction to go forward.

***Both rudders are stuck together and cannot be turned in separate directions, as with the motors. Both

motors are controlled off the same pin on the moteino and so differential steering was not possible. This means both motors will always run at the same speed. This limits the types of movement available from the boat as it is not possible to turn on the spot. A consideration must therefore be given elsewhere in the code for large turning circles.

2.2.2 Pi Code

2.2.2.1 Navigation

For global navigation there are several different designs commonly used; waypoint navigation and vector field navigation***. Waypoint navigation is the simplest to implement as this involves creating waypoints and heading directly for them, one after another, using the robots current location and desired location. It may also be extended to use cross track error to bring the boat back on the line between the previous waypoint and the next waypoint (as this minimises the distance travelled).

Vector fields use a much more complex strategy. Different objects, movements and behaviours can be described by equations that put a "force" on the boat *** (so obstacles would have a repelling force, and waypoint an attractive force). The forces will depend (in most cases) on distance from a the location of a waypoint or obstacle. These forces are therefore directional and can be represented as a vector. By adding the vectors, an overall force is found which is the direction (and perhaps speed) the the boat should go at. This is much more complex to visualise than waypoint navigation, but for achieving complex behaviour the code is much simpler with vector fields and is much more efficient.

The waypoint navigation cannot easily handle avoidance of an obstacle; a series of points around the obstacle must be plotted, whereas with vector fields a point with a repulsive force can be added at the obstacles location with a weight that relates to the size of the obstacle. The obstacle can be easily remembered throughout the rest of the robots navigating with no extra logic needed, whereas with waypoint navigation a new course must be plotted around the obstacle every time the robot approaches it.

2.2.2.2 Behaviours

The behaviour of the robot should be selectable and easily changeable. There are simple behaviours the robot could use to fulfil the task of area scanning a lake, and there are much more complex behaviours.

The simplest behaviour possible is navigating toward a single point. More complex than this is navigating to a series of points. These two behaviours are very common ***.

With this project it is hard to find algorithms that instruct a robot to scan an area in the most efficient way: path finding algorithms would find the shortest route between two specified points, but would not yield the most efficient way to navigate to every point within an area. This problem is similar to the travelling sales man problem ***, but the points are generally in a predictable layout, so a computationally heavy algorithm like those needed to solve the travelling salesman problem*** are not needed.

2.2.2.3 GPS Driver

To begin with the GPS was going to be attached to the Arduino...***.

In the end it was decided the GPS should be placed on the Pi. In order to control it it was best to make a driver for it, again so that it could be easily swapped for new hardware, but also to enable different systems to be used to interface with it. GPSD is very commonly used with Python to interface with a GPS, as it handles it very well and makes the data from the GPS easily readable. Therefore it was decided to use GPSD.

2.2.2.4 Vector Classes

Having decided in subsubsection 2.2.2.1 to use vector field navigation ***

2.2.2.5 Logs

2.2.3 Arduino-Pi Communication

***Need a way of sending messages both ways between the pi and Arduino. It would be complicated to have both the pi and Arduino able to initiate messages between them, as this could cause clashes. A slave/master system is more appropriate, with the pi as the master, sending the initial messages.

The messages will consist of a single letter (e.g. 'e'), or single letter followed by a number in brackets (e.g. 'h(276)') if a number is required. This is for efficiency; the as messages on the Arduino are stored as a char array and so must be compared char by char, rather than a simple comparison of strings (comparison of strings would make extracting the number that needs to be saved more challenging, and would probably result in the String being converted into a char array). It is sufficient enough to use a single letter for the meaning of the message; and there are only 4 - 5 *** meanings that need to be transmitted, so there is no danger of running out of letters to use.

***The use of brackets means that the code can be sure exactly where the number is expected to start and end – if there was no character signifying the end of the message, if an extra number was transmitted accidentally by another part of code, or through noise on the connection, then the number read could be very wrong. It also makes defining the end of the message easier, as it may not be known how the message will be terminated (a new line, carriage return or null terminator are all valid options).

It would be ideal to have a confirmation message to so that the message had been received every time a message is sent, but implementing this seems too over engineered for this system; if a single message is missed, there will be a new message in around a second. The only place where there needs to be confirmation, is during the shutdown sequence. Here the Arduino needs to turn the motors off before it is safe for a person to pick it out of the water, and so a missed message saying to shutdown cannot be ignored. Therefore, during shutdown the pi will ask the Arduino to sleep by sending the letter 'e', and will expect it to respond with 'e' if the message is received. If the pi does not get the response within a set time*** it will send the message again. If after five*** attempts the Arduino still has not responded, it can be assumed something has gone wrong and that the python code should exit anyway.

Chapter 3

Implementation

3.1 Tools

Originally this software was developed using PlatformIO *** as this gives a more C like file structure to the code, and allowed drivers to be easily imported into test files to allow the drivers to be tested. Unfortunately PlatformIO is not easily installed on a Raspberry Pi, where as the Arduino IDE can be easily installed, and so the code was adapted to be used with the Arduino IDE.

This project used GitHub for version control and for backing up code. As this project is open-source, it is held on GitHub under the GPL-3 license and is therefore available for others to use.

3.2 Hardware

The first step before writing the lower level Arduino was was to decide exactly which type of Arduino was to be used and which environment the code would be developed in. There was found to be very little difference between different Arduinos (apart from which pins are defined for different functions). As there is not much hardware (only three pieces, requiring 4 pins) it was decided there was no need to move away from the Moteino board already in the boat as it fits the requirements (enough pins, uses 3.3V output, can use serial to communicate with it). An arduino Uno was used during some of the early stages of coding as the Moteino was stuck to the boat so could not be used outside of the Robotics Lab. Later a Moteino not attached to the boat was used for development, before uploading code to the boats' moteino (as early development may have caused the hardware attached to the boats' moteino to act in unexpected ways).

It also had to be decided which board would control the higher level logic. A raspberry Pi 3 was the most suitable board available; has multiple USB ports for the GPS and Moteino to be connected to, has WiFi for easy control (can easily SSH in to start the code, see output on a separate machine from some distance, can edit code from another machine) and is small and lightweight. Previous attempts to give this boat a control system have used small laptops (which are considerably bigger than the Pi) or no *** at all (reducing the computing power of the system considerably).

3.3 Arduino Code

3.3.1 PID Controller

The PID was implemented using the difference between two headings (the current heading and the desired heading) as P. Finding the difference between two headings was challenging due to the use of a circular scale, as it is not simply subtracting one value from the other as would be the case with a linear scale (the smallest angle between 5 and 355 degrees is 10, but by simply subtracting these values you get 350 or -350). With this circular scale, the values wrap at 360.

There are solutions that are extremely simple which use trigonometric functions to calculated the difference between the two headings easily, preserving the sign (which indicated a clockwise/anticlockwise turn). But these functions are computationally inefficient, so it is best to avoid using them where possible.

A more efficient way is using the equations below to calculate different angles between the headings:

$$(360 - A) + B \tag{3.1}$$

$$B - A \tag{3.2}$$

$$(B - 360) - A \tag{3.3}$$

The result of Equation 3.1 , Equation 3.2 and Equation 3.3 with the lowest absolute value is the smallest angle that can be moved through to get from A to B. A positive result indicates a clockwise turn, and negative indicates an anti-clockwise turn. Which equation gives the smallest result is dependant upon which quadrant the two heading lie (0-90,90-180,180-270 or 270-360). This solution works, but may not be the most efficient solution.

To tune the PID loop, different methods of tuning were researched. The two main methods investigated were the ZieglerNichols method, and manual tuning. According to a review of these methods by Wikipedia [10], the manual tuning requires experience whereas the Ziegler-Nichols method does not. Because this robot is a boat and tuning of the PID requires access to a large body of water for the boat to move on, it was impractical to tune the PID manually, so it was decided to use the Ziegler-Nichols method instead.

To make the boat turn tight corners it was decided to make the boat travel at half the maximum speed that the boat can travel at.

3.3.2 Rudder Driver

To calibrate the rudders, a test program was used that took the rudders left, right and centre to test if the directions were correct, and to see if the limits were too high or low.

3.3.3 Motor Driver

Due to the modular design of the system, and given that all parts relating to the motors should be easily swappable, it was decided that the main code should only perform logic in units of ms^{-1} rather than the raw motor values (which here is given in degrees). This is so that any motor working with different units or scales can still be used.

Therefore, the driver includes functions that translate from the scale the motors use to ms^{-1} and back. The maximum, minimum and stop raw values are held in the driver, and can be translated and returned to the main program by calling `getMaxSpeed`, getMinSpeed and getStopSpeed for the main program to work with (e.g. find a half max forward speed, by finding the average of getStopSpeed and getMaxSpeed). There is also a stopMotors function that will stop the motors (which is useful at the end of the program). There was the idea to detach the motors from the pin to prevent them being written to later in the program, but this would instead cause the values written to the motors to be left floating (changing randomly over time) which would be undesirable and dangerous (if removing the boat from the water at the end of a run, the motors could start turning at any speed at any time).

In initMotors, there is a five second delay after the motors are set to their armed value, which is required by this type of motor before any other values will work. This delay may be larger than necessary, but ensures that the minimum time is definitely reached. The need for a delay was noted in Aled*** Davies' original code for these boats, which implemented semi autonomous features (such as heading holding).

The difficult part of this design is that it may not be known what speed the motors move the boat at, and this will vary under different conditions (e.g. high winds, strong currents).

The motors were calibrated by firstly using a simple test program that sends varying raw values to the motors and seeing how the motors respond. It was found that the theoretical maximum value of 180 was too high for the motors and so the motors would not turn on.

3.3.4 Compass Driver

The compass library is based around the Adafruit code *** libraries designed to be used with this compass. The example code to use the library was copied and adapted for use with this system; notably the heading was converted from radians to degrees.

By testing this simple driver on the compass, it was found that the compass readings were not correct. It was assumed that a simple zero error would be the problem, but further testing revealed this was not the case (see results in Appendix B). The compass measured different changes in degree at different bearings even though it was being moved through the same angle. Thus the initial design of simply adding an offset to the compass would not work.

To investigate this further, the compass was glued into the boat which was placed on a wheeled platform and rotated through different angles. A calibrated compass was placed on the boat. The boat was turned in steps of 20 degrees *** from 0 to 340 according to the boats compass, and the actual heading according to the calibrated compass was recorded. By plotting this on a graph in LibreOffice Calc*** it was clearly visible that there was not a linear trend between the boats compass headings and the real headings.

To compensate for this, a trend line was plotted through the points and the resulting equation was used to transform the compass headings into actual headings.

An important point to note about this equation is that when the actual heading becomes 0, the points will shift on the graph so that a trend line cannot be found ***. For this reason, the value the compass reads as 0 (due north) was recorded, and any values greater than this must have 360 taken away from them before being passed to the transformation. The results of this can be seen in figure 3.1.

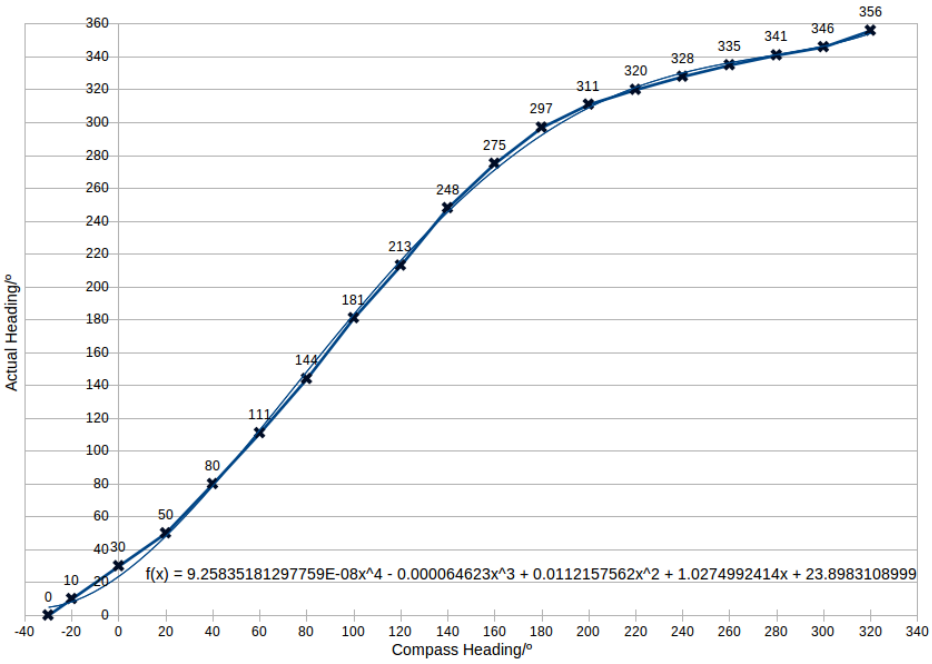


Figure 3.1: Graph of compass heading vs actual heading. The equation is that of the trend line.

This transformation was added to the compass driver, so that any values returned to the main program will be calibrated. It was checked that values of 360^2 could be held by a float in Arduino C as this comes to a very large number (1.68^{10}). Fortunately floats can be between 3.4028235×10^{38} and as low as $-3.4028235 \times 10^{38}$ [11] so there is no way that any values can wrap in this equation.

3.4 Pi Code

The first step in implementing the higher level logic on the Pi was deciding which operating system and language should be used.

3.4.1 Navigation

Vector fields were chosen as the navigation system. To aid in the implementation of navigation, a class called Location was implemented. This simple class holds latitude and longitude in both radians and degrees. It was decided to implement this class like this so that the conversion only has to be done once, saving on unnecessary repetition of the conversion during calculations. This could have been implemented using a numpy array*** instead of a new class, but this would have lead to the confusion between latitude and longitude and which way around they are held in the array (which would cause confusion when implementing complicated equations).

Three different vectors types designed were called Lines, Points and Planes. These objects all create a force in a specific direction, which when converted to vectors and added together results in the direction the boat should be travelling.

Line objects represent a line between two Locations, where the force between the Locations acts perpendicular to the lines, and at the ends of the lines the force acts in the direction from the boats current location to the location of the end of the line (see Figure 3.2).

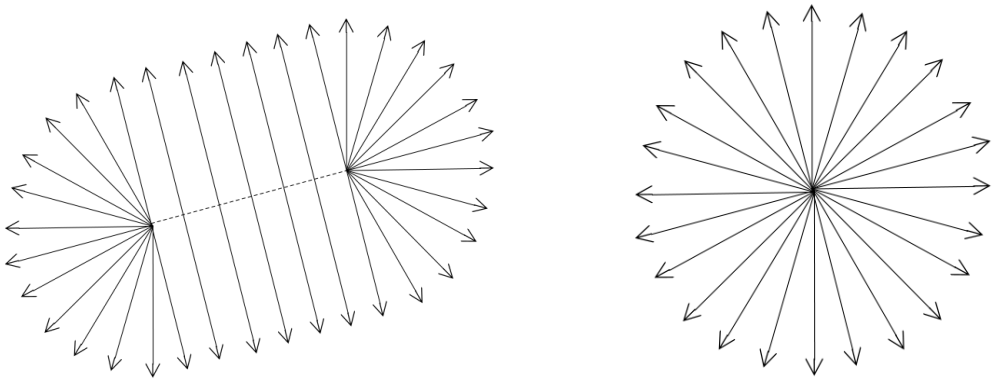


Figure 3.2: Visualisation of the direction forces will act with the Line and Point objects. The arrows represent the direction of a repulsive force; an attractive force would be antiparallel.

The forces on the Lines and Points change with distance to the object. For attractive objects the equation $F = mr^2$ was chosen, where F is the overall force on the boat, m is the weighting of the object and r is the distance from the object. For repulsive forces the equation $F = \frac{1}{mr^2}$ was chosen. These are represented in Figure 3.3.

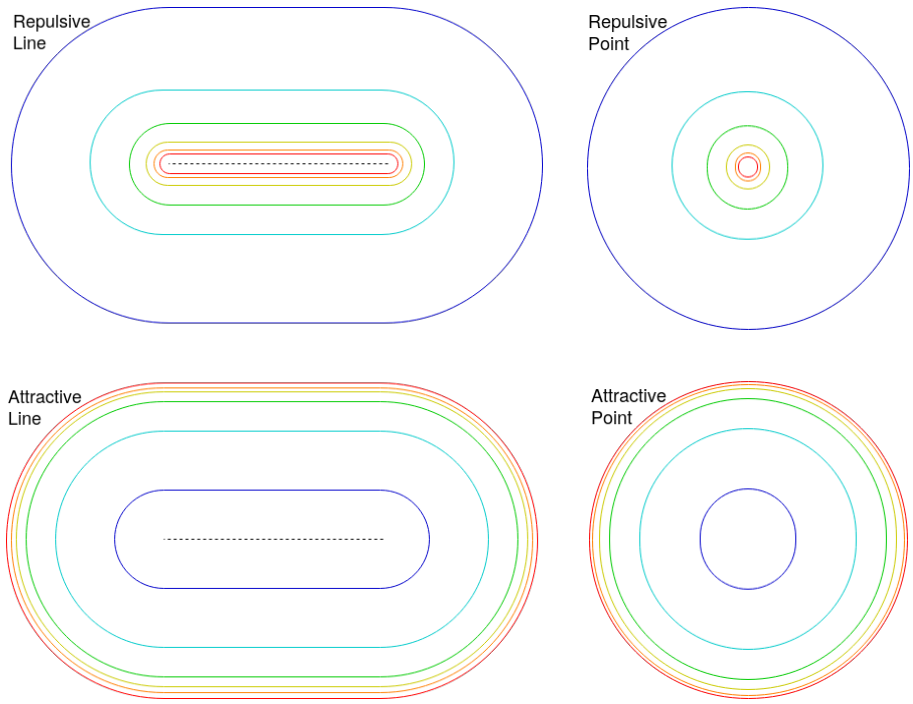


Figure 3.3: Visualisation of the strength of forces acting on the boat with distance from the object. The coloured lines represent equipotential forces, red representing a very high magnitude force force and blue a very low magnitude force.

These lines and points can be added together in different ways at different locations to make the boat exhibit a desired behaviour.

The input files were formatted as .csv as this is a common format and python has modules that can be included to read .csv files easily. Example files from all the tests can be seen in section B.1.

3.4.2 Behaviours

After exploring several options, a simple algorithm was devised for scanning bodies of water of almost any shape; given the outline of the lake as points defining the perimeter, the two furthest points apart are found. Between these two points, lines perpendicular to the line joining the two points are found at regular intervals (the frequency at which a user would like traces of the water to be taken). The ends of these lines are limited to 5m within the boundary of the water, and attractive Point objects are initialised at these points. The lines between the points are also initialised as Lines. By storing successive waypoints and lines in a list, and iterating over this list, the waypoints can be navigated in a logical order to scan the entire area (the lines would be used to keep the boat on a straight line between the points). See Figure 3.4 for a visualisation of this.

Another strategy explored was the spiral, where points are placed 5 metres in from the points defining the boundaries of the water until the centre is reached, drawing lines between points equidistant from the perimeter, and following those lines. This would be much more complex to implement, and a comparison of these strategies as seen in Figure 3.4, led to the conclusion that the spiral covers more distance than the square-wave strategy, or at best covers the same distance.

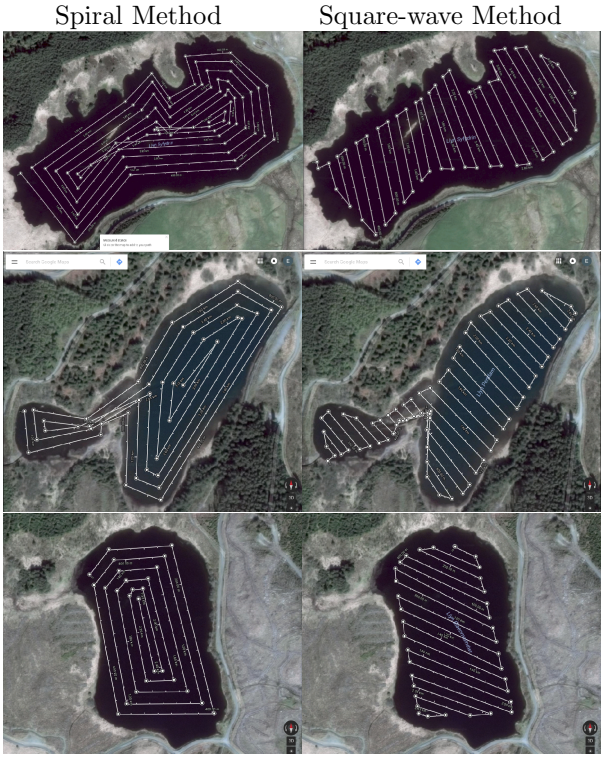


Figure 3.4: A comparison of possible area scanning strategies. The top lake yields a result of ***km for the spiral, and ***km for the square wave, the middle a result of ***km and ***km, and the bottom *** km and ***km for the spiral and square-wave strategies respectively.

***move this t̂o testing?

The square-wave strategy appears to work in most shaped lakes but there are cases where it will not work; most notably it will not work on an ox-bow lake, nor a meandering river (though it should be noted that this boat should only be used at most very slow moving rivers in order to avoid been dragged downstream – a lake is the preferred environment).

3.4.3 GPS Driver

Implementing the GPS driver was challenging as the GPSD software had to be installed and then investigation had to be done into how to use it. ***

3.4.4 Logs

3.5 Arduino-Pi Communication

this was interesting as the python and arduino C default print in two different formats. The python needs to be cast to b' ' *** before it can be printed. It was decided to leave the arduino in its default read/write format as libraries or logic to convert the values would waste memory and time in conversion.

Serval serial communication libraries were investigated in Arduino C and Python, that claimed to handle easy communication between the two. *** After spike tests it was decided that these libraries were difficult to use and therefore were not used.

The python also by default included the \n and \r characters in the read in string (so if the arduino send

'e', the python code would read and save this as 'e\n\r'). This is not ideal for easy comparison of strings, so the .strip() function was used to remove all syntax*** from

Chapter 4

Testing

4.1 Preliminary Tests

At the very start of the project the existing system was tested on the water to see how fast the boat can travel, how fast it is able to turn (and the size of the turning circle) and how much it is affected by external factors (such as waves, currents and wind). This was done by moving the boat under remote control.

Data was recorded by the logging system from the previous, semi-autonomous control system.

4.2 Arduino Tests

Unit tests were nearly possible when using PlatformIO but in reality it was impossible to make them work. When testing in ArduinoIDE there is no support for unit tests, so alternative approaches had to be taken.

4.2.1 PID Controller

Testing the PID tuning proved difficult, as this must be done on the water, and in order to check the PID tuning, there must be a way of seeing if the boat is oscillating about a heading or not. This is difficult as this cannot be done from shore; one of the main motivations for this project is the impossibility in telling if the boat is travelling in a straight line or not from shore. It would be possible to record the location of the boat according to GPS as it travelled, but GPS is only accurate to 5***meters which is not accurate enough for fine tuning. The added fact that waves and currents will effect the motion of the boat unpredictably means that fine tuning the PID was impossible.

Some tuning of the PID was done during the land tests described in subsection 4.7.1, but the tuning was found to be wrong during the water tests (described in subsection 4.7.2. This is due to an underestimation of the speed of the boat and therefore how fast the boat must move its rudders in order to stay on course.

In the end, the Ziegler-Nichols methods implemented were removed during the lake tests and the manual method was implemented instead (as this is the simplest method) by increasing the value of P until the rudders moved sensibly and proportionately to a change in heading (turning the boat on the spot). The PID was not tuned any further as this was sufficient for the boat to move on the correct heading.

A way to improve the tuning in future would be to enable the PID to be adjusted on the fly (through use of wifi to the Pi and passing messages to the moteino via serial, or through another moteino talking to the boats moteino over radio). By having the PID easily adjustable, and then following the robot in a boat so it can be seen close up how the robot is moving, it should be possible to finely tune the PID. This was not possible in this project as there was not time, and fine tuning was not critical to making the control system functional.

4.2.2 Rudder Driver

4.2.3 Motor Driver

4.2.4 Compass Driver

The main tests for the compass driver involve testing the calibration, as described in the implementation in subsection 3.3.4.

Looking at Figure 3.1 in subsection 3.3.4, it can be seen that the final equation still does not perfectly calibrate the compass, as there are places where the trend line deviates from the data points; specifically between a compass heading of -20^o and 20^o(which corresponds to an actual heading of 10^o to 50^o). The deviation is at most 10^o, so this should not have too great an impact on the navigation of the boat; the heading may be out by 10^o at some times but the PID*** should compensate for this. The most impact this will have is in making the boat curve towards its target instead of taking a direct course. Further investigation of this problem is discussed in subsection 3.3.4.

Having tested the final equation, it was observed that the compass showed the corrected heading when turned to 90^o, 180^o, 270^o and 0^o. This was a drastic improvement upon the original driver.

4.3 Arduino-Pi Communication

Communication was tested using the main code on the Arduino and python test code on the Pi. The python test code included only bits of code necessary to send messages in order to solely test the functioning of the communications, for these bits of code to be later integrated into the main python code.

4.4 Python Unit Tests

4.4.1 GPS Driver

4.4.2 Vector Classes

4.5 Logs

Testing of the logs was done by running the program and seeing if the logs.csv file had new lines in the correct format. This was done by visual inspection and by passing the file to GPS Visualizer *** and seeing if the file could be read.

After the first test around the***

4.6 Navigation and Behaviours

In order to test the navigation, the behaviours (which handle navigation) had to be implemented, and were therefore tested together.

4.6.1 First Castle Grounds Test

4.6.1.1 Method

In order to test the heading resulting from the vector field at different locations, a single point was placed in a field by a known landmark (so the location could easily be identified), in this case the intersection of the

paths in front of the memorial. A perimiter was plotted the field, as defined by the paths around the feild. The point and the boundary were both given a weighting of 10 and -10 respectively.

The pi was then moved around the field semi-randomly to see how the heading varied in different locations. The heading and distance to the waypoint were printed to the screen, as monitored by a laptop, from which there was an SSH session into the Pi.

4.6.1.2 Results

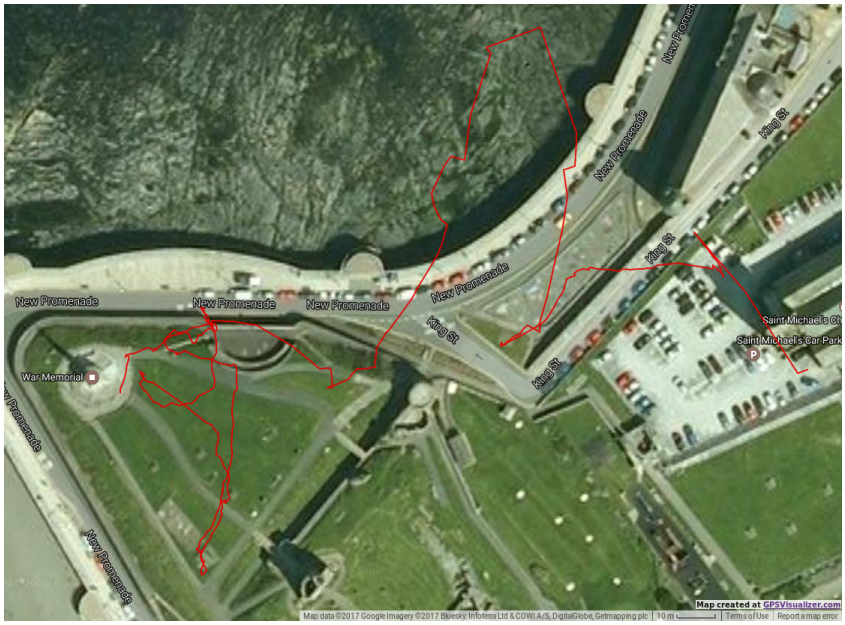


Figure 4.1: Carrying the Pi and GPS around a field to test the heading given by the Pi as given by the single waypoint behaviour. The red line in both images show how the Pi's location changed (according the the GPS), with the zig-zag indicating the desired path. The arrows on the right image show the desired heading at each location.

4.6.1.3 Results

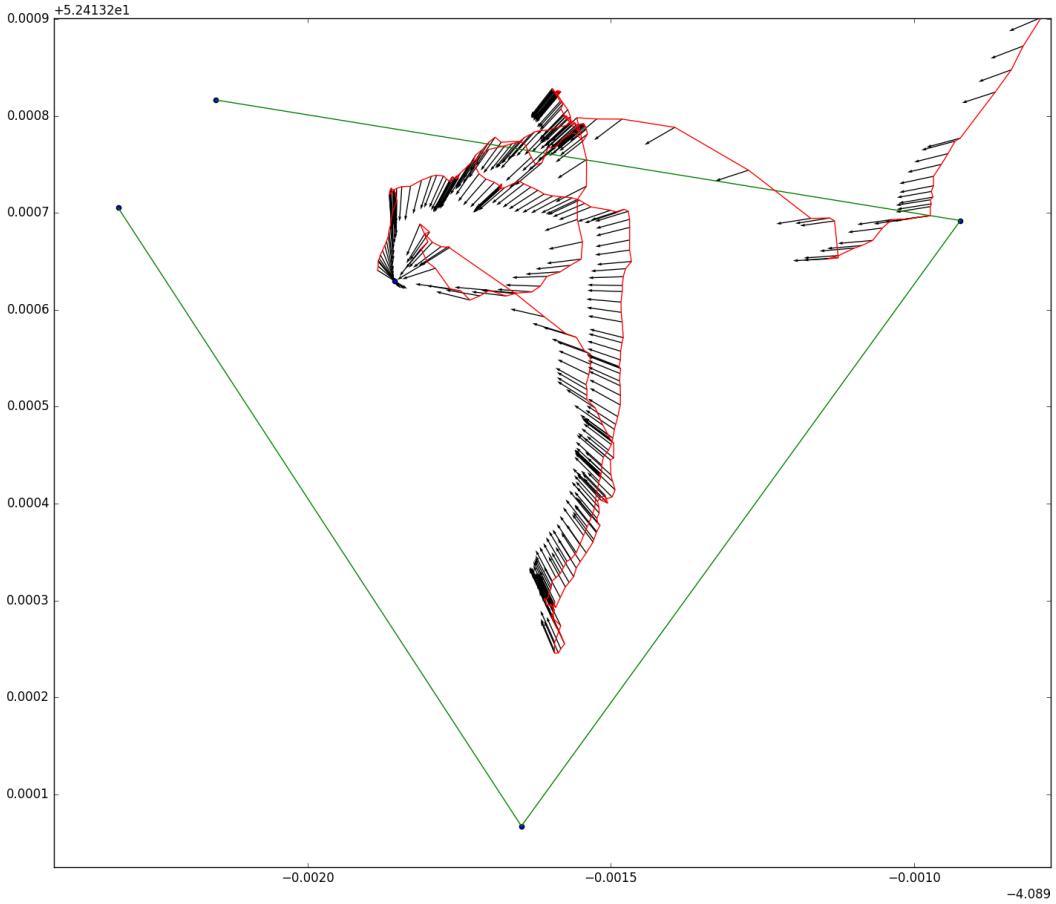


Figure 4.2: Carrying the Pi and GPS around a field to test the heading given by the Pi as given by the single waypoint behaviour. The red line in both images show how the Pi's location changed (according the the GPS), with the zig-zag indicating the desired path. The arrows on the right image show the desired heading at each location.

4.6.1.4 Analysis

It was found that the vector field was not responding correctly to the boundary of the field. It was expected that near the boundary the heading would become perpendicular to the boundary (in order to steer the boat away from it), especially as the force would tend to infinity the closer to the boundary it was.

This is most likely because there was no limit on the attractive force of the point, so when the current location is far from the waypoint, the force tends to infinity. This means that for the boundary force to become significant, it too would have to tend to infinity, which would require being extremely close to the line. Looking at the results of the plotted vector, it can be seen that specific points were in fact directly on the line but were still not pointing in the correct direction.

To solve this problem of attractive objects overpowering repulsive objects, a limit was placed on the size of the attractive vectors. This means that the boat will always tend to head towards attractive objects, except where there is an obstacle, and avoiding obstacles takes priority.

4.6.2 Second Castle Grounds Test

4.6.2.1 Method

Testing of the single waypoint behaviour*** was test using a raspberry pi with only a gps attached. Every new desired heading was saved along side a the current GPS location, and in later tests the time stamp. GPS co-ordinates outlining a feild and waypoints within the feild were found using GPS Visualizer and were passed to the code as further discussed in subsection 3.4.1. To test the algorithm, the Pi was moved around the feild in a zig-zig shape to cover the whole field, including right up to the field boundaries and beyond.

4.6.2.2 Results

The logged locations can be seen in Figure 4.3 for the castle field tests.

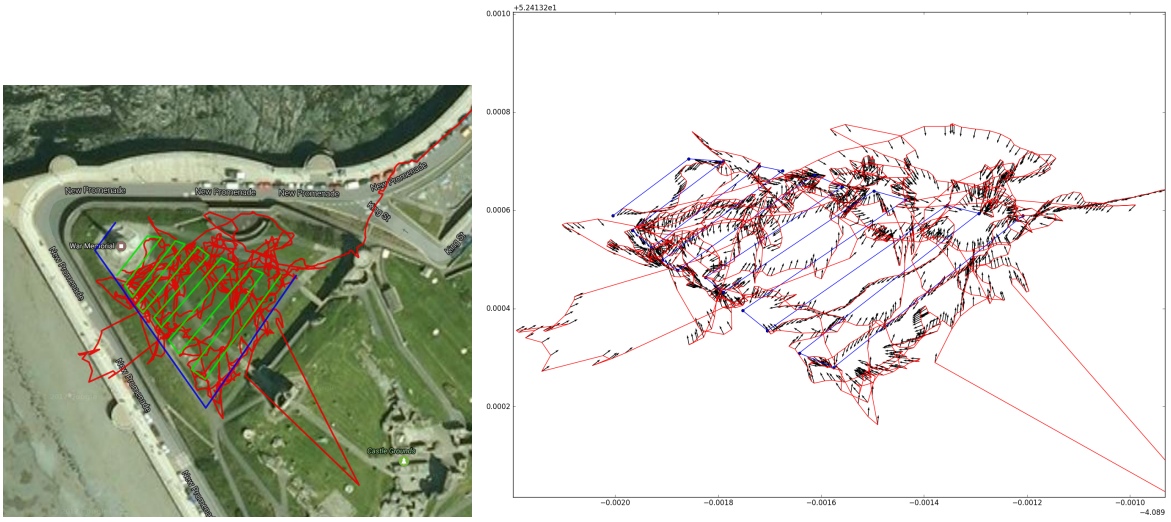


Figure 4.3: Carrying the Pi and GPS around a field to follow the waypoints (as planned by the simplescan behaviour), resulted in unreadable data being logged, but feedback during the tests confirmed that the behaviours were acting correctly. The red line in both images show how the Pi's location changed (according the the GPS), with the zig-zag indicating the desired path. The arrows on the right image show the desired heading at each location.

4.6.2.3 Analysis

It was found by looking at the desired heading on when at each location within the field, the algorithm was working correctly. The logged data for these tests is very hard to interpret (see Figure 4.3), but for these tests logging was to serve as a potential supplement to the test and as a record of the test, rather than being an integral part of the test.

4.6.3 Large Field Test

After seeing the field by the castle was too small for accurate readings, it was decided to move to a bigger field with less structures around to block the GPS signal. The bigger field also better replicates the size of a lake or body of water likely to be used with the boat.

Another advantage to using this feild is that it is a football pitch with markings so it could be easily seen while carrying out the test where the Pi was in relation to where it should be navigating to (i.e. the long legs

of the course should result in the Pi giving a heading to move parrallel with the width of the pitch). This made the test much easier to carry out.

4.6.3.1 Method

Again, a series of waypoints were plotted in the field with the outside edge of the field as the 'water' boundary.

For this test a magnetic orienteering compass was used to find the exact direction the Pi was saying to move in. It was then attempted to move in that direction to test how well the navigation worked.

More specifics of the test are outlined in the appendices ***

4.6.3.2 Results

The results can be seen in Figure 4.4 and Figure 4.5.

Figure 4.4: Carrying the Pi and GPS around a field to follow the waypoints (as planned by the simplescan behaviour), resulted in unreadable data being logged, but feedback during the tests confirmed that the behaviours were acting correctly. The red line in both images show how the Pi's location changed (according the the GPS), with the zig-zag indicating the desired path. The arrows on the right image show the desired heading at each location.

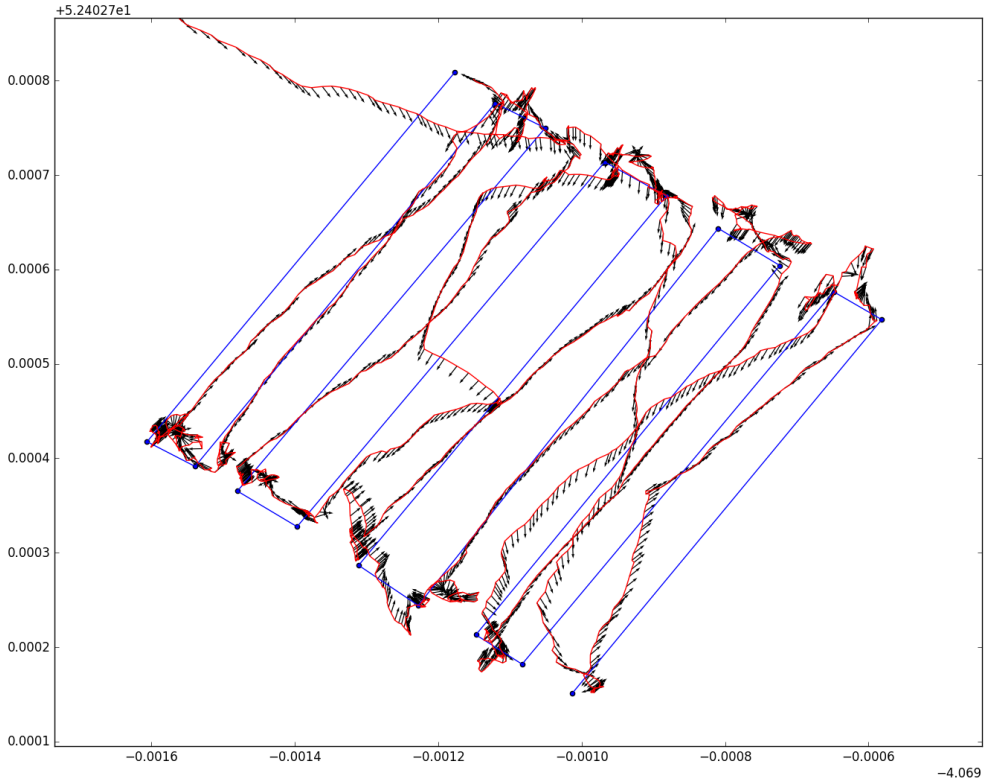


Figure 4.5: Carrying the Pi and GPS around a field to follow the waypoints (as planned by the simplescan behaviour), resulted in unreadable data being logged, but feedback during the tests confirmed that the behaviours were acting correctly. The red line in both images show how the Pi's location changed (according the the GPS), with the zig-zag indicating the desired path. The arrows on the right image show the desired heading at each location.

4.6.3.3 Analysis

***need to analyse these results

4.7 Overall Functionality

To test the functionality of the overall control system two different types of test were used; land tests and water tests.

4.7.1 Land Tests

Firstly, the boat needed to be tested on the land, as testing a control system on the water when it has not been tested all together before could result in losing the boat.

4.7.1.1 Method

Therefore so called "Car park tests" were carried out on the boat; this involves loading the full program as though it is about to go out on the water, but instead carrying it or placing it on a trolley (see Figure 4.6) to be wheeled around a field or large open space (such as a car park). By watching the rudders and adjusting the boats heading according to the rudders, it is possible to simulate how it would move on the water.

With this technique it is possible to tell if the final behaviour of the boat is correct, and it is possible to tell if it is safe enough to place on the water (e.g. it doesn't try to move in the completely wrong direction at any point).

4.7.1.2 Results

Figure 4.6: Testing boat in a field on a trolley. The right hand picture shows the route the GPS recorded. The purple zig-zag shows the most efficient route between the waypoints chosen (with waypoints at each turn in the purple line). The boat started at the top of the red line, and ended at the dark purple line on the right. The different colours indicate a change in which waypoint is trying to be reached.

4.7.1.3 Analysis

With the first land test, the boat seemed to move in large loops rather than directly between the waypoints, though this could be as a result of the GPS inaccuracies (this will make the boat want to go on a different heading as it appears to be in a different location to where it is in reality).

The GPS location results for this test are shown in Figure 4.6 . It can be seen that there is a lot of variation with the GPS readings, as expected with this type*** of GPS, and in a somewhat built up location. The bright green line and dark blue line are particularly inaccurate, but this was the case only for a short amount of time.

As the GPS was jumping so much, it was hard to tell how well the control system was working so it was decided to test it on land again using points further spread out (***

4.7.2 Water Tests

4.7.2.1 Method

4.7.2.2 Results

On first observation, the results of the water test looked very promising. As can be seen in Figure 4.8, during both runs of the test the boat followed the northward and southward sides of the route almost perfectly both times. The general direction for the rest of the route was correct, except at the start of the run both times where the boat was moving on a bearing that was roughly 30^o clockwise of where it should have been moving. This was thought to be a result of an error in heading difference calculations (as discussed in ***), but this was not fully fixed between tests.

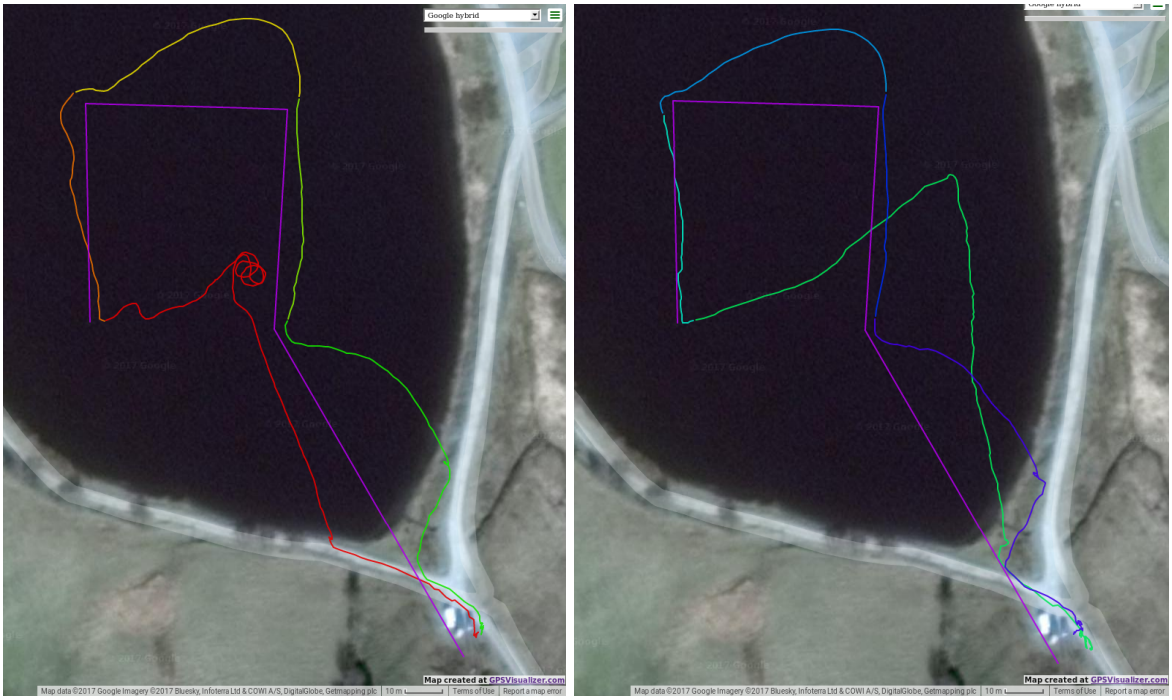


Figure 4.7: The lake.

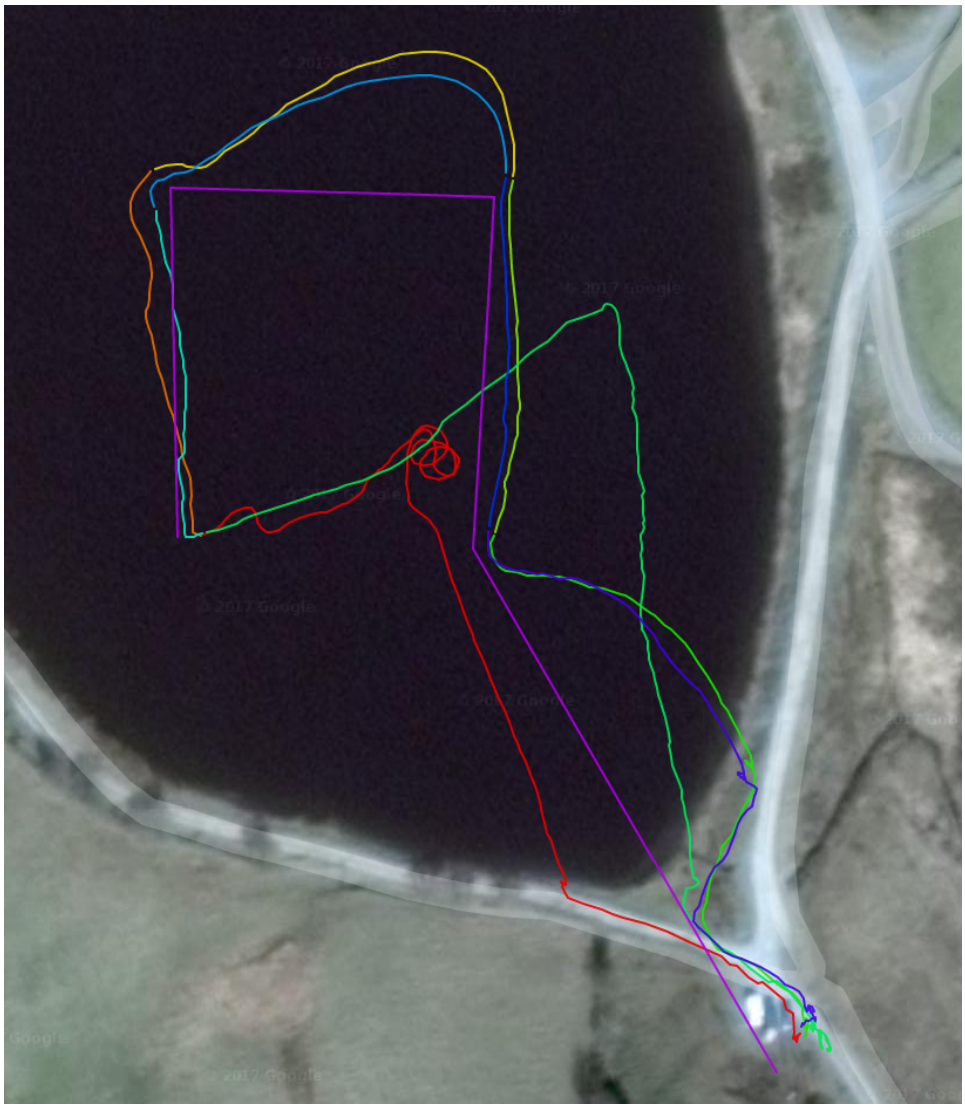


Figure 4.8: Both traces from lake tests. The first test is the trace that starts turquoise and goes to purple as the trace moves clockwise. The second test is the trace that starts red and fades through orange, yellow, green.

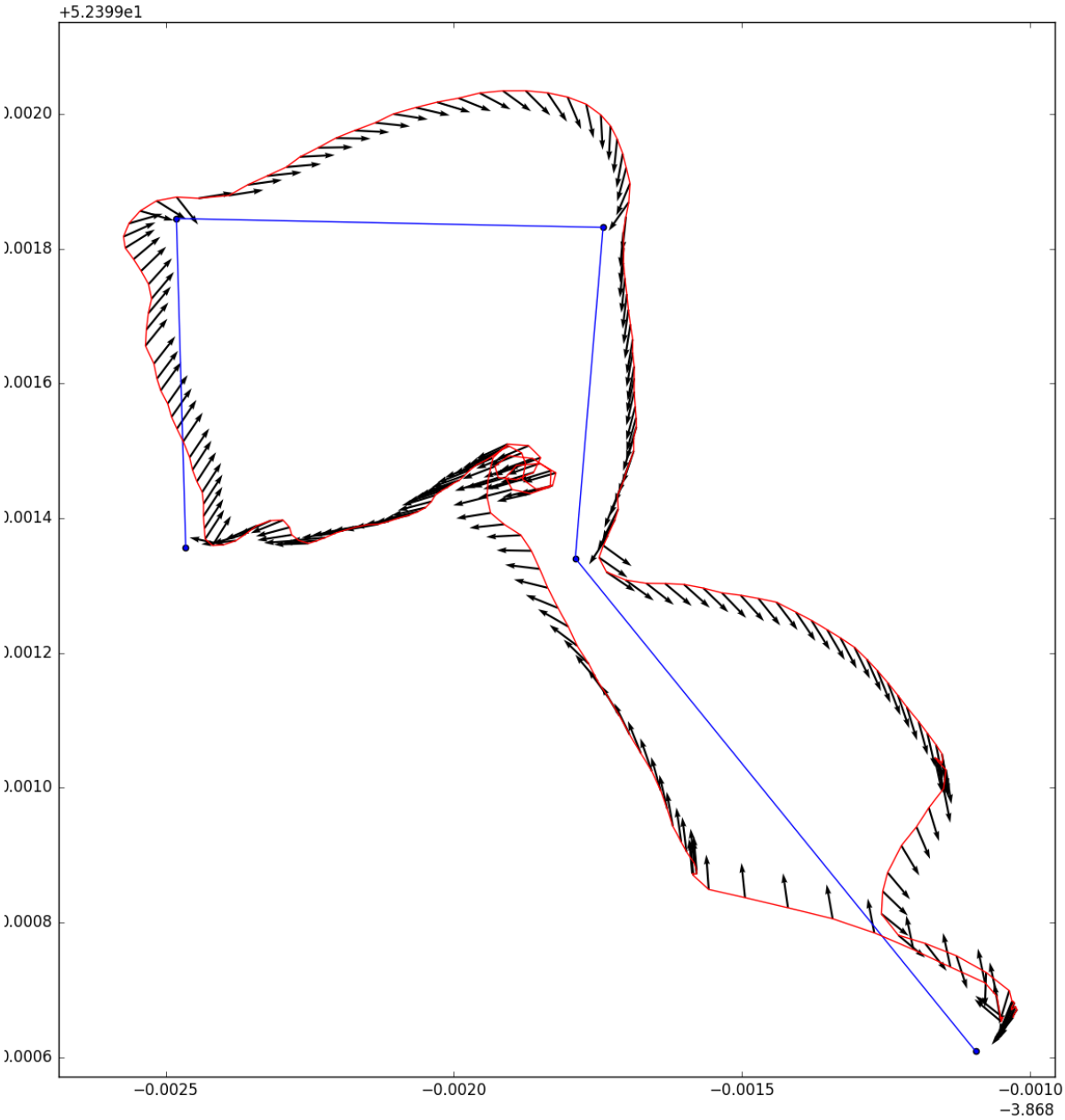


Figure 4.9: Plot of latitude vs. longitude of the boat during the first lake test (red line) with arrows at every 3rd point indicating the Pi's desired heading. A plot with arrows at every point resulted in arrows too small be seen here (a .svg file is included in the code submission). The arrows direction is adjusted in accordance with the scale of the x and y axis, so absolute angle may not be accurate here, the relative angle is (an arrow pointing directly at a waypoint will always do so no matter the change in scale).

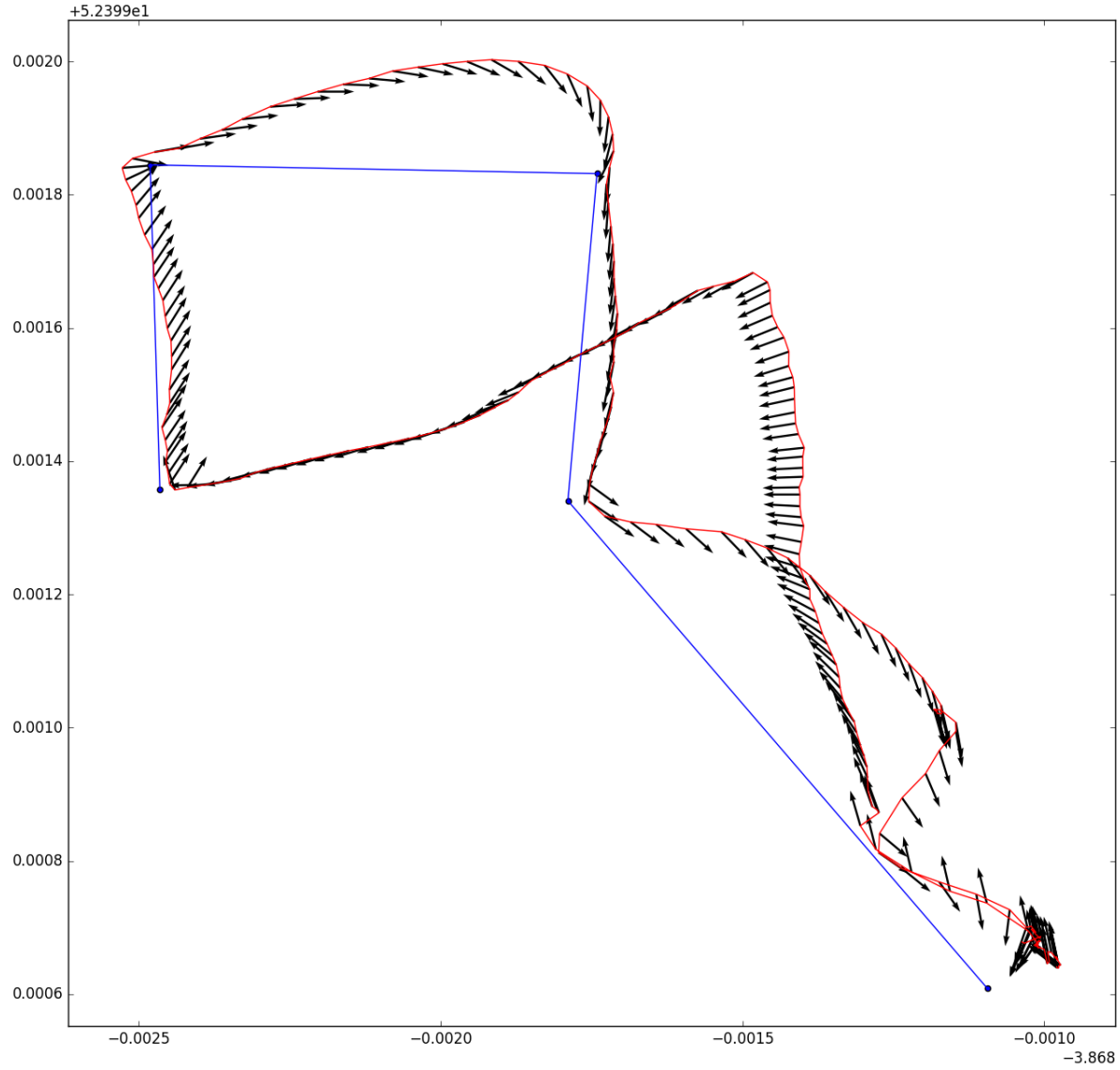


Figure 4.10: Plot of latitude vs. longitude of the boat during the first lake test (red line) with arrows at every 3rd point indicating the Pi's desired heading. A plot with arrows at every point resulted in arrows too small be seen here (a .svg file is included in the code submission). The arrows direction is adjusted in accordance with the scale of the x and y axis, so absolute angle may not be accurate here, the relative angle is (an arrow pointing directly at a waypoint will always do so no matter the change in scale).

4.7.2.3 Analysis

Further processing of these results, plotting the desire heading at each location narrowed down whether the python code or the Arduino code was at fault for the detours at the start and the looping behaviour at the north-most leg of the course.

As can be seen in *** and *** , during the second half of detour at the start of both tests the python code was not at fault as it was saying to move almost directly towards the first waypoint. Initially Pi's desired heading was slightly too clockwise but became correct after the first half of the detour.

Between the first and second waypoints, the boat managed to move in the correct direction but this is not in fact the route the Pi was saying to take. The desired headings from the python code were around 50^o clockwise of where they should have been.

The northward drift between the second and third points was most likely due to the wind and waves (as they were generally blowing east or north east). This drift should have been corrected by the python code (as discussed in ***) but was not (the expected result would be seeing the arrows pointing more southward as the drift got larger, around a third of the way along this leg).

Chapter 5

Evaluation

Further testing needs to be carried out before this system completely satisfies the aims of this project (in its current state the boat will make it to the waypoints but will not necessarily take the most efficient route).

The aims of this project (as set out in section 1.2) were acheived to the following extents:

This project did not initially identify the full aims of this project correctly, simply stating the aim of working control system code. Having worked through this project it is evident that there are different levels of 'working' and that it is not as simple as working or not working. The control system in its current state does work, and the boat is able to navigate to the required waypoints, but this requires manual input of waypoints in the correct order into the correct files, and the course the boat takes is not efficient.

In some ways this project has succeeded, as the boat is now able to scan a body of water autonomously, even if it is somewhat inefficient.

Appendix A

Appendices

A.1 section A.1: Ethics Questionnaire

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
eas12@aber.ac.uk

Full Name
Elizabeth Stone

Please enter the name of the person responsible for reviewing your assessment.
Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
rrz@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
MMP Backpackable Robot Boats for Sonar Surveys

Proposed Start Date
30/01/2017

Proposed Completion Date
08/05/2017

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
Develop a control system for a small light-weight motor boat in order to scan a body of water autonomously.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
No

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here:

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.
Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.
Yes

Please include any further relevant information for this section here:

A.2 section A.2: Additional Libraries

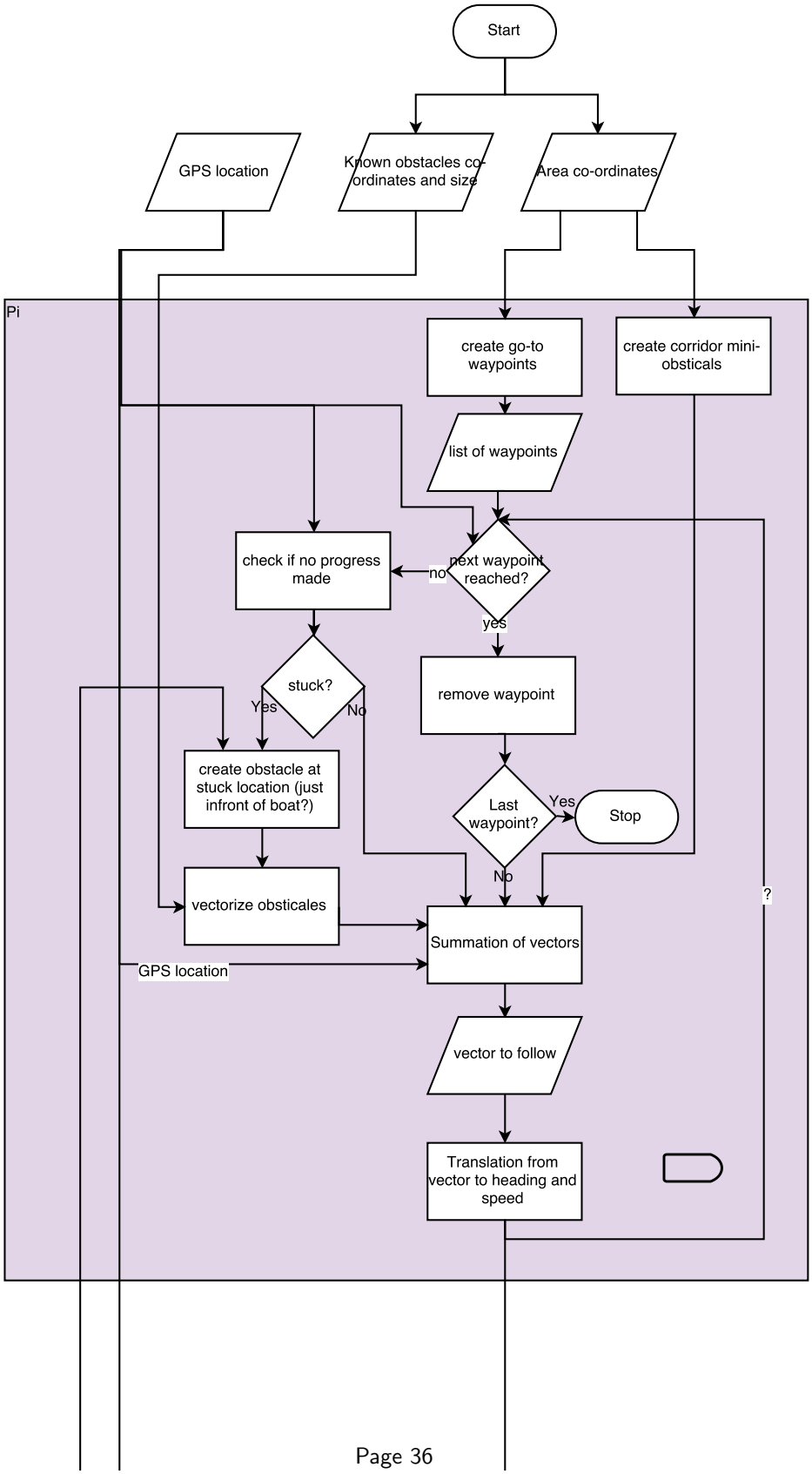
The following libraries, or code, were used or built upon in this project.

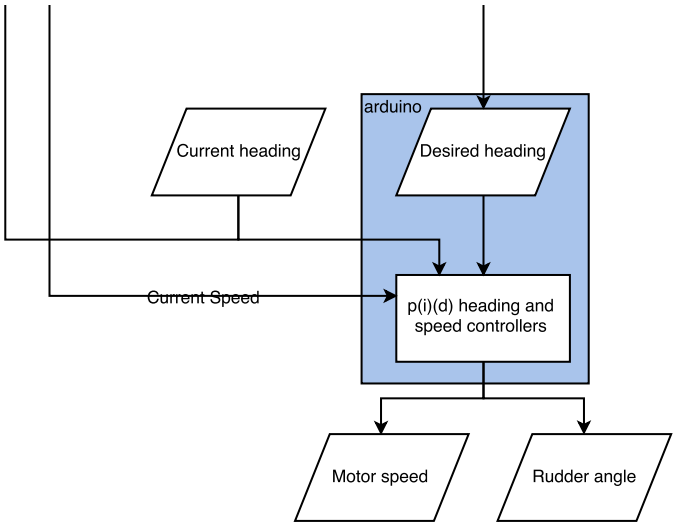
- **Arduino, avr/sleep, Wire, Servo** Come as standard with the ArduinoIDE. These libraries were used without modification.
- **Adafruit_HMC5883_U** and **Adafruit_Sensor** are libraries developed by Adafruit for use with the HMC5883 compass. They are released under the BSD license and Apache License, Version 2.0 respectively.
- **numpy, unittest, csv, time, threading,serial** are modules included by default with python.
- **gps**

A.3 Tools

- GitHub (https://github.com/) and Git
- **PlatformIO and Atom** PlatformIO is an integrated development environment (IDE) that can be used to aid development of arduino code, and is a plugin for the Atom IDE.
- ArduinoIDE
- gedit (https://software.opensuse.org/package/gedit)
- python
- GPS Visualizer (http://www.gpsvisualizer.com/)
- Google Maps (full version) (https://www.google.co.uk/maps/)
- draw.io

A.4 section A.4: Initial Design Flow Diagram




arduino
Current heading
Desired heading
Current Speed
p(i)(d) heading and speed controllers
Motor speed
Rudder angle

Appendix B

Compass Test Results

Compass heading/^{o}	Actual heading/^{o}	Compass Error/^{o}
0	30	-30
20	50	-30
40	80	-40
60	111	-51
80	144	-64
100	181	-81
120	213	-93
140	248	-108
160	275	-115
180	297	-117
200	311	-111
220	320	-100
240	328	-88
260	335	-75
280	341	-61
300	346	-46
320	356	-36
330	0	-30
340	10	-30

B.1 section B.1: Simple Algorithm Tests

B.1.1 Castle grounds Test

Water csv file:

Object csv file:

B.1.2 Second Castle grounds Test

Water locations:

```
52.4139445,-4.0910125
52.4138627,-4.0911117
52.4133948,-4.0905726
52.4137874,-4.0901327
```

Waypoint Locations:

B.2 section B.2: Previous Code

The following code segments from the file pCont_Correction.ino were studied, and reproduced here, with permission. This code was studied to find the previous pins and calibration values for the servos as a starting point for my own calibration. This code is owned by Aled Davies, and was made to control previous systems on the same boat as used in this project. ***The full version will be included in the code submission for further reference.

```
15 Servo servo;  //this is the servo for turning
16 Servo esc;  //the esc obviously
17
18 #define speedesc 110  //this is the speed you will drive, choose what you want
19
20 SoftwareSerial ss(RXPin, TXPin);
21
22 void setup()
23 {
24   Wire.begin();
25   Serial.begin(115200); //braud rate for serial monitor
26   ss.begin(GPSBaud);
27   esc.attach(9);  //esc is attached to pin 9
28   servo.attach(8); // servo is attached to pin 8
29   esc.write(80); //this is the value that will arm the ESC (needs tinkering)
30
31 delay(1000);
32 }
```

```
184   if(turn==8)
185   {
186     if(abs(x4) > 2)
187     {
188       servo.write(90 - (abs(P * x4))); //turn right in proportion to the difference
189     }
190     else
191     {
192       servo.write(90); //set the rudders to straight
193     }
194     delay(60);
195     return;
196   }
197
198   if(turn==5)
199   {
200     if(abs(x4) > 2)
201     {
202       servo.write(90 + (abs(P * x4))); //turn right in proportion to the difference
203     }
204     else
205     {
206       servo.write(90); //set the rudders to straight
207     }
208     delay(60);
209     return;
210   }
211
212   if(turn==3);
213   {
214      servo.write(90);  //set the rudders to straight
215      delay(60);
216      return;
217   }
218 }
```

Bibliography

[1] R. Bates, "Travels with applied geophysics - dendrochronology glen affric, 2014." http://geophysicistatlarge.blogspot.co.uk/search/label/dendrochronology, 2014. Accessed 2017-02-08.

This blog talks about a previous use of the boats, under remote control, that are to be used in this project .

[2] S. Claxton, "Cost effective sondes for monitoring marine terminating outlet glaciers."

A discussion of marine probes to be released in dangerous waters around the edge of glaciers. , 2017.

[3] INNOC, "World robotic sailing championship, international robotic sailing conference." http://www.roboticsailing.org/, 2017. Accessed 2017-05-02

List of all World Robotic Sailing Championships and International Robotic Sailing Conferences.

[4] A. Unknown, "Sailbot international robotic sailing competition." http://sailbot.org/, 2017. Accessed 2017-05-02

The home page for the 2017 International Robotic Sailing Competition .

[5] ENSTA Bretagne, "Wrsc 2016 tracking." http://trackingwrsc.fe.up.pt/replay, 2015. Accessed 2017-05-02

Replays of the challenges at the World Robotic Sailing Championships.

[6] J. C. Alves, "Robotic sailing 2016," in *Proceedings of the 9th International Robotic Sailing Conference*, 2016.

Discussion of robotic sailing boats, particularly how they behave and navigate .

[7] AberSailbot, "abersailbot." https://github.com/abersailbot, 2017. Accessed 2017-04-30

The github pages for AberSailbot and their code for controlling their robotic sailing boats.

[8] L. Taylor, "boatd." https://github.com/boatd, 2017. Accessed 2017-04-30

The robotic sailing daemon used to control robotic sailing boats.

[9] R. Siegwart, I. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*. Intelligent robotics and autonomous agents, MIT Press, 2011.

This book talks about the different control systems that can be used .

[10] Colaboration, "Pid controller." https://en.wikipedia.org/wiki/PID_controller, 2017. Accessed 2017-04-30

Talks about PID controllers and the different ways of tuning them .

[11] Arduino, "float." `https://www.arduino.cc/en/reference/float`, Year unknown. Accessed 2017-04-28

Reference pages for the Arduino language. This talks about the range of values a float can take.