Bachelor Project - CAPTAIN

Appendix

Bachelor - Electronics Engineering
7. Semester project Fall 2017
Group: 17115

Aarhus University, School of Engineering
Supervisor : Michael Alrøe

19. December 2017

CAPTAIN

Nicolai K. Bonde
Studienr. 201405146
AU ID: au512366

Troels Ringbøl Brahe
Studienr. 20095221
AU ID: au283169

Abstract

English Abstract

Resumé

Dansk resume

Contents

1 Preface **3**

2 Introduction **3**

3 Requirements **3**
- 3.1 Use case 3 - Edit parameter profile 4
- 3.2 Use case 11 - Calculate coverage path 5
- 3.3 Use case 12 - Run coverage path 5
- 3.4 Use case 5 - Request diagnostics 5

4 Scope **5**

5 Method **6**

6 Analysis **6**

7 Architecture **6**
- 7.1 Hardware architecture 6
- 7.2 Software architecture 8

8 Design **12**

9 Implementation **12**
- 9.1 Hardware 13
  - 9.1.1 Mechanics 13
  - 9.1.2 Electronics 13
- 9.2 Software 14
  - 9.2.1 Programming languages 14
  - 9.2.2 Compiler 14
  - 9.2.3 Code 15

10 Test **15**

11 Results **15**

12 Discussion **15**

13 Conclusion **15**

14 Future work **15**

1 Preface

2 Introduction

3 Requirements

To get a sense of the system and what requirements the system should have, some mock-ups of a ui were created and can be found in the documentation in section 2.1 on page 7.

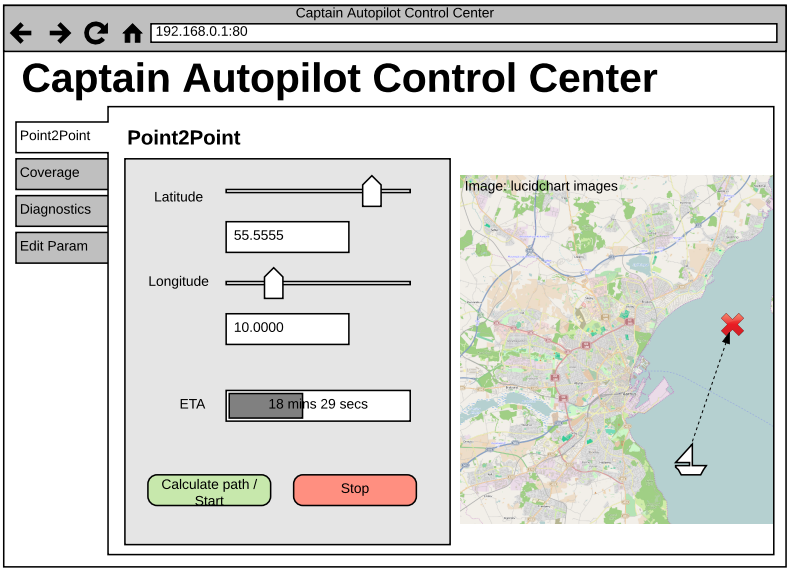

Figure 1: Mockup for the Point to point menu

An example of one of these mock ups is for describing a way point to navigate towards, can be seen in figure 1. It describes how the a user should interact with the system, and how the system should communicate to the user.

To describe the functionallity in further detail, a user case driven approach has been used. First of all the actors of the system has to be identified. In figure 1 is the use case diagram for the system, on the right are the actors that initiate a use case. On the left the other actors are.

Initiating actors or primary actors of the CAPTAIN system, are a technician, and a user. The technician is an actor who setups the system, and has a more in depth knowledge of the system then the user. The user could be anyone, since all of the complicated work should be handled by the system or the technician.

The usecase diagram on figure 2 also list 13 different use cases. A use case describe a way to use the system, in this system they mainly describe a button of function that can

Figure 2: Use case diagram

be started by the user. In this section a few use cases will be looked at, but for all the fully dressed use cases, one can have a look at the documentation section 2.3 on page 12.

3.1 Use case 3 - Edit parameter profile

One of the first things that was realized the system needed, was a way to describe parameters of the system. Parameters could be PID-loop terms, or the size of the boat, i fact anything that could be of use to the system. So to be able to save these parameters, the concepts of a parameter profile came to be. A parameters profile is essentially a list of any kind of parameters. Use case 3 - "Edit parameter profile", is the use case used to change the values of the of an already created parameter profile. It's important to note that it is the technician who initiates this use case, since know what exactly the values should be could require some deeper knowledge of the system.

3.2 Use case 11 - Calculate coverage path

There are a tonne of ways to navigate way points, first of one needs to decide on how to describe a way points. So for this system way points can be describe in two ways, either as a single point, or as a rectangle. For the rectangle the system should calculate a path that covers a the rectangle with lines that have a predefined distance between them. This use case should be initialized by the user, as a simple press on the user interface. In return the user interface should display the calculated path, so the user can tell if the path is what they wanted.

3.3 Use case 12 - Run coverage path

With a calculated path, ei. list of way points to follow, the boat should be able to follow these points. When the user presses a button label "Run" Use case 12 - "Run coverage path" is initiated, and it should not finish until the boat has reached the last way point of the list of way points. The boat should get though the way points using a control loop. While the boat is running a the estimated time en-route should be displayed along with the current position of the boat.

3.4 Use case 5 - Request diagnostics

At any point in time, it might be convenient for the user or the technician, to know how the boat is doing. In other words, getting the diagnostics data from the boat. diagnostics data might include GPS information, what position the rudder is set to and so forth. This use case can, as mentioned be initiated by either the user or the technician, by the press of a button in the user interface.

4 Scope

The scope of this project has been analyzed with the use of the MoSCoW method. This method is used to prioritize what should be worked on in the project. The method is separated into 4 levels of priority; **Must**, **Should**, **Could**, and **Won't**.

The following priorities have been chosen for this project:

Must
- Navigate way points from user input
- Be compatible with NMEA protocol GPS input
- Use GPS for localization
- Implement a PID control loop

Should
- Control thrusters in two-thruster catamaran
- Use a graphical user interface for user interaction
- Be able to change the PID parameters

Could
- Control wheel in outboard motor on boat
- Be generic enough to use with other engine types

Won't
- Use polygon-coverage for a specified area
- Avoid obstacles

5 Method

6 Analysis

7 Architecture

In this section the hardware and software architecture will be discussed. The architecture is the ground work that enables the design of the system. Taking the requirements and dividing it in to blocks, modules, etc.

For a more in depth explanations of the different subjects discussed in this section, one can have a look at the System architecture, section 5, on page 38 in the documentation.

7.1 Hardware architecture

To get a better understanding of the components what are need for this system. A block definition diagram or BDD was devised, as seen on figure 3.

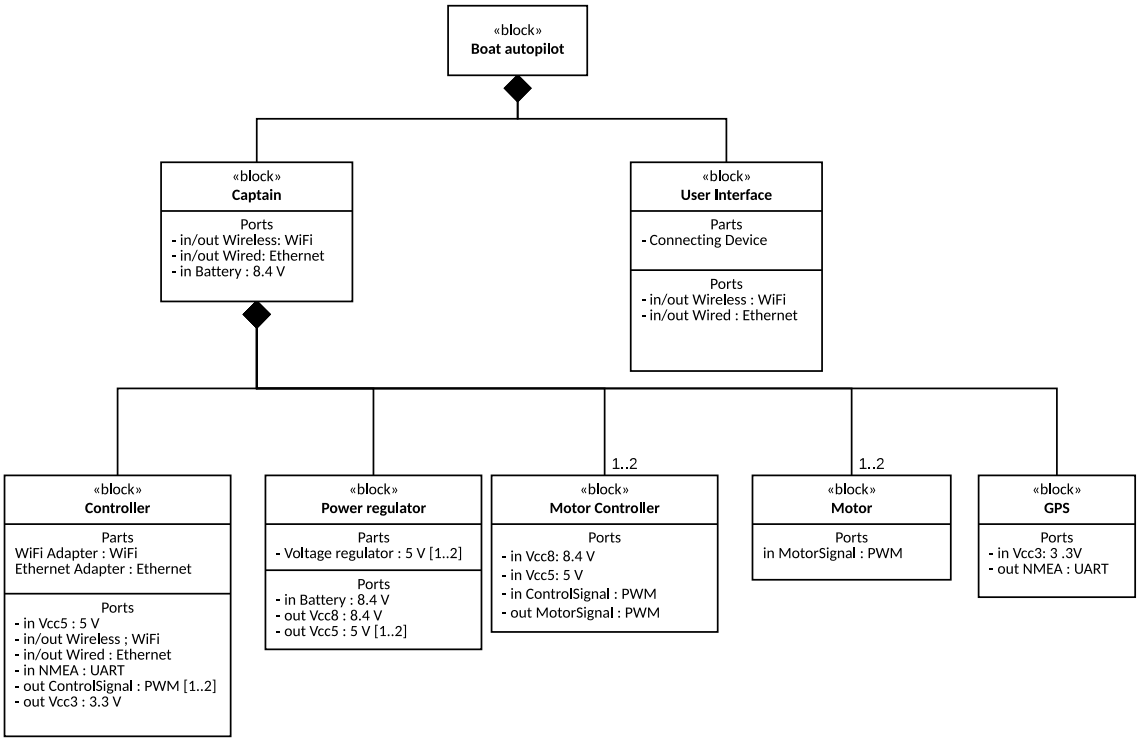

Figure 3: General block definition diagram (BDD)

The system can be broken down into two separate parts. One is the user interface, this is a clients personal computer. This personal computer needs to have either WiFi or an Ethernet port, this is so it can connect to the system. The system is the other part, and it has been named Captain.

Captain is the hardware platform for this project, and it also needs WiFi and an Ethernet port, so it can be connected to. Further more it needs an external battery to power it. Captain is also built up of subcomponents, or parts. these parts are; a GPS, 1 to 2 motors, 1 to 2 motor controllers, a power regulator, and a controller. The controller is the brain of the operation, it communicates with the user interface, and dictate what the motors should do, and it reads from the GPS receiver. The power regulator is used to regulate the battery voltage, so the controller, the motor, and the motor controller can use it. The motor controller is used to take the control signals from the controller, and drive the motor with them.

With the block now defined, an IBD or internal block diagram can be created, and seen in figure 4. This diagram describe how the different blocks of the BDD connect to each other, via the signals that are defined in the BDD as well. A full signal list and description can be found in the documentation in section 5.1.1.2 on page 40.

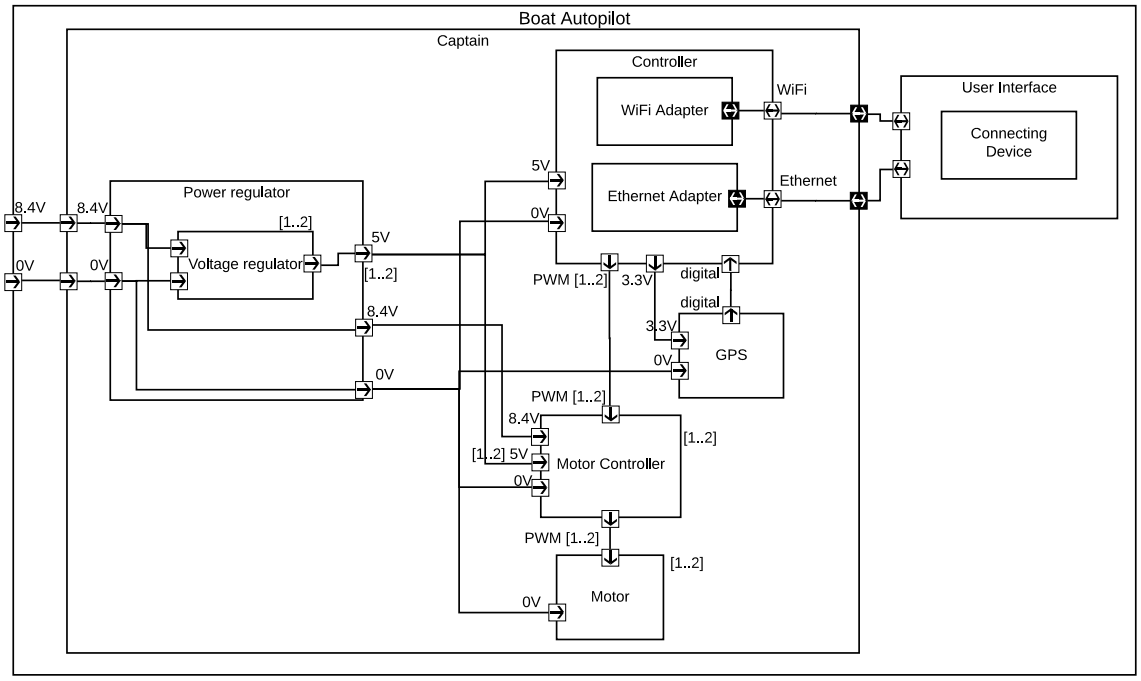

Figure 4: General internal block diagram (IBD)

7.2 Software architecture

The software architecture is described with a domain model, an application model and several sequence diagrams.

Lets start out by having a look at the domain model, it can be seen on figure 5. The domain model is used to describe the system should act to an actor interacting with it. In the domain model it can be seen how the user or technician can interact with the web interface, and how it then communicates to the controller. The controller acts on the motors, which in turn change the boats position. This that affects, what the GPS receiver reads, and this information is then passed on to the controller. The information

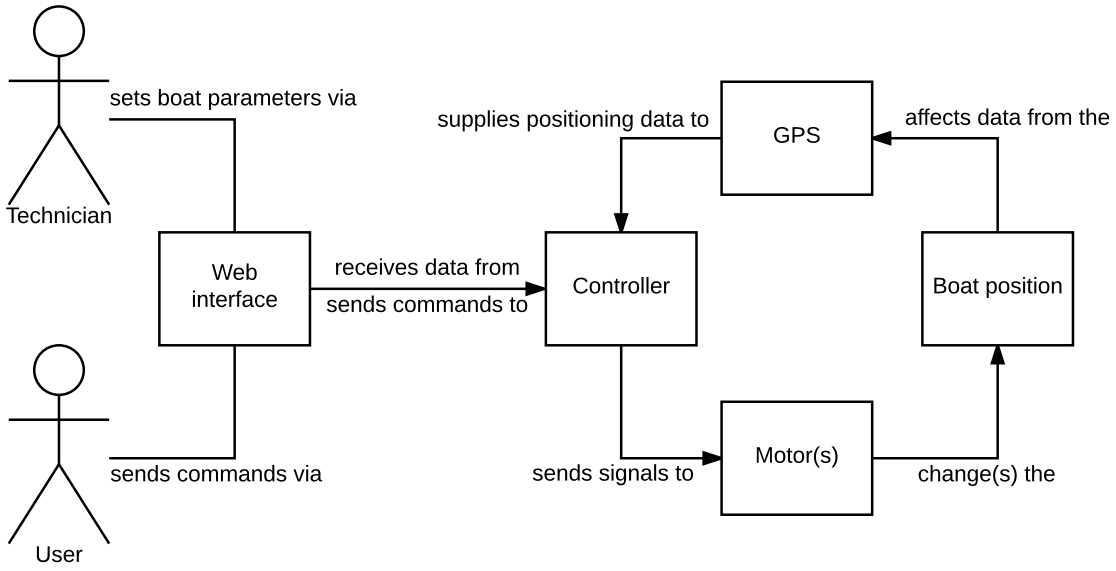

Figure 5: Domain model

With a domain model and use cases, an application model can be created, as seen in figure 6. It describes the functionality of the block from the domain model, it also classifies the blocks as either boundary, control or entity. A boundary block is something that interacts with the real world, a control block is the block that mediates functionality of boundary blocks and entity blocks. The entity block is a representation of information.

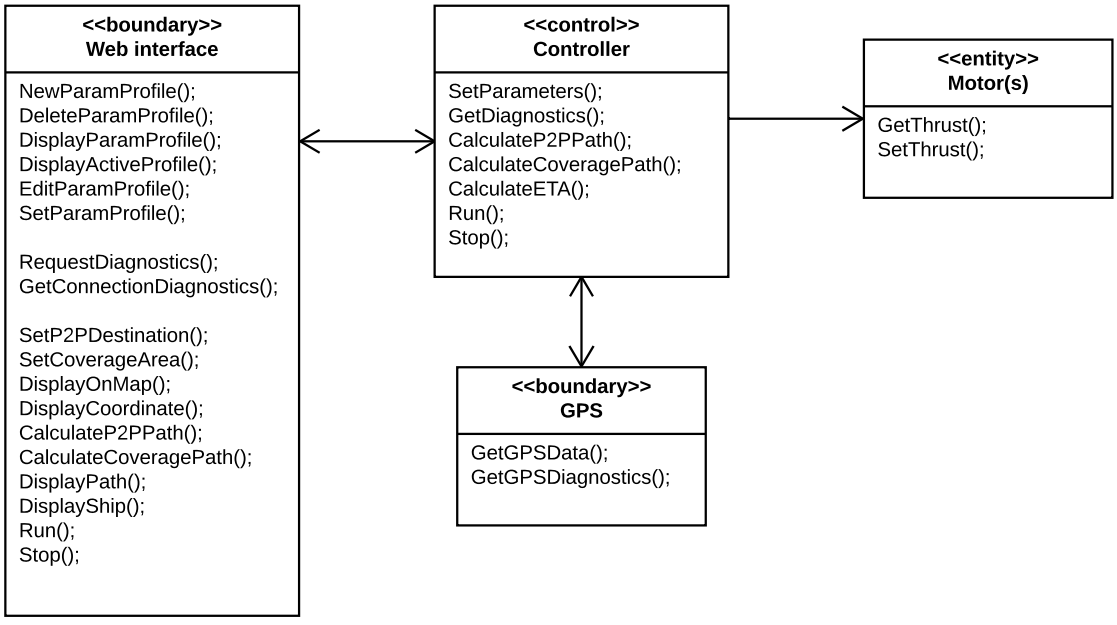

Figure 6: Application model

With the functionality of the blocks figured out in the application model, its time to look at sequence diagrams. There are a lot of sequence diagrams, so only a few interesting ones will be discussed in this report. For all the sequence diagrams, have a look in the documentation in section 5.2 on page 41.

The sequence diagrams follow the use cases, therefore lets have a look at the once that correspond to the once discussed in the requirements section, that is; use case 3 - "Edit parameter profile", use case 11 - "Calculate coverage path", use case 12 - "Run coverage path", and use case 5 - "Request diagnostics".

Figure 7 explains how a technician edits a parameter. First the parameter profile is displayed, then it is edited by the technician.

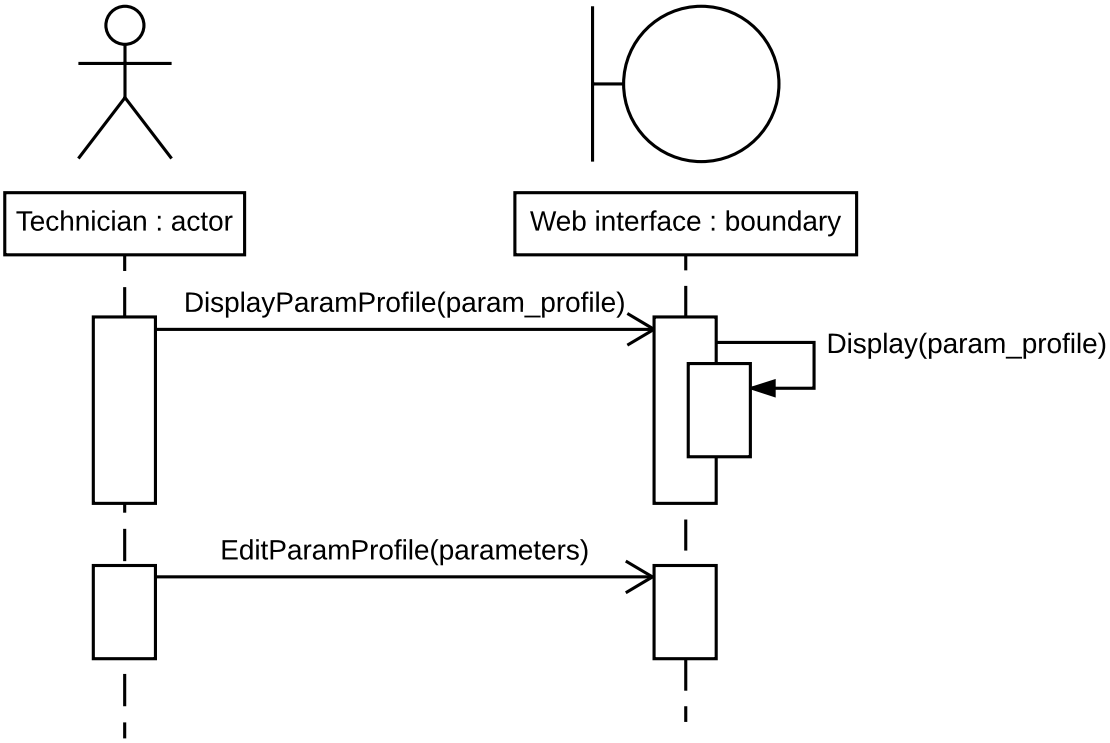

Figure 7: Sequence diagram for use case 3 - "Edit parameter profile"

Figure 8 is showing how a user tells the system how to calculate a coverage path. This is done thought he user interface, which tells the controller to calculate a path. The controller uses GPS data to calculate the path and then returns the path to web interface. Then the path is displayed.

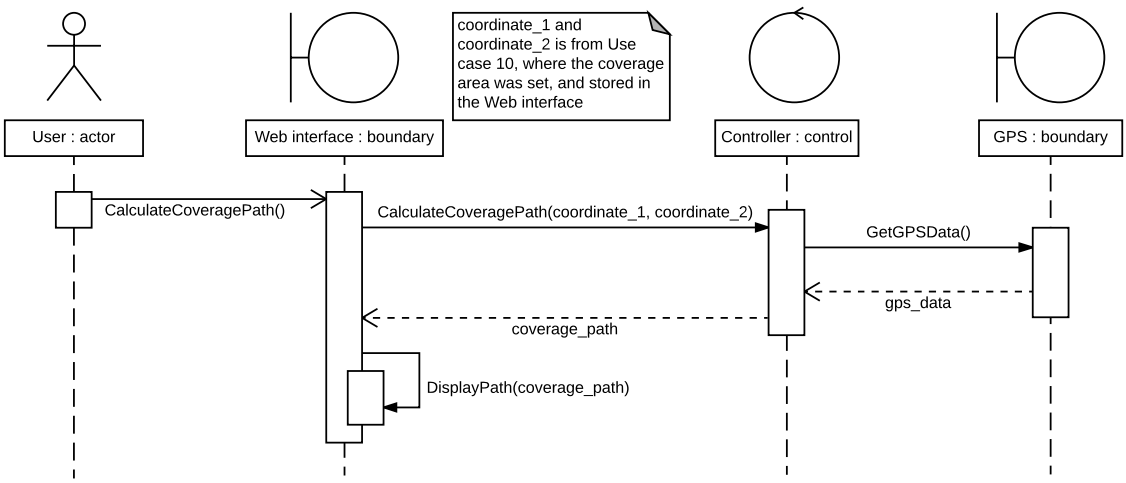

Figure 8: Sequence diagram for use case 11 - "Calculate coverage path"

Figure 9 illustrates what happens in use case 12. When the user tells the web interface to run it relays the message to the controller which in turn starts a loop. This loop gets GPS data and tells the motors what to do. It tells the web interface to display the boat position. The controller also calculates the estimated time en-route and the web interface displays it.

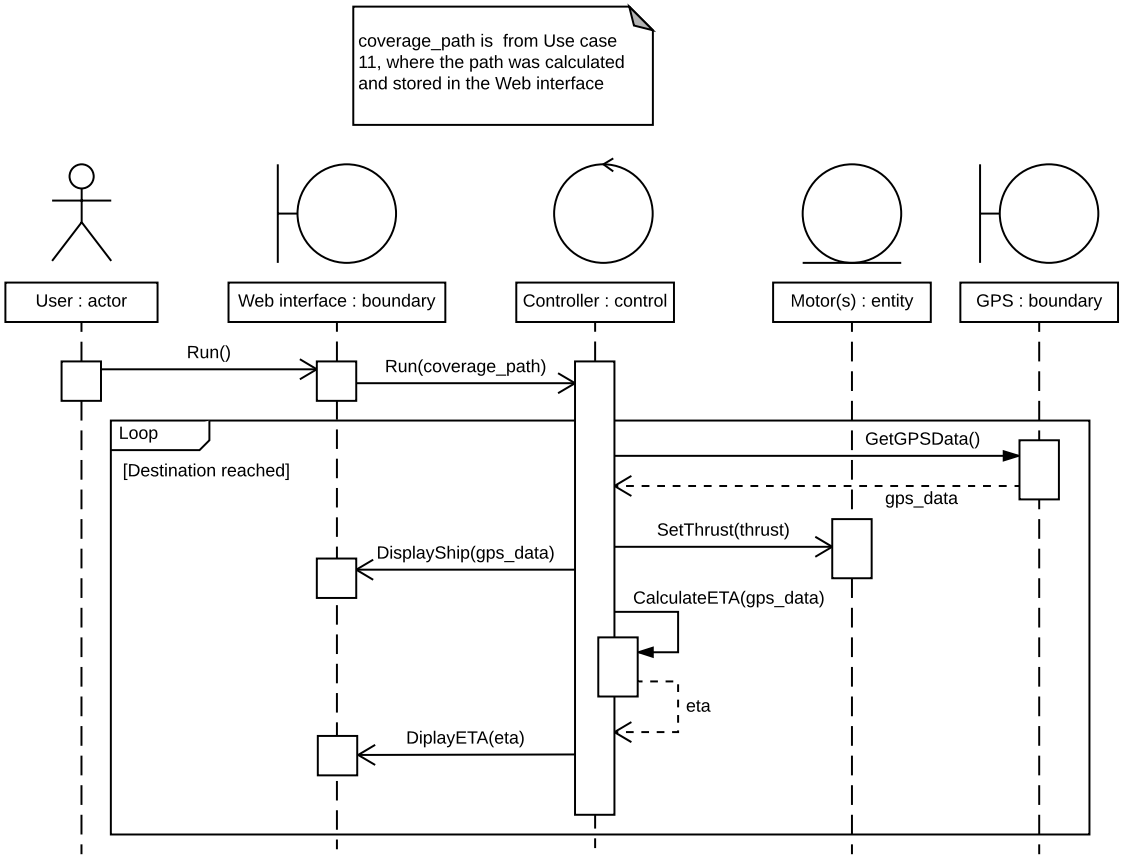

Figure 9: Sequence digram for use case 12 - "Run coverage path"

Lastly, figure 10 show how the system get and displays diagnostics data. When the

user requests diagnostics data, the web interface asks the controller for it. The controller responds by getting the current thrust of the motor and the GPS data along with the GPS diagnostics data. The controller also get connection diagnostics from the web interface, all of this is then displays on the web interface.

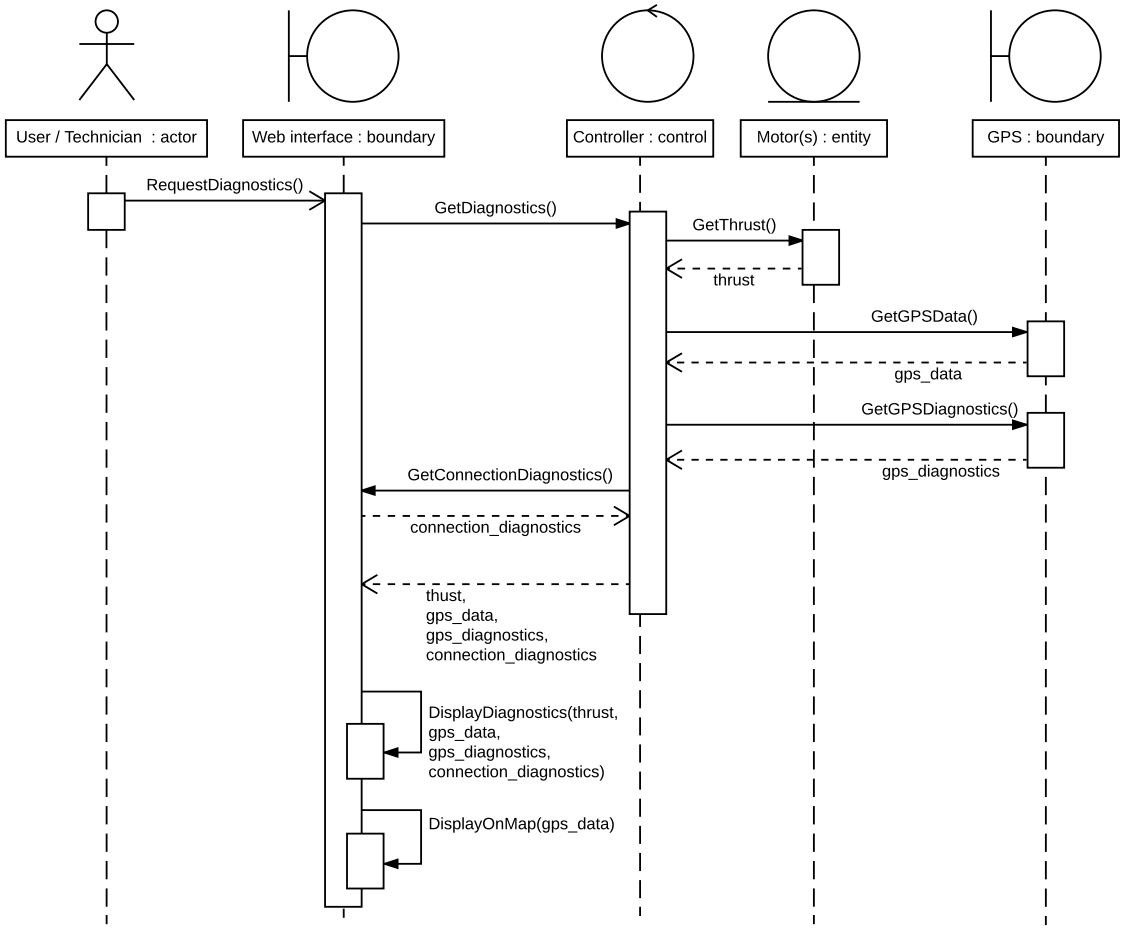

Figure 10: Sequence diagram for use case 5 - "Request diagnostics"

8 Design

9 Implementation

In this section the implementation of Captain system will be discussed. This includes mechanical and electronic hardware choices, as well as software choices. More detailed descriptions of every thing discussed in this section can be found in the documentation in section 6, on page 52.

9.1 Hardware

There are 2 types of hardware in this project, mechanical and electronic. The mechanics are the boat and rudder and such. The electronics are motor driver, power regulator and so forth.

9.1.1 Mechanics

The mechanics for this project, are the boat saint princess, as seen in figure 11. This boat was chosen simply because it was what was available, and since the project has been rather generic about the choice of hardware up to this point, it doesn't really matter. The boat has a motor in a so call D-drive inboard configuration[**motor_config**], which means that the motor is inside the boat and is connected to the propeller via a shaft that goes out through the hull of the boat. The propeller sits just in front of the rudder, which is controlled by a servo on the inside. The hull of the boat is a deep vee hull[**hull-types**], it is a planning mode hull, which makes wakes smaller then other hull types.

Figure 11: Saint Princess, a remote controlled yacht[**saint_princess**]

9.1.2 Electronics

All of the electrical components of the system were connected together on a perfboard. The perfboard is used to distribute the power from the battery to voltage regulators. The signals from the Raspberry pi are also routed through the perfboard, to the servo and the motor controller. All of the signals are routed through detachable wires, to make it easy to measure and rearrange in case some thing was wrong. This perfboard system can be seen in figure 12.

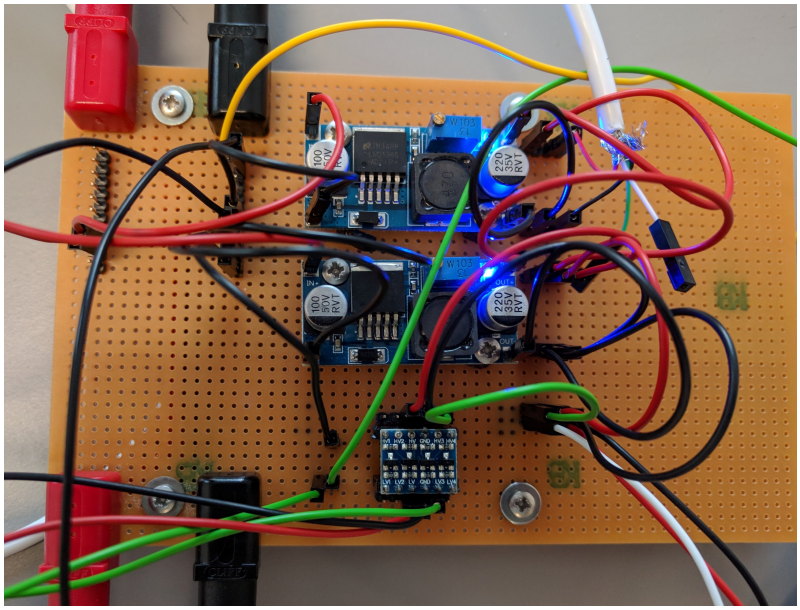

Figure 12: Perfboard, used to connect all components

9.2 Software

In this section, the choices of compilers and programming languages will be discussed. Also the concrete code implementation will be looked at, at a very high level.

9.2.1 Programming languages

This system is built up of 2 distinct parts, at least when discussing software. There is the website user interface, and then there is the controller or what could be thought of as the backend. The website user interface, is marked up in HTML and CSS with the help of a framework called Boostrap [**boostrap**]. In terms of code, javascript was used with AngularJS as a framework.

Boostrap was chosen simply because it makes it easier to create a website that look somewhat descent. AngularJS was chosen because it makes data binding to the html page very simple, and because it supposedly is a widely used javascript framework.

For the other part of the system, the controller. C++ was the language of choice, this simply because a it is what is familiar. C++ has the advantage of being a very fast. But then the draw back of being a rather low abstraction.

9.2.2 Compiler

For the website there is no real compiler since javascript is an interpreted. But the server that runs the website is a nodejs server. Nodejs is runtime built on the Chrome V8 javascript engine. The client browser used to display the website was a Chrome web browser which is also built on the Chrome V8 javascript engine.

The controller written in C++ and is therefore compiled, the compiler used while developing the system was the was the VS 2017 C++ Build tools. The compiler used

on the Raspberry pi is GCC 6.3.0 for raspbian. They are both C++11 compatible which makes some things easier while developing.

9.2.3 Code

The code

10 Test

11 Results

12 Discussion

13 Conclusion

14 Future work

There are a lot of things that could be better in the Captain project, and there are also some additions that would just make for a better product.

One thing that was put under won't in the MoSCoW is polygon coverage, this was simply do to the fact that polygon coverage is a very large subject. Usually when one wants to cover an are it is not a square. Even then generally its not square with the longitude and latitude lines, like the system is limited to at this time.

Another thing that would be a nice to have is a way to navigate a none straight point to point path, if maybe a user wants to cover a river, this would be very impractical with the current point to point system.

The idea of saving an already calculated path, so it can be run in the exact same way at a later time. This would make it possible to compare data collected on separate occasions and eliminate the position differences as a factor.

A thing that might not be optimal in the system at the moment is the communication between the website and the controller. It is handled by reading and writing to files. This was done for simplicity, but it might be a better idea to make the controller host a http server so it can handle GET and POST requests it self.

With the current implementation the controller only supports one boat type. But the software design is built up of rather generic interface so it should be possible in the future to include a variety of boat types.

One thing that we were intending to get done for the project was an echo sounding

device that would measure the depth of the sea bed below the boat. But we didn't end up getting around to it.

The system could be classified as an mobile autonomous robot, but at this stage it does not have obstacle avoidance. This could be used to make sure that the boat is not going to run ashore, hit a bouy or even another boat.

If the system was ever to become anything other then a scale test bed, then it would both need a internet connection vis GSM. It would also probably need an RTK GPS receiver. An real time kinematic GPS receiver is sub centimeter precise and some can have interpolated position every 1/10 of a second, which could make the autopilot much more stable.

Lastly on the topic of sensors, it would probably be a good idea to have some sort of battery monitoring system, if the system is going to be battery powered.