Design and Implementation of Software for the ROV Neptunus

Jostein Munz

Master of Science in Cybernetics and Robotics
Submission date: July 2015
Supervisor: Sverre Hendseth, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU

Master thesis

Design and Implementation of Software for the ROV Neptunus

By:
Jostein Munz

Supervisor:
Sverre Hendseth

Co-supervisors:
Roger Skjetne
Andreas Reason Dahl

Department of Engineering Cybernetics
Norwegian University of Science and Technology

July 13, 2015

Scope of work

Background

The Remotely Operated Vehicle(ROV) Neptunus is a small, low cost ROV to be developed for academic or recreational use. The scope of this master thesis is to design and develop the software for this ROV. The functional requirements for the ROV are:

- Common ROV control and payload functionality.
- A user interface consisting of an operator interface and a developer interface.
- Two HMI platforms; pc and tablet. The pc HMI shall have support for joystick control.
- Remote monitor and control functionality. Methods for taking and assigning command including fail-safe.
- A built-in dynamic ROV simulator mode.
- Feedback control and state estimation functionality.
- Interface to the Oculus Rift system.

Work description

1. Perform hardware and software design for ROV Neptunus. Expand the functional requirements in detail. Review and conclude on relevant software frameworks.
2. Implement the user interface with priority on the developer interface.
3. Implement minimum required functionality to control the ROV, then expand the implementation scope as time allows.
4. Documentation of the system and relevant software tools shall be inlcuded in the report to facilitate further development. The ease of further development shall also be considered in decisions for the software and hardware solution chosen.
5. Develop and perform tests based on functional requirements and known issues.

Abstract

There has been an increased interest in researching and developing technical solutions for underwater vehicles the last years. In most cases, underwater vehicles are expensive and not affordable to private individuals. Development of low cost ROVs has lately been increasing supported by an active Do-It-Yourself (DIY) community. The OpenROV project has contributed to this community by producing cheap ROVs with open source software, encouraging users to develop their own plugins. Development of a lost cost ROV were carried out fall 2014, as a project thesis [1]. The result was a low-cost ROV prototype, called Neptunus using OpenROV computer hardware and software.

The growth of the internet over the last decades has motivated the development of web applications. The communication between web servers and clients has been dominated by the use of the Hyper Text Transfer Protocol (HTTP) and the request-response pattern. In the recent years, developments have been made to facilitate bi-directional communication protocols. The WebSockets protocol provides this, as an alternative to the HTTP protocol. Node.js is a runtime environment that uses the Google V8 JavaScript engine to run Node.js web servers. The use of the WebSockets protocol is facilitated in Node.js web servers, making it a desirable choice for real-time applications.

Design of hardware and software solutions for the ROV Neptunus was performed. This includes review of frameworks and software tools. A Node.js web server was implemented on a BeagleBone computer running Linux Ubuntu. The Node.js specific framework Express.io was used. A user interface consisting of an operator interface and a developer interface was implemented in JavaScript. The operator interface contains methods for controlling the ROV with the use of keyboard, touchscreen and joystick. The operator interface contains functionality vital in development such as logging and plotting.

A control module was implemented to control the ROV with either motion control or manual control. This included a thrust allocation procedure for the motion control mode. A camera module for streaming of the video was implemented. The mjpg-streamer application was used to stream the video from the camera to the client. A simulator module was implemented to facilitate parallel development. Remote monitoring and control over WiFi was implemented. This included methods for taking and giving control, as only one operator can be in control at any time. All the implemented functionality was tested. Documentation of the system and relevant software tools for development was included.

Preface

This thesis was written at NTNU spring of 2015. Testing was performed at the Marine Cybernetics lab at NTNU, campus Tyholt.

I wish to thank my supervisor, Sverre Hendseth and my co-supervisors Roger Skjetne and Andreas Reason Dahl for the help they have provided. I also wish to thank Mauro Candeloro and Martin Ludvigsen for their involvement in the project. Last, but not least, I want to thank everyone who helped me at Marine Cybernetics, the company, in particular Mikhail Lindblad.

Contents

List of Figures

Acronyms

ROV	Remotely Operated Vehicle
HMI	Human Machine Interface
DIY	Do-It-Yourself
DNV GL	Det Norske Veritas Germanische Lloyd
DOF	Degrees Of Freedom
DP	Dynamic Positioning
IMU	Inertial Measurement Unit
NED	North, East, Down
HTTP	Hyper Text Transfer Protocol
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet
CGI	Common Gateway Interface
TCP	Transmission Control Protocol
IP	Internet Protocol
I/O	Input / Output
API	Application Programming Interfaces
UART	Universal Asynchronous Transmitter/Reciever
ROS	Robotics Operating System

1 Introduction

1.1 Background

There has been an increased interest in researching and developing technical solutions for underwater vehicles the last years. Sophisticated technology has been developed to perform complex underwater operations with high precision, such as seabed mapping, online underwater monitoring, subsea installations, and maintenance on pipes. In most cases, underwater vehicles are expensive and not affordable to private individuals. Development of low cost ROVs has lately been increasing supported by an active DIY community. The OpenROV project has contributed to this community by producing cheap ROVs with open source software, encouraging users to develop their own plugins. Development of a lost cost ROV were carried out fall 2014, as a project thesis [1]. The result was a low-cost ROV prototype, called Neptunus. The OpenROV computer hardware and software was used for the ROV Neptunus during this development.

The growth of the internet over the last decades has motivated the development of web applications. The communication between web servers and clients has been dominated by the use of the Hyper Text Transfer Protocol (HTTP) and the request-response pattern. In the recent years, developments have been made to facilitate bi-directional communication protocols. The WebSockets protocol provides this, as an alternative to the HTTP protocol. Node.js is a runtime environment that uses the Google V8 JavaScript engine to run Node.js web servers. The use of the WebSockets protocol is facilitated in Node.js web servers, making it a desirable choice for real-time applications.

1.2 Thesis organization

The use of Node.js as web server solution is discussed and concluded in chapter 4.1. This is chronologically after Node.js has been presented as background material in chapter 2.6. This is purposely done to separate the background chapters from the result chapters. The same applies to the choice of hardware and frameworks.

The product specifications were incrementally developed. In the thesis, the final version is presented. The exception to this is the testing chapter. This describes the incremental development process.

The terms ”Internal acceptance test”, ”factory acceptance test” and ”customer acceptance test” are used in the testing chapter. These expressions are common in the industry, and they have a particular meaning. The usage in this thesis is not meant to reflect the industry usage wholly, but the terms were commonly used during the project work, so they were kept in the report.

In the user interface, pictures of the camera image is included. This was intended to be in-water images, but the lab pool was unexpectedly closed for repair.

2 Web programming

2.1 JavaScript

JavaScript is the scripting language used for the behavior of a web site. It is an interpreted language in contrast to compiled languages like C and Java. JavaScript is written and saved as plain text files of the .js type. JavaScript is a weakly typed and dynamic language. Arrays are dynamic in JavaScript and the type of variables does not have to be set. The return statement is optional. Using all the available input parameters to function calls is also optional. JavaScript supports object oriented programming, but all objects are defined as functions rather than classes.

2.2 CSS

Cascading style sheet, or CSS for short, is used to style web sites. It is written into .css files in plain text in the same manner as JavaScript files. CSS code can be used to control colors, fonts, size and alignment of elements among other things. Elements of the website can be grouped into classes. The CSS styling can then be applied to individual classes to avoid manually styling all elements.

2.3 HTML

Hyper Text Markup Language or HTML for short is used to make the content of a webpage. Examples include tables, paragraphs, text, images and headings. Other relevant items used are:

Radio buttons is a type of control element. It is a collection of several buttons where only one can be chosen. Radio buttons have visual appearance of dots.

Form input is used to submit texts. This can for example be a user name, information or input to a search query.

A Scrollpane is a pane that provides bars for scrolling when the content of the pane exceed the layout boundaries for the pane.

Styling and scripting can be done within HTML code, but most web servers have separate .js and .css files and link to them in the HTML code.

2.4 Server and client

Web servers are mainly used to for serving the world wide web. The terms *web server* and *client* can be illustrated by explaining what happens when a website is accessed. A user opens a browser and types the website URL into the URL bar. An Hypertext Transfer Protocol (HTTP) request is sent to server. Server will serve (send) HTML to the client. Any JavaScript files or CSS files that the HTML invokes are also sent. The client web browser will read, interpret and execute these files. When the browser has finished this work, the client can view the web site.

The three most used web servers are Apache, Microsoft IIS and nginx [2]. These can not do much alone, so they are used together with other server side programs such as scripts and databases. This is referred to as the server stack. Examples of common server stacks are LAMP (Linux, Apache, mySQL and PHP) and Microsoft IIS in combination with Active Server Pages (ASP) and Microsoft SQL Server running on the Windows OS. A common way of connecting the server to programs used server side is through a Common Gateway Interface (CGI). Common scripting languages for web servers are PHP, python, Perl and ASP. Ruby on Rails and Django are common frameworks used in web applications, written in Ruby and Python, respectively.

Communication between a server and a client is mostly done with the HTTP protocol. This is a request-response based system. The client posts a request to the server and the server responds. The request must contain a request method. The request methods available are: GET, HEAD, POST, PUT, DELTE, TRACE, OPTIONS, CONNECT and PATCH. The server responds with a HTTP status code, and if the request is successful, the content requested. The HTTP status codes are typically a three digit code, where the first digit represent the type. 1xx is for Information, 2xx for Success, 3xx for Redirection, 4xx for Client Error and 5xx for Server Error. Common status codes include; 200: Continue, 404: Not Found and 500 Internal Server Error. The HTTP protocol also includes a header section with HTTP header fields. Examples of such fields are: ”Connection: keep-alive”, ”Accept-Charset: utf-8” and ”Content-Length: 429”.

2.5 WebSockets

WebSockets is an alternative protocol to HTTP for communication. It facilitates bidirectional communication between the client and the server without the need for an client request. It is build on Transmission Control Protocol and Internet protocol (TCP/IP). WebSockets does not share the use of HTTP status code and header fields. This leads to less traffic on the connection.

2.6 Node.js

Node.js, or Node for short, is an open source runtime environment that can be used to create web servers. Node in itself was written in C, C++ and JavaScript. Node applications are written in JavaScript or other languages that compile to JavaScript such as CoffeeScript or Microsoft TypeScript [3]. The Google V8 JavaScript engine is used to interpret the code in Node applications. The use of the V8 engine for Node applications can be compared to how the Java Virtual Machine (JVM) is used for Java. Node facilitates the use of the WebSockets protocol in addition to HTTP, creating a bi-directional connection between server and client.

Node can in itself be a fully functional server stack. This is in contrast to the traditional server stacks mentioned in chapter 2.4. According to [4], Node does not work well with relational databases such as SQL databases. MongoDB is a document-oriented database that is commonly used in conjunction with Node.

Web servers have typically dedicated one OS process or thread for every client connection. Node instead runs on a single thread with an event loop. This reduces connection overheading. A Node applications serving a million connections simultaneously was demonstrated in [5]. Using the common method of one thread per connection, [4] estimates a cap of 4000 connections with a similar setup. This solution however comes with the drawback that CPU intensive operations done in the web server can choke traffic for all the connections on the server. In [6] a Node web server was compared to a Apache with Java EE. Node was 20% faster.

Node comes with a low level I/O API utilizing non-blocking events and callbacks. Anonymous functions are also widely used in this API. This is illustrated in listing 1, used to print the IP of clients upon connection and disconnection. This together with the two way communication between server and client makes Node a popular choice for real time systems. Since Node runs on a single thread, one must be careful not to get exceptions bubbled to the top loop crashing everything. Error handling in Node is preferably done by returning errors to calling functions. This is in contrast to throwing exceptions which is popular in for example Java.

A Node project has a main class, typically named app.js, similar to the main function in C. The project is run by a terminal command "node app.js". Linking in Node is done in runtime as it is interpreted and not compiled. The "require()" statement is used to import other Node scripts and core Node modules.

Listing 1: Listening for client disconnect

```
app.io.on("connection", function(socket){
    var ip = socket.handshake.address.address;
    console.log("Client connected with ip:" + ip);
    socket.on("disconnect", function(){
        console.log("Client disconnected with ip:" + ip);
    });
});
```

Node has a packet management system called npm. Anyone can publish npm modules and there exist more than 150 000 publicly available modules.

One of the most popular npm packages is Express. It is a framework for Node that according to [4] "is the de-facto standard for the majority of Node.js applications today". It provides, among other things, templates and tools to handle routing, error handling and static serving. "Hello world" for an Express project, taken from the Express' web site [7], is shown in listing 2. The server will respond with "Hello world!" for requests to the root URL(/) or route. For every other path, it will respond with the HTTP status code "404 Not found". The Express package can also be used to generate a more comprehensive server with boilerplate code for error handling, logging, routing and more. This is done from the command line.

Listing 2: Hello world!

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

var server = app.listen(3000, function () {
  var host = server.address().address;
  var port = server.address().port;

  console.log("Example app listening at
                localhost:3000");
});
```

Socket.io is another widely used npm package. It is a library for bidirectional communication between a server and a client using the WebSockets protocol. It also provides both a client and server API. These APIs are semantically similar, but the server side contains Node code, while the client is plain JavaScript. Events can be emitted and listened to by both the server and the client.Listing 3 and 4 shows how the server and client listens to and emit events. The server receives a req object that contains some connection information in addition to the data sent from the client. The client however, receives the data that is sent directly.

Using socket.io it is also possible to group together clients in a room, and choose to broadcast a message to a given room from the server.

Listing 3: Server listen and emit

```
io.on("clientEvent", function(req){
  console.log("Recieved data: " + req.data);
});

io.emit("serverEvent", serverData);
```

Listing 4: Client listen and emit

```
io.on("serverEvent", function(data){
  console.log("Recieved data: " + data);
});

io.emit("clientEvent", clientData);
```

The express framework and the socket.io library have been combined to a single framework named express.io. In listing 1 Express.io is used. The app object is associated with express and the io object is assosicated with socket.io.

3 Remotely Operated Underwater Vehicle

Remotely operated underwater vehicles, commonly known as ROVs, are unmanned vehicles operating with a tether. The tether, commonly referred to as an umbilical cord, maintains a data connection between a topside computer and the ROV. The tether also provide electrical power in many cases. Batteries are occationally used in small ROVs instead. Figure 3.1 shows the NTNU ROV Minerva.

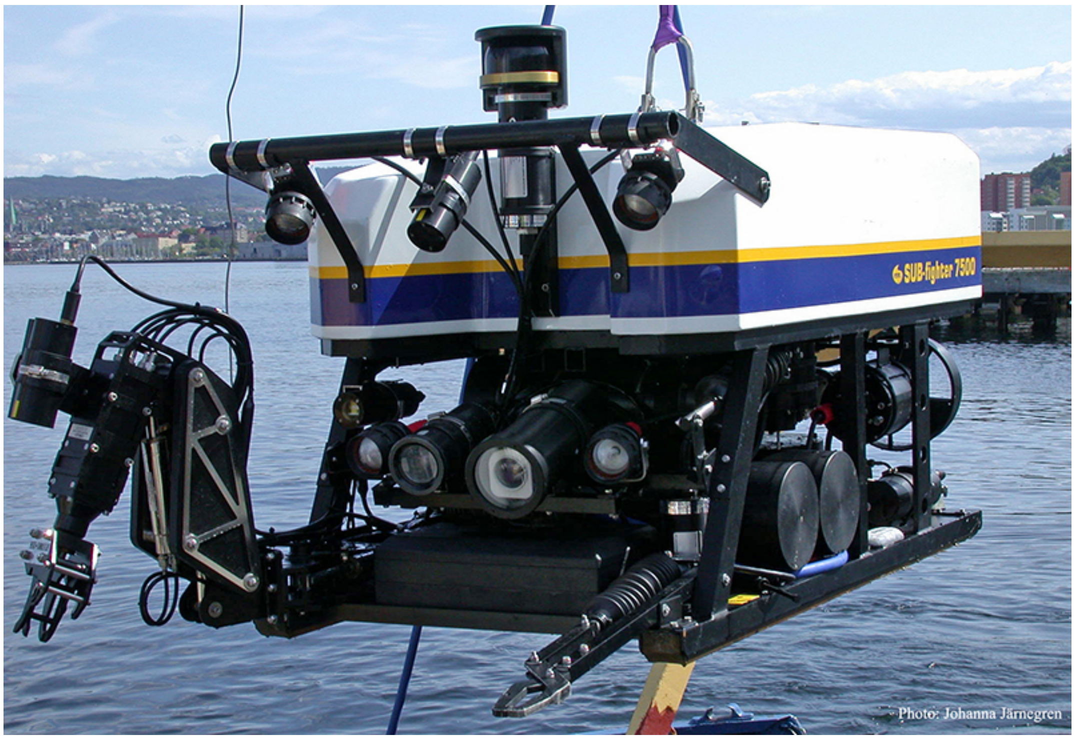


Figure 3.1: The ROV Minerva. Courtesy of NTNU

ROVs were initially developed mostly by the US military in the 1960s for recovery missions [8]. In the 1970s, development of ROVs for use in the offshore oil and gas industry started. This development has been ongoing since. ROVs have also been developed for detection and elimination of underwater mines. Research, academics, and marine agriculture are among other areas of use. Typical sensors and tools for industrial scale ROVs include IMU, acoustic positioning system, depth sensor, magnetometer, camera and robotic manipulators.

ROVs have traditionally been too expensive for recreational use. However, the last decade, a do-it-yourself (DIY) community around recreational use of ROVs have been growing. It is reasonable to assume that the OpenROV project has been a considerable driving force behind this. OpenROV is a small ROV, illustrated in figure 3.2, at the cost around 900$ unassembled or 1500$ fully essembled. It comes with open source software, and the OpenROV forums have around 4000 registered users. Student ROV competitions has been held since 2001 by The Marine Advanced Technology Education Center (MATE) [9].

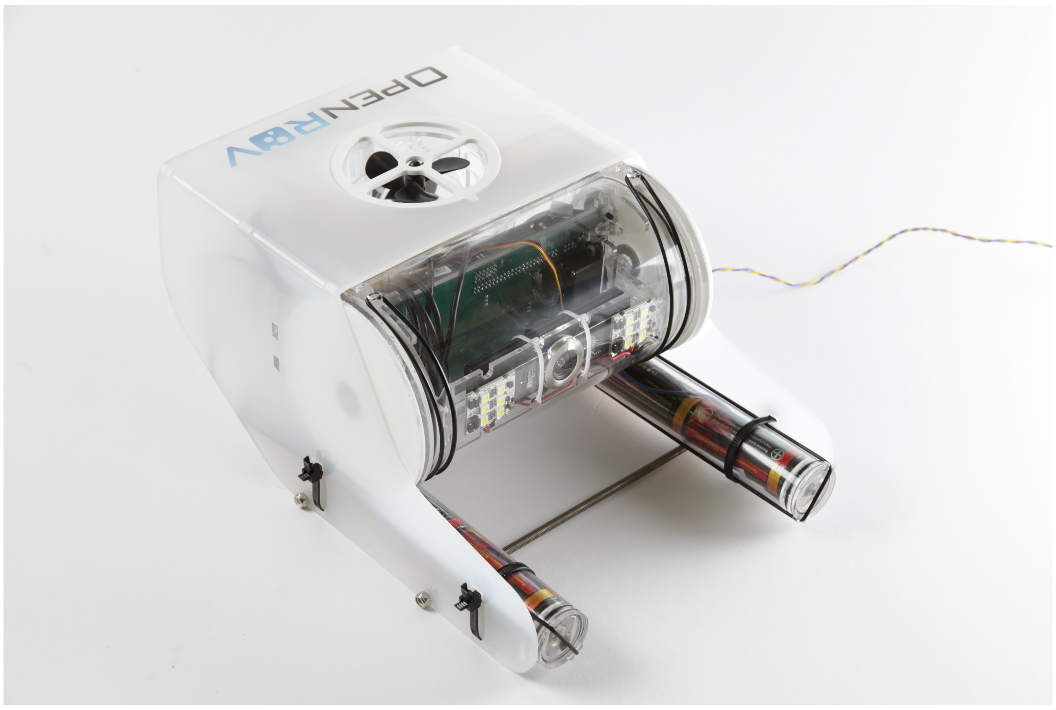


Figure 3.2: The ROV OpenROV

3.1 Reference frames

Two frames of reference used in marine craft navigation is the North East Down (NED) and the body fixed (BODY) reference frames.

The NED frame of reference is a plane tangential to the earth's surface at the location of the craft. The x axis points towards true north, y axis east and z down, perpendicular to x and y. Navigation in the NED reference frame is commonly referred to as flat earth navigation. The NED reference frame is not an inertial frame, but for most marine crafts it can assumed to be.

The BODY reference frame moves together with the craft. The BODY frame of reference can be used to define six Degrees Of Freedom (DOF) for a craft; surge, sway, heave, roll, pitch and yaw. Motion in the surge, sway and heave directions are forward/backward, sideways and up/down motion. These three motions describe the linear velocity of the BODY frame with respect to the NED frame. The x axis of the BODY frame is positive forwards, the y axis is positive to the starboard side of a craft and the z axis is positive downward. Roll, pitch and yaw defines rotations about the BODY axes. p, q and r denoted the angular velocity of the BODY frame with respect to the NED frame.

3.2 Kinematics

The general marine craft equations of motion in 6-DOF according to [10]:

$$\dot{\eta} = J_{\Theta}(\eta)\nu \tag{3.1}$$

$$M\dot{\nu} + C(\nu)\nu + D(\nu)\nu + g(\eta) + g_0 = \tau + \tau_{\text{wind}} + \tau_{\text{wave}} \tag{3.2}$$

where η is position and orientation (euler angles) in the NED frame. ν is linear and angular velocity in the BODY frame. R is the rotation matrix from BODY to NED. T is the transformation matrix that relates the derivative of the euler angles to the BODY angular velocity. τ_{wind} and τ_{wave} are typically negligible below 10m.

$$\eta = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^{\top} \tag{3.3}$$

$$\nu = \begin{bmatrix} u & v & w & p & q & r \end{bmatrix}^{\top} \tag{3.4}$$

$$\tau = \begin{bmatrix} X & Y & Z & K & M & N \end{bmatrix}^{\top} \tag{3.5}$$

$$J = \begin{bmatrix} R(\Theta) & 0 \\ 0 & T_{\Theta}(\Theta) \end{bmatrix} \tag{3.6}$$

3.3 OpenROV

The front of OpenROV was illustrated in figure 3.2. The backside of the OpenROV is illustrated in figure 3.3. The OpenROV comes with a BeagleBone show in figure 3.4 with Linux Ubuntu, and an Arduino show in figure 3.5.

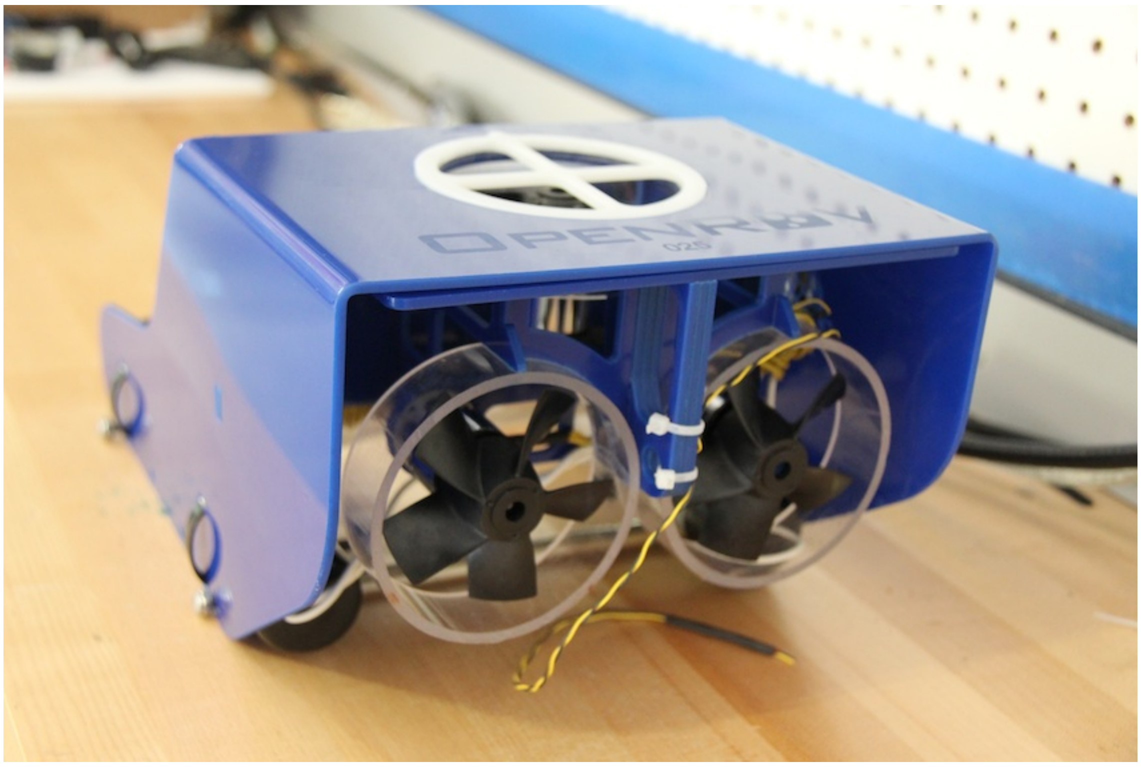


Figure 3.3: Backside of the ROV OpenROV

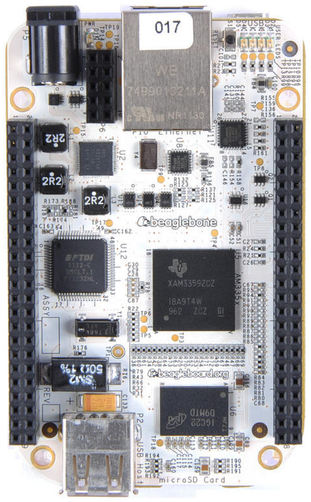


Figure 3.4: BeagleBone

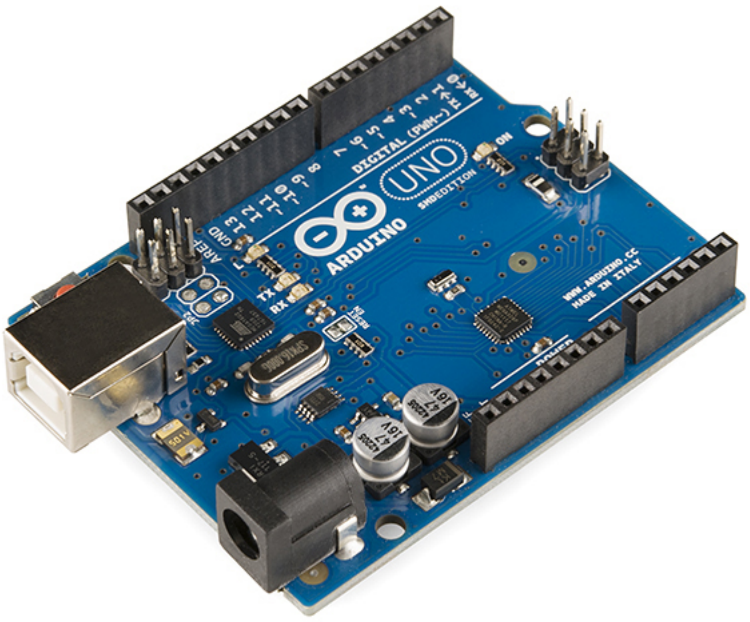


Figure 3.5: Arduino

The OpenROV also comes with three motors, six batteries, three speed controllers, two HomePlug adapters, one servo for camera tilt, a Genious F100 HD webcam, lights, a laser, an IMU and a depth sensor. The HomePlug adapter is used to connect ethernet to twisted pair. This is done to reduce umbilical size.

OpenROV runs a Node web server on the BeagleBone. Communication between BeagleBone and Arduino over serial port using an universal asynchronous reciever/transmitter (UART). The Arduino is responsible for power management and it contains drivers for the motors, lights, laser, servo and sensors. These are written in C++. The webcam is connected to the BeagleBone through USB.

3.4 Neptunus

The Neptunus ROV project was started fall of 2014 by a group of students at NTNU. Their work is documented in [1]. The main goal of the project was to test an OpenROV, and design a new ROV with the same cost range as OpenROV with experiences from the testing.

Weaknesses in the OpenROV design according to [1] based on the testing:

- Hydrodynamic inefficient design.
- Inefficient thruster build leading to substantial thruster loss, especially in vertical direction.
- Undesired pitch motion when moving in the surge or heave direction. This was concluded due to the placement of the thrusters. Resultantly hard to maintain constant depth when moving.
- Unstable communication. This was concluded to be caused by the HomePlug adapters and the umbilical.
- Sub-optimal umbilical attachment point. It was speculated that this contributed to the undesired pitch motion.
- Water leakage during sea trials
- Occational video dropout during rapid change in thrust command. It was speculated that this was due to transient voltage drop.

The following contributions to the Neptunus ROV were made by [1]:

Design and construction of a new hull. Hydrodynamic efficiency and stability was addressed in detail.

New thruster configuration. Increased spacing between the propellers for reduction in thrust loss in both vertical and horizontal direction. The propellars were also carefully aligned with the vessel axis to avoid undesired pitch moments when moving the surge or heave direction.

Power setup was changed. A 12V dc transformer connected to a regular 230V outlet on topside was added. Power setup on the ROV was modified to accommodate this. The batteries from OpenROV was kept.

The umbilical cord was modified to incorporate wiring for the 12V power source. The OpenROV HomePlug adapters were removed, and ethernet was used all the way from topside to ROV. The umbilical attachment point on the ROV was changed.

Design of Oculus Rift as a input method to control the ROV was performed.

The software and computer hardware from the OpenROV was not modified in [1]. The cylinder holding the computer hardware and the camera from ROV was kept,

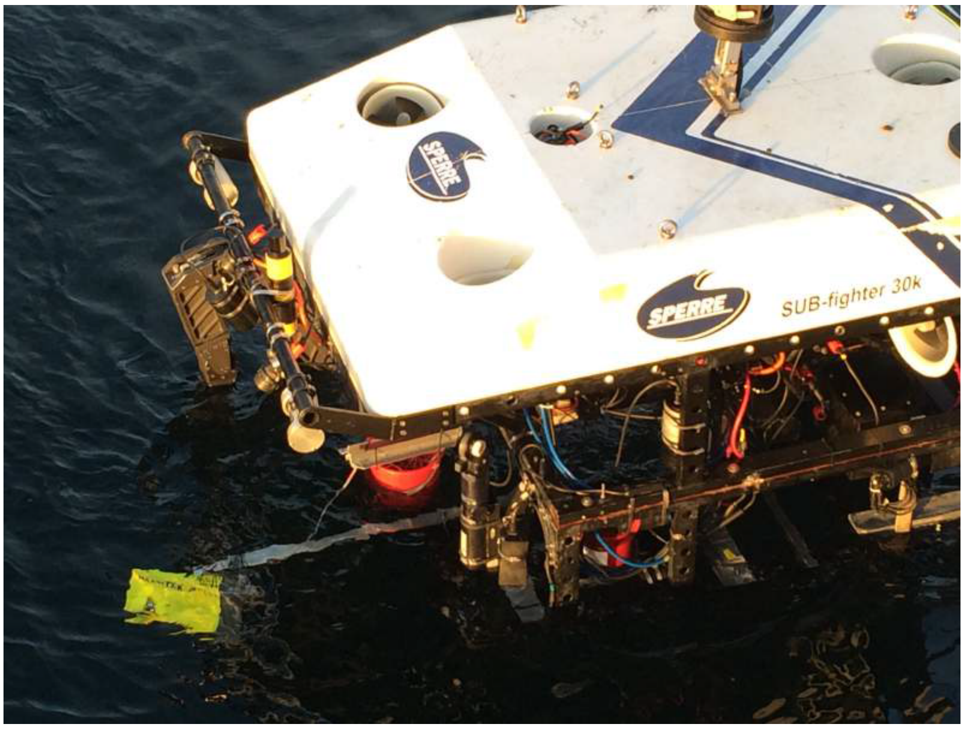


Figure 3.6: The ROV Neptunus and ROV SF 30K. Courtesy of NTNU

but it was placed vertically instead in the Neptunus. Due to a wiring error in the process of making Neptunus, the port motor has opposite directions of the other two. The ROV Neptunus is shown in figure 3.7. The Neptunus and the NTNU SF 30K (courtesy of NTNU) is pictured together in figure 3.6 (photo by Asgeir Sørensen). Photos were obtained from [1].

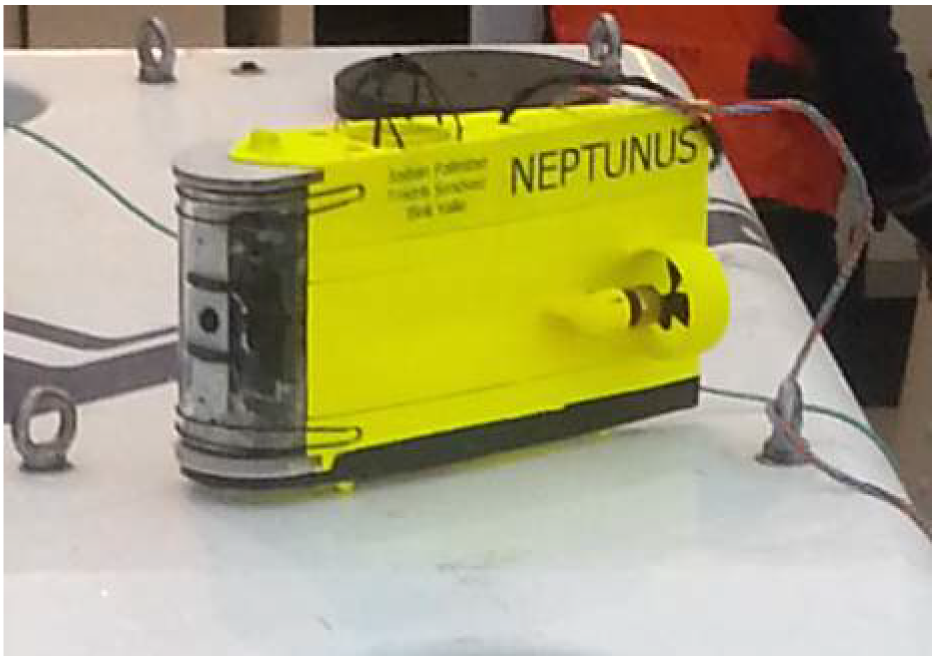


Figure 3.7: The ROV Neptunus

4 Choice of system solution

4.1 Computer hardware solution

The requirement of remote control from the product specifications limits the options for the system solution. Two solutions were considered, a web server solution and a solution using Nation Instruments CompactRIO hardware. The second solution provide remote monitoring and control functionality, but the cost of the hardware is on the scale of thousands of dollars. Due to the budget of the project, the CompactRIO was concluded not to be a valid option and a web server solution was chosen. The hardware used in the OpenROV was reviewed and deemed fit for the product specifications. As stated in chapter 3.4, the OpenROV was tested in [1]. There were no indications of weakness in hardware design revealed during this testing. Consequently it was decided to keep the hardware from the OpenROV in the Neptunus design.

4.2 ROV specific solution

Two ROV solutions were considered; Robotics Operating System (ROS) and LSTS Unified Navigation Environment (DUNE).

DUNE, not to be confused with the C++ library: "Distributed and Unified Numerics Environment", is a runtime environment for Autonomous Underwater Vehicles (AUVs) and ROVs, written in C++. DUNE is developed and maintained by a group of academics from the University of Porto. DUNE was discarded as an option mainly because no practically feasible solution for remote control was found.

ROS is primarily a robotics framework widely used in industry and academics. It has over 3000 publicly available packages according to [11], and it is free to use. It has a modular design where an application is built by one or mode blocks of code called nodes. One node typically performs one function. Nodes can be connected together making ROS applications highly scalable. ROS has been developed to be used on Linux primarely. C++, Python and some other languages can be used to write ROS applications. For client use, a JavaScript library called roslibjs can be used. It should not be problematic to write a web server inside a ROS application fulfilling the requirement of remote control. If ROS was to be implemented, it would be installed on the BeagleBone. ROS can interface with the serial port used to connect the BeagleBone to the Arduino.

The main advantage of using ROS would be that modules for navigation, control and possibly video streaming already exist. This was weighted against the time spent on initial installation and overhead of a ROS application. The product specifications states that the navigation and control modules shall contain a dynamic filter and DP with the sub-functionality of autopilots on heading and depth. Considering the quality of the sensors used, it would be natural to use basic controllers such a PID. More sophisticated controllers such as backstepping and feedback linearization require more accurate sensors than a PID controller. Consequently, the potential value of using ROS is diminished, as the implementation time of the navigation and control is estimated to be relatively small. Tuning of the parameters in a PID controller and a dynamic filter can often be more time consuming than the implementation itself. This work have to be done regardless of the user of ROS. On this basis it was concluded not to use ROS.

The option to no use any ROV specific frameworks or tools simplifies the learning curve for new developers entering the project, aiding the ease of further development as stated in the scope of work.

4.3 Software solution

As stated in section 4.1, a web server solution was chosen. Node was chosen as the server solution. Main reasons for this include:

- Rapid development with JavaScript as the only programming language across the server stack. This also facilitates the ease of further development.
- Two-way communication between server and client.
- Motor drivers already exist from OpenROV.

The Express.io frameworks was chosen as it suits the application. It facilitates rapid construction of the web server and the use of WebSockets protocol makes it easy for the server to send measurements and other updates.

In web servers it is common to use a client side tools to control the layout and appearance of a web page. This was considered, but as stated in the scope of work, emphasis shall be put on the developer interface. Consequentially, it was decided not to use any client side frameworks or tools as pure HTML and JavaScript was concluded to suffice. AngularJS and ReactJS can be reviewed in further work on this.

5 Product specifications

The product specifications were made by incremental testing and feedback from persons involved in the project. This chapter contains the final version of the product specifications. The testing chapter contains documentation of the incremental changes made.

The server shall include the following modules:

Navigation module for implementation of dynamic filters for state estimation. A Kalman Filter or passive observer can be used for this purpose.

Control module carrying out thrust and control requests from the client.

Camera module streaming the video from the webcam to the client.

Simulator module replacing the measurements with simulated measurements when activated. Based on a dynamic model and a discrete method for solving the differential equations of the system.

The ROV shall have remote control functionality with the following features:

- Accessible through WiFi and internet
- Only one client in control at any one time
- Transfer of control by request and acknowledge

The ROV shall also have a *Safety mode*. This shall stop the ROV when the client in control disconnects or loses focus of the window. It shall also trigger when run mode, control mode or input mode is changed.

The *operator interface* shall contain the following:

Camera stream image

Radio buttons for choosing input mode

- Keyboard
- Touchscreen
- Joystick
- Oculus

Radio buttons for choosing Control modes

- Motion control: In this control mode, the ROV is controlled in the surge, yaw and heave degrees of freedom.
- Manual control: Individual motor control.
- DP control: Dynamic positioning.

Radio buttons for choosing Run modes

- Normal
- Simulator

Display of the following measurements

- Heading [deg]
- Depth [m]

The following control commands

- Set autopilot depth
- Set autopilot heading
- Set gain
- Set lights intensity
- Set laser intensity

The *developer interface* shall contain the following:

- A logger than prints information to the user with time stamp. It shall be possible to control what information is displayed here.
- Measurements, commanded thrust, estimated states and simulated states, if the simulator is active, shall be saved. There shall be plotting functionality of saved data. Up to 10 plots, with the possibility to add and remove plots. There shall also be a option to delete saved data to reset the graphs.
- Relevant information shall be logged to file.

6 System overview

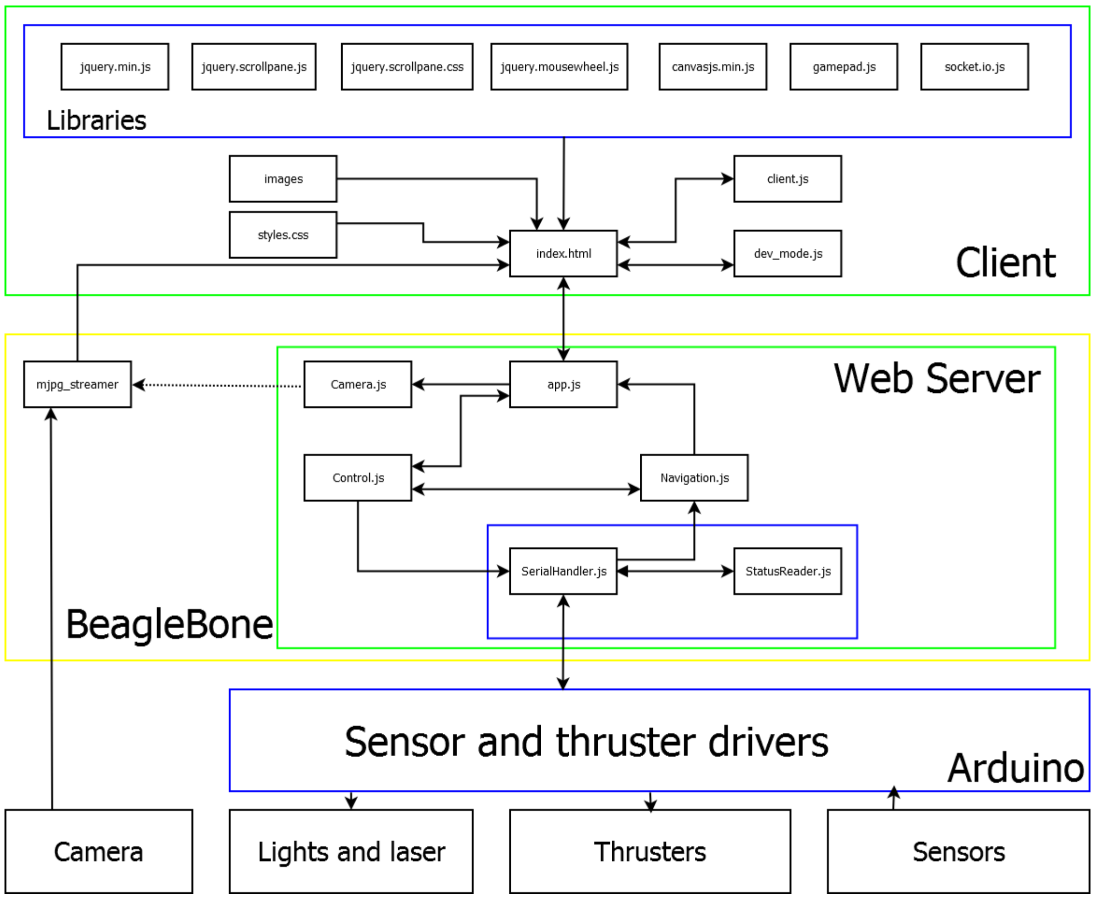


Figure 6.1: System overview

An illustration of the system topology in normal running conditions can be seen in 6.1. The whole lines represent control and data flow. The dotted line between Camera.js and mjpg-streamer represents instantiation. The green boxes are the client and the server software. These make up the content of the zip file delivered with the thesis.

The blue boxes are libraries used:

jQuery libs: jQuery is a large JavaScript library. It was used for a wide range of activities in the client. Examples include the ScrollPane, event handlers and identification of HTML elements.

canvas.js is used for plotting.

gamepad.js is a library for taking game controller and joystick inputs. This was used to control the ROV with an XBOX one controller

socket.io.js is the client side script for the socket.io library. This is automatically generated when using the express.io framework.

SerialHandler.js was made by OpenROV. Handles communication to the Arduino over a serial port. Messages are written to, and read, from a buffer.

StatusReader.js was also made by OpenROV. Used to read and interpret buffer content.

Sensor and thruster drivers written by OpenROV utilizing Arduino libraries. The software on the Arduino was not modified during the thesis work. The code can be found at [12].

Control.js contains thrust allocation and thrust commanding functionality, a stop method for stopping all thrusters, functionality to set lights and laser and placeholder functions for DP and autopilots. The Navigation.js file contains placeholders for dynamic filters. Other navigational methods, for example based on camera or laser, can be put here when developed. The arrow between Control.js and Navigation.js is a reference placeholder to be used in the feedback loop when implementing autopilot functionality.

Mjpg streamer is a command line application that is being used to stream video from the webcam to the client. It broadcasts the stream over http on a separate port. The webcam is connected to the BeagleBone with a regular USB cable.

The communication between the BeagleBone and Arduino is done through a serial port.

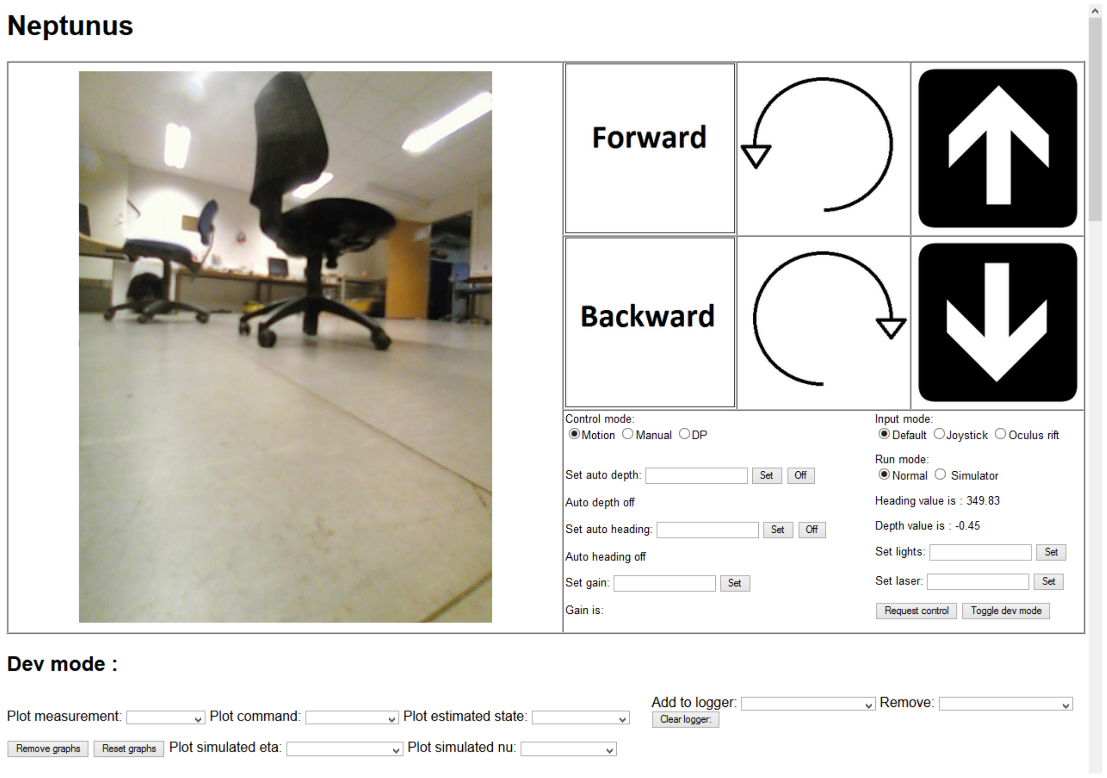
The communication between index.html and app.js is facilitated by the socket.io library through the WebSockets protocol.

The client.js and dev-mode.js are the scripts that governs the user interface and the developer interface, respectively.

7 User Interface

The user interface is displayed by entering the web server IP and port in a web browser URL bar in the following format: *ip:portnr*. All the major browsers are supported.

7.1 Operator interface



Control mode: [✓ Motion] Manual DP

Set auto depth: Set Off

Auto depth off

Set auto heading: Set Off

Auto heading off

Set gain: Set

Gain is:

Input mode: [✓ Default] Joystick Oculus rift

Run mode: [✓ Normal] Simulator

Heading value is : 349.83

Depth value is : -0.45

Set lights: Set

Set laser: Set

Figure 7.1: Operator interface

The operator interface shown in figure 7.1, with the top of the developer interface showing at the bottom, contains the functionality specified in the product specification with the default settings. The six images to the top right are used for controlling the ROV with touch input. Any touchscreen device can be used such as tablets, smartphones and touch monitors. The default option for the input mode is either keyboard or touchscreen depending on the platform used. If both are available simultaneously, then both can be used. If one were to input conflicting

Table 7.1: Keyboard input

Key	**Command (motion)**	**Command (manual)**
W	Positive surge	Vertical thruster up
Q	Positive heave	Port thruster forward
D	Positive yaw	Starboard thruster backward
S	Negative surge	Vertical thruster down
E	Negative heave	Starboard thruster forward
A	Negative yaw	Port thruster backward

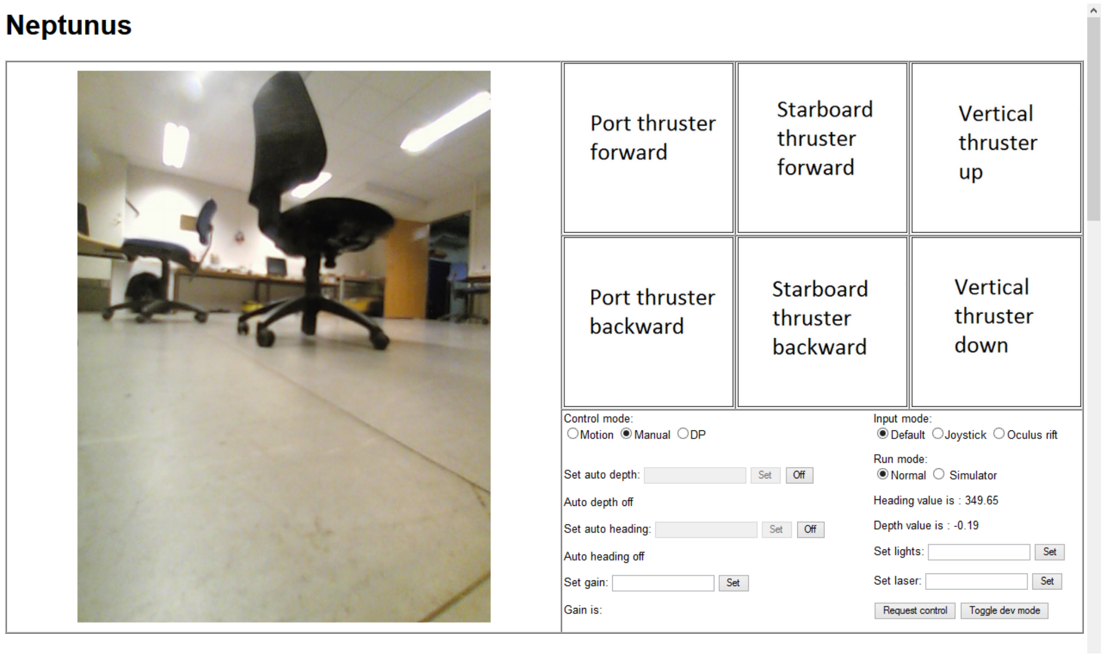
Table 7.2: Joystick input

Key	**Command (motion)**	**Command (manual)**	**Value**
Right stick Y	Surge	Starboard thruster	[-1, 1]
Left stick Y	Heave	Port thruster	[-1, 1]
Right stick X	Yaw	-	[-1, 1]
Left bottom shoulder (LT)	-	Vertical thruster down	[0, 1]
Right bottom shoulder (RT)	-	Vertical thruster up	[0, 1]

inputs on a keyboard and a touchscreen simultaneously, the most recent command would overwrite the previous command.

Table 7.1 shows keyboard input keys and corresponding thrust requests. Table 7.2 described joystick input. An XBOX One controller illustrated in figure 7.2 was used for testing, but any similar game controller such as Playstation or Nintendo controller with PC drivers should work.

Figure 7.2: XBOX one controller



Control mode: Motion [✓ Manual] DP

Set auto depth:

Auto depth off

Set auto heading:

Auto heading off

Set gain:

Gain is:

Input mode: [✓ Default] Joystick Oculus rift

Run mode: [✓ Normal] Simulator

Heading value is : 349.65

Depth value is : -0.19

Set lights:

Set laser:

Figure 7.3: User interface in manual control mode

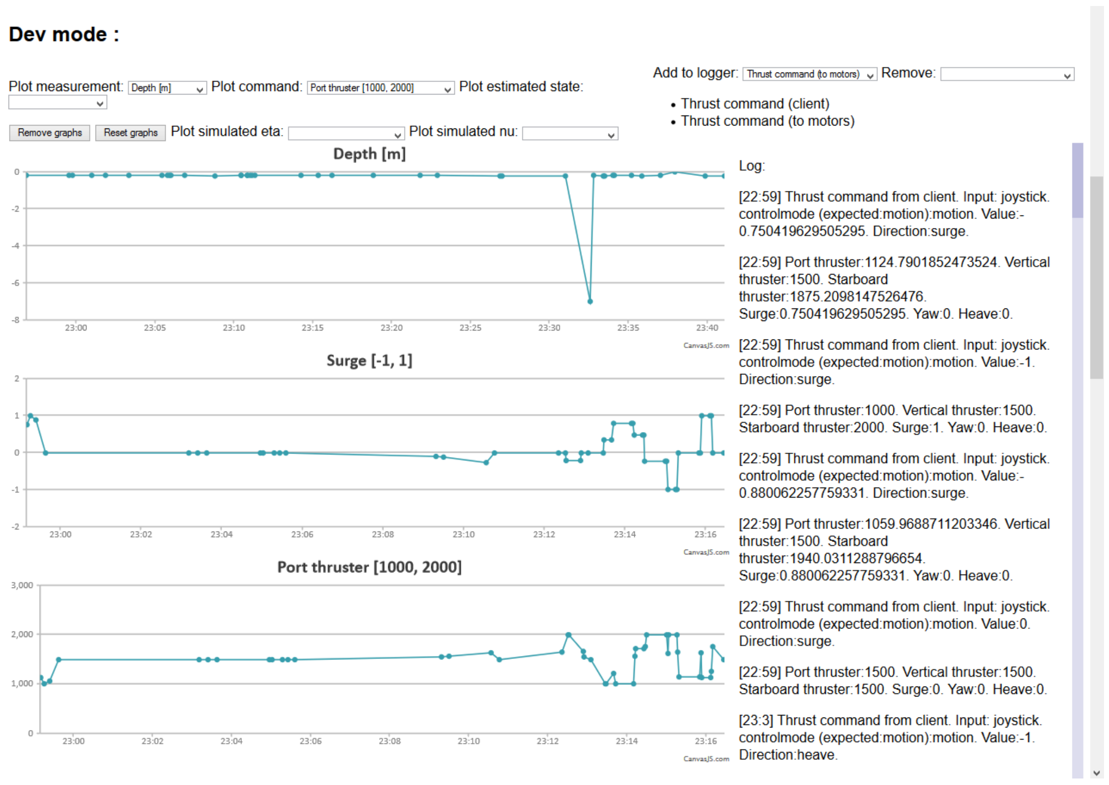
When switching to manual control mode the pictures are replaced. Figure 7.3 illustrates this. The auto pilots on depth and heading are disabled when manual control is chosen.

The DP control mode, heading auto control, depth auto control and the oculus rift input mode are placeholders. The framework around these inputs have been made, such that when DP mode, auto heading or auto depth is requested, a signal will be sent to the control module. The control module contains empty functions to be used for this. When DP mode is chosen auto depth and auto heading are disabled in the client. Control input from keyboard, touchscreen or joystick are rejected when DP or Oculus is chosen as input. The valid input range of the lights and laser form is [0, 255]. Valid input for the gain is [0, 1]. Invalid requests will be rejected in the web server by the Control.js module.

Event handlers were implemented such that any swapping between control mode, input mode or run mode triggers a signal to the server ordering the ROV to stop. The stop signal will also be triggered if focus on the window is lost by listening for the "onblur" event. This happens if the window is minimized or if other windows pop up and take focus.

The "Request control" button is for the remote control and transfer of control functionality. The "toggle dev-mode" button is used to show or hide the developer interface.

7.2 Developer interface


Dev mode :
Plot measurement: Depth [m]
Plot command: Port thruster [1000, 2000]
Plot estimated state:
Remove graphs
Reset graphs
Plot simulated eta:
Plot simulated nu:
Add to logger: Thrust command (to motors)
Remove:
• Thrust command (client)
• Thrust command (to motors)
Depth [m]
Surge [-1, 1]
Port thruster [1000, 2000]
CanvasJS.com

```
Log:

[22:59] Thrust command from client. Input: joystick.
controlmode (expected:motion):motion. Value:-
0.750419629505295. Direction:surge.

[22:59] Port thruster:1124.7901852473524. Vertical
thruster:1500. Starboard
thruster:1875.2098147526476.
Surge:0.750419629505295. Yaw:0. Heave:0.

[22:59] Thrust command from client. Input: joystick.
controlmode (expected:motion):motion. Value:-1.
Direction:surge.

[22:59] Port thruster:1000. Vertical thruster:1500.
Starboard thruster:2000. Surge:1. Yaw:0. Heave:0.

[22:59] Thrust command from client. Input: joystick.
controlmode (expected:motion):motion. Value:-
0.880062257759331. Direction:surge.

[22:59] Port thruster:1059.9688711203346. Vertical
thruster:1500. Starboard
thruster:1940.0311288796654.
Surge:0.880062257759331. Yaw:0. Heave:0.

[22:59] Thrust command from client. Input: joystick.
controlmode (expected:motion):motion. Value:0.
Direction:surge.

[22:59] Port thruster:1500. Vertical thruster:1500.
Starboard thruster:1500. Surge:0. Yaw:0. Heave:0.

[23:3] Thrust command from client. Input: joystick.
controlmode (expected:motion):motion. Value:-1.
Direction:heave.
```

Figure 7.4: Developer interface

Figure 7.4 shows the developer interface. The left side is used for adding, removing or resetting plots. The following measurements, commands, estimated states and simulated states can be plotted, as table 7.3 illustrates.

Figure 7.5 and 7.6 illustrates the plotting functionality. All the plots are from the same run. Note that the port thruster signal has opposite effect as the two others, as the thruster direction is reversed as explained in chapter 3.4.

Table 7.3: Data saved for plotting

Measurement	**Command**	**Estimated state**	**Simulated state**
Depth [m]	Surge	Estimated roll	Simulated x
Roll [deg]	Heave	Estimated pitch	Simulated y
Pitch [deg]	Yaw	Estimated heading	Simulated z
Heading [deg]	Starboard thruster	Estimated depth	Simulated ψ
Voltage useage [V]	Port thruster		Simulated u
Current usage [A]	Vertical thruster		Simulated w
			Simulated r

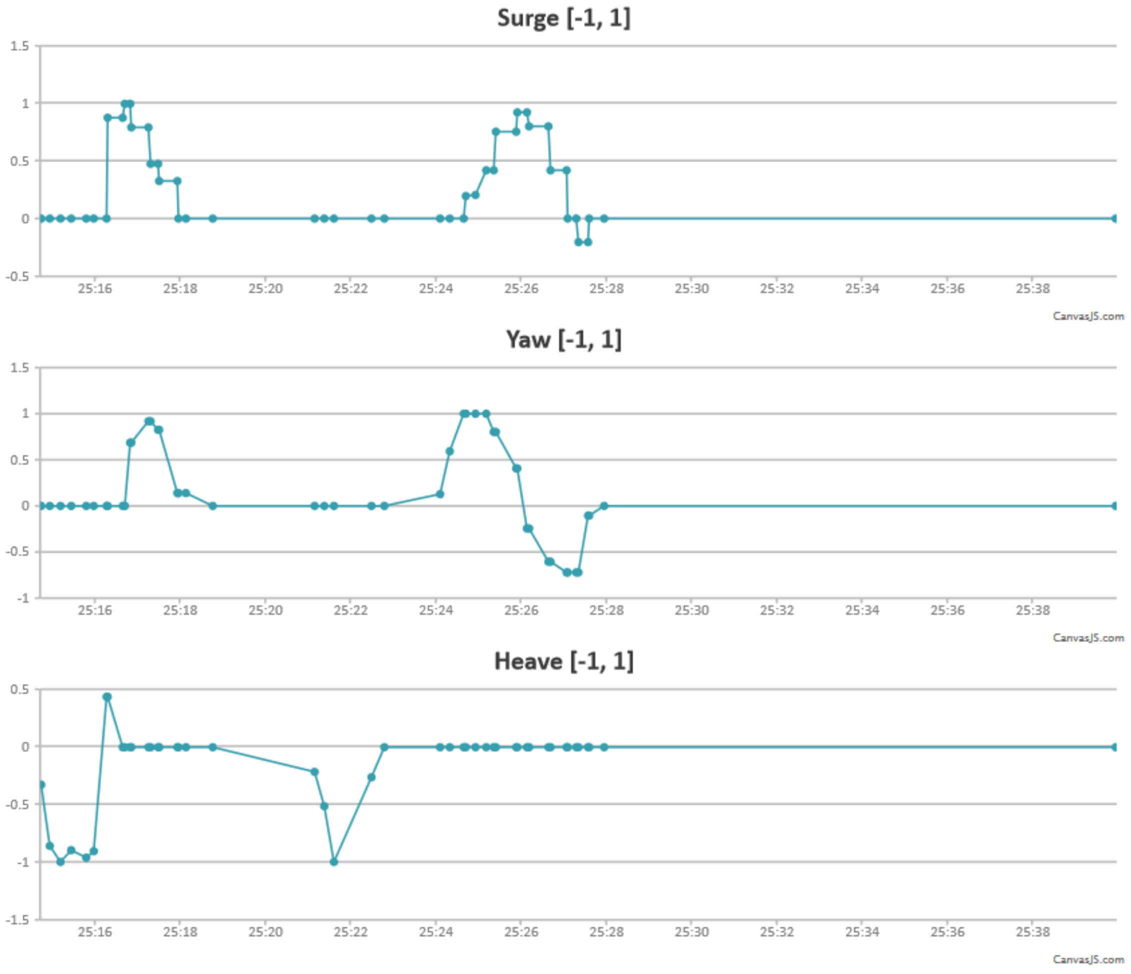


Figure 7.5: Plot of surge, yaw and heave

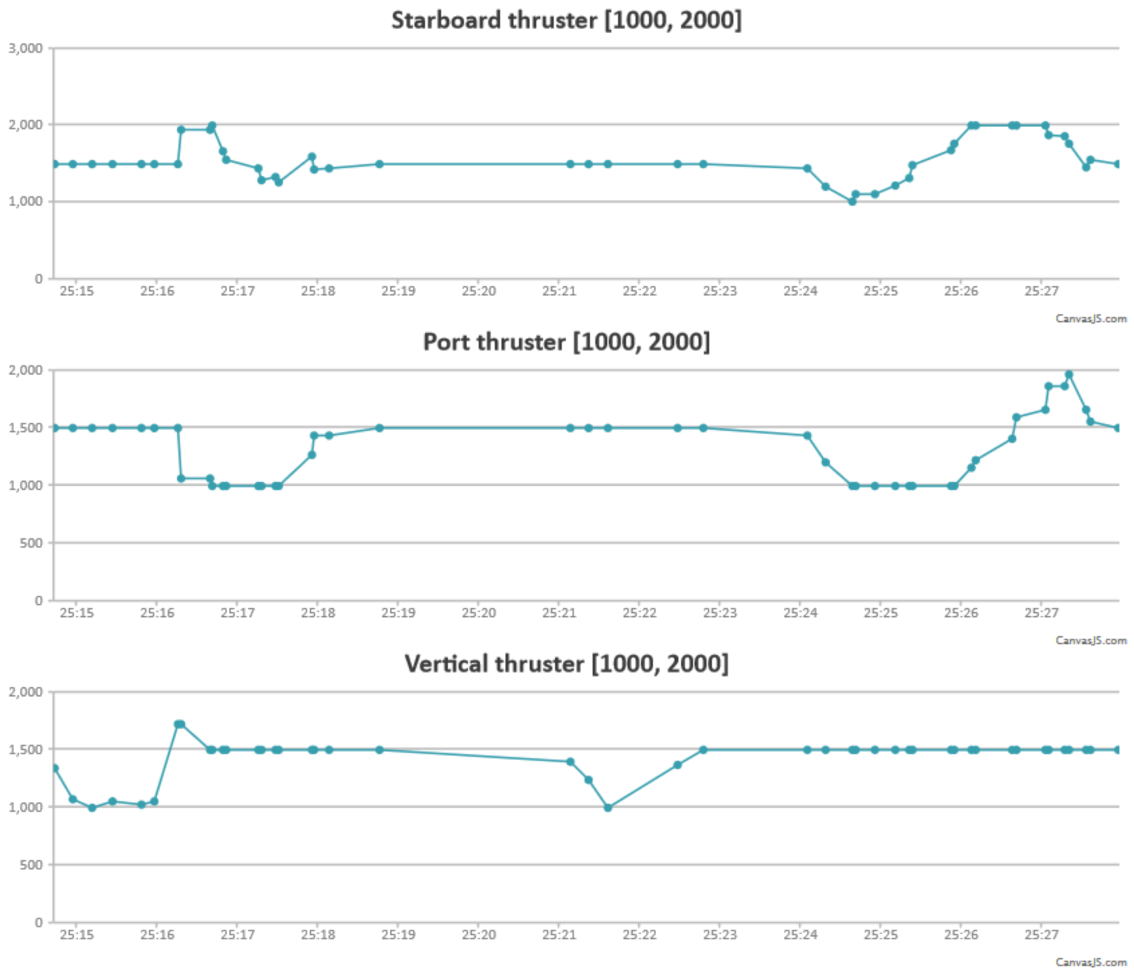


Figure 7.6: Plot of starboard, port and vertical thruster

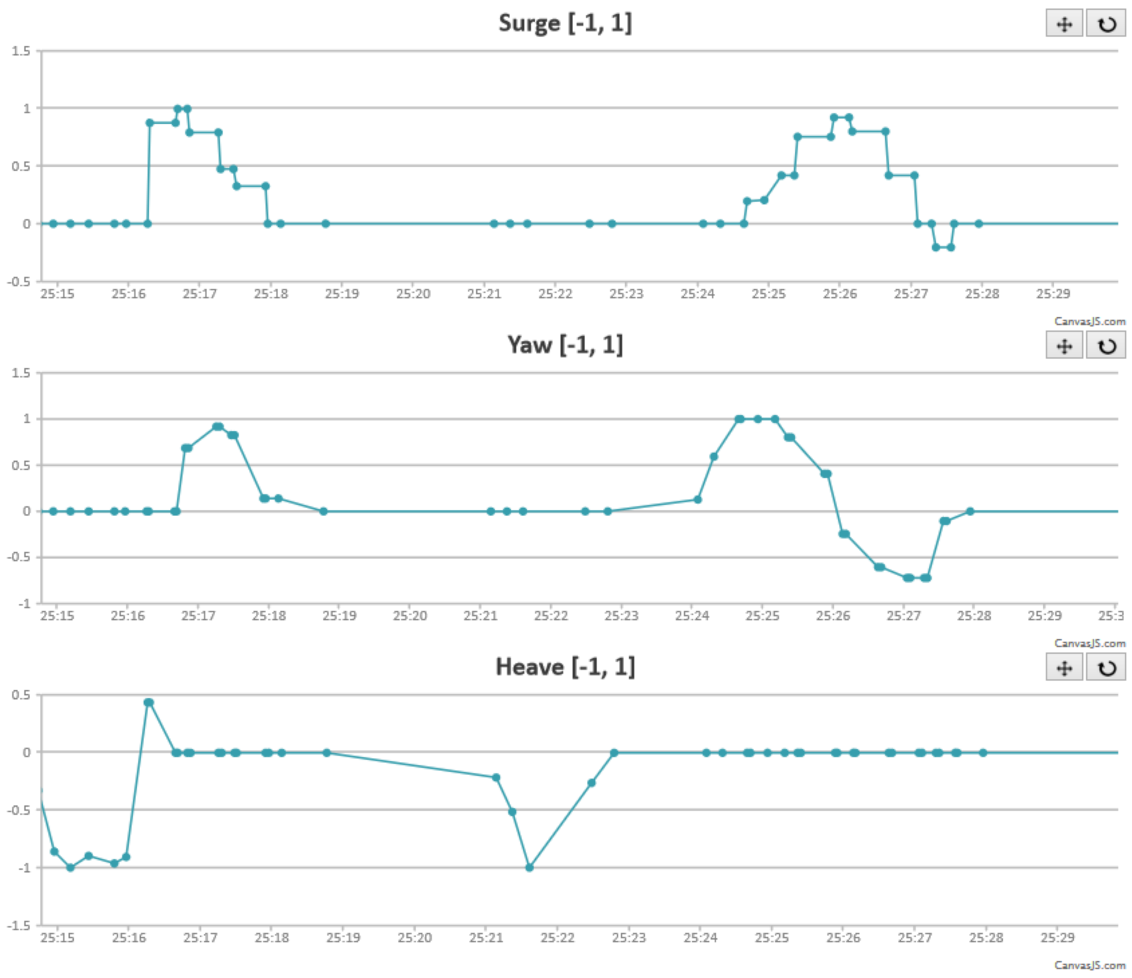


Figure 7.7: Zoomed version of surge, yaw and heave

The CanvasJS library was used for the plotting functionality. It has zooming and panning functionality for the x axis while the y axis is automatically scaled to fit. Figure 7.7 shows a zoomed in version of figure 7.5 The two boxes in the top right corner is used to toggle between panning and zooming, and to zoom out. When hovering the cursor over a data point, a small window appears displaying the x and y value of the data point.

Up to a maximum of 10 plots can be displayed simulataniously. The ”remove plots” button clears the plots from the screen, but the data is still stored. The ”reset graphs” will delete all data using for plotting, but not remove the plot windows from the interface.

The right side of the developer interface is used for logging. This functionality is for displaying information that is not feasible to plot. A dropdown menu is used to select what type of information to be displayed. Another dropdown is used to deselect displayed information. The currently chosen information types are shown below the dropdowns in a list. In real time, info is displayed on a ScrollPane with a time stamp. In contrast to the plotting functionality, the logger does not store the information, and cannot display data retroactively.

List of available choices for the logger:

- Gain change
- Thrust command (client)
- Thrust command (to motors)
- Transfer of control
- Autopilot depth
- Autopilot heading
- Stop

8 Signal flow and processing

8.1 Data flow

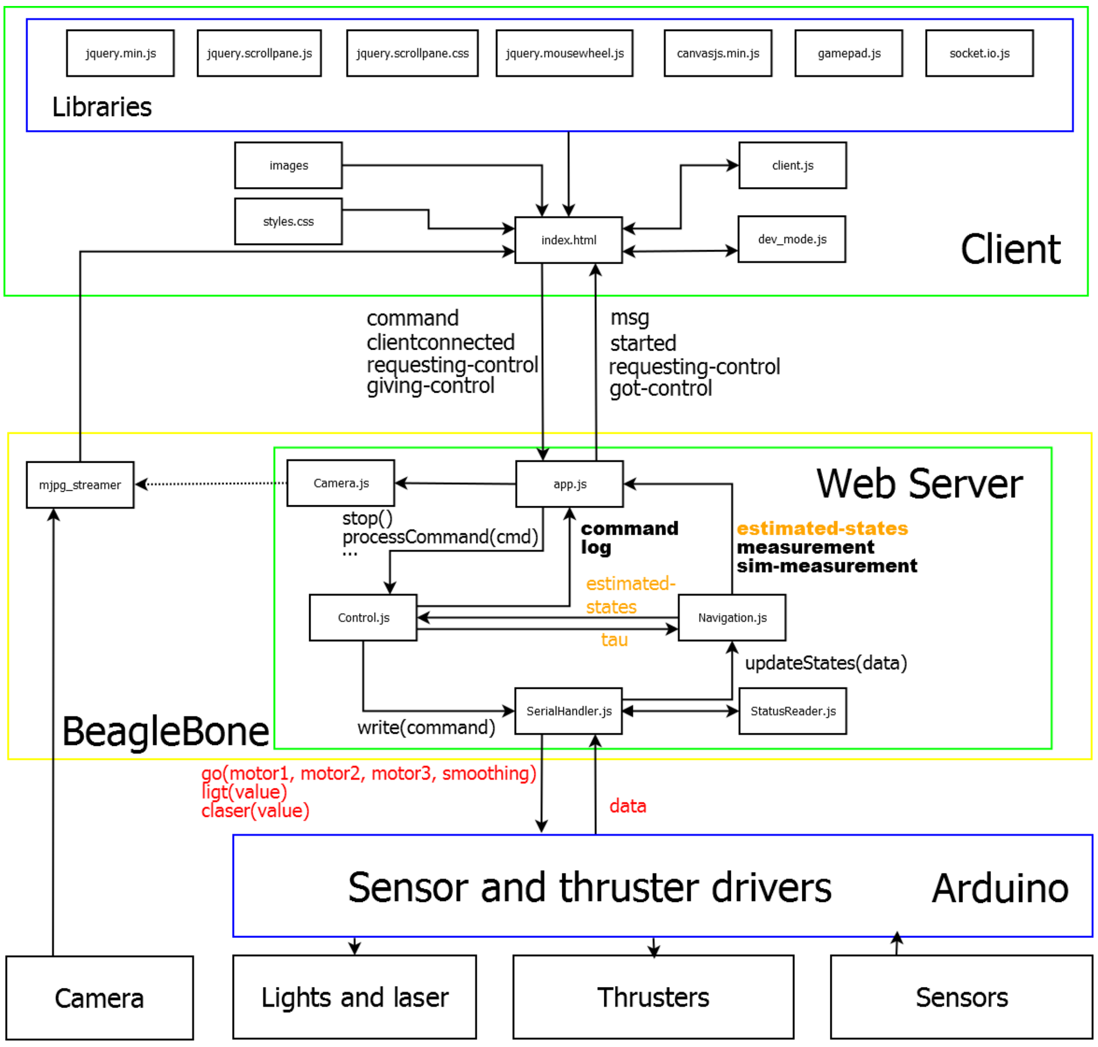


Figure 8.1: Signals

Figure 8.1 shows the main signal flow in the system. Parenthesis indicates function call. All data exchanged between server and client is done by emitting events. These emits are listed in figure 8.1 between app.js and index.html. Listing 5 shows the code that emits a thrust command from the client to the server. Listing 6 shows a part of the function in the server that handles the incoming command event.

Listing 5: Thrust command from client

```
var command = {
                    type: "thrust-command",
                    input: "default",
                    controlmode: controlMode,
                    val: value,
                    keypress: pressType,
                    dir: direction
                    };
io.emit("command", command);
```

Listing 6: Thrust command recieved in server

```
app.io.route("command", function(req){
     requestIP = req.handshake.address.address;
     if(isInControl(requestIP)){
          switch(req.data.type){
               case "thrust-command":
                    control.processCommand(req.data);
                    break;
           case "stop":
                    control.stop();
                    break;
                ....
          }
     }
});
```

The command event does not only contain thrust commands. The following list contains the types of objects emitted with the command event.

- thrust-command
- stop
- runmode-normal
- runmode-simulation
- set-gain
- set-lights
- set-laser
- auto-heading
- auto-depth
- auto-heading-off
- auto-depth-off
- dp

Figure 8.1 shows two function calls from app.js to Control.js: processCommand(cmd) and stop(). processCommand(cmd) is the function that processes thrust commands. The three dots in the figure represents a set of functions that handles the rest of the different command event types. For example, if the command object has type set-gain, then the function control.setGain(value) is called. The write(command) function from Control.js orders SerialHandler.js to write the argument to the buffer and send it over the serial port. The red text is what is being sent and read from the buffer. The data object sent from SerialHandler.js to Navigation.js contains all the measurements. Table 8.1 lists the measurements found in the data object.

Table 8.1: Measurements

Variable	**Measurement**
data.deap	Depth [m]
data.roll	Roll [deg]
data.pitc	Pitch [deg]
data.hdgd	Heading [deg]
data.vout	Voltage [V]
data.iout	Current [A]

The server contains an EventEmitter object that is used to send information back to app.js from different modules of the web server. This is represented by the bolded texts in figure 8.1. The emitter emits a ”toClient” event. The object passed in the event contains a type and a content property. The names of the bolded texts are the types that are being sent. For example, measurements are sent from the Navigation.js module to app.js by the following line of code:

Listing 7: measurement emit

```
statusEmitter.emit("toClient",
                   {
                    type:"measurement",
                    content:data
                   });
```

app.js listens to the "toClient" event. When triggered, the object emitted is passed on to the client with the "msg" event. The content of the object is then used to display measurements, plotting and logging. Measurements, thrust commands, estimated states and simulated states are saved in the client by the `dev_mode.js` script. This process starts automatically such that plot data recording starts on the client connects and not just when the plot is requested. The data is saved as objects with the data stored together with a Date object as private variables. The Date object saves the current time upon creation, and it is used for the x axis to the plots. Simulated states will not be saved during normal run mode and vice verca for measurements.

The orange texts in figure 8.1 are placeholders. Dynamic filter needs to be implemented to estimate states. This will require knowledge of the thrust commands given (tau). The estimated states will be used in the Control.js module in DP control and autopilots.

8.2 Deadband function

When the user presses a valid keyboard button, moves the joystick or presses a touchscreen element, the input mode is first checked. If input from wrong source, the signal is rejected. If a joystick is used the signal is sent through a deadband test before being sent to the server. In pseudocode, the signal denoted x:

Listing 8: Deadband function

```
function deadband(x)
{
    if(abs(x) > epsilon){
        send thrust command;
    }else{
        stop;
    }
}
```

This is to prevent unintentional thrust commands when the stick is resting around zero. It would also be difficult to completely stop without this functionality as the stick sensors are quite sensitive. epsilon = 0.1 was chosen for testing.

8.3 Wild points

Wild points in the measurements were detected during testing. It was concluded that the source of the majority of the wild points was most likely caused by errors in the buffer used to read data from the serial port in the SerialHandler.js script. There were several arguments for this conclusion. The wild points were frequently close to 0.1, 10, 100 or 1000 times the approximated real value. Measurements of pitch, roll and heading were at times outside the range of what an IMU will output. It was also observed at times that the measurement value for heading would be "pitc", which is the variable name for the pitch measurement.

A filter was implemented in the dev-mode script to reject wild points. It checks if the measurement is a number within a defined interval. For roll, pitch and heading this was [0, 360], current [0.01, 4], voltage [10, 13] and depth [-10, 200]. Figure 8.2 illustrates wild points after filter implementation. It is left as further work to improve on this.

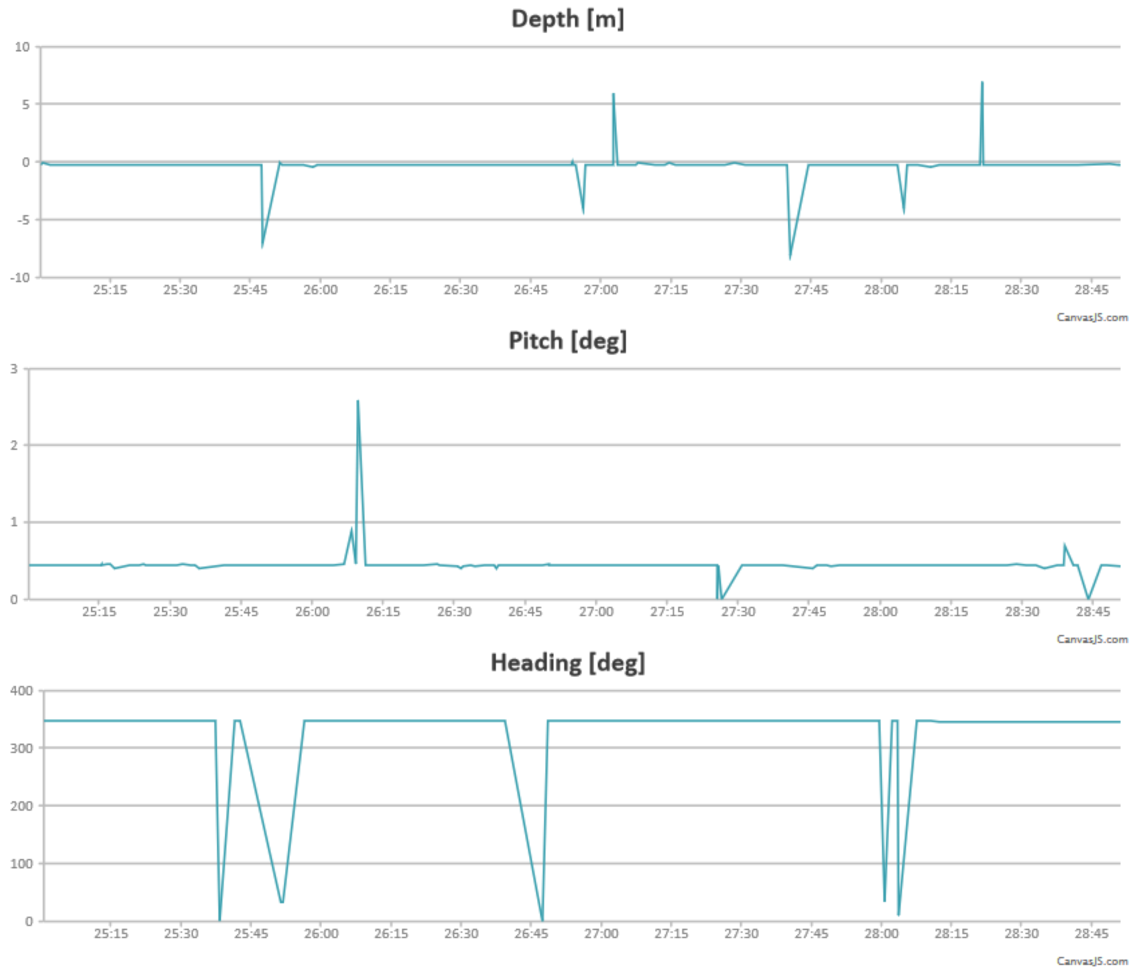


Figure 8.2: Wild points

8.4 Serial port API

Documentation of the serial port API made by OpenROV is partially documented in [13]. The relevant functions used are:

- "go(motor1, motor2, motor3, smoothing);"
- "ligt(lightIntensity);"
- "claser(laserIntensity);"

The first, second and third motor is the port, vertical and starboard thruster, respectively. The smoothing argument is whether or not to activate a smoothing filter on thrust commands in the motor drivers. The input range for the motor arguments are [1000, 2000], where 1000 is full speed backward and 2000 is full speed forward. The ligt and claser functions sets the lights and lasers. The input range on the lights and laser is [0, 255]. In the OpenROV documentation, the motor arguments are denoted with type:milliseconds. This is because PWM signals are used for the motor controllers. Appendix A.8 further elaborates on the serial port API.

9 Thrust Allocation and Control

Thrust commands are passed to the Control.js module with the command object of type "thrust-command". The properties of the object and the values they can have are as follows:

$$\text{var command} = \begin{cases} \text{type:} & \text{"thrust-command"} \\ \text{input:} & \text{"default", "joystick"} \\ \text{controlmode:} & \text{"motion", "manual"} \\ \text{value:} & \{-1,0,1\} \text{ or } [-1,1] \text{ or } [0,1] \\ \text{dir:} & \text{"surge", "heave", "yaw"} \\ \text{thruster:} & \text{"port", "starboard", "vertical"} \end{cases}$$

The command object will contain either the "dir" property when motion control is used, or the "thruster" property when manual control is used. The value range depends on input mode. $\{-1,0,1\}$ for default and either [-1, 1] or [0, 1] for joystick as shown in table 7.2).

The Control.js module saves the values for the desired thruster speed. When a new thrust command is received, the desired thruster speed is updated and sent to the motors. In manual control mode, the incoming commanded value is multiplied by an adjustable scalar gain k = [0, 1] and scaled from [-1, 1] to [1000, 2000] in accordance with the range mentioned in chapter 8.4. In motion control, the commanded value is multiplied by k, then a thrust allocation is performed before the value is scaled as in manual control mode.

9.1 Control modes

Three control mode choices are available in the user interface:

- Manual control
- Motion control
- DP

In the manual control mode, the user input is mapped straight to the individual thruster. Motion control is used to control the vessel in the surge, yaw and heave degrees of freedom. The DP control mode option was included as a placeholder for future implementation.

9.2 Control functions

- Autopilot heading
- Autopilot depth

These control functions were included in the client as placeholders. Functionality to process the input in the client and pass the request to the control module was implemented.

9.3 Thrust allocation

The Control.js module saves the direction and value of an incoming thrust command when in motion control mode. This is saved in the thrust vector τ_{motion}. u denotes the desired thruster force.

$$\tau_{\text{motion}} = \begin{bmatrix} X & Z & M \end{bmatrix}^\top \tag{9.1}$$

$$u = \begin{bmatrix} u_{\text{port}} & u_{\text{vertical}} & u_{\text{starboard}} \end{bmatrix}^\top \tag{9.2}$$

X is the surge value, Z the heave value and M the yaw value. The thrust allocation was developed by simply looking at the geometry of the thrusters and the different forces and moments that they produce. The scalar gain was applied and saturation elements were applied to limit the signal to [-1, 1].

$$u = \begin{bmatrix} \text{sat}(k(X+M)) & kZ & \text{sat}(k(X-M)) \end{bmatrix}^\top \tag{9.3}$$

The port thruster propeller is mounted the opposite way of the two others as mentioned in (ref intro neptunus). The port signal is therefore inversed before the thrust command is sent to the motor drivers. Theory on thrust allocation can be found in [10].

10 Camera stream

Two tools for camera streaming were considered; "mjpg-streamer" and "FFmpeg".

FFmpeg is a large multi purpose project for processing video and audio. It has tools for recording, converting and streaming among other functionalities [14].

mjpg-streamer is the tool used by OpenROV. It is a lightweight command line application that is designed to stream video with decent quality and minimal resource usage. From the website: "It was written for embedded devices with very limited resources in terms of RAM and CPU." [15] [16].

mjpg-streamer was chosen for easy setup and use, as well as the performance. The documentation of mjpg-streamer is sparse compared to FFmpeg, but it is quite simple to install and use. The mjpg-streamer fills precisely the purpose of the application. If post processing of the video was required, FFmpeg would be a more suited option. The procedure for setting up mjpg-streamer is explained in appendix A.7. It is run with the command in listing 9.

Listing 9: Starting the mjpg-streamer

```
spawn(mjpg-streamer -o "outputArguments"
                              -i "inputArguments");
```

spawn() is a method of the ChildProcess class, which is a part of the Node API. It executes the function argument as a command line instruction. This is done in a separate process that outputs info about the state of the process through data streams. This allows for error messages to be parsed back to Node if the process fails.

Available input arguments for the mjpg-streamer are; input device, resolution, framerate and format. Output arguments are; www-folder-path, port, password, commands. A list of supported resolutions can be found in [17]. 600x800 was used for the tests and in the pictures from chapter 7.

The stream is accessed in client.js and displayed by the following function;

Listing 10: Video streaming

```
function startVideo()
{
  io.on("started", function(){
    var videoElement = document.getElementById("sourcevid");
    var adress = "http://192.168.0.12:3031/?action=stream";
    videoElement.setAttribute("src", adress);

    var videoContainer = document.getElementById("container");
    videoContainer.style.MozTransform = "rotate(270deg)";
    videoContainer.style.WebkitTransform ="rotate(270deg)";
    videoContainer.style.oTransform = "rotate(270deg)";
    videoContainer.style.transform = "rotate(270deg)";
    videoContainer.style.msTransform = "rotate(270deg)";
  });
}
```

The *started* event is emitted by the server after it has loaded the video. 192.168.0.12 is the IP of the web server and 3031 is the port argument specified in the initialization of mjpg-streamer. This is a different port than the one used for communication between app.js and the client. As mentioned in chapter 3.4, the camera is mounted such that the image needs to be rotated in the client. Several function calls for the *rotate(deg)* function is necessary because of differences in the browsers.

11 Remote control

The system specifications states that the ROV shall be controllable locally with WiFi and globally using the internet. Only one client shall be in control at any given time, and the ROV shall have functionality for transfer of control between clients.

11.1 Remote control

A WiFi connection was established by connecting the ROV umbilical to a router. Accessing the BeagleBone and starting the web server was done through ssh. The procedure of connecting to the BeagleBone is further elaborated in appendix A.2.

The common method of assigning an IP address to a computer is through dynamic address assignment. This is typically done with Dynamic Host Configuration Protocol (DHCP). With this method, a router will handle allocation of IP addresses. An IP address also can be assigned manually as a static IP address. A connection to the BeagleBone can be achieved with both dynamic and static IP assignment. If a dedicated router is used, the method of static IP assignement is convenient for the given problem. If one does not have multiple connections to the BeagleBone available, the use of static IP address will make it easier to re-establish a cable connection if the wireless connection is unsuccessful.

The ROV was successfully run with static IP addressing over local WiFi connection. Attempts were made to control the ROV over the internet with static and dynamic IP, however unsuccessfully.

11.2 Industry standards for transfer of control

The major classification societies have difference in their practice, but they all require only one operator to be in control of a vessel at any time. DNV GL has the following standards for transfer of control [18];

- The main operator station shall be able to take control without acknowledge, but an audible warning must precede the transfer of control.
- Transfer of control from other stations must be by a request and acknowledged before initiated. This is unless the operating stations are within visual and audio contact range.

- Set-points and parameters for autopilots, DP and other control functions must be synchronized between work stations before transfer of control. This may require manual adjustment of levers and joysticks. This is commonly referred to as *pick-up procedure*

11.3 Implementation of transfer of control

A default master client is chosen by specifying the IP address in app.js. Any other client can connect to the ROV, observe it and display plots, but only the master can send commands. A client can request control by clicking the "request control" button in the user interface. A "request-control" event is then emitted to the server. If the requesting client is not already the master the "request-control" event is sent from the server to the master client. This triggers a popup window in the master client informing of the IP of the requesting client and a prompt to give control. If the prompt is accepted, a signal is sent to the server which transfers control. The new master is then alerted that it's request was accepted. The ROV is also stopped when transfer of control is performed. When DP and autopilot functionality is implemented, pick-up procedures should be implemented.

If the master client disconnects, the ROV will stop in accordance with the *safety mode* in product specifications. This is achieved by the code listing 11.

Listing 11: ROV stop when disconnected

```
app.io.on("connection", function(socket){
    var ip = socket.handshake.address.address;
    socket.on("disconnect", function(){
        if(ip == masterIP){
            console.log("Master Client disconnected with ip:
                        " + ip + " stopping");
            control.stop();
        }
    });
});
```

12 Simulator

The software system in simulator mode is illustrated in figure 12.1. It simulates the ROV in four degrees of freedom; position and heading. The thrust commands from the client is routed to the simulator module instead of the motors. The simulator calculates the movement of the vessel in the time domain. This is calculated from a discretized version of the system model. The simulated states are outputted back to the user interface instead of measurements from the sensors when the simulator is active.

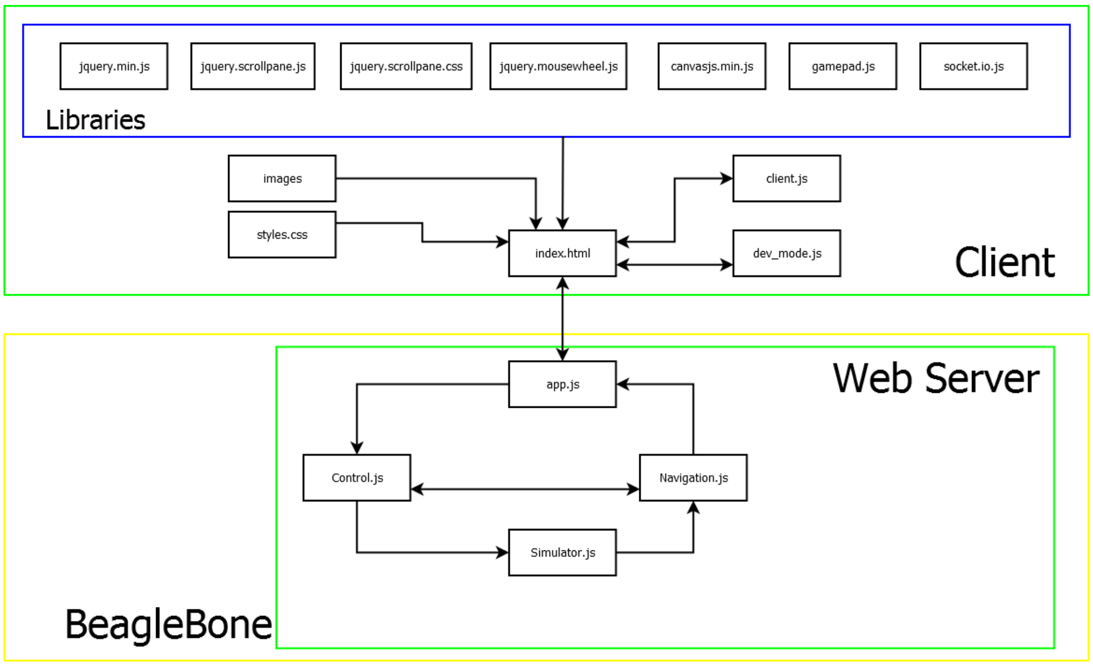


Figure 12.1: System overview with simulator active

The kinetics from [10] is used, adjusted to four degrees of freedom. Pitch and roll was modeled to be constant equal to zero. Assuming negligible coriolis and centripetal force due to low speeds, the equations of motion:

$$\dot{\eta} = J(\psi)\nu \tag{12.1}$$
$$M\dot{\nu} + D(\nu)\nu + g = \tau \tag{12.2}$$

$$\eta = \begin{bmatrix} x & y & z & \psi \end{bmatrix}^\top \tag{12.3}$$

$$\nu = \begin{bmatrix} u & v & w & r \end{bmatrix}^\top \tag{12.4}$$

$$J = \begin{bmatrix} R(\psi) & 0 \\ 0 & 1 \end{bmatrix} \tag{12.5}$$

Where η denotes the NED frame position and yaw angle. ν denotes the BODY frame velocities. $R(\psi)$ is the rotation matrix about the down axis. As the ROV has no lateral or azimuth thrusters it does not generate any sway motion and $\tau_{\mathrm{v}} \equiv 0$. The force vector produced by the motors in 4-DOF is then:

$$\tau = \begin{bmatrix} fkX \\ 0 \\ fkZ \\ 2fklM \end{bmatrix} \tag{12.6}$$

$\tau_{input} = \begin{bmatrix} X & Z & M \end{bmatrix}^\top$ and k as defined in chapter 9. f is the force produced by the thrusters, f = 12N at maximum capacity according to [19]. l is the yaw moment arm measured to be 0.045m.

Coefficient matrices according to [19] reduced to 4-DOF

$$M = M_A + M_{RB} \tag{12.7}$$

$$M_A = \begin{bmatrix} 2.055 & 0 & 0 & 0 \\ 0 & 14.6033 & 0 & -0.8926 \\ 0 & 0 & 2.3295 & 0 \\ 0 & -0.8926 & 0 & 0.1498 \end{bmatrix} \tag{12.8}$$

$$M_{RB} = \begin{bmatrix} 3.4600 & 0 & 0 & 0 \\ 0 & 3.4600 & 0 & 0 \\ 0 & 0 & 3.4600 & 0 \\ 0 & 0 & 0 & 0.399 \end{bmatrix} \tag{12.9}$$

$$D = D_l + D_q(\nu_r) \tag{12.10}$$

$$D_l = \begin{bmatrix} 2.2907 & 0 & 0 & 0 \\ 0 & 4.9804 & 0 & 0 \\ 0 & 0 & 15.1897 & 0 \\ 0 & 0 & 0 & 0.2605 \end{bmatrix} \tag{12.11}$$

$$D_q(\nu_r) = \begin{bmatrix} 4.008|u_r| & 0 & 0 & 0 \\ 0 & 35.216|v_r| & 0 & 0 \\ 0 & 0 & 10.304|w_r| & 0 \\ 0 & 0 & 0 & 0.320|r| \end{bmatrix} \tag{12.12}$$

$$g = \begin{bmatrix} 0 \\ 0 \\ 0.0234 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{12.13}$$

Where $D_q(\nu_r)$ is the quadratic damping coefficient matrix and D_l is the linear damping coefficient matrix.

Functionality for using the simulator with manual control mode was not implemented. This can be achieved by implementing functionality to perform the mapping described in chapter 9.3 in reverse.

A numerical solution to the dynamic system was achieved by the use of the Euler method with step size dt = 0.01.

The MathJS JavaScript math library was used to perform matrix operations in the simulator module. Comments to the MathJS library can be found in appendix A.10.

A demonstration of the simulator was performed. The resulting plots of the simulated values is shown in figure 12.2, 12.3, 12.4 and 12.5.

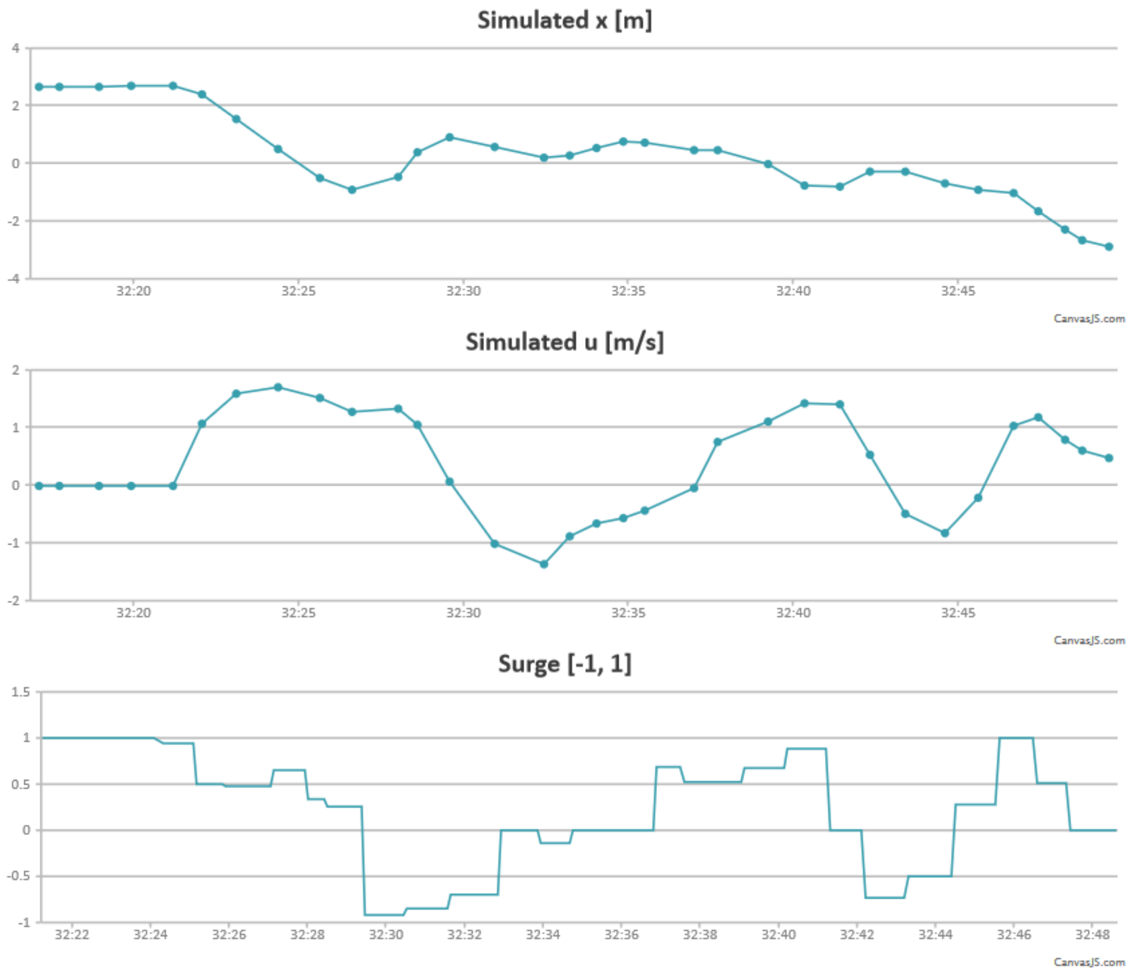


Figure 12.2: Simulated x, u and surge

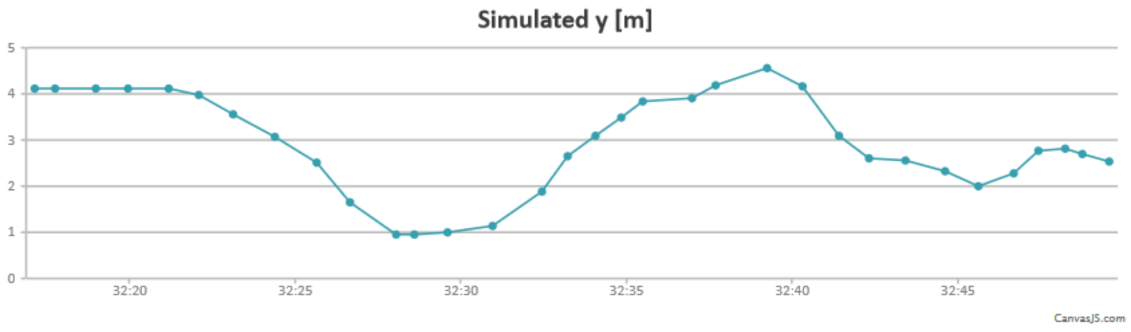


Figure 12.3: Simulated y

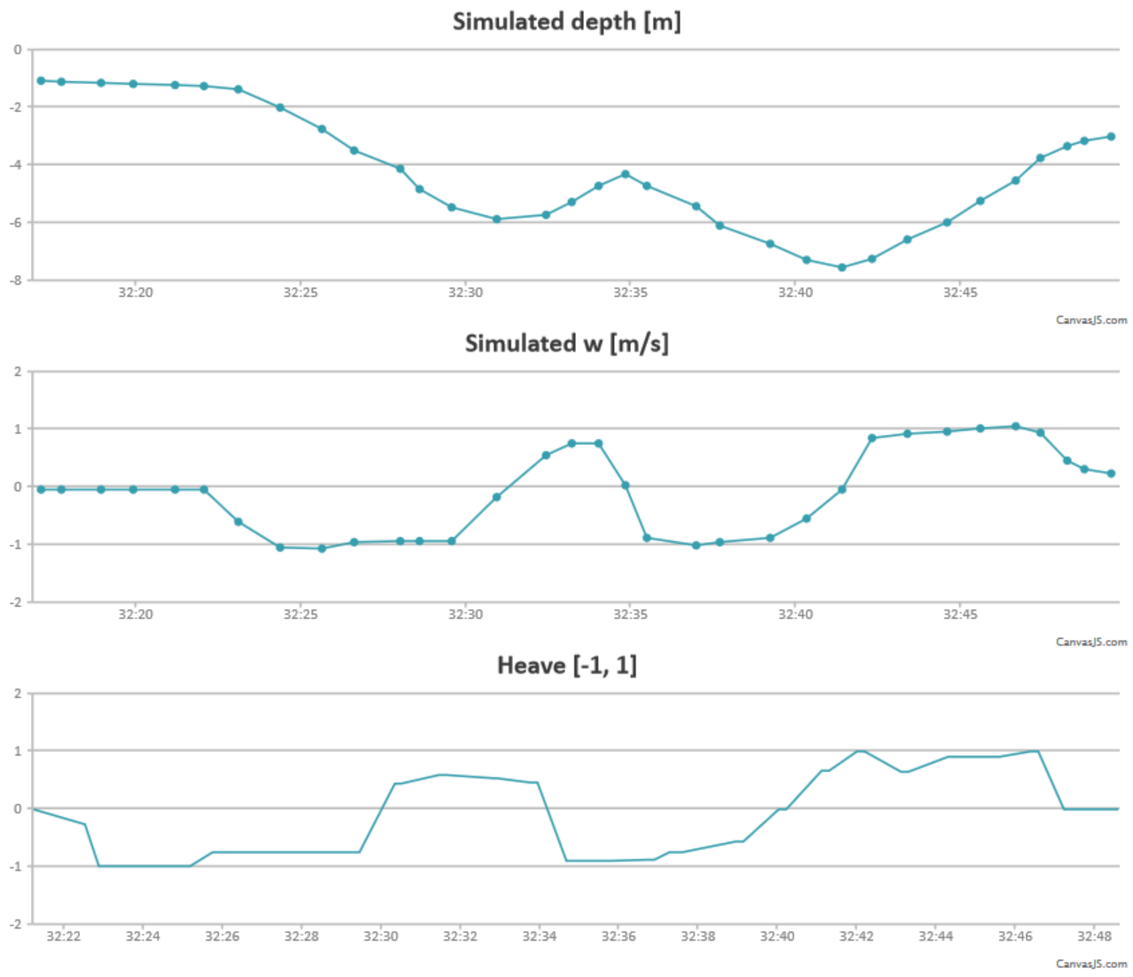


Figure 12.4: Simulated depth, w and Heave

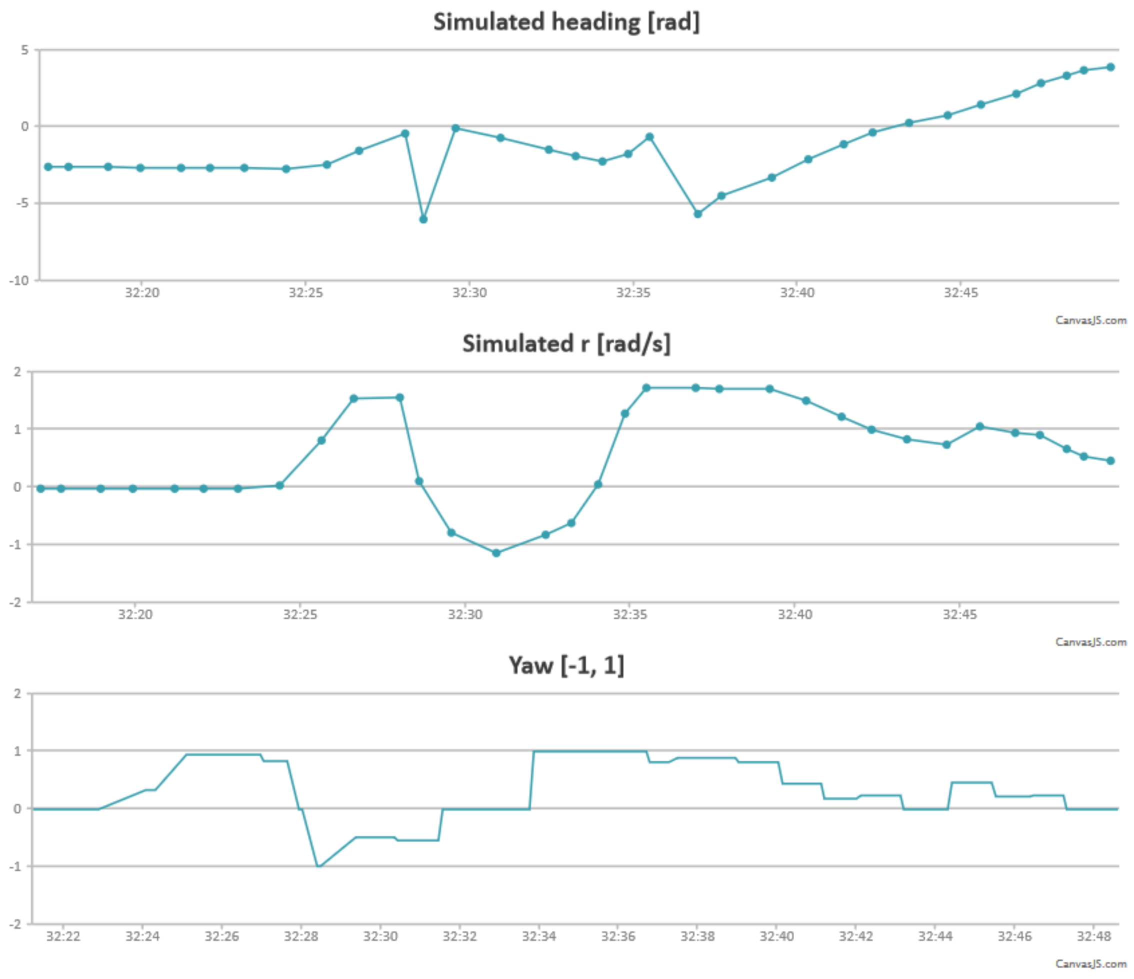


Figure 12.5: Simulated heading, r and Yaw

13 Testing

As specified in the scope of work, testing of the developed software was planned and executed.

The tests were performed at Marine Cybernetics lab at Tyholt. Batteries and 12V dc power supply was used on all tests. Power supply max current output was set to 3.13A.

Preface and preliminary test: There is an available option to turn on smoothing of thrust signal in motor drivers. Prior to the test, a land test was performed without smoothing. The software crashed repeatedly when performing rapid changes in thrust direction. The was a known issue, described in chapter 3.4.

13.1 Internal acceptance test, 21st of May 2015

Test setup: Ethernet cable was used to create a direct link between topside pc and ROV. The smoothing parameter of thrust signal was used on all tests.

Test scope:

- Control functionality
- User interface
- Voltage and current
- Lights

Test description	**Result**	**Comment**
Current and voltage test. Full speed forward and up. On land	1.41A, 12V	Voltage change was smooth
Current and voltage test. Full speed backward and down. On land	1.37A, 12V	Voltage change was smooth
Current and voltage test. Full speed forward and up. In water	3.13A, 10.9V	Voltage change was smooth
Current and voltage test. Full speed backward and down. In water	3.13A, 10.9V	Voltage change was smooth
Forward	OK	
Move backward	OK	
Move up	OK	
Move down	OK	
Turn starboard	Not OK	Turned port
Turn port	Not OK	Turned starboard
Manual control mode	-	Not implemented
Touch screen input	-	Not implemented
Camera streaming	OK*	Quality and consistency fine. Image needs to be rotated -90 degrees
Turn on lights	OK*	Dimming functionality wanted. A random delay

Test discussion and subsequent corrections

An error in the thrust allocation was found and corrected as a result. Upon further inquiry into the delay in lighting, it was discovered that the command to turn on or off lights only was executed when the next thrust command was sent. Functionality was added to re-send the previous thrust command after a light command, such that lighting happened instantly.

13.2 Factory acceptance test, 28th of May 2015

Test setup: Ethernet cable to router. The router broadcasted the signal locally and a laptop was used to access the ROV through WiFi.

Test scope:

- Motion control mode
- Manual control mode
- Touchscreen as input using Ipad
- Joystick as input using XBOX one controller
- Remote control over WiFi.
- Lights
- Camera

Test description	**Result**	**Comment**
Starboard thruster forward	OK	
Starboard thruster backward	OK	
Port thruster forward	Not OK	Vertical thruster up
Port thruster backward	Not OK	Vertical thruster down
Vertical thruster up	Not OK	Port thruster forward
Vertical thruster down	Not OK	Port thruster backward
Move forward	Not OK	Turned starboard
Move backward	Not OK	Turned port
Turn starboard	Not OK	Moved backward
Turn port	Not OK	Moved forward
Move up	OK	
Move down	OK	
Lights	OK	
Camera	OK	

Test discussion and subsequent corrections

The first six tests were performed in manual control mode, the following six in motion control mode. The thrust tests were performed with keyboard and joystick with identical results. Ipad was tested, but no thrust input worked here. It was discovered that when pressing and holding on an image on an Ipad, the browser will pop up a prompt to save the image. This event blocked the intended use of the thrust command images. Listing 12 shows how this problem was handled in the client.

Listing 12: Deny image save

```
document.body.style.webkitTouchCallout="none";
```

As the tests were conducted it was suspected that there was a sign error in the control module causing a reversal in the port thruster. It was discovered that the thrusters responded differently to the function used to control the ROV. go(1700, 1700, 1700, 1) would make the starboard and vertical thruster produce forward thrust, while the port thruster produced negative thrust. Upon further inquiry, it was revealed that this was due to a wiring error in the hardware. Software corrections were implemented to compensate for this.

The results of test 3 through 6 was found to be a syntax error and corrected.

13.3 Customer acceptance test, 4th of June 2015

Test setup: Same as in the factory acceptance test.

Test scope:

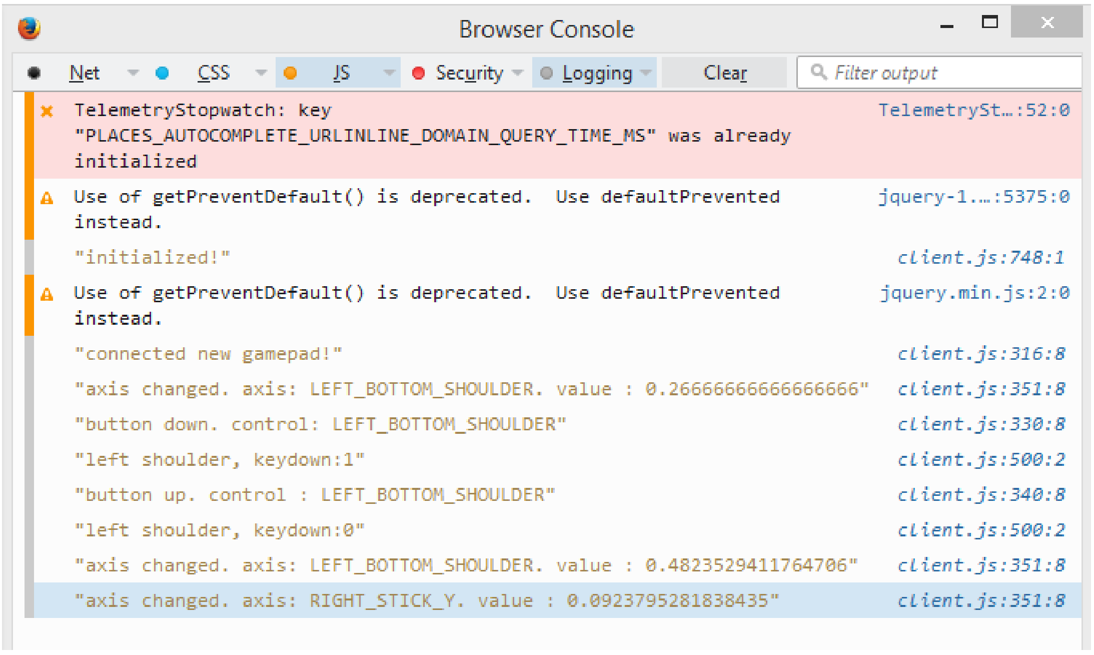
- Manual control.
- Motion control.
- Touchscreen as input method using Ipad and Samsung S3 smart phone.
- Transfer of control.
- Developer interface.
- Simulator.

Test description	Result	Comment
Manual control joystick	Not OK	The ROV occasionally does not stop in the vertical direction
Motion control joystick	OK	
Manual and motion control keyboard	OK	
Manual and motion control Ipad	OK*	The rotation of the camera image not supported in the Safari browser. Change image text for manual
Manual and motion control Samsung S3	OK	
Transfer of control	OK	
ROV stop when master disconnects	OK	
Plotting	OK*	Detection of frequent wild points. Functionality to reset graphs wanted
Logger	OK*	Writing the log to file is wanted
Simulator	OK	

Test discussion and subsequent corrections

The laser was suppose to be tested, but the hardware was not available. Simulator run plots were illustrated in chapter 12. Support for camera rotation in the Safari browser was added, shown in chapter 10. A wild point filter was added to reduce the amount of wild points in the measurement plots. Implementation of filter is found in chapter 8.3. Functionality to reset the graphs were added, explained in chapter 7.2. The pictures used for manual control mode with touch input was changed. ”Thruster 1”, ”Thruster 2” and ”Thruster 3” was changed to ”port thruster”, ”vertical thruster” and ”starboard thruster”. The vertical thruster image text was changed from ”forward” and ”backward” to ”up” and ”down” to clarify their functionality. Logging to file was left as further work.

The error in manual control mode with joystick input was found to be caused by an error in the gamepad.js library. The library emits two kinds of events when buttons are pushed; an ”axis-changed” event when a stick is moved and a ”button up/down" event when a button is pressed or released. The left and right bottom shoulder buttons are used for controlling the vertical movement of the ROV in manual control mode. The bottom shoulder buttons emit both the axis-changed event and the button up/down event. The error occurs because the order the two events are emitted is non-deterministic. The ROV did occasionally not stop because it first received the event that the button was released, then it received the event that it was halfway released. Figure 13.1 shows the log from such an event. When the button is pushed, the axis changed is triggered before the button down event. When the button is released, the button up is triggered before axis changed. Solutions to this error is left as further work.



```
TelemetryStopwatch: key                                                  TelemetrySt…:52:0
"PLACES_AUTOCOMPLETE_URLINLINE_DOMAIN_QUERY_TIME_MS" was already
initialized
Use of getPreventDefault() is deprecated.  Use defaultPrevented          jquery-1.…:5375:0
instead.
"initialized!"                                                           client.js:748:1
Use of getPreventDefault() is deprecated.  Use defaultPrevented          jquery.min.js:2:0
instead.
"connected new gamepad!"                                                 client.js:316:8
"axis changed. axis: LEFT_BOTTOM_SHOULDER. value : 0.26666666666666666"  client.js:351:8
"button down. control: LEFT_BOTTOM_SHOULDER"                             client.js:330:8
"left shoulder, keydown:1"                                               client.js:500:2
"button up. control : LEFT_BOTTOM_SHOULDER"                              client.js:340:8
"left shoulder, keydown:0"                                               client.js:500:2
"axis changed. axis: LEFT_BOTTOM_SHOULDER. value : 0.4823529411764706"   client.js:351:8
"axis changed. axis: RIGHT_STICK_Y. value : 0.0923795281838435"          client.js:351:8
```

Figure 13.1: Console log of gamepad.js bug

14 Discussion

A table element was used for the layout in the Operator interface. According to [20], it is bad practice to do this. However, it was stated in the scope of work that the focus should be on the developer interface and the development of functionality, rather than visual appearance. It is left as further work to improve the client form and layout.

The remote control functionality was only demonstrated over local WiFi connection. In theory, it should be minimal modifications to expand to global connection over internet. The main point of the scope of work was to facilitate the design and implementation such that remote control over internet is possible. This was demonstrated with local connection. It is left as further work to troubleshoot this.

The testing revealed a bug in gamepad library used for joystick control as described in chapter 13. Solving this problem with logics can be difficult due to the non-deterministic nature of the problem. The state of the buttons can be polled and checked when an axis-changed event is received, but since the bug is not consistent, this can create problems with false positives. A simpler solutions is to avoid the problems by using different components of the joystick as input. As the two sticks on the joystick is already used for the starboard and port thrusters, there is no other available inputs with continous signal range, so buttons such as A and B have to be used. The manual control mode is not the main control mode, and the use of binary inputs for one thruster here does not cause a great deterioration in performance. It is left as further work to implement a solution to this problem.

Implementing a full DP mode on the ROV may not be feasible as the ROV does not have any absolute position measurements. The sub-functionality of constant depth autopilot and constant heading autopilot is certainly feasible due to depth sensor and magnetometer measurements. Currently the only way to obtain position is by integrating IMU measurements, which introduces drift in state estimation. Camera and laser based methods such as Simultaneous Localization and Mapping (SLAM) can be considered. However, this technique is quite advanced, and it is typically implemented on sophisticated systems with expensive sensors. Expansion of the ROV sensor suite can also be considered, but the sonar technology used for underwater positioning is typically expensive.

If full DP is not implemented, the DP button can be used for simply holding heading and depth, or it can trivially be renamed or removed.

Since it shall be possible to control the ROV over the internet, it may be subjected to malicious attacks. The remote control was implemented without particular considerations on security. As the system is now, the risk and consequence of an attack may not be large. This may be considered in the future.

The software solution developed as a whole, and Node in particular, were found to be satisfactory choice for the system. There were no particular difficulties encountered with Node during the development. As the OpenROV used similar solutions, some parts of the code is similar. This is mostly restricted to the camera module. The coding style in the OpenROV is different than the one used. OpenROV have adhered to the loose and dynamic nature of the JavaScript language, while a stricter and more structured style as often seen in C++ and Java code, have been used for this work. Functions have been small and modular. OpenROV have used bigger function with a wider scope. OpenROV also employed widespread use of event emitters within the web server. Regular function calls have been used instead of event emitters when possible.

15 Conclusion

Design of hardware and software solutions for the ROV Neptunus was performed. This includes review of frameworks and software tools. A Node.js web server was implemented on a BeagleBone computer running Linux Ubuntu. The Node.js specific framework Express.io was used. A user interface consisting of an operator interface and a developer interface was implemented in JavaScript. The operator interface contains methods for controlling the ROV with the use of keyboard, touchscreen and joystick. The operator interface contains functionality vital in development such as logging and plotting.

A control module was implemented to control the ROV with either motion control or manual control. This included a thrust allocation procedure for the motion control mode. A camera module for streaming of the video was implemented. The mjpg-streamer application was used to stream the video from the camera to the client. A simulator module was implemented to facilitate parallel development. Remote monitoring and control over WiFi was implemented. This included methods for taking and giving control, as only one operator can be in control at any time. All the implemented functionality was tested.

16 Further work

- DP functionality with the sub-functionality of automatic hold on depth and heading. As stated in the discussion section, position control in the xy-plane may not be feasible. This should be considered before attempting to implement full DP.
- Implementation of a dynamic filter such as Kalman Filter.
- Automatic logging to file functionality.
- Move the wild point filter to the navigation module to remove errors before measurements enter the dynamic filter. The wild point filter should be improved in order to remove more of the wild points. This can be done by checking every measurement against previous measurement values. This problem also motivates improvement of the module that reads the buffer from the serial connection. This can be done in conjunction with the task of writing new drivers for the motors and sensors.
- Troubleshoot the problem with remote control over the internet.
- Turning lights on and off with joystick control.
- Improve client layout and appearance. Review tools for layout of web site such as ReactJS or AngularJS. Change the touch input to more elegant solutions such as swiping instead of clicking on images.
- Improve simulator module to six degrees of freedom. Incorporate thruster and umbilical effects. This can include drag force from umbilical, thruster losses and thruster rise time. Implement changes to facilitate stand-alone simulator use. The current software contains references to OS native code in the serialport library used in SerialHandle.js crashing the application if run on any OS that is not ARM-Linux. This can be solved by specifying command line arguments when starting the application. This is done in the same manner as argc/argv in C programs. It can be considered to include the option of manual thrust mode in simulation mode.
- Integration with Oculus rift. Design and implementation of Oculus Rift control for ROV was done in [21]. Analysis of joystick control for ROVs with a constant-jerk reference model and filter-based reference models was done in [22]. The output from the oculus rift system can be interpreted similarly to a 3-DOF joystick, and the control theory from [22] can be used.

References

[1] Jostein Follestad, Fredrik Sandved, and Eirik Valle. Low cost rov design, based on testing, simulations and analysis of openrov, 2014.

[2] May 2015 web server survey. http://news.netcraft.com/archives/2015/05/19/may-2015-web-server-survey.html.

[3] Coffeescript on node.js. https://books.google.no/books?id=Oda-MgEACAAJ&dq=nodejs&hl=en&sa=X&redir_esc=y.

[4] http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js, year not stated, but not before 2013.

[5] http://blog.caustik.com/2012/08/19/node-js-w1m-concurrent-connections/, 2012.

[6] http://blog.shinetech.com/2013/10/22/performance-comparison-between-node-js-and-java-ee, 2013.

[7] http://expressjs.com/starter/hello-world.html.

[8] http://www.rovmarine.it/en/home-eng/14-not-categorized/16-the-history-of-rovs.

[9] http://www.marinetech.org/regional-contest.

[10] Thor I. Fossen. Handbook of marine craft hydrodynamics and motion control, 2011.

[11] http://www.ros.org/is-ros-for-me/.

[12] https://github.com/OpenROV.

[13] https://github.com/OpenROV/openrov-software/blob/master/docs/OpenRovSoftwareArchitectureOverview.pdf.

[14] https://www.ffmpeg.org/.

[15] https://code.google.com/p/mjpg-streamer/.

[16] https://www.raspberrypi.org/forums/viewtopic.php?f=43&t=45178, 2012.

[17] https://wolfpaulus.com/jounal/embedded/raspberrypi_webcam/.

[18] Dnv gl rules and classifications of ships / high speed, light craft and naval surface craft, 2011.

[19] Fredrik Sandved. Remote control and path-following for c/s enterprise 1 and rov neptunus(msc), 2015.

[20] https://css-tricks.com/complete-guide-table-element/, 2013.

[21] Eirik Valle. Marine telepresence system (msc), 2015.

[22] Fredrik Dukan. Rov motion control systems (phd), 2014.

[23] http://derekmolloy.ie/beaglebone/getting-started-usb-network-adapter-on-the-beaglebone/, 2013.

[24] http://stackoverflow.com/questions/19481415/share-the-internet-access-from-laptop-to-beaglebone-black-and-then-access-it-thr, 2014.

[25] http://www.tomshardware.co.uk/faq/id-1925829/change-default-internet-connection-sharing-address-range.html, 2013.

Appendices

A

A.1 Content of the zip

The .zip file delivered contains the software developed. This does does not include Arduino code or the mjpg-streamer. The public, routes and bin folders in the zip is not used in the project. They were generated automatically by Express.io.

A.2 Accessing the BeagleBone

PuTTY is a program that can be used to log on to the BagleBone with an ethernet or USB connection. The ethernet connection on the BeagleBone has a default static IP adress of "192.168.254.1" and the USB IP is also static with an address of "192.168.7.2". This is when the OpenROV OS image is used. When accessing the BeagleBone through PuTTY, the username should be "rov".

A.3 Uploading code to the BeagleBone

Uploading code from a windows computer to the BeagleBone was done using the open-source program WinSCP. WinSCP has a graphical user interface that supports drag-and-drop file transfer. It is fairly straightforward to use, but directory permission on the BeagleBone restricts eligible destination directories. The rov user does not have permission to write to most directories. Changing ownership of folders is possible, but discouraged for users not seasoned in a Linux environment. To view the permission of files within the working directory, the command "ls -l" can be used. This will display ownership of files as well as read, write and execute permission. The "/opt/scripts" directory is one of those which is owned by the rov user.

A.4 Starting the web server

The web server was started by logging on to the BeagleBone with PuTTY, navigation to the /opt/scripts/dev-master folder and executing the command: "node app.js".

A.5 Achieving internet access on the BeagleBone

Internet access on the BeagleBone was achieved by sharing a laptops internet connection through ethernet. Note that this is not for remote control purpose, as outside entities cannot connect to the ROV as it is hidden behind the pc. The purpose of this was mainly to download an npm package called serialport directly to the BeagleBone. When this package is downloaded, native files are automatically compiled and included. Consequently, if this is downloaded to a windows machine, windows native files are included. Using a Linux laptop was also not an option, as the BeagleBone required ARM-Linux files.

Internet access on the BegleBone can be achieved through an ethernet or USB connection. Instructions for accessing the internet on the BeagleBone with a USB connection can be found in [23]. A USB connection may be prefered over ethernet according to [23], due to easier networking protocol. Unfortunately, there were errors in the particular hardware used preventing the usage of USB. Instead, internet access through ethernet connection on the BeagleBone was achieved. The following paragraph contains instructions on this procedure on a windows pc.

The procedure found in [24] was followed. However, when the internet was shared, Windows changed the IP address of the local connection automatically. Windows have reserved a collection of IP addresses specifically for that use. Modifications in the Windows registry in accordance with [25] was made such that the assigned IP address coincide with the same subnet as the BeagleBone. The IP address of the BeagleBone could have been changed to accommodate the Windows one, but it was preferred not to edit the networking settings of the BeagleBone. The reason for this was that the ethernet connection was the only available connection method at the time, and editing the networking settings in the BeagleBone meant risking losing the only connection. When the remote control functionality was developed, an addition connection to an HDMI screen had been acquired.

There were variables existing in the Linux path environment related to proxying. These had to be removed from the environment in order for a functioning internet connection. The npm system also had some proxy variables set that had to be removed in order to use.

A.6 Hardware vulnerability

It was experienced that a BeagleBone board stopped working entirely. No connection could be established through either USB or ethernet. After re-flashing the OS image to the BeagleBone, it functioned correctly.

A.7 Video stream setup

Setting up the mjpg-streamer was relatively straightforward. It was downloaded and linked to path. The video device, the input path and the output path that mjpg-streamer must have appropriate permission levels for the application to use them. This was done. Alternatively, the mjpg-streamer can be run as super user with the sudo command.

A.8 Displaying data sent over serial connection

The data sent over the serial port between the Arduino and the BeagleBone can be displayed in the console while operating the BeagleBone. The command for this is: cat /dev/ttyO1. When driving the ROV with the OpenROV software, some commands were observed that can also be found in the API [13]. The following was observed: ”cmd:thro(25)”, ”cmd:yaw(-40)” and ”cmd:lift(25)”. These commands appeared to be thruster commands as the corresponding thrusters were simultaneously activated. The parameters used for the function calls are inconsistent with the range that is listed in the API. Attempts were made to reproduce the response from these commands. Recreation of the signals over the serial port (not using the OpenROV software) was successful, but there was no thruster response. The ”go()” function described in chapter 8.4 was simultaneously tested (with success) to disregard hardware failure as a cause.

There were no observations of the ”go()” function in the console log when the cat command was used. However, the object used for measurements was observed. The output from the cat command was in this case consistent with the API in [13].

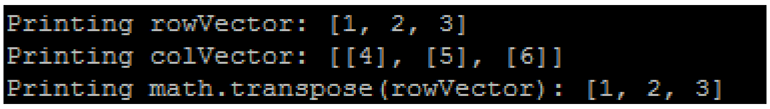
There were observations of two signals on the port ”motor” and ”mtargs”, these were suspected to be status updates on the motors, and not commands.

A.9 Troubleshooting unresponsive motor

The motors on the ROV are connected to electronic speed controllers. These have physical switches that can easily be flicked by accident. This should be checked if motors suddenly do not respond.

A.10 MathJS

Math.js or MathJS is a mathematics library. It has support for matrix operations including transposing. Transposing a row vector, however, will return the row vector and not a column vector. This is illustrated in the screenshot in figure A.1. Upon use, an undeclared variable was use in a multiplication function. Instead of throwing an error, it returned a matrix consisting of undefined elements.



```
Printing rowVector: [1, 2, 3]
Printing colVector: [[4], [5], [6]]
Printing math.transpose(rowVector): [1, 2, 3]
```

Figure A.1: MathJS