AALBORG UNIVERSITY

STUDENT REPORT

ED5-1-E16

Modelling and Control of an Underwater Vehicle with Focus on Depth Control

Students:
Allan Gjerlevsen
Emil Már Einarsson
Thomas Thuesen Enevoldsen

Supervisors:
Christian Mai
Simon Pedersen

December 19, 2016

AALBORG UNIVERSITY

STUDENT REPORT

School of Information and Communication Technology
Niels Bohrs Vej 8
DK-6700 Esbjerg
http://sict.aau.dk

Title:
Modelling and Control of an Underwater Vehicle with Focus on Depth Control

Theme:
Scientific Theme

Project Period:
Fall Semester 2016

Project Group:
ED5-1-E16

Participant(s):
Allan Gjerlevsen
Emil Már Einarsson
Thomas Thuesen Enevoldsen

Supervisor(s):
Christian Mai
Simon Pedersen

Copies: 3

Page Numbers: 85

Date of Completion:
December 19, 2016

Abstract:

This report will describe the mathematical modelling, controller development and implementation of a modern control system on an UUV. The chosen platform was the OpenROV 2.8, using the OpenROV IMU/Pressure sensor for acquiring environmental data. The main focus is to develop a depth controller using state-space with the pressure data to determine current depth. Based on the dynamics of the ROV and thrusters a general mathematical model is created describing the movement in the 6 degrees of freedom, which are later simplified to only the dynamics related to the depth control. During the system identification the relevant parameters for the mathematical model were determined and linearised accordingly. Based on time-domain specifications control gains and estimator gains are determined and simulated with both integral control and reference scaling for reference tracking. The theoretical controllers are simulated and compared with the linear/non-linear versions of the plant. Finally the control implementation is done using C, where the resulting controller yields acceptable reference point tracking within $\pm$3cm.

Contents

Preface **ix**

Acronyms **x**

1 Introduction **1**

1.1 Introduction 1

2 Problem Description **3**

2.1 OpenROV 2.8 3

2.1.1 Specifications 4

2.2 Control Methods 4

2.3 Problem Delimitation 5

3 Physical Platform **6**

3.1 Hardware 6

3.1.1 Acrylics 6

3.1.2 Internal Electronics 7

3.1.3 Tether and Communication 9

3.1.4 Thrusters 11

3.1.5 IMU/Compass/Depth Module 12

3.1.6 Assembly 13

3.2 Logic Analysis 14

4 Mathematical Modelling **16**

4.1 ROV Dynamics 16

4.1.1 Up and Downward Motion 17

4.1.2 Forward and Backwards Motion . . . 18
4.1.3 Rotational Movement . . . 19
4.2 Thruster Modelling . . . 22
4.2.1 Brushed DC Motor Model . . . 23
4.2.2 Brushless DC Motor Model . . . 24
4.2.3 Propeller Thrust and Torque . . . 26
4.3 State-Space Representation . . . 27

5 System Identification 28
5.1 Accelerometer Data . . . 28
5.2 Thruster Test . . . 30
5.2.1 Thruster Coefficients . . . 34
5.3 Depth Experiments . . . 37
5.4 Depth Drag Coefficient Determination . . . 39

6 State Space 41
6.1 Model Simplification . . . 41
6.2 State-Space Control . . . 42
6.3 Full-state Feedback . . . 43
6.4 Observer . . . 45
6.5 Integral Control . . . 47

7 Controller Development 48

8 Non-linear and Linear Model Comparison 53
8.1 Open Loop . . . 53
8.2 Estimator with Reference Scaling . . . 55
8.3 Estimator with Integral Control . . . 57

9 Implementation 59
9.1 Discretization and Difference Equations . . . 59
9.2 C Implementation . . . 62
9.3 Testing . . . 64

9.3.1 Initial Estimator and Integral Control 64
9.3.2 Improved Estimator and Reference Scaling 65

10 Discussion **69**

11 Conclusion **71**

References **72**

12 Appendix **75**
12.1 Pre-Dive Check List 75
12.2 Post-Dive Check List 76
12.3 Experimentation Pictures 77
12.4 Raw Propeller Test 78
12.5 Code 79
12.5.1 Data Logging C 79
12.5.2 Thruster Test C 80
12.6 Controller Implementation in C 81

Preface

The project entitled *Modelling and Control of an Underwater Vehicle with Focus on Depth Control* was made by three students from the Electronics and Computer Engineering programme at Aalborg University Esbjerg, for the P5 project during the fifth semester.

We would like to thank our supervisors Christian Mai and Simon Pedersen for their exceptional supervision and guidance throughout the project. Especially related to the development and implementation of the modern control theory.

We would also like to thank Auraskolen Hjerting, Esbjerg for giving us access to their swimming pool to conduct our larger scale experiments. Finally we would like to thank Natasha Bani-Sadr for providing us with underwater footage from our experimentation.

Aalborg University, December 19, 2016.

Allan Gjerlevsen
<agjerl13@student.aau.dk>

Emil Már Einarsson
<eeinar14@student.aau.dk>

Thomas Thuesen Enevoldsen
<tten14@student.aau.dk>

Acronyms

SNAME The Society of Naval Architects and Marine Engineers

UART Universal Asynchronous Receiver/Transmitter

MIMO Multiple Input Multiple Output

BLDC Brushless Direct Current

SISO Single Input Single Output

6DoF 6 Degrees of Freedom

PID Proportional-Integral-Derivative

UUV Underwater Unmanned Vehicles

ROV Remotely Operated Vehicle

ESC Electronic Speed Control

IMU Inertia measurement unit

I2C Inter-Integrated Circuit

GUI Graphical User Interface

CRC Cyclic Redundancy Check

KVL Kirchhoff Voltage Law

EU European Union

IO Input/Output

SSH Secure Shell

Chapter 1

Introduction

1.1 Introduction

The focus of this project will be on implementing a control system for a Remotely Operated Vehicle (ROV). The main focus is to design and implement a controller for the depth where the user can choose a desired set point. The controller will be designed according to some specified time-domain requirements. The particular ROV platform used is the Open-Source OpenROV project [1], where in this case the OpenROV 2.8 was available for assembly. In order to create a control solution for the said ROV it is necessary to create a mathematical model of the system and thereafter identify the different model parameters in order to simulate and design controllers for the ROV.

ROVs have in recent years risen in popularity, especially flying drones have gained massive traction and have become a recent trend within research and hobbyists. Underwater Unmanned Vehicles (UUV) are also becoming more and more favoured since they have the capability to reach depths deeper than divers are able to. Humans who wish to do deep dives need to go through decompression stops as they ascent in their dive. This takes a great amount of time and increases the missions preparation drastically, but is necessary due to safety reasons for the divers. Therefore, it is an inconvenience sending humans to the depth of the ocean. One of the problems with the control of underwater vehicles is that underwater there is a lack of usable external signals that can be used for positioning. The main issue faced is with the difficulty of transmitting radio waves through water, and it is increasingly more difficult in oceans, due to the high concentration of salt in the ocean. Water is very conductive and in the case of sea water very absorptive [2]. Acoustic technology has been used as a mean to communicate with underwater vehicles, but high costs are associated with this method. This results in ROVs utilising tethered connections, meaning that there is a wire connecting the ROV and base-station. The tether service numerous purposes, some of them being to send and receive control commands, transmit environmental and telemetry data, images, video feeds, battery voltages etc [3].

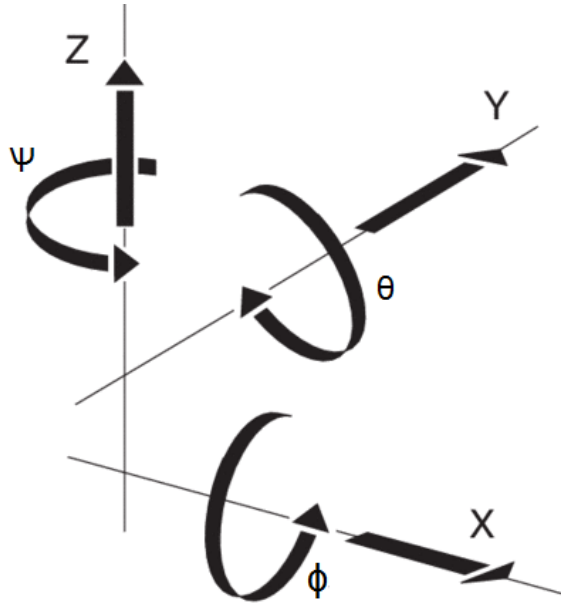


Figure 1.1: 6 degrees of freedom [4]

When moving in water, the available movements are similar to those available in space, and can be seen in figure 1.1. The ROV is said to have 6 Degrees of Freedom (6DoF), where three of the movements are linear along the x, y and z-axis and the other three rotational about those axis. Names associated with the 6 available movements can be seen in table 1.1 and named based the The Society of Naval Architects and Marine Engineers (SNAME) standard. This naming scheme will be used throughout the report to indicate the direction and type of movement [4] [5].

Movement	Movement Notation
Motion in x-direction	Surge
Motion in y-direction	Sway
Motion in z-direction	Heave
Rotation about the x-axis	Roll
Rotation about the y-axis	Pitch
Rotation about the z-axis	Yaw

Table 1.1: Table of notation for the ROV movement [5]

Chapter 2

Problem Description

2.1 OpenROV 2.8

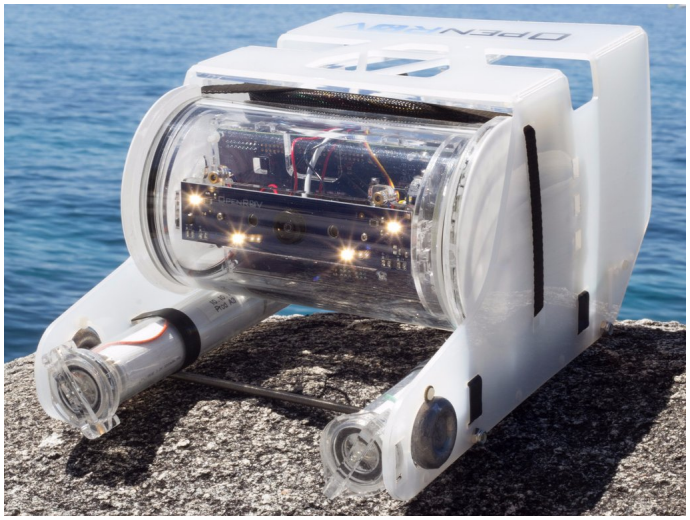


Figure 2.1: OpenROV 2.8 [1]

As mentioned in the introduction, this project concerns the mathematical modelling and control implementation for an ROV. The ROV in question is the OpenROV 2.8 which we tasked ourselves with assembling, since the University had this kit available. This particular ROV is fully controllable for surge, heave and yaw movement.

The OpenROV is equipped with three thrusters, a single thruster with a two bladed propeller for heave, and two thrusters with three bladed propellers for surge and yaw movement. Besides the base OpenROV kit an addon in the form of the OpenROV Inertia measurement unit (IMU)/Compass/Depth Module will be installed, which has been purchased alongside the OpenROV kit. This module provides depth measurements, current heading, water temperature and accelerometer/gyrometer values.

2.1.1 Specifications

The following information is a selection of the relevant specifications taken from the official OpenROV product page [6].

Physical Specifications

- Weighs 2.7kg
- 30cm x 20cm x 15cm
- Neutrally buoyant

Performance Specifications

- Maximum depth 100m
- Maximum forward speed 2 knots (1m/s)
- Operational within water temperatures $-10°C$ to $50°C$

Instrumentation

- BeagleBone Black and Arduino MEGA microprocessors
- External Inter-Integrated Circuit (I2C) bus
- IMU and depth sensor

2.2 Control Methods

There are two main branches within control theory, these being classical and modern control theory. Classical control theory often deals with Single Input Single Output (SISO) systems, using transfer functions and the frequency domain. A commonly used controller within classical control theory is the Proportional-Integral-Derivative (PID) controller. Modern control theory, also known as state-space, uses differential equations in the time domain. State-space is advantageous to use when the system that needs controlling has Multiple Input Multiple Output (MIMO). State-space uses the state-variable method for describing differential equations. Using this method, all the differential equations are written as a set of first-order differential equations in terms of the vector valued state of the system, where the solution is visualized as trajectory of the state vector in space. Due to the ROV being a MIMO system, the control method of choice will be state-space, due to its advantages when dealing with MIMO systems.

2.3 Problem Delimitation

In order to ensure the project is within the desired scope, the project is delimited in order to fulfil specified success criteria. The following requirements are used to determine the success of the project as specified below.

- Creation of a mathematical model describing the 6DoF
- Design of a state-space controller for the depth
- Implementation of a state-space controller for the depth

By creating a general 6DoF model for the ROV it is possible to re-implement said mathematical model on different ROVs, by modifying the general coefficients. The OpenROV will be modelled with the assumption that it can be fully controllable in all 6DoF. The design of the control system will have its main focus on controlling the depth and will theoretically developed and simulated in order to create a controller suitable for the desired time-domain specifications. The specific type of state-space controller will be determined later in the report since this will depend on the available sensor information and its quality. The reasoning behind focusing on depth control, and not controlling the motion in the remaining directions, is due to the limited scope in order to focus on the controller implementation in a single direction more thoroughly.

Once the OpenROV kit has been assembled, the implementation of the control system can take place and testing of the real-life system can be performed. The above time-domain specifications will act as design criteria for the development of a controller and also as the testing requirements, and be used to judge the performance of the theoretical controller during simulation, but also serve as base for comparison with the implemented controller.

- Maximum overshoot: 10%
- Steady-state error: 1%
- Settling time: 10 seconds

Chapter 3

Physical Platform

3.1 Hardware

3.1.1 Acrylics

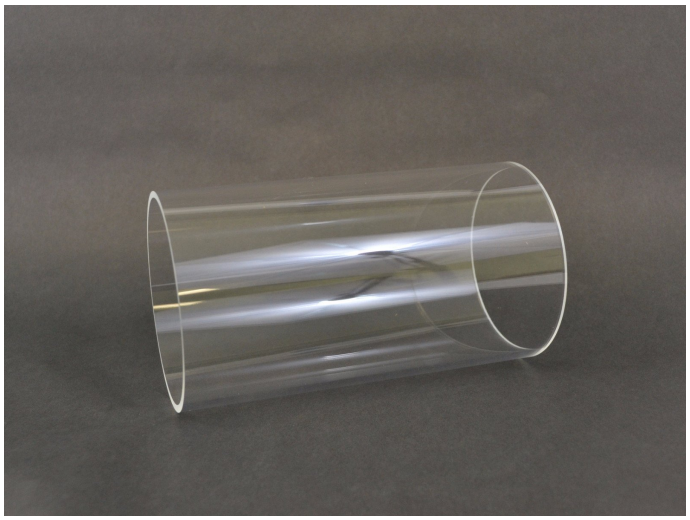
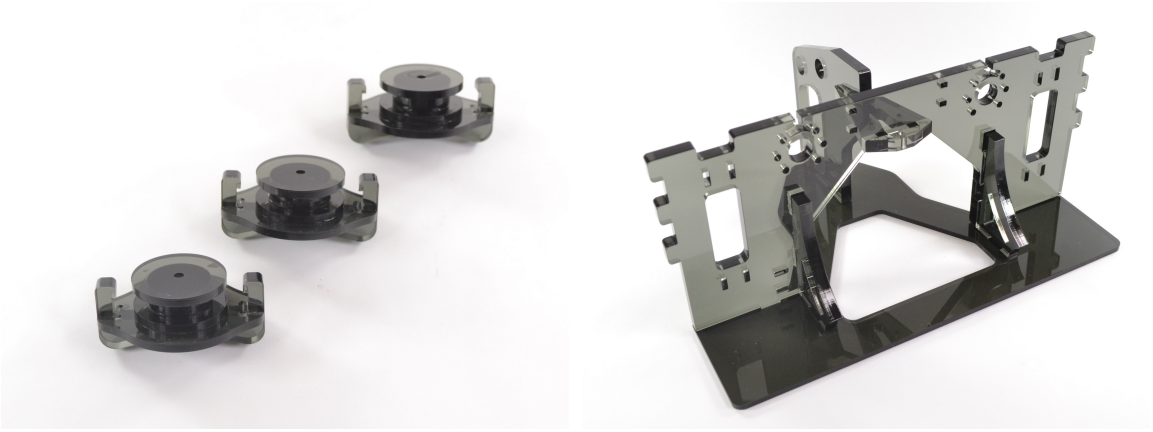


Figure 3.1: OpenROV 2.8 main tube [7]

The ROV uses a lot of acrylics and plastics for the main construction of its body. One of the benefits of using acrylics is that it makes the assembly process easier, keeps the cost low and creates more visibility to the insides of the ROV itself. Figure 3.1 shows the main tube which stores all the internal electronics. Besides the main tube, the main frame, battery tubes and end caps are also made out of acrylic.



(a) Batteryboc endcaps [7] **(b)** Acrylic frame [7]

Figure 3.2: Parts of the acrylics

Some major disadvantages by using acrylic are that it is fragile in terms of impact resistance and the acrylic also acts as an isolator which may influence heat sensitive electronics. Other possible materials for the ROV components could be aluminium or other metals. Aluminium features better cooling properties and in general would create a more sturdy housing. Using a full aluminium case would increase the manufacturing cost and also complexity since the main tube must feature a slot for the camera. Using a metal body also increases the overall weight of the vehicle.

3.1.2 Internal Electronics

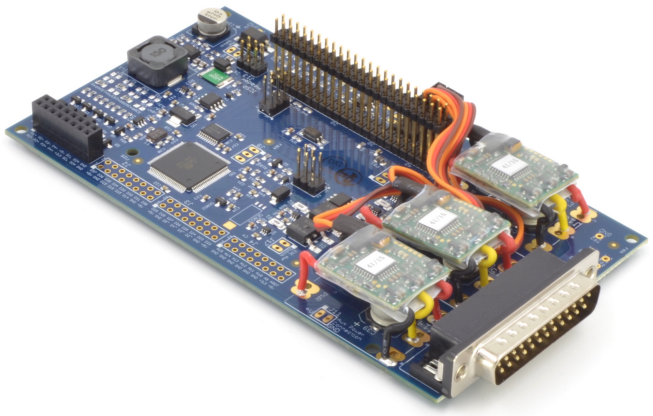


Figure 3.3: OpenROV 2.8 control board [7]

The OpenROV uses its own custom controller board. This controller board serves the purpose of actuating the thrusters and also interfacing with the sensors. The board features an embedded Arduino Mega 2560 that acts as addi-

tional Input/Output (IO) for external devices and add-ons. The board comes pre-programmed with OpenROV firmware and an Arduino Bootloader. This firmware is programmed to communicate with the BeagleBone, its main task is to send the IMU data to the BeagleBone and activate the thrusters depending on the commands coming from the BeagleBone over serial communication. The controller board communicates with the IMU using I2C and with the BeagleBone using serial. Since the controller board spends most of its time in a watertight enclosure, the control board also features an embedded power switch that allows the ROV to be turned on and off remotely through the tethered connection. Due to the limited space inside the main tube a well built and size-optimized control board is crucial. The controller board attaches to one of the end caps for the main tube using a Male DB-25 connector, which is visible on figure 3.3, knowing the pin out of the end caps connector and it is possible to design a custom controller board that is compatible with the casing availble casing. [8]

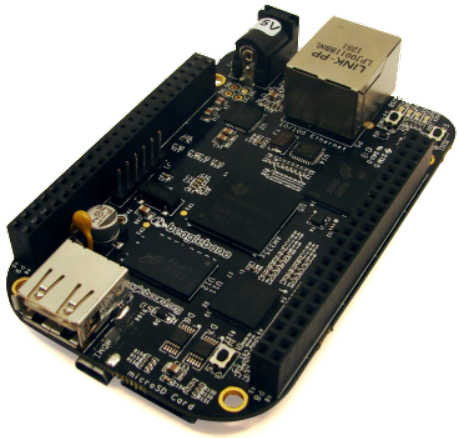


Figure 3.4: The BeagleBone Black [7]

The BeagleBone serves as the main processor for the ROV. It processes all the incoming data from the controller board over the serial connection and also transmits the different control commands sent over ethernet from the base station. The BeagleBone runs a custom debian image, which has been tweaked by the OpenROV team, in order to be optimized for the platform. The cockpit interface used to control the ROV runs directly on the BeagleBone on a Node.js server, which features video streamed from a USB connected webcam on the BeagleBone itself. The processing power of the BeagleBone is sufficient for the ROV project. If more processor power is needed it can be replaced with a similar device that runs a linux destribution [9]. Below is a list of hardware specifications for the BeagleBone Black.

- Processor: AM335x 1GHz ARM® Cortex-A8
- 512MB DDR3 RAM
- 4GB 8-bit eMMC on-board flash storage
- 2x 46 pin headers

- Ethernet

3.1.3 Tether and Communication

Figure 3.5: The tether used for the OpenROV [7]

The OpenROV 2.8 includes a 100m long 2x24 AWG twisted wire. A disadvantage with this particular tether is that it is negatively buoyant meaning it will sink towards the bottom and add act as an additional drag on the backside of the ROV. Due to the sinking effect there is an increased chance that the tether gets caught on an object at the bottom. The OpenROV is specified by the manufactures to be operational up to a depth of 100m. The tethered connection using the Tendra Homeplug, that uses Ethernet as the communication protocol, functions up to 300m. Adding a longer tether than the included one creates a risk of loosing transmission speed and possibily also connection issues. By only using the 100m tether also prevents the ROV from going deeper than specified maximum depth. [10]

The Tendra Homeplugs are powerline Ethernet adapters which have been modified in order to allow communication between the OpenROV topside interface board and the base-station. It uses the IEEE 802.3 and 802.3/u protocols which are the standard protocols for physical layer and data link layer Ethernet. The Tendra Homeplug can stream data at 200Mbps which makes it suitable for video streaming and transmission of other large data quantities. Other possible tether connections could be using an Ethernet cable (cat5e or cat6) compared to the two-wire tether using the modified Homeplugs. A downside of using an Ethernet cable is the additional added weight on the tether since it has 4 twister pairs of wire. The advantage is a speed increased up to 1000Mbps functional at 100m. A third alternative is using fibre optics but it is fragile and expensive. Ultimately it would provide a 2.5Gbps communication though in most applications overkill in terms of speed [11] [12].

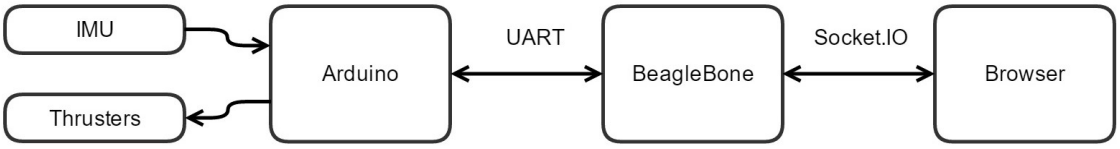


Figure 3.6: OpenROV 2.8 communication diagram [13]

As mentioned previously, the platform uses multiple communication protocols between the different hardware. At the lowest level the sensors and motors communicate with the IO pins on the Arduino where the thrusters are controlled by sending commands to the Electronic Speed Control (ESC)s and the IMU data is retrieved over I2C. The Arduino communicates with the BeagleBone using Universal Asynchronous Receiver/Transmitter (UART) which is serial communication. The main loop written on the Ardunio serves as a command manager where it then sends and receive commands accordingly. The BeagleBone runs a Node.js server that parses the information provided from the Arduino over serial and publishes it over the tethered connection using Socket.IO. The browser on the topside computer then listens to the Socket.IO over the Ethernet connection and renders a webpage Graphical User Interface (GUI) using JavaScript. The cockpit then listen to key presses from the topside computer and functions as a two way connection which sends the desired command back over Ethernet that will then get transmitted to the Arduino.

3.1.4 Thrusters

(a) The set of motors [7] **(b)** The set of propellers [7]

Figure 3.7: Motors and propellers

Figure 3.8: Electronic speed control Afro 12-amp [7]

The motors currently in use are three DST-700 brushless DC motors. The main advantage of these motors is a really low cost at only 40$ for a set of 3. The downside is the short life span and the maintenance required due to the cheap construction of the motors. If they are not cleaned and kept lubricated with silicone lube they will start seizing up and create unwanted additional friction. The DST-700 needs modifications in order be used in salt water environments since the salt water ruins the coating on the coils inside the motor. Other alternative to these motors could be the BlueRobotics T100 thrusters that cost 119$.

3.1.5 IMU/Compass/Depth Module

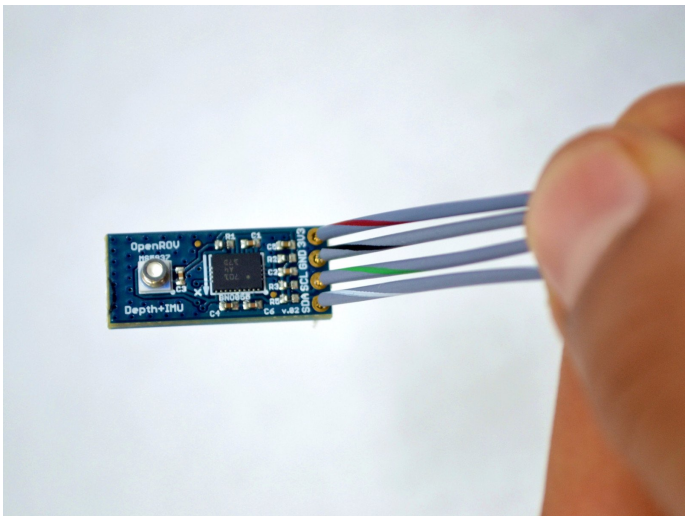


Figure 3.9: IMU/Compass/Depth Module [7]

The OpenROV has an IMU/Compass/Depth module that can be bought as an accessory. The IMU, the BNO055 [14] can measure inertial forces acting on the movement of the body, it also has a a gyroscope that can track the rotational movements. The compass module is utilized to determine the orientation (heading) with reference from the magnetic north. The IMU combines these sensors with on-board sensor fusion, in the datasheet there is limited information regarding the actual performance of the outputs utilizing the sensor fusion. Attached to the board is also a pressure sensor, the MS5837 [15], that can be used to measure depth.

3.1.6 Assembly

The assembly of the ROV was done using their online documentation which features detailed guides on how to acrylic cement the different pieces and also where to apply the epoxy. One of the main issues that was encountered during the build process was finding a suitable acrylic cement since the ones listed on in the OpenROV assembly guide were unavailable in the European Union (EU). This was due to the chemicals composition of the acrylics cements and were therefore banned due to health reasons. A suitable replacement was hobby glue from Revell, see figure 3.10, which is used to assemble acrylic parts on hobby models.

Figure 3.10: Acrylic hobby glue

Due to the choice of acrylic cement the liquid glue does not properly evaporate when attempting to combine the acrylic pieces. This chosen acrylic cement also does not produce a strong bond, at least not compared to the recommend glue that was unavailable. The end-caps for the battery tube are significantly more fragile compared to the rest of the ROV since they are handled whenever batteries need changing and are made our of multiple thin pieces of acrylic. This means they can be easily be fractured and a great care is needed to watch out for them not braking.

3.2 Logic Analysis

When implementing the control system the plan is to do so directly on the BeagleBone, and therefore the Node.js server would be disabled. Since disabling the Node.js server means disabling the cockpit, it was then necessary to analyse the syntax of the commands transmitted between the BeagleBone and the Arduino over serial. Through the documentation and the open source code it was specified that it uses the UART pins and communicates at a baud rate of 115200. Using this information it was possible to attach a logic analyser in order to retrieve the commands by pressing the different keys in the cockpit before disabling it. The logic analyser hardware available was the LHT00SU1 Oscilloscope and Logic Analyzer used for the data acquisition and the Usbee suite for the analysis. A picture of the LHT00SU1 can be seen in figure 3.11.

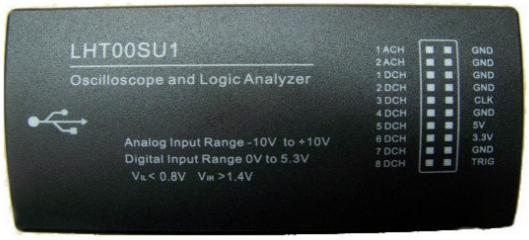


Figure 3.11: LHT00SU1 Oscilloscope and Logic Analyzer [16]

Some sample data received by the logic analyser can be seen in figure 3.12, which in this case indicates a positive thrust in the surge direction at 25% thrust. The keys in the cockpit were systematically pressed to send all the available commands to the Arduino. The hexadecimal value in the front of the command is the Cyclic Redundancy Check (CRC) value and is used for ensuring that the received command was correct which was implemented by default to ensure valid commands were received/transmitted.

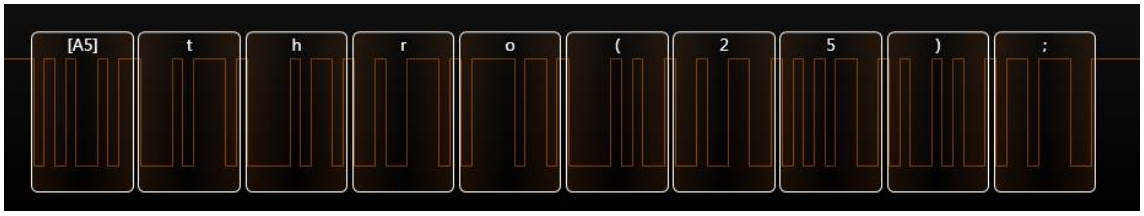


Figure 3.12: Example of a throttle command

Table 3.1 shows a list of the available commands that can be sent to the ROV from the BeagleBone directly over serial. The CRC values has been removed and the input values for the commands substituted with x or y. Where the x value indicates a range from -100 to 100, and represents percentage thrust in either direction. To stop the thrusters the value 0 is sent to the active thruster. The y value is for the

lighting and lasers, the main lights can be adjusted in brightness from 0 being off to 255 being full brightness. The laser are either on or off, indicated by either 0 or 255. The two last commands are for calibration of the IMU and are used for zeroing of the tilted angles or depth sensor.

Action	Command
Surge throttle	throt(x);
Yaw throttle	yaw(x);
Heave throttle	lift(x);
Lights	ligt(y);
Laser	claser(y);
Zero IMU	zeroimu();
Zero depth	dzer();

Table 3.1: Table of commands for the ROV(x = -100 to 100, y = 0 to 255)

The main motivation for doing the logic analysis was to save time since the controller implementation on the BeagleBone can be created to be compatible with the available set of commands on the Arduino. Also, the OpenROV project has a large code base and due to their coding structure, style and lack of documentation, time and effort was saved from not having to search through the entire code, in order to determine the communication between the GUI and the Arduino. When the serial communication had been found a command could be sent through the linux terminal in the BeagleBone using the echo command like seen in 3.1

```
1  echo "thro(25);" -n > /dev/ttyO1
```

Since the cockpit GUI will be disabled, the ROV will be controlled through Secure Shell (SSH). In order the send the commands from table 3.1, it is possible do to so directly through the terminal once the BeagleBone has been accessed via. SSH.

```
1  sprintf(cmd, "echo \"thro(%d);\" -n /dev/ttyO1", (int)u);
   system(cmd);
```

The implementation will take place using C, where the desired input value is added to a string and then passed to the system function, which then performs a system call.

Chapter 4

Mathematical Modelling

4.1 ROV Dynamics

Name	Description	Unit
F_d	Drag force	N
F_{dr}	Rotational drag force	N
F_b	Buoyancy force	N
F_g	Gravitational force	N
F_t	Thrust force from motor	N
F_{tr}	Thrust force from right motor	N
F_{tl}	Thrust force from left motor	N
F_{sum}	Summation of forces	N
m_{rov}	Mass of the ROV	kg
I_{rov}	Moment of inertia	$kg \cdot m^2$
vol_{rov}	Volume of the ROV	m^3
C_d	Drag constant	–
ρ_{water}	Water density	kg/m^3
V	Velocity	m/s
A	Cross sectional area	m^2
b_d	Linear drag coefficient	–
a_d	Quadratic drag coefficient	–
ϕ	Roll position	rad
θ	Pitch position	rad
ψ	Yaw position	rad
x	Displacement along the x-axis	m
z	Displacement along the z-axis	m

Table 4.1: Table of nomenclature for the ROV dynamics

Dot notation will be used in order to describe the first and second order derivatives in the following subsections on mathematical modelling.

4.1.1 Up and Downward Motion

When dealing with the upwards and downwards motion of the ROV it is important to take the buoyancy into consideration. There are three possible buoyancy cases: Positive, negative and neutral. The ROV will be as close to neutrally buoyant as possible, in order to help with positioning it.

$$\rho_{water} vol_{rov} g = m_{rov} g \tag{4.1a}$$

If the ROV is at rest or moving at a constant velocity equation 4.1a holds, this means that the ROV is neutrally buoyant.

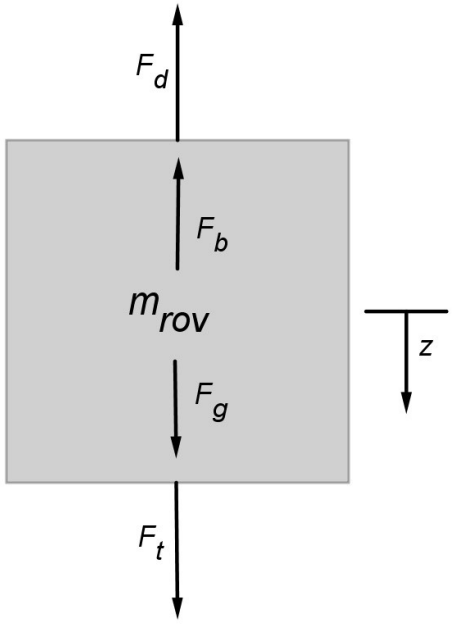


Figure 4.1: Free-body diagram for up and down motion

In order to determine the forces influencing the system a free body diagram was created, as shown in figure 4.1. By summing all of the forces and setting it equal to newtons second law the following relationship is obtained for the movement along the z-axis.

$$m_{rov}\ddot{z} = F_t + F_g - F_d - F_b \tag{4.2}$$

A known formula for the drag force is as following, where C_d is the drag coefficient depending on the ROVs characteristics.

$$Drag = 0.5\rho_{water} C_d \dot{z}^2 A \tag{4.3}$$

Equation 4.3 is a formula for drag, but due to its limitations when linearised around an operating point of 0, a different approach is taken, where it is chosen to model the drag as an equation consisting both the quadratic and linear drag. [17]

$$F_d = a_d \dot{z}^2 + b_d \dot{z} \tag{4.4}$$

The coefficients a_d and b_d in equation 4.4 are compact of expressions containing all of the similar characteristics for the ROV as in equation 4.3. In order to use equation 4.4 for both directions of movement it is necessary to modifying the equation in order to maintain the proper sign.

$$F_d = a_d \dot{z}|\dot{z}| + b_d \dot{z} \tag{4.5}$$

By combining the above mentioned equations for the z-directional movement, with downwards motion being the positive direction of movement, the expression is as follows.

$$m_{rov}\ddot{z} = F_t + m_{rov}g - a_d \dot{z}|\dot{z}| - b_d \dot{z} - \rho_{water} vol_{rov} g \tag{4.6}$$

Equation 4.6 can be simplified by assuming that $m_{rov}g - \rho_{water} vol_{rov} g = 0$, as mentioned previously due to it being neutrally buoyant.

$$\ddot{z} = \frac{F_t}{m_{rov}} - \frac{(a_d \dot{z}|\dot{z}| + b_d \dot{z})}{m_{rov}} \tag{4.7}$$

4.1.2 Forward and Backwards Motion

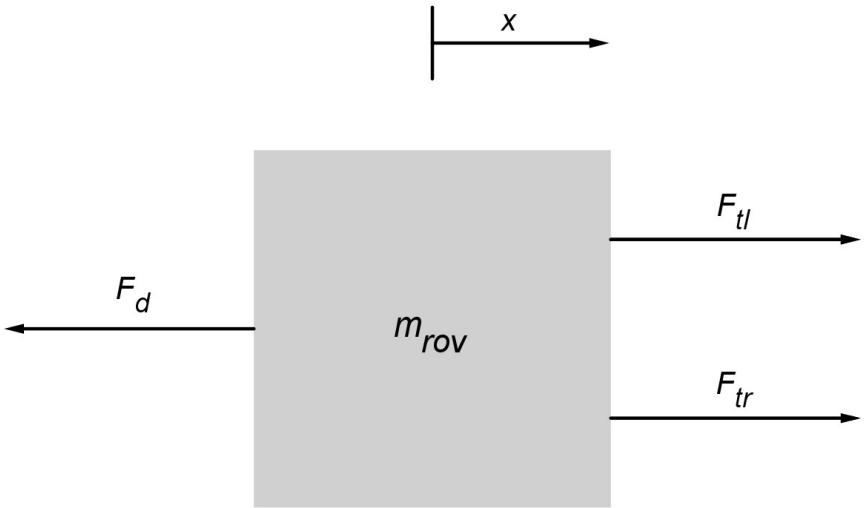


Figure 4.2: Free-body Diagram for the forwards and backwards motion

Using the free-body diagram in figure 4.2, it is possible to determine the forces acting on the ROV while it is moving along the x-axis.

$$m_{rov}\ddot{x} = F_{tr} + F_{tl} - F_d \tag{4.8}$$

The equation for drag is the same as for the z-directional movement, with a different set of values for the constants. By inserting the know expression for drag, the movement is determined by the following equation.

$$\ddot{x} = \frac{F_{tr}}{m_{rov}} + \frac{F_{tl}}{m_{rov}} - \frac{(a_d \dot{x}|\dot{x}| + b_d \dot{x})}{m_{rov}} \tag{4.9}$$

4.1.3 Rotational Movement

Yaw

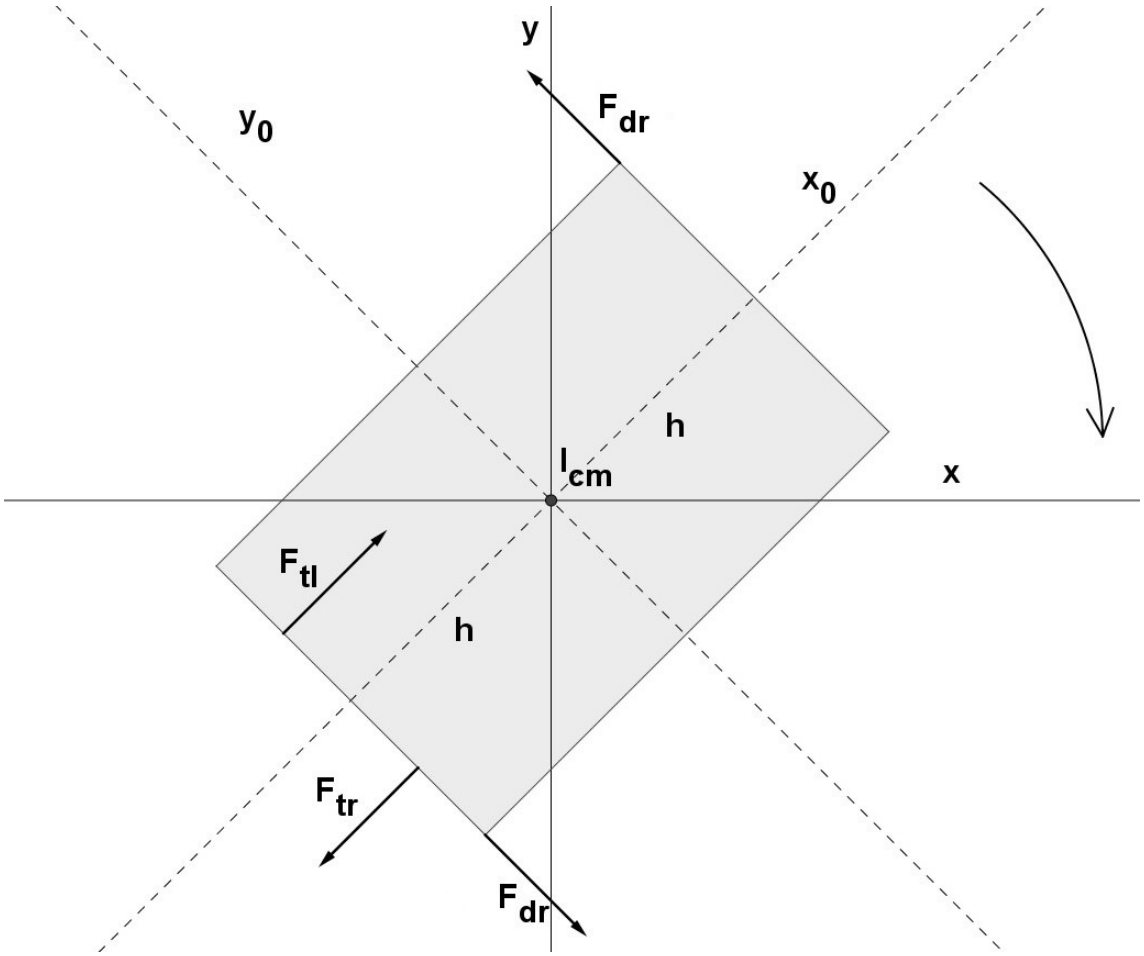


Figure 4.3: Free-body diagram for yaw

Based on the free-body diagram two forces from the thrusters act upon the ROV during yaw movement, creating a rotational torque about the z-axis, with an opposing torque in the form of drag.

$$I_{rov}\ddot{\psi} = \tau_{thrusters} - \tau_{drag} \tag{4.10}$$

Rotation is created around the z-axis by having the left and right thrusters (F_{tl}, F_{tr}) powered on in opposite directions. The drag torque is caused from the rotation is calculated using the equation previously mentioned in equation 4.3, where the linear velocity is substituted by angular velocity. The drag force influences both sides of the ROV as it rotates, this being the upper right-hand side and lower left-hand side when split down the middle. As noted on figure 4.3 the drag force F_{dr} is divided into two equally sized forces.

$$2F_{dr} = 2(a_d\dot{\psi}|\dot{\psi}| + b_d\dot{\psi}) \tag{4.11}$$

In order to convert the two drag forces into torque the force F_d is multiplied by the distance h and then by two. This is equivalent to multiplying the drag force by the length of the ROV. This results in the counter-torque or rotational drag force.

$$\tau_{dr} = 2hF_d = LF_d = L(a_d\dot{\psi}|\dot{\psi}| + b_d\dot{\psi}) \tag{4.12}$$

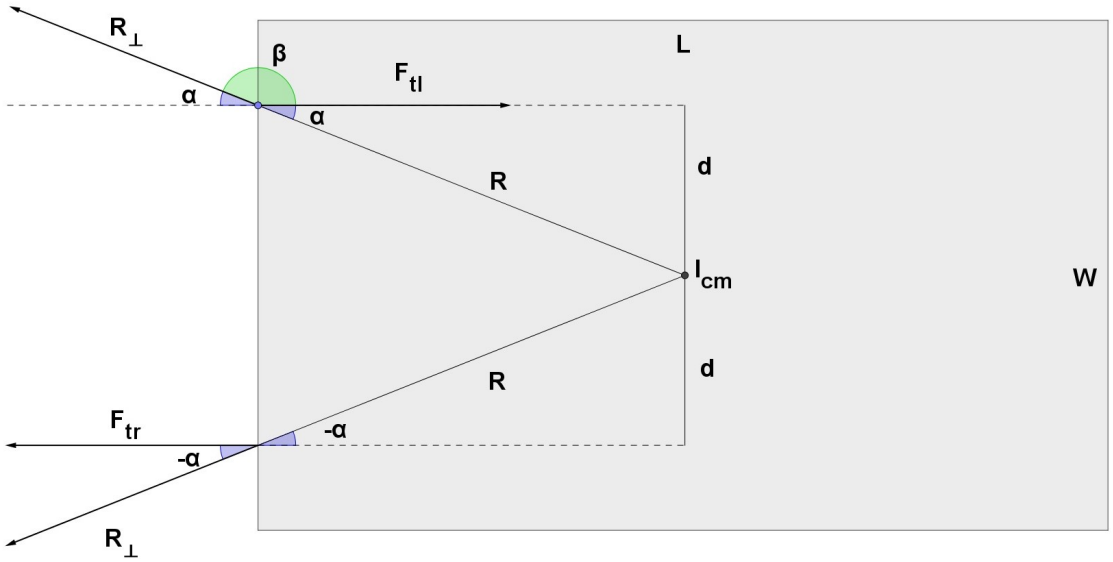


Figure 4.4: Force projection F_{tr} and F_{tl}

The force projections for the thrusters can be calculated from the above image, where α is calculated based on the centre of mass and the distances R and d.

$$\tau_{tl} = F_{tl}R\sin(\beta) = F_{tl}R\sin(\alpha) \tag{4.13a}$$
$$\tau_{tr} = F_{tr}R\sin(-\alpha) \tag{4.13b}$$

The ROV rotates around its centre of mass, therefore it is possible to use a known formula that describes the moment of inertia, for the centre of a rectangle since the centre of mass is at the centre of the ROV. Equation 4.14 shows the before mentioned expression.

$$I_{rect} = m\frac{1}{12}(l^2 + w^2) \tag{4.14}$$

A differential equation describing the yaw-movement based on figure 4.3 and 4.4 is then formed and is found in equation 4.15.

$$I_{rov}\ddot{\psi} = F_{tl}R\sin(\alpha) + F_{tr}R\sin(-\alpha) - L(a_d\dot{\psi}|\dot{\psi}| + b_d\dot{\psi}) \tag{4.15}$$

By inserting the expression for the moment of inertia and isolating the highest order derivative, the differential equation looks as following in equation 4.16.

$$\ddot{\psi} = \frac{R\sin(\alpha)}{m\frac{1}{12}(l^2+w^2)}F_{tl} + \frac{R\sin(-\alpha)}{m\frac{1}{12}(l^2+w^2)}F_{tr} - \frac{La_d}{m\frac{1}{12}(l^2+w^2)}\dot{\psi}|\dot{\psi}| - \frac{Lb_d}{m\frac{1}{12}(l^2+w^2)}\dot{\psi} \tag{4.16}$$

Pitch and Roll

The dynamics for the pitch and roll are similar to the yaw, since there is some torque generated from some input thrust, where there also is an associated counter-torque in the form of rotational drag.

$$\ddot{\theta} = \frac{1}{I_{pitch}}\tau_p - \frac{1}{I_{pitch}}\tau_d \tag{4.17}$$

Where τ_p is so some torque used to create the pitching motion and τ_d being the drag, in terms of a counter torque.

$$\ddot{\phi} = \frac{1}{I_{roll}}\tau_r - \frac{1}{I_{roll}}\tau_d \tag{4.18}$$

Similar as with the pitch dynamics, the roll is described using a torque for the roll motion and counter torque for the drag. Each motion in the yaw-pitch-roll direction have their own inertial calculations.

4.2 Thruster Modelling

Name	Description	Unit
L_a	Armature inductance	H
R_a	Armature resistance	Ω
V_i	Motor voltage	V
k_e	Motor constant	-
k_t	Motor constant	-
$i(t)$	Motor current	A
$\dot{\theta}$	Angular velocity	rad/s
$\ddot{\theta}$	Angular acceleration	rad/s^2
J	Motor inertia	$kg \cdot m^2$
B	Viscous friction	Nms

Table 4.2: Table of nomenclature for the ROV

The thrusters for the ROV consists of three Brushless Direct Current (BLDC) motors, where the single thruster for heave is equipped with a 2-bladed propeller and the two thrusters for surge and yaw movement with 3-bladed propellers. They are controlled using three input wires and by activating two phases at once, using the third to sense its position. These motors are sensorless and use three Afro 12AMP ESCs to control the speed and direction. These ESCs are equipped with small microcontrollers that determines the sequence of which the input leads for the BLDC are powered and their phase. The strategy is to first create a simplified model for a regular DC motor, then modify this model to in-cooperate the BLDC characteristics, then finally the propeller dynamics are added as a load torque.

4.2.1 Brushed DC Motor Model

The following subsection focuses on a brushed DC motor, in order to derive an expression for the electrical and mechanical part individually. Figure 4.5 shows the electrical model and the mechanical free body diagram of a regular DC motor.

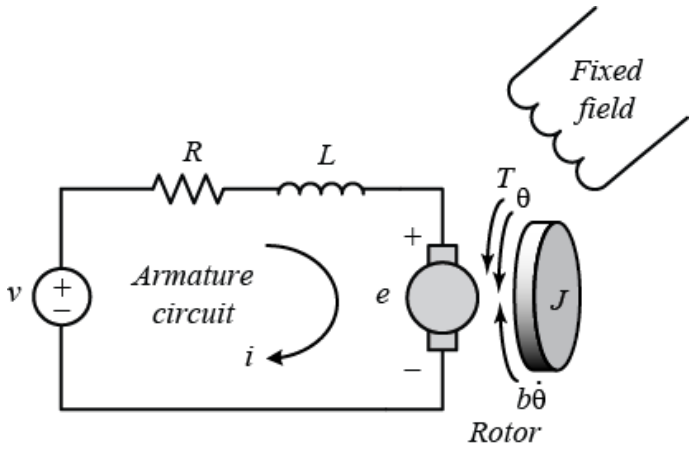


Figure 4.5: Schematic for a DC motor containing both the electrical and mechanical part [18]

Through the use of Kirchhoff Voltage Law (KVL), it is possible to determine the differential equation for the electrical part of the DC motor, this contains the armature resistance, inductance and backEMF.

$$L_a\dot{i}(t) + R_a i(t) = V(t) - k_e\dot{\theta}(t) \tag{4.19a}$$

$$L_a\dot{i}(t) = V(t) - k_e\dot{\theta}(t) - R_a i(t) \tag{4.19b}$$

$$\dot{i}(t) = \frac{V(t) - k_e\dot{\theta}(t) - R_a i(t)}{L_a} \tag{4.19c}$$

By isolating in terms of the highest derivative, the electrical model looks as shown in equation 4.19c. The equations concerning torque generated from the electrical force and also the mechanical shaft is shown in equations 4.20a and 4.20b.

$$T_m = k_t i_a(t) \tag{4.20a}$$

$$T_m = J\ddot{\theta}(t) + B\dot{\theta}(t) + T_l \tag{4.20b}$$

Equation 4.20b is derived from the free body diagram shown in figure 4.5, where T_l is the torque load from the propeller. By rearranging the equation it is possible to isolate the torque load as seen in 4.21.

$$k_t i_a(t) = J\ddot{\theta}(t) + B\dot{\theta}(t) + T_l \tag{4.21a}$$

$$T_l = k_t i_a(t) - B\dot{\theta}(t) - J\ddot{\theta}(t) \tag{4.21b}$$

Where the integrated current value from 4.19c is used as input for 4.21b.

4.2.2 Brushless DC Motor Model

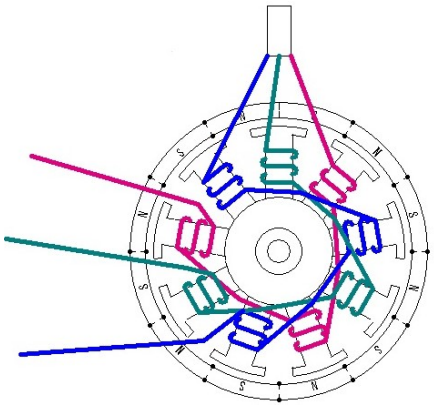


Figure 4.6: Magnet typical magnet layout for star winded BLDC [19]

As mentioned earlier, the ROV is equipped with three 3-phase BLDC motors. These motors are known as so-called outrunner BLDCs, which means that the permanent magnets are placed in the outer ring with the electromagnets placed on the inner (See figure 4.6). Compared to a brushed DC motor the commutation is achieved electronically, instead of using brushes, which is one of the typical downsides of using a brushed DC motor.

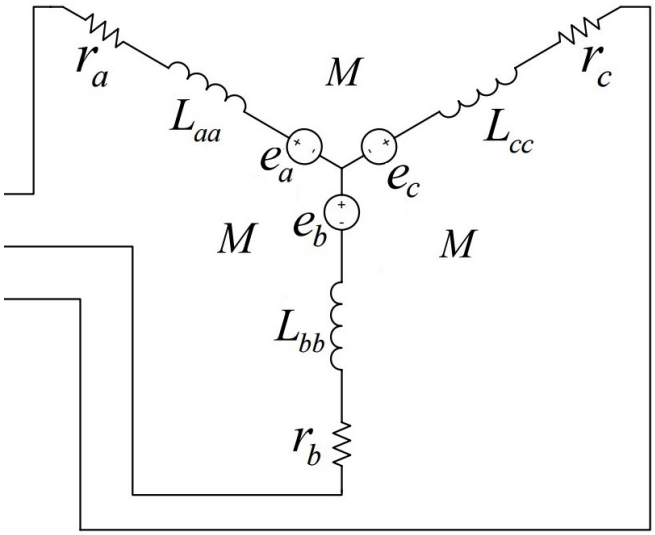


Figure 4.7: Electrical schematic for the BLDC star windings [20]

Typically higher-end BLDCs are equipped with hall-effect sensors in order to determine the current motor position in order to know when to power the next set of phases. But in this case the BLDCs are sensorless and relies on the backEMF measured from the non-active input lead [21].

By comparing figure 4.6 and 4.7 the three leads that are connected on figure 4.6 are equivalent to the center of the diagram in the star formation, shown in figure 4.7.

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} + \begin{bmatrix} r_a & 0 & 0 \\ 0 & r_b & 0 \\ 0 & 0 & r_c \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} L_{aa} & L_{ba} & L_{ca} \\ L_{ab} & L_{bb} & L_{cb} \\ L_{ac} & L_{bc} & L_{cc} \end{bmatrix} \begin{bmatrix} \dot{i}_a \\ \dot{i}_b \\ \dot{i}_c \end{bmatrix} + \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix} \tag{4.22}$$

Through analysis of the electrical schematic, it is possible to obtain three equations as shown in equation 4.22. Where there is one equation for each input lead, where the values v_s respectively are the differently phased input voltages [20]. In order to simplify these equations, the inductances and mutual inductance are determined to be equal to each other, and similarly with the resistance.

$$r_a = r_b = r_c = r \tag{4.23a}$$

$$L_{aa} = L_{bb} = L_{cc} = L \tag{4.23b}$$

$$L_{ab} = L_{ba} = L_{bc} = L_{cb} = L_{ca} = L_{ac} = M \tag{4.23c}$$

By inserting the simplified expression from equations 4.23a,4.23b and 4.23c into equation 4.22, the following simplified expression is obtained.

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} + \begin{bmatrix} r & 0 & 0 \\ 0 & r & 0 \\ 0 & 0 & r \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} L-M & 0 & 0 \\ 0 & L-M & 0 \\ 0 & 0 & L-M \end{bmatrix} \begin{bmatrix} \dot{i}_a \\ \dot{i}_b \\ \dot{i}_c \end{bmatrix} + \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix} \tag{4.24}$$

The equations shown in 4.24 are similar to the voltage equation found in the brushed DC motor case, where the backEMF in equation 4.19c is denoted as in equation 4.25.

$$e = k_e \dot{\theta} \tag{4.25}$$

As shown in equation 4.20a (where $k_t = k_e$ in SI units), the electrically generated torque can be modified to accompany the three backEMF terms compared to the single one, which was the case for the single-phase motor [22].

$$T_{BLDC} = k_e i = \frac{e}{\dot{\theta}} i = \frac{1}{\dot{\theta}} (e_a i_a + e_b i_b + e_c i_c) \tag{4.26}$$

Where the expression from equation 4.26 can be inserted into the to equation describing the mechanical rotation of the rotor, as shown in equation 4.27.

$$T_{BLDC} = J\ddot{\theta}(t) + B\dot{\theta}(t) + T_l \tag{4.27}$$

4.2.3 Propeller Thrust and Torque

Name	Description
D	Propeller diameter
K_Q	Propeller torque constant
K_T	Propeller thrust constant
u_a	Ambient water speed
J_0	Advance Ratio
Z	Number of propeller blades
$\frac{P}{D}$	Propeller pitch ratio
$\frac{A_E}{A_O}$	Propeller expanded-area ratio
R_n	Reynolds Number
t_p	Maximum blade thickness
c_p	Chord length of blade section
n	Revolutions per second

Table 4.3: Table of nomenclature for the ROV

The ROV uses three thrusters in order to create movement as mentioned previously in the report. These being two rear thrusters for surge, sway and yaw movement and one vertical thruster for heave motion. As possible equation for the amount of thrust generated by the propeller can be seen in equation 4.28.

$$F_t = \rho_{water} D^4 K_T n|n| \tag{4.28a}$$

This non-linear equation depends on the water density, propeller diameter, propeller rate and the thrust coefficient. The thrust coefficient K_T can be defined as function dependant on several values that are specific to the propeller characteristics.

$$K_T = f_1\left(J_0, \frac{P}{D}, \frac{A_E}{A_O}, Z\right) \tag{4.29}$$

Where the advance ratio, J_0, is defined as following in equation 4.30.

$$J_0 = \frac{u_a}{nD} \tag{4.30}$$

Similar to the thrust equation, an equation for the propeller torque depends on the water density, propeller diameter, propeller rate and a torque coefficient, where the torque coefficient is a function also dependant on values specific to the given propeller characteristics.

$$T_l = \rho_{water} D^5 K_Q |n| n \tag{4.31}$$

$$K_Q = f_2\left(J_0, \frac{P}{D}, \frac{A_E}{A_O}, Z, R_n, \frac{t}{c}\right) \tag{4.32}$$

Equation 4.31 shows the equation for the propeller torque and equation 4.32 the function for the torque constant [23] [24].

4.3 State-Space Representation

This section combines the equations specified during the mathematical modelling and puts them into the state-space representation. As seen below in equation 4.33, the equations for the translational motion (Equations 4.7, 4.9) in the xyz-directions are put into matrix form. The sway movement is not controllable directly by a thruster and therefore has no thrusters associated with it. The following representation is a placeholder for the linearised equations of the system dynamics, since state-space requires linear differential equations.

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \frac{-F_{d_x}}{m_{rov}} & 0 & 0 \\ 0 & \frac{-F_{d_y}}{m_{rov}} & 0 \\ 0 & 0 & \frac{-F_{d_z}}{m_{rov}} \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} + \begin{bmatrix} \frac{1}{m_{rov}} & \frac{1}{m_{rov}} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m_{rov}} \end{bmatrix} \begin{bmatrix} F_{tr} \\ F_{tl} \\ F_t \end{bmatrix} \tag{4.33}$$

Similar to the translational movement, the equations for the rotational motion is placed into matrix form and is shown in equation 4.34. Where the positive and negative gain is calculated based on the force decomposition of the thrusters.

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{-F_{dr_\phi}}{I_{rov}} & 0 & 0 \\ 0 & \frac{-F_{dr_\theta}}{I_{rov}} & 0 \\ 0 & 0 & \frac{-F_{dr_\psi}}{I_{rov}} \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{-gain}{I_{rov}} & \frac{gain}{I_{rov}} & 0 \end{bmatrix} \begin{bmatrix} F_{tr} \\ F_{tl} \\ F_t \end{bmatrix} \tag{4.34}$$

Based on the problem delimitation in section 2.3, the goal is create depth control. Therefore the matrices for the translational motion must be augmented in order to include states for the respective positions. Equation 4.35 then displays the augmentation of the translation motion matrices, but also the combination with the rotational motion.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{-F_{d_x}}{m_{rov}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{-F_{d_y}}{m_{rov}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{-F_{d_z}}{m_{rov}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{-F_{dr_\phi}}{I_{rov}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-F_{dr_\theta}}{I_{rov}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-F_{dr_\psi}}{I_{rov}} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m_{rov}} & \frac{1}{m_{rov}} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m_{rov}} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{-gain}{I_{rov}} & \frac{gain}{I_{rov}} & 0 \end{bmatrix} \begin{bmatrix} F_{tr} \\ F_{tl} \\ F_t \end{bmatrix} \tag{4.35}$$

The coefficients for the drag forces in the 6 directions of movement are going to be determined in the following chapter and therefore currently substituted with place holder names, as seen in equation 4.35. This representation expects the input thrust to be in newtons.

Chapter 5

System Identification

5.1 Accelerometer Data

In order to find the coefficients defined in the mathematical modelling, linear and rotational drag coefficients must be determined. By integrating the acceleration data from the IMU it is possible to retrieve the information needed to calculate the drag at different velocities.

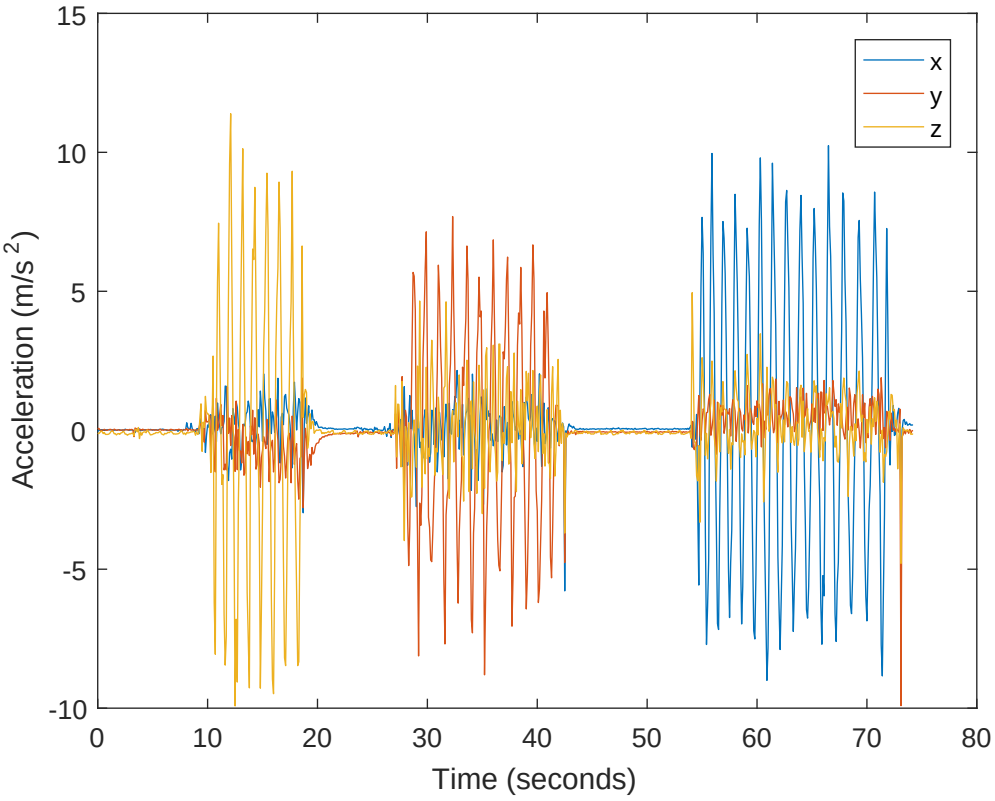


Figure 5.1: Accelerometer data

Some initial experimentation was performed by linearly shaking the ROV up and down in the z-direction, side to side in the y-direction and backwards and forwards in the x-direction. These results can be seen in figure 5.1, where the figure has a clear indication in which direction the ROV is being accelerated.

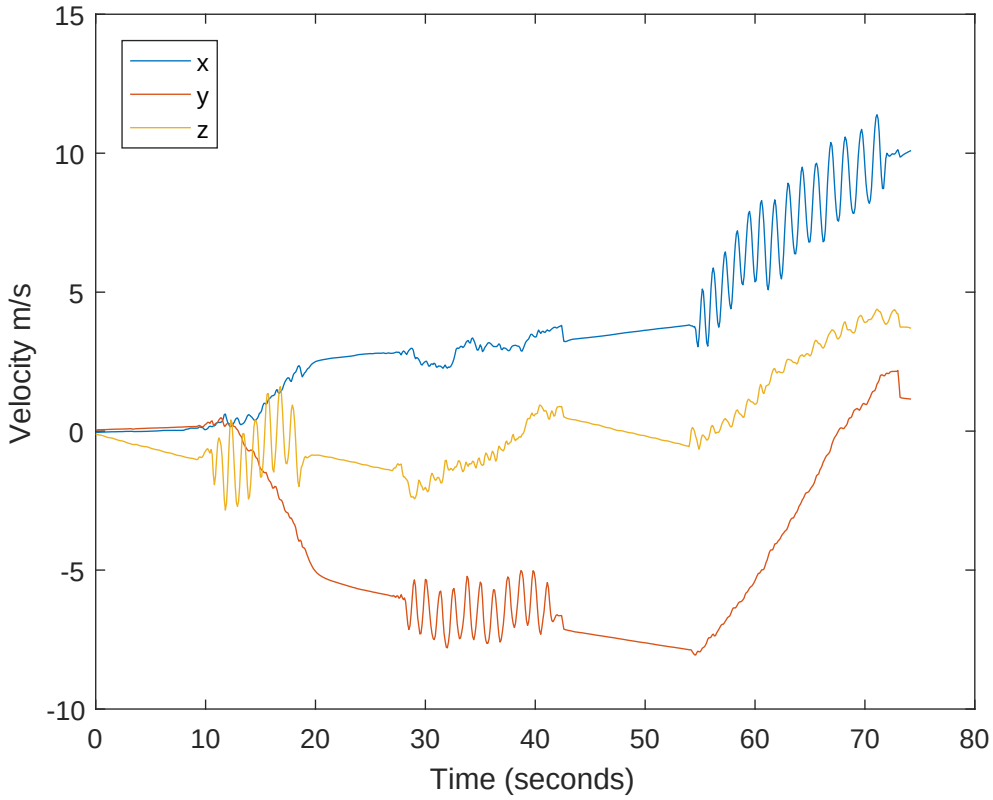


Figure 5.2: Integrated accelerometer data

Figure 5.2 shows the integration of the data from figure 5.1. While the ROV is accelerated in the z-direction there is noise present in the xy-directions, while integrating the data there is a noticeable drift caused from the integration due to the noise. The cause of the noise might be due to the sensitivity settings of the accelerometer or the actual choice of unit itself, since this particular IMU uses sensor fusion in order output linear acceleration and specifies no performance specifications for the linear acceleration.

5.2 Thruster Test

To determine the force generated from the thrusters with the different propellers attached, it was necessary to create a testing rig for the individual thruster cases and measure the force generated. It was decided to use a metal frame with a pendulum attached in the middle. The thrusters were attached to the bottom of the pendulum and then submerged into a bucket of water. A wire was secured to a higher point of the pendulum and the other end attached to a digital newton meter. The set-up can be seen in figure 5.3.

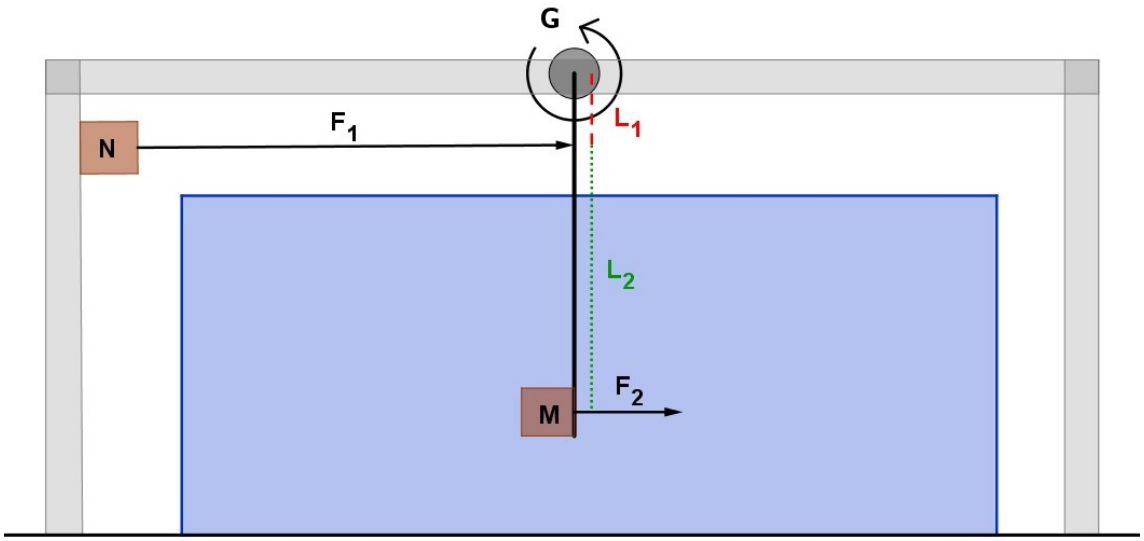


Figure 5.3: Set-up of the motor test

The Newton meter used for the data acquisition is the PASCO PS-3202. It has a measurable force range of ±50N. The newton meter can be seen in figure 5.4. The software used for the data logging was PascoCapstone.

Figure 5.4: PASCO PS-3202 [25]

In figure 5.6 raw data from one of the experiments is plotted, where the data is sampled at 1KHz. In order to control the thruster, different thrust percentages are sent to the control board, where the thrust steps up 10% every 5 seconds until it reaches 100%. The measured data gets more unreliable at higher speeds, that is due to the size of the bucket since the thrusters created a whirlpool effect that can been seen in figure 5.5.

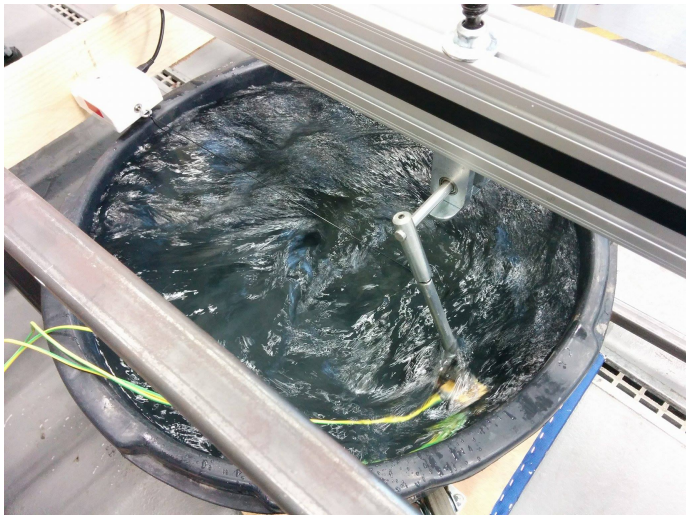


Figure 5.5: Set-up of the motor test

For a more precise measurement at higher speed a bigger testing facility is needed. But since the operation point is around the 0% thrust and mostly will only powered for a short time at the higher values it was decided that the test was successful. Raw data for 2 and 3 propellers both forwards and backwards can be seen in appendix 12.4.

```
  for (i = 10 ; i < 101; i = i + 10){
2   //Creates the string containing the desired thrust value
    sprintf(cmd,"echo \"thro(%d);\" -n > /dev/ttyO1", i);
4   //Send the command
    system(cmd);
6   printf("force perc: %d \n", i);
    sleep(5);
8 }

10   system("echo \"thro(0);\" -n > /dev/ttyO1");
```

Code Listing 5.1: C Implementation for thruster test

Code listing 5.1 shows the C implementation for the experiment, where as described earlier the thrust is increased by 10% every seconds until it reaches 100%.

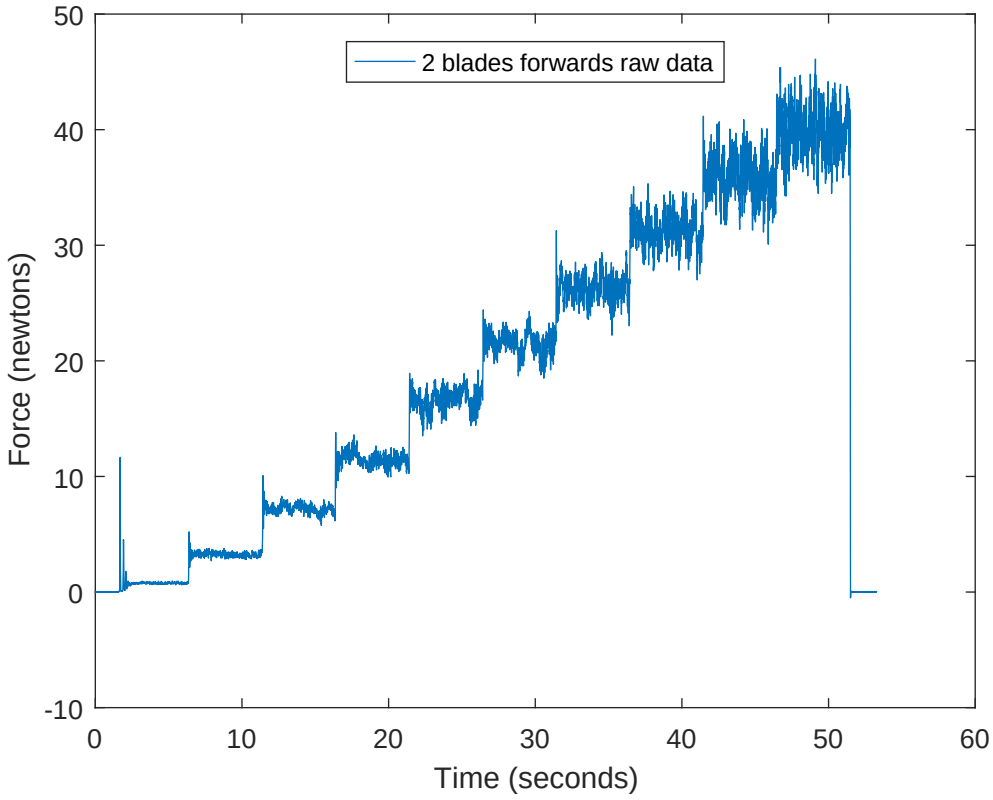


Figure 5.6: Raw data of a 2 propeller forward test

Since the Newton meter is measuring the force above the actual force created by the propeller, it is necessary to calculate the true thrust based on the raw data. The equation for the torque acting on the arm is used to calculate the real force generated by the motor. In figure 5.3 G represent a free turning pivot point.

$$G = L \cdot F \tag{5.1}$$

The lengths of the pendulum were measured $L_1 = 0.06m$ and $L_2 = 0.14m$. The length of the entire arm to the motor is $L_1 + L_2 = 0.20m$. By combining the know information it is then possible to calculate the scaling between the measured force and actual force.

$$F_{motor} = \frac{L_1 \cdot F_{measured}}{L_1 + L_2} = 0.3 \cdot F_{measured} \tag{5.2}$$

Force generated from a motor with a two bladed propeller going backwards and forwards, after the appropriate scaling can be seen in figure 5.7. The thrust plotted based on the measured mean value at the different thrust percentages. Since there is a difference in force generated from forward and backwards movement the mean value is found and shown in yellow.

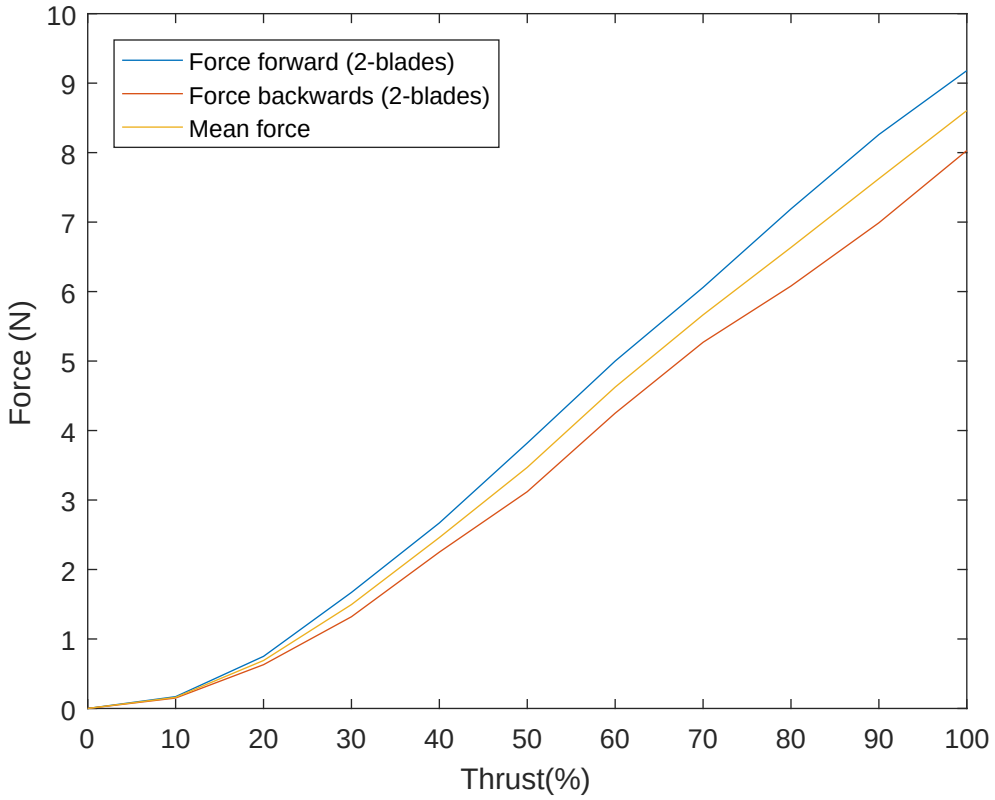


Figure 5.7: Scaled and the mean of a 2 propeller forward test

These experiments and the force calculated will be utilized in order to determine the drag-force acting on the ROV.

5.2.1 Thruster Coefficients

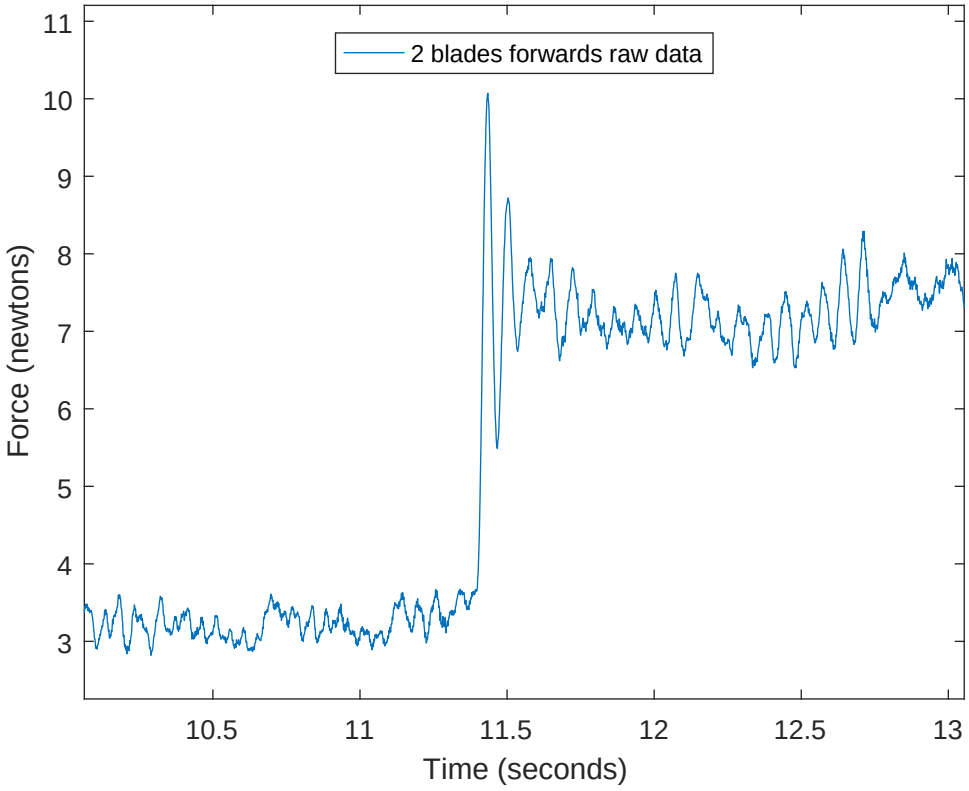


Figure 5.8: Time constant of the motor stepping up 10%

Figure shows the change from 10% to 20% thrust, this change happens within 0.08 seconds and account for the combined dynamics of the propeller, BLDC and ESC. Due to the fast response of the thruster it has been decided to use grey-box modelling since it is possible to obtain the combined performance of the entire thruster system. From this it is concluded that the time constant of the thruster is fast enough to be modelled as a single unit, rather than determining the individual coefficients specified in the mathematical modelling.

Based on the experiments performed with the thrusters, as shown in figure 5.7, it is possible to create a curve that specifies the force produced at the different input trust percentages. This is needed due to the thrust factors specified in the cockpit operating at thrust percentages different than those used while performing the experimentation in section 5.2.

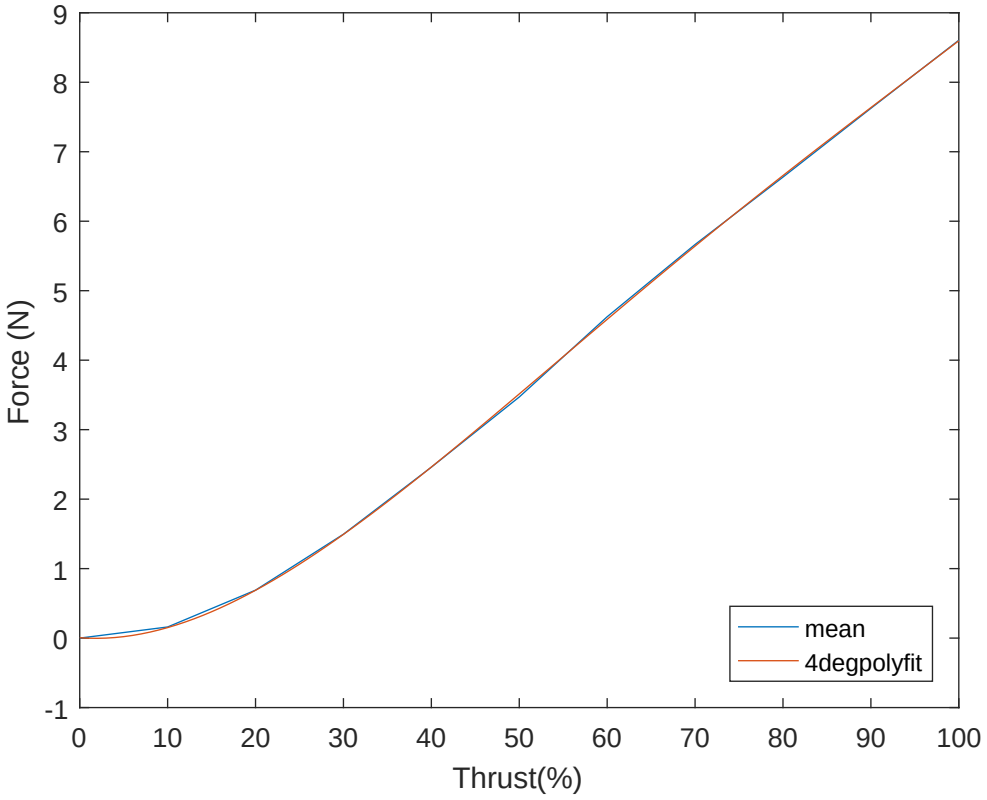


Figure 5.9: Curve fit in order to determine force at certain thrust percentages

The experimental data taken at percentages from 0% to 100% is curve fit in order to determine the force at the four thrust factors used in section 5.3. Based on the data from figure 5.9 it is possible to determine the force values associated with 12%, 25%, 40% and 70% thrust.

$$TF1 = 12\% = 0.2315N \tag{5.3a}$$
$$TF2 = 25\% = 1.0639N \tag{5.3b}$$
$$TF3 = 40\% = 2.4620N \tag{5.3c}$$
$$TF4 = 70\% = 5.6419N \tag{5.3d}$$

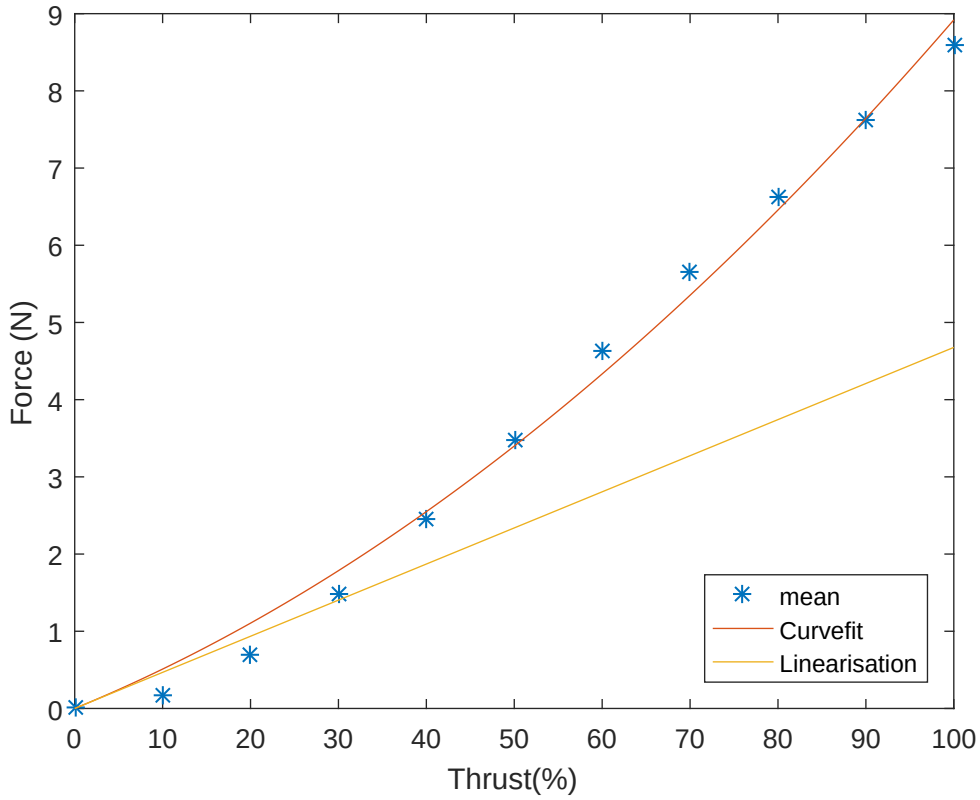


Figure 5.10: Lower degree curve fit and linearisation for the thrust percentage

The curve fit shown in figure 5.9, is not suitable for linearisation around 0%, which is the operating point of the system. This is due to the curvature on the polynomial at lower percentages. Therefore a more suitable curve fit has been created with the intentions of using it for the linearisation of the system. Figure 5.10 shows the new fit with the linearisation added along side it.
This linearisation overcompensates in terms of force at the lower values, but due to deadzone at the lower thrust percentages the overcompensation becomes negligible. The linearisation captures the characteristic of the thruster most accurately between 20% and 30%, which also is going to be the speed range of which the ROV adjusts according to set point changes.

$$y = 0.0004242x^2 + 0.0468x \tag{5.4}$$

Equation 5.4 is the expression for the curve fit shown in figure 5.10.

$$y = 0.0468x \tag{5.5}$$

The linearised expression is shown in equation 5.5.

5.3 Depth Experiments

In order to identify the parameters specified in the mathematical model, z-directional movement experiments were performed in order to retrieve data for the drag coefficients. These depth experiments were performed in an indoor swimming pool with a pool depth of 1.8 meters. As previously mentioned, the ROV is controllable through an interface, which was used to perform these experiments in order to prevent damaging the ROV. The manual operation through the cockpit of the thrusters was done in order to prevent the ROV from crashing into to the bottom of the test pool. Due to the limitations of the ROV cockpit 4 experiments were performed at 4 out of the 5 available thrust factors, the fifth thrust factor operates at 100%, which resulted in the ROV not reaching a constant velocity due to the lack of depth in the pool.

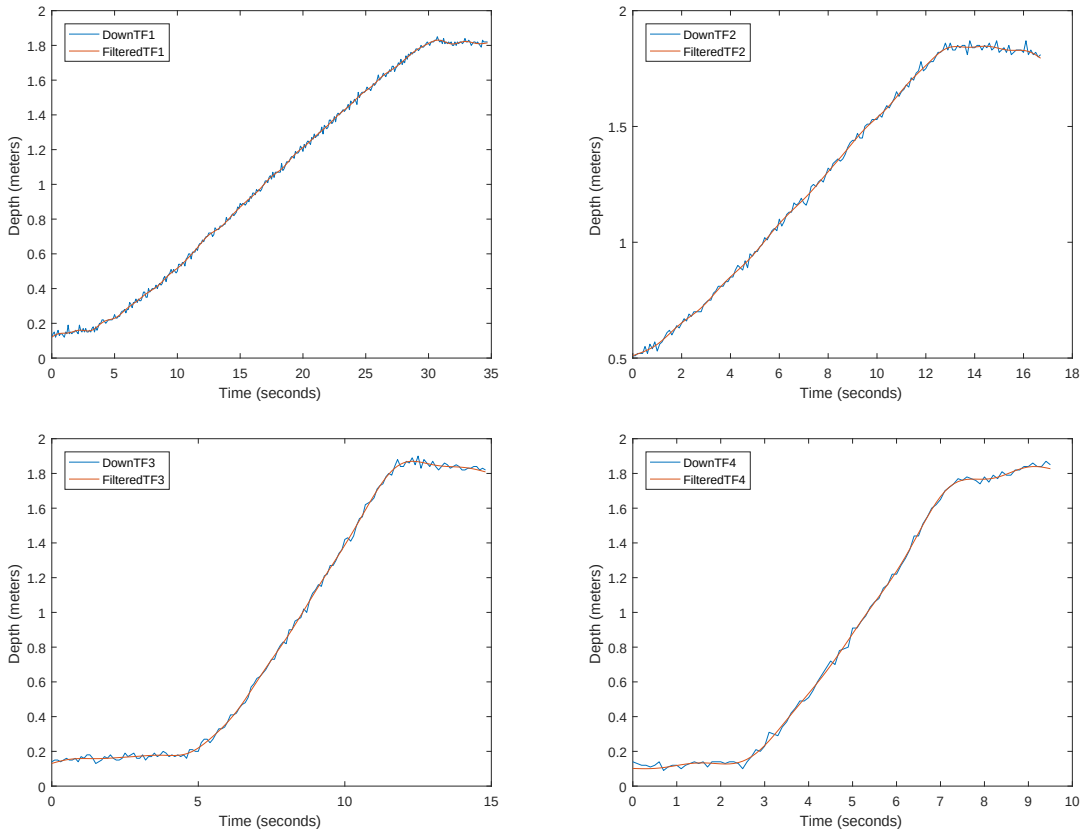


Figure 5.11: Filtered and unfiltered depth data

The depth is calculated using the pressure sensor on the IMU board and is sampled at 10Hz. As shown in figure 5.11 the original measured data has a noticeable amount of noise, therefore some offline filtering was performed and the filtered signal was then overlayed on top of the original. The phase shift created by the filter is irrelevant when working with the data offline, due to the coefficients for the mathematical model being based on recorded values.

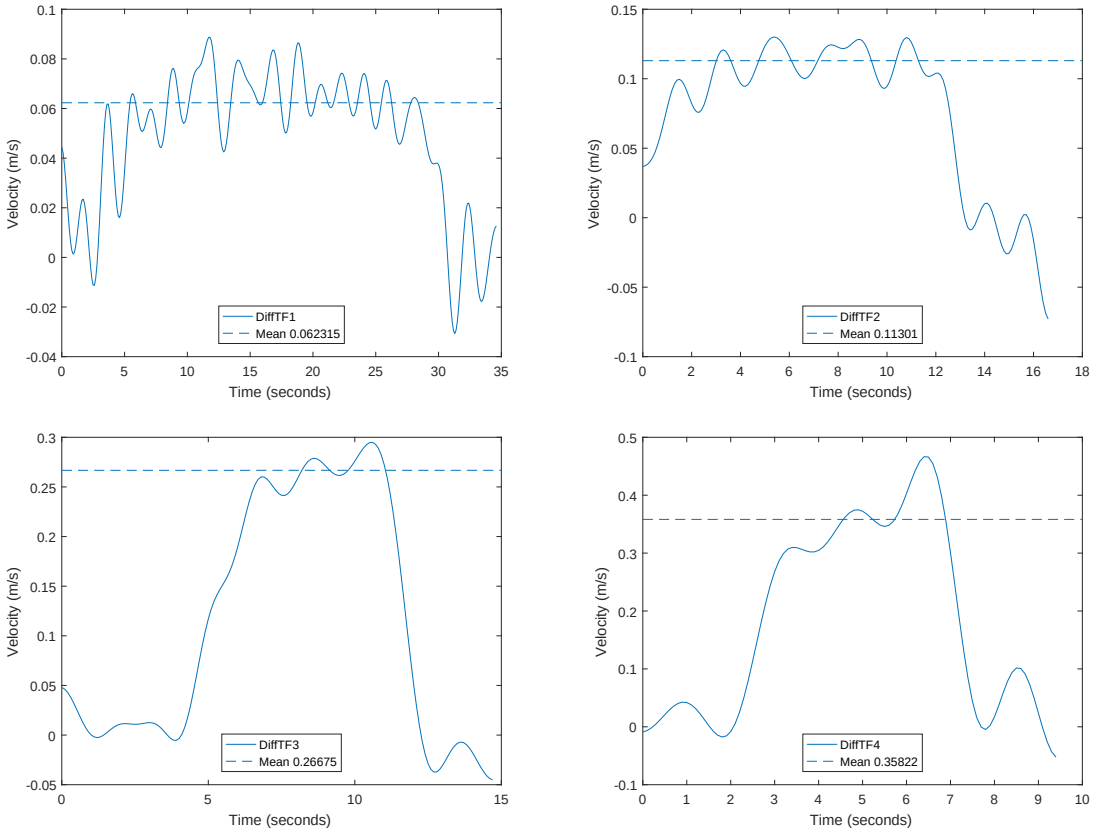


Figure 5.12: Differentiated depth data

Figure 5.12 shows the differentiated depth data, this is done in order to retrieve the velocity over time. The mean value is the calculated due to the visible variation in velocity after it reaches its constant velocity since the value of which the velocity is constant is needed. The pressure sensor provided more reliable data for the velocity in the z-direction compared to the accelerometer, as shown in a previous section.

5.4 Depth Drag Coefficient Determination

During the mathematical modelling, the following expression for the movement in the z-direction was determined.

$$m_{rov}\ddot{z} = F_t + F_g - F_d - F_b \tag{5.6}$$

By assuming that the ROV is neutrally buoyant as specified, the expression is simplified.

$$m_{rov}\ddot{z} = F_t - F_d \tag{5.7}$$

When the ROV has zero acceleration, the input thrust force will be equal to the drag force. And the drag coefficients can therefore be determined.

$$F_t = F_d \tag{5.8a}$$
$$F_t = a\dot{z}^2 + b\dot{z} \tag{5.8b}$$

The values inserted for F_t are the measured force values from equation 5.3, where the velocity $\dot{z}$ is the mean values calculated from the depth experiments in figure 5.12.

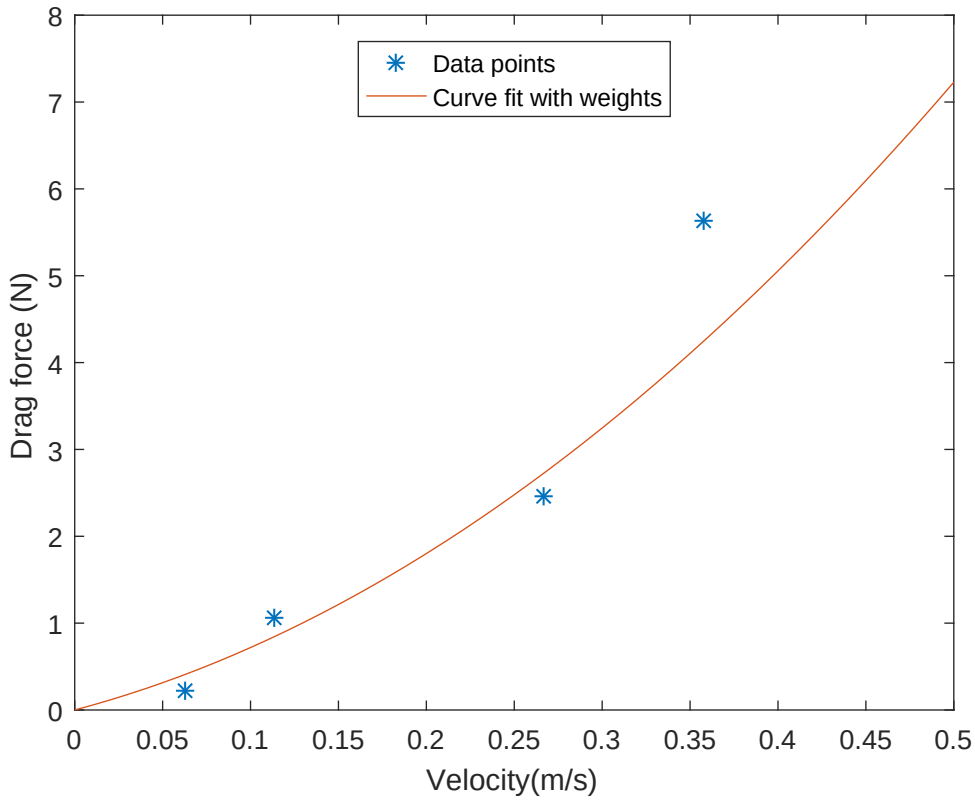


Figure 5.13: Curve fit for the drag force at different velocities

In order to improve the reliability of the calculated mean values for the velocities in figure 5.12, the curve fit shown in figure 5.13 uses weighting for each point that is

equal to the reciprocal of the variance. Due to the high variance at the data point with the highest mean velocity, it is taken less into consideration whilst creating the curve fit. The curve fit is based on the equation specified for the drag force that consists of a non-linear and linear element, as shown in equation 4.4.

$$F_d = 18.17\dot{z}^2 + 5.371\dot{z} \tag{5.9}$$

The resulting non-linear drag expression is then shown in equation 5.9.

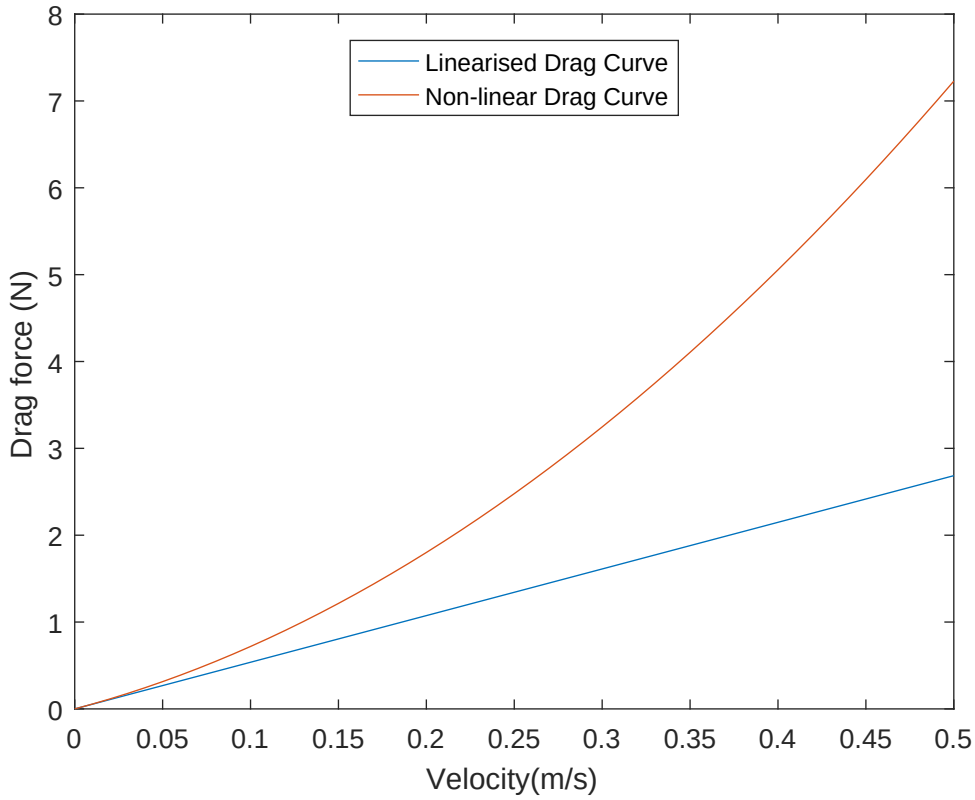


Figure 5.14: Linearisation of the drag force

Through linearisation with the operating point at 0, the linear formulation for the drag simplifies to the following.

$$F_d = 5.371\dot{z} \tag{5.10}$$

Chapter 6

State Space

6.1 Model Simplification

Based on the representation formulated in section 4.3 and the coefficients calculated during the system identification it is possible to create the state-space representation needed for the controller design. Due to simplifications and results from the system identification a SISO system has been created, since the limited amount of time prevents further identification methods for the drag coefficient for the remaining movements. Other simplifications were also made on the system, in order to create a valid system for the controller development, one being the linearisation of the various equations due to the requirements specified by linear control design.
Another simplification is the choice to the use the mean values of the remaining coefficients. Where the choice has been made to assume the forward and backwards thruster dynamics to be equal instead of modelling each direction separately. The drag coefficient for both the upward and downward directions are also assumed to be equal for the sake of simplicity, since taking multiple sets of drag coefficients or thrusters dynamics into consideration complicates the controller development and implementation. The drag coefficient used for this control design has been calculated based on the downwards motion.

$$\ddot{z} = \frac{gain}{m_{rov}} F_t - \frac{b_d}{m_{rov}} \dot{z} \tag{6.1}$$

Equation 6.1 shows the form of the differential equation for the motion along the z-axis, where the *gain* represents the conversion between input thrust percentage and b_d the linear drag coefficient. Then by inserting the calculated drag coefficient and thruster gain, the expression is as follows in equation 6.2. The thruster gain is necessary in order to make the model compatible with the later controller implementation, which will actuate the thruster using a given thruster percentage. The calculated gain then maps the thruster percentage to the desired force which the model uses to operate.

$$\ddot{z} = \frac{0.0468}{2.749} F_{t\%} - \frac{5.371}{2.749} \dot{z} \tag{6.2}$$

Equation 6.2 is then put into state-space form, with the velocity and position as states and choosing the z position as the output for the system.

$$\begin{bmatrix} \dot{z} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -1.9538 \end{bmatrix} \begin{bmatrix} z \\ \dot{z} \end{bmatrix} + \begin{bmatrix} 0 \\ 0.0170 \end{bmatrix} \begin{bmatrix} F_{t\%} \end{bmatrix} \tag{6.3a}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} z \\ \dot{z} \end{bmatrix} \tag{6.3b}$$

6.2 State-Space Control

The main idea of state-space control is to describe dynamic systems as a set of linear first order differential equations, using the idea of system states. Based on the differential equations is it possible create the state vector $\boldsymbol{x}$ for all the states and from these equations construct an $\boldsymbol{A}$-matrix, which contains the dynamics of the plant and a $\boldsymbol{B}$-matrix, referred to as the input matrix that consists of the actuator dynamics. These vectors and matrices are the combined as shown in equation 6.4a, where x is the state vector and u is the input. The output (y) is determined by a $\boldsymbol{C}$-matrix also called output matrix and a D-matrix, which is a scalar of u also referred to as feed forward matrix, see equation 6.4b.

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u \tag{6.4a}$$

$$y = \boldsymbol{C}\boldsymbol{x} + Du \tag{6.4b}$$

An advantage of this control method is that the vectors and matrices shown in equation 6.4 can be extended to become a MIMO system without much additional effort required [26]. As shown previously in equation 4.35, the combined state-space representation for all the movements possible with the ROV uses three inputs and is therefore classified as a multiple input system. If a classical control system was to be implemented on such a system, similar to equation 4.35, it would require a closed loop system for each differential equation, resulting in 6 separate closed loops and controllers. State-space allows all the equations to be combined by expanding the matrix dimensions accordingly and also developing controller suited for the matrix form.

6.3 Full-state Feedback

When designing a controller using full-state feedback it is assumed that all of the elements from the state vector are available. Which in this case requires the knowledge of the current position and velocity of the ROV in z-direction. The feedback matrix $\boldsymbol{K}$ as shown on figure 6.1, is a matrix consisting of constants that are calculated based on the desired time-domain specifications.

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{K}\boldsymbol{x} \tag{6.5}$$

Using the block diagram it is possible to determine the feedback law to be $u = -\boldsymbol{K}\boldsymbol{x}$, which then can be substituted into equation 6.4a that then yields equation 6.5.

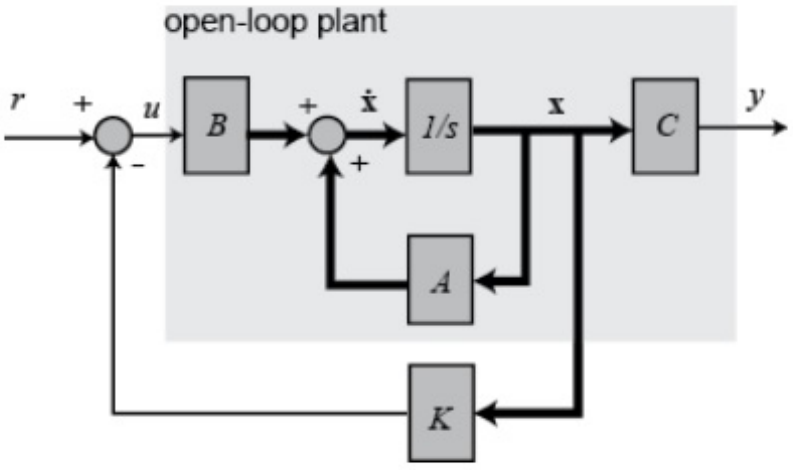


Figure 6.1: Block diagram of a full state feedback system. [27]

In order to control the system according to its specifications for the settling time and overshoot, the desired poles can be placed as needed in the $\boldsymbol{K}$-matrix. The gains allow the selection of dominant pole locations so that they satisfy the given specifications, which in this case are specified in section 2.3.

$$\boldsymbol{K} = \begin{bmatrix} K_1 & K_2 & \dots & K_n \end{bmatrix} \tag{6.6}$$

For an nth-order system there are n entries in the $\boldsymbol{K}$-matrix. In order to freely put poles anywhere in the complex plane the system has to be controllable. This can be checked by calculating the controllability, which can be seen in equation 6.7, this matrix must have full rank for the system to be controllable.

$$Co = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{A}\boldsymbol{B} & \boldsymbol{A}^2\boldsymbol{B} & \dots & \boldsymbol{A}^{n-1}\boldsymbol{B} \end{bmatrix} \tag{6.7}$$

By using equation 6.8a and 6.8b it is possible to determine the damping ratio and the natural undamped frequency of the system, which depends on the specified time-domain requirements. Where in equation 6.8a ζ is the damping ratio, Mp is the specified percentage overshoot and in equation 6.8b ts is the desired settling time and w_n the natural frequency.

$$M_p = e^{-\pi\zeta/\sqrt{1-\zeta^2}} \tag{6.8a}$$

$$e^{-\zeta w_n t_s} = 0.01 \tag{6.8b}$$

The calculated natural frequency and damping ratio are then inserted into equation 6.9 and solved for s in order to get the desired pole locations.

$$(s^2 + 2\zeta w_n s + w_n^2) = 0 \tag{6.9}$$

By combining the desired pole locations from equation 6.9 with the characteristic equation for the closed loop in equation 6.10 the $\boldsymbol{K}$-matrix can be calculated. [28]

$$(s - p1)(s - p2) = det[s\boldsymbol{I} - (\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})] \tag{6.10}$$

In order to introduce the reference input into the full-state feedback control, the feedback law can be modified to $u = -\boldsymbol{K}\boldsymbol{x} + r$, which will leave the system with a non-zero steady-state error. This can be dealt with using reference scaling denoted as $\bar{N}$, so that the feedback control law becomes $u = -\boldsymbol{K}\boldsymbol{x} + \bar{N}r$. The scaling value is calculated so that there zero steady-state error, which in this case yields the matrices in equation 6.11.

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & D \end{bmatrix} \begin{bmatrix} \boldsymbol{N_x} \\ N_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} \tag{6.11}$$

$\boldsymbol{N_x}$ and N_u respectively are the required scaling to achieve zero steady-state error, where the scaling modifies feedback law to $u = -\boldsymbol{K}(\boldsymbol{x} - \boldsymbol{N_x}r) + N_u r$.

$$\begin{bmatrix} \boldsymbol{N_x} \\ N_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{C} & D \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} \tag{6.12}$$

By rearranging equation 6.11 it is possible to solve for the scaling values, as shown in equation 6.12.

$$u = \boldsymbol{K}(\boldsymbol{x} - \boldsymbol{N_x}r) + N_u r = -\boldsymbol{K}\boldsymbol{x} + \bar{N}r \tag{6.13}$$

The values are then inserted into the control law and by collecting terms including the reference r it possible to calculate the scaling denoted as $\bar{N}$ [28].

6.4 Observer

Full state feedback has its limitations since all states have to be known at all times. If guaranteed knowledge of all states is not possible an estimator can be used in order to estimate the states of the system and be used to calculate the needed control signal. A system needs to be observable in order to use an observer, the system is observable if its observability matrix (O) has full rank.

$$O = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ . \\ . \\ \boldsymbol{CA}^{n-1} \end{bmatrix} \tag{6.14}$$

An estimator consists of a copy of the original system dynamics and is used to compare the expected output with the actual output, in order to create a state estimate. A block diagram of an estimator alongside the open-loop plant can be seen in figure 6.2.

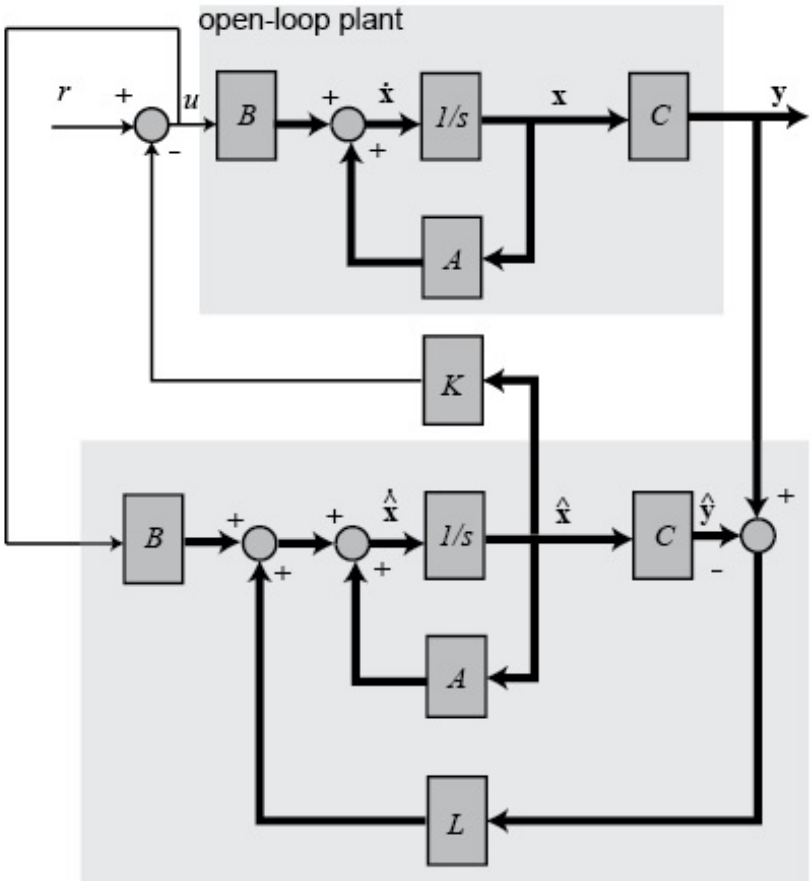


Figure 6.2: Block diagram for closed loop estimator. [27]

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}u + \boldsymbol{L}(y - \boldsymbol{C}\hat{\boldsymbol{x}}) \tag{6.15}$$

$$\hat{y} = \boldsymbol{C}\hat{\boldsymbol{x}} \tag{6.16}$$

The estimator has a feedback gain connected to the error between the measured output and estimated output, which gonverns the convergence speed of the estimated state vector to the actual states. The estimator needs to be faster than the original system, a general rule of thumb is that the poles should be 2-6 times larger than those in the original system, though the faster the estimator the more noise sensitive it becomes [29]. The feedback gain for the $\boldsymbol{L}$-matrix can be calculated similar to the $\boldsymbol{K}$-matrix. $\boldsymbol{L}$ using equation 6.17 by adding desired pole locations.

$$(s - p1)(s - p2) = det[s\boldsymbol{I} - (\boldsymbol{A} - \boldsymbol{LC})] \tag{6.17}$$

$$\boldsymbol{L} = \begin{bmatrix} L_1 \\ L_2 \\ . \\ . \\ L_n \end{bmatrix} \tag{6.18}$$

Combining the old system and the estimator a new system can be created with new $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$-matrices. [27]

$$\boldsymbol{A}_{es} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{BK} & \boldsymbol{BK} \\ zeros(size(\boldsymbol{A})) & \boldsymbol{A} - \boldsymbol{LC} \end{bmatrix} \tag{6.19}$$

$$\boldsymbol{B}_{es} = \begin{bmatrix} \boldsymbol{B}\bar{N} \\ zeros(size(\boldsymbol{B})) \end{bmatrix} \tag{6.20}$$

$$\boldsymbol{C}_{es} = \begin{bmatrix} \boldsymbol{C} & zeros(size(\boldsymbol{C})) \end{bmatrix} \tag{6.21}$$

Where the new $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$-matrices can be substituted into equation 6.4a, combining the state-vector of the plant and the estimated state-vector.

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\tilde{\boldsymbol{x}}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{BK} & \boldsymbol{BK} \\ zeros(size(\boldsymbol{A})) & \boldsymbol{A} - \boldsymbol{LC} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \tilde{\boldsymbol{x}} \end{bmatrix} \tag{6.22}$$

By analysing the block-diagram in figure 6.2 it is possible to determine the calculation of control variable u, where it is dependant on the designed control gains from the matrix $\boldsymbol{K}$ and the reference including the scaling as mentioned in section 6.3.

$$u = -\boldsymbol{K}\hat{\boldsymbol{x}} + \bar{N}r \tag{6.23}$$

6.5 Integral Control

When implementing reference inputs for the state-space controller using $\bar{N}$, it will not guarantee any robustness in terms of reference point tracking, due to the sensitivity to inaccuracy between the mathematical model and real plant. Any changes in the systems parameters will lead to a non zero steady state error. Instead a more robust alternative to reference tracking is integral control, which uses previous errors between the reference and measured output to track the error and eliminate it. A block diagram of a system with integral control can be seen in figure 8.5.

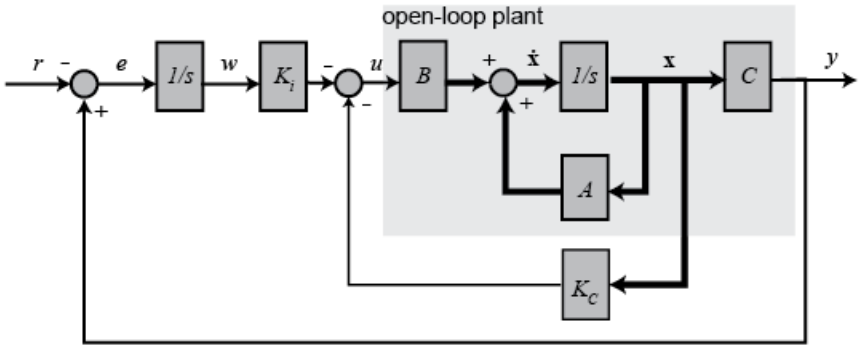


Figure 6.3: Block diagram of an integral control system. [30]

By using integral control it introduces a new state, w, to the system and the reference r becomes an additional input. There the state space model is augmented in order to in-cooperate the new state and will look as stated in equation 6.24.

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \boldsymbol{A} \\ 0 & \boldsymbol{C} \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} + \begin{bmatrix} \boldsymbol{B} \\ 0 \end{bmatrix} u - \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \tag{6.24}$$

The uses of integral control introduces an integral gain, which can be tuned based on the calculated gains in control matrix $\boldsymbol{K}$. The reason why integral control is favoured over scaling the reference input is due to the fact that integral control accumulates the errors and therefore acts as memory of the previous errors and can adjust accordingly, where the scaling using $\bar{N}$ can do no such thing.

Chapter 7

Controller Development

After the construction of the $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$-matrix, as described in equation 6.3, the behaviour of the open loop system was simulated and the results can be seen in figure 7.1.

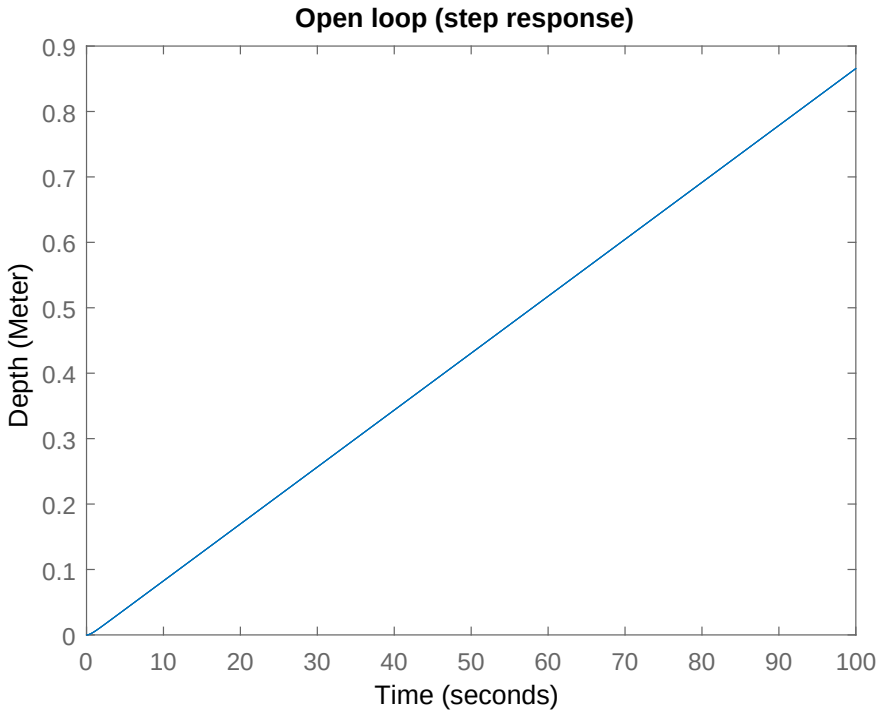


Figure 7.1: Simulation of open loop system.

As expected the system was stable, changing its position slowly over time when subjected to a step response at 1% thrust. In order to reach a steady-state based on a reference input a feedback loop needs to be introduced. This can be done using full-state feedback. To ensure that full-state feedback is applicable to this system it needs to be checked to see if it is controllable according to the theory in section

6.7. Using Matlab the controllability matrix was constructed and it had full rank, which means that the system is controllable. To find suitable poles depending on the time-domain specifications from 2.3, equation 6.8a and 6.8b was utilized to calculate the equivalent damping ratio and natural frequency at the desired specifications.

$$\zeta = 0.591 \tag{7.1a}$$
$$w_n = 0.779 \tag{7.1b}$$

The poles are then calculated by substituting ζ and w_n into equation 6.9 and solving for the poles, this was done using Matlab resulting in pole locations as seen in equation 7.2.

$$pole1 = -0.46 - 0.628i \tag{7.2a}$$
$$pole2 = -0.46 + 0.628i \tag{7.2b}$$

The feedback gain can then be constructed in the $\boldsymbol{K}$-matrix by adding the poles into equation 6.10 or using a pole placement command in Matlab, this then yields the control gains in equation 7.3.

$$\boldsymbol{K} = \begin{bmatrix} 35.7 & -60.77 \end{bmatrix} \tag{7.3}$$

After adding full state feedback and simulating the results as shown in figure 7.2, it produces an overshoot that is below 10 % and the settling time is 10 seconds, which satisfy the specified system specifications.

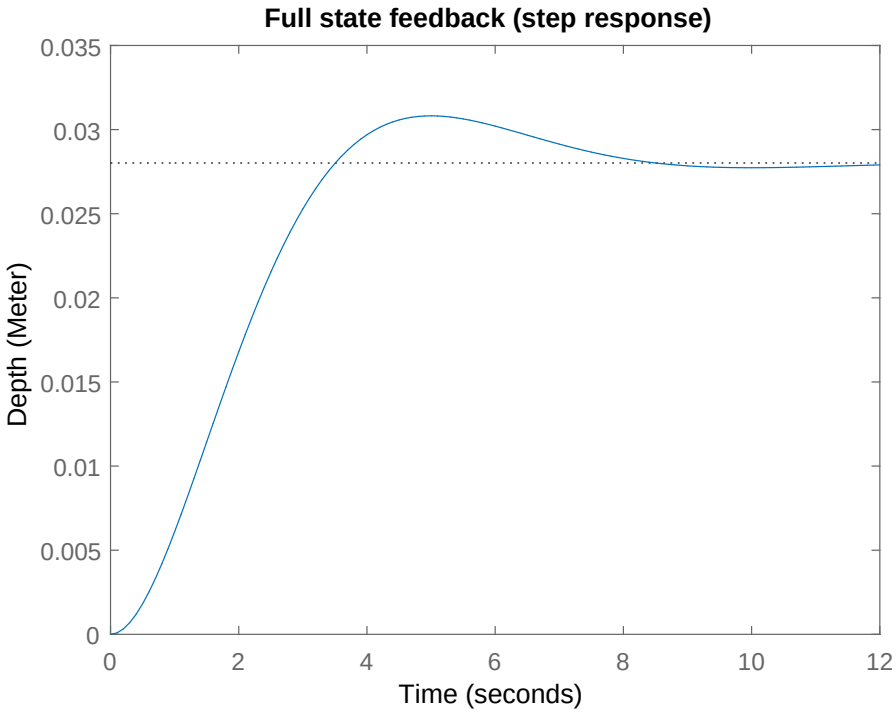


Figure 7.2: Full state feedback.

The steady state error, however, is large and scaling for the reference input needs to be added to the system. The procedure for calculating $\bar{N}$ is shown in equation 6.13, when calculated based on the control gains in equation 7.3 yields a scaling factor of $\bar{N} = 35.7$. As seen in figure 7.3 adding the reference input scaling reduces the steady state error to less than 1 percent and all the system specifications are met.

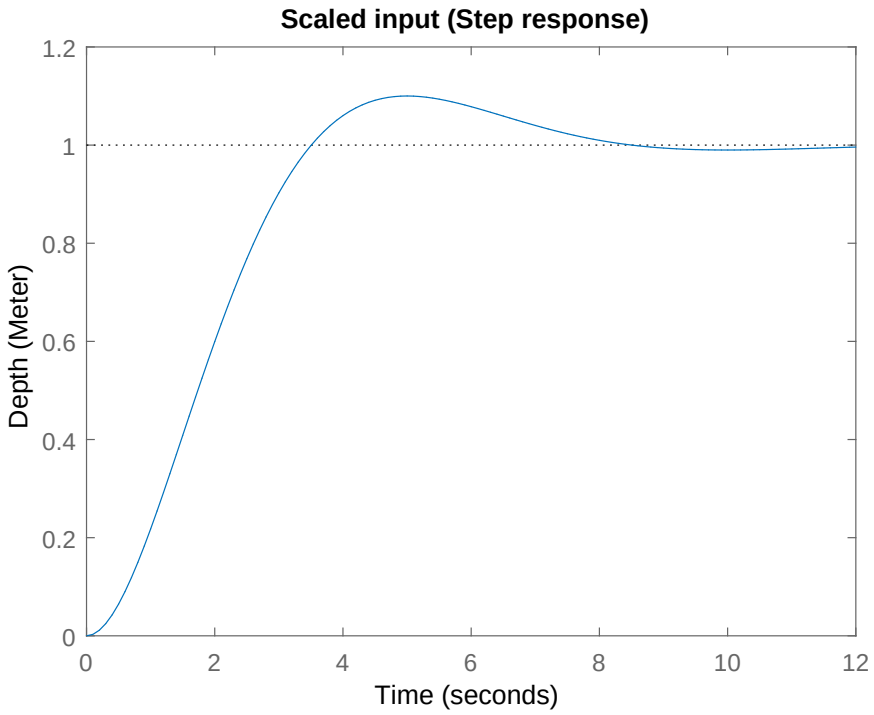


Figure 7.3: Full state feedback with scaled input.

As mentioned earlier in section 6.4 all states have to be known at all time in order to utilize full-state feedback. Due to limited sensor data and the sometimes unreliable sensor measurements it was chosen to use an estimator in order estimate the system states. In order to use an estimator the observability needs be checked using equation 6.14, which when calculated returned that the system is observable. The estimators poles are then chosen to be twice as large compared to the poles from the original system, this results in the estimator being twice as fast as the original system.

$$pole1_{es} = 2 \cdot pole1 = 2(-0.46 - 0.628i) = -0.92 - 1.257i \tag{7.4a}$$
$$pole2_{es} = 2 \cdot pole2 = 2(-0.46 + 0.628i) = -0.92 + 1.257i \tag{7.4b}$$

The feedback gain $\boldsymbol{L}$ was then calculated using equation 6.17 or using a pole placement command in Matlab. This then results in gains seen in equation 7.5.

$$\boldsymbol{L} = \begin{bmatrix} -0.112 \\ 2.647 \end{bmatrix} \tag{7.5}$$

The step response behaviour of the plant combined with the estimator using reference scaling can be seen in figure 7.4 and similar to the full-state feedback also satisfied the time-domain specifications of 10% overshoot and a settling time of 10 seconds.

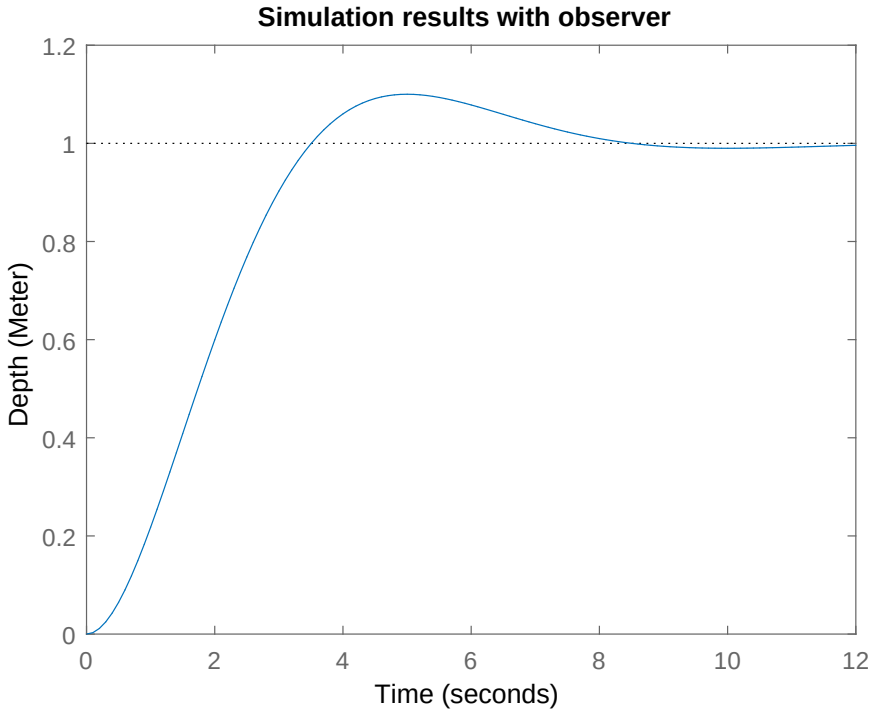


Figure 7.4: Step response with an estimator

The combined plant and estimator matrices are then expressed as new $\boldsymbol{A}$,$\boldsymbol{B}$ and $\boldsymbol{C}$-matrices using equation 6.19 , 6.20 and 7.10. The new matrices were used to simulate the system and are seen in equation 7.6, 7.7 and 7.8.

$$\boldsymbol{A}_{es} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -0.61 & -0.92 & 0.61 & -1.03 \\ 0 & 0 & 0.11 & 1 \\ 0 & 0 & -2.65 & -1.95 \end{bmatrix} \tag{7.6}$$

$$\boldsymbol{B}_{es} = \begin{bmatrix} 0 \\ 0.61 \\ 0 \\ 0 \end{bmatrix} \tag{7.7}$$

$$\boldsymbol{C}_{es} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \tag{7.8}$$

The dimensions of the original matrices have changed since 2 states, the estimated states($\tilde{\boldsymbol{z}}$ and $\dot{\tilde{\boldsymbol{z}}}$), have been added the original system and knowing $\tilde{\boldsymbol{z}} = \boldsymbol{z}$ - $\hat{\boldsymbol{z}}$ the

new combined state space model can be seen in equation 7.9.

$$\begin{bmatrix} \dot{\boldsymbol{z}} \\ \ddot{\boldsymbol{z}} \\ \dot{\tilde{\boldsymbol{z}}} \\ \ddot{\tilde{\boldsymbol{z}}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} - \boldsymbol{BK} & \boldsymbol{BK} \\ zeros(size(\boldsymbol{A})) & \boldsymbol{A} - \boldsymbol{LC}) \end{bmatrix} \begin{bmatrix} \boldsymbol{z} \\ \dot{\boldsymbol{z}} \\ \tilde{\boldsymbol{z}} \\ \dot{\tilde{\boldsymbol{z}}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}\bar{N} \\ zeros(size(\boldsymbol{B})) \end{bmatrix} F_{t\%} \tag{7.9}$$

Since $\boldsymbol{A}$, $\boldsymbol{BK}$, zeros(size($\boldsymbol{A}$)) and $\boldsymbol{LC}$ are all 2×2 matrices The new $\boldsymbol{A}_{es}$-matrix will have dimension 4×4. $\boldsymbol{B}\bar{n}$ is a 2×1 matrix and zeros(size($\boldsymbol{B}$)) is also a 2×1 matrix so the new $\boldsymbol{B}_{es}$-matrix will have dimension 4×1.

$$\boldsymbol{C}_{es} = \begin{bmatrix} \boldsymbol{C} & zeros(size(\boldsymbol{C})) \end{bmatrix} \tag{7.10}$$

Since $\boldsymbol{C}$ has dimension 1×2 and zeros(size($\boldsymbol{C}$)) also have dimension 1×2, $\boldsymbol{C}_{es}$ will have dimension 1×4.

Chapter 8

Non-linear and Linear Model Comparison

When linearising a system it is important that the behaviour and important characteristics are similar to the non-linear system In order to gain similar performance when implementing a linear controller on the real-life non-linear plant. Therefore in order to check validity of the linearised model the two systems will be compared in simulations.

8.1 Open Loop

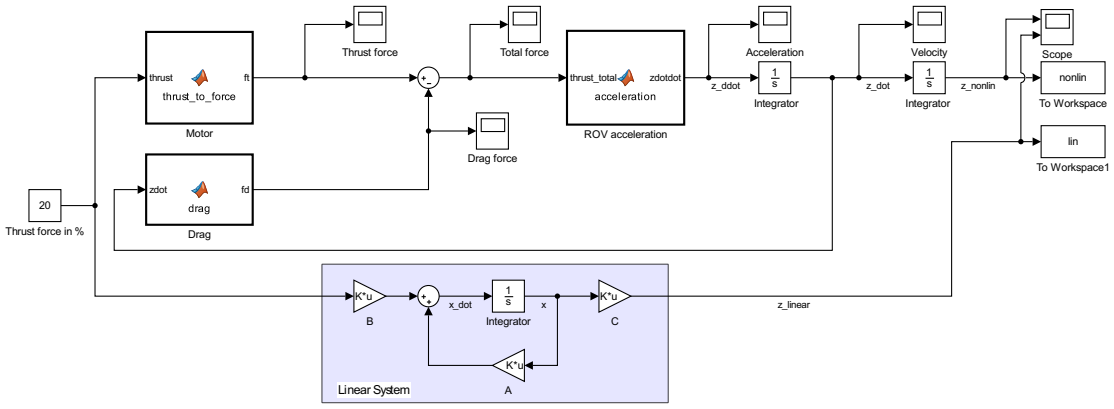


Figure 8.1: Simulink implementation for the open-loop linear/non-linear comparison

First the open loop behaviour is simulated and compared where the Simulink implementation can be seen in figure 8.1. The non-linear model is structured in separate function blocks where each block contains their individual non linear dynamics and is summed or substracted accordingly. The function block containing the non-linear drag dynamics uses the calculated expression from equation 5.9 and thrust dynamics from equation 5.4. Main block consisting of the ROV dynamics uses equation 4.6

where the buoyancy is assumed neutral. The simulation performed in figure 8.2 is done with an input percentage of 20% thrust.

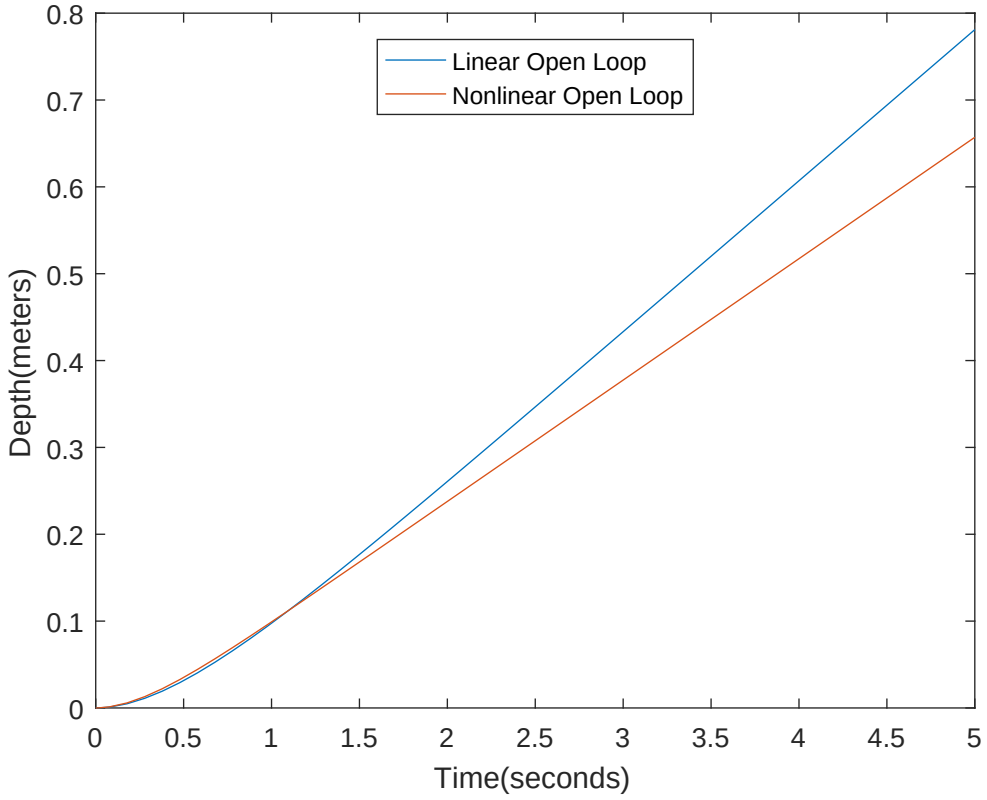


Figure 8.2: Open loop comparison between the linear/non-linear model

From figure 8.2 it can seen that the models are very similar during the initial seconds then over time the linear system moves at a greater velocity compared to non-linear. This was expected due the linear model having less drag compared to the non-linear model. Since the quadratic part of equation 4.4 adds additional drag and will also increase the drag at higher velocities compared to the linearised expression.

8.2 Estimator with Reference Scaling

Figure 8.3 shows the Simulink implementation of the designed estimator alongside the non-linear system. The input to the non-linear system is altered depending on the estimated states and reference scaling, similarly to how the linear system is controlled.

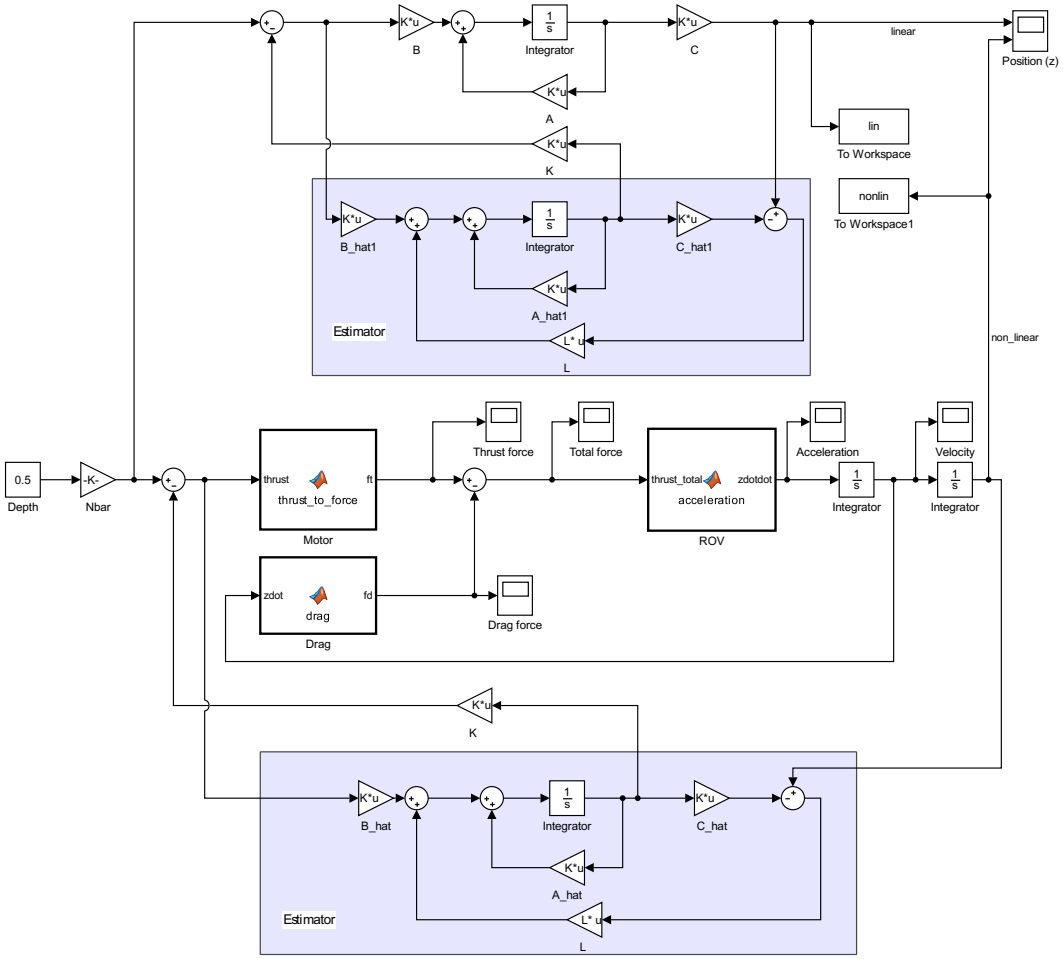


Figure 8.3: Simulink implementation for the estimator linear/non-linear comparison

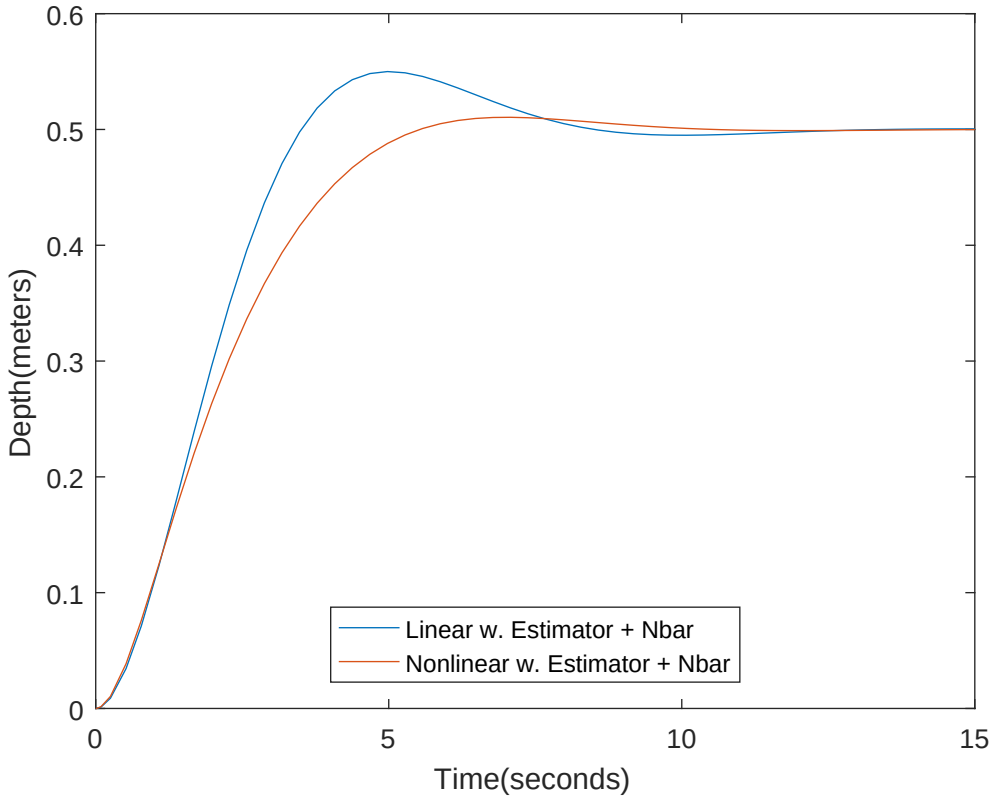


Figure 8.4: Comparison of linear and non linear model including an estimator.

The result of the simulation can be seen in 8.4, where the reference point is set to 0.5 meters. It shows that the linear model tracks the non-linear model very precisely in the beginning of the simulation and also at the steady-state. When the system is at greater velocities the non-linear model's quadratic drag coefficient influences the performance of the system, where in this case, it dampens the overshoot which is acceptable due to the requirement being less than 10%.

8.3 Estimator with Integral Control

The final comparison is between the linear and non-linear models using an estimator and integral control as seen in figure 8.5.

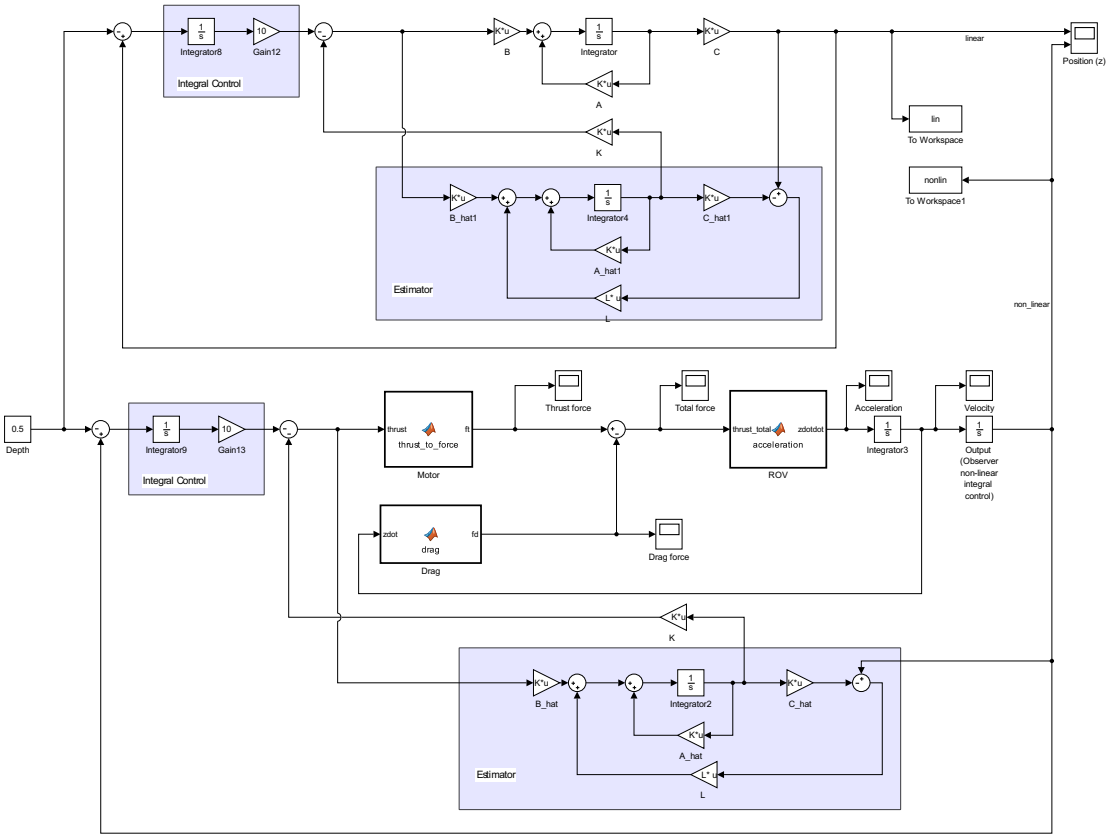


Figure 8.5: Simulink implementation of non-linear/linear using an estimator and integral control

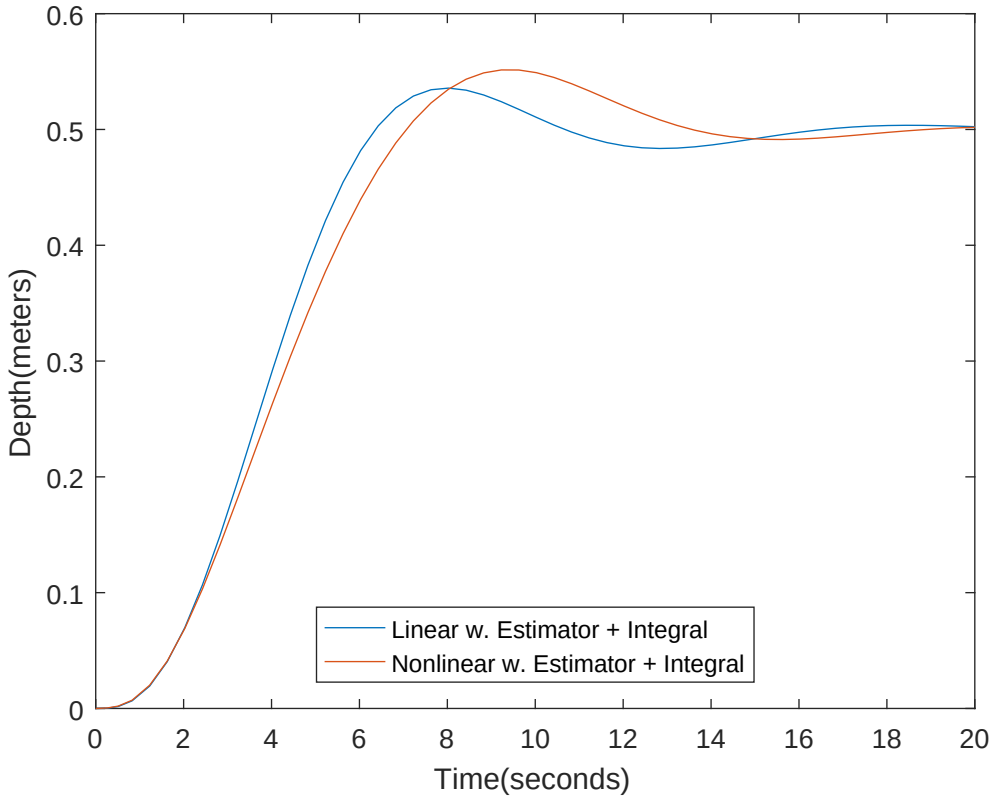


Figure 8.6: Comparison of linear and non linear model.

The simulation shown in figure 8.6 shows behaviour similar to the previous model but in this case the linear model has less overshoot than the non-linear one. The linear model tracks the non-linear model precisely in the beginning but the non-linear model takes longer to settle at the steady-state compared to linearised system. This can be due to the integral windup introduced alongside the integral control.

Chapter 9

Implementation

9.1 Discretization and Difference Equations

The controller is to be implemented on the BeagleBone which currently communicates with the thrusters and IMU over serial, in order to retrieve the depth data, the incoming serial data is parsed and the depth is stored in a separate variable. Currently the depth data is sampled at 10Hz, which is going to be the sampling time of system if the controller is compatible.

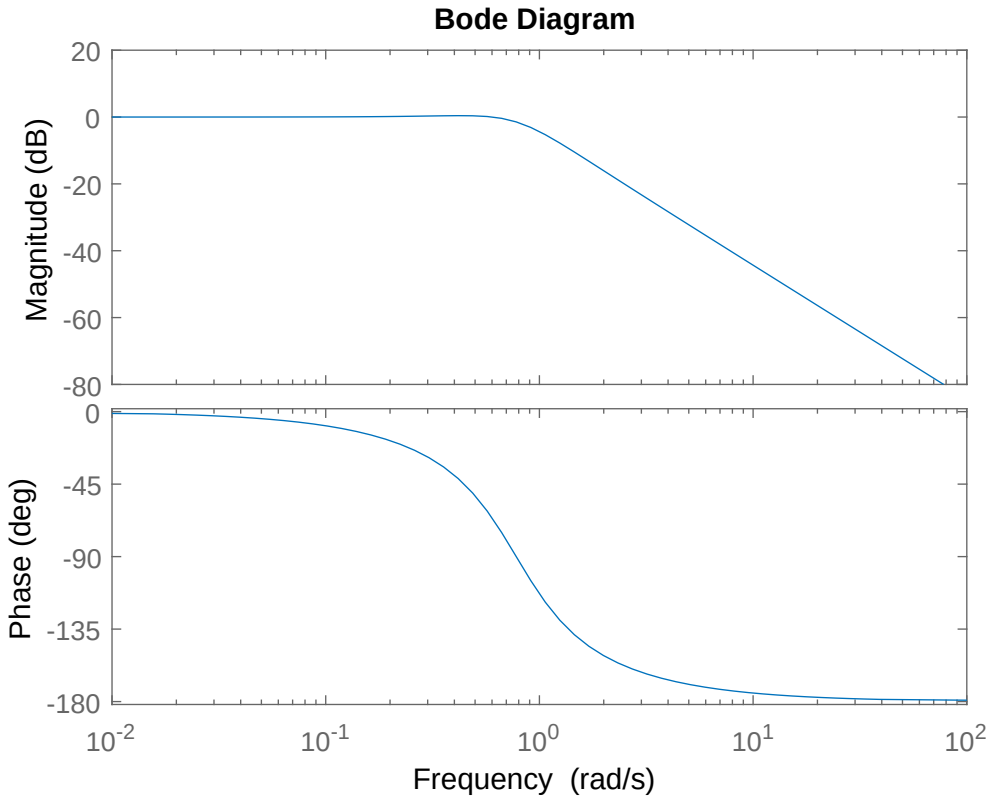


Figure 9.1: Bode diagram for the closed loop system with an estimator

In order to check the compatibility of the controller for implementation at 10 Hz, it is necessary to check the closed loop system bandwidth. Figure 9.1 shows the bode plot of the closed loop system, where the bandwidth is at $-3dB$, which in this case is $0.9028\frac{rad}{s}$.

$$0.9028\frac{rad}{s} = 0.1437Hz \tag{9.1}$$

The choice of sampling frequency should be 20 times greater than the closed loop system bandwidth, which in this case means that the sampling frequency should at least be 2.874Hz. This means that the frequency choice of 10Hz is well within the requirements of the closed loop system.

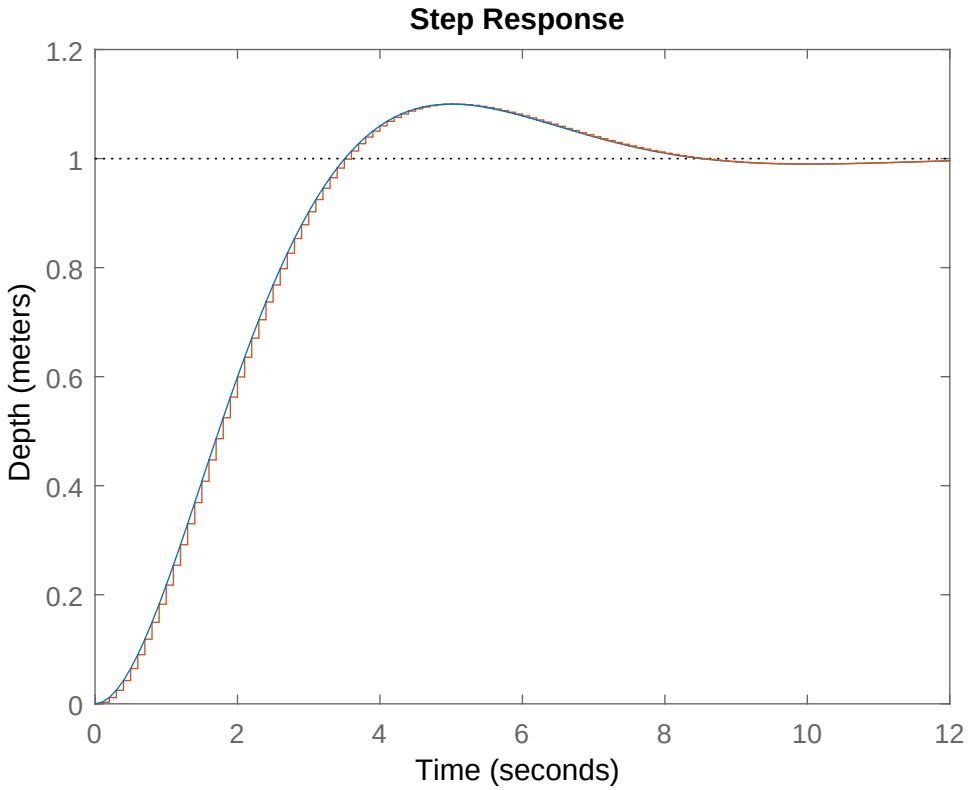


Figure 9.2: Comparison between the discrete and continues observer

Using Matlab a discretization of the controller using an estimator is discretized and the step response is plotted on top of the continuous step response, this can be seen in figure 9.2.

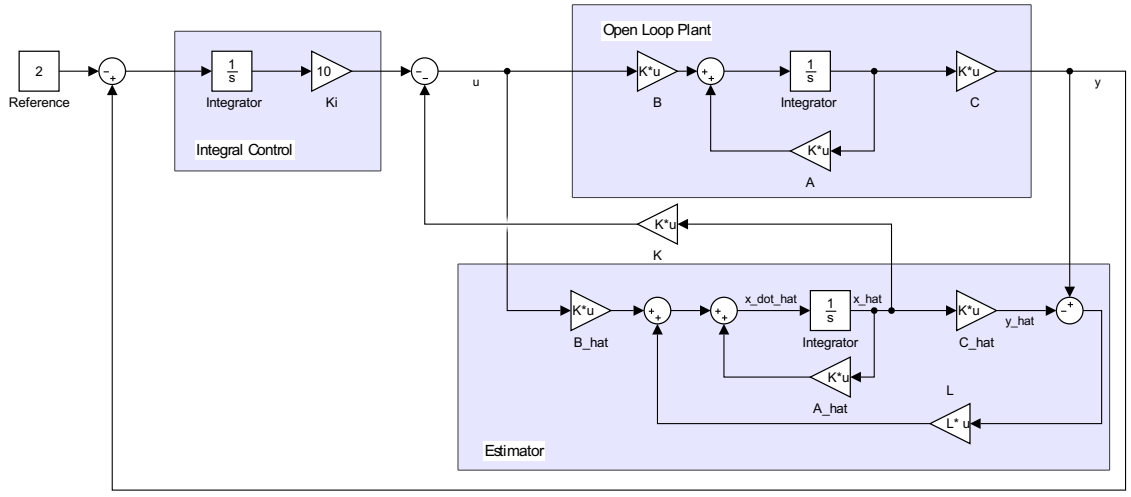


Figure 9.3: Block diagram for the estimator with integral control

In order to implement the controller, the block diagram can be utilized in order to create the expressions needed to calculate the control variable u, this expression is derived from the block diagram shown in figure 9.3 and formulated in equation 9.2.

$$u = \frac{K_i}{s}(y - r) - \boldsymbol{K}\hat{\boldsymbol{x}} \tag{9.2}$$

The control variable u, which is the input to the thrusters, is calculated using the summation of previous reference errors multiplied by the integral gain and using the designed control matrix **K** multiplied by the estimated states.

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A}\hat{\boldsymbol{x}} + \boldsymbol{B}u + \boldsymbol{L}(y - \boldsymbol{C}\hat{\boldsymbol{x}}) \tag{9.3}$$

An expression for the derivative of estimated states can also be derived from the block diagram in figure 9.3 and be found in equation9.3.

$$\frac{\hat{\boldsymbol{x}}(t_{k+1}) - \hat{\boldsymbol{x}}(t_k)}{h} = \boldsymbol{A}\hat{\boldsymbol{x}}(t_k) + \boldsymbol{B}u(t_k) + \boldsymbol{L}(y(t_k) - \boldsymbol{C}\hat{\boldsymbol{x}}(t_k)) \tag{9.4}$$

Difference equations formed based on the expression for $\dot{\hat{\boldsymbol{x}}}$. The derivative in equation 9.3 can be estimated by a difference, where h represents the chosen sampling time.

$$\hat{\boldsymbol{x}}(t_{k+1}) = \hat{\boldsymbol{x}}(t_k) + h(\boldsymbol{A}\hat{\boldsymbol{x}}(t_k) + \boldsymbol{B}u(t_k) + \boldsymbol{L}(y(t_k) - \boldsymbol{C}\hat{\boldsymbol{x}}(t_k))) \tag{9.5a}$$

$$\hat{\boldsymbol{x}}(t_{k+1}) = (\boldsymbol{I} + h\boldsymbol{A} - h\boldsymbol{L}\boldsymbol{C})\hat{\boldsymbol{x}}(t_k) + h\boldsymbol{B}u(t_k) + h\boldsymbol{L}y(t_k) \tag{9.5b}$$

By further simplifying equation 9.4 the expression in equation 9.5b is found. This equation can then be implemented on the BeagleBone and should only be executed at an interval defined by h, which in this case would be 10Hz, since the this is sampling period.

9.2 C Implementation

In order to ensure that equation 9.5b is recalculated every 10Hz, interrupts can be utilized on the BeagleBone in order to update the estimated states. In this case an interval timer was configured to trigger every 0.1 seconds and in that way update the estimate accordingly. Since the BeagleBone runs a flavour of debian Linux it is possible to set up an interval timer in software.

```
//Sets up the SIGARLM to call update_estimate when triggered
if (signal(SIGALRM, (void (*)(int)) update_estimate) == SIG_ERR) {
  perror("Unable to catch SIGALRM");
      exit(0);
}
//Calculates the interval in seconds, based on input ms
it_val.it_value.tv_sec =     INTERVAL/1000;
//Calculate the interval in useconds, based on input ms
it_val.it_value.tv_usec =    (INTERVAL*1000) % 1000000;
it_val.it_interval = it_val.it_value;
//Sets the interval timer
if (setitimer(ITIMER_REAL, &it_val, NULL) == -1) {
  perror("error calling setitimer()");
  exit(0);
}
```

Code Listing 9.1: Setting up the interval timer for 10Hz

Code listing 9.1 shows how the function *update_estimate()* is associated with the SIGARLM, which is a signal that activates whenever a specified time interval is expired. Whenever 10Hz has elapsed it will update the calculations specified in equation 9.5a. The matrix operations are done using multidimensional arrays and for loops.

```
//Calculates h*A*x and stores it in  Ax
for(i=0; i<2; i++)
{
  for(j=0; j<1; j++)
  {
    Ax[i][j] = 0;
    for(k=0; k<2; k++)
      Ax[i][j] = h * A[i][k] * x[k][j] + Ax[i][j];
  }
}
```

Code Listing 9.2: Matrix multiplication

The *update_estimate()* function does all the matrix multiplications separately using for loops and then adds all the results together. Code listing 9.2 show the matrix multiplication of $\mathbf{A}$ and $\mathbf{x}$ while taking the sampling period h into consideration also.

```
1  //Calculates x  based on previous x, Ax, Bu and est (L(y-Cx))
   for(i=0; i<2; i++)
3  {
     for(j=0; j<1; j++)
5      x[i][j] = x[i][j] + Ax[i][j] + Bu[i][j] + est[i][j];
   }
```

Code Listing 9.3: Adding all the matrix multiplication

The estimate needed to calculate the control variable is shown in code listing 9.3, which is the addition of all the needed matrix products. The remaining calculations for the *update_estimate()* function can be seen found in appendix 12.3.

```
   //Runs continuously until SIGINT
2  while(input <= 0) {
     //Reads from serial until a line terminating character
4    res = read(fd, buf, 30);
     //Set the end of string so it can be printed
6    buf[res] = 0;
     fprintf(fp,"%s", buf, res);
8    //Parses the incoming data
     sscanf(buf, "deap;%f;", &depth);
10   y = depth;
     //Checks if the calculated control variable is oversaturated
12   if(u > 100.0) u = 100.0;
     if(u < -100.0) u = -100.0;
14   //Sends the calculated u to the thrusters
     sprintf(cmd,"echo \"lift(%d);\" -n > /dev/ttyO1", (int)u);
16   system(cmd);
     //Status print to the terminal for current depth, integral and
    thruster value
18   printf("int: %f - u: %f - depth: %f\n",integral,u, depth);

20 }
```

Code Listing 9.4: Main loop for the controller implementation

The main loop shown in code listing 9.4 reads the incoming depth data from the serial connection and handles the actuation of the thruster by echoing the desired thruster value over serial. The main loop handles the saturation since the thruster input range is from -100 to 100. In order to stop the control loop the program waits for an interrupt from the user that then terminates the endless loop and finishes the program. The full code for the controller implementation can be found in appendix 12.6.

9.3 Testing

9.3.1 Initial Estimator and Integral Control

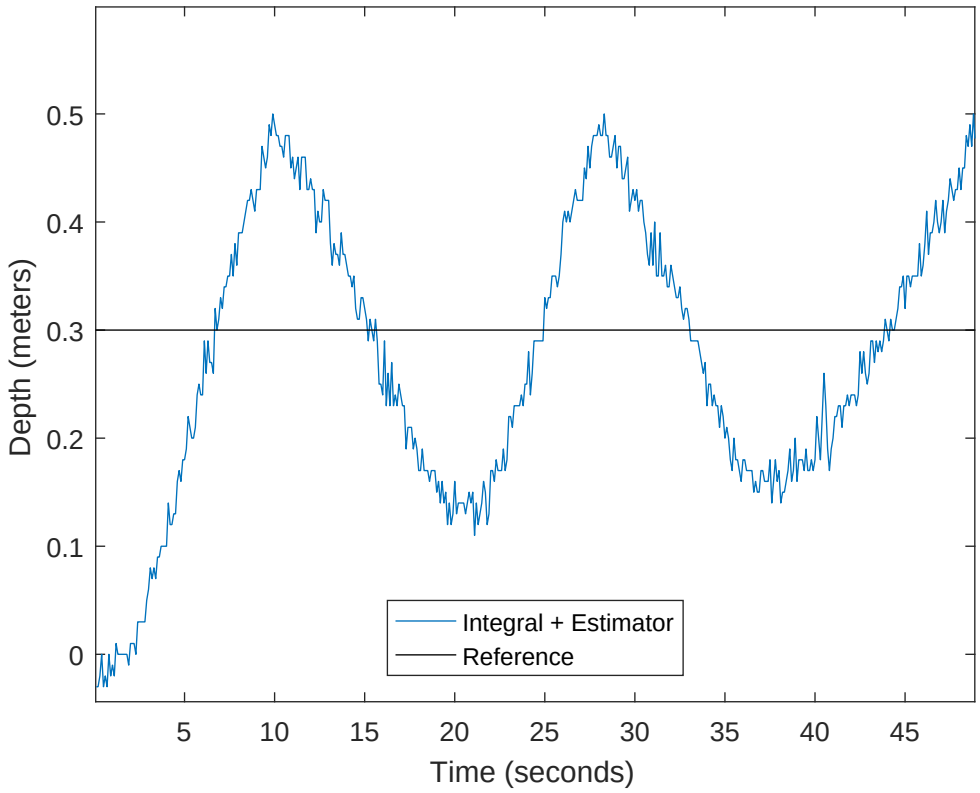


Figure 9.4: Depth test with ref 0.3m

Figure 9.4 shows the initial depth test with a reference point of 0.3 meters. The implemented controller uses the control gains designed in equation 7.3 and estimator gains designed in equation 7.5. Integral control was implemented due to the desired robustness in terms of reference tracking and was implemented with an integral gain of 10, which is equivalent to $\frac{1}{3}$ of the primary control gain from equation 7.3.
The ROV fluctuates around the reference point at almost ±20cm. Initially it was assumed that the instability happened due to saturation issues, after investigating the control signal it was discovered not to be the case. The integral control was also implemented without anti-windup, which also impacts the performance of the controller introducing a larger overshoot. The estimator in this initial test was designed to be twice as fast compared to the control gains, which possibly also meant that the state estimation was too slow.

9.3.2 Improved Estimator and Reference Scaling

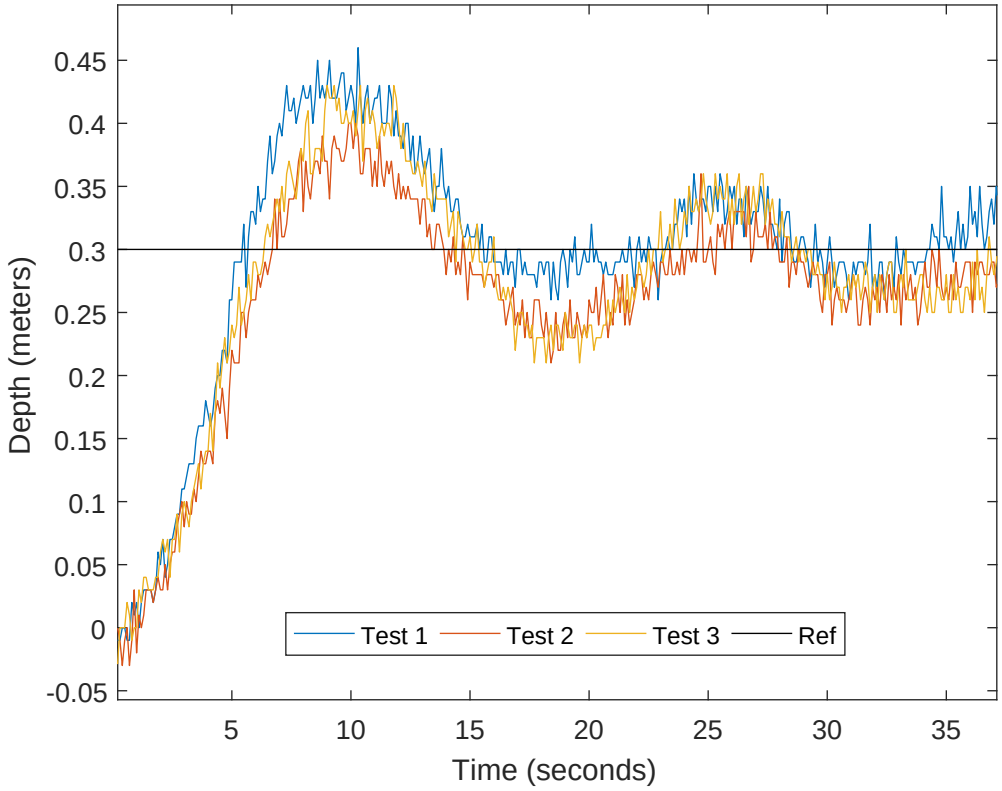


Figure 9.5: Depth test with ref 0.3m

After creating a new estimator, which poles are 6 times faster than the poles used for the control gains and switching to reference scaling using $\bar{N}$, the overall performance and steady-state error is improved to an acceptable result. Figure 9.5 shows the comparison between three experiments with a reference point of 0.3m. The fluctuation has been reduced to ±5cm after the initial overshoot, compared to the previous ±20cm.

$$K = \begin{bmatrix} 35.6977 & -60.7509 \end{bmatrix} \tag{9.6a}$$

$$L = \begin{bmatrix} 4.5724 & 6.8266 \end{bmatrix}^T \tag{9.6b}$$

$$\bar{N} = 35.6977 \tag{9.6c}$$

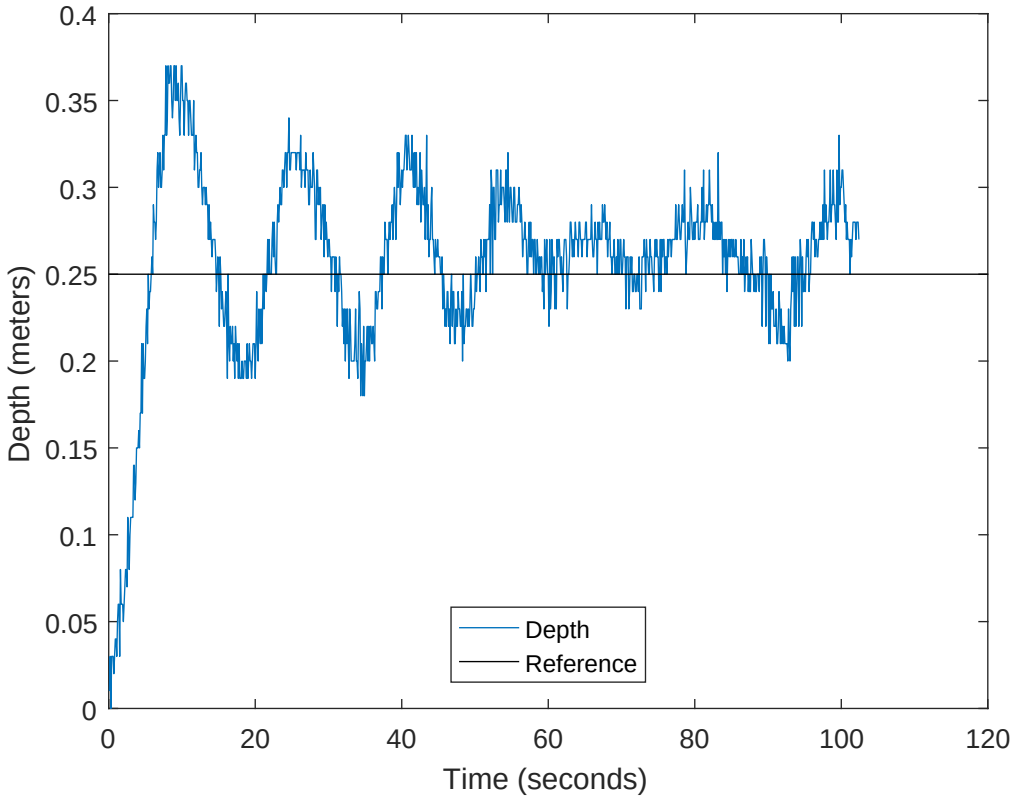


Figure 9.6: Depth test with ref 0.25m

Figure 9.6 is simular to figure 9.5 just over a longer time span in order to show stability. These test were performed on separate days and on the second day, shown in figure 9.6, the buoyancy of the ROV was not as neutral as the previous days. This does not seem to have had a large impact on the overall performance of the system. The lack of convergence towards zero steady-state error is partly due to the use of input scaling ($\bar{N}$) compared to using integral control, but also because of the deadzone of the thrusters. Through visual inspection there was no noticeable movement from the ROV when the thrusters were given less than 7% thrust, this could have been dealt with, but due to lack of time only over-saturation was taken into consideration during the implementation.

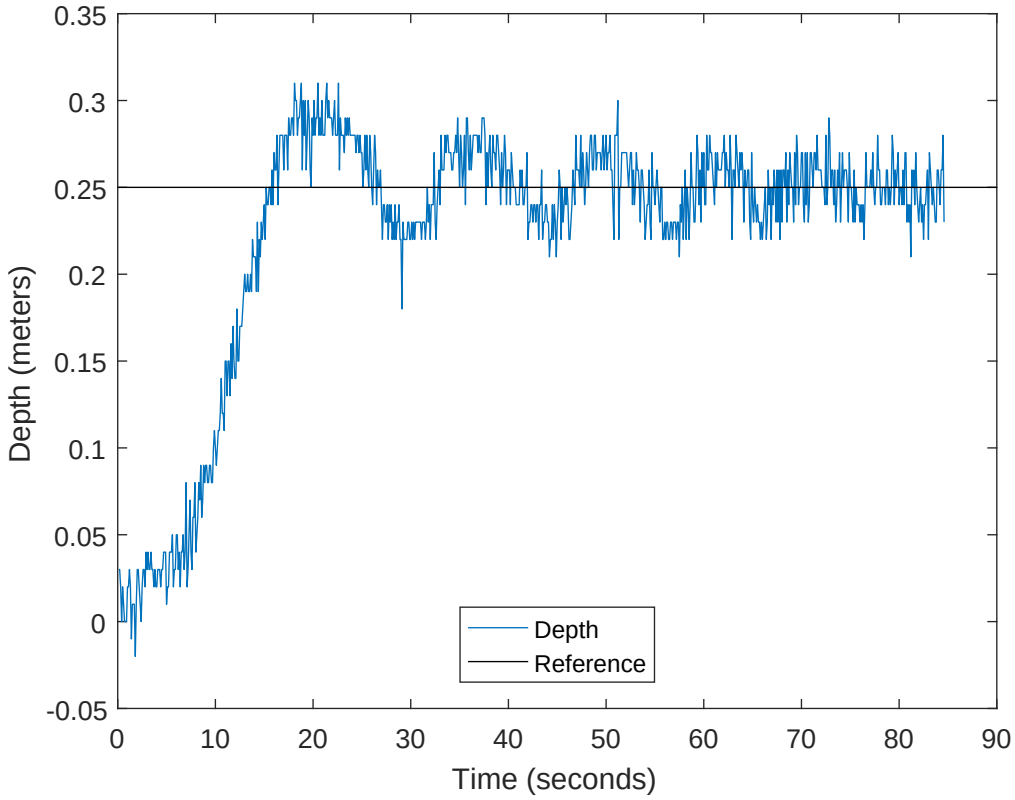


Figure 9.7: Depth test with ref 0.25m

Figure 9.7 shows the behaviour of an overdamped controller developed specifically for the testing. This was to check the performance of the system if the theoretically allowed overshoot was removed, which in practice greatly decreased the initial overshoot of the system to around 5cm compared to the 10cm in the previous tests. The ROV also fluctuates around steady-state ±3cm compared to previous ±5cm.

$$K = \begin{bmatrix} 18.2807 & -1.9274 \end{bmatrix} \tag{9.7a}$$

$$L = \begin{bmatrix} 4.5724 & 6.8266 \end{bmatrix}^T \tag{9.7b}$$

$$\bar{N} = 18.2807 \tag{9.7c}$$

Similar to figure 9.6 the results from figure 9.7 might not reach zero steady-state error due to the lack of integral control and the thrusters deadzone. The new control gains showed above are calculated based on a 0.01% overshoot and a settling time of 10 seconds, with the estimator poles being 6 times faster than poles calculated for the $\boldsymbol{K}$ matrix.

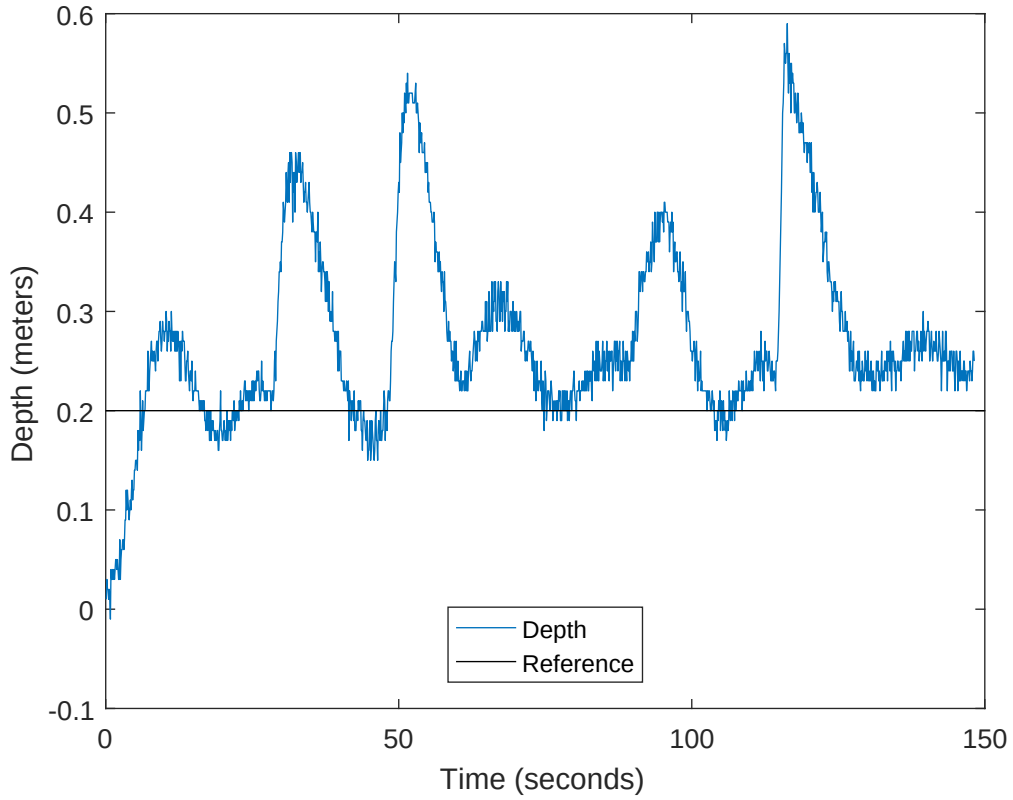


Figure 9.8: Disturbance test with a reference of 0.2m

Using the same overdamped controller as shown in figure 9.7, a final test was performed where the ROV was manually disturbed. As shown in figure 9.8 the ROV is able to recover from different amounts of disturbance, the depth was limited to 0.6m due to size limitations of the testing environment.

Chapter 10

Discussion

This discussion will compare the initial requirements specified in the problem description with the final outcome of the project. It will also address results and issues encountered throughout the entirety of the project. Finally some improvements will be suggested for future work related to the project.
During the initial stages of the project a lot of time was spent sourcing equipment needed to assemble the ROV kit and understanding the included Open-Source software. Fortunately, through the logic analysis it was possible to obtain serial commands, which aided the process of creating the custom implementations using C.

- Creation of a mathematical model describing the 6DoF
- Design of a state-space controller for the depth
- Implementation of a state-space controller for the depth

Looking back at the requirements set in 2.3 and as seen above, the project was split into three core requirements. The creation of a generalised mathematical model was achieved by taking into account differential equations describing the motions in the 6DoF. Unfortunately during the system identification, due to the noise and drift present on the accelerometer experiments, it was not possible to acquire data in order to determine the coefficients for the entire mathematical model. Only depth data was retrieved since these coefficients were determined based on measurements from the pressure sensor. The benefit of creating the generalised description of the mathematical model is that for future reference the model coefficients can be tweaked accordingly and be identified if the platforms allows so. But since the scope was delimited to implementing a depth controller, only identifying the coefficients for a subset of the mathematical model is acceptable.

Initially, due to the mathematical model describing all degrees of movement, the plan was to implement a MIMO control system and based on this information the choice to use state-space was made, since there were some clear benefits over using classical control theory when dealing with MIMO systems. In the end, a controller was only developed for the heave movement, which was a SISO system.

- Maximum overshoot: 10%
- Steady-state error: 1%
- Settling time: 10 seconds

As defined in section 2.3 and shown above, the time-domain specifications were initially 10% overshoot and a settling time of 10 seconds. The simulations of the development controllers were all within the specified requirements and are therefore deemed a success. The linearisation of the system, based on the comparison between the linear and non-linear model, showed good tracking in terms of performance and in some cases better performance. The initial gains developed for the estimator were only twice as fast as the system poles, this was due to a concern with noise sensitivity, since the depth positional data had some noisy elements to it when used during the system identification. The slow gains then resulted in slow estimation of states that ultimately caused poor performance during the implementation. The use of integral control also created some issues, this was due to poor tuning of the integral gain but also the lack of anti-windup. Since the implementation took place during the final period of the project, due to a lack time it was decided not to go forward with the integral control and instead use the reference scaling ($\bar{N}$). This resulted in a fluctuating steady-state, but for an initial implementation without integral control it works much better than expected, since the reference scaling is sensitive to general inaccuracies in the mathematical model. The creation of an overdamped controller was done in order as an to attempt limit the fluctuation around the reference point, but also to mimic a more realistic real life implementation, since overshooting a set point could cause potential harm to the vehicle. Therefore depending on the scenario a possibility to choose between two different controllers could be appropriate, where one overshoots less than the other and tracks a given reference point with more precision.

In order to create a better controller for the system, a more advanced controller can be developed. During the creation of the state-space representation it was chosen that the drag coefficient for the up and down motion to be equal and the thrust generated from the thruster in both directions to be equal. By having two separate controllers that are active depending on the travel direction it is possible to more accurately estimate the performance of the ROV. Throughout the project the accuracy of the neutral buoyancy changed significantly, where at times the ROV would be greatly negatively or positively buoyant. Since the ROV only weighs 2.7 kg a small change in weight significantly effected the overall buoyancy. After doing the implementation and testing it was proposed that a future implementation uses positively buoyant vehicle, since always having to actuate the thruster to keep it at given location would help in terms of maintaining depth at a given reference point. A downside to positive buoyancy is the added power consumption of continuously running the thruster to maintain the desired depth, this can be solved by instead implementing a ballast tank in order to control the depth, and would introduce a whole new set of dynamics instead of the thruster for heave movement. By instead having positive buoyancy as proposed it is then possible to model the controller in such a way that the deadzone of the thrusters are minimized, in order to avoid the current deadzone consisting of approximately $\pm 7\%$ thrust.

Chapter 11

Conclusion

As stated in the problem description, the project requirements were split into three sub-requirements. Through the mathematical modelling a general model describing the ROV was achieved and modified successfully based on the parameters identified during experimentation. With more time available and by improving the performance of the IMU, it will be possible to find the remaining parameters in order create a MIMO control system for the ROV. Based on the available sensor information the generalised mathematical model was modified to become a SISO system. A suitable controller was developed in form of an estimator, using control gains designed based on specific time-domain specifications. During the controller development, multiple controllers were theoretically developed in order to aid the implementation. The implementation was formed from the block diagram for the observer, where the difference equations were obtained for the state estimate ($\dot{\hat{\boldsymbol{x}}}$) and the equation for the control signal generation (u). Due to a lack of time during implementation, a controller using reference input scaling ($\bar{N}$) was implemented, compared to the initially planned integral control. The results from testing were a success, since it was within a certain margin in terms of tracking the given reference. Though, the implemented controller did not meet the theoretically specified time-domain specifications.

As discussed in the previous section, possible future work could be based on finding the remaining parameters for the MIMO system and to create a suitable implementation. This requires obtaining information from the IMU or a similar device, which will be used for system identification and for environmental readings during the controller implementation. The final goal with the MIMO system could be to three dimensionally position the MIMO whilst also implementing heading control for yaw.

References

[1] OpenROV. OpenROV. http://www.openrov.com, 2016, Accessed 4.11.2016.

[2] Stephen Robinson. Underlying physics and mechanisms for the absorption of sound in seawater. http://resource.npl.co.uk/acoustics/techguides/seaabsorption/physics.html, 2016, Accessed 6.11.2016.

[3] David Hambling. What drones did for the sky, robot subs are about to do for the sea. http://www.popularmechanics.com/technology/robots/news/a22903/rov-sea-drones/, September, 2016.

[4] Newport Corporation. Motion basics and standards. https://www.newport.com/n/motion-basics-and-standards, 2006, Accessed 3.11.2016.

[5] Edward V. Lewis. Principles of naval architecture second revision. http://opac.vimaru.edu.vn/edata/EBook/Principles%20of%20Naval%20architecture.pdf, 1989.

[6] OpenROV. OpenROV v2.8 kit. http://store.openrov.com/products/openrov-v2-8-kit, Accessed 5.10.2016.

[7] OpenROV. OpenROV Store. http://store.openrov.com/collections/2-8-replacement-parts, July 2014, Accessed 3.12.2016.

[8] OpenROV. Control board. http://store.openrov.com/collections/2-8-replacement-parts/products/openrov-controller-board-2-8, Accessed 1.12.2016.

[9] OpenROV. Beagleboneblack. http://store.openrov.com/collections/2-8-replacement-parts/products/beaglebone-black, Accessed 1.12.2016.

[10] OpenROV. Tendra homeplug adapter. http://store.openrov.com/collections/2-8-replacement-parts/products/tenda-homeplug-adapter-pair, Accessed 1.12.2016.

[11] Nikhilesh Jasuja, Pooja Sehgal, and Kate T. Cat5e vs. cat6. http://www.diffen.com/difference/Cat5e_vs_Cat6, 16 November 2016, Accessed 1.12.2016.

[12] Francis Idachaba, Dike U. Ike, and Orovwode Hope. Future trends in fiber optics communication. http://www.iaeng.org/publication/WCE2014/WCE2014_pp438-442.pdf, July 2014, Accessed 3.11.2016.

[13] Charles Cross. Openrov communication. https://forum.openrov.com/t/message-transfer-between-bbb-and-arduino/4239/6, April 2016, Accessed 3.12.2016.

[14] Bosch. BNO055 - intelligent 9-axis absolute orientation sensor. https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf, Accessed 16.12.2016.

[15] TE CONNECTIVITY. MS5837-30BA - Digital Pressure and Altimeter Sensor Module. http://www.te.com/usa-en/product-CAT-BLPS0017.html, Accessed 16.12.2016.

[16] Logic Analyzer. LHT00SU1. http://www.banggood.com/LHT00SU1-Virtual-Oscilloscope-Logic-Analyzer-I2C-SPI-CAN-Uart-p-988565.html, April 2016, Accessed 3.12.2016.

[17] W. Wang and C.M Clark. Modeling and Simulation of the VideoRay Pro III Underwater Vehicle. *IEEE OCEANS 2006*, 2006.

[18] DC motor electrical schematic and free body diagram. http://ctms.engin.umich.edu/CTMS/Content/MotorPosition/System/Modeling/figures/motor.png, Accessed 1.11.2016.

[19] Bavaria-Direct. DELTA vs. WYE (star) termination. http://www.bavaria-direct.co.za/info/, 2013, Accessed 29.11.2016.

[20] Salih Baris Ozturk. Modelling, simulation and analysis of low-cost direct torque control of PMSM using hall-effect sensors. http://oaktrust.library.tamu.edu/bitstream/handle/1969.1/4905/etd-tamu-2005C-ELEN-Ozturk.pdf, 2005.

[21] Steven Keeping. Controlling sensorless, BLDC motors via BackEMF. http://www.digikey.dk/en/articles/techzone/2013/jun/controlling-sensorless-bldc-motors-via-back-emf, 2013, Accessed 16.12.2016.

[22] Ganesh C. Investigations on parameter Estimation and controller design For BLDC drive fed position Control system. http://shodhganga.inflibnet.ac.in/handle/10603/38954, 2014.

[23] M.W.C Oosterveld and P. Van Oossanen. Further computer-analysed data of the wageningen b-screw series. *International Shipbuilding Progress*, 22(251), 7 1975.

[24] Thor I. Fossen. *Marine Control Systems*, chapter 1.12.1. 1 edition, 2002.

[25] Pasco. Wireless newton meter. https://www.pasco.com/prodCatalog/PS/PS-3202_wireless-force-acceleration-sensor/index.cfm, April 2016, Accessed 3.12.2016.

[26] Gene F Franklin, J David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, chapter 7.1. Pearson, 7 edition, 2015.

[27] Introduction: State-space methods for controller design. http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlStateSpace, 2014.

[28] Gene F Franklin, J David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, chapter 7.5. Pearson, 7 edition, 2015.

[29] Gene F Franklin, J David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, chapter 7.7. Pearson, 7 edition, 2015.

[30] DC Motor Position: State-Space Methods for Controller Design. http://ctms.engin.umich.edu/CTMS/index.php?example=MotorPosition&section=ControlStateSpace, 2014.

Chapter 12

Appendix

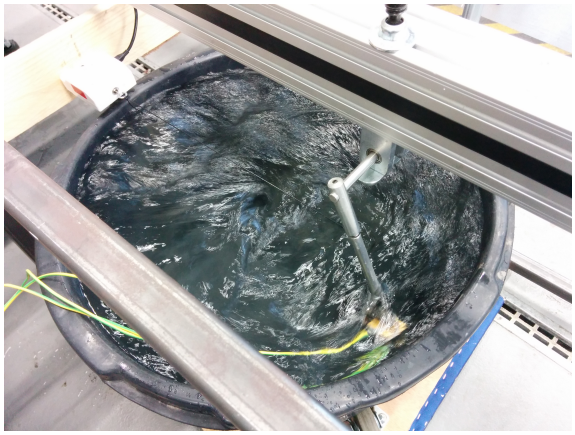
12.1 Pre-Dive Check List

Battery tubes
☐ Insert batteries into the battery tubes. Negative end pointing forwards
☐ Place battery adapters in the tubes.
☐ Check if end cap o-rings are intact.
☐ Lubricate o-rings with (name) before closing end caps.
☐ Check if they are sealing correctly.
Main tube
☐ Check if the o-rings are intact.
☐ Clean o-rings and lubricate them before sealing off the ends.
☐ Check that the sealings are ok. (1 mm surface contact)
☐ Check if syringe plunger is intact
☐ Lubricate it before inserting it into top main tube end cap.
☐ Attach main tube to frame and tighten the strap.
Motors
☐ Ensure motor bells and propellers are clear of wires.
☐ Check if the wires are intact.
☐ Lubricate all the motors with silicone and rotate them.
☐ Let it dry for a minute.

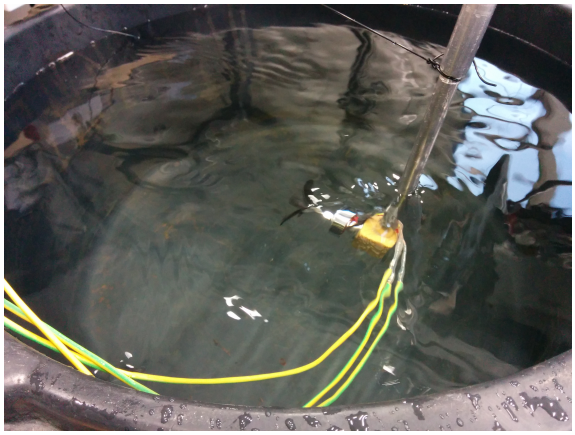
12.2 Post-Dive Check List

Motors
☐ Rinse the motors in fresh water.
☐ Dry motors with compressed air.
☐ Spray silicone on motors and bearings.
☐ Rotate motors.
Battery tubes
☐ Open battery tubes and remove the batteries.
Main tubes
☐ Open main tube in both ends and clean the tube.
In general
☐ Allow the ROV to dry out completely before packing it up.

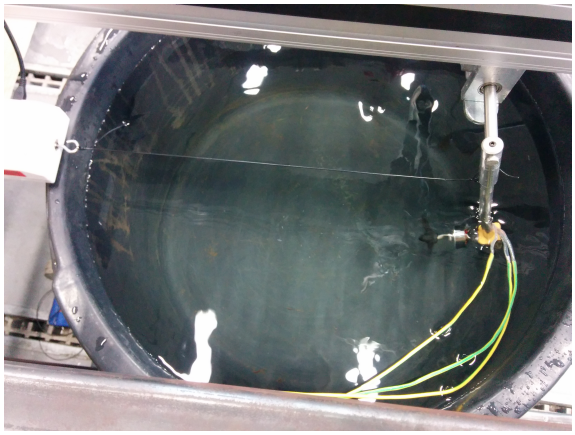
12.3 Experimentation Pictures



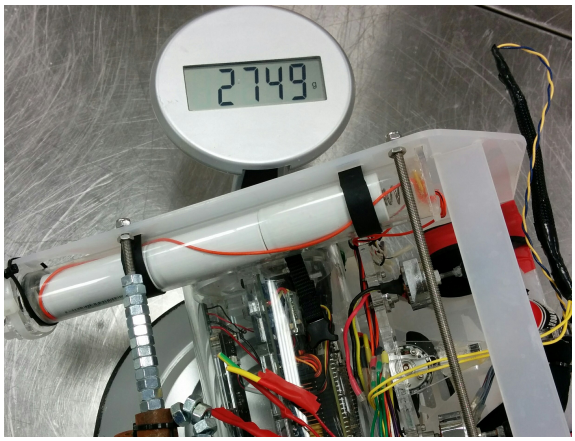
(a) Whirlpool created from the thruster



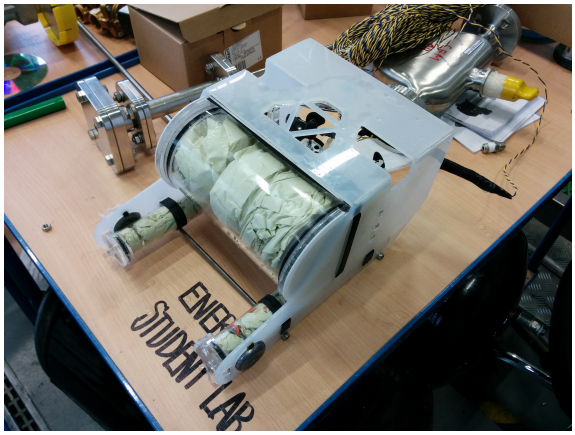
(b) Thruster test



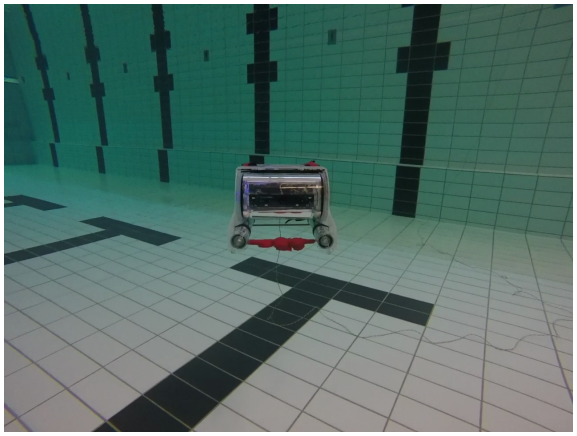
(c) Thruster test



(d) Mass of ROV



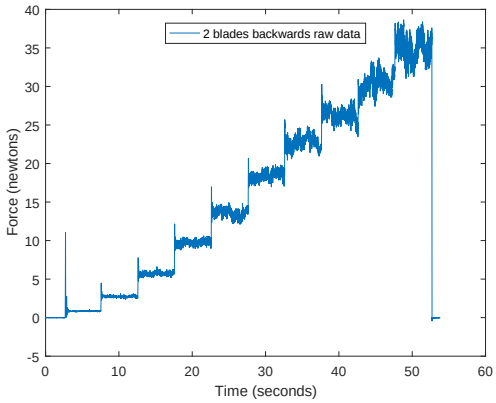
(e) Leakage test preparation



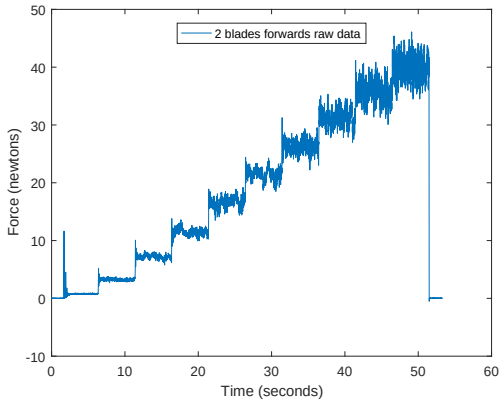
(f) Underwater in action

Figure 12.1: Experimentation pictures

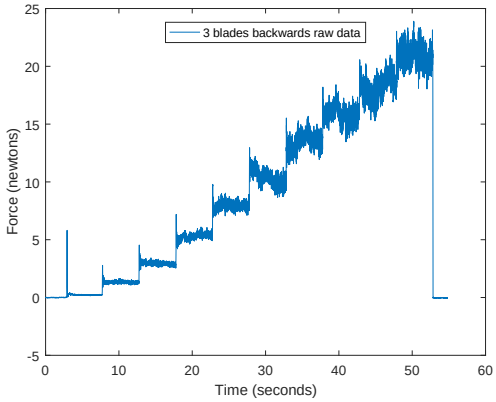
12.4 Raw Propeller Test



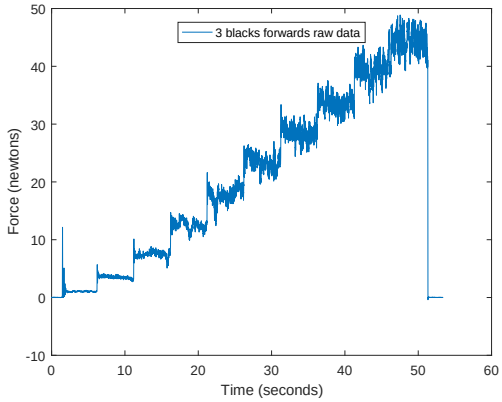
(a) Raw 2 propellers backwards



(b) Raw 2 propellers Forwards



(c) Raw 3 propellers backwards



(d) Raw 3 propellers forwards

Figure 12.2: Raw propellers test

12.5 Code

12.5.1 Data Logging C

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>

#include <signal.h>

//Baudrate for the serial connection
#define BAUDRATE B115200
//Serialport
#define MODEMDEVICE "/dev/ttyO1"
#define _POSIX_SOURCE 1

#define FALSE 0
#define TRUE 1


int stop;

void inthand(int signum){
  stop = 1;
}

int main(int argc, char *argv[])
{
  int fd, c, res;
  //Structs for the  settings
  struct termios oldtio, newtio;
  //Buffer for the data read over serial
  char buf[10];
  //Creates the signal for the input interrupt
  signal(SIGINT, inthand);
  //Creates the file pointer
  FILE *fp;
  //Opens the file based on argument
  fp = fopen(argv[1],"w");
  //Opens the serialport and checks for errors
  fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
  if (fd < 0) { perror(MODEMDEVICE); exit(-1); }
  //Clears the settings structure
  bzero(&newtio, sizeof(newtio));
  //BAUDRATE: Set Baudrate
  //CRTSCTS : Output hardware flow control
  //CS8     : 8n1 (8bit,no parity,1 stopbit)
  //CLOCAL  : local connection, no modem contol
  //CREAD   : enable receiving characters
```

```
51    newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
      //IGNPAR  : ignore bytes with parity errors
53    newtio.c_iflag = IGNPAR;
      //Raw output
55    newtio.c_oflag = 0;
      newtio.c_lflag = ICANON;
57    // now clean the modem line and activate the settings for the port
      tcflush(fd, TCIFLUSH);
59    tcsetattr(fd,TCSANOW,&newtio);

61    //Runs continously till it receives a SIGINT
      while(!stop) {
63      //Reads from serial until a line terminating character
        res = read(fd, buf, 10);
65      // set end of string, so it can printf
        buf[res] = 0;
67          //Prints the current values to the terminal
        printf("%s", buf, res);
69      //Save the values read over serial in a txt file
        fprintf(fp,"%s", buf, res);
71      }
      //Closes the file pointer
73      fclose(fp);

75      printf("Exited safely\n");

77      return 0;
}
```

Code Listing 12.1: Data logging in C

12.5.2 Thruster Test C

```
#include <stdlib.h>
2 #include <stdio.h>
#include <time.h>
4

int main(){
6

  sleep(2);
8 int i;
  char cmd[50];
10

  for (i = 10 ; i < 101; i = i + 10){
12   //Creates the string containing the desired thrust value
     sprintf(cmd,"echo \"thro(%d);\" -n > /dev/ttyO1", i);
14   //Send the command
     system(cmd);
16   printf("force perc: %d \n", i);
     sleep(5);
18 }

20   system("echo \"thro(0);\" -n > /dev/ttyO1");
     return 0;
```

```
}
```

Code Listing 12.2: Thruster test in C

12.6 Controller Implementation in C

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

//Interval in ms for the timer
#define INTERVAL 100
//Baudrate for the serial connection
#define BAUDRATE B115200
//Serialport
#define MODEMDEVICE "/dev/ttyO1"
#define _POSIX_SOURCE 1

#define FALSE 0
#define TRUE 1
//Struct for setting itimer
struct itimerval it_val;
int fd, c, res;
//struct for setting serial
struct termios oldtio, newtio;

//Buffers for incoming data and for the commands
char buf[30], cmd[20];

//Variables relevant for the control implementation
float A[2][2] = {{0,1.0000},
                 {0,-1.9538}};

float B[2][1] = {{0},
     {0.0170}};

float C[1][2] = {1,0};

//Control gains

//10  second settling time 10% overshoot
//float K[1][2] = {35.6977,  -60.7509};
//float L[2][1] = {{3.5724},
//                 {14.8672}};
//float Nbar = 35.6977;
```

```
48 //10second settling time 0.01% overshoot
   float K[1][2] = {18.2807,  -1.9274};
50 float L[2][1] = {{4.5724},
                   {6.8266}};
52 float Nbar = 18.2807;

54 //Initial values
   float x[2][1] = {0}, Ax[2][1] = {0}, Bu[2][1] = {0}, est[2][1] = {0},
      Cx = 0, u = 0;
56 float y = 0, y_cx, integral = 0;

58 //Reference point r, integral gain ki, sampling period h
   float r = 0.3, ki = 5,h = 0.1;
60
   int i,j,k, input = 0;
62
   //Function prototypes
64 void initialization(void);
   void quit_program(int sig);
66 void update_estimate(void);

68 int main(int argc, char *argv[])
   {
70   //Run the initialization for the serial
       initialization();
72     //Sets up the file pointer
       FILE *fp;
74     //Opens the file specified in arguments
       fp = fopen(argv[1],"w");
76
   //Zeroes the depth sensor
78     printf("Zero'd the depth sensor\n");
       sprintf(cmd,"echo \"dzer();\" -n > /dev/ttyO1");
80     system(cmd);

82     float depth;

84 //Runs continuously until SIGINT
   while(input <= 0) {
86     //Reads from serial until a line terminating character
       res = read(fd, buf, 30);
88     //Set the end of string so it can be printed
       buf[res] = 0;
90     fprintf(fp,"%s", buf, res);
       //Parses the incoming data
92     sscanf(buf, "deap;%f;", &depth);
       y = depth;
94     //Checks if the calculated control variable is oversaturated
       if(u > 100.0) u = 100.0;
96     if(u < -100.0) u = -100.0;
       //Sends the calculated u to the thrusters
98     sprintf(cmd,"echo \"lift(%d);\" -n > /dev/ttyO1", (int)u);
       system(cmd);
100    //Status print to the terminal for current depth, integral and
      thruster value
       printf("int: %f - u: %f - depth: %f\n",integral,u, depth);
```

```
102
  }
104
    //Once the SIGINT has ended the above loop the interval timer is
    ignored
106   signal(SIGALRM, SIG_IGN);
    //Small pause to ensure proper file closure and thruster shutdown
108   usleep(1000000);
    fclose(fp);
110   //Turns of the thruster
    sprintf(cmd,"echo \"lift(0);\" -n > /dev/ttyO1");
112 system(cmd);
  printf("Exited safely\n");
114
    return 0;
116 }

118 //Function to stop the main control loop
  void quit_program(int sig){
120   input = 1;
  }
122
  //Function for updating the control variable
124 void update_estimate(void){
  //-------------------------------------------------------------------
126 //Calculates h*A*x and stores it in  Ax
  for(i=0; i<2; i++)
128 {
    for(j=0; j<1; j++)
130   {
      Ax[i][j] = 0;
132   for(k=0; k<2; k++)
        Ax[i][j] = h * A[i][k] * x[k][j] + Ax[i][j];
134   }
  }
136 //-------------------------------------------------------------------
  //Calculates h*B*u and stores it in  Bu
138 for(i=0; i<2; i++)
  {
140   for(j=0; j<1; j++)
    {
142     Bu[i][j] = 0;
      for(k=0; k<1; k++)
144     Bu[i][j] = h * B[i][k] * u + Bu[i][j];
    }
146 }
  //-------------------------------------------------------------------
148 //Calculates L(y - C*x) and stores it in  est
  for(i=0; i<1; i++)
150 {
    for(j=0; j<1; j++)
152   {
      Cx = 0;
154   for(k=0; k<2; k++)
        Cx = C[i][k]*x[k][j] + Cx;
156   }
```

```
    }
158
    y_cx = y - Cx;
160
    for(i=0; i<2; i++)
162 {
      for(j=0; j<1; j++)
164   {
        est[i][j] = 0;
166     for(k=0; k<1; k++)
          est[i][j]=h*L[i][k]*y_cx + est[i][j];
168   }
    }
170 //-------------------------------------------------------------------------
    //Calculates x  based on previous x, Ax, Bu and est (L(y-Cx))
172 for(i=0; i<2; i++)
    {
174   for(j=0; j<1; j++)
        x[i][j] = x[i][j] + Ax[i][j] + Bu[i][j] + est[i][j];
176 }
    //-------------------------------------------------------------------------
178 //Calculates control variable u, first Kx then the remainder of the
     statement

180 //Integral for the integral control
    integral = integral + (y - r)*h;
182
    //Calculate Kx
184 for(i=0; i<1; i++)
    {
186   for(j=0; j<1; j++)
      {
188     u = 0;
        for(k=0; k<2; k++)
190       u = K[i][k]*x[k][j] + u;
      }
192 }

194 //control variable calculation for the integral control
    //u = -1*u - ki*integral;
196
    //Control varialbe calculation using Nbar
198 u = -1*u + Nbar*r;

200 }

202 //Function for initializing the signals and serial
    void initialization(void){
204
        //Opens the serialport and checks for errors
206 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
    if (fd < 0) { perror(MODEMDEVICE); exit(-1); }
208 //Clears the settings structure
    bzero(&newtio, sizeof(newtio));
210 //BAUDRATE: Set Baudrate
    //CRTSCTS : Output hardware flow control
```

```
212   //CS8     : 8n1 (8bit,no parity,1 stopbit)
      //CLOCAL  : local connection, no modem contol
214   //CREAD   : enable receiving characters
      newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
216   //IGNPAR  : ignore bytes with parity errors
      newtio.c_iflag = IGNPAR;
218   //Raw output
      newtio.c_oflag = 0;
220   newtio.c_lflag = ICANON;
      // now clean the modem line and activate the settings for the port
222   tcflush(fd, TCIFLUSH);
      tcsetattr(fd,TCSANOW,&newtio);
224
      //Sets up the SIGARLM to call update_estimate when triggered
226   if (signal(SIGALRM, (void (*)(int)) update_estimate) == SIG_ERR) {
        perror("Unable to catch SIGALRM");
228         exit(0);
      }
230   //Calculates the interval in seconds, based on input ms
      it_val.it_value.tv_sec =     INTERVAL/1000;
232   //Calculate the interval in useconds, based on input ms
      it_val.it_value.tv_usec =    (INTERVAL*1000) % 1000000;
234   it_val.it_interval = it_val.it_value;
      //Sets the interval timer
236   if (setitimer(ITIMER_REAL, &it_val, NULL) == -1) {
        perror("error calling setitimer()");
238     exit(0);
      }
240
      //Sets up the SIGINT to end the control loop in userinterrupt
242   signal(SIGINT, quit_program);
}
```

Code Listing 12.3: Controller Implementation in C