Norwegian University of
Science and Technology

Surface Assisted Autopilot for Remotely Operated Vehicle

Eirik Storesund

Master of Science in Cybernetics and Robotics
Submission date: July 2018
Supervisor: Thor Inge Fossen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

Acknowledgements

This thesis concludes the authors' 2-year Master's degree programme in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU).

I would like to express my appreciation to Torgeir Trøite and Oliver Skisland for giving me the opportunity to write my master thesis at Water Linked AS.

I also wish to thank my advisor, Thor Inge Fossen, Professor of Guidance, Navigation and Control at the Department of Engineering Cybernetics NTNU for providing me with key insights when needed.

To Eirik Storås Thorbjørnsen and Audun Aarnes at Water Linked AS, you have been greatly helpful when dealing with hardware and software, and for dragging me out of the office to go climbing.

Additionally, I would like to thank Trude Støren, CEO of Embida AS for proofreading the report.

Contributions

Water Linked provided all necessary hardware for the Underwater Modem and the BlueRov2. Inventas AS provided the Intuitive Input Device used for controlling the ROV. The main contributions by the author is listed below.

- Software Development Life Cycle literature survey.
- Customizing the hardware on the BlueRov2 to enable wireless control.
- Topside computer software (Python scripts)
  - Interfacing with Xbox controller and Intuitive Input Device.

  - Encoding of small sized data payloads to provide control input for the ROV.
  - Interfacing with Topside Master unit.
- Subsea software (Python scripts and ROS)
  - Interfacing with Subsea Master unit.
  - Setting up software on the BlueRov2 Onboard Computer (Raspberry Pi). Includes Ubuntu Mate, ROS, MAVROS and Python.
  - Decoding small sized data payload.
  - Implementing the ROS package MAVROS to enable safe and robust control of the ROV by interfacing with the onboard Pixhawk autopilot.
  - Configure the Pixhawk Flight Controller Unit running ArduSub firmware.
- Test and validate wireless control of the BlueRov2 by using testing scheme found in the literature survey.

Trondheim, July 2018 Eirik Storesund

Abstract

The following thesis was done in collaboration with Water Linked AS and the Norwegian University of Science and Technology. The task at hand was to demonstrate wireless control of an ROV implemented using Water Linked's Underwater Modem as a part of the SWARMs (Smart and Networking Underwater Robots in Cooperation Meshes) research project. The final demonstration took place at Trondheim Biological Station 26. June 2018.

The Underwater Modem makes wireless underwater communication possible. The Downlink provides communication from a topside computer to a subsea ROV, while the Uplink provides a video stream subsea to topside. This thesis focus solely on using the Downlink to facilitate a safe and robust system for wireless control of a BlueRov2 from BlueRobotics.

The BlueRov2 is an affordable, high-performance ROV which is highly customizable. The ROV is normally connected with a tether to a topside computer, but by using the Underwater Modem system, the tether is omitted. The Downlink requires an additional waterproof payload on the ROV, a Subsea Master unit, which was mounted on the ROV.

To control the ROV, software for both a Raspberry Pi on board the ROV (OBC) and for a topside computer were be made. Two different devices were integrated to provide pilot input for the ROV; an Xbox controller, and the Intuitive Input Device (IID) provided by Inventas AS. To be able to control the ROV using a limited amount of data, the transmitted data were parsed and encoded. The control commands provided control of four degrees of freedom, as well as giving the opportunity to turn off and on lights, change the control mode, and adjust pilot gain.

On the OBC, the control commands are received and decoded. To control the ROV, a variety of software is used. ArduSub runs on the Pixhawk Flight Controller Unit (FCU), and the ROS framework is used to interface with the FCU through the MAVROS package. How the software on both the topside computer and the OBC is implemented is explained in detail in the thesis.

Prior to the demonstration of the system in the SWARMs project, extensive testing was performed. To find tools and methods for testing, Software Development Life Cycle methods were explored. This provided testing tools that enabled a successful demonstration.

Sammendrag

Denne masteroppgaven ble gjort i samarbeid med Water Linked AS og Norges tekniske- og naturvitenskapelige universitet (NTNU) våren 2018. Hovedoppgaven besto av å demonstrere trådløs fjernstyring av en ROV under vann ved bruk av Water Linked produktet Underwater Modem. Dette ble gjort i forbindelse med det industrielle forskningsprojektet SWARMs (Smart and Networking Underwater Robots in Cooperation Meshes). Demonstrasjonen ble utført ved Trondheim biologiske stasjon 26. juni 2018.

Underwater Modem muliggjør kommunikasjon under vann. Downlink gjør at man kan kommunisere mellom en topside PC (over vann) og en ROV under vann, mens Uplink muliggjør det å strømme bilder fra en ROV til en topside PC. Denne masteroppgaven fokuserer på å bruke Downlink for å demonstrere sikker og robust trådløs styring av en BlueRov2 ROV.

BlueRov2 er en rimelig, høyytelses ROV som er svært tilpassbar. ROVen er normalt koblet til en topside PC gjennom en tether, men ved bruk av Underwater Modem systemet blir tetheren utelatt. Flere oppgradering på ROVen måtte bli gjort for å legge til rette for trådløs styring.

For å styre ROVen trådløst, ble software laget for både en Raspberry Pi ombord av ROVen og på topside PCen. To enheter ble integrert for å kunne manøvrere ROVen, en Xbox-kontroller og Intuitive Input Device (IID) fra Inventas AS. ROVen blir styrt med en begrenset datamengde, 4 byte/s. For å gjøre opp for dette, ble de sendte datapakkene enkodet og dekodet. Kontrollkommandoene gir styring i fire frihetsgrader, samt gir mulighet til å slå av og på lys, endre kontrollmodus og justere forsterkning.

På Raspberry Pien mottas kontrollkommandoene som blir dekodet og tatt i bruk. En rekke programvarer ble brukt for å kunne kontrollere ROVen. ArduSub kjører på Pixhawk Flight Controller Unit (FCU), og ROS rammeverket brukes som grensesnittet med FCUen, gjennom MAVROS-pakken. Hvordan programvaren på både topside PCen og Raspberry Pien ble implementert, forklares i detalj i avhandlingen.

Før demonstrasjonen i SWARMs prosjektet ble omfattende testing utført. For å

finne verktøy og metoder for testing, ble Software Development Life Cycle (SDLC) metoder utforsket. Dette ga testverktøy som sørget for at demonstrasjonen ble en suksess.

Table of Contents

Preface i

Abstract iii

Sammendrag v

Table of Contents ix

List of Figures xii

List of Tables xiii

List of Code Excerpts xv

Abbreviations xvi

1 Introduction 1

1.1 Background 1
1.2 Motivation 2
1.3 Problem Description 2
1.4 Thesis outline 3

2 Water Linked Underwater Modem 5

2.1 Overview 5
2.2 Downlink 6
2.3 Uplink 6

3 Hardware and Software **11**
3.1 Hardware . . . 11
3.1.1 BlueRov2 . . . 11
3.1.2 Input Devices . . . 14
3.2 Software . . . 15
3.2.1 Gazebo . . . 15
3.2.2 Robot Operating System . . . 16
3.2.3 ArduSub . . . 17

4 Theory **19**
4.1 Software Development Life Cycle . . . 20
4.2 Traditional SDLC models . . . 20
4.2.1 Waterfall model . . . 20
4.2.2 V-model . . . 21
4.3 Prototyping Model . . . 22
4.4 Incremental / Iterative model . . . 24
4.5 Agile Development . . . 25
4.5.1 Agile Manifesto . . . 25
4.5.2 Extreme Programming . . . 27
4.5.3 Feature Drive Development . . . 28
4.6 Testing in Software Development Life Cycle . . . 29

5 Implementation **31**
5.1 Overview . . . 32
5.2 Topside software . . . 33
5.2.1 Input devices . . . 34
5.2.2 Encoding . . . 35
5.2.3 Transmitting UDP packets . . . 38
5.3 Subsea software . . . 39
5.3.1 Receive UDP packets asynchronously . . . 39
5.3.2 Decoding . . . 39
5.3.3 MAVROS . . . 40
5.4 Linux Services . . . 42

6 Testing **49**
6.1 Unit Testing . . . 49
6.2 Integration Testing . . . 51

6.3 System Testing 51
6.4 Acceptance Testing 53

7 Results and the way forward **55**
7.1 Results 55
7.2 Conclusion 56
7.3 Further work 57

References **59**

Appendices **61**
A.1 List of software resources 1
B.2 Subsea data log excerpt 3

List of Figures

1.1 Water Linked AS logo 1
1.2 SWARMs logo 2

2.1 Underwater Modem overview 6
2.2 Master-D1 digital computing-board 7
2.3 Setup of the Underwater Modem Downlink 8
2.4 Setup of the Underwater Modem Uplink 9

3.1 BlueRov2 with Heavy Configuration. 12
3.2 Custom BlueRov2 13
3.3 The Pixhawk4 Flight Controller Unit 13
3.4 The Raspberry Pi 3 Onboard Computer 14
3.5 Xbox controller, default input device 14
3.6 Intuitive Input Device (IID) from Inventas AS 15
3.7 Simulating BlueRov2 in Gazebo 16

4.1 Waterfall development life cycle 21
4.2 V-model development life cycle 23
4.3 The Prototyping model 23
4.4 The incremental / iterative SDLC model 25
4.5 The agile development process 26
4.6 Extreme Programming (XP) vs Waterfall and Iterative Model 27
4.7 The Feature Driven Development method 28
4.8 Testing scheme 30

5.1 Software overview 32
5.2 Topside program flow 33
5.3 Xbox controller input device 35
5.4 SWARMS Intuitive Input Device 35
5.5 Data frame sent by the Downlink 36
5.6 Data byte for forward and lateral axis 36
5.7 Data byte for throttle, yaw and buttons 37
5.8 UDP packet to be sent to Topside Master over Ethernet 38
5.9 Sending UDP packets to Topside Master 38
5.10 Subsea software overview 40
5.11 ROS node overview 47

6.1 Testing in indoor tank 52
6.2 Testing outdoors 52

7.1 Delay between each received data frame on the Onboard Computer . 56

List of Tables

2.1 Water Linked Underwater Modem specifications 7

3.1 Hardware BlueRov2 12
3.2 Control modes of the ROV provided by ArduSub 17

4.1 Waterfall model strengths and weaknesses 22
4.2 V-model strengths and weaknesses 22
4.3 Prototype model strengths and weaknesses 24
4.4 Incremental / Iterative model strengths and weaknesses 24
4.5 Extreme Programming strengths and weaknesses 28
4.6 Feature Driven Development strengths and weaknesses 29

6.1 System Testing requirements 51
6.2 Acceptance Testing requirements 53

7.1 SWARMs demonstration results 56

List of code excerpts

5.1 Encode forward and lateral joystick command 37
5.2 Encode throttle, yaw and button command 43
5.3 Asynchronous UDP Client 44
5.4 Message handler 44
5.5 Decode control commands 45
5.6 Scale integer to floating number 45
5.7 Publishing joystick messages in ROS 46
5.8 Linux Service 46
6.1 Unit-test example 50

Abbreviations

AUV	Autonomous Underwater Vehicle
DOF	Degree Of Freedom
FDD	Feature Driven Development
FIFO	First In First Out
GPS	Global Positioning System
IID	Intuitive Input Device
OBC	Onboard Computer
PWM	Pulse Width Modulation
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
SDLC	Software Development Life Cycle
SWARMs	Smart and Networking Underwater Robots in Cooperation Meshes
UDP	User Datagram Protocol
USV	Unmanned Surface Vehicle
XP	Extreme Programming

Chapter 1

Introduction

1.1 Background

This master thesis is done in collaboration with Water Linked AS, the inventors of the “Underwater GPS” for acoustic positioning. Water Linked has joined the industrial based research project SWARMs, where the aim is to make AUVs, ROVs and USVs further accessible and useful. For this research, Water Linked is adopting an ROV from BlueRobotics called BlueRov2. The BlueRov2 is normally connected to a topside computer using a tether, where all control signals and communications go through.

Water Linked uses their acoustic technology to make underwater communication possible, a product they called the “Water Linked Underwater Modem”. This way, the ROV can be controlled wirelessly in real-time from a topside computer.

Figure 1.1: Water Linked AS logo. Image courtesy: [19]

1.2 Motivation

The SWARMs Project is a 17.3 Million Euro industrial research project with over 30 partners representing 10 European countries. The primary goal for this project is to expand the use of underwater and surface vehicles (AUVs, ROVs, USVs) to facilitate the conception, planning and execution of maritime and offshore operations and missions [17]. Its use-cases include corrosion prevention, seabed mapping and pollution monitoring.

One of the main objectives is to "Apply communication concepts ensuring smooth functioning while also exploring new, innovative technologies" [17]. Water Linked provides the Underwater Modem for this purpose. The integration and validation of the project was set to 11-24 June 2018 in Trondheim, Norway. The final demonstration was set to 26. June.

Figure 1.2: SWARMs (Smart and Networking Underwater Robots in Cooperation Meshes) logo. Image courtesy: [17]

1.3 Problem Description

The main deliverable of the work described in this thesis is the SWARMs demonstration 26. June 2018 in Trondheim, Norway. Here, the Underwater Modem is used to demonstrate wireless control of the ROV using the software produced in the duration of the project.

The work consists of writing software for a topside computer and a Raspberry Pi on board the ROV. Additionally, a testing scheme along with different tools is used to ensure that the system meet its requirements. The tasks includes

- Software Development Life Cycle (SDLC) literature survey.
- Explore and use testing tools used in successful SDLC models.

- Enable wireless underwater control of a BlueRov2 ROV using Water Linked Underwater Modem. This includes
  - Set up necessary hardware and software
  - Provide pilot input from a device, Xbox Controller or Intuitive Input Device delivered by Inventas AS
  - Develop and implement encoding / decoding of small-sized data payloads to control the ROV.
  - Interface with the Pixhawk autopilot using Robot Operating System (ROS) framework for controlling the ROV
- Apply testing tools to the system, validating robust and safe control of the ROV

1.4 Thesis outline

The thesis is organized in the following manner:

- Chapter 2 gives an overview of the Underwater Modem system, including the Downlink and Uplink.
- Chapter 3 introduces important hardware and software used in this project.
- Chapter 4 gives an introduction to SDLC as well as several SDLC models. Additionally, different software testing tools is explored.
- Chapter 5 gives a detailed explanation of the software developed during this project.
- Chapter 6 describes how the different software testing tools were used.
- Chapter 7 presents the results, concludes the thesis and presents a few ideas for future work.

Chapter 2

Water Linked Underwater Modem

2.1 Overview

Water Linked Underwater Modem enables wireless underwater communication. Although underwater communication systems exists today, Water Linked's system is small-sized, low-priced and works well in highly reflective environments.

Figure 2.1 shows an example of communication between a topside computer and a subsea ROV. The Downlink (topside $\rightarrow$ subsea) provides wireless control of the ROV by sending control commands from a topside computer using an input device. The Uplink (subsea $\rightarrow$ topside) enables streaming of low-quality video/images from the ROV to the topside computer. Table 2.1 summarizes the system specifications.

Note that the Topside Master and Subsea Master units are a part of the Underwater Modem system, and use the same type of hardware (see figure 2.2). The topside computer is a PC that interfaces with input devices and the Topside Master. The Onboard Computer (OBC) is a Raspberry Pi inside the ROV that interfaces with the Subsea Master and controls the ROV.

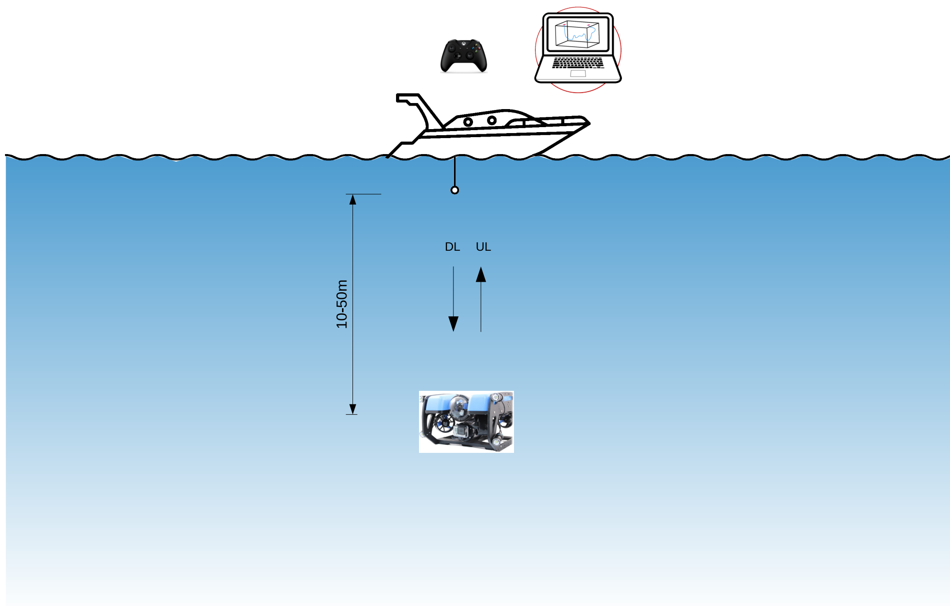

Figure 2.1: Underwater Modem overview. The Downlink (DL) enables communication from a topside computer to the ROV. The Uplink (UL) provides a video stream from the ROV to the topside computer.

2.2 Downlink

As stated earlier, the Downlink enables communication between a topside computer and the Raspberry Pi Onboard Computer (OBC) on the ROV. The hardware needed is shown in figure 2.3. An input device (i.e Xbox controller) provides input for the topside PC that transmits encoded data as UDP packets to the Topside Master unit.

A Locator A1 hydro-acoustic device is used for transmitting the data acoustically, while a Receiver D1 device connected to the ROV receives the acoustic signals. On the Subsea Master, the signals are processed and the data is sent as UDP packets to the Raspberry Pi OBC.

2.3 Uplink

The Uplink provides a low quality video stream from the ROV to the topside computer. Figure 2.4 show the required hardware. The OBC compresses the video

Table 2.1: Water Linked Underwater Modem specifications

Topside Master unit	Master D1 WL-21008
Subsea Master unit	Master D1 WL-21008
Locator	Locator A1 WL-21009
Receiver	Receiver D1 WL-21005
Downlink	>= 32 bits per second
	< 1000 ms latency from pilot input to BlueRov thrusters
Uplink	> 10 kbit per second peak
	< 1000 ms latency from BlueRov camera to topside computer

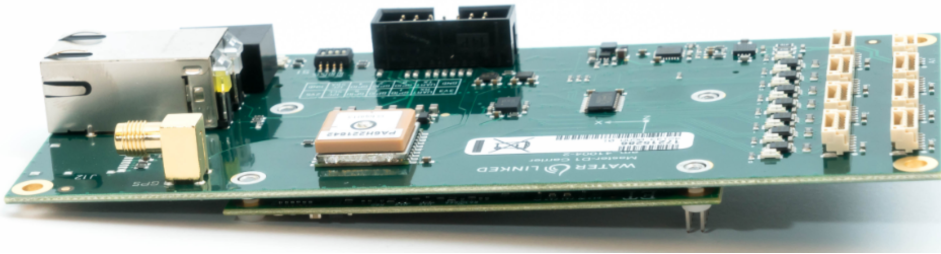

Figure 2.2: Master-D1 digital computing-board. Used in both the Topside and Subsea Master units. Image courtesy: [19]

taken from the camera on board the ROV and sends the decoded data to the Subsea Master as UDP packets. As with the Downlink, transducers are being used to send data acoustically to the Topside Master. From here, the data can be decoded and presented as a video stream on a monitor. Note that the work presented in this thesis is only focused on using the Downlink.

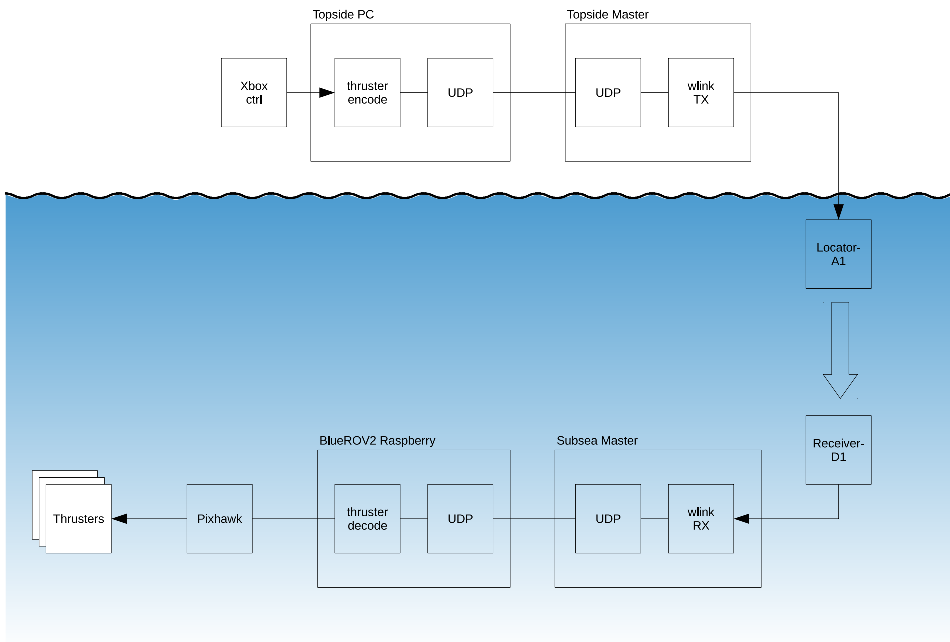

Figure 2.3: Setup of the Underwater Modem Downlink enabling wireless underwater communication

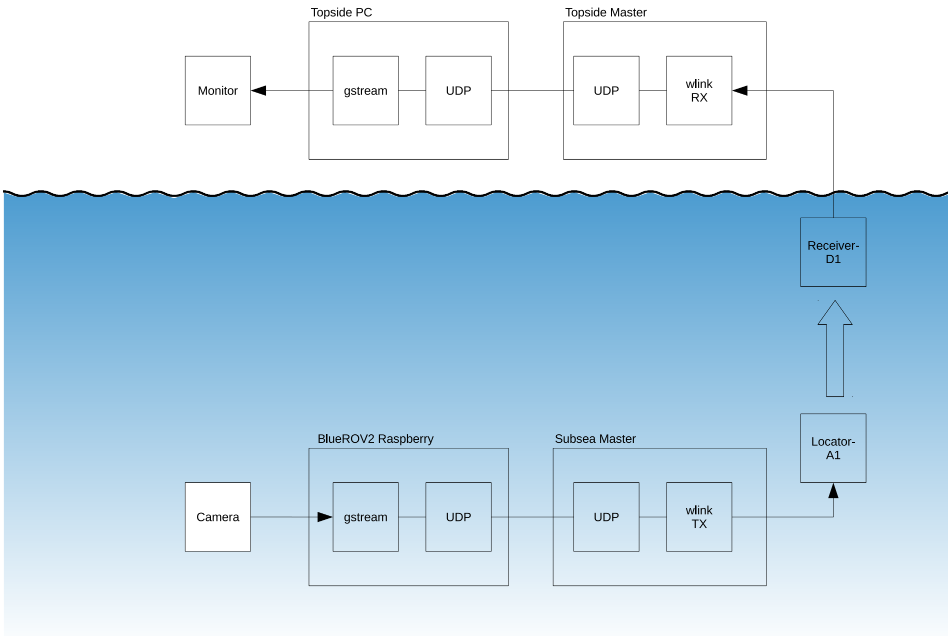

Figure 2.4: Setup of the Underwater Modem Uplink providing tether-less video stream

Chapter 3

Hardware and Software

3.1 Hardware

This section the most important hardware, including the custom built BlueRov2 and the input devices.

3.1.1 BlueRov2

The BlueRov2[1] is an affordable, high-performance ROV delivered by BlueRobotics located in California. It contains open source electronics and software and is highly customizable with several different modules and configurations. The ROV used in this project utilises the "Heavy Configuration Retrofit Kit". This means that instead of the standard 6 thrusters, it is expanded to 8 thrusters, enabling control of all six degrees of freedom. Figure 3.1 shows a BlueRov2 with the Heavy configuration.

In addition to the heavy configuration, other payloads are mounted on the ROV. The first is a waterproof container for carrying a 16V LIPO battery. Since the ROV is going to be controlled without a tether, this is a must. The second payload is another waterproof casing, containing the Subsea Master Modem which enable underwater communication. The resulting ROV, shown in figure 3.2 is the experimental platform for this work.

[1] https://www.bluerobotics.com/store/rov/bluerov2/

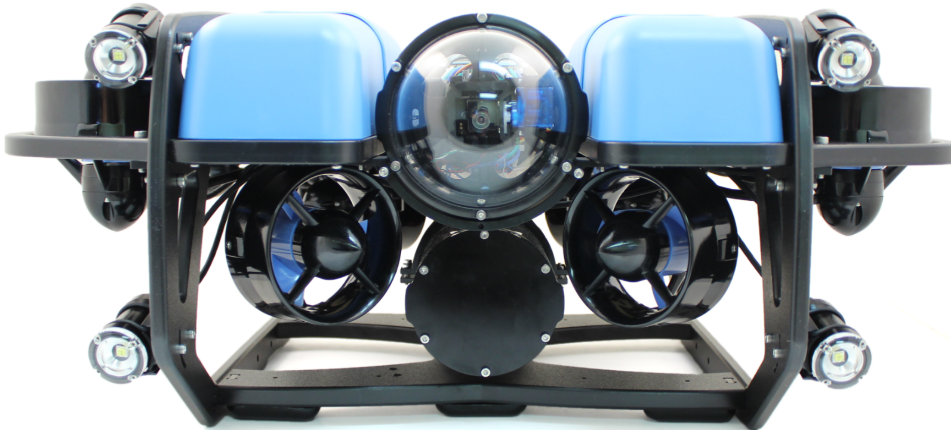

Figure 3.1: BlueRov2 with Heavy Configuration. Image courtesy: `https://www.bluerobotics.com/store/rov/brov2-heavy-retrofit-r1-rp/`

Table 3.1 lists the most important pieces of hardware on the ROV. The eight thrusters are controlled by the BlueRobotics Basic 30A ESC (electronic speed controllers). The Pixhawk FCU (see figure 3.3), running ArduSub firmware, receives input from all the sensors on board to provide control of the vehicle.

Table 3.1: Hardware BlueRov2

Electronic speed controller	BlueRobotics Basic 30A ESC
Onboard Computer	Raspberry PI 3
Flight Controller Unit	Pixhawk4
Subsea Master Modem	Master D1 WL-21008
Locator	Locator A1 WL-21009
Receiver	Receiver D1 WL-21005

The Subsea Master Modem is mounted as an additional payload to provide tetherless control of the ROV. To be able to use the Subsea Master, three transducers are mounted on the ROV, a receiver for the Downlink and two locators for the Uplink. The Onboard Computer (OBC), a Raspberry Pi 3 running Ubuntu Mate (figure 3.4, interfaces with the Subsea Master and runs the software that interfaces with the Pixhawk FCU.

Figure 3.2: Custom BlueRov2 from BlueRobotics built for wireless control. The bottom payload contains the Subsea Master. The middle payload contains the battery and the top payload is the electronic housing where most of the electronics is located. The D1 Receiver needed for the Downlink is mounted on the side of the ROV.

Figure 3.3: The Pixhawk4 Flight Controller Unit. Image courtesy: https://docs.px4.io/en/flight_controller/pixhawk4.html

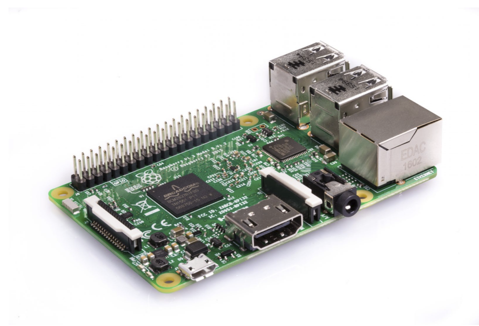

Figure 3.4: The Raspberry Pi 3 Onboard Computer. Image courtesy: `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`

3.1.2 Input Devices

To be able to control the ROV, an input device must be used for piloting. An Xbox controller (figure 3.5) was integrated early and is the default device when controlling the ROV. As a contribution to the SWARMS project, Inventas AS developed the Intuitive Input Device (IID) which needed to be integrated in this project. It consists of two platforms with several buttons and knobs having 3D mouse capabilities. This gives the pilot opportunity to give input in 12 degrees of freedom. A figure of the IID is showed in figure 3.6.

Figure 3.5: Xbox controller, default input device. Image courtesy: `https://www.microsoft.com/accessories/nb-no/products/gaming/xbox-360-wireless-controller-for-windows/52a-00005`

Figure 3.6: Intuitive Input Device (IID) from Inventas AS. Note that the device consists of the two platforms on either side of the keyboard, as well as an control box not shown in the figure. Image courtesy: Inventas AS

3.2 Software

In this section, important software used during in the project is introduced. These include

- Gazebo: 3D simulator used for Software In The Loop Testing by simulating the BlueRov2 in an underwater environment.
- Robot Operating System (ROS): A flexible framework for writing robot software.
- Ardusub: open-source firmware for ROVs and AUVs running on the Pixhawk FCU.
- Python: Python scripts is used for enabling tether-less control of the ROV. These are introduced in chapter 5: Implementation.

Note that a list of all important software used in the thesis is collected in in appendix A.1.

3.2.1 Gazebo

Gazebo is a three dimensional dynamic simulator that accurately and efficiently simulates robots in complex indoor and outdoor environments. It is easy to integrate Gazebo with other frameworks such as ROS, which makes it ideal for this project.

Gazebo is used for simulating the BlueRov2 in an underwater environment, providing a full physics engine. One of the advantages of using this platform is that

it enables the possibility to test the ROV without using its hardware, so called Software In The Loop testing (SITL). Correct pilot input can be tested, as well as verifying that the correct modes of the ROV is set.

One of the limitations of using Gazebo and SITL is that it cannot verify that the software works on the actual hardware on the ROV. For this, Hardware In The Loop (HITL) testing is needed

Figure 3.7 shows the environment when simulating the BlueRov2 in Gazebo.

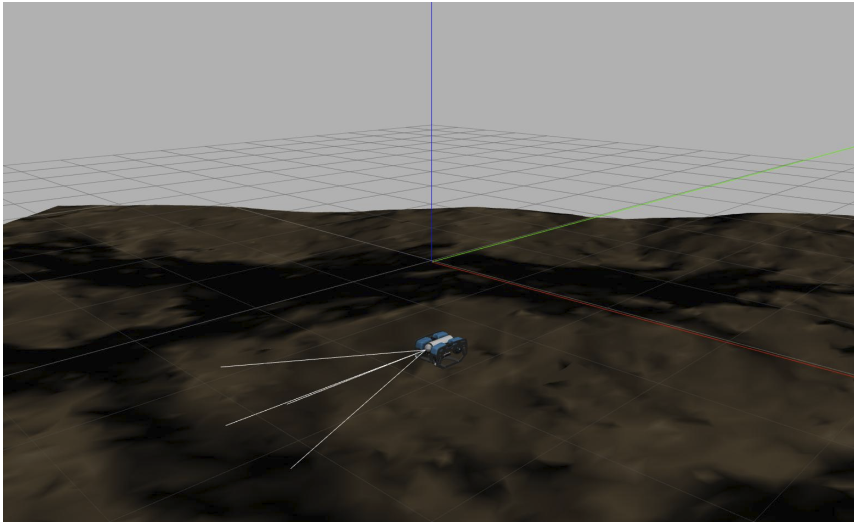

Figure 3.7: Simulating BlueRov2 in Gazebo

3.2.2 Robot Operating System

ROS is a flexible framework for writing robot software. It contains a large collection of open-source software libraries. A ROS system contains a number of independent nodes. The nodes communicate with each other using a publisher / subscriber messaging model. One big advantage of ROS is that the nodes do not have to be on the same platform or even the same architecture.

One important ROS package used in this project is MAVROS. It extends the MAVLink[2] protocol for use in a ROS environment. MAVROS interfaces with the Pixhawk FCU, running ArduSub firmware.

[2] MAVLink (Micro Air Vehicle Link) is a protocol for communicating with small unmanned vehicles.

An important concept in ROS is "topic". Topics use the publish/subscribe paradigm so that nodes can exchange messages. An extensive list of topics can be accessed through the MAVROS node. Examples are `/mavros/global_position/` which can be used to give the ROV reference coordinates for autonomous control. `mavros/hil/` for Hardware In The Loop testing, and `mavros/rc/` which can be used for controlling the servo that tilts the camera. In this thesis, the most important topics include `mavros/state` and `mavros/rc/override`.

The publish / subscribe model used in ROS is a very flexible communication paradigm, but its many-to-many one-way transport is not suitable for request / reply interactions, such as changing control modes. Request / reply is done via a ROS Service. Setting lights, changing modes and arming/disarming the vehicle are all done through services.

3.2.3 ArduSub

ArduSub is a fully-featured open-source software for controlling ROVs and AUVs. It is a part of the ArduPilot project, and was originally derived from the ArduCopter code. ArduSub includes capabilities such as feedback-stabilization control, depth- and heading-hold, and autonomous navigation. A list of the different modes used in the thesis is shown in table 3.2.

Table 3.2: Control modes of the ROV provided by ArduSub

Mode	Description
Manual	Manual mode passes the pilot inputs directly to the motors, with no stabilization. ArduSub always boots in Manual mode
Stabilize	Stabilize mode is like Manual mode, with heading and attitude stabilization
Depth hold	Depth Hold is like Stabilize mode with the addition of depth stabilization when the pilot throttle input is zero. A depth sensor is required to use depth hold mode

A great feature with ArduSub is that it provides a platform for Software In The Loop Testing. It simulates a Pixhawk FCU, enabling integration tests to be done without the use of hardware. In addition, it is easily interfaced with Gazebo.

Chapter 4

Theory

This chapter gives an introduction to Software Development Life Cycles (SDLC) along with several popular SDLC models. Furthre, software testing tools used in successful SDLC models will be explored. These tools will be used when developing, integrating and testing the software described in this thesis.

Keywords

- **Methodology:** A recommended way of doing something. A software development methodology will therefore be a recommended way to develop a software. This refers to different models, i.e Waterfall Model, Expert Programming or SCRUM. The term methodology is interchangeable with the term method.
- **Stage:** A SDLC is a series of stages within a methodology followed in the process of developing and testing software. A stage is a segment of an SDLC that consists of certain types of activities. The term stage is often interchangeable with phase, depending of the actual literature.

4.1 Software Development Life Cycle

A Software Development Life Cycle is a construction imposed on the development of a software product [6]. It involves methodologies for designing, building and maintaining software systems. There exists many SDLC models, such as the Waterfall Model, Incremental / Iterative Model or SCRUM, each describing approaches to activities or tasks that take place during the development life cycle.

In the following sections, different SDLC models are explored. This includes the traditional Waterfall, V-model and Iterative and Incremental Model, and a few models that use the Agile Methodology. Common for all SDLC model is that the software development process is divided into phases such as Requirement Analysis, Design, Coding, Testing, and Maintenance [12]. These activities are carried out in different ways depending on the actual model. These terms are explained below

- Requirement Analysis: initial stage of the SDLC. The goal of this stage is to understand the client's needs and requirements and document them properly.
- Design: The first step to move from the problem domain towards the solution domain. The goal of this stage is to transform the requirements into structure.
- Coding: The solution provided in the Design stage is converted into code.
- Testing: The most important stage. Testing contributes to the delivery of high-quality software products [15].
- Maintenance: This stage starts after delivery of the product. Any modifications due to errors are implemented in this stage.

4.2 Traditional SDLC models

4.2.1 Waterfall model

The Waterfall model is known as one of the traditional SDLC methods [16]. Figure 4.1 show the five different stages in the classical Waterfall model. It is based on the assumption that each stage is executed a single time in a specific sequence. Hence, the Waterfall model has a linear sequential flow, top-to-bottom. It emphasises planning in early stages, and it ensures finding design flaws before they develop.

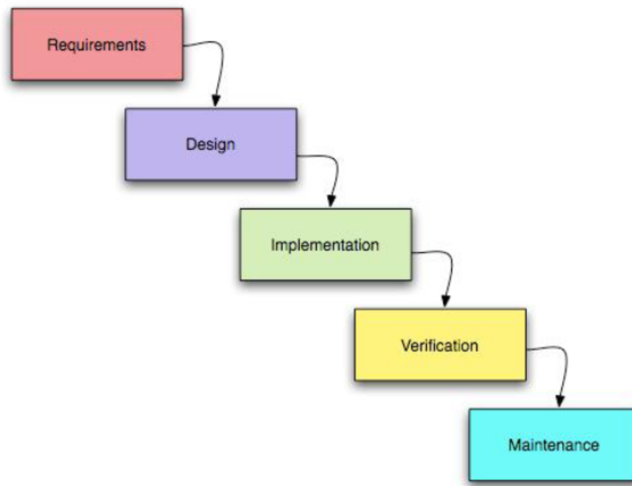

Figure 4.1: Waterfall development life cycle. Image courtesy: [7]

Table 4.1 shows the strengths and weaknesses of the Waterfall Model [1]. It is clear that the simplicity of the model affects the flexibility and it's ability to handle large and complex projects. Ibid points out types of project when the Waterfall model may be used:

- When the quality of the deliverable is more important than cost or schedule.
- When requirements are very well known, clear and fixed.
- Porting an existing product to a new platform.

4.2.2 V-model

Like the Waterfall model, the V-Shaped life cycle [8] is a sequential path of execution of processes. Each stage must be completed before the next stage begins. Figure 4.2 shows the stages of the life cycle.

Unlike the Waterfall model, testing is much more emphasized in the V-model. The testing procedures are developed early, before any coding is done, during each of the stages preceding implementation.

Strengths and weaknesses of the V-model are shown in table 4.2. It shares many of the same attributes as the Waterfall model. They also share some use cases - the

Table 4.1: Waterfall model strengths and weaknesses

Strengths	Weaknesses
Simple to explain and implement	Inflexible
Easy to estimate cost	All requirements must be known up-front
Minimises planning overhead	Backing up to solve mistakes is difficult
Works well on mature products	High amounts of risk and uncertainty
Provides structure to inexperienced teams	A non-documentation deliverable is only produced in the final stage
Define before design, design before code	Customers may have little opportunity to preview the systems until it may be too late

V-model should be used in a small to medium sized project, where the requirements and development tool are well known.

Table 4.2: V-model strengths and weaknesses

Strengths	Weaknesses
Simple to explain and implement	Very inflexible, like the Waterfall model
Each phase has specific deliverables	Adjusting scope is expensive and difficult
Higher chance of success over the Waterfall model due to the development of test plans early in the life cycle	All requirements must be known up-front
Provides structure to inexperienced teams	A non-documentation deliverable is only produced in the final stage
Works well for small projects where requirements are easily understood	The model does not provide a clear path for problems found during testing phases

4.3 Prototyping Model

According to [9], the Prototyping Model places more effort on creating the actual software, instead of concentrating on documentation through the development process. This model requires more user / client involvement and allows them to see

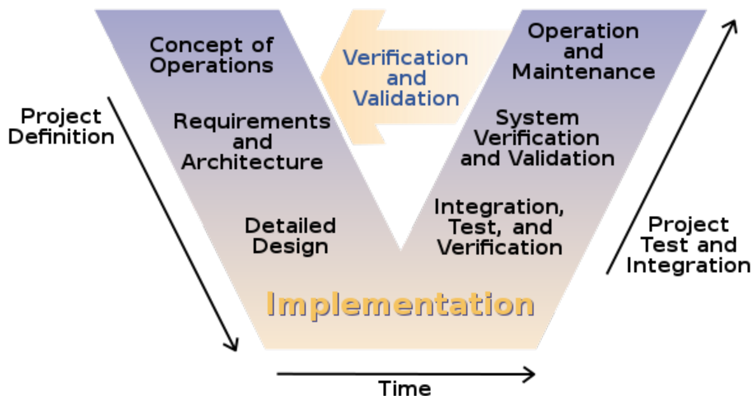

Figure 4.2: V-model development life cycle. Image courtesy: `https://en.wikipedia.org/wiki/V-Model_(software_development)`

and interact with a prototype to provide more complete feedback. This is because the actual software is released several times. A prototype is a working model that is a functionally equivalent to a component of the product. Figure 4.3 shows the development cycle of this model.

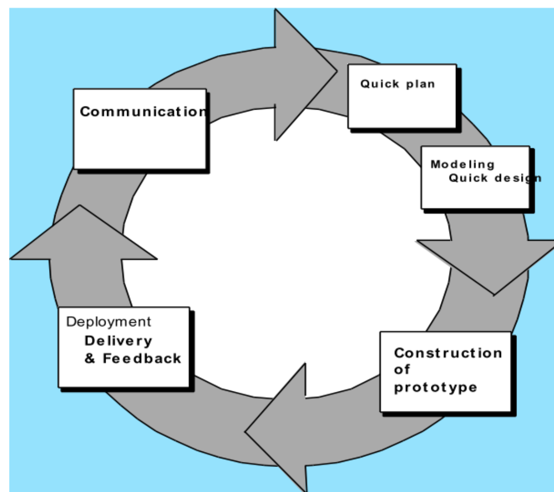

Figure 4.3: The Prototyping model. Image courtesy: [18]

Strengths and weaknesses of the Prototype Model are listed in table 4.3. The model should be used when it is required to have a lot of interaction with the end users or clients. Web interfaces are best suited for this model [9].

Table 4.3: Prototype model strengths and weaknesses

Strengths	Weaknesses
May improve the quality if requirements and specifications	Focusing on a limited prototype can distract developers from analysing the complete project
Provides better and more complete feedback and specifications	Clients may think every prototype is intended to be thrown away
The client is more involved	
Reduced time and cost	

4.4 Incremental / Iterative model

[12] describes this model as a combination of elements of the waterfall model in an iterative fashion. Multiple development cycles takes place and produces deliverable increments of the software (see figure 4.4). This model constructs a partial implementation of a total system in each increment, going through the entire cycle each time. Table 4.4 shows the advantages and disadvantages of this model.

Table 4.4: Incremental / Iterative model strengths and weaknesses

Strengths	Weaknesses
Develop high-risk or major functions first	Requires good planning and design
Each phase has specific deliverables	Requires early definition of a complete and fully functional system to allow for the definition of increments
Risk is spread across smaller increments instead of concentrating in one large development	The model does not allow for iterations within each increment
Reduces the risk of failure and changes to the requirements	

The incremental / iterative model should be used when the requirements are clearly defined and understood. It is also beneficial to use this model when the product needs to go to the market early and when a new technology is being used.

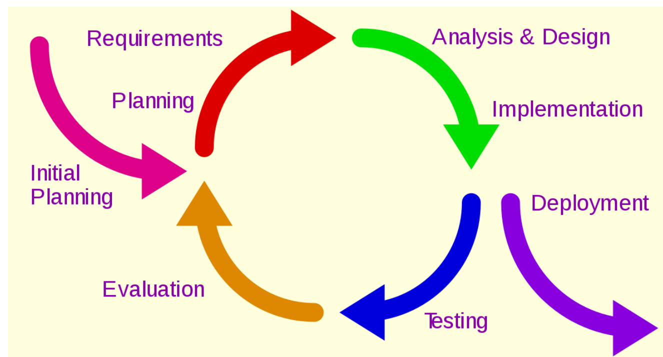

Figure 4.4: The incremental / iterative SDLC model. Image courtesy: `https://en.wikipedia.org/wiki/Iterative_and_incremental_development`

4.5 Agile Development

Unlike conventional SDLC models, Agile Development is a time-based process model that aims to deliver software products quickly by using light-weight processes, modular process structures and iterative process actions [3] (see figure 4.5). There exists several models that follow the agile methodology, including Extreme Programming (XP), Feature Driven Development (FDD), SCRUM, Kanban and Lean Software Developent. All methods are unique in their approaches, but they all share a common vision and core values. These principles are gathered in the "Manifesto for Agile Software Development".

4.5.1 Agile Manifesto

The Agile Manifesto has its origin from 2001 in the Wasatch mountain of Utah, where seventeen software developers representing different SDLC methodologies met to talk and discuss to find common ground in software development. What emerged was the Manifesto for Agile Software Development [5] with the following 12 principles [2]:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

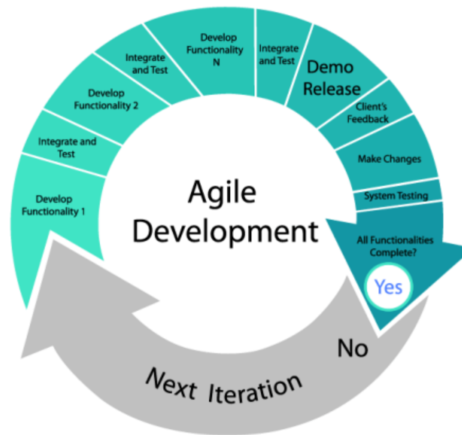

Figure 4.5: The agile development process. Image courtesy: `https://number8.com/common-mistakes-using-agile-method/`

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

4.5.2 Extreme Programming

According to [4], Extreme Programming turns the conventional software process sideways. Instead of planning, analyzing and designing far into the future, XP programmers do all of these activities, in small amounts, throughout the development (see figure 4.6).

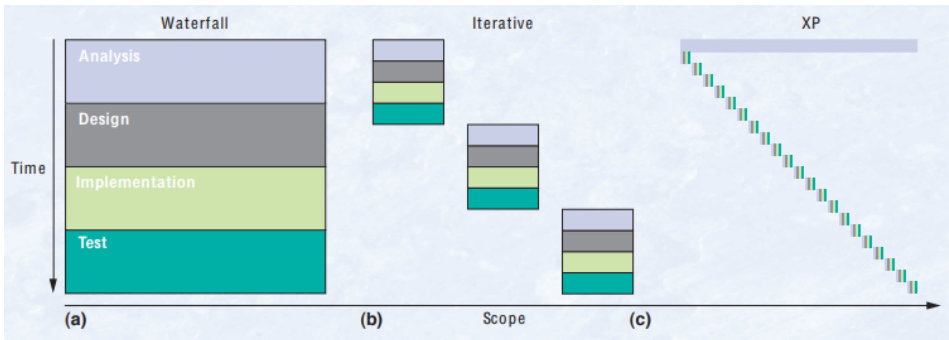

Figure 4.6: a) Waterfall Model. b) Iterative Model. c) Extreme Programming (XP). Image courtesy: [4].

Below is a summary of some of the major practices in XP

- Small releases: New releases are made often, anywhere from monthly to daily.
- Simple design: The design runs all the tests, communicates everything the programmers want to communicate, no duplicate code. "Say everything once and only once".
- Tests: Unit tests are widely used.
- Refactoring: The design of the system is evolved through transformations of the existing design.
- Pair programming: All production code is written by two people.
- Continuous integration: New code is integrated with the current code after no more than a few hours. All tests must pass or the changes are discarded.

Table 4.5 shows the advantages and weaknesses of XP.

Table 4.5: Extreme Programming strengths and weaknesses

Strengths	Weaknesses
Lightweight methods suits small-medium sized project	Difficult to scale up to large projects
Iterative	Pair programming is costly
Emphasis on final product	Needs experience

4.5.3 Feature Drive Development

Feature Driven Development (FDD) is a production process which is highly oriented on resulting out small blocks of client valued functionality (see figure 4.7. This drives developers to come up with working features once every two weeks typically and it can track down the project progress with precision. FDD, which is one of a number of agile development processes, is an iterative and incremental software development process having the main purpose of delivering tangible working software repeatedly in a timely manner.

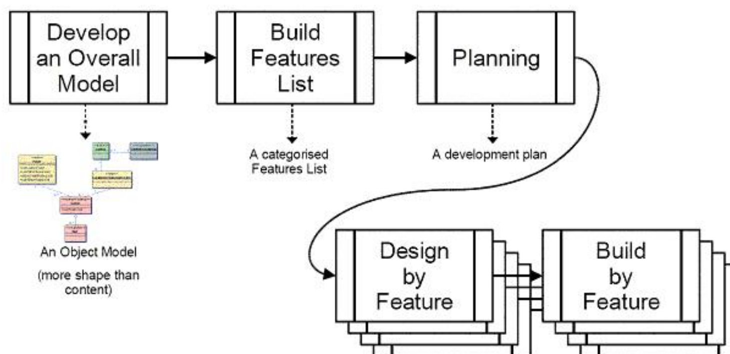

Figure 4.7: The Feature Driven Development method. Image courtesy: `http://www.martinbauer.com/Articles/FDD-and-Project-Management`

The model consists of five basic processes [13]:

1. Develop overall model
2. Build feature list
3. Plan by feature
4. Design by feature

5. Build by feature

Advantages and disadvantages is listed in table 4.6

Table 4.6: Feature Driven Development strengths and weaknesses

Strengths	Weaknesses
Iterative	Errors are often discovered late
Emphasis on quality at all steps	Not as powerful on smaller projects
Frequent deliverables	No written documentation

4.6 Testing in Software Development Life Cycle

"It is clear that without testing it is not possible to implement an effective product" [11]. Software testing is a very broad term containing activities along the development cycle and beyond, involves many technical and non-technical areas, and is aimed at different goals. [18] defines software testing as the process of executing a program or system with the intent of finding errors.

Different tools can be used for testing software during different phases in a software life cycle. A huge amount of software test tools and methods can be found in literature and online. For the purpose of this thesis, tools introduced in [14] will be used. Among the several tools presented in this book, the following will be used

- Unit Testing: Verify that the smallest components of a system functions properly, i.e pressing a button.
- Integration Testing: The system run tasks that involves more than one component to verify that it performs accurately.
- System Testing: Tests that verifies the operation of the entire system.
- Acceptance Testing: Real-world test. Client accepts the product or not.

The testing tools listed above is used extensively in the V-model. To counter the disadvantages of the V-model, features from the Agile Methodology and Incremental / Iterative Model are implemented. Instead of testing after all code is written, small modules are tested throughout the development using unit tests. Furthermore, integration tests are carried out frequently to ensure that the modules work as intended when put together. When it's time to test system features, system testing

is completed. Acceptance tests were done during the SWARMs validation-week prior to the demonstration.

An advantage of this scheme is that it allows for developing and testing small modules of the system throughout the life cycle. Additionally, it allows for "backtracking" if some of the tests fails. Figure 4.8 shows the testing scheme. Chapter 6 explains how these tests were implemented in this project.

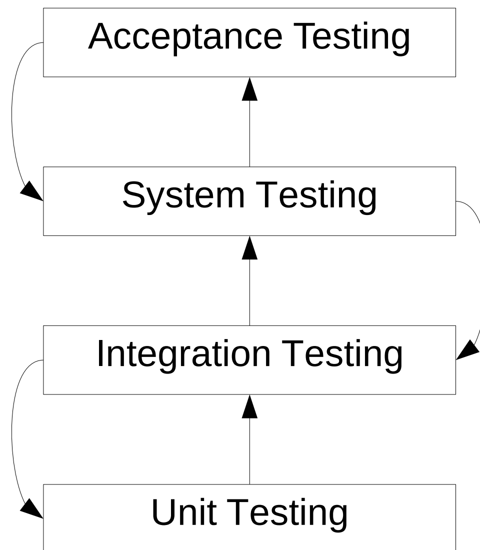

Figure 4.8: Testing scheme. Unit and integration testing are done more frequently than system- and acceptance testing. System testing is done whenever larger features needs testing, and acceptance testing was done prior to the SWARMs demonstration.

Chapter 5

Implementation

This chapter gives an insight of key aspects of the software running on the topside computer and the subsea Onboard Computer (OBC). Appendix A.1 lists the most important software packages along with online resources.

Keywords

- **Input device:** A device providing control commands from a pilot. It refers to either an Xbox controller or Intuitive Input Device (IID).
- **OBC:** Onboard Computer. Refers to the Raspberry Pi on board BlueRov2.
- **ROS:** Robot Operating System. A flexible framework for writing robot software.
- **Socket:** A network socket is an internal endpoint for sending or receiving data within a node on a computer network.
- **UDP:** User Datagram Protocol. A minimal message-oriented networking protocol for connection-less transfer of information.

5.1 Overview

Figure 5.1 shows an overview of the software developed to run on the topside and subsea computer. Software related to the Underwater Modem is not a part of this. This means that the `topside` only interfaces with the Topside Master at a specific IP address and UDP port. Likewise for `subsea`, it only interfaces with the Subsea Master mounted on the BlueRov2. The communication between Topside and Subsea Master will be treated as a ”black box“.

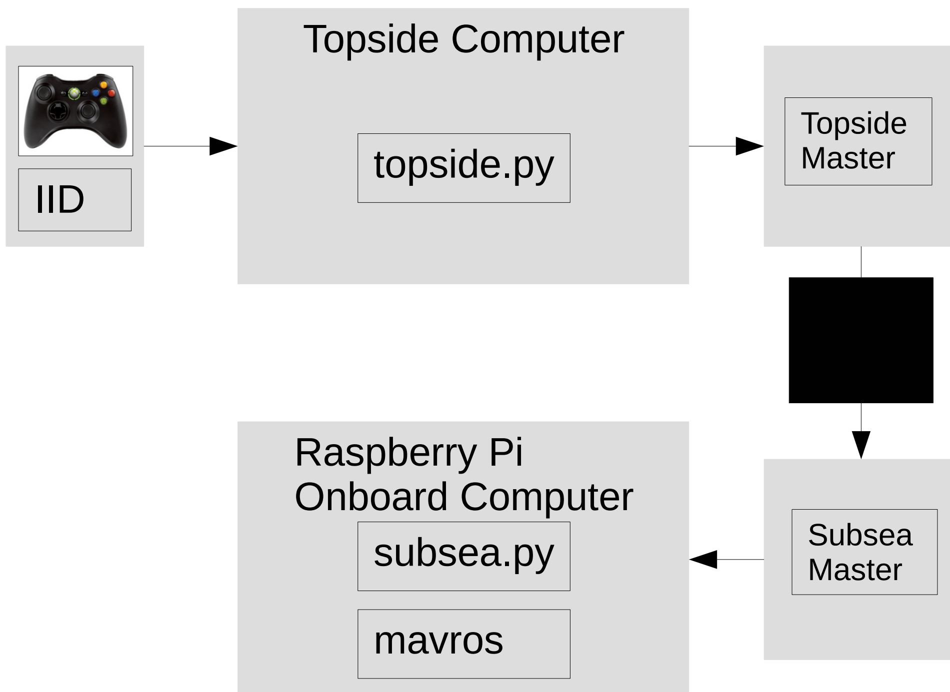

Figure 5.1: Software overview

The software on both the topside and subsea computer is written in Python and can run on any computer capable of running Python 2.7 or Python 3.0. Additionally, it is required to have access to an Ethernet port as well as an USB port for the input device.

5.2 Topside software

The purpose of the topside software is to get input from an input device, encode the control commands and send them as UDP packets to the Topside Master over Ethernet in a reliable way. Additionally, the UDP data is logged along with a time stamp to be able to correlate packages which are sent, with received packages on the OBC. An overview of the topside program flow is shown in figure 5.2.

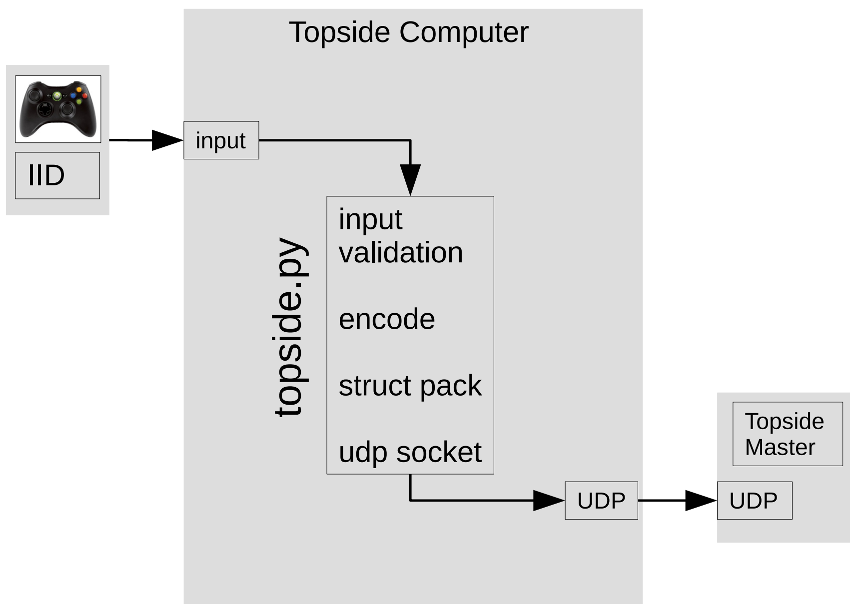

Figure 5.2: Topside program flow

Several Python packages are used, with `struct`, `argparse`, `xboxdrv` and `socket` being the most important.

- `struct` is used to pack integers to binary numbers. Essential for encoding control commands from the input device.
- `argparse` provides a mean to run a Python script with arguments, generates help messages and issues errors when the user gives the program invalid arguments. If a user wants to figure out how to run the program, one simply uses the following command in a terminal window: `python topside.py -h`. The following text is printed to the screen:

```
usage: topside.py [-h] [--ip IP] [--port PORT] [--device DEVICE]

Topside Underwater Modem driver for controlling BlueRov2

optional arguments:
-h, --help       show this help message and exit
--ip IP          IP adress to send UDP packets to.
                 Default "192.168.2.95" (topside master IP).
                 Type: [str]
--port PORT      Port to send UDP packets to.
                 Default 5005 (topside master port).
                 Type: [int]
--device DEVICE  Input device, "xbox" or "iid".
                 Default "xbox". Type [str]
```

- `xboxdrv` provides a reliable way of interfacing with an Xbox controller.
- `socket` enables low-level network interfacing. It is used for sending data as UDP packets to a specific IP address and port in a network.

5.2.1 Input devices

Figure 5.3 shows the default input device, a wireless Xbox controller and how the controls are allocated on the device. The axes on the figure refers to nomenclature used in the ArduSub software. Throttle refers to the heave movement of the vehicle. Yaw axis controls the heading, forward axis controls the surge and the lateral axis controls the sway movement. Roll and pitch is not controlled.

Figure 5.4 shows the SWARMS Intuitive Input Device from Inventas AS. Interfacing with the IID was done by receiving input as UDP packets over Ethernet from its control box. Below is a list of attributes of each device, compared with each other.

- Usage: As the Xbox controller is widely known and used, the IID takes more getting used to. The controls on the IID are stiff, which makes it hard for a user to move the joysticks, without moving the platform.
- Integration: The Xbox controller is plug and play, while the IID needed custom code to get working.

- Options: The IID have more degrees of freedom of control on two 3D joysticks, as well as more buttons and knobs. It has potential for controlling advanced vehicles with different payloads.

Figure 5.3: Xbox controller input device

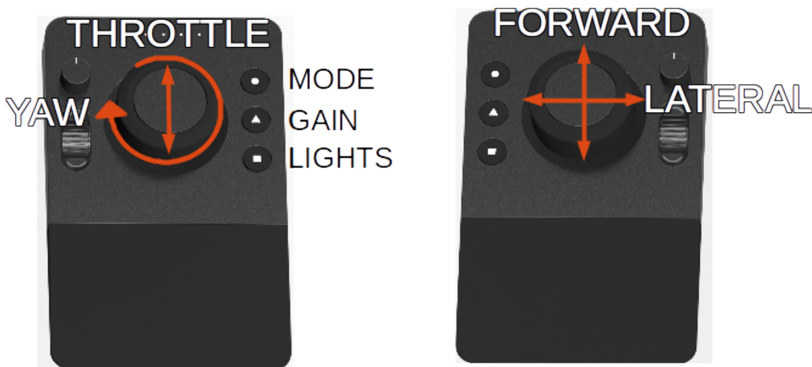

Figure 5.4: SWARMS Intuitive Input Device

5.2.2 Encoding

To be able to control the ROV with a small amount of data, the control commands needs to be encoded. The Downlink provides a data stream of 4 bytes per second,

with a latency between 200 and 500 ms under good conditions. The data is sent as frames of 2 bytes, therefore all control signals from the input device are packed into 2 bytes (see figure 5.5).

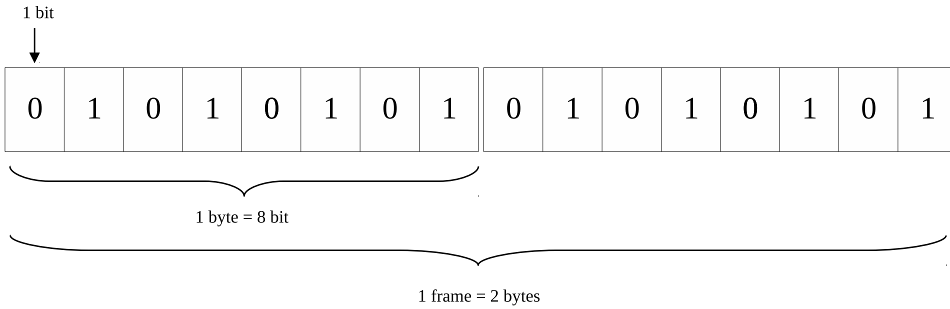

Figure 5.5: Data frame sent by the Downlink

The forward and lateral commands from the input device are packed into the first byte of the frame. This is done by representing both of the right joystick axes as 4 bits each. With 4 bits, we can get $2^4 = 16$ different values, in the range $\left[0 \to 15\right]$. Both of the axes outputs a floating number between $\left[-1.0 \to 1.0\right]$. This gives a resolution of $\frac{1.0-\left(-1.0\right)}{16} = 0.125$ for each axis. The first 4 bits of the first byte (most significant bits, MSB) are allocated for the forward axis, and the last 4 bits (least significant bits, LSB) are allocated for the lateral axis (see figure 5.6).

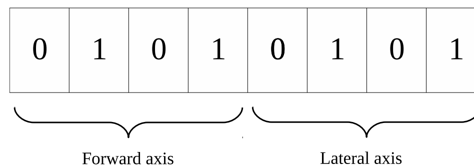

Figure 5.6: Data byte for forward and lateral axis

Code excerpt 5.1 shows how the joystick command is encoded. cmd1 refers to the forward command, while cmd2 refers to the lateral command. First, a clamp function makes sure the commands are valid by limiting them to $\left[-1.0 \to 1.0\right]$. Additionally, it sets the commands to zero if $|\text{cmd}| < 0.1$. Next, the commands are scaled so that the values are in the range of $\left[0 \to 15\right]$. To pack the commands into a single byte, the binary left shift operator $\ll$, and the bitwise OR operator $|$ is used. Lastly, the command is packed as a binary number using the `struct.pack()` function call.

```
145 def encode_joy_cmd(cmd1, cmd2):
146
147     cmd1 = clamp(cmd1, -1.0, 1.0, 0.1)
148     cmd2 = clamp(cmd2, -1.0, 1.0, 0.1)
149
150     msb = int(7.5 * cmd2 + 7.5)
151     lsb = int(7.5 * cmd1 + 7.5)
152
153     cmd = msb << 4 | lsb
154
155     return struct.pack('B', cmd)
```

Python code excerpt 5.1: Encode forward and lateral joystick command

Next, the throttle and yaw axis of the joysticks needs to be encoded, as well as the three buttons. Figure 5.7 shows how the command is structured as a single byte. Note that only two bits are allocated to the the yaw command, leaving the two other bits for the buttons. This will give the yaw command a range of $[0 \rightarrow 3]$. Doing so, the yaw command will provide binary control around the yaw axis, without modulation.

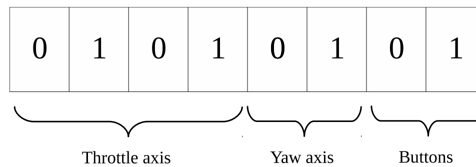

Figure 5.7: Data byte for throttle, yaw and buttons

Code excerpt 5.2 shows the encoding of last byte of the data frame. cmd1 and cmd2 refers to the throttle and yaw command respectively, and the next three arguments are the button inputs. The clamping of the input commands are done similarly as the first byte, the only difference being that the yaw command is set to zero if $|\text{cmd}| < 0.25$. The buttons are encoded in such a way that if no buttons are pressed, the value is set to 0. If one of the buttons are pressed, the button command will range from $[1 \rightarrow 3]$, depending on what button is pressed. Note that only three button inputs can be encoded using two bits. Even though we have a range of $[0 \rightarrow 3]$, the last value must be assigned when no buttons are pressed.

5.2.3 Transmitting UDP packets

After encoding the pilot inputs, the commands are sent as a two byte UDP packet over Ethernet using the `socket.sendto()` function call, see figure 5.8. As the Downlink is subject to packet loss and has limited bandwidth, some special considerations need to be taken into account, see figure 5.9. Firstly, the transmission rate of the UDP packets is set to 10 Hz and updating pilot inputs is set to 100 Hz. Secondly, the the command inputs are subject to a deadband check. This means that if the absolute value of the pilot input in the joystick axes are less than 0.125, the commands are set to zero. Lastly, if no buttons are pressed or the joystick axes is within the deadband, the UDP packets are sent 5 times before stopping. This is to ensure that the Downlink isn't overloaded with useless commands.

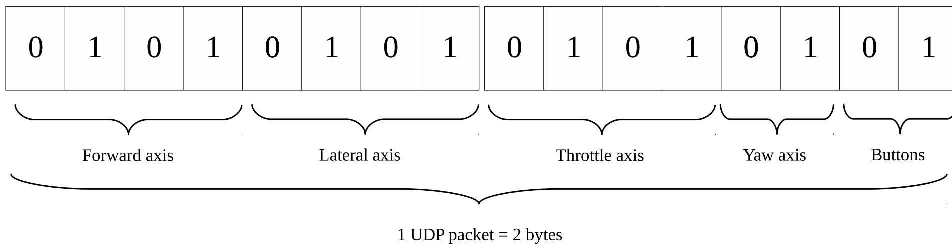

Figure 5.8: UDP packet to be sent to Topside Master over Ethernet

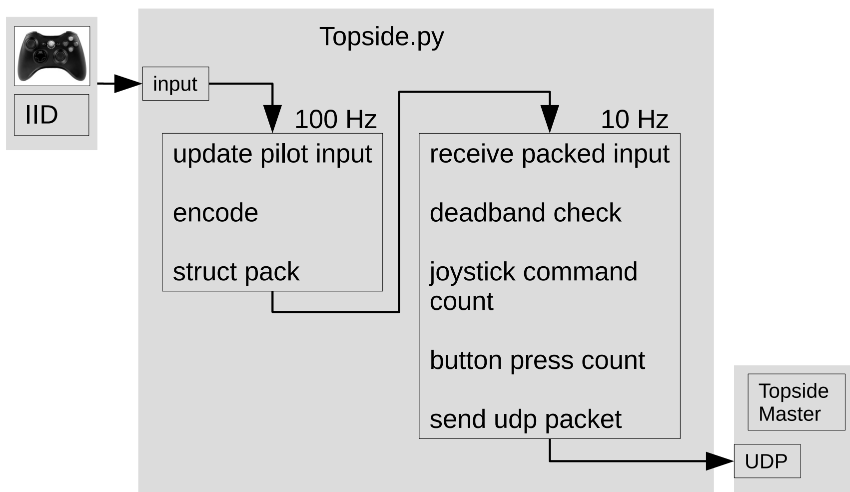

Figure 5.9: Sending UDP packets to Topside Master

5.3 Subsea software

An overview of the subsea program flow is shown in figure 5.10. Its purpose is to is to receive data from the Subsea Master, decode the messages, and control the ROV by interfacing with the Pixhawk Flight Controller using MAVROS. In contrary to the topside software where a single thread is running, the subsea software needs to be able to handle multiple threads and concurrency. UDP packets are received from the Subsea Master asynchronously, then logged and decoded at the same time as interfacing with the Pixhawk FCU. Important python packages include `rospy` and `asyncore` as well as the ROS package `MAVROS`.

- `rospy` is a Python library for using ROS. It enables interfacing with the ROS framework and the MAVROS package.
- `MAVROS` is a ROS package used for interfacing with the Pixhawk FCU by extending the MAVLink protocol used for controlling the ROV.
- `asyncore` provides infrastructure for asynchronously handling socket servers and clients. It is used for receiving UDP packets.

5.3.1 Receive UDP packets asynchronously

Code excerpt 5.3 show the asynchronous UDP client class. When creating a `AsyncUdpClient` object, it creates a socket used for receiving packets. The `handle_read()` function is called every time there is something to read. Here, the received data is handled using the `msg_handler()` function (see code excerpt 5.4). The message handler logs the data and checks if the received packet has a frame of two bytes. If it does, it calls the `decode_joy_msg()` function and updates the joystick input.

5.3.2 Decoding

Code excerpt 5.5 shows how the received UDP packet is decoded. First, it starts by unpacking the data using the `struct.unpack()` function call. `data[0]` is the first byte of the frame and corresponds to the x and y-axis of the right joystick. `data[1]` is the second byte of the frame and corresponds to the x and y-axis of the left joystick, as well as the button presses.

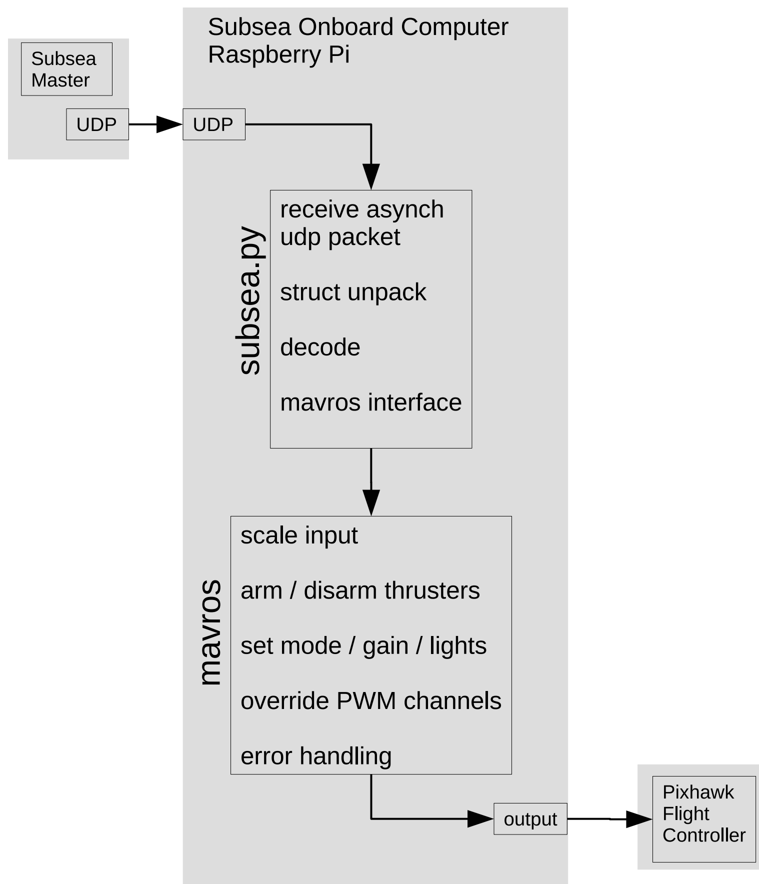

Figure 5.10: Subsea software overview

Next, the correct bits are extracted using the right shift binary operator $\gg$ and the bitwise AND & operator. Lastly, the extracted data is scaled to fit in the $\left[-1.0 \rightarrow 1.0\right]$ range using the `scale_int_to_float()` function (see code excerpt 5.6).

5.3.3 MAVROS

MAVROS is a ROS package used for interfacing with the Pixhawk FCU in the ROS framework. One of the advantages of ROS is that it offers thread-safe operations using publishers and subscribers. Figure 5.11 shows how the MAVROS node interacts with the "wlink" node by sharing topics using the publisher / subscriber

model.

Another advantage of ROS is that it packs data into standard data structures, such as the `Joy.msg` structure. Any input device can be packed into this structure. After the data has been decoded, the joystick input is updated, see code excerpt 5.7.

The input is published into the `/Joy` topic. A publisher is created in the following way:

```
joy_pub = rospy.Publisher("/Joy", sensor_msgs.msg.Joy, queue_size=5)
```
.

This enables us to subscribe to this topic whenever a `Joy.msg` structure is published using a callback function:

```
rospy.Subscriber("/Joy", sensor_msgs.msg.Joy, joy_callback)
```
.

The callback function `joy_callback()` is called every time something is published to the `joy_pub` topic. Note that a FIFO queue of size 5 collects up to 5 joystick messages, to ensure that no received data is lost. The `joy_callback()` function is where most of the control of the ROV happens. Here, the joystick input is being put to use, lights are turned on and off, and mode and gain is set. The different modes used is listed in chapter 3, table 3.2.

The thrusters of the ROV is controlled by setting the PWM values with the use of the `override_msg` structure in MAVROS. These are values between $\left[1100 \rightarrow 1900\right]$, which corresponds to the positive width of a PWM signal in milliseconds. A value of 1100 is provides full force backwards, 1500 is zero thrust and 1900 is full force forward. The gain setting will enable the pilot to use 50%, 75% or 100% of the PWM values $\left[1500 \pm 300\right]$.

Another important callback function is `state_callback()`. It is called every second when the topic `/mavros/state` is published to. While `Joy_callback()` controls the ROV, `state_callback()` handles timeouts, errors and safety. An example of this is that if data hasn't been received in 3 seconds, it will disarm the vehicle. This is very useful during testing when the ROV is being pulled in and out of water. The lights will turn off to save battery time, if no data has been received from the Subsea Master the last 30 seconds. The most important safety feature is that the ROV will automatically move to the surface if no data is received within the last 30 seconds.

Setting lights, arm / disarm, and set the mode of the vehicle is done through ROS services.

5.4 Linux Services

To ensure that the OBC runs all necessary software on start-up, Linux services are made. A Linux service is an application that runs in the background of the operating system to carry out essential tasks. Two services are made on the OBC, one for starting the Python script, `subsea.py` and one for starting MAVROS. Code excerpt 5.8 shows the service that starts a shell script, `run-wlink`, responsible for running `subsea.py` on start-up.

```
115 def encode_joy_btn_cmd(cmd1, cmd2, cmd3, cmd4, cmd5):
116 
117     cmd1 = clamp(cmd1, -1.0, 1.0, 0.1)
118     cmd2 = clamp(cmd2, -1.0, 1.0, 0.25)
119 
120     msb = int(7.5 * cmd2 + 7.5)
121 
122     if(cmd1 < -0.25):
123         val = 0
124     elif(cmd1 > 0.25):
125         val = 2
126     else:
127         val = 1
128 
129     lsb = val << 2
130 
131     btn_cmd = 0
132     if(cmd3 == 1):
133         btn_cmd = 1
134     elif(cmd4 == 1):
135         btn_cmd = 2
136     elif(cmd5 == 1):
137         btn_cmd = 3
138 
139     lsb |= btn_cmd
140 
141     cmd = msb << 4 | lsb
142 
143     return struct.pack('B', cmd)
```

Python code excerpt 5.2: Encode throttle, yaw and button command

```
307 class AsyncUdpClient(asyncore.dispatcher):
308     def __init__(self, host, port):
309         asyncore.dispatcher.__init__(self)
310         try:
311             self.create_socket(socket.AF_INET, socket.SOCK_DGRAM)
312             self.set_reuse_addr()
313             self.bind((host, port))
314         except:
315             # cleanup asyncore.socket_map before raising
316             self.close()
317             raise
318 
319     # This is called every time there is something to read
320     def handle_read(self):
321         data, addr = self.recvfrom(4096)
322         if(msg_handler(data) == True):
323             pass
324         else:
325             rospy.logdebug("Data handler failed\n")
326 
327     def handle_close(self):
328         self.close()
```

Python code excerpt 5.3: Asynchronous UDP Client

```
271 def msg_handler(data):
272     rospy.logdebug("msg_handler data: {} data length:
    ↪  {}".format(data, len(data)))
273     log_data(data)
274 
275     if(len(data) == 2):
276         joy_data = decode_joy_msg(data)
277         update_joy_input(joy_data)
278         return True
279     else:
280         return False
```

Python code excerpt 5.4: Message handler

```python
233 def decode_joy_msg(data):
234     joy_decoded = {'lx': 0, 'ly': 0, 'rx': 0, 'ry': 0, 'btn': 0}
235 
236     rx_ry = struct.unpack('B', data[0])
237     lx_ly_btn = struct.unpack('B', data[1])
238 
239     joy_decoded['ry'] = int(rx_ry[0]) >> 4
240     joy_decoded['rx'] = int(rx_ry[0]) & int(0x0F)
241     joy_decoded['ly'] = int(lx_ly_btn[0]) >> 4
242     joy_decoded['lx'] = (int(lx_ly_btn[0]) >> 2) & int(0x03)
243     joy_decoded['btn'] = int(lx_ly_btn[0]) & int(0x03)
244 
245     joy_decoded['rx'] = scale_int_to_float(joy_decoded['rx'])
246     joy_decoded['ly'] = scale_int_to_float(joy_decoded['ly'])
247     joy_decoded['ry'] = scale_int_to_float(joy_decoded['ry'])
248     joy_decoded['lx'] = joy_decoded['lx'] - 1.0
249 
250     return joy_decoded
```

Python code excerpt 5.5: Decode control commands

```python
230 def scale_int_to_float(value):
231     return (float(value) - 7.5) / 7.5
```

Python code excerpt 5.6: Scale integer to floating number

```python
def update_joy_input(joy_decoded):
    joy_msg.axes[JOY_LX] = set_joy_deadzone(joy_decoded['lx'])
    joy_msg.axes[JOY_LY] = set_joy_deadzone(joy_decoded['ly'])
    joy_msg.axes[JOY_RX] = set_joy_deadzone(joy_decoded['rx'])
    joy_msg.axes[JOY_RY] = set_joy_deadzone(joy_decoded['ry'])

    for i in range(0,4):
        joy_msg.buttons[i] = 0
    joy_msg.buttons[joy_decoded['btn']] = 1

    rospy.logdebug(joy_msg)
    joy_pub.publish(joy_msg)
```

Python code excerpt 5.7: Publishing joystick messages in ROS

```
Unit]
Description=Water Linked Modem driver

[Service]
ExecStart=/usr/bin/run-wlink
Restart=always
RestartSec=10
Environment="HOME=/home/pi"

[Install]
WantedBy=multi-user.target
```

Python code excerpt 5.8: Linux service for starting scripts at startup.

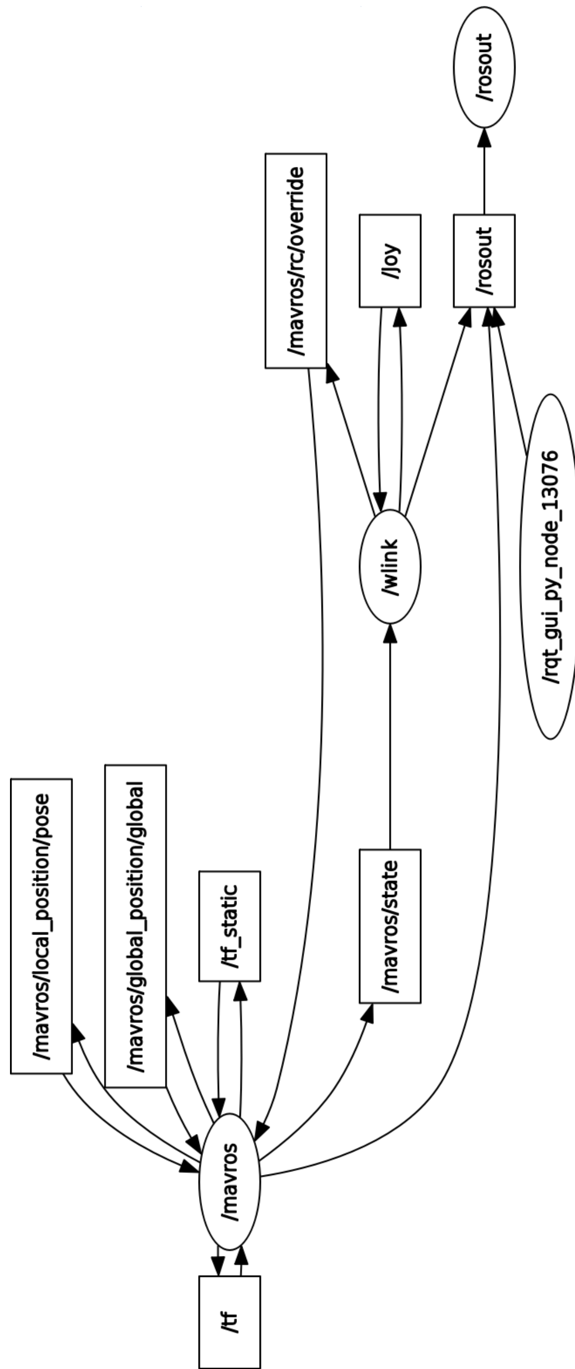

Figure 5.11: ROS node overview

Chapter 6

Testing

> "The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component"
>
> — Definition of testing [10]

Software testing comes in many forms and shapes. There is a large amount of tools and methodologies available as stated in chapter 4. This chapter gives a presentation of how testing tools were used in the duration of the thesis. The different tools include:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

6.1 Unit Testing

[10] defines Unit Testing as "Testing of individual hardware or software units or groups of related units." To tests modules and methods in topside.py and subsea.py, two Python packages were used, `unittest` and `nosetests` (see Appendix A.1).

These tools were used throughout the thesis and proved to be useful tools for finding bugs before integration into a larger system.

Code excerpt 6.1 shows an example of testing the `clamp()` method. The `assertEqual()` function takes the function to be tested and expected return value. After setting this up, the command `nosetests` is written in the terminal. `nosetests` looks for all test cases in a folder and runs them. If one of the tests is unsuccessful, the following output is shown in the terminal:

```
user@hostname ~/\$ nosetests

=============================================================
FAIL: test_clamp (test.TestStringMethods)
-------------------------------------------------------------
Traceback (most recent call last):
  File "test.py", line 8, in test_clamp
    self.assertEqual(clamp(0.1, -1.0, 1.0, 0.25), 0.2)
AssertionError: 0 != 0.2
-------------------------------------------------------------
Ran 3 tests in 0.102s
FAILED (failures=1)
```

```python
import unittest
from topside import clamp


class TestStringMethods(unittest.TestCase):

    def test_clamp(self):
        self.assertEqual(clamp(0.1, -1.0, 1.0, 0.25), 0.0)

    def test_clamp_max(self):
        self.assertEqual(clamp(1.1, -1.0, 1.0, 0.25), 1.0)

    def test_clamp_min(self):
        self.assertEqual(clamp(-1.1, -1.0, 1.0, 0.25), -1.0)
```

Python code excerpt 6.1: Unit-test example

6.2 Integration Testing

Integration testing is testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them [10]. The tool used was Software In The Loop Testing with Gazebo and running ArduSub simulating the Pixhawk FCU. This testing phase revealed several bugs when integrating the different modules together. This was especially beneficial when testing ROS and MAVROS related software. Unit tests are unpractical to perform on these modules due to that the entire control software must run to validate that it works.

6.3 System Testing

[10] defines System Testing as testing conducted on a complete integrated system to evaluate the system's compliance with its specified requirements. Requirements were made to perform the system testing of the ROV, see table 6.1.

Table 6.1: System Testing requirements

Attribute	Description
Pilot joystick input	Pilot input is correctly encoded and decoded. The axes controlled is what the ROV does
Pilot button press	The buttons do the intended action. This includes changing mode, turning on and off lights and setting the gain
Latency	Latency between pilot input and ROV movement should not exceed 1000 ms
Safety features	3 seconds without pilot input disarms the vehicle
	30 seconds without pilot input turns off the lights and returns the ROV to the surface
Robustness	The Downlink performs well in highly reflective environments as well as under optimal conditions.
Data rate	4 bytes per seconds or more
Input device	Validate both Xbox controller and IID as input devices

Most of the system tests were done in a water tank located in the office of Water Linked AS. See figure 6.1. Outdoor testing was done after validating indoor testing, see figure 6.2. Testing indoors gave the opportunity to test in a highly reflective

environment. Having a low packet-loss rate indoors is a good indication that there will be low-packet loss outdoors. One of the restrictions of testing indoors is limited range. Outdoor testing confirmed that the Downlink worked well up to a range of 100 meters.

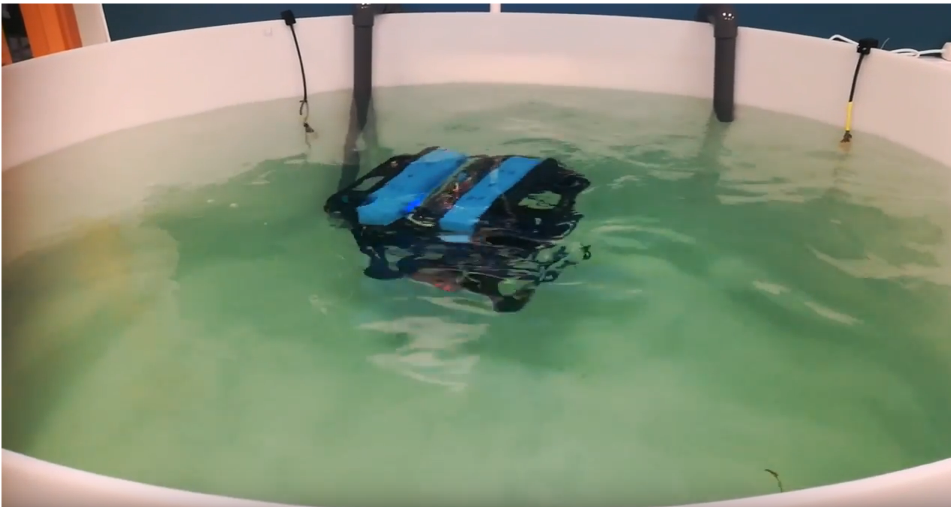

Figure 6.1: Testing in indoor water tank.

Figure 6.2: Testing outdoors at Brattørkaia, Trondheim. Note that a rope is attached to the ROV. This was due to it being the first test done outdoors.

The System Test period lasted for 4 weeks and both the Underwater Modem and ROV software were tested in parallel. The System Tests unveiled many ways to

improve listed below.

- Depth mode was beneficial for controlling the ROV. It was first set as stabilize mode only, but it turned out that the positive buoyancy of the ROV made it difficult to keep it underwater when stationary.
- Gain was adjusted to improve mobility. Indoor testing made it difficult to adjust this setting due to the size of the tank. Outdoor testing were necessary for adjusting the gain parameter.
- The sensitivity of the IID joysticks needed to be scaled down.
- Light strength were adjusted.

The requirements were met during the duration of the System Testing phase. An excerpt of the logged data is shown in appendix B.2.

6.4 Acceptance Testing

According to [10], Acceptance Testing is defined as a formal testing to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system. The acceptance criteria in this case is shown in table 6.2. Note that these requirements are for the Downlink only, the Uplink requirements were handled by Water Linked. The customer refers to the SWARMs officials, who determines whether or not to accept the system or not.

Table 6.2: Acceptance Testing requirements

Attribute	Description
System test requirements	All requirements must be met according to table 6.1.
Integration on-site	Controlling the ROV with the Xbox controller must be proved at the test site. Additionally, the IID is to be controlled from a control room. Communications and networking are done through the SWARMs network.

Chapter 7

Results and the way forward

7.1 Results

The demonstration in the SWARMs research project at Trondheim Biological Station (TBS) 26. June was a success. Along with Water Linked employees, it was demonstrated that the ROV could be controlled in a safe and robust matter according the the acceptance test requirements listed in table 7.1.

Figure 7.1 shows the delay between each received data frame on the OBC during a period of $\approx$ 10 minutes. The logging was done during one of the acceptance tests in the integration week of the SWARMs project. The log clearly shows that data frames are received every 500 ms most of the time. Delay over 500 ms indicates packet loss.

The testing tools used in the thesis also proved to be useful. Indoor testing provided a highly reflective testing environment. Outdoor testing validated the Downlink up to 100 meters. Several challenges were discovered and resolved along the way, suggesting that the testing scheme worked for this kind of project.

Table 7.1: SWARMs demonstration results

Attribute	Description	Result
Pilot joystick input	Pilot input is correctly encoded and decoded	Confirmed
Pilot button press	Change mode, turning on and off lights and set the gain	Confirmed
Safety features	3 seconds without pilot input disarms the vehicle	Confirmed
	ROV returns to the surface after 30 seconds	Confirmed
Robustness	The Downlink performed well	Confirmed
Input device	Xbox controller and IID	Confirmed
Integration on-site	ROV was controlled with an Xbox controller at the test site. IID was controlled from the control room off-site	Confirmed
Range	<= 100 meters	20 meters
Latency	<= 1000 ms	200-1000 ms
Data rate	>= 4 bytes/s	4 bytes/s

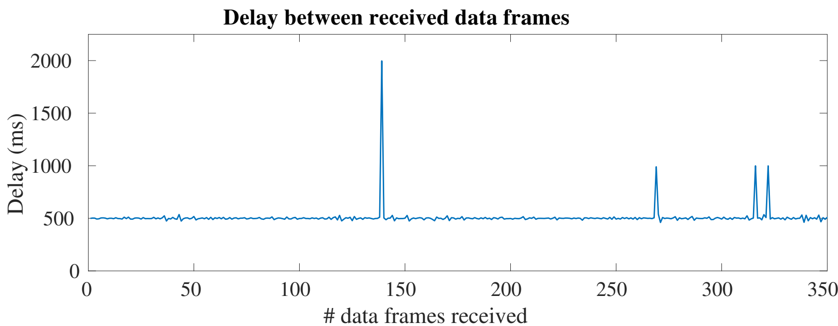

Figure 7.1: Delay between each received data frame on the Onboard Computer. One data frame corresponds to 2 bytes. The logging is done in a time period of ≈ 10 minutes

7.2 Conclusion

One of the main objectives of the work described in this thesis was to demonstrate wireless control of an ROV using Water Linked's Underwater Modem in the SWARMs research project. The ROV was custom built for this purpose by adding to additional thrusters and mounting the Subsea Master payload. Pilot input was given with an Xbox controller as well as the Intuitive Input Device provided by

Inventas AS.

To be able to control the ROV with a low amount of data, the data is encoded on the topside computer and decoded on the onboard computer.

The OBC runs the ROS package MAVROS to interface with the Pixhawk autopilot. Several features such as security, mode, gain and lights is implemented to provide a safe and robust way of controlling the ROV.

Several Software Life Cycle models were explored to find the approach to be used during the development of the software. A testing scheme was applied to the project to ensure that the requirements were met during the SWARMS demonstration.

The SWARMs demonstration was a success and the final product met all of the requirements specified.

7.3 Further work

Define different message types

Today's implementation only transmits pilot inputs to the ROV. An alternative method would be to reserve 2-3 bits in the data frame to act as "message type" bits. Using 3 bits, $2^3 = 8$ different message type could be transmitted. This could include joystick message, button message, GPS coordinates, start / stop video recording etc. The message handler on the OBC must be expanded to parse and decode these messages.

Autonomous missions

Equipping the ROV with Underwater GPS from Water Linked, the position of the ROV can be seen on a topside computer. This position can be sent back to the ROV using the Underwater Modem, giving it both global and local coordinates. Then, MAVROS can be used to facilitate autonomous missions.

AUV conversion

Normally, the Underwater GPS has a topside unit with a baseline of 4 receivers, and the ROV has one locator. Flipping this configuration, having the baseline on the ROV and the locator at topside, the ROV knows its position underwater. This is a good step of the way to convert it into an AUV.

ROV satellite

The BlueRov2 can be used as a communication satellite for other underwater vehicles. Equipping the ROV with a Topside Master could enable communication between several other vehicles under water, provided they are equipped with Subsea Master units. It would be preferable to connect a tether to the BlueRov2, such that all communications can be monitored topside, while the ROV is following the other vehicles at a distance up to 100 meters. This could increase the range of autonomous mission significantly, as well as receiving data from the missions in real-time.

References

[1] Alshamrani Adel and Bahattab Abdullah. “A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model”. In: *IJCSI Int. J. Comput. Sci. Issues* 12.1 (2015).

[2] *Agile Alliance*. `https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/`. [Online; accessed June 2018]. 2018.

[3] “Agile Software Process and its experience”. In: *Proc. 20th Int. Conf. Softw. Eng.* (1998).

[4] Kent Beck. “Embracing Change with Extreme Programming”. In: *Computer* 32.10 (Oct. 1999), pp. 70–77.

[5] Kent Beck et al. *Manifesto for Agile Software Development*. 2001.

[6] T Bhuvaneswari and S Prabaharan. *A Survey on Software Development Life Cycle Models*. Vol. 2. 2013.

[7] Gerald D. Everett and Raymond McLeod Jr. *Software Testing*. Hoboken, New Jersey: John Wiley & Sons, Inc., 2007.

[8] Kevin Forsberg and Harold Mooz. “The Relationship of System Engineering to the Project Cycle”. In: June 1994 (1996).

[9] Rajendra Ganpatrao Sabale and Ar Dani. "Comparative Study of Prototype Model For Software Engineering With System Development Life Cycle". In: *IOSR J. Eng.* (2012).

[10] Anne Geraci et al. *IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries.* 1991.

[11] Tanu Jindal. "Importance of Testing in SDLC". In: *International Journal of Engineering and Applied Computer Science (IJEACS)* (2016).

[12] Naresh Kumar, A. S. Zadgaonkar, and Abhinav Shukla. "Evolving a New Software Development Life Cycle Model SDLC-2013 with Client Satisfaction". In: *Int. J. Soft Comput. Eng.* 3.1 (2013).

[13] Steve R. Palmer and Mac Felsing. *A Practical Guide to Feature-Driven Development.* 1st. Pearson Education, 2001.

[14] William E. Perry. *Effective Methods for Software Testing.* 2006.

[15] Roger S Pressman. *Software Engineering A Practitioner's Approach 7th Ed - Roger S. Pressman.* 2009.

[16] Dr. Winston W. Royce. "Managing the Development of large Software Systems". In: *Ieee Wescon* August (1970).

[17] *SWARMs Research Project.* `http://www.swarms.eu/`. [Online; accessed June 2018]. 2018.

[18] Maneela Tuteja and Gaurav Dubey. "A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models". In: *Int. J. Soft Comput. Eng.* (2012).

[19] *Water Linked AS Homepage.* `https://waterlinked.com/`. [Online; accessed June 2018]. 2018.

Appendices

A.1 List of software resources

- Ardusub
  https://www.ardusub.com/
- ROS
  http://www.ros.org/
- Gazebo
  http://gazebosim.org/
- bluerov_ros_playground
  https://github.com/patrickelectric/bluerov_ros_playground
- MAVROS
  http://wiki.ros.org/mavros
- Xboxdrv
  https://gitlab.com/xboxdrv/xboxdrv
- **Python packages:**
  - asyncore https://docs.python.org/2/library/asyncore.html
  - argparse https://docs.python.org/2/library/argparse.html
  - rospy http://wiki.ros.org/rospy
  - struct https://docs.python.org/2/library/struct.html
  - socket https://docs.python.org/2/library/socket.html
  - nosetests https://www.mankier.com/1/nosetests-2.7
  - unittest https://docs.python.org/2/library/unittest.html

B.2 Subsea data log excerpt

The following list is a log of the data received on the BlueRov2 from the Subsea Master. It logs the data as hexadecimal numbers and the time difference between each data frame. This log is made when giving a variable pilot input, without stop.

Time received:	1528288657
Time diff:	0.5124
Byte #0:	0x7
Byte #1:	0x74
Time received:	1528288657
Time diff:	0.51594
Byte #0:	0x7
Byte #1:	0x74
Time received:	1528288658
Time diff:	0.47263
Byte #0:	0x7
Byte #1:	0x74
Time received:	1528288658
Time diff:	0.4974
Byte #0:	0x7
Byte #1:	0x78
Time received:	1528288659
Time diff:	0.49905
Byte #0:	0x17
Byte #1:	0x78
Time received:	1528288659
Time diff:	0.50271
Byte #0:	0x77
Byte #1:	0x74
Time received:	1528288660
Time diff:	0.49877
Byte #0:	0x77
Byte #1:	0x74
Time received:	1528288660
Time diff:	0.49703
Byte #0:	0x77
Byte #1:	0x74
Time received:	1528288661
Time diff:	0.50267
Byte #0:	0x17
Byte #1:	0x74
Time received:	1528288661
Time diff:	0.49959
Byte #0:	0x6
Byte #1:	0x74