Development and Implementation of Computer-Based Control System for ROV with Experimental Results

Espen Tolpinrud

Master of Science in Engineering and ICT
Submission date: June 2012
Supervisor: Asgeir Johan Sørensen, IMT

Norwegian University of Science and Technology
Department of Marine Technology

NTNU – Trondheim
Norwegian University of
Science and Technology

Master Thesis

Spring 2012

Development and Implementation of Computer-Based Control System for ROV with Experimental Results

Keywords: Computer-Based Control Systems, Software Development
Thread Programming, LabView, HMI

Author:
Espen M. Tolpinrud

Supervisor:
Professor Asgeir J. Sørensen

Co-Advisors:
Martin Ludvigsen &
Fredrik Dukan

Delivered: 10.06.2012

Availability: Open

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MASTER THESIS IN MARINE CYBERNETICS

SPRING 2012

FOR

STUD. TECHN. ESPEN TOLPINRUD

Development and Implementation of Computer-Based Control System for ROV with Experimental Results

Work description
The usage of Remotely Operated Vehicle (ROV) is expanding at a rapid pace. This gives a demand for control systems that is easy to use. The implementation should also be carried out in an intuitive and thought through way such that further development is not hindered. This thesis continuous the work done in the fall 2011 project (Development of computer based control systems with ROV example) where the solutions discussed will be carried out. Implementation will take place in LabView with object oriented perspective. Unit-testing will be conducted after new functionalities have been added to the system, with the exception of user testing of the user interface which must be coordinated with voluntary participants. As the implementation reaches a complete stage Hardware-In-the-Loop (HIL) testing will be used to test communication and cooperation between the different modules in the control system. The finalization of the system will be done by a sea trial.

Scope of work
1. Create a Software Requirement Specification (SRS) for the system.
2. Define constraints and interfaces between program modules.
3. Design a basic and an advanced Graphical User Interface (GUI) for the control system.
4. Implement the requirements specified in the SRS in LabView.
5. Conduct necessary testing for the system.
    - User testing of the GUI.
    - Unit-testing of functions and modules.
    - HIL testing and sea trial.

The report shall be written in English and edited as a research report including literature survey, description of mathematical models, and description of control algorithms, simulation results, model test results, discussion and a conclusion including a proposal for further work. Source code should be provided on a CD with code listing enclosed in appendix. It is supposed that Department of Marine Technology, NTNU, can use the results freely in its research work, unless otherwise agreed upon, by referring to the student's work. The thesis should be submitted in three copies within June 10th.

Advisors: Martin Ludvigsen Fredrik Dukan

Professor Asgeir Sørensen
Supervisor

Preface

This Master thesis was written during the spring semester 2012 at the Norwegian University of Technology and Science (NTNU). The main motivation for writing this thesis is to create a computer-based control system, which is more flexible regarding development and commissioning than its predecessor. It has also been a goal to enlighten the understanding of what a Real-Time system is, and how to implement control systems with those characteristics. There has also been a focus on how techniques commonly used in the computer science industry, can be utilized in the development process of a computer-based control system.

Quotations are in this thesis done in italic font with the reference to the source right before the quotation starts. In definitions and examples, the reference is stated in the definition- and example-header if it is a direct quotation. Notice that quotation marks are used if the quotation is integrated in the text.

Section 4.3 and Section 6.2 have been written in cooperation with the Master student Viktor Berg.

Espen Tolpinrud,
Trondheim, June 10, 2012

Abstract

The demand for ROV operations has increased the last couple of decades. Still, operations are heavily dependent on an experienced ROV pilot, but by developing a sophisticated control system, operations can be performed with higher accuracy than before. In addition the need for constant supervision will be reduced. It is however important to acknowledge the fact that with increased level of automation, fault tolerance must follow in order to maintain the reliability.

This thesis explore the various aspects development of a sophisticated computer-based control system involves. As an overview, this includes planning, implementation, and commissioning, as well as all the steps between. The development process utilized concepts from extreme programming in order to bring structure to the planning and implementation phase.

The new structure of the control system use an Object-Oriented architecture in order to create a generic setup. Commissioning work is then limited to setup of a configuration file, and signal processing between the system and the ROV.

Together with the control system, a user interface has been created. The user interface aims at making ROV operation more user-friendly, while at the same time include the more advanced features. Usability testing have been conducted on both the user interface and the control system.

The control system has been tested in a sea trial with the ROV SF 30k. The results were promising even though it was the first time this type of a control system had been connected to and used on SF 30k.

Acknowledgments

I am grateful to my supervisor Professor Asgeir J. Sørensen for his guidance both in school and in life general. His enthusiasm together with his knowledge and insight in the control industry is truly inspiring. Further, huge thanks to my advisor phd. cand. Fredrik Dukan for always being helpful regarding questions and other problems. Dr. Martin Ludvigsen deserves a thank you for his effort organizing the preparations on the ROV 30k. The crew on R/V Gunnerus also deserves a thank you for their hospitality during sea trials.

Huge gratitude to my classmates in room C1.072, especially Mats Nåvik Hval and Viktor Berg for all the help, technical discussions and inspiration they have given me during the semester. Mr. Berg deserves an extra thank you for helping me out with bringing over modules from the old control system to the new one.

I would also like to thank Robert Nordan, Pål Levold, Anna Makrygiannis and Frøydi Røe Flobakk for participating in usability testing of the control system. Without their help many faults and errors had stayed hidden.

Lastly, I want to thank my family for always being there, and believing in me.

Espen Tolpinrud,
Trondheim, June 10, 2012

Nomenclature

$\boldsymbol{\eta}_d$	Desired position
$\boldsymbol{\eta}_{ref}$	Position reference
$\boldsymbol{e\hat{t}a}$	Estimated position
χ	Number of flop per cycle on a given processor
σ_{flop}	Number of flop per iteration
ϖ	Data size in byte
f_{core}	Clock Frequency for a given processor
$flop$	FLoating-point OPeration
$flops$	FLoating-point Operations Per Second
N	Number of elements to be stored
n_{core}	Number of cores
n_{iter}	Number of iterations
AHRS	Attitude and Heading Reference System
ASCII	American Standard Code Information Interchange
AUR	Applied Underwater Robotics
CPM	Control Plant Model
cRIO	compactRIO
DOF	Degree Of Freedom
DP	Dynamic Positioning
DVL	Doppler Velocity Log
ECF	Explicit Complementary Filter
FPGA	Field-Programmable Gate Array

GUI	Graphical User Interface
HIL	Hardware-In-the-Loop
HiPAP	High Precision Acoustic Position
HMI	Human Machine Interface
HMI Light	Hydrargyrum Medium-arc Iodide Light
I/O	Input/Output
IDE	Integrated Development Environment
LQR	Linear-Quadratic Regulator
LVOOP	LabView Object-Oriented Programming
NED	North-East-Down
OO	Object Oriented
PID	Proportional-Integral-Derivative
PPM	Process Plant Model
ROV	Remotely Operated Vehicle
RT	Real-Time
SIL	Software-In-the-Loop
SRS	Software Requirement Specification
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UTM	Universal Transverse Mercator
UUV	Unmanned Underwater Vehicle
WF	Wave Frequency
WP	WayPoint
XML	eXtended Markup Language

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

In 2010 a DP system[5],[6] was implemented on Minerva. The current control system stem from this, as well as contributions from several Master and PhD. students. There have been no rules or guidelines regarding development, and it seems that every contribution is created with low to none consideration to the rest of the system. The result of this is a surface structure in the block diagram that is barely readable, and time consuming to understand. The size of the block diagram is approximately 19000 $\times$ 4800 pixels. This equals about 10 $\times$ 5 TVs with full HD resolution or 14 $\times$ 7 regular laptop screens.

The way the program has been created gives an impression of fast and ad-hoc solutions. This can be a reflection from lack of predefined interfaces between the modules. When developing large systems it is of major importance to have clearly defined interfaces for the modules, especially if there are more than one developer.

The lack of proper documentation makes it even worse to maintain the control system, hence further development is not advised from the current foundation.

This thesis is written as a contribution to the Applied Underwater Robotics Laboratory[7] (AUR-Lab). The AUR-lab is a reliance from NTNU where several fields of study participate in research and development in the ocean space. This include areas like marine biology and archeology, and robotics. AUR-Lab was officially opened in August 2011. However, the concept had been in use for at least a couple of years already. As of spring 2012 a new addition to the robotics infrastructure were taken in use, the Remotely Operated Vehicle (ROV) 30k together with the control system developed in this thesis.

1.1 Background and Motivation

In the last couple of decades the demand for underwater operations and hence Underwater Vehicles have had a strong increase. The main contributor of the demand is the oil and gas industry as new oil reserves is found on depths outside the reach of a human diver.

1.1.1 Underwater Vehicles

The history of underwater vehicles is quite young compared to the ancient history of surface vessels. The earliest surface vessels is assumed to have been created for about 45 000 years ago, however the earliest documentation of organized ship use is dated to about 4th century BC. Underwater vehicles emerged in the 17th century as a way to explore the sea floor. The military potential was however quickly discovered.

Definition 1.1. *From [8]*

Underwater Vehicle: *small vehicle that is capable of propelling itself beneath the water surface as well as on the water's surface. This includes unmanned underwater vehicles (UUV), remotely operated vehicles (ROV), autonomous underwater vehicles (AUV) and underwater robotic vehicles (URV). Underwater vehicles are used both commercially and by the navy.*

1.1.2 Unmanned Underwater Vehicles

The first UUVs were torpedoes or torpedo shaped, and were not taken into commercial use until the oil and gas industry recognized the potential within these machines. Unmanned Underwater Vehicles are often used as a generic term for AUVs and ROVs.

Autonomous Underwater Vehicles

In 1957 the first AUV was developed at the Applied Physics Laboratory at the University of Washington. This was called Special Purpose Underwater Research Vehicles (SPURV), and was used to study diffusion, acoustic transmission and submarine wakes. Today AUVs are often used in survey missions like mapping the seabed or inspection of laid pipelines.

Remotely Operated Vehicles

The first ROVs were built in the 1950s by the Royal Navy (British Armed Forces), and early development was funded by the US Navy. However, later on it was commercialized as the oil and gas industry had an increasing demand in usage of ROVs. An ROV is usually a highly maneuverable unmanned underwater vehicle connected to a surface vessel through an umbilical. On the surface vessel, an ROV pilot controls the movement of the vehicle through a control system interface. ROVs are today commonly used in the offshore oil and gas industry, as well as inside the field of military and science. ROVs are categorized by their size, weight, ability and power.

Micro small in size and weight, usually below 3 kg.

Mini larger than the micro ROV, usually around 15 kg.

General typically less than 5 hp propulsion. May carry a small manipulator, and max operation depth is less than 1000 meters.

Light Work Class typically less than 50 hp propulsion. Can carry more than one manipulator, and max operation depth is less than 2000 meters.

Heavy Work Class typically less than 220 hp propulsion. Can carry at least two manipulators, and max operation depth is down to 3500 meters.

Trenching/Burial typically more than 200 hp. Have the ability to carry a cable laying sledge. Max operation depth is up down to 6000 meters.

1.1.3 ROV Minerva

Minerva is a SUB-Fighter 7500, and was a specially designed ROV by Sperre A/S in 2003 to fulfill the needs of the research facilities at NTNU. With a maximal depth of 700 meters and 2 brake horse power (bhp) propulsion on each thruster, Minerva is classified as a general ROV. Minerva is being used in a wide range of applications such as biological research, geological surveys and development of control systems.

1.1.4 ROV SF 30k

SF 30k (Shown in Figure 1.1) is, as Minerva, also created by Sperre A/S, and is a SUB-Fighter 30k. It was used by Trondheim Biological Station before NTNU and AUR-Lab took over the operation in December 2010. The propulsion on SF 30k is 4 bhp on each thruster, while the umbilical is 1100 meters long. This results in a classification between Light Work Class and General type of ROV.

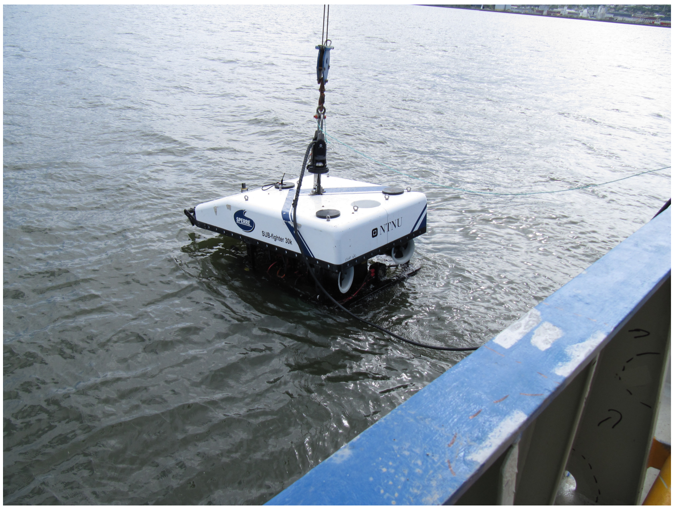


Figure 1.1: Launching of the ROV SF 30k.

1.1.5 Control Systems for Marine Vessels

The first navigation control systems found on ships emerged at the latter half of the 19th century. These were servo-controlled steering systems[9]. Later in 1911 Elmer Sperry introduced the first automated closed loop control for ship steering mechanism.

Modern marine control systems follow the schematic layout shown in Figure 1.2 where the modules are separated in different time domains depending on their update frequency.

The mission management contains planning of the mission, and execution of time consuming applications which are too heavy to be run at real-time. The main focus of this thesis however deal with the real-time control section, including monitoring of the system through an user interface.

1.1.6 Computer-Based Control Systems

In the early 1950s the idea of using computers in control systems emerged. At the beginning applications in missiles and aircrafts were investigated. A couple of years later, in 1956, Thomson Ramo Woodridge (TRW) and Texaco studied the feasibility of using computer control in the process industry, and in 1959 the system went on-line [10]. In the following years and throughout the 1960s, many feasibility studies of computer-based control systems in the process industry were carried out and published in the journal *Control Engineering*.

The first control computers run as an operator guide or set-point control, due to

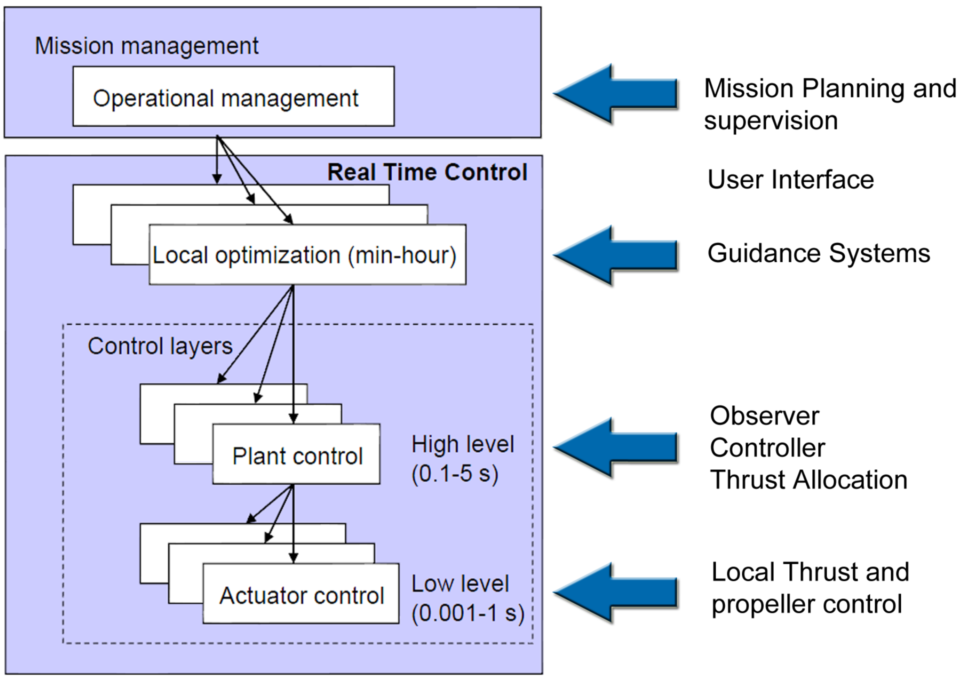


Figure 1.2: Control Structure, modified from [1].

their lack of reliability. This was categorized as supervisory modes of an operation. However, as the computer technology evolved in the 1960s, Direct Digital Control (DDC) emerged. This mode gave the computer direct control of the process. The DDC systems were more flexible than the old analog systems as no rewiring were needed. The additional cost of adding another control loop were also small compared to the old analog systems. The down side of DDC systems were however the initial cost and the new way of designing control systems.

With the introduction of minicomputers in the second half of the 1960s, there was a rapid increase of computer controlled applications. This was possible as the computers became smaller in size, were faster at computations and were more reliable than their predecessors. During the 1970s the microprocessors were introduced as well as the financial costs were heavily reduced, especially during the 1980s. This opened the possibility to use computer-based control systems in a larger scale than before, and today practically all new control systems are based on computer control.

1.1.7 Real Time Systems

For a control engineer it is important to know what a Real-Time (RT) system is, as almost all control systems also are RT systems. A RT system can be defined as:

Definition 1.2. *Real-Time System:*
From [11]

> *Any system in which the time at which output is produced is significant. This is usually because the input correspond to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable timeliness.*

From [12]

> *...any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period.*

From [13]

> *A real-time system is a system that is required to react to stimuli from the environment (including the passage of physical time) within time intervals dictated by the environment.*

The definitions presented above in Definition 1.2 cover a wide range of computer activities. In [14] it is stated that "*the correctness of a real-time system depends not only on the logical result of the computation, but also on the time at which the results are produced*". What this means in practice depends on the application and the criticality level of the timing. In order to distinguish between the different levels of RT, additional terminology must be added. It is common to use soft, firm and hard as description of the RT level requirements. In this thesis Definition 1.3 is proposed for RT system categorization.

Definition 1.3. *Motivated by [14], the following definition for Real-Time system categorization is proposed.*

Soft Real-Time - *Allow a process to exceed a dead line, but the result is degraded. This then has a bad influence on the overall system quality.*

Firm Real-Time - *An occasionally missed dead line is tolerated. However the system quality may be degraded by this. The usefulness of the result after a dead line is said to be zero.*

Hard Real-Time - *No missed deadlines are tolerated. If this occurs it results in a total system failure.*

In order to develop a RT software application, every component needs to be set up for RT usage. Synchronous programming languages have to be used together with a RT network on a RT operating system, especially if the application is said to be mission critical.

1.2 Contributions

The contributions of this thesis is within development, implementation and commissioning of a computer-based control system with all its aspects. The author has in addition helped out in the preparations of the ROV SF 30k. Contributions can be categorized as following:

- DP Control system:
  - Planning of the software architecture based on an Object-Oriented structure.
  - Implementation of the planned software architecture in LabView with both HIL-testing and sea trial as verification.
  - Creating an Application Programming Interface for the control system. (Appendix A)
- Graphical User Interface:
  - Implementation of a completely independent user interface to be used with the control system.
  - Conduct usability testing of the user interface to verify user-friendliness.

Further, definitions of the following topics are proposed.

- RT system categorization is proposed (Definition 1.3).
- Class terms used in Object-Oriented programming (Definition 2.1).
- Access Scope used in Object-Oriented programming (Definition 2.2).
- Event as used in Event-driven programming (Definition 2.3).
- Deadlock (Definition 2.15).
- Livelock (Definition 2.17).

Source code for the project can be found in the repository folder at: `https://lerke.felles.ntnu.no:8443/svn/AURcontrolsystem`. A Subversion (SVN) tool must be used in order to access the files. The folder also has limited access, and clearance must be given by the system administrators. The repository location is in the AUR-folder on the Lerke-server, `\\lerke.felles.ntnu.no\AUR`.

1.3 Outline of Thesis

Chapter 2 presents how software can be developed in a structured manner. An introduction to the extreme programming technique, Scrum, is given, as well as outlining of software testing, especially on computer-based control systems. Concepts of Object-Oriented programming and Event-driven programming are described, as well as programming techniques used in development of RT systems.

Chapter 3 outline the Software Requirement Specification used in the development process. This chapter can be treated like an own document.

Chapter 4 explains how the control system, Njord, is build up. This include an overall layout, class relations, communication interface and fault tolerance. Further, the simulation testing is discussed.

Chapter 5 contains the concept of the user interface, Frigg. Documentation of the different panels and their functionalities are explained. The results from usability testing is also outlined.

Chapter 6 describe the preparations and the execution of the sea trial. Results from the first DP operation is presented.

Chapter 2

Software Development

This chapter aims at giving the reader a proper foundation of understanding on how a large computer-based control system can be implemented by using different tools and procedures. In Section 2.1 the planning and execution phase of a software project is outlined. Concepts when designing a user interface is explained in Section 2.2. Section 2.3 gives a brief introduction on software testing. The following section, Section 2.4, introduce the aspect of fault tolerant systems. Lastly, in Section 2.5, introduces the reader to programming techniques commonly used in Real-Time applications.

2.1 Planning and Execution of Software Development

2.1.1 Scrum

There are several different ways of developing computer programs. In the late 1990s and early 2000 extreme programming gained popularity. Extreme programming introduced an agile working environment which lead to large flexibility and also more efficient working. A branch inside of extreme programming is scrum[15],[16]. This is a methodology embraced by many software developers, but also in corporate settings, scrum has become popular.

Scrum can be difficult in the beginning as the team members are unaccustomed to the changes introduced by this procedure. Other factors can be that the project is complex such that there is a need for more advanced techniques in order to reduce the complexity level and make it manageable. It is important that the developing environment is set up for scrum as well.

In scrum there are several terms and expressions defining the different parts of the procedure. The most important are described in this section. For further reading, see [15] and [16].

Product Backlog is a high-level list containing what to be developed during the process and is maintained during the project. Each entry in this backlog has a broad description of the functionality prioritized by business value and is managed by the product owner. During the sprint planning these entries are given a development effort by the team. The sorting of the list is then dependent on business value, and if there are two entries with same value the one with lower development phase is prioritized as this will give a larger increase on the Return-Of-Investment (ROI).

Sprint Backlog is a list of the tasks to be performed in the sprint. Each task should give a work load between four to sixteen hours. The details in the list is sufficient such that everyone in the team knows what to do. It is recommended that the tasks is not to be assigned to, but rather picked by the team members according to what their skill set are. This way of choosing work tasks encourage self-organization in the team.

User stories are one or more sentences that capture the essence of what the user want to achieve. They are created by the product owner or in some cases by the customer, and are their way of influencing the development process. A user story is usually on the following form:

As a <role>, I want <goal/desire> so that <benefit>

Sprints are the interval of time when implementation takes place. These periods usually last between one to four weeks. During a sprint the team works with the project in an intensive manner.

When working with scrum there are three characteristic roles. These are:

Scrum Master - This person is responsible for the scrum process, and makes sure it is progress in the project. The Scrum Master also make sure the team stay focused at the task at hand and achieve the sprint goals.

Product Owner - Sometimes called the "voice of the customers". The product owner ensure that the team deliver value to the business. The person also create user stories to be used in the product backlog.

Team - A group of five to nine people usually with cross-functional skills. Their task is to deliver the product. The team is preferably self-organized and self led, but also with use of team management policies.

In addition to these roles, there are:

Stakeholders - Also called users or customers. They are the reason for the project and whom the end result will benefit. Stakeholders are only active in the scrum process during sprint review meetings.

Managers - People responsible for setting up the development environment.

An important part of scrum is “scrum-meetings”. These are divided into several different types:

Daily Scrum - A short meeting, often limited to 15 minutes where every team member answer the following questions:

- What have you done since yesterday?
- What are you planning to do today?
- Any impediments or stumbling blocks?

This meeting should also occur at the same time and place every day.

Scrum of Scrums - A meeting at the end of the day that focus on integration and areas with overlap in the work. A designated person from each team is attending the meeting such that every team is represented. The Scrum of scrums meeting follow the same procedure as Daily Scrum described above, but the following questions are also included:

- What has your team done since we last met?
- What will your team do before we meet again?
- Is anything slowing your team down or getting in their way?
- Are you about to put something in another team’s way?

Sprint planning meetings - This is a meeting that is held in the beginning of every sprint cycle. Here the purpose is to prepare the “Sprint Backlog” and identify amount of work to be done during the sprint. The meeting has a time limit of eight hours, where the four first is allocated to product owner and the team for a dialog for prioritizing the product backlog. The next four hours is only for the team where they discuss a plan for the sprint. This last part results in a sprint backlog.

Sprint review meetings - This is one of two meetings to be hold at the end of a sprint. The meeting consist of a discussion of what was completed and not. A presentation for the stakeholders about the achieved goals is also scheduled. This type of meeting has a time limit on four hours.

Sprint retrospective - This is the second meeting in the end phase of a sprint. All the team members reflect on the past sprint and try to come up with improvements to be implemented in the work strategy immediately. The questions to answer is:

- What went well during the sprint?
- What could be improved in the next sprint?

This meeting has a three hour time limit.

Notice that the time limits given above is for sprint lengths of four weeks. If the sprint is shorter, the meeting lengths should be scaled accordingly. I.e. if the sprint is a two week process, the meetings should be half of what it is for four weeks.

2.1.2 Requirement Analysis

When developing software it is important to know what the application is intended for. In order to to so a requirement analysis is needed. Requirements express *what* the application is meant to do, not *how* to implement it. [2] gives the following example for an accounting application:

Correct - The system shall allow the user to access his account balance.

Incorrect - Customers' account balances will be stored in a table called "balance" in an Access database.

It should however be noted that the incorrect requirement given above as a matter of fact can be correct if the user specifically asks for, in this case, that account balances is stored in an Access database with the name given "balance". This is then the exception of the rule.

Requirements can be divided into layers of increasing details. A requirement given with low details in a higher level, can in a lower level be the source for one or more specific requirements. The total output from this analysis is then called Software Requirement Specification (SRS) or just Requirement Specification. Requirement analysis can be divided into two groups, Customer Requirements or C-Requirements and Developer Requirements or D-Requirements. C-Requirements are overall requirements describing the purpose of the application. They are created as the name indicates by the customers, and is expressed through a language clear to them. On the other hand the D-requirements are more specific and structural in their formulation. These requirements are created by the developers. C-Requirements is then characterized as fist level, and D-requirements is second level, where the different levels are as described in the previous paragraph.

In order to make full use of SRS they have to be written down as it works like a road map for the developers. Without a reliable source for the requirements it is hard to know if a goal is accomplished, perform proper tests and also satisfy the customers. Another important aspect is detection of defect requirements. A defect requirement can be costly if not noticed until the end of the project. It is therefore advised to invest time in finding such defects at an early stage of the development process, and with use of the written SRS this can be done in a structural manner. However, it often happen that the customers do not know what they want in the start phase, which makes early defect detection impossible. It is then important to have a close cooperative relation with the customers together with structured iterative working loops.

The template from [17] is given in Appendix B, Definition B.1.

2.1.3 Object-Oriented Programming and Architecture

The main advantage with the use of Object-Oriented (OO) Architecture is the intuitive way of modeling the different components in a system. The main building block in OO Architecture are classes. An instance of a class is called object, hence the name Object-Orientation. A class must always have one default constructor, but it is often that classes have more than one constructor. Constructors should be designed such that they initialize a new instance of the class depending on the input arguments in the constructor.

Definition 2.1. *Suggesting the following definition:*

Class *is a data type that have both state and behavior. These qualities are from Data Members and Methods.*

Data Members *enables an instance of a class to maintain state. A Data Member is a data field type.*

Methods *enables an instance of a class to have behavior, sometimes depending on the state. Methods are functions and subroutines specifically made for the class or an instance of the class.*

OO programming support encapsulation of data members. This is used to protect the values from volatile access, and limitation of property rights. Usually, write- or set-methods are private for the class. "Private" is a keyword used to define the access scope of data members and methods. The different access scopes are defined in Definition 2.2.

Definition 2.2. *Proposing the following definition for Access Scope:*

Public *access scope allows access for everyone.*

Protected *access scope allows access for the class and all sub classes of the particular class.*

Private *access scope allows only the class itself to have access.*

Community or friend *access scope allows a function outside the class to have access to private and protected data members and methods.*

A huge advantage with OO programming is the class inheritance feature. Inheritance is, as the name indicates, properties and qualities passed down to one or more child classes. A widely used notation when talking about inheritance is parent and child, where a child inherit from a parent.

A common use of inheritance is by utilizing abstract classes. These classes then works as a template for the common properties among the child classes. Methods can be specified to be overridden in child classes. This means that a method body is redefined, but the method interface stays the same.

Alongside inheritance, polymorphism is achieved. By this it is meant that an object can act as an instance of another class as long as it is inside of the inheritance chain. This is commonly used when calling overridden methods, especially if the caller may change.

In OO programming methods can be specified to be either dynamic or static dispatched. A dynamic dispatched method is related to the calling instance and its data members. On the other side, a static dispatched method is a generic method which yields for all instances of the class. This means that a static dispatched method will give the same output from a given input, regardless the calling object, while the dynamic dispatched often is dependent on data members, which in turn are related to the calling object.

2.1.4 Event-Driven Programming and Architecture

Event-driven programming is usually used as a supplementary programming paradigm where its functionality is integrated with the main concept. The most common form of this is exception handling, but also when creating GUI, the event-driven paradigm is widely used. The principle consists of listeners and events. When a listener detects a change in condition, e.g. a button is pressed, an event is thrown, and a predefined piece of code is executed to match the desired action from the user. This gives a dynamic execution where the executed code is in accordance to what is desired.

Definition 2.3. *Proposing the definition:*
An ***Event*** *is an action, for which a given reaction may or may not be created. Further, an event must be added to a listener's event-list in order to be detected.*

From Definition 2.3 it is said that all possible events may not be handled by the program. The reason for this is that it is neither necessary nor appropriate to do so. However, when utilizing event-driven programming as error handling, it is a goal to include restoration code for everything that can go wrong, even though this is most unlikely to achieve.

When analyzing event-driven programs there are no one-way stream in the flow, hence the implementation can look messy and hard to understand. It is recommended to look at the independent components and their invocation code rather than the whole system at once.

2.2 Designing User Interface

Development of a Human Machine Interface (HMI), or Graphical User Interface (GUI), is slightly different from other programs. Input from a human user may not always follow the guidelines of what is intended. Other differences revolve around the external look of the user interface.

In [18] a list of eight guidelines has been assembled. These principles derive from experience and are redefined over several decades. It should however be noted that a complete and universal list cannot be made as validation and tuning is required for each design domain, still it is a good starting point.

Definition 2.4. *The Eight Rules of Interface Design as stated in[18]:*

1. ***Strive for consistency.***
2. ***Cater to universal usability***
3. ***Offer informative feedback***
4. ***Design dialogs to yield closure***
5. ***Prevent errors***
6. ***Permit easy reversal of actions***
7. ***Support internal locus of control***
8. ***Reduce short-term memory load***

Especially point 5, Prevent errors, in Definition 2.4 must be emphasized. To reduce the possibility of the user creating a system error is of major importance. This include limiting button access, checking of input fields and such.

2.3 Software Testing

Testing is the only way to ensure system behavior according to specifications. It is therefore good practice to allocate time in the development phase to conduct various tests on parts or the whole system. It is important to derive a test plan and follow it. This increases efficiency and time consumption is kept at a minimum.

Definition 2.5. *In [2] it is stated:*

Goal of testing - *Maximize the number and severity of defects found per dollar spent. Thus: test early*

Limits of testing - *Testing can only determine the presence of defects, never their absence. Use proofs of correctness to establish "absence"*

2.3.1 Testing of Control Systems

A common practice inside software engineering is to use unit testing.

Unit testing is the earliest types of testing available when developing a program, and is a structural testing procedure. A unit is defined as the smallest testable parts in the program. This can be a function, or an interface such as a class.

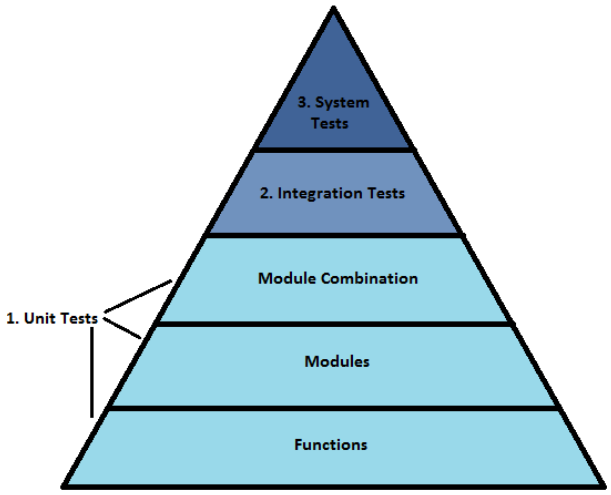


Figure 2.1: Schematic Illustration of unit testing as presented in [2].

The testing procedure aims at verifying each part to ensure correct behavior. In Figure 2.1, a setup for a test schedule is shown. By utilizing this testing procedure, errors introduced by the programmer are put to a minimum at an early stage of the development process. In order to reduce faults and errors caused by communication between modules a full system test is needed. HIL-testing has become more popular in the control environment the later years. Despite the name "Hardware-In-the-Loop", HIL-testing is a computer-based software testing procedure. In order to perform HIL testing an accurate simulator for the vessel is needed. The simulator should be able to simulate dynamic response of the vessel, thruster and propulsion systems, sensors, position reference systems, power generation distribution, main consumers and other relevant equipment[19]. The control system and the simulator is then connected to each other through the control system interface. The reason for this is to create a realistic testing environment and also ensure proper information exchange between the control system and the plant. The control system interface is often a network connection or a bus interface.

HIL-testing is used to detect faults, errors and failures in the controller software. Verification of the control system can also be done through this. HIL-testing can contribute to perform virtual sea trials even before the vessel itself is built. This leads to a more finalized version of the program to be implemented on the vessel, which again gives shorter commissioning, integration and tuning times.

The main down side with HIL-testing is the constraint of Real-Time clock speed. A new way of performing testing could then be through Software-In-the-Loop (SIL) testing as described in [20]. This involves that the control system and the simulation system runs at the same clock speed, and is then not restricted to real time clock speed as the HIL testing is. This gives the ability to execute more tests in shorter time. A critical requirement for the SIL testing is though that there are no user inputs during the simulation. To overcome this, a predefined configuration file can be created with programmed user inputs. It is however important to acknowledge that SIL testing cannot replace HIL testing, but the two can be used in different testing procedures to utilize their strong sides. SIL testing can have faster simulation time compared to HIL testing and can be especially suited for long time simulations. On the other hand HIL testing gives the ability to perform on-line tuning, as well as real time results. HIL testing should then be used especially in finalization tests.

When testing it is advised that another person performs them. By doing so the risk of hiding weak points is reduced. Also, during HIL or system tests, persons without any experience with the program should have a go at it. This will often bring out unknown errors due to unforeseen scenarios by the engineer.

2.3.2 Testing of User Interfaces

User interfaces is usually tested in a bit different manner. There is no way of telling a good interface from another until it has been used by the customer since it is the customer that decides whether or not the interface fulfill their expectations and needs. Often an expert is called in to perform a review of the interface. These Expert Reviews often aim at locating faults and weaknesses in the interface.

There are different types of expert reviews which all have their own characteristics.

Definition 2.6. *Expert Reviews as presented in [18]:*

- ***Heuristic evaluation.*** *The expert reviewers critique an interface to determine conformance with a short list of design heuristics, such as the Eight Golden Rules (Definition 2.4). It makes an enormous difference if the experts are familiar with the rules and are able to interpret and apply them.*
- ***Guidelines review.*** *The interface is checked for conformance with the organizational or other guidelines document. Because guidelines documents may contain a thousand items or more, it may take the expert reviewers some time to absorb them and days or weeks to review a large interface.*
- ***Consistency inspection****. The experts verify consistency across a family of interfaces, checking the terminology, fonts, color schemes, layout, input and output formats, and so on within the interfaces as well as in the documentation and online help.*
- ***Cognitive walkthrough****. The experts simulate users walking through the interface to carry out typical tasks. High-frequency tasks are a starting point, but rare critical tasks, such as error recovery, also should be walked through.*

- ***Metaphors of human thinking (MOT)****. The experts conduct an inspection that focuses on how users think when interacting with an interface. They consider metaphors for five aspects of human thinking: habit, the stream of thoughts, awareness and associations, the relation between utterances and thoughts, and knowing.*
- ***Formal usability inspection****. The experts host a courtroom-style meeting, with a moderator or judge, to present the interface and to discuss its merits and weaknesses. Design-team members may rebut the evidences about problems in an adversarial format.*

Another common practice when developing user interfaces is to include the customer at an early stage. This makes it possible to define the customers desires and constraints, and then design the user interface. Testing of the suggested solution is an important part of this methodology. One or more end-users are asked to participate in testing of the interface. In the testing procedure a prototype or working prototype is presented to the test subject. The test subject is then given one or more tasks and will try to perform these task with the prototype. The prototype can be a schematic illustration, or a small program. However, it is important that the prototype is a good reflection of the real interface.

This type of testing is often called user testing or usability testing. The testing procedure is summarized in Definition 2.7

Definition 2.7. *translated from Norwegian in [21]:*

- *Create realistic tasks for the user and find other users with the same knowledge level as the expected end-users. Do not use persons familiar with the product.*
- *Describe the purpose with the test. Let them know that you are testing the product, not them. They can also quit at any time. Ensure them that it does not matter if they are doing something wrong, as it will enlighten problems in the design.*
- *Give a brief presentation about the equipment*
- *Explain how to think loud, and ask the test subjects to verbalize what they are thinking as they use the product.*
- *Describe the task, and introduce the product to the users.*
- *Ask if there are any questions before they start the testing. Afterwards start the observation.*
- *Finish the observation. Tell the user what you found and answer their questions.*
- *Use the results.*

Depending on where the development process is, different tests are conducted.

Exploration test is early testing of the planned product or user interface. The purpose of this test is to check the concept or idea behind the solution. The test products are usually images or other prototypes that show the main structure of the whole solution.

Evaluation test are tests conducted during the development phase. Here the motive is to test whether or not the solution works in practice. The tests are conducted on working prototypes with at least parts of the end functionalities implemented.

Acceptance test is the finalization test. The product is set up against several predefined requirements about user friendliness. A common practice is to ask the test person some follow-up questions right after the test is finished. This is then used to determine the users experience of the product.

Figure 2.2 and 2.3 are taken and translated from [3]. They show the flow of work when designing and testing a user interface.

2.4 Fault Tolerance in Software

2.4.1 Introduction to Fault Tolerance

When performing safety critical operations it is of great importance that the applications have some sort of fault tolerance. The definitions of fault, error, and failure are as described in Definition 2.8

Definition 2.8. *The following descriptions are based on [22] and [4]:*

Fault - *a mechanical or algorithmic cause of an error.*

Error - *designates the part of the state that is incorrect.*

Failure - *when the behavior of a system deviates from that which it is specified for.*

In control theory a fault is separated into the three following categories.

Definition 2.9. *[4] states:*

Plant faults - *Such faults change the dynamical I/O properties of the system.*

Sensor faults - *The plant properties are not affected, but the sensor readings have substantial errors.*

Actuator faults - *The plant properties are not affected, but the influence of the controller on the plant is interrupted or modified.*

When talking about fault tolerance there are some terms used to describe the performance of the system.

Definition 2.10. *From [4] it is given that:*

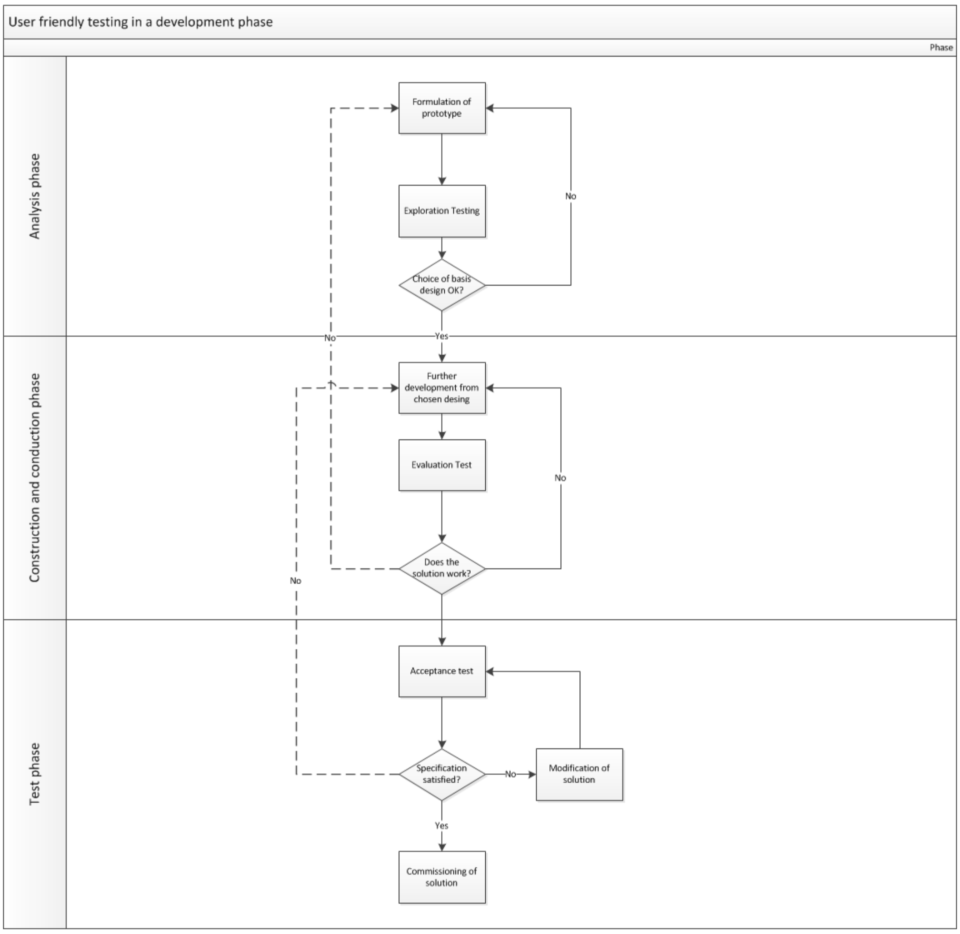


Figure 2.2: User friendly testing in development phase, translated from [3].

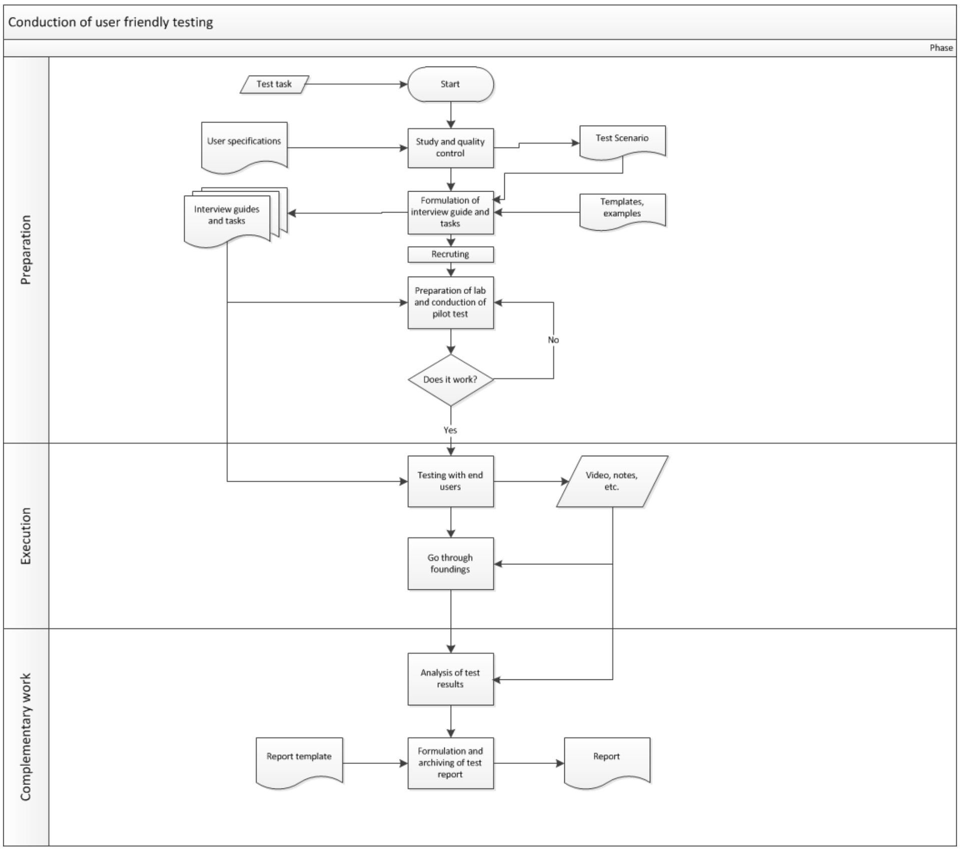


Figure 2.3: Conduction of user friendly testing, translated from [3].

Safety *describes the absence of danger. A safety system is a part of the control equipment that protects a technological system from permanent damage. It enables a controlled shut-down, which brings the technological process into a safe state.*

Reliability *is the probability that the system performs its intended function for a specified period of time under normal conditions.*

Availability *is the probability of a system to be operational when needed. Contrary to reliability it also depends on the maintenance policies, which are applied to the system components.*

Dependability *lumps together the three properties of reliability, availability and safety. A dependable system is a fail-safe system with high availability and reliability.*

Fault tolerant systems are divided into different levels as shown in Definition 2.11.

Definition 2.11. *From [14] but also described in [4]:*

Full fault tolerance[14] **or fail-operational**[4] - *the system continues to operate in the presence of faults, albeit for a limited period with no significant loss of functionality of performance.*

Graceful degradation (or fail soft)[14] **or fail-graceful**[4] - *the system continues to operate in the presence of errors, accepting a partial degradation of functionality or performance during recovery or repair.*

Fail safe[14],[4] - *the system maintains its integrity while accepting a temporary halt in its operation.*

Which level of fault tolerance to implement, depends on the application. All safety-critical systems require in theory full fault tolerance. However in practice this is often not achievable, instead systems that can suffer from physical damage have several layers of graceful degradation. This can for instance be found in combat aircrafts. On the other hand a safety system only needs to be fail safe. This will ensure a minimizing of environmental damage, as well as the vessel is protected from damaging itself.

Inside the field of Dynamic Positioning (DP) there are several strict requirements for fault tolerance. This yields especially for ships and vessels operating in safety critical regions, such as around drilling platforms. It is of major importance that these systems continue to be functional, even in the presence of one or several errors. When implementing fault tolerance in a system, a requirement is to have feedback on the sensors. The purpose with fault tolerance is to locate the fault or error, and reduce it before the system reaches a failure mode. Without feedback this is almost impossible, since there is no certain way of saying where the error is.

By identifying the faults that can occur in the system, it is possible to create methods to reduce their impact. But even with unlimited resources, it is almost impossible to find every fault that can happen in a modern system. It is therefore equally important to take into account the unknown faults and errors as well, and have a way to recover from them. An alternative inside the field of control theory is a comparison check between simulated and measured values. If the values deviate from each other, this indicates that a fault is present. The problem with this method is that not all signals or states that can be measured due to financial costs or complex solutions. A common practice is then to use a plant model, typically a Process Plant Model (PPM) for the simulation, and a Control Plant Model (CPM) for the controller. The movement of the vessel is then compared to the one given from the PPM. Again, if it is deviation in the values, a fault is most likely the cause. However, it is now hard to locate where the fault occurred.

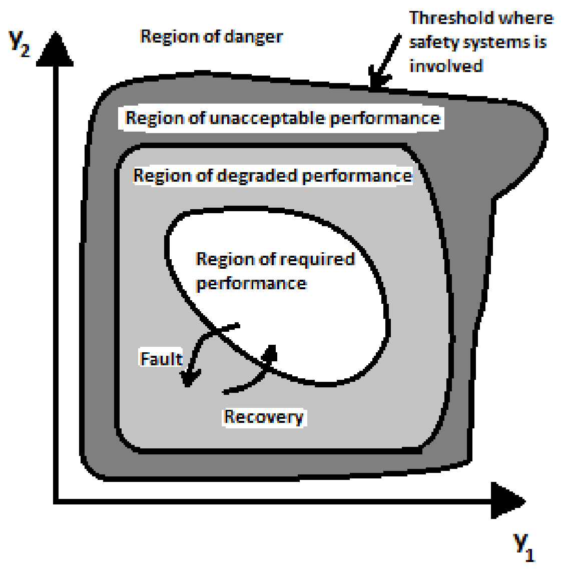

Y2
Region of danger
Threshold where safety systems is involved
Region of unacceptable performance
Region of degraded performance
Region of required performance
Fault
Recovery
Y1

Figure 2.4: Regions of required and degraded performance from [4].

In Figure 2.4, an illustration of fault detection and fault tolerance with comparison is shown. The figure shows where a given output indicates a fault in the system, as the output pair y_1 and y_2 should be inside the region of required performance. By looking at the faults defined in Definition 2.11, Figure 2.4 typically yields for plant faults.

The error recovery process can mainly be done in two different ways, forward error recovery and backward error recovery. In a Fail Safe system, backward error recovery is sufficient where the system will be put in the last safe state before continuing its operation. In forward error recovery the system accepts the presence of an error and tries to cope with it by isolating it while other modules covers the functionality of the faulty component. In this way, the operation continues, as if nothing was wrong, or with some degraded performance.

2.4.2 Redundancy

There are several meanings of redundancy. From a programmers point of view, redundancy is to be avoided. The reason for this is that there are usually no upsides to have the same piece of code several places in the program. When taking the control engineers point of view, redundancy can be used in a good way to increase the availability of the complete system by having multiplicity of components. Components is in this context defined as the different parts and modules of the complete system, such as a sensor, the whole computer-based control system or a thruster. The difference between these two ways of looking at redundancy is the level of scope. The programmer works at a much more detailed and low level scope than the control engineer. Redundancy is however a topic that needs carefully analysis and caution in order to be used as its full.

One obvious way to utilize redundancy in software is to have two instances of a program running in parallel with a switching mechanism. This method is also called process pairs. The first program to start becomes the master, while the other are slaves. Only the master sends information to the vessel, but all the programs do the calculations. The master program also broadcast a status message to the other participants. If this message stops, the next program in line takes the masters place. This is a kind of dynamic redundancy, since the number of running programs can be changed without the need of other specifications. On the downside, a design error in the program will in this case not be correctly handled, unless there are some sophisticated fault tolerance implemented as well.

Another way of redundancy in the software is inspired from the hardware redundancy such as Triple Modular Redundancy (TMR) and N Modular Redundancy (NMR) as described in [14]. The system then has several versions of the program running concurrent to each other where everyone does the calculations needed. A making or voting process determines the correct output to be sent to the vessel. These programs should be developed independently and with different programming languages, or at least with different compilers on different processors. The reason for this is to reduce the chance of error redundancy by the program and processor. This is a technique used by the airplane business. However, the downside with this method is the financial costs.

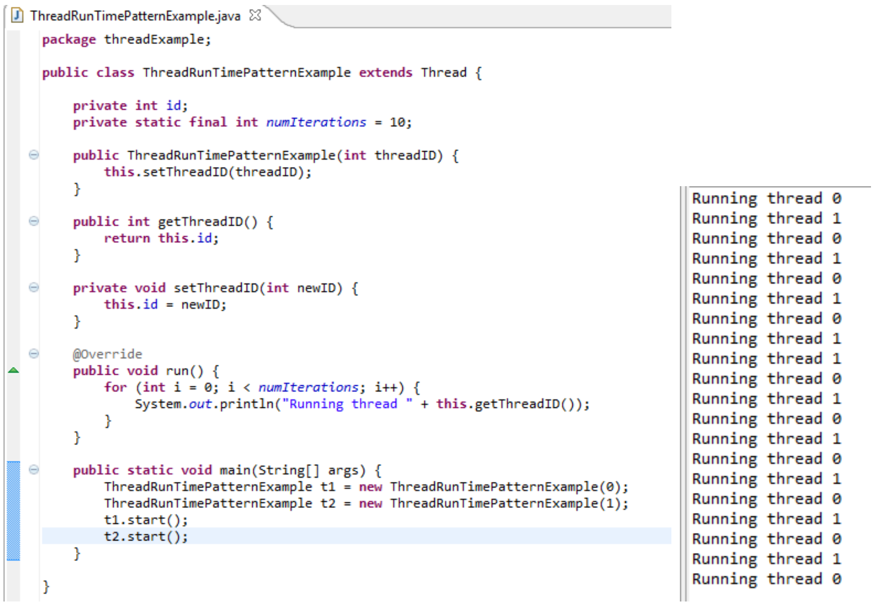
The common factor with these approaches it that the program is the redundant component, not the code inside of the program.

Redundancy can increase the reliability for the system if caution is taken. The program developed needs thorough testing to get rid of design errors. This is important, since the programs run concurrently with each other, and the errors can propagate faster and create dangerous situations. It is therefore advised to keep the redundancy in software to a minimum.

2.5 Advanced Programming Techniques

This chapter is meant to give an introduction to advanced programming techniques commonly used when developing real time and concurrent systems. The topics include synchronization, thread scheduling and network programming.

When developing systems where RT demands are crucial, programs consisting of threads are often used. The reason for this is that different and independent processes can be encapsulated in threads that will run concurrently. However, the pitfall with threads is that the execution order of things is not set in stone. In order to clarify the term parallel or concurrent execution, the example in Figure 2.5 is used. The figure shows the code and the output for a small Java program created to illustrate the problem.



```java
package threadExample;

public class ThreadRunTimePatternExample extends Thread {

    private int id;
    private static final int numIterations = 10;

    public ThreadRunTimePatternExample(int threadID) {
        this.setThreadID(threadID);
    }

    public int getThreadID() {
        return this.id;
    }

    private void setThreadID(int newID) {
        this.id = newID;
    }

    @Override
    public void run() {
        for (int i = 0; i < numIterations; i++) {
            System.out.println("Running thread " + this.getThreadID());
        }
    }

    public static void main(String[] args) {
        ThreadRunTimePatternExample t1 = new ThreadRunTimePatternExample(0);
        ThreadRunTimePatternExample t2 = new ThreadRunTimePatternExample(1);
        t1.start();
        t2.start();
    }

}
```

```
Running thread 0
Running thread 1
Running thread 0
Running thread 1
Running thread 0
Running thread 1
Running thread 0
Running thread 1
Running thread 1
Running thread 0
Running thread 1
Running thread 0
Running thread 1
Running thread 0
Running thread 1
Running thread 0
Running thread 1
Running thread 0
Running thread 1
Running thread 0
```

Figure 2.5: Example of running pattern with threads.

The OS scheduler determines which process to be run in what order. Often for small tasks such as in the example the scheduler will switch back and forth between the processes on each step. However, as seen in the output this is not always the case, even though the action in each thread is the same.

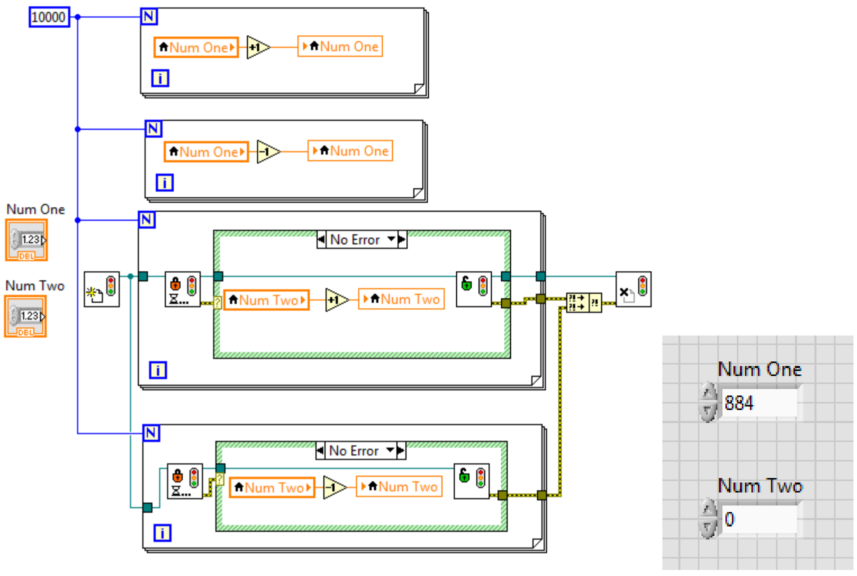


Figure 2.6: Illustrative example of Race Condition and avoidance.

2.5.1 Race Condition

To make matters worse, thread programming also introduce Race condition. The LabView block diagram shown in Figure 2.6 illustrates the danger of Race Condition in a system. Each thread pair share a common variable, where one thread increments the value of the variable and the other decrements its value. The two upper loops use global/local variables directly without any synchronization. The end result can be seen in the "Num One" field on the right. When running the program the output is never known, but in this case it is in the bounded area $< -10000, 10000 >$. The two loops on the lower half in Figure 2.6 use semaphores to solve the synchronization problem.

Definition 2.12. *From [14]:*
Data Race Condition *is a fault in the design of the interactions between two or more tasks whereby the result is unexpected and critically dependent on the sequence or timing of accesses to shared data.*

With Race Conditions present in the system, correct behavior can neither be guaranteed nor proven. Race Conditions can only arise when threads or processes access the same shared variable. Hence as a rule of thumb, shared or global variables are to be kept at a minimum when creating code.

2.5.2 Semaphores

From time to time the use of shared variables cannot be avoided in a RT system. In order to avoid the occurrence of a race condition, mutual exclusion on the variable must be created. This can be done with Semaphores.

Definition 2.13. *From [23]:*
*A **semaphore** is like an integer, with three differences:*

1. *When you create the semaphore you can initialize its value to any integer, but after that the only operations you are allowed to perform are increment (increase by one) and decrement (decrease by one). You cannot read the current value of the semaphore.*
2. *When a thread decrements the semaphore, if the result is negative, the thread blocks itself and cannot continue until another thread increments the semaphore.*
3. *When a thread increments the semaphore, if there are other threads waiting, one of the waiting threads gets unblocked.*

Definition 2.14. *In this thesis the following definition of Mutual Exclusion is proposed: Mutual Exclusion (mutex) is a semaphore mechanism used to control concurrent access to shared resources, and limits the access to one (or a limited number of) process(es) at a time in order to avoid Race Condition.*

As the Definition 2.14 states, the mutex ensure that only one process or thread have rights to a shared variable at any time. If a variable is locked when a thread wants to access it, the thread has to wait for the variable to become available. Caution must be taken when using mutex, as dead locks and starvation can occur. By using the example, illustrated in Example 2.1, of dining philosophers these problems can be enlightened.

Example 2.1. *In [24] dining philosopher problem is stated as:*

In ancient times, a wealthy philanthropist endowed a College to accommodate five eminent philosophers. Each philosopher had a room in which he could engage in his professional activity of thinking; there was also a common dining room, furnished with a circular table, surrounded by five chairs, each labeled by the name of the philosopher who was to sit in it. The names of the philosophers were $PHIL_0$, $PHIL_1$, $PHIL_2$, $PHIL_3$, $PHIL_4$, and they were disposed in this order anticlockwise around the table. To the left of each philosopher there was laid a golden fork, and in the center stood a large bowl of spaghetti, which was constantly replenished.

A philosopher was expected to spend most of his time thinking; but when he felt hungry, he went to the dining room, sat down in his own chair, picked up his own fork on his left, and plunged it into the spaghetti. But such is the tangled nature of spaghetti that a second fork is required to carry it to the mouth. The philosopher *therefore had also to pick up the fork on his right. When he was finished he would put down both his forks, get up from his chair, and continue thinking. Of course, a fork can be used by only one philosopher at a time. If the other philosopher wants it, he just has to wait until the fork is available again.*

When a philosopher sits down at the table and attempts to take a fork it can be said that a mutex is invoked. If the fork is available the philosopher gets it. At first this seems like an okay solution. But what happens if all philosophers starts eating at the same time, and everybody is grabbing the fork at their immediate left? The philosophers now needs a second fork in order to eat, however, all forks are in use and none of the philosophers have two forks. They are now in a dead lock situation!

Definition 2.15. *Proposing the following definition of Deadlock:*
Deadlock *is a state where all processes in a system are blocked.*
Definition 2.16. *In [25] four conditions are listed as necessary and sufficient for the occurrence of Deadlocks.*

Serially reusable resources: *the processes involved share resources which they use under mutual exclusion.*

Incremental acquisition: *processes hold on to resources already allocated to them while waiting to acquire additional resources.*

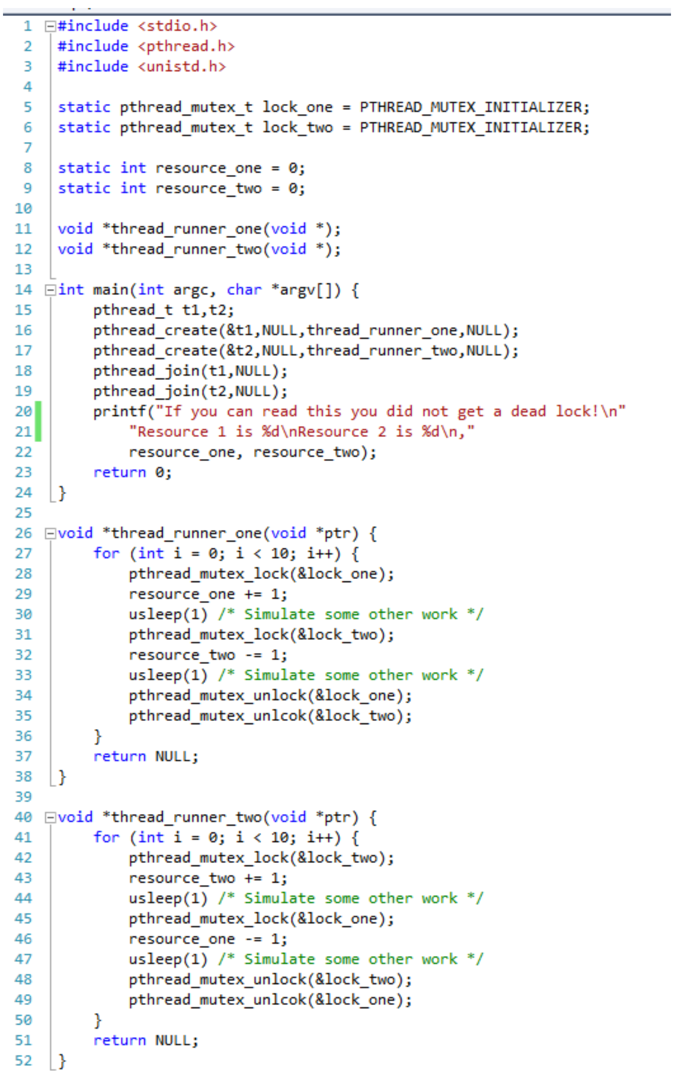
No preemption: *once acquired by a process, resources cannot be preempted (forcibly withdrawn) but are only released voluntarily.*

Wait-for cycle: *a circular chain (or cycle) of processes exists such that each process holds a resource which its successor in the cycle is waiting to acquire.*

In Figure 2.7 a simple example where dead lock occur, is shown. Each thread demands access to the same variables but not necessarily in the same order, as well as there are no preemption.

In order to avoid Deadlocks a different synchronization mechanism must be used. A suggestion is to have a maximum waiting time for accessing resources. By going back to the dining philosophers (Example 2.1) a new rule is now added. A philosopher can only hold a fork and wait up to 10 minutes to get access to the second fork before laying the first fork back on the table. The possibility of deadlocks are now eliminated as the "No preemption" (Definition 2.16) condition have been removed. Once again the philosophers find themselves at the table feeling hungry. At the same time all philosophers pick up the fork at their immediate left. Everyone waits 10 minutes, and then lay down their forks, just to pick it up again. The philosophers does not experience the deadlock as earlier, but change between states without having any progress. This is called a Livelock.

Definition 2.17. *Proposing the following definition:*
Livelock *is a system condition where the processes are limited to switch between a reduced set of states, and the system is not making any progress.*



```c
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

static pthread_mutex_t lock_one = PTHREAD_MUTEX_INITIALIZER;
static pthread_mutex_t lock_two = PTHREAD_MUTEX_INITIALIZER;

static int resource_one = 0;
static int resource_two = 0;

void *thread_runner_one(void *);
void *thread_runner_two(void *);

int main(int argc, char *argv[]) {
    pthread_t t1,t2;
    pthread_create(&t1,NULL,thread_runner_one,NULL);
    pthread_create(&t2,NULL,thread_runner_two,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    printf("If you can read this you did not get a dead lock!\n"
        "Resource 1 is %d\nResource 2 is %d\n,"
        resource_one, resource_two);
    return 0;
}

void *thread_runner_one(void *ptr) {
    for (int i = 0; i < 10; i++) {
        pthread_mutex_lock(&lock_one);
        resource_one += 1;
        usleep(1) /* Simulate some other work */
        pthread_mutex_lock(&lock_two);
        resource_two -= 1;
        usleep(1) /* Simulate some other work */
        pthread_mutex_unlock(&lock_one);
        pthread_mutex_unlcok(&lock_two);
    }
    return NULL;
}

void *thread_runner_two(void *ptr) {
    for (int i = 0; i < 10; i++) {
        pthread_mutex_lock(&lock_two);
        resource_two += 1;
        usleep(1) /* Simulate some other work */
        pthread_mutex_lock(&lock_one);
        resource_one -= 1;
        usleep(1) /* Simulate some other work */
        pthread_mutex_unlock(&lock_two);
        pthread_mutex_unlcok(&lock_one);
    }
    return NULL;
}
```

Figure 2.7: Example Code of a C-program with Deadlock.

As seen from definition 2.17, the main difference from the Deadlock condition is that the processes are not locked in one state. It should be mentioned that a Livelock condition may not always occur as it is also dependable of the chance of bad timing. However, if it is a possibility of Livelock (or Deadlock), the synchronization mechanisms must be changed to avoid this.

When operating with synchronization, a third state can occur. This is called starvation.

Definition 2.18. *From [25]:*
*...**Starvation**, the name given to a concurrent-programing situation in which an action is never executed...*

Starvation is often a result of a timing problem, and can be hard to detect as it may not always surface. Starvation can also occur when a process experience infinite overtaking. In [24] this was illustrated by:

> *...Suppose that a seated philosopher has an extremely greedy left neighbour, and a rather slow left arm. Before he can pick up his left fork, his left neighbour rushes in, sits down, rapidly picks up both forks, and spends a long time eating. Eventually he puts down both forks, and leaves his seat. But then the left neighbour instantly gets hungry again,rushes in, sits down, and rapidly snatches both forks, before his long-seated and long-suffering right neighbour gets around to picking up the fork they share. Since this cycle may be repeated indefinitely, a seated philosopher may never succeed in eating...*

This means that a given process almost never gain access to a desired resource. This could be fixed by letting this particular task have infinite wait rather than giving up after some time.

2.5.3 Network Programming

The most common practices inside Network Programming or Socket Programming are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Both of these protocols can be used for message passing, but only TCP can guarantee that a complete message is transmitted and received. Each server-client connection requires their own port on the network, but when this connection is set up, messages can be sent both ways.

A common practice when using TCP is to have one thread for sending, and one for receiving messages. The reason for this is that each "send message" function needs a "receive message" function on the other side, otherwise a dead lock occur. By having these functions in their separated threads, a dead lock will not have an impact on the rest of the system. This solution gives flexibility in the message passing as there is no regulation on how often, or how many messages has to be sent and received. This is especially useful when dealing with human interactions.

The sending thread should have a limited frequency in order to avoid stacking up messages at the receiving end.

UDP on the other hand has the ability to broadcast messages on one port. This is handy when “I am alive”-messages is needed, or the same information needs to be transmitted to many listeners. As with TCP a limited frequency should be used when broadcasting. A high frequent broadcast can lead to network overload, which again leads to collapse of the whole network.

Chapter 3

SRS for ROV DP system

This chapter contains the Software Requirement Specification (SRS) for the control system developed during spring 2012, and can be treated as a separate document. It follows the template outlined in Appendix B, Definition B.1. The document is written in a concise way, without any outlining and explanation of different topics.

3.1 Introduction

This document gives a description of the specifications to be implemented in the new control system for ROVs in association of NTNU.

3.1.1 Purpose

This SRS is to be used as a part of the development and restructuring of the computer-based DP control system for ROVs at NTNU.

The targeted group for this SRS is the developers of the control system at the department of Marine Technology, as well as the project managers connected to the AUR-Lab.

3.1.2 Scope

Two computer programs are to be developed in this process, a control system communicating with the ROV, and an independent GUI showing the necessary information to the pilot. A network connection between the control program and the GUI ensure exchange of information between them.

The control program shall be able to perform three different modes:

- DP Operations
  - Point to Point
  - Station Keeping
- Tracking and Path following
  - Lines between WP, fixed heading
  - Lawn Mower pattern
  - Survey lines
  - WP
- Joystick Mode
  - Manual Thrust Allocation
  - Position Reference
  - Velocity Reference

3.1.3 Definitions

SRS: Software Requirement Specification

ROV: Remotely Operated Vehicle

DP: Dynamic Positioning

AUR-Lab: Applied Underwater Robotics Laboratory

GUI: Graphical User Interface

WP: WayPoint

OO: Object Oriented

UTM: Universal Transverse Mercator

XML: eXtended Markup Language

3.1.4 References

3.1.5 Overview

3.2 Description

This program is created for controlling ROVs. Firstly it is aimed at the ROV Minerva, but an overall goal is to create software that with ease can be ported to other ROVs. The program is dependent of the hardware on the ROV.

3.2.1 Product Perspective

This control system is created with the programming language LabView and is therefore dependent of LabView development environment during implementation. As a final product the program is ported to an executable file which can be run independently.

The program is following an OO structure.

The GUI is to be developed in correspondence with collaboration with students from other study branches like Marine Biology and Marine Archeology, as well as Marine Cybernetic. It should be noted that there is two main modes in the GUI. Basic view is aimed at the average ROV Pilot during everyday operation. Advanced view is created for debugging, tuning and performance analysis, and is therefore mainly used by the developers of the program during testing procedures.

3.2.2 Product Functions

The control system has three main modes. These are DP operations, tracking and manual control. The user change between these modes manually with different buttons or switches in the user interface. Some security in the switching should be present such that unnecessary and dangerous switching is avoided.

DP Operations

In this mode the user can choose either to stay at the current position or change position to some new coordinates plotted in an input field or point on a map.

Tracking and Path following

WPs are given by the pilot such that a path is created by the reference model. WPs can be given through manually typed coordinates in an input field, predefined coordinates from file, or points from a map.

Different modes of behavior can be chosen depending on the operation. These include Aggressive Mode, Balanced Mode and Careful Mode. An own mode for pursuing a moving target is also included. Here the following distance and cross track error can be adjusted.

Joystick Mode

The ROV is to be controlled by input from a joystick. Direct thrust allocation shall be default startup mode. Other modes include position and velocity reference.

3.2.3 User Characteristics

The targeted group for this program is the marine subsea industry, such as marine biology, archeology, but also crew from the oil and gas industry. The regular end-users have low or none experience with control theory. However, all of the users have or are taking higher education, mostly master degree. The users need to have knowledge on how to use a computer and connect to external hardware, such as USB connections.

3.2.4 Constraints

The development environment for the control system is limited to LabView due to the control box between the computer and the ROV which is created by National Instruments. The user interface can be created in another language, but as LabView is exceptionally good at gages and instrument presentation it is almost certain that LabView is used for this as well.

3.2.5 Assumptions and Dependencies

The computers used to run the program is less than five years old and running a Windows operating system connected to Internet.

3.3 Specific Requirements

3.3.1 External Interface Requirements

User Interfaces

The basic view gives all the necessary information in order to functionally operate the ROV at any circumstances. The user interface is developed with LabView. An event driven structure is essential for the user interface as commands will be taken care of as they are given by the pilot. The gages present at the user interface are:

- Depth and altitude measurements
- Heading
- Signal check indicator

In addition a map view must be present. It should also be possible to have the map in a separate window.

The different modes are activated by switches or buttons by clicking on them. Only one mode can be active at a time.

A joystick shall be used to control the ROV. Configurations for different joysticks shall be possible.

Manipulator control and light settings must be easily accessible from the top layer as well as there should be speed key mapping to the joystick.

The advanced user interface view contains all information needed in order to perform proper debugging, testing and tuning of the system. This includes tuning variables for the controllers and observers, as well as graphs for positions, velocities, thruster- and sensor output.

Possibility for live video feed from the ROV through network must be checked.

Hardware Interfaces

The program uses a National Instruments compactRIO to communicate with the ROV. The host computer then needs at least an Ethernet port. An additional USB port is needed for the joystick. The user interface will fit a screen resolution of $1920 \times 1080px$ (HD Resolution).

Software Interfaces

The system is composed of two main modules, the user interface and the control system. On the user interface an event driven structure is chosen, while the control system has an OO-structure which is then separated into the following different modules:

- Vessel
- Observer
- Guidance
- Controller
- Thrust Allocation
- Thruster
- Sensor
- Supervisor

Each module is to be implemented as a class. The Observer, Guidance and Controller should be abstract classes to be inherit from by child classes. The Sensor class can also be a parent class.

The communication between the modules are only through the vessel object and especially its state cluster. Implicit communication is through a supervisor that ensures correct behavior in each module.

Communication Interfaces

Between the user interface and the control system there are a TCP/IP connection for information exchange. The communication between the control system and the ROV is also through an Ethernet interface. Information to be sent is

- Position values
- Velocity values
- Altitude values
- Thruster values
- Sensor values
  - MTi
  - Doppler Velocity Log
  - MRU6
  - ECF

The compactRIO is connected to the host PC through an Ethernet cable. Bit values are sent to the compactRIO through serial port connections, and passed on to the control system with the ethernet cable.

3.3.2 Functional Requirements

DP Operations

The DP operation mode include operations like station keeping and A to B movement. DP operations take either a UTM coordinates directly from the Eiva NaviPac. An alternative is to use a local coordinate system where the coordinates are given from the mother ships (in this case R/V Gunnerus) GPS and MRU system together with a HiPAP. A translation method between UTM and local coordinates must be implemented.

In DP mode with joystick (see also section 3.3.2), the stepping size in x, y, and z direction is 0.1 m-1 m, while the heading has 1°-10°. These are given through direction- and push buttons on the joystick.

If no specific coordinates are given to the DP system when the mode is activated, the current position is used in station keeping until new command arrives.

New coordinates is given through one of five modes:

- Change from current position in NED
- New position in NED
- Distance and Bearing
- Change in Surge and Sway direction
- From WP in map

Tracking and Path Following

In this mode the pilot gives the control system WPs such that a path can be generated. The WPs is given through one of the following ways

- Manual coordinate input in array
- Predefined coordinates from file
- Point and click method in radar map, maybe in coordination with manual input
- Lawn mower path generation with predefined parameter inputs.

The aggression level for the ROV can be adjusted by a slider or predefined modes like

- Aggressive
- Balanced
- Careful

These will change the behavior of the ROV while performing tracking and path following. The aggressive mode will make the ROV go quick, but at the cost of higher cross track error. On the other side the careful approach will give low cross track error, and relatively low speed.

Manual Control Mode

The manual control mode gives the ability to control the ROV through a joystick. The Joystick can either set desired thrust in wanted direction, or position and velocity reference. The manual thrust allocation mode must be activated by default when the system starts.

The position and velocity reference can be given through body and NED coordinates.

3.3.3 Other Functional Requirements

Possibility for station keeping in heading and depth/altitude must be available in all modes. 2D and 3D map of environments around the ROV. The program will read a configuration file at the beginning of the running process in order to set up the proper values for the system. The configuration file is written in XML format. The file contains the following fields

- ROV Data
  - ROV Name
  - Number of DOFs
  - Mass, Moment and Inertia
  - Linear Damping
  - Non Linear Damping
  - Coriolis effects
  - Added mass
  - Buoyancy
  - Umbilical data
    - Weight
    - Diameter
    - Length
- Control System with both DP and Tracking specifications
  - Observer

        * Gains
    - Controller
        * Gains
    - Navigation and Reference Models
        * Initial values.
    - Thrusters
        * Type
        * Specification
        * Position and orientation
        * Saturation
        * Coefficients for thruster expression
        * Rotation axis if able
    - Sensor
        * ID
        * Name
        * Type
        * Position and orientation

3.3.4 Performance Requirements

Only one pilot can connect and control the ROV at a time. The video feed can however if the bandwidth allows it be accessed by several client computers.

3.4 Design Constraints

The program is running on a 5-7 Hz frequency sampling time. This is given by the read-frequency between the compactRIO and the host PC.

3.5 Software System Attributes

3.5.1 Reliability

It is desired that the software is fail-safe such that consistency is preserved. In case of a major failure the backup joystick provided by the manufacturer is to be activated, either automatically if possible, or through a switch.

3.5.2 Availability

When the system is running, backup of the last steady state should be stored. If the system experiences a crash and needs a restart, this will happen automatically and the last steady state is read from the backup log in order to maintain integrity.

If the connection between the user interface and the control system is lost, the control system shall automatically enter DP operation and wait for the connection to be reestablished.

3.5.3 Security

Global variables are to be kept at a minimum in the system, but where it is necessary, proper synchronization techniques must be used. Input fields must have a max variable size such that memory is not overwritten. If this is not present it opens for intentionally crash of system or manipulated behavior of the system.

3.5.4 Maintainability

The software must be maintained with ease. It is important that at least key features in the program are well written in an intuitive way such that new developers understand what has been done. The software is divided into modules that can easily be substituted by similar blocks.

3.5.5 Portability

The software is mainly developed for windows based computers, but can with ease be exported to both Linux- and Mac-based operating systems by compiling for this. The control system is also mainly created for the ROV Minerva, but it is important to create the system such that another ROV can be used with another configuration file at initialization.

3.6 Other Requirements

The computer needs at least LabView Run-Time Engine installed in order to run the system. An installer such that everything will be set up correctly without the need of complex configuration and debugging must be created. This will include installation of LabView Run-Time Engine if the computer have not the environment already installed.

Chapter 4

The Control System - Njord

This chapter describes the implementation of the new control system. It has been given the name “Njord” after the Norse God of Sea Navigation. The control system was implemented by using LabView Object-Oriented Programming (LVOOP). Other OO languages could also have been used without any change in the structure.

In section 4.1 the structure of the program is outlined. The following section, section 4.2 explains the initialization process. Further in section 4.3 the transformation to 6DOF is described. Section 4.4 outline the communication between the control system and the other systems. In section 4.5 the testing of the control system is explained. This is followed by section 4.6, where the fault tolerance for the system is discussed. Lastly in section 4.7, issues and known problems are listed.

4.1 Structure

In the old control system the whole program was in one big thread. This made it slightly easier to develop, and exchange information between modules. However, the modularization were floating and it could be hard to separate different parts of the system from each other.

In the new system, Njord, thread programming has been taken in use. As it was a wish of having separate user interface and control system, this was a necessity. The system has four threads running concurrently, one for the control loop, and three for communication purposes. These are outlined respectively in algorithm 1, 3, 4 and 5.

Figure 4.1 shows the relationship between the different classes in Njord. This then constitute the new software architecture of the system. In the diagram, inheritance

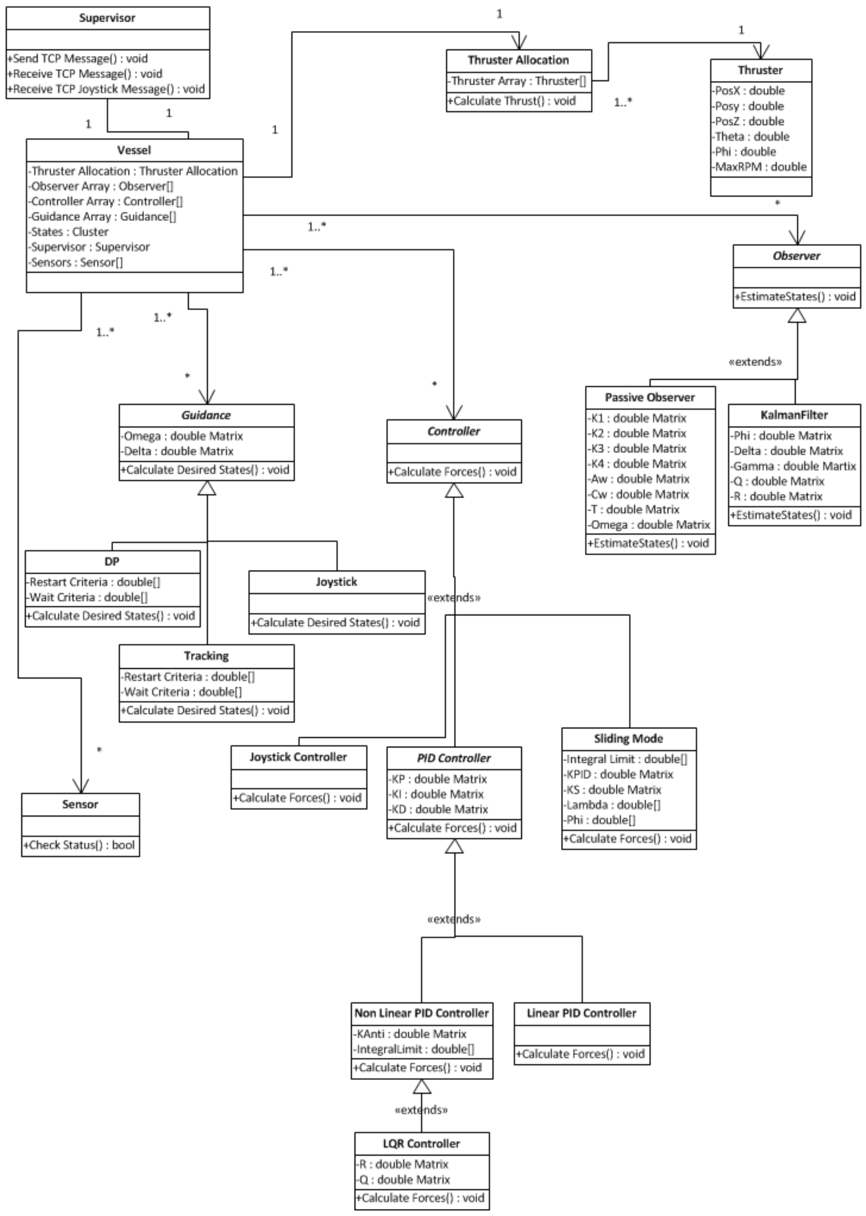


Figure 4.1: Class Relations in Njord.

relationship is marked with an arrow with hollow arrow head, as well as a text with "«Extends»" on the arrow body. Class relations are shown with the arrows with open arrow heads. In addition, these arrows have a relation indicator. The indicators are as follows:

```
1    - Only one
*    - Optional, zero or more
1..* - At least one
```

The Vessel class is the main element which ties all components together, and has at least one object from each super class other than itself.

Algorithm 1 Main Thread.

```
function MainThread(Vessel vessel)
    while !ExitFlag do
        Semaphore.wait()
        Supervisor.IterationNumber ← $LoopNumber$
        vessel.Supervisor ← $Supervisor$
        Call SignalProcessing() on Vessel
        Call ControlStrucure() on Vessel
        Call SendMessagecRIO() on Vessel
        Supervisor ← $Vessel.Supervisor$
        Semaphore.signal()
    end while
end function
```

The MainThread shown in Algorithm 1 will run until an exit signal from the user interface is given. If the GUI experience an error and crashes, the MainThread will still be running and make sure the ROV keeps its position.

The Control Structure will, with help from the Supervisor object, determine which observer, navigation and controller module to be used. A vessel can have different setups for the different modes in order to achieve optimal performance. This makes the system flexible and tolerable for different states without much extra work. For further inclusion of new observers, guidance strategies, or controllers the control structure needs no modification at all, unless there are some special conditions and logic that also must be taken care of.

The Receive Joystick Command Thread receive messages at a rapid pace in order to have tolerable sampling rate of the user input. The loop receive messages even when the joystick mode in the user interface is inactive. A flag in the message indicates whether or not the joystick mode is active. When deactivating the joystick mode, the control system goes to DP at the current position, awaiting new commands.

The Sending Thread will send a message every iteration. The frequency is set to be $20Hz$ in order to achieve a good sampling rate on the map visualization and graph plots. Before and during program execution, the sending thread can be set to ignore locking of the semaphore. This will not give any problems as the sending function only reads the values in the supervisor object without writing anything back. This can be compared to taking a copy of a document rather than using the original.

As stated in section 2.5.3, a receive function will wait for a message to arrive. In

Algorithm 2 Control Structure.

```
function ControlStructure
    Update Tuning Matrices
    if MTi is Active then
        Call EstimateStates() on Attitude Observer
    end if
    if Control Loop is Active then
        Call EstimateStates() on Desired Observer
    else
        Call EstimateStates() on Kalman Filter
    end if
    Call EstimateStates() on Altitude Observer
    if MTi is Active then
        Update Measured States
    end if
    if Control Loop is Active then
        Call CalculateDesiredStates() on Desired Guidance
        if Use Altitude Guidance then
            Call CalculateDesiredStates() on Altitude Guidance
        end if
        Make States continuous
        Call CalculateTau() on Desired Controller
    else
        if Joystick Mode Selected then
            Call CalculateDesiredStates() on Joystick Guidance
        end if
        Make States continuous
        Call CalculateTau() on Joystick Controller
    end if
    Call CalculateThrust() on Thruster Allocation
end function
```

Algorithm 3 Joystick Message Receive Thread.

```
function RECEIVEJOYSTICKCOMMANDTHREAD
    loop
        if !Connection.Healthy() then
            ListenForClients() in 5000$ms$
        end if
        if No Error then
            Semaphore.wait() in 25$ms$
            if !Semaphore.TimedOut then
                Supervisor.ReceiveJoystickTCPMessage()
                Semaphore.signal()
            end if
        end if
    end loop
end function
```

Algorithm 4 Send TCP Message.

```
function SENDTCPMESSAGE
    while !ExitFlag do
        if Communication Problems then
            Supervisor.OperationType ← $DPOperation$
        end if
        if !Connection.Healthy() then
            ListenForClients() in 5000$ms$
        end if
        if No Error then
            if !Ignore Semaphores then
                Semaphore.wait() in 25$ms$
            end if
            if !Semaphore.TimedOut then
                LocalSupervisor ← $GlobalSupervisor$
                if !Ignore Semaphores then
                    Semaphore.signal()
                end if
            end if
            SendMessage() on LocalSupervisor
        end if
    end while
end function
```

Algorithm 5 Receive TCP Message.

```
1: function ReceiveTCPMessage
2:     while !ExitFlag do
3:         Semaphore.wait() in 25$ms$
4:         if !Semaphore.TimedOut then
5:             Supervisor.ReceiveMessage()
6:             Semaphore.signal()
7:         end if
8:     end while
9: end function
```

order to avoid a forever waiting thread and then an eventual dead lock, a time out limit is set on the receive function such that the thread will run and check for messages rather than stand and wait for one to maybe arrive.

4.2 Initialization

In order to set up the system, a configuration file is used. This file contains initial and default values for all the components in the system. The reason for using such a file is to make the system less tailor made for one ROV. This will also make commissioning work easier as the necessary work can be limited to tuning of modules and parameter determination.

The file format is chosen to be eXtended Markup Language (XML) as it is structured and easy to read both for humans and machines. XML is a tag-based metalanguage, which means that an author is able to generate tags that specify the structure [26]. The syntactical layout XML gives, is advantageous regarding parsing of the file during initialization of the system.

4.3 6-DOF Adaptation

The old control system yielded for 4-DOF, as well as it was specially made for the ROV Minerva with some ad-hoc solutions. One of the primary goals for the new system was to make it more accustomed to several UUVs, with main focus on ROVs. In order to do so, all equations and system modules had to be expanded to 6-DOF or at least work with a 6-DOF setup.

4.3.1 Observers

Sector Kalman Filter

The Kalman Filter observer from the old system has been translated to the new structure. The output is transformed to work with a 6-DOF system. However, the observer is not a 6-DOF observer!

The linearization has only been done with respect to 36 different yaw angles. If this were to be done for both roll and pitch as well, the initialization would need to iterate and store values from 36^3 iterations for a sampling interval of 10 degrees in each rotation direction. The performance requirement needed can be calculated by the following formulas:

$$N \times \varpi = Memory\, requirement \tag{4.1}$$

N is the number of elements to be stored and ϖ is the data size. The processor usage can be found by identifying FLoating-point OPerations (flop) in the routine and how many FLoating-point Operations Per Second (flops) the processor is capable to perform.

$$\sigma_{flop} \times n_{iter} = flop \tag{4.2}$$
$$\chi \times_{core} \times n_{core} = flops \tag{4.3}$$
$$\frac{flop}{flops} = t \tag{4.4}$$

σ is the number of flop per iteration, n is the number of iterations. In (4.3) χ is number of flop the processor is able to perform on each cycle. On modern processors this is usually 4. f is the clock frequency or cycle frequency of the processor, and m is the number of cores available on the processor.

$$(2 \times (18 \times 36^3 \times 18) + 18 \times 36^3 \times 6) \times 8\, B = 282175488\, B \cong 282\, MB \tag{4.5}$$

The expression in (4.5) applies for the linearization matrices Φ, Γ and Δ in the Kalman Filter. 282 MB free memory is needed for each Kalman Filter object initialized in the system in order to store these matrices.

$$6943\,\frac{flop}{iteration} \times 36^3\,iterations = 323932608\,flop \tag{4.6}$$

$$4\frac{flop}{cycle} \times 1.6e9\frac{cycles}{second} \times 4\,cores = 25.6e10\,flops \tag{4.7}$$

$$\frac{323932608\,flop}{25.6e9flops} \cong 0.01seconds \tag{4.8}$$

The values in (4.3) is based on the authors PC which has a Intel i7-720QM [27], hence the numeric result in (4.4) is only valid for this processor. It should however be noted that the difference in run-time performance for the different processors is of $O(10^{-2})$, hence the calculation time for the matrices can be neglected.

With the results given in (4.5) and (4.8) there should in theory be no problem running a 6 DOF linearized Kalman Filter on a modern PC. However, the system performance may be reduced due to large memory operations, especially if global variables are used [28], [29].

Passive Nonlinear Observer

A Passive Nonlinear observer was implemented in order to have access to a 6 DOF observer for the system. The algorithms outlined for this observer described in [8] and [1] applied to a 3 DOF surface vessel in DP, hence the equations had to be slightly modified in order to be suitable for an ROV. The notation used in (4.9) is explained in C.

$$\dot{\hat{\xi}} = A_w + K_1(\omega_0)\tilde{y} \tag{4.9}$$

$$\dot{\hat{\eta}} = J(\eta)\hat{\nu} + K_2\tilde{y} \tag{4.10}$$

$$\dot{\hat{b}} = -T^{-1}\hat{b} + K_3\tilde{y} \tag{4.11}$$

$$M\dot{\hat{\nu}} = -D_L\hat{\nu} - D_{NL}|\hat{\nu}|\hat{\nu} + J^{-1}\hat{b} + \tau + J^{-1}K_4\tilde{y} - C\hat{\nu} \tag{4.12}$$

$$\hat{y} = \hat{\eta} + C_w\hat{\xi} \tag{4.13}$$

In the implementation $y = \eta$ was chosen in order to have full 6 DOF system. It is also possible to expand the the observer to include velocity measurements.

The observer gains must be tuned to retain its passivity, and to relate the gains to the dominating wave response frequencies. [1] proposes the following rules for passive observer tuning.

The four gain matrices $\boldsymbol{K}_1, \boldsymbol{K}_2, \boldsymbol{K}_3, \boldsymbol{K}_4$ are all diagonal:

$$\boldsymbol{K}_1 = \begin{bmatrix} diag\{k_1, \cdots, k_6\} \\ diag\{k_7, \cdots, k_12\} \end{bmatrix} \tag{4.14}$$

$$\boldsymbol{K}_2 = diag\{k_13, \cdots, k_18\} \tag{4.15}$$

$$\boldsymbol{K}_3 = diag\{k_19, \cdots, k_24\} \tag{4.16}$$

$$\boldsymbol{K}_4 = diag\{k_25, \cdots, k_30\} \tag{4.17}$$

where the values $k_1, \cdots, k_30$ have the following proposed values:

$$k_i = \begin{cases} -2(\zeta_{ni} - \zeta_i)\frac{\omega_{ci}}{\omega_i} & \text{if } i = 1, \cdots, 6 \\ 2\omega_i(\zeta_{ni} - \zeta_i)\frac{\omega_{ci}}{\omega_i} & \text{if } i = 7, \cdots, 12 \\ 2\omega_{ci} & \text{if } i = 13, \cdots, 18 \end{cases} \tag{4.18}$$

where $\omega_{ci} > \omega_i$ is the filter cut-off frequency. $\zeta_{ci} > \zeta_i$ is a tuning parameter (between 0.5 and 1.0). $k_i, i = 19, \cdots, 24$ should be large enough to ensure proper bias estimation. The values $k_i, i = 25, \cdots, 30$ have to be tuned according to the final performance of the observer.

$\omega_i = \frac{2\pi}{T_{pi}}$ is the peak response frequency. In most cases, this frequency can be approximated to the peak wave frequency. This is especially true for ROVs, which are much more sensitive to the different wave frequencies than, for example, a ship.

Adaptive Observer

In addition to the 4-DOF Kalman Filter and the 6-DOF Nonlinear Passive Observer, an adaptive observer was implemented into the system. This observer is designed for situations where the parameters of $\boldsymbol{A_w}$ are not known ([30]). This particular feature makes the observer able to adjust wave filter gains to the current sea state through an iterative process, thus making it viable to use in the wave zone for prolonged periods of time."It is uncommon to conduct important missions with an ROV in the wave zone. However, with the addition of the adaptive observer, it is possible to extend the range of ROV operations closer to the surface."

Adaptive observer design is based on the augmented observer equations ((4.19) through (4.23)).

$$
\begin{aligned}
\dot{\hat{\boldsymbol{\xi}}} =& \boldsymbol{A}_w\hat{\boldsymbol{\xi}} + \boldsymbol{K}_{1h}\tilde{\boldsymbol{y}}_f && (4.19)\\
\dot{\hat{\boldsymbol{\eta}}} =& \boldsymbol{J}(\eta)\hat{\boldsymbol{\nu}} + \boldsymbol{K}_2\tilde{\boldsymbol{y}} + \boldsymbol{K}_{2l}\boldsymbol{x}_f + \boldsymbol{K}_{2h}\tilde{\boldsymbol{y}}_f && (4.20)\\
\dot{\hat{\boldsymbol{b}}} =& -\boldsymbol{T}_b^{-1}\hat{\boldsymbol{b}} + \boldsymbol{K}_3\tilde{\boldsymbol{y}} + \boldsymbol{K}_{3l}\boldsymbol{x}_f && (4.21)\\
\boldsymbol{M}\dot{\hat{\boldsymbol{\nu}}} =& -\boldsymbol{D}\hat{\boldsymbol{\nu}} - \boldsymbol{J}^\top(\eta)\boldsymbol{G}\hat{\boldsymbol{\eta}} + \boldsymbol{J}^{-1}(\eta)\hat{\boldsymbol{b}} + \boldsymbol{\tau}\\
& + \boldsymbol{J}^{-1}(\eta)(\boldsymbol{K}_4\tilde{\boldsymbol{y}} + \boldsymbol{K}_{4l}\boldsymbol{x}_f + \boldsymbol{K}_{4h}\tilde{\boldsymbol{y}}_f) && (4.22)\\
\hat{\boldsymbol{y}} =& \hat{\boldsymbol{\eta}} + \boldsymbol{C}_w\hat{\boldsymbol{\xi}} && (4.23)
\end{aligned}
$$

where $\boldsymbol{K}_{1h} \in \mathbb{R}^{12\times 6}$ and $\boldsymbol{K}_{2l}, \boldsymbol{K}_{2h}, \boldsymbol{K}_{3l}, \boldsymbol{K}_{4l}, \boldsymbol{K}_{4h} \in \mathbb{R}^{6\times 6}$ are additional gain matrices to be determined. A new state has been introduced: $\boldsymbol{x}_f$, which is the low-pass filtered innovation. It is defined as shown in (4.24) ([30])).

$$
\dot{\boldsymbol{x}}_f = -\boldsymbol{T}_f^{-1}\boldsymbol{x}_f + \tilde{\boldsymbol{y}} = -\boldsymbol{T}_f^{-1}\boldsymbol{x}_f + \tilde{\boldsymbol{\eta}} + \boldsymbol{C}_w\tilde{\boldsymbol{xi}} \qquad (4.24)
$$

where $\boldsymbol{x}_f \in \mathbb{R}^6$, and $\boldsymbol{T}_f = diag\{T_{f1}, \cdots, T_{f6}\}$ contains filter constants. The high-pass filtered innovation $\tilde{\boldsymbol{y}}_f$ can be derived from 4.24 in the form of equation 4.25

$$
\tilde{\boldsymbol{y}}_f = \dot{\boldsymbol{x}}_f \qquad (4.25)
$$

The adaptive observer operates with the estimation problem where the parameters of $\boldsymbol{A}_w$ are unknown ([30]). The elements in the matrix vary depending on the current sea-state. Assuming decoupled motions, the composition of the matrix $\boldsymbol{A}_w$ is shown in (4.26).

$$
\boldsymbol{A}_w(\boldsymbol{\theta}) = \begin{bmatrix} \mathbf{0}_{6\times 6} & \boldsymbol{I}_{6\times 6} \\ -\boldsymbol{\Omega}^2 & -\boldsymbol{\Delta\Omega} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{0}_{6\times 6} & \boldsymbol{I}_{6\times 6} \\ -diag(\boldsymbol{\theta}_1) & -diag(\boldsymbol{\theta}_2) \end{bmatrix} \qquad (4.26)
$$

where $\boldsymbol{\theta} = \left[\boldsymbol{\theta}_1^\top, \boldsymbol{\theta}_2^\top\right]^\top$. The vectors $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \in \mathbb{R}^6$ contain the unknown values that are to be estimated. In order for the adaptive law to operate, it is assumed that these values are constant or at least slowly-varying compared to the system states.

Thus, the new Wave Frequency (WF) model is modified as shown:

$$
\dot{\hat{\boldsymbol{\xi}}} = \boldsymbol{A}_w(\hat{\boldsymbol{\theta}})\hat{\boldsymbol{\xi}} + \boldsymbol{K}_{1h}\tilde{\boldsymbol{y}}_f \qquad (4.27)
$$

The parameter update law for $\boldsymbol{\theta}$ is as follows:

$$
\dot{\hat{\boldsymbol{\theta}}} = -\boldsymbol{\Gamma}_w\boldsymbol{\Phi}(\hat{\boldsymbol{\xi}})\boldsymbol{C}_h\tilde{\boldsymbol{x}}_a = -\boldsymbol{\Gamma}_w\boldsymbol{\Phi}(\hat{\boldsymbol{\xi}})\tilde{\boldsymbol{y}}_{\boldsymbol{f}}, \boldsymbol{\Gamma} > 0 \qquad (4.28)
$$

The regressor matrix $\boldsymbol{\Phi}(\hat{\boldsymbol{\xi}})$ is defined as:

$$\boldsymbol{\Phi}^{\top}(\hat{\boldsymbol{\xi}}) \triangleq \left[diag(\hat{\boldsymbol{\xi}}_1) \quad diag(\hat{\boldsymbol{\xi}}_2) \right] \tag{4.29}$$

The tuning matrices $\boldsymbol{K}_{1h}, \boldsymbol{K}_{2l}, \boldsymbol{K}_{2h}, \boldsymbol{K}_{3l}, \boldsymbol{K}_{4l}$ and $\boldsymbol{K}_{4h}$ should be tuned in a similar manner to the original matrices $\boldsymbol{K}_1, \boldsymbol{K}_2, \boldsymbol{K}_3$ and $\boldsymbol{K}_4$, as described in Section 4.3.1.

The complete structure of the adaptive observer is displayed in Figure 4.2. The structures for the bias and wave estimators, as well as the wave filter, are shown in Figures 4.3, 4.4 and 4.5.

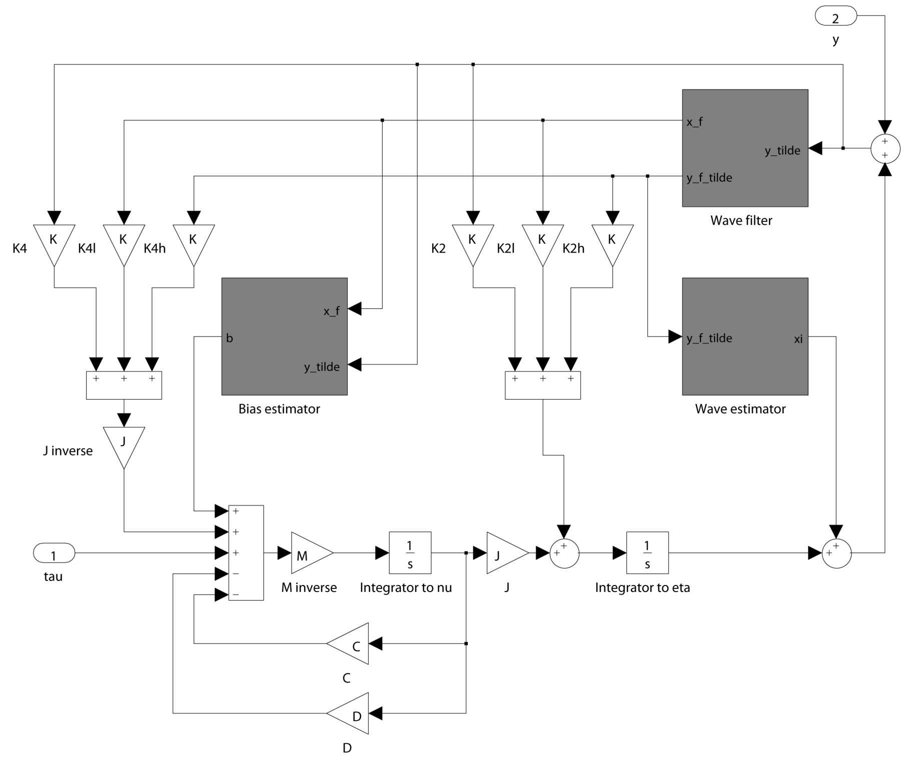


Figure 4.2: Complete model of the adaptive nonlinear passive observer.

4.3.2 Guidance Systems

There are two main guidance systems implemented in the system as of June 2012. These are Dynamic Positioning and Joystick mode. The DP guidance system is based on the work done in [31]. Some small modifications in vector and matrix

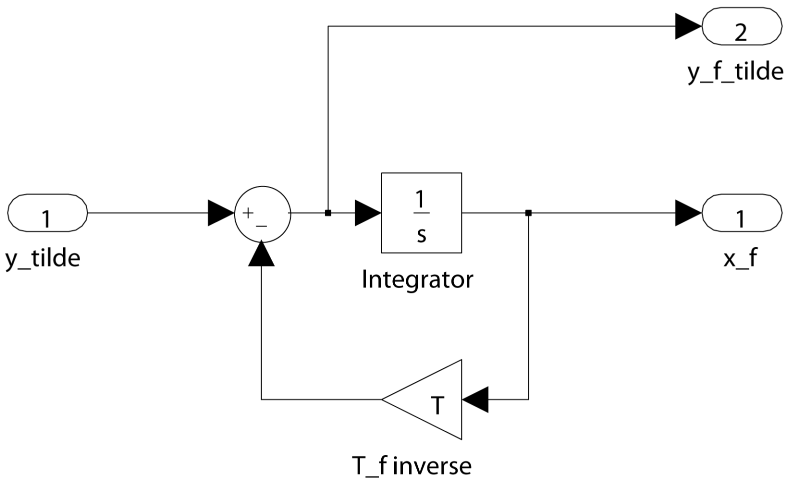


Figure 4.3: Wave filter subsystem including adaptive law.

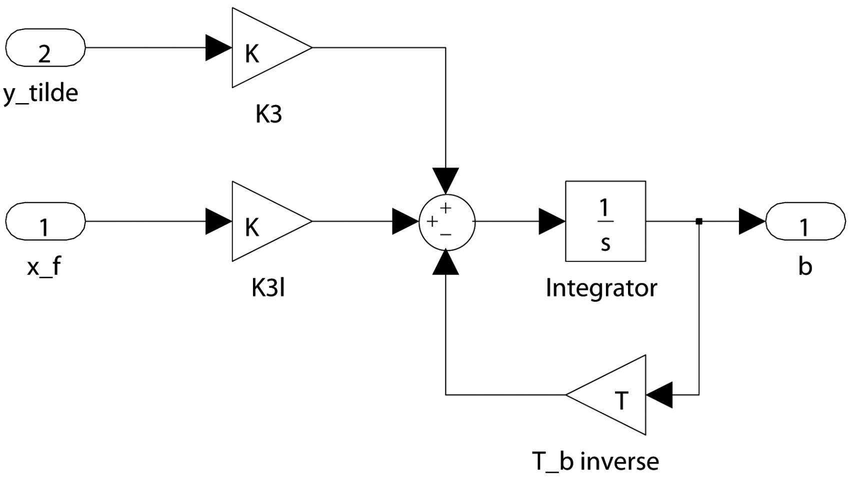


Figure 4.4: Modified bias estimator.

dimension have been done, as well as a new architecture of the method. However, the logic is the same.

The Joystick mode is based on the work done in [32]. Some small modifications have been done in the implementation here as well as the joystick input have been mapped to a number $\in [-100, 100]$ rather than using a bit value $\in \left[-2^{15}, 2^{15}\right]$. This mode has three main modes, direct Thrust Allocation, Position Reference, and Velocity Reference. The joystick frame can also be modified between body and

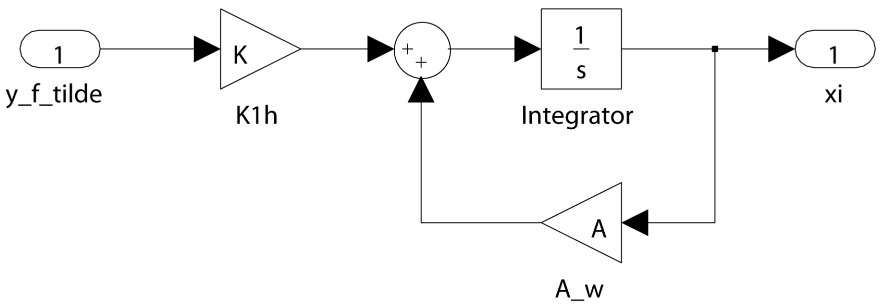


Figure 4.5: Modified wave estimator.

NED.

Implementation of a third guidance strategy was attempted. This was a tracking model with a synthetic reference model[33]. However, the complex nature of the guidance strategy created problems during implementation and due to shortage of time, it was abandoned.

All guidance systems implemented are set up for 6 DOF reference generation. However, only 4 DOFs (Surge/North, Sway/East, Heave/Depth and Yaw/Heading) are given reference values from the user.

4.3.3 Controllers

A selection of controller algorithms was developed for ROV Minerva. When the system was rewritten to accommodate 6 DOFs, the controllers had to be redesigned to be able to control pitch and roll as well as the other 4 DOFs. There are 4 different controllers implemented in the system:

- Linear PID Controller - a very basic control algorithm.
- Nonlinear PID Controller - an expansion of the linear PID Controller, including a feed-forward term. In the old control system for ROV Minerva, this controller was preferred for full scale tests [6]. The controller also includes an optional speed controller.
- LQR Controller - built on the framework of the nonlinear PID controller, but calculates the proportional and derivative controller gain matrices (respectively K_p and K_d) by solving the Riccati equation.
- Sliding Mode Controller - controls the system by forcing it to converge to a stable set of state variables, the so-called *sliding surface.*

The nonlinear PID, LQR, and the Sliding Mode controller are based on the ones described in [6].

Every controller requires adjustments in 2 main areas: integrators and tuning matrices. Due to the way the integrators are implemented, they have to be expanded from 4-DOF to 6-DOF, most notably the integrator initializing vectors and integrator limits. The tuning matrices also have to be adjusted, in order to accommodate the additional 2 DOFs.

The 6-DOF implementation is backwards-flexible. The controllers can now be used for controlling 6-DOF systems, while retaining the possibility of controlling systems with a reduced number of DOFs. This is achieved by simply setting the controller gains for the irrelevant DOFs to 0.

4.3.4 Thrust allocation

During the implementation of the thrust allocation algorithm, several alterations to the system were made in order to increase its adaptability in case of thruster failure. The thrust allocation matrix is calculated in real-time during the operation of the system. This allows to make it possible to dynamically alter the matrix in the case of thruster failure. Any thrusters set to be disabled will be excluded during computation of the matrix. In the future, an algorithm to detect any disabled thrusters can be created, which would allow for the operation of the ROV to be automatically adjusted in case of partial failure.

During the calculation of the thrust allocation matrix, a check is performed on the contributions of the thrusters for different DOFs. Since the algorithm is numeric, small rounding errors are present when calculating contributions that are close to 0. In order to avoid unnecessary strain on the ROV motors, errors below 10^{-6} are rounded down to 0.

The control vector $\boldsymbol{u}$ that comes out of the thrust allocation algorithm is converted to $\boldsymbol{n}[rpm]$ via (12.227) in [8], and is then limited using a saturation function. This function prevents the values in the rpm vector from increasing past the maximum allowed rotational velocity.

4.4 Communication Between Systems

As the user interface no longer is a part of the control system, the run time performance is more stable, and less likely to be provoked by heavy user input functions. All communication between Njord and Frigg is through TCP/IP.

4.4.1 Network Communication

The communication protocol between the control system and the GUI is through TCP/IP. This means that the control system and the user interface can run on separate computers as long as there are some sort of network between them. The control system should be run on a server or a real time system to ensure stability. The network messages sent to Njord are based on a number system where each number indicates which action to be performed.

```
<L><C>:<M>
L - Length of string casted to string. Always 4 bytes long.
C - Code for action. Notice separation with ``:''.
M - Message body. Dependant of the code.
```

The message sent to the user interface has the following setup:

```
Package 1:
        <L><P><P_est><P_des><S><V><V_est><V_des>
Package 2:
        <L><T>,<n_thrusters>,<R>
Package 3:
        <L><A>,<a_est>,<A_des>,<A_approx>,<D>
Package 4:
        <L><n_MT_DATA>,<MT_DATA><E><E_est><O><O_est>
L - Length of string casted to string. Alway 4 bytes long.
P - Position vector. Values separated with ``,''
V - Velocity vector. Values separated with ``,''
S - Numeric Representation of a boolean array. Used to send sensor
    status.
T - Thrust and moment vector. Values separated with ``,''
R - RPM values separated with ``,'' followed by the Thruster names.
A - Altitude
D - Data from the DVL unit. Values separated with ``,''
E - Euler angles vector. Values separated with ``,''
O - Rotational velocity vector. Values separated with ``,''
```

Package 1 - 4 are sent in a serial event. By this, it is meant that when Package 1 has arrived at the user interface, the next package is sent and so forth.

4.4.2 compactRIO

For this control system the compactRIO (cRIO) 9870 module[34] is used to communicate with the ROV. The cRIO contains a reconfigurable Field-Programmable Gate Array (FPGA) as well as a embedded controller. In addition expansion cards can be added to the cRIO in order to customize the Input-Output (I/O) configuration. The cRIO used together with Njord has expansion card for serial-port connections. They follow the RS-232 standard for serial port communication, and the hardware interface use the DE-9 port standard. The serial ports are used to receive data from the ROV- and the navigation-PC.

The cRIO is first of all used as a communication device where the interpretation of the serial-port signals is done on the FPGA. However, it is a wish to run the whole control system on the cRIO and take use of the RT-processor. With the cRIO AUR-Lab holds, this is not possible due to lack of storage space. However, with a larger storage device it should be possible to achieve RT-control with the use of Njord on cRIO.

4.5 Testing of the Control Structure

The unit testing started at Module Combination-Level, with exception of small independent functions and procedures were tested at forehand of the full control loop test. This resulted in many errors to be fixed simultaneously and the testing procedure took a while. However, this is hard to avoid as there were no good testing environment as well as many of the modules are dependent on each other.

The Module Combination-Level testing were conducted in a SIL-Test setup for Minerva. This means the control loop were integrated in the simulator and were given signals as function arguments rather than reading from the control box. This made testing of the control system easier, as there was no need for the cRIO.

Integration tests took place with HIL-Test setup, also for Minerva. As well as functionalities in the program, the communication through the cRIO were tested. Testing indicated good promise of the system.

During the testing the maximum waiting time of the communication threads had to be adjusted as the system had a chance of starvation on the sending thread. This thread is especially vulnerable for this situation as it runs on a timed loop, as well as having a time-out limit on the synchronization. By removing the time-out limit, i.e. the lock will wait for access, the starvation problem disappeared. However, the sampling rate of the sending thread seemed a bit slow. Further, a small modification was made in the sending thread, where the user can choose to deactivate the synchronization. As mentioned in Section 4.1 this will not result in any of the problems discussed in Section 2.5, as well as the performance is increased. This types of synchronization avoidance must however be used with caution, and is only valid in this case as the sending thread only reads the shared Supervisor-object.

The first system test on the ROV Minerva resulted in a strange and unknown error in the communication between the control system and the ROV. The control system received sensor data, but were not able to send commands to the ROV, even though no errors emerged on the control system side of the system. This stumbled both the author and co-advisor phd. cand. Fredrik Dukan, especially since the system worked well in HIL-test setup. Later HIL-tests were not conducted, but during the sea trial (Section 6.2) communication worked both ways as intended. It is possible that the problem mentioned above was a result of a failed compilation on the FPGA module on the cRIO.

4.6 Fault Tolerance in Njord

Larger parts of the control system has of June 2012 unfinished fault tolerance implementation. However, the structural layout of Njord is set up for error handling such that forward error recovery can be attempted.

In the communication threads a check for healthy network connectivity is performed at every iteration. If the connection breaks, the vessel is put in station keeping. The thread continues with attempts to reestablish the connection with the user interface until a successful connection, or it is aborted.

The “receive message”-thread has a time-out limit on the TCP-listening function. This is used in order to avoid Deadlock and Livelock in the system as may or may not arrive. If the TCP read function times out, the system throws the LabView Error “Error 56”. As this is an error thrown by provoking the system, and has a known source, it can be cleared without any concern. This then makes the thread ready for its next iteration.

It can be appropriate to check for empty arrays and matrices online. If a variable is empty, the previous stored value is used in order to maintain consistency. At the same time an error message must be thrown such that the user is notified.

In case of a more serious failure, the vessel could be put in station keeping with a fail-safe control loop to give the ROV pilot time to switch to manual control. These should be failures that require a restart of the system. A more seamless solution for these types of failures could be redundant setup of control systems. An assumption to make this case valid must then be that the failure propagates from a fault not present in all instances of the running control systems. By utilizing broadcasting on UDP/IP ports an active-passive setup can be created. When the active control system breaks down, the passive control system detects a halt in the broadcasting and takes the place as the active one.

4.7 Known Problems

When changing from third to second quadrant in heading, the reference model in DP-operation may misbehave. Setting a new reference with zero heading will make the reference stable again. An alternative is to enter joystick mode, as this will also calm the system.

The Passive Nonlinear Observers cannot run during deployment of the ROV as the integrators have a risk of creating Not-a-Number (NaN) when the measurement signals kicks in. The problem occur when the signals are given in UTM coordinates. Otherwise the observer works properly.

Chapter 5

The Graphical User Interface - Frigg

An important, but often neglected part of a computer-based control system is the user interface. The ROV operator may not always have an education inside the field of marine technology or control theory, hence the system should be intuitive and easy to use such that a larger group can utilize its potential. The new user interface have been designed with this in mind. It has also been given the name "Frigg" after the Norse Goddess of Destiny.

In Section 5.1 the concept of the user interface is explained. Further in Section 5.2 the different panels and their functionalities are described. User testings have also been conducted in three rounds. This is outlined in section 5.3.

5.1 Concept

A key factor when developing a user interface is to recognize who the end-user is. Should the operation of an ROV be limited to persons with education inside the field of control theory? Or should it be available for others as well? Frigg is designed to be user friendly in every day usage. However, for the enthusiastic control engineer, advanced views and settings are available in the menus, and through short keys. This way the ROV operator may, if it is desired, look at plots and tuning settings in order to adjust the behavior of the control system.

Frigg is implemented by using event driven architecture in LabView. LabView makes the creation of GUI quite simple as drag-and-drop methods can be used, without the concern of long and complex initialization methods.

A criteria for the user interface was that it should be independent from the control system. This was solved by using TCP/IP connection in order to transmit messages

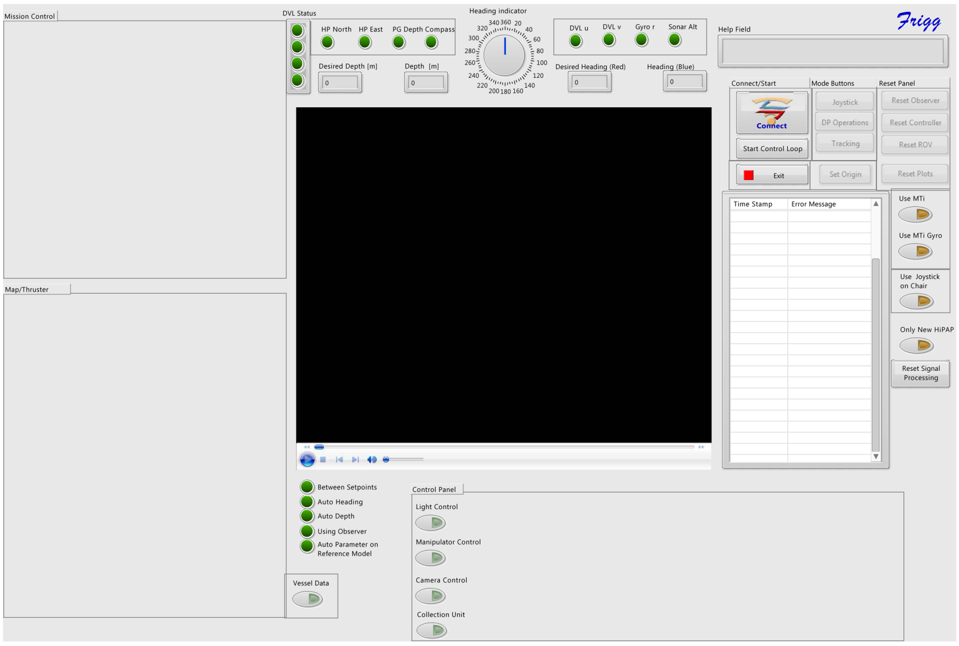


Figure 5.1: The user interface, Frigg.

and commands between these two systems. This opens up for the ability to develop the control system in a different programming language, without the need of changing the user interface, and vice versa. Another quality with this type of independence is that Njord and Frigg can run on separate computers, hence the risk of crashing the computer with the control system is reduced, which means the availability of the control system is increased.

It is a known fact that a person is easily confused and distracted if presented to too much data at once. Hence, a goal for the user interface was that only necessary information should be displayed at the *top level*. At the same time there should be easy access to different functionalities without having to search for it in different places.

The video player in the middle of the interface can be used to stream live video from the ROV. One requirement in order to achieve this is that the video is set up for broadcasting on an IP-address. Another is that the bandwidth of the connection support the amount of data traffic this requires. This gives the ability to remotely operate the ROV and its control system from a different geographical location.

5.2 Panels, Dialog Boxes and Functionalities

Frigg contains three main-areas where panels can be inserted. These are Mission Control, Map Visualization, and Control Panel. Mission Control contains the panels from DP Operation (Section 5.2.1), Tracking (Section 5.2.2) and Joystick Mode (Section 5.2.3). The Control Panel area displays Light Control (Section 5.2.5), Manipulator Control (Section 5.2.6), Camera Control (Section 5.2.4), and Collecting Unit Control (Section 5.2.7). Map Visualization shows, as the name indicates, the map (Section 5.2.8). Possibilities for 3D visualization have been created, but as this was a non-critical part if the user interface, it was down prioritized and will not be mentioned further in this chapter.

In addition to the panels, dialog boxes are created for certain functionalities. These are Online Tuning (Section 5.2.9), Graph View (Section 5.2.10), Help Center (Section 5.2.11), Startup Dialog Box (Section 5.2.12), Set Origin Dialog Box (Section 5.2.13), and Options Dialog Box (Section 5.2.14).

5.2.1 DP Operations

The panel shown in Figure 5.2 makes the user able to change the DP position in five different ways. The new position is sent to the control system when pressing the Apply Button.

Change NED

The user input is added to the current position of the vessel in order to create a new desired position.

New NED

This is the most common way of changing set point during DP operation. The input from the user will be the new desired position for the vessel.

Distance and Bearing

Distance and Bearing is the same as giving position in polar coordinates. The program gets the current position, and calculate the new destination based on the input from the user by using equation 5.1 and 5.2. The depth and heading is set manually in their respectively input fields.

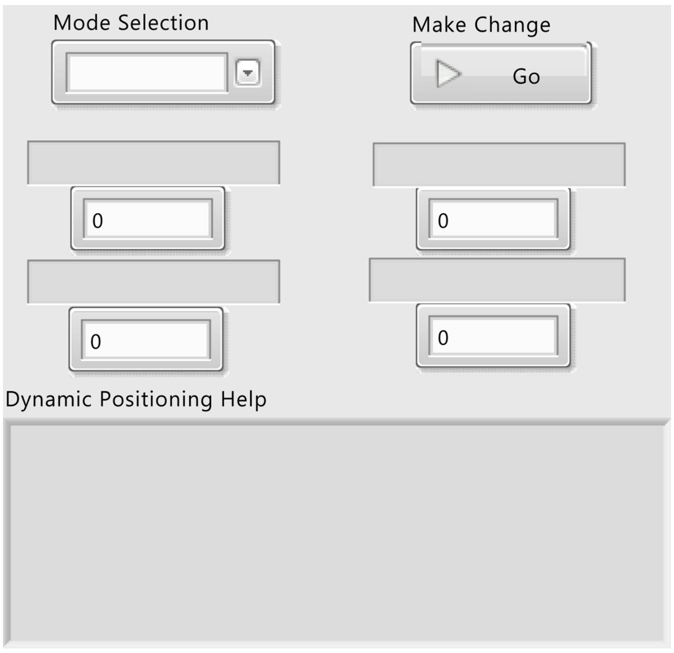


Figure 5.2: DP Panel in Frigg.

$$n = n_0 + distance \times cos(bearing) \tag{5.1}$$
$$e = e_0 + distance \times sin(bearing) \tag{5.2}$$

Surge and Sway

When using “Surge and Sway”-mode, the program will calculate the new NED position depending on the current heading and position of the vessel with the equations 5.3 and 5.4. The depth and heading angle input represents their new positions, and is not added to the current position.

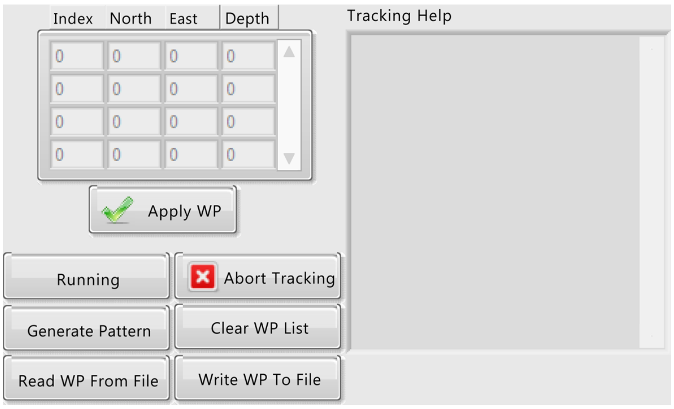
$$n = n_0 + surge \times cos(\psi_0) + sway \times sin(\psi_0) \tag{5.3}$$
$$e = e_0 + surge \times sin(\psi_0) + sway \times cos(\psi_0) \tag{5.4}$$

From Map

This mode retrieves the last placed WP from the map visualization, Section 5.2.8 and use the coordinates to determine the north and east position. The depth and heading angle must be given manually.

5.2.2 Tracking

Figure 5.3 displays the tracking panel as it will appear in Frigg. The tracking panel allows the user to enter WP coordinates manually in the table. Another alternative is to generate a lawn mower pattern. When pressing the “Generate Pattern”-button a dialog box with the correct input fields, pops up. A third alternative for generating a pattern is through the map visualization outlined in Section 5.2.8. WPs are placed by double clicking in the map, and is automatically appended to the WP-list.



Index	North	East	Depth
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Figure 5.3: Tracking Panel in Frigg.

Commands as “Pause” and “Abort” are placed in the tracking panel in order to minimize the spreading of the tracking functionality. It is also possible to save the WP-list to file, as well as import WPs from files. The WP-files have been chosen to have the extension “*.wp”, but are regular American Standard Code Information Interchange (ASCII) files. The advantage of using a self defined file extension is that it reduces the chance of wrong file input from the user.

5.2.3 Joystick

When activating the joystick mode the panel shown in Figure 5.4 is displayed in the “Mission Control”-frame. In order to give the ROV pilot maximum control, the joystick input is being modified. Modifications include dead zone recognition and scaling of the output range.

The reason for the usage of dead zone functions is to ignore small and unwanted deflections on the joystick. A naive way of implementing this is to check if the joystick deflection is larger than the given dead zone limit. If the deflection is not larger, then the joystick command is set to zero. This gives a step in the joystick axis commands which can be undesired as it can be hard to control and regulate.

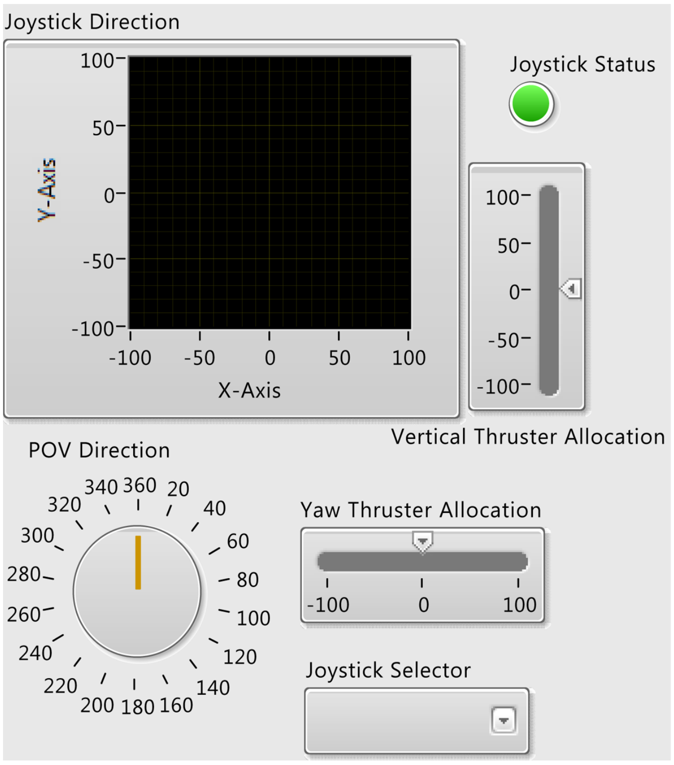


Figure 5.4: Joystick Panel in Frigg.

A suggested solution to this problem is to use the formula

$$x_{mod}(x) = \begin{cases} sign(x)\frac{x_{max}}{x_{max}-DZ_x}(abs(x) - DZ_x) & \forall\, x_{max} \geq |x| \geq DZ_x \\ 0 & \forall\, |x| < DZ_x \end{cases} \tag{5.5}$$

where $x_{mod}(x)$ gives the modified joystick command, x_{max} is the maximum deflection value of the joystick. DZ_x is the specified dead zone in x-direction and x is the current joystick deflection. The formula assume linear relation between joystick deflection and joystick command value.

Figure 5.5 shows the characteristics of the different approaches regarding dead zone mentioned above. The red line is the standard no dead zone output. The blue is the naive method, while the yellow follows the equation 5.5.

In addition to a separate commercial joystick plugged into the computer the joystick integrated in the ROV pilot chair, Figure 5.6, can be used. The joystick command is however not handled by the user interface. The user interface notifies the control system to take use of the joystick by reading the telegram buffer sent from the ROV control panel.

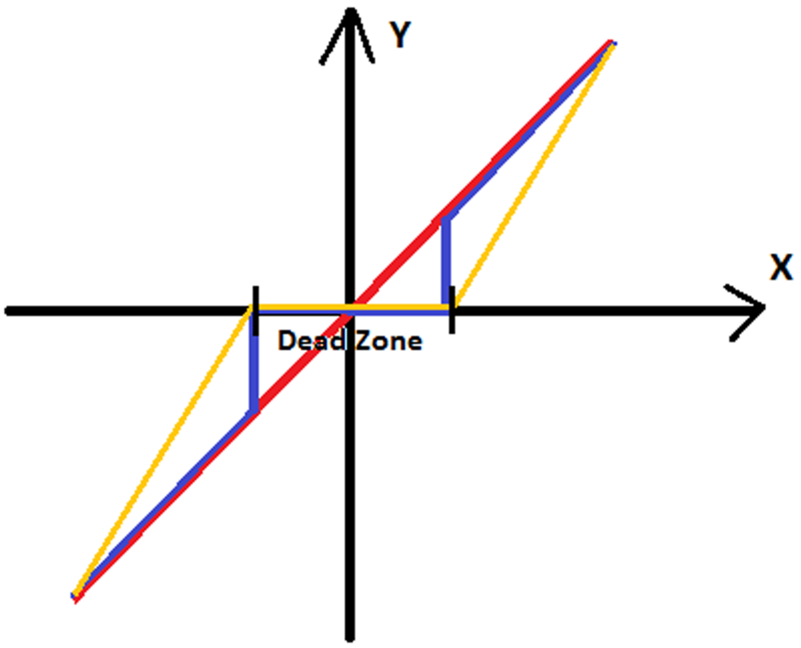


Figure 5.5: Joystick Axis Command Value Characteristics.

5.2.4 Camera Control

The panel shown in Figure 5.7 is used to control the camera settings on the ROV. This include pan, tilt and zoom of the main camera, as well as changing which extra camera to be active in the ROV video.

5.2.5 Light Control

The light control panel is shown in Figure 5.8. Light settings are set in this panel, and information is sent to the control system which takes care of the communication to the ROV.

The Hydrargyrum Medium-arc Iodide Lights (HMI Lights) needs to be handled with caution, they cannot be switched on again after being turned off, until about 5 minutes have passed. Pseudo code of the implementation is shown in algorithm 6.

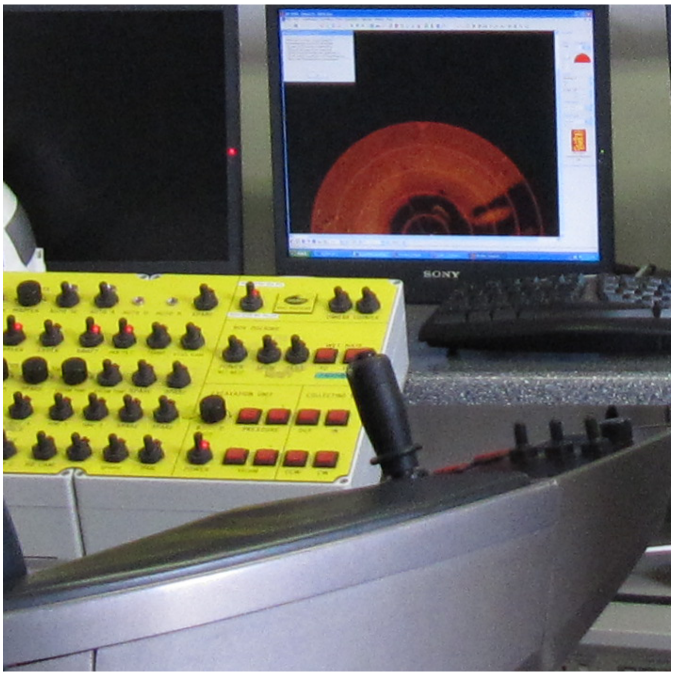


Figure 5.6: Joystick Integrated on ROV Pilot Chair.

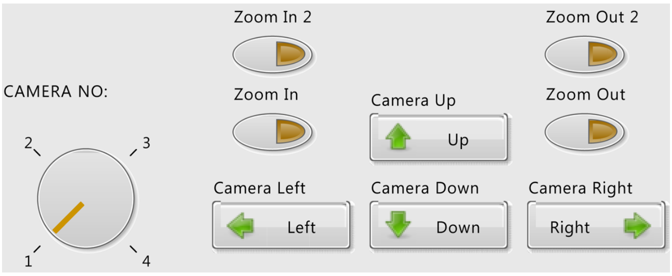


Figure 5.7: Camera Control Panel in Frigg.

5.2.6 Manipulator Control

The manipulator panel shown in Figure 5.9 is designed to work with the manipulator arm installed at Minerva. A safety mechanism has been implemented on the power switch such that if the arm is idle 15 seconds, the manipulator engine is turned of. Another restriction is that only one junction can be controlled at once. This was done in order to reduce the possibility of mixed inputs from the user.

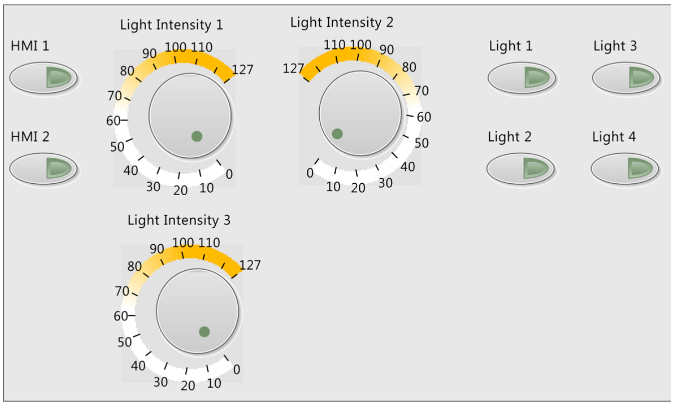


Figure 5.8: Light Control Panel in Frigg.

Algorithm 6 Pseudo Code for the HMI Timer Button

```
function HMIButtonPressedEvent(void)
    if Semaphore.free() && HMI then
        Semaphore.signal()
    else
        HMI ← $false$
    end if
end function
function TimerFunction(void)
    loop
        if !(HMI || Semaphore.free()) then
            Semaphore.wait()
            Reset Timer
        else
            Let Timer go
        end if
    end loop
end function
```

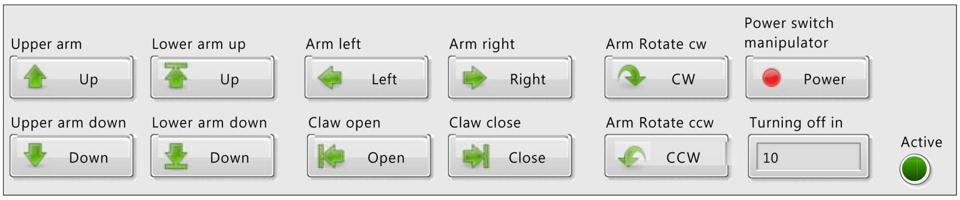


Figure 5.9: Manipulator Control Panel in Frigg.

5.2.7 Collecting Unit

Figure 5.10 shows the panel used to control the collecting unit on the ROV. This panel was originally created tailor made for 30k. Functionalities include driving the collecting unit in and out, and rotate the barrel back and forth.

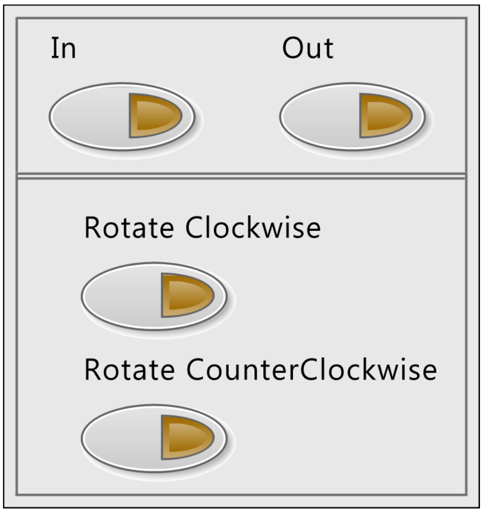


Figure 5.10: Collecting Unit Control Panel in Frigg.

5.2.8 Map Visualization

As briefly mentioned in Section 5.2.2, a new feature in the control system is the use of WP generation in the map view in the user interface. By adding a mouse click listener in the event structure, the user can with ease set new WPs by double clicking on the desired position. As the mouse position is given in pixel location on the screen a transformation is needed in order to use the coordinates. The function to convert the mouse pointer to map coordinates is given in equation 5.6 and 5.7.

$$X = XScale.Minimum + (Coords.Horizontal - PlotBounds.Left) \frac{XScale.Maximum - XScale.Minimum}{PlotAreaSize.Width} \tag{5.6}$$

$$Y = YScale.Minimum + (Coord.Vertical - PlotBounds.Top) \frac{YScale.Maximum - YScale.Minimum}{PlotAreaSize.Height} \tag{5.7}$$

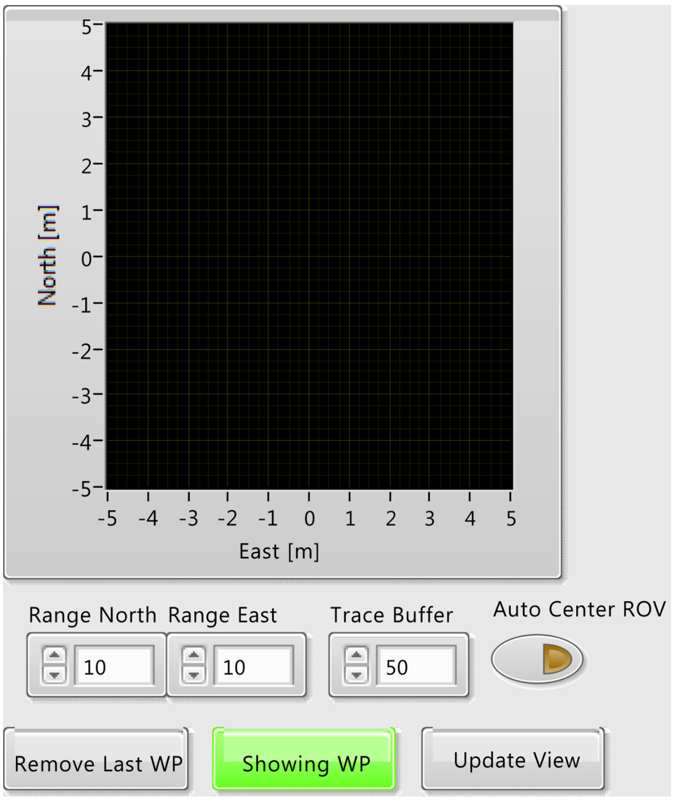


Figure 5.11: Map Visualization in Frigg.

The naming follows what is used in LabView where XScale/YScale.Minimum and XScale/YScale.Maximum are the axis boundaries. Coord.Horizontal/Vertical is the mouse coordinate. PlotBounds.Left/Top is the number of pixels from the left/top screen edge to the plotting object. PlotAreaSize.Width/Height is the distance in pixels of the plotting area.

In addition to the Map Visualization Panel, a dialog box with a map view is implemented. This view can be seen in Appendix ref. It must be mentioned that the map view in the dialog box only displays the position of the mother ship and the ROV, not WP and such.

5.2.9 Online Tuning

A requirement for the user interface was that there should be possible to adjust the tuning values on both controllers and observers in RT while the program was running. This was solved by creating different panels displayed in a frame, as shown in Figure 5.12, for the different components to be tuned. Each panel generates a string from one or more matrices which in turn are sent to the control system. In order to limit the necessary size of the messages sent, the user can choose to extract the diagonal elements and send them, instead of the full matrix. The user also has to specify which tuning matrices to be updated.

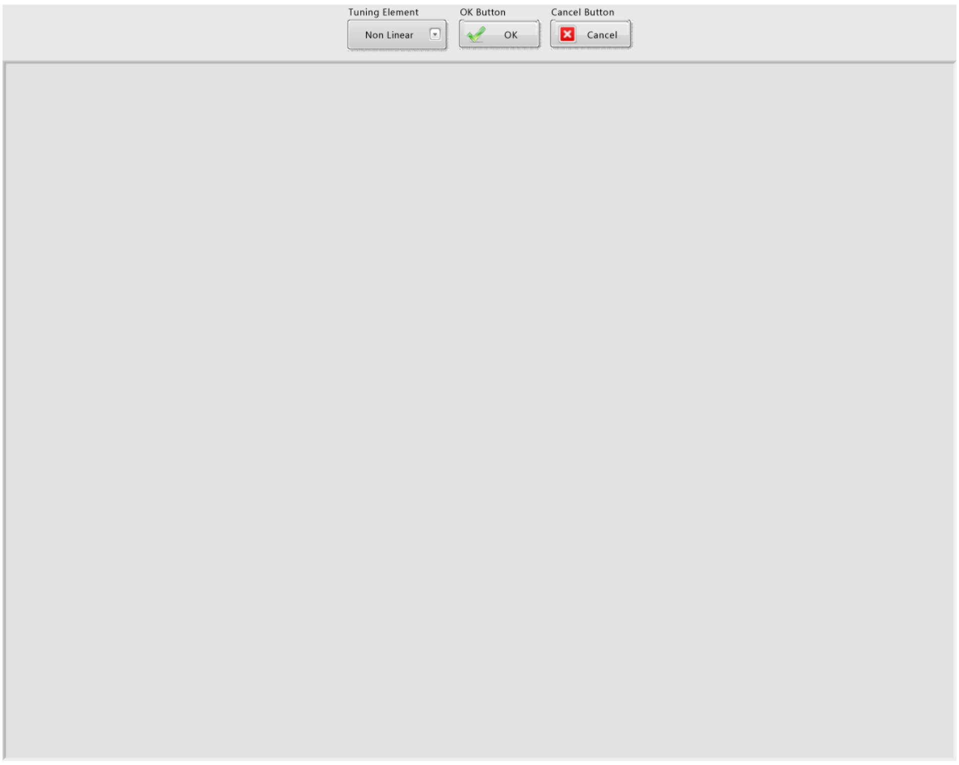


Figure 5.12: Tuning Panel in Frigg.

Before tuning can be done, a list of the available controllers and observers has to be obtained. This is done by pressing the “Update List”-button. A request is then sent to the control system, which responds with a list of its available controllers and observers. The Drop-Down Selector Menu in the Tuning Panel is then filed out with these elements. The Tuning Panel has then no fixed list, but adepts itself to represent the setup used by the control loop.

The tuning panels can be found in Appendix F, and include Nonlinear PID Tuning (Figure F.1), LQR Tuning (Figure F.2), Linear PID Tuning (Figure F.3), and Kalman Filter Tuning (Figure F.4).

Figure 5.13: Graph View in Frigg.

5.2.10 Graph View

In order to determine the performance of the control loop and its components, plots of the different states is a necessity. These are shown in a separate dialog box which must be activated by the user. The Graph View window, shown ing Figure 5.13, contains a panel frame where the selected Plot Panel is shown. Selection of which plots to be shown is done with a Drop-Down Selection Menu. As of June 2012 the Graph View includes Position Plots (Figure E.1), Velocity Plots (Figure E.2), Thruster Plots (Figure E.3), Altitude Plots (Figure E.4), MTi Data Plots (Figure E.5) and Explicit Complementary Filter (ECF) Plots (Figure E.6).

The different Plot Panels can be found in Appendix E.

5.2.11 Help Center

A Help Center (Figure 5.14) has been created in order to make Frigg easier to learn and understand. It is accessible from the Top Menu Bar and on the short key “F1”. A navigation tree is used to select which panel to be showed inside the frame.

In Appendix G the different panels added to the help center are shown. These panels are the Main Display (Figure G.1), Joystick Help (Figure G.2), Joystick Button Configuration (Figure G.3), Tracking Help (Figure G.4), Camera Panel Help (Figure G.5), Light Panel Help (Figure G.6), and Map Visualization Help (Figure G.7).

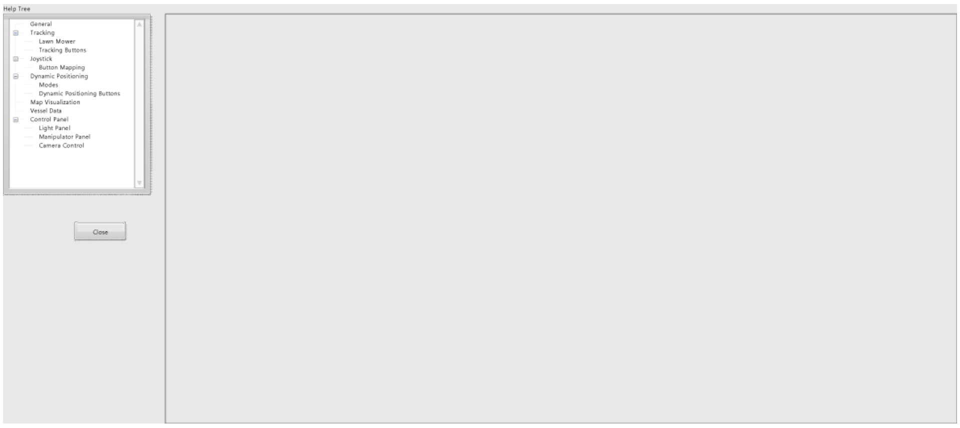


Figure 5.14: Help Center for Frigg.

5.2.12 Startup Dialog Box

When the user attempts to connect to Njord, or another compatible control system, the dialog box shown in Figure 5.15 appears. The user has to write the IP address of the computer running the control system as well as the communication ports. An option for a restart connection is also present, and can be used if the control loop is already running while the user interface crashed.

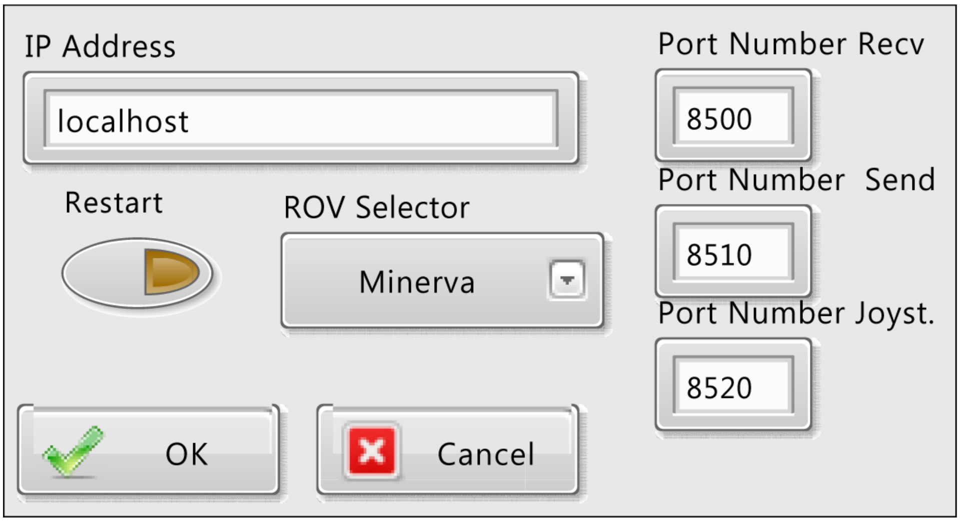
5.2.13 Set Origin Dialog Box

The Set Origin Dialog Box displayed in Figure 5.16 pops up when the user activates the control loop, and press the “Set Origin”-button on the Front Panel of Frigg. The Drop-Down Selector Menu which coordinate system the position is to be displayed in. ROV-coordinate system indicates that the current position is to be set to origin. The manual selection allows the user to set origin to a predefined coordinate set in the two input boxed, and UTM follow, as the name indicate, the UTM-coordinate system.

5.2.14 Options Dialog Box

All preferences and options are gathered in the dialog box shown in Figure 5.17. Again, sub-panels are used to show different sites which is determined by the the Drop-Down Selector.

In Appendix D the different panels are displayed. These include DP Options Panel (Figure D.1), Tracking Options Panel (Figure D.2), Joystick Option Panel (Figure D.3), Port Cofiguration Panel (Figure D.4), and Power Instruments Panel (Figure D.5).


IP Address: localhost

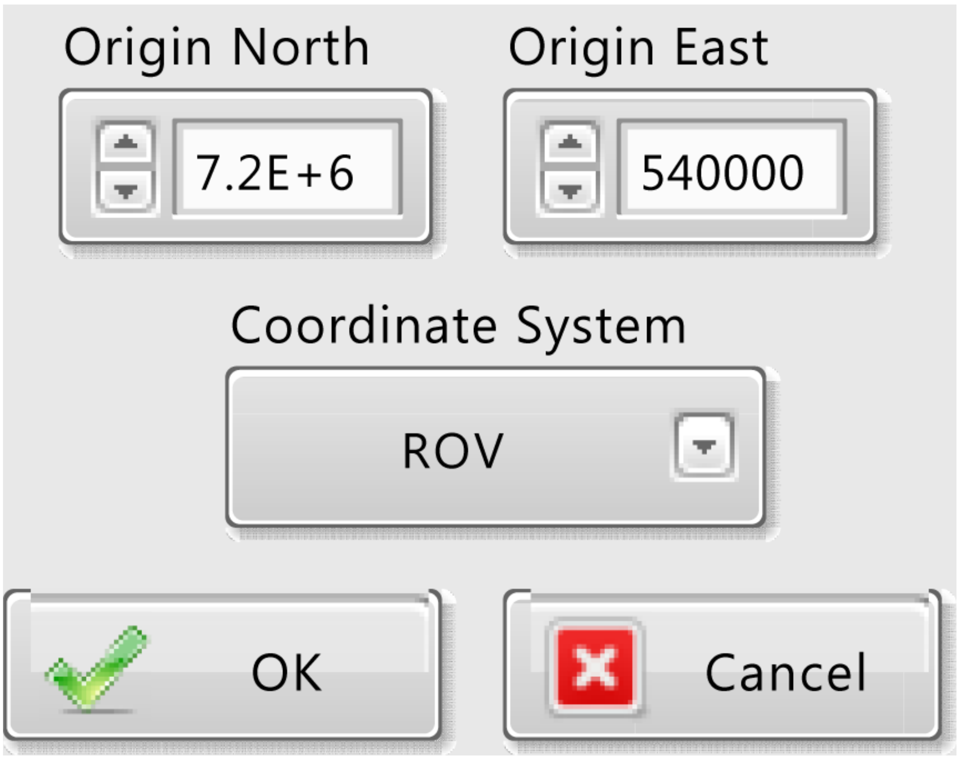
Port Number Recv: 8500

Port Number Send: 8510

Port Number Joyst.: 8520

ROV Selector: Minerva

Figure 5.15: Startup Dialog Box in Frigg.


Origin North: 7.2E+6

Origin East: 540000

Coordinate System: ROV

Figure 5.16: Set Origin Dialog Box in Frigg.

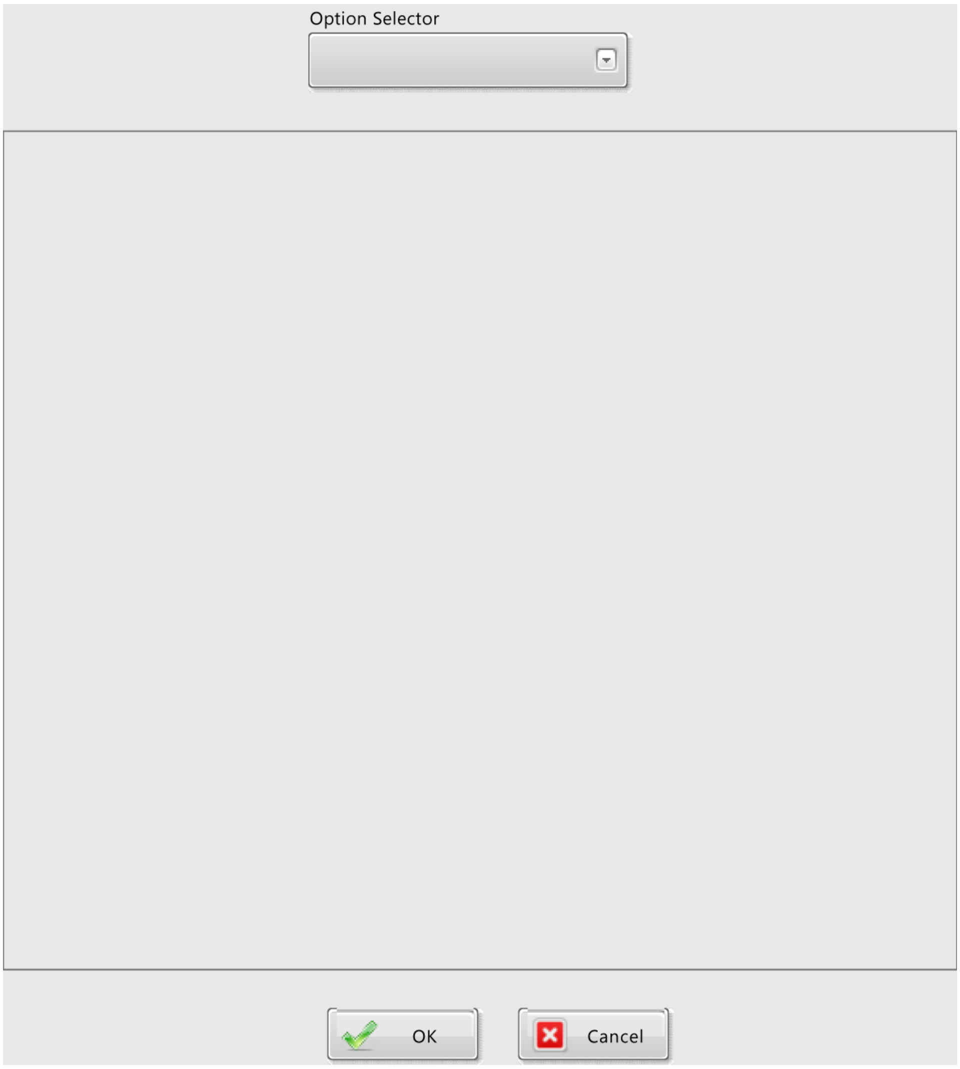


Figure 5.17: Option Dialog Box in Frigg.

5.3 Usability Testing

As stated in section 5.1 the aim for the GUI is to be intuitive and user friendly for a large range of users. In order to test this, a selection of voluntary co-students were asked to operate the system in a simulation. The testing was conducted in two phases. The first phase where conducted with a GUI connected to a server program and were aimed to find faults and errors due to wrong user input or behavior. The requirement for the testers were that they could not have knowledge of how to operate the system at beforehand. This can be compared to “Monkey Testing” or more specific “Dumb Monkey Testing” as the input, in this case from the human user, were uniformly distributed.

Between each tester, the system went through a development phase to fix the faults and errors found. This way each user test evolved the system to a more reliable state. The development model has then been similar to the iterative development process.

The second and third phase of the testing were conducted in HIL-test setup and were aimed at detecting instabilities in the control system as well as the user interface.

5.3.1 User-Friendly Testing

Several co-students were asked to have a go at the user interface during the development phase. They were deliberately not given any training at beforehand, so that they had to try and figure out how the program worked by exploratory browsing. After they had tested the system, they were asked how they experienced the user quality of the GUI. The feedback from the first test phase dealt with the confusion around connecting to the control system and that it was hard to get started. The terminology also seemed a bit confusing for non-marine students.

Before the next phase, the user interface experienced a face lift as an attempt to make the system easier to get started with. The connection button were given a new icon, a distinctive text, and were resized to a larger button. Small hints and explanations were also added in a text box when the mouse pointer hover over buttons or areas of interest.

The second test phase reported that it was still hard to figure out the order of button presses in order to start the system, i.e. first connection button, then bring the ROV to a depth where position signals were given, and then start the control loop. On the other side, after the starting procedure, the system seemed intuitive for the tester.

As a measure to limit the choices given to the user at start up, it was implemented that only the “Connect”- and “Exit”-button are available. When the user has connected, the “Start Control Loop”-button is enabled, and the joystick can be used to drive the ROV to an appropriate starting point. When the “Start Control Loop”-button is activated, the rest of the functionalities is enabled.

In the third testing phase, the tester experienced the user interface as intuitive and easy to use as their choices during start up were limited by reduced accessibility of the buttons. During this testing phase none of the test subjects were able to make the system crash, hence the interface can be said to be robust.

5.3.2 Functionality Testing

Early functionality tests were aimed at detecting faults and errors in the subroutines in the user interface. The majority of these tests have been conducted by the developer as new functionalities have been added to the system.

In the early stages of the development phase, a “server” program was created in order to verify the TCP messages sent from the GUI, as well as checking the values from the subroutines. The server program also sent messages to the user interface in order to simulate signals from an ROV, and at the same time check the position display functionalities.

Chapter 6

Commissioning

Prior to the sea trials for the ROV 30k that took place in the end of May, the author contributed in practical tasks and configuration of the navigation system setup. The work was done at Trondheim Biological Station (TBS). Areas that had to be checked included the system computers, installing sensors, finding the proper weight - buoyancy ratio, and installation and testing of the new manipulator arm[35]. In Section 6.1 a brief introduction to the navigation-PC used in ROV operations is given. Further in Section 6.2

6.1 NaviPac and Communication String Setup

NaviPac[36] is a Navigation Software developed by the Danish company Eiva A/S[37]. This software runs on a computer where the sensor data is sent to such that an output string can be generated by NaviPac for use on a host PC through a control box.

The sensor string is generated by the NaviPac program by defining a "Data Output". The user then has to define recognition code, separator character and the order of data to be sent. It is possible to combine information from several sensors in one string. When using such a string in a control system it is important that the developer has a proper documentation on how the string is set up.

A Doppler Velocity Log (DVL)[38] was used to measure heading, depth, altitude, and velocity, while a High Precision Acoustic Position (HiPAP)[39] sensor gave the measured position in north and east coordinates. With these combined the NaviPac String was chosen to be

`!Init;e;n;v;u;`a_1`;`a_2`;`a_3`;`a_4`;d;c;es;ns;cs`

`n - North`

```
e - East
v - Velocity in sway
u - Velocity in surge
a1 - altitude measurement 1
a2 - altitude measurement 2
a3 - altitude measurement 3
a4 - altitude measurement 4
d - depth
c - compass heading
es - East Mother Ship
ns - North Mother Ship
cs - Compass Heading Mother Ship
```

6.2 Wet Test and Sea Trial

A couple of days before the sea trial for the ROV 30k, a series of wet tests were conducted. These were aimed at checking the functionality of original system and ensuring that it worked properly. The vessel's behavior had to be modified such that it had a good balance both in weight-buoyancy ratio and weight distribution. After some weight had been added to the ROV, it seemed more relaxed and easy to control.

The night before the sea trial, the ROV was finally ready to be connected to the new control system. However, during the sea trials, time had to be used on establish full communication between the control system and the ROV. As this was the first time the control system were connected to 30k, there were problems in the sensor setup which had to be fixed before the ROV could be launched in the sea. After the communications were established, the manual thrust control mode was used to check maneuverability given the thruster allocation developed in [40]. The new control system was superior to the original system in this area as the desired thrust is run through thruster allocation. Unwanted movement was then to some degree accounted for, e.g. compensation in pitch angle when going forward.

Due to limited window of operation, tuning of the Kalman Filter was prioritized instead of the Nonlinear PID controller, as it was a necessity in order to be able to try the DP system. After reducing the response of the estimated surge and sway velocities, $\hat{u}$ and $\hat{v}$, the observer behaved in accordance with the measured values. The bias component had to be tuned as well, as there were stationary deviations in the North and East position estimates.

Figure 6.1 shows the position data for 30k during DP operation. From 10 to about 90 seconds the ROV performed station keeping at its position at the activation time for DP Mode, $\boldsymbol{\eta}_{ref} = [0, 0, 33.5, 2.2, 1.8, -50]^\top$. Even though the Nonlinear PID controller was not tuned, it managed to keep the ROV on position. The lack of controller tuning can especially be seen in the transit between two set points. After

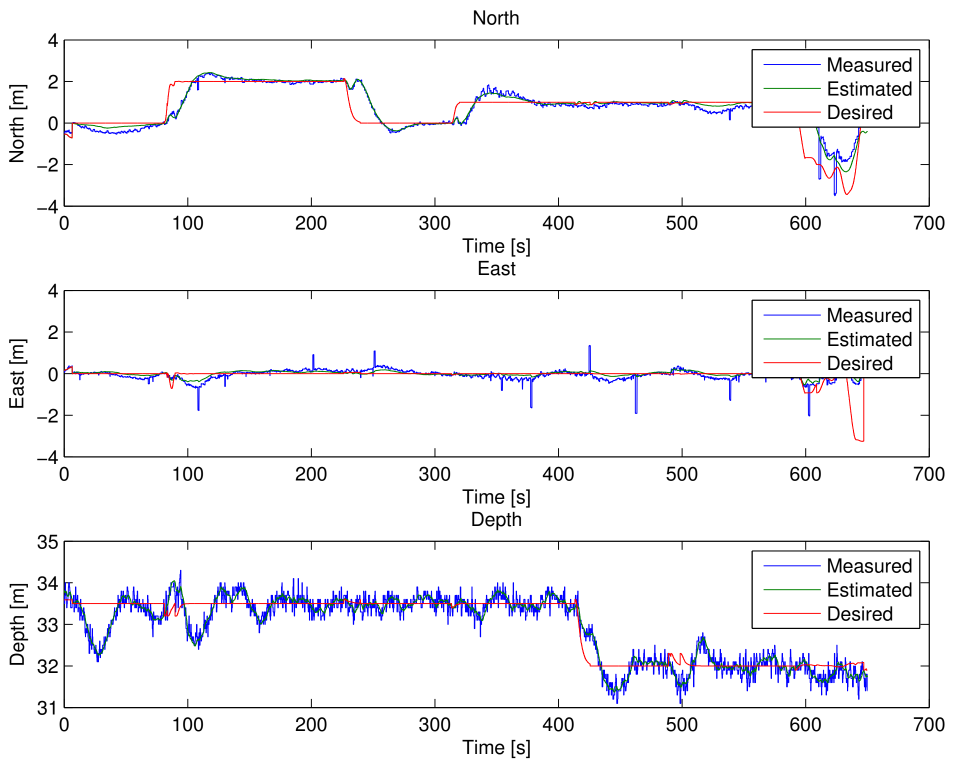


Figure 6.1: Plots of Translational motion during DP test.

about 90 seconds a new position was given to the ROV, $\boldsymbol{\eta}_{ref} = [2, 0, 33.5, 0, 0, 0]^\top$. The first graph in Figure 6.1 indicates that the proportional gain is too low. This is in accordance with the fact that the tuning values used are exactly the same as for Minerva, which is a much smaller and more agile ROV. The reference model parameters in the guidance system were also suited for Minerva rather than 30k, which can be seen by the rapid increase in desired position.

The orientation of the ROV during DP operation is shown in Figure 6.2. Graph number three is of most interest as it illustrates the heading angle of . It can be seen that the controller makes the ROV follow the reference. However, the same problem as described above yield in yaw as the controller is too weak to reduce the error in a timely manner. The orientations of the ROV is measured with a Xsense MTi[41], which is a gyro-enhanced Attitude and Heading Reference System (AHRS). This sensor has its own observer and is more accurate than the compass measurements from the DVL. The estimated attitude values are sent to the Kalman Filter in order to estimate the yaw rate, $\hat{r}$, as well as being used in the estimated position vector, $\hat{\boldsymbol{\eta}}$. Because of this the plot of the measured heading values is hidden behind the estimated values.

In Figure 6.2, the first station keeping period (10-90 seconds) has desired values in accordance to the logic mentioned above, i.e. station keeping of position at activation time for DP Mode. When a new reference position is given, a fault occurs in the desired roll value. This should have been set to zero. However, as both roll and pitch are not independently controlled due to the ROV thruster configuration, the fault never propagates and becomes a failure.

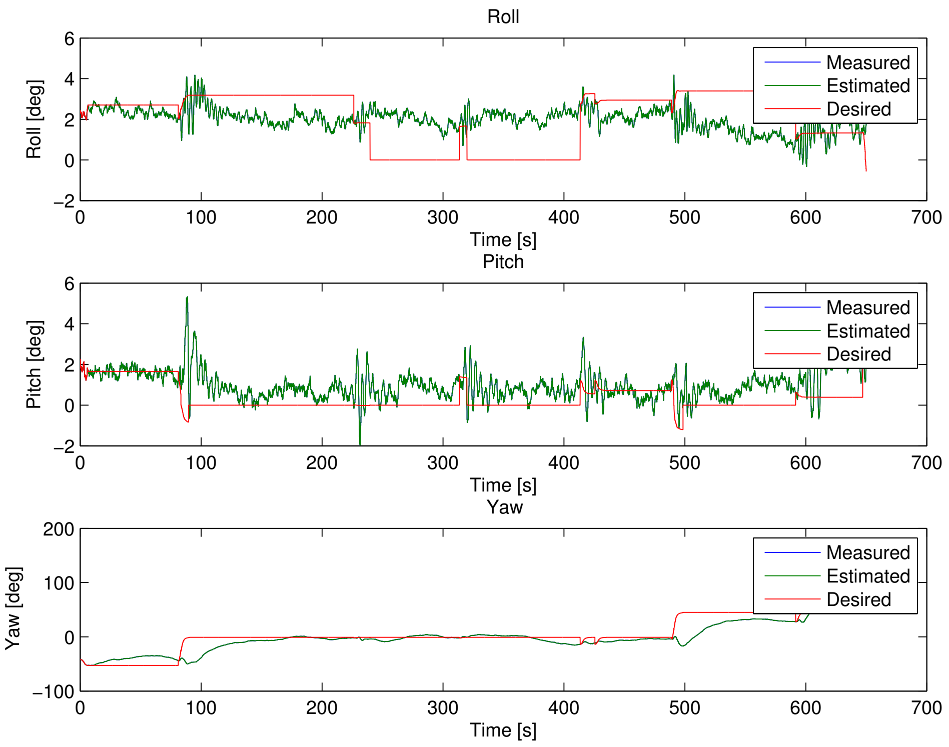


Figure 6.2: Plots of Rotational motion during DP test.

After about 680 seconds in Figure 6.1 and 6.2, or 50 seconds in Figure 6.3 and 6.4 manual thrust control with joystick was activated. These plots indicate that the observer manages to follow the measurements in a satisfactory manner, hence the Kalman Filter is working properly.

In Joystick Mode with Force control frame, the desired position values are set to the estimated values, $\boldsymbol{\eta}_d = \hat{\boldsymbol{\eta}}$. This results in a plot with only one graph for yaw angles as measured, estimated and desired values are the same.

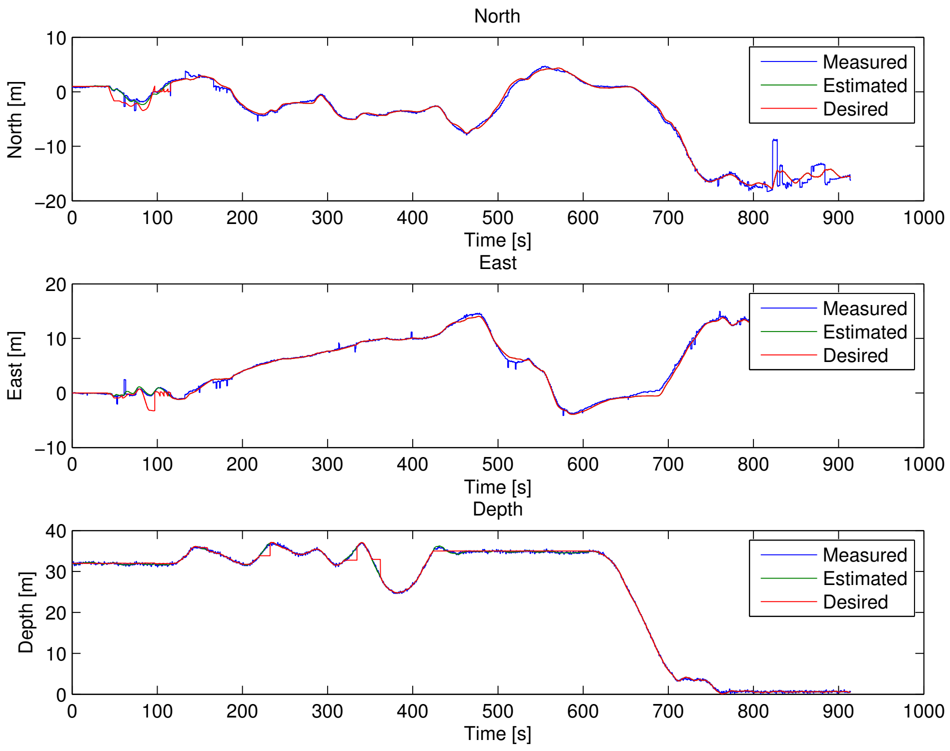


Figure 6.3: Plots of Translational motion during Direct Thrust Allocation Mode with Joystick.

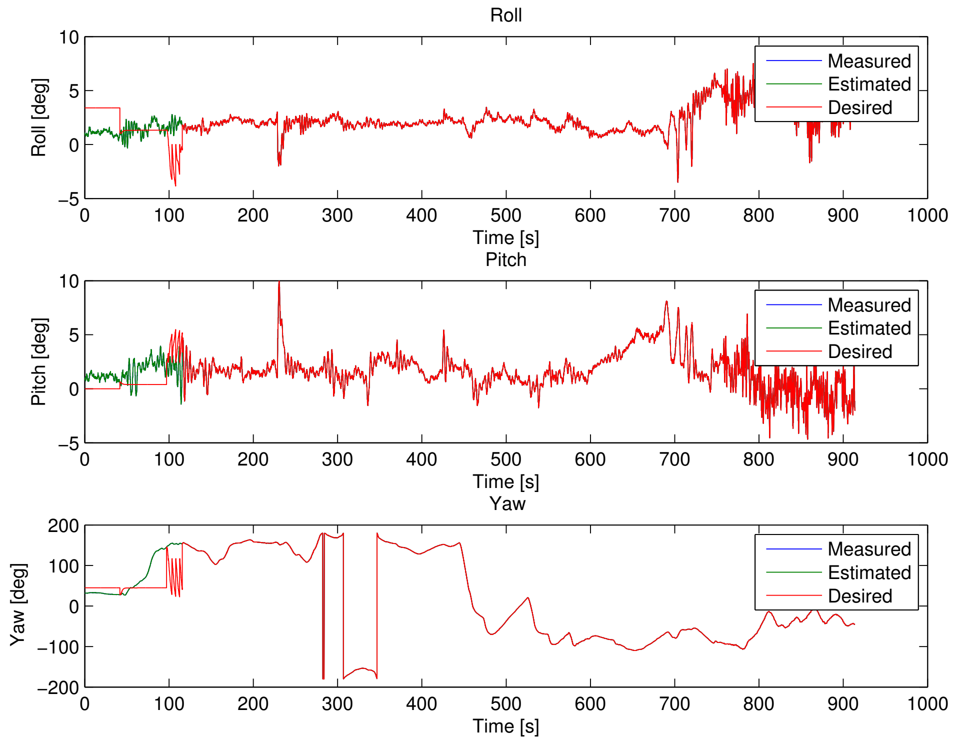


Figure 6.4: Plots of Rotational motion during Direct Thrust Allocation Mode with Joystick.

Chapter 7

Concluding Remarks

7.1 Conclusion

Development of computer-based control systems can be a complex and difficult process. The widen usage of computers in the control industry have led to new challenges for the control engineer not only in form of different faults and errors, but also the implementation procedure has changed. It is of major importance that the system is looked upon as a whole. By utilizing scrum or other software development procedure, solid communication among the developers is ensured. Software procedures also bring along the SRS such that there are no doubt on what the goal is.

It must be emphasized that the planning phase is one of the most important parts when developing new software systems. If a good development plan is deduced, then the implementation work will be easier to carry out. The same yields for testing. A test plan is strongly advised. Software testing is an own field of study, and has huge influence on the development phase. It is common to include the customers in the planning phase as they contribute with wanted specification and can help out with user-testing. During the beginning of the thesis the SRS (Chapter 3) was created in order to have a structured layout of what to be implemented later on. This document could with advantage, have been created in cooperation with both supervisor and co-advisors to ensure correct interpretation of the requirements.

In projects with low to none communication between the developers and the end-users, the customers usually end up with a system they do not understand, as well as lack of functionality of what they asked for. With scrum an increased connection between the developers and the end-users is guaranteed as the concept contains mandatory meetings between these groups. This helps the developers to understand what they are developing, as well as the customers get what they are paying for. The development process have not followed all of the scrum characteristics. However, an open dialog have been maintained between all involved parts to ensure a successful end result.

When creating systems that is planned for further development, documentation is needed in order to bring new developers up to speed. Documentation also helps interfacing different modules of the system. Adequate comments in the code is also helpful for newcomers as it will help telling what an algorithm does without having to investigate every detail. In accordance with the development of Njord, an Application Programming Interface (API)[42] have been created (Appendix A). The API is intended to be used as a reference book for future developers such that it is easy to find information about the different modules in the program. Hence, it is important that the API is updated together with the control system.

The implementation process for the complete system with both Njord and Frigg took about four months. The main challenge when implementing the control system from scratch was the lack of testing environments. This led to a huge testing process in the later part of the development phase, rather than conducting tests as functions and modules became finished. Frigg was however developed in parallel with the Njord. A server program was created to simulate the control system such that both communication and event handling could be tested.

A valid resource for error detection is usability testing. Co-students were asked to try out the user interface to see if they were able to operate the system. As the tester explored the system, they performed actions that the author did not foresee, which again led to an error of failure. Later usability tests were conducted with both SIL and HIL setup, where the control system also were tested. The usability tests uncovered errors and failures not detected by standard tests, hence helped to create a more stable and less faulty system.

The new software architecture have shown promising trends both in performance and implementation work. As the interface between the different sections in the program now are predefined, the integration of newly implemented modules require no modification in the control structure. This results in three steps the developer needs to take when implementing a new control theory module. The first step is to implement the equations and the logic required. Afterwards the configuration file must be updated to include the new tags needed in order to initialize the new module. Lastly in the third step reading methods for the new tags must be created in the “Read Configuration File”-method. The required knowledge of the complete system is then reduced, hence further development is easier.

There are no working tracking guidance implemented in the system as of June 2012. The intended tracking guidance strategy proved to be too complex, and tailor made for the old system. With more time, it is possible that the module could have been made to work.

Unfortunately there was only time for one sea trial with the new control system. However, the results were promising considering the circumstances. With one more day at the sea, it is believed that the performance of the DP operation with the

ROV 30k would be enhanced.

By the authors experience, LabView is not suited for development of heavy and complex systems such as the control system. During the development phase the Integrated Development Environment (IDE) crashed many times, often with no warnings or chance to save the work. However, LabView is a good language for inexperienced programmers as syntactical faults are easily detected by the IDE as well as the building blocks are accessible without the need of knowing obscured syntax.

Even though with the problems with LabView mentioned above, further development of Njord and Frigg is encouraged. Especially the architecture of Njord is believed to create a solid foundation, and even with advanced programming techniques used in the framework, implementation of new control theory modules can be done with ease. With some small modifications in the control structure, hybrid control can be achieved. Further, the potential for fault tolerant control and redundant software setup, combined with the user-friendly graphical interface, the complete systems usability is increased to a new level.

7.2 Recommended Further Work

Even though the system works in both DP operation and Joystick Mode, there are lots of work to be carried further. The most important aspect to fix, is the logic used in DP Guidance. The reference model has a strange behavior when moving from one point to another. It may be appropriate to change the whole guidance block, as the logic has been done in a cumbersome way. In addition, a weakness in the filter-based reference model used by the DP guidance strategy is that the integrators are coupled such that all DOFs must be within a limit in order to stop integrating.

Further it is recommended that a guidance system for Tracking will be implemented. An attempt to get the guidance strategy outlined in [33] working have been done by the author without success. The logics implemented in this guidance block is very complex and difficult to understand. Also here it is advised to revise the logic implementation, and try to make it easier.

As discussed in Section 4.6, the reliability of the control system can be increased by using redundant versions of the control system. By utilizing UDP/IP connection two or more instances of the control system can communicate with each other with “I am alive”-messages. If the active control system breaks down, the waiting one will step up and take over the control. This will make the DP system for the ROV more in accordance to a DP2 or DP3 classification[43] for ships. If this is to be implemented it is important to set up communication to all the instances of the control system from the user interface.

It is strongly advised to develop the Error handling further such that Njord is able to fix smaller problems on-line, without the need of a restart. It is recommended that Frigg is made fully fault tolerant as well, such that unnecessary crashes of the software can be avoided.

The 3D visualization of the ROV in Frigg can be further improved to include full thruster display and attitude orientation. This could later be extended to 3D map display. Mapping of the sea bed could also be a nice feature to include. Images of the sea bed could be stored in a database and accessed depending on the UTM coordinates.

The tuning panel can be expanded to include fast scalar tuning. By this it is meant that gains and matrices can be scaled by multiplying them with a number. This will create a fast and easy way to tune the different modules.

It is recommended to expand the library of controllers and observers such that the system can offer a wide range of configurations. This can be expanded to utilize hybrid control. Another alternative is to have tailor made configurations for different sea states, such that the control system can run with a supervisory control using the conditions as an input. A manual selector to override the decision should in such cases also be included. A wide selection of controllers and observers will also bring a professional feel to the whole system, and commissioning of ROVs will be closer to what is found in the ship industry.

Bibliography

[1] A. J. Sørensen. *Lecture Notes Marine Control Systems.* Department of Marine Technology, NTNU, UK-2011-76, 2011.

[2] E. J. Braude. *Software Engineering - An Object oriented Perspective.* Wiley and Sons, Inc., 2001.

[3] Ø. Olsen. *Praktisk Brukertesting.* Statistisk Sentralbyrå, 2006.

[4] M. Blanke, M. Kinnaert, J. Lunze, and M. Starowiecki. *Diagnosis and Fault-Tolerant Control.* Springer, 2003.

[5] F. Dukan, M. Ludvigsen, and A. J. Sørensen. Dynamic positioning system for a small size rov with experimental results. Technical report, Norwegian University of Science and Technology, 2011.

[6] M Kirkeby. Comparison of controllers for dynamic positioning and tracking of rov minerva. Master's thesis, Norwegian University of Science and Technology, 2010.

[7] NTNU. Applied underwater robotics laboratory. `http://www.ntnu.no/aur-lab`, 2011. [visited 14.12.2011].

[8] T. I. Fossen. *Handbook of Marine Craft hydrodynamics and Motion Control.* John Wiley and Sons Ltd., 2011.

[9] S. Bennett. Nicolas minorsky and the automatic steering of ships. *Control Systems Magazine*, 1984.

[10] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems.* Prentice Hall Inc., 1997.

[11] J. Daintith and E. Wright. *Oxford Dictionary of Computing.* Oxford University Press, 2008.

[12] S. Young. *Real Time Languages: Design and Development.* Ellis Horwood, 1982.

[13] B. Randell, J.-C Laprie, H. Kopetz, and B Littlewood. *Predictably Dependable Computing Systems.* Springer Verlag, 1995.

[14] A. Burns and A. Wellings. *Real-Time Systems and Programming Language 4th edt.* Addison-Wesley, 2009.

[15] K Schwaber and J. Sutherland. *The Scrum Guide.* Scrum.org, 2011.

[16] A. Pham and Pham P.-V. *Scrum in Action: Agile Software Project Management and Development.* Course Technology, 2012.

[17] IEEE Computer Society. *IEEE Recomended Practice for Software Requirements Specifications.* IEEE, 1998. reafirmed 2009.

[18] B. Shneiderman and C. Plaisant. *Designing the User Interface.* Pearson Addison-Wesley, 2010.

[19] A. J. Sørensen, Pedersen E., and Ø. Smogeli. Simulation-based design and testing of dynamically positioned marine vessels. In *MARSIM'03*, August 2003.

[20] J. Bélanger, P. Venne, and Paquin J.-N. The what, where and why of real-time simulation. *Planet-RT*, 10(2), 2010.

[21] E. Hvam. Brukergrensesnitt og menneske - maskin interaksjon. `http://www.sv.ntnu.no/psy/bjarne.fjeldsenden/TermPapers/EvaHvam.html#_Toc450895986`, 2000. [visited 12.12.2011].

[22] B. Randell, P. Lee, and P.C. Treleaven. Reliability issues in computing system design. *ACM Computing Surveys (CSUR)*, 10(2):123–165, 1978.

[23] Allen B. Downey. *The Little Book of Semaphores.* Free Software Foundation, 2008.

[24] E. W. Dijkstra. *Communicating Sequential Processes.* Prentice Hall International, 1985.

[25] J. Magee and J. Kramer. *Concurrency - State Models and Java Programs.* John Wiley and Sons, Inc., 2006.

[26] L. W. Lacy. *OWL: Representing Information Using the Web Ontology Language.* Trafford Publishing, 2004.

[27] Intel Corporation. Intel processor specifications. `http://www.intel.com/content/www/us/en/processor-comparison/processor-specifications.html?proc=43122`, 2009. [visited 18.05.2012].

[28] National Instruments. Using local and global variables carefully. `http://zone.ni.com/reference/en-XX/help/371361E-01/lvconcepts/using_local_and_global/`, 2008. [visited 18.05.2012].

[29] National Instruments. Are labview global variables good or bad, and when is it ok to use them? `http://zone.ni.com/devzone/cda/tut/p/id/5317`, 2011. [visited 13.12.2011].

[30] J. Strand and T. I. Fossen. Nonlinear passive observer design for ships with adaptive wave filtering. In H. Nijmeijer and T.I. Fossen, editors, *New Directions*

in nonlinear observer design, volume 244 of *Lecture Notes in Control and Information Sciences*, pages 113–134. Springer Berlin / Heidelberg, 1999. 10.1007/BFb0109924.

[31] S. Ø. Kørte. Guidance and control strategies for uuvs. Master's thesis, Norwegian University of Science and Technology, 2011.

[32] F. Dukan and A. J. Sørensen. Joystick in closed-loop control of rovs with experimental results. Technical report, Norwegian University of Science and Technology, Department of Marine Technology and Centre for Ships and Ocean Structures, 2012.

[33] D. A Fernandes, F. Dukan, and A. Sørensen. Reference model for higher performance and lower energy consumption in motions of marine crafts: Theory and experiments. *IEEE Journal of Oceanic Engineering*, 2011.

[34] National Instruments. compactrio. `http://www.ni.com/compactrio/`, 2011. [visited 14.12.2011].

[35] Kraft Robotic Science and Technology. Raptor force feedback manipulator. `http://krafttelerobotics.com/products/raptor.htm`, 2012. [visited 27.05.2012].

[36] Eiva A/S. About navipac. `http://download.eiva.dk/online-training/About%20NaviPac.htm`, 2012. [visited 27.05.2012].

[37] Eiva A/S. Eiva a/s home. `http://www.eiva.dk/`, 2012. [visited 27.05.2012].

[38] Teledyne RD Instruments. Workhorse navigator doppler velocity log. `http://www.rdinstruments.com/navigator.aspx`, 2012. [visited 01.06.2012].

[39] Kongsberg Maritime A/S. Hipap 501 - high precision acoustic positioning and underwater navigation system. `http://www.km.kongsberg.com/ks/web/nokbg0240.nsf/AllWeb/245AB9FBAACCD3C6C1256A7E00385C5C?OpenDocument`, 2012. [visited 01.06.2012].

[40] V. Berg. Development and commissioning of a dp system for rov sf 30k. Master's thesis, Norwegian University of Science and Technology, 2012.

[41] Xsense. Mti. `http://www.xsens.com/en/general/mti`, 2012. [visited 01.06.2012].

[42] Various. Application programming interface. `http://en.wikipedia.org/wiki/Application_programming_interface`, 2011. [visited 15.12.2011].

[43] Det Norkse Veritas A/S. *Special Equipment and Systems - Dynamic Positioning Systems*, pages 1–40. Det Norske Veritas A/S, 2011. Part 6, Section 7 in Rules for Classification of Ships.

Appendix A

API Njord

NORWEGIAN UNSIVERSITY OF SCIENCE AND TECHNOLOGY

Application Programming Interface

Njord

Espen Tolpinrud

07.06.2012

Revision: 1.0.0

This document is meant to be used as an Application Programming Interface (API) for future control system developers.

Table of Content

Introduction

It is recommended to have some basic knowledge of LabView and programming before reading this documentation.

Explanation of expressions

Cluster is a gathering of value types much like struct in other programming languages.

Matrix is a typedef 2D Array of Double
Complex Matrix is a typedef of 2D Array of Complex Numbers.

Objects

Vessel.lvclass

Labview Object.lvclass

∟ Vessel.lvclass

Implemented by Espen Tolpinrud.

The Vessel class is a representation of the craft in the control system.

Constructor summary

Name	Note
Vessel()	Creates a Vessel Object where the TCP port number is set to default, 8500.
Vessel(UInt 16 port)	Creates a Vessel Object where the TCP port number is set to be the given port
Vessel(FPGA VI Ref)	Creates a Vessel Object where the FPGA VI Ref is set. The TCP ports is set to default

Field summary

Data Type	Name	Note
1D Array of Observer.lvclass	Array Altitude Observer	Array of Observer objects for Altitude
1D Array of Controller.lvclass	Array Controllers	Array of Controller objects
1D Array of Observer.lvclass	Array Observer	Array of Observer objects for position, velocity and acceleration estimation.
Double	Buoyancy	The Buoyancy value for the vessel
1D Array of Double	CB	The Center of Buoyancy for the vessel (Maybe to be removed)
1D Array of Double	CG	The Center of Gravity for the vessel (Maybe to be removed)
Cluster of {UInt16,1D Array of Double}	Coordinate Reference	Contains the type of coordinate system to be used together with an array of offset values.
Matrix	Coriolis_RB	A Matrix with Coriolis effects for the vessel
Int32	DOF	The number of Degrees of Freedom in the system
Int32	DP Altitude Observer Index	The Index pointing to the observer object to be used from Array Altitude Observer during DP operation
Int32	DP Controller Index	The Index pointing to the controller object to be used from the Array Controllers during DP operation
Int32	DP Guidance Index	The Index pointing to the guidance object to be used from the Guidance Array during DP operation
Int32	DP Observer Index	The Index pointing to the observer object to be used from the Array Observers
1D Array of Int32	Drop Out Array	Array indicating healthy and unhealthy sensors.
Cluster of {1D	DVL	Cluster containing information from the Doppler

Array of Double, 1D Array of Int32}		Velocity Log - Value (1D Array of Double) - Signal (1D Array of Int32)
FPGA Interface	FPGA VI Reference	A Reference to the FPGA VI implemented on the NI compactRIO.
Double	Height	The height of the vessel (Maybe to be removed)
Int32	Joystick Guidance Index	The Index pointing to the guidance object to be used from the Guidance Array during Joystick Mode
Double	Length	The length of the vessel. (Maybe to be removed)
Matrix	Linear Damping	Matrix containing linear damping values for the vessel
Matrix	Mass_A	Matrix containing the Added Mass effects for the vessel
Matrix	Mass_RB	Matrix containing the Rigid Body Mass for the vessel
1D Array of Double	Max Thrust	Array with maximum thrust in each degree of freedom for the vessel.
1D Array of Double	Max_Acc	Array with maximum acceleration defined in body coordinates in each degree of freedom for the vessel.
String	Name	Name of the ROV
Matrix	NonLinear Damping	Matrix containing nonlinear damping values for the vessel.
1D Array of Double	Nu_max	Array with maximum velocity defined in body coordinates in each degree of freedom for the vessel.
1D Array of Cluster of {Int16, Int8, Int8, Int8, Int8}	Port Configuration	Array with communication information used to communicate with the NI compactRIO. The cluster contains the fields: - Baud Rate (Int16) - Data Bits (Int8) - Parity (Int8) - Stop Bits (Int8) - Flow Control (Int8)
Double	Sampling Time	The sampling time for the control loop. Determines the frequency of iteration.
Cluster of 4 {Cluster of {1D Array of Double, 1D Array of Double, 1D Array of Double, 1D Array of Double}, Cluster of { Double, Double, Double, Double, Double}, 1D Array of Double, 1D Array of Double, Cluster of {4 1D Array of	States	Contains the different states for the system - Measured (Cluster) o Eta (1D Array of Double) o Nu (1D Array of Double) o Eta_dot (1D Array of Double) o Nu_dot (1D Array of Double) - Estimated (Cluster) o Eta (1D Array of Double) o Nu (1D Array of Double) o Eta_dot (1D Array of Double) o Nu_dot (1D Array of Double) - Desired (Cluster) o Eta (1D Array of Double) o Nu (1D Array of Double) o Eta_dot (1D Array of Double)

Double}}		o Nu_dot (1D Array of Double)
		- Previous Desired (Cluster)
		  o Eta (1D Array of Double)
		  o Nu (1D Array of Double)
		  o Eta_dot (1D Array of Double)
		  o Nu_dot (1D Array of Double)
		- Tau (1D Array of Double)
		- Rpm (1D Array of Double)
		- Altitude (Cluster)
		  o Measured
		  o Estimated
		  o Desired
		  o Previous Desired
		  o Approximated
		- Attitude
		  o Theta Measured
		  o Theta Estimated
		  o Omega Measured
		  o Omega Estimated

Supervisor.lvclass	Supervisor	A Supervisor object used to communicate with a user interface and ensure correct behavior in the system.
Thrust Allocation.lvclass	Thrust Allocation	A Thrust Allocation object used to keep track of the thrusters for the vessel
Int32	Tracking Altitude Observer Index	The Index pointing to the observer object for altitude to be used from the Array Observer Altitude during Tracking
Int32	Tracking Controller Index	The Index pointing to the controller object to be used from the Array Controllers during Tracking
Int32	Tracking Guidance Index	The Index pointing to the guidance object to be used from the Guidance Array during Tracking
Int32	Tracking Observer Index	The Index pointing to the observer object to be used from the Array Observers during Tracking
1D Array of Sensor.lvclass	Used Sensors	Array containing all sensors used by the vessel.
Double	Width	The width of the vessel. (Maybe to be removed)

Methods summary

Data Type	**Name with input arguments**	**Note**
Void	Add Element in Array Altitude Guidance{Guidance.lvclass new Altitude Guidance	Inserts a new altitude guidance object in the array.

Int32	Add Element in Array Altitude Observer (Observer.lvclass new Altitude Observer)	Inserts a new Altitude Observer object in the Array Altitude Observer and returns the index of the new element.
Int32	Add Element in Array Controller (Controller.lvclass new Controller)	Inserts a new Controller object in the Array Controllers and returns the index of the new element.
Int32	Add Element in Array Guidance (Guidance.lvclass new Guidance)	Inserts a new Guidance object in the Array Guidance and returns the index of the new element.
Int32	Add Element in Array Observers (Observer.lvclass new Observer)	Inserts a new Observer object in the Array Observers and returns the index of the new element.
Void	Add Element in Available Sensors (Sensor.lvclass new Sensor	Inserts a Sensor object in the Available Sensors array
Void	Add Element in Used Sensors (Sensor.lvclass new Sensor)	Inserts a Sensor object in the Used Sensors array.
Void	Control Structure ()	Performs an iteration of the control loop. Must be placed in an outer loop structure in order to work properly.
Void	Discrete to Continuous (Void)	Transforms the angels in estimate and desired states from discrete to continuous.
Int32	Find Controller by Name (String name)	Finds the index of the controller with the specified name in the controller array. If not present the method returns -1.
Int32	Find Observer by Name (String name)	Finds the index of the observer with the specified name in the observer array. If not present the method returns -1.
Int32	Find Sensor by ID (String Sensor ID)	Searches through the Used Sensors array and returns the Index of the desired Sensor, or -1 if no match is found.
Observer.lvclass	Get Active Altitude Observer ()	Returns the Altitude Observer object for the active task. Default task is DP operation.
Controller.lvclass	Get Active Controller ()	Returns the Controller object for the active task. Default task is DP operation.
Guidance.lvclass	Get Active Guidance ()	Returns the Guidance object for the active task. Default task is DP operation.
Observer.lvclass	Get Active Observer ()	Returns the Observer object for the active task. Default task is DP operation.
Void	Measurement Substitution (Void)	Substitutes the angle measured state with the ones from MTi.
Void	Read Configuration File ()	Initializes the invoker and makes the system ready for execution.

4 1D Array of Double	Read FPGA FIFO ()	Reads the FIFO stack on the NI compactRIO and return arrays with the sensor values. NB! This method is only to be used together with a NI compactRIO unit. Use “Read ROV String_SIL” for non NI compactRIO setup. Return values is: - MRU - MT_DATA - Signals out - Dvl_m
2 1D Array of Double	Read ROV String_SIL (String ROV string, String NAVI string)	Reads the sensor strings and returns the sensor values. This is a substitution for the Read FPGA FIFO () for setup with no NI compactRIO setup. Return Values is: - Signals out - Dvl_m
Void	Signal Processing (1D Array of Double MRU, 1D Array of Double MT_DATA, 1D Array of Double Signals in, 1D Array of Double dvl_m)	Translates the sensor values into position and velocity measurements. The Sensors health is checked. State variables are set in the invoking vessel object.
Void	Store Active Altitude Observer (Observer.lvclass observer)	Replaces the Altitude Observer object for the active task. Updates will then be carried on in the system.
Void	Store Active Controller (Controller.lvclass controller)	Replaces the Controller object for the active task. Updates will then be carried on in the system.
Void	Store Active Guidance (Guidance.lvclass guidance)	Replaces the Guidance object for the active task. Updates will then be carried on in the system.
Void	Store Active Observer (Observer.lvclass observer)	Replaces the Observer object for the active task. Updates will then be carried on in the system.
Void	Write to cRIO ()	Create and write the telebuf to the NI compactRIO FIFO stack. NB! To not use this unless a NI compactRIO unit is connected. Use “Write to cRIO_SIL” instead.
1D Array of UInt8	Write to cRIO_SIL ()	Create the telebuf and send it as output. This is a substitute for the “Write to cRIO”.

Supervisor.lvclass

Labview Object.lvclass
∟ Supervisor.lvclass

Implemented by Espen Tolpinrud.

The Supervisor object works as a communication administrator and execution flow manager.

Constructor Summary

Name	Note
Supervisor()	Default constructor to create a supervisor object.

Field Summary

Data Type	Name	Note
Boolean	Auto Depth	Used to give message about running auto depth in joystick control.
Boolean	Auto Depth Tele	Used to give message about running auto depth in joystick control. Message from ROV control panel.
Boolean	Auto Heading	Used to give message about running auto heading in joystick control. Message from Joystick.
Boolean	Auto Heading Tele	Used to give message about running auto heading in joystick control. Message from ROV control panel.
Cluster of { Boolean, Int8, Double, Double, Double, Double, Double}	Altitude	Cluster containing: - Altimeter Active (Boolean) - Beam Select (Int8) - Measured (Double) - Estimated (Double) - Desired (Double) - Previous Desired (Double) - Approximated (Double)
Cluster of { Boolean, Boolean, Boolean, Int32 }	Altitude Cluster	Cluster containing: - Use Integrator (Boolean) - Use FeedForward (Boolean) - Use Reference Model (Boolean) - Desired Altitude
Cluster of { 4x 1D Array of Double}	Attitude	Cluster containing: - Theta Measured (1D Array of Double) - Theta Estimated (1D Array of Double) - Omega Measured (1D Array of Double) - Omega Estimated (1D Array of Double)
Cluster of { Boolean}	Attitude Cluster	Cluster containing: - Initialize (Boolean)

Cluster of { Int32, Boolean, Boolean, Boolean, Boolean}	Camera Control	Cluster containing: - Camera Number (Int32) - Camera Up (Boolean) - Camera Down (Boolean) - Camera Left (Boolean) - Camera Right (Boolean)
Cluster of { 4x Boolean}	Collecting Unit	Contains: - In (Boolean) - Out (Boolean) - Rotate CW (Boolean) - Rotate CCW (Boolean)
UInt16	Control Mode	Selector for the different control modes the vessel can operate in during Joystick mode.
Boolean	Control System Active	Flag used to determine if the control loop is to be run.
UInt16	Coordinate System	Selector for the different coordinate reference systems the vessel can operate in.
Cluster of {1D Array of Double, 1D Array of Double, 1D Array of Double, 1D Array of Double}	Desired	Desired State values from the vessel.
1D Array of Double	Dvl data	Sensor values from the Doppler Velocity Log.
String	Error Msg	Contains an Error Message from the system, if everything is OK, it should say “OK”.
Cluster of {1D Array of Double, 1D Array of Double, 1D Array of Double, 1D Array of Double}	Estimated	Estimated State values from the vessel.
Boolean	Exit Flag	Change when user interface demands it
UInt16	Frame Mode	Selector for the different frame modes the vessel can operate in during joystick mode

Cluster of { Boolean, Boolean, Int8, Boolean, Boolean, Cluster of {Double, Double, Double, Double}, Cluster of {Boolean, 1D Array of Double}}	Guidance Cluster	Cluster contains: - Set Origin (Boolean) - New WP (Boolean) - Types of Change (Int8) - Make Change (Boolean) - Optimal Heading (Boolean) - New SetPoint o North o East o Depth o Heading - Update Reference o Mode Change o Current Position
Int32	Iteration Number	The iteration number for the control loop

Cluster of {Cluster of {Double, Double, Double}, Cluster of {Double, Double, Double}}	Joystick Axis	Cluster with joystick axis inputs.
		Contains:
		- Axis Translation
		  o X-Axis
		  o Y-Axis
		  o Z-Axis
		- Axis Rotation
		  o X-Axis
		  o Y-Axis
		  o Z-Axis

Cluster of {Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Int32}	Joystick Commands	Cluster with joystick button inputs. Contains: - Button 1 (Boolean) - Button 2 (Boolean) - Button 3 (Boolean) - Button 4 (Boolean) - Button 5 (Boolean) - Button 6 (Boolean) - Button 7 (Boolean) - Button 8 (Boolean) - Button 9 (Boolean) - Button 10 (Boolean) - Button 11 (Boolean) - Button 12 (Boolean) - POV Direction (Int32)

Cluster of { 2 Double Matrix, 2 Boolean}	Kalman Filter Observer Tuning	Contains new tuning values for the Kalman Filter observer. Elements are: - R (Double Matrix) - Q (Double Matrix) - Update R (Boolean) - Update Q (Boolean) When the Booleans are set to true the corresponding matrices in the Non Linear PID object are updated to contain the values from the matrices in this cluster.
Cluster of {Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Int32, Int32}	Light Settings	Cluster with Light control inputs. Contains: - Light 1 - Light 2 - Light 3 - Light 4 - HMI 1 - HMI 2 - Light Intensity 1 - Light Intensity 2
Cluster of {Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean, Boolean }	Manipulator Control	Cluster with Manipulator control inputs. Contains: - Power Switch (Boolean) - Upper Arm Up (Boolean) - Upper Arm Down (Boolean) - Lower Arm Up (Boolean) - Lower Arm Down (Boolean) - Arm Left (Boolean) - Arm Right (Boolean) - Claw Open (Boolean) - Claw Close (Boolean) - Rotate Clockwise (Boolean) - Rotate Counterclockwise (Boolean)
Cluster of {1D Array of Double, 1D Array of Double, 1D Array of Double, 1D Array of Double}	Measured	Measured State values from vessel.
1D Array of Single	MT_DATA	Contains values from the MT sensor. Used in Attitude Observer.
Boolean	MTi Active	Flag to determine if MTi Observer shall run
Boolean	MTi Gyro	Flag to determine if MTi Gyro sensor shall be used.
Boolean	New WP List	Flag to notify about new WP List.

Cluster of { 4 Double Matrix, 4 Boolean}	Non Linear PID Tuning	Contains new tuning values for the Non Linear PID Controller. Elements are: - KP (Double Matrix) - KD (Double Matrix) - KI (Double Matrix) - KA (Double Matrix) - Update KP (Boolean) - Update KD (Boolean) - Update KI (Boolean) - Update KA (Boolean) When the Booleans are set to true the corresponding matrices in the Non Linear PID object are updated to contain the values from the matrices in this cluster.
Boolean	Only New HiPAP	Flag to determine if only new HiPAP measurements are to be sent to the observer.
UInt16	Operation Type	Selector which keeps track of which mode is active for the vessel the supervisor object belongs to.
Cluster of {Double, Double}	Origin	Represents the UTM coordinates for a desired origin. Contains: - North (Double) - East (Double)
Cluster of { 10x Boolean}	Power Instruments	Contains - HPR (Boolean) - Sonar 1 (Boolean) - Sonar 2 (Boolean) - Doppler (Boolean) - Laser (Boolean) - KRAFT (Boolean) - Pan/Tilt (Boolean) - Transponder (Boolean) - Still Cam (Boolean) - HD Cam (Boolean)
Boolean	Reset Controller	Flag to reset integrators in the controllers.
Boolean	Reset Observer	Flag to reset the values in the observers.
Boolean	Reset ROV	Flag to set the desired position to current estimated position.
Boolean	Reset SP	Flag used to notify if signal processing is to be reset.
1D Array of Double	RPM	RPM values for the vessel the supervisor object belongs to.
Cluster of {1D Array of Int32, 1D Array of Boolean}	Sensor Status	Contains: - Sensor Drop Out (1D Array of Int32) - Sensor Health (1D Array of Boolean)
2D Array of Double	Table of WP	An array with the waypoints to be used in tracking.
1D Array of Double	Tau_scaled	A percentage representation of the desired thrust in each Degree of Freedom.
1D Array of String	Thruster Names	A string array with the thruster names.

Cluster of{Cluster of {Double, Double, Double, Double, Double, Double, Double, Double}, 2D Array of Double, Cluster of { Double, Double, Double, Double, Double, Double, Double, Double, Double}, UInt16, Boolean, Boolean, Boolean, Boolean}	Tracking Cluster	A Cluster with tracking performance settings. Contains: - Performance Behavior o % Velocity max lin to exp (Double) o % Velocity zero lin to exp (Double) o % Total distance at max velocity (Double) o Max Angular Velocity (Double) o Time to max linear velocity (Double) o Min distance at max velocity (Double) o Max linear velocity (Double) o Time to max angular velocity (Double) - Table of WP (2D Array of Double) - Tracking Options o Superior Bound (Double) o Inferior Bound (Double) o Duration No operation period between motions (Double) o Tolerance of initial angle (Double) o Distance to activation waiting function (Double) o Distance to deactivation waiting function (Double) o (2) Fixed Heading (Double) o (3) PoI N (Double) o (3) PoI E (Double) - Heading Mode (UInt16) - Start (Boolean) - Use Position (Boolean) - Pause (Boolean) - Abort (Boolean)
Boolean	Update Ports	A Boolean to notify the system about new port configurations.
1D Array of Cluster of {Int16, Int8, Int8, Int8, Int8}	Updated Port Configurations	Array with communication information used to communicate with the NI compactRIO. The cluster contains the fields: - Baud Rate (Int16) - Data Bits (Int8) - Parity (Int8) - Stop Bits (Int8) - Flow Control (Int8)
Boolean	Use Altitude Guidance	A Boolean to notify the system about using altitude guidance.
Boolean	Use Chair Joystick	A Boolean to notify the system about using the chair joystick instead of the one connected to the host pc. Yields only for 30k.

Method Summary

Data Type of Return	Name with input arguments	Note
TCP Network Reference	Receive TCP Message (TCP Network Reference ref)	Receive run-time commands through the TCP protocol. Usually from a user interface.
TCP Network Reference	Receive TCP Message Joystick (TCP Network Reference ref)	Receive Joystick commands through the TCP protocol and fills out Joystick Axis and Joystick Commands. Usually from a user interface.
TCP Network Reference	Send TCP Message (TCP Network Reference ref)	Sends status messages through the TCP protocol, usually to a user interface.

Method Details

Receive TCP Message (TCP Network Reference ref)

This method is used to listen for new inputs from the user interface. Each message contains a code and a message body.

Code	Message Body	Note
1:	1	Exit message to the system.
2:	Double North,Double East,Double Depth,Double Heading	Notification of new DP position. The coordinates should be given in NED coordinates with Heading in radians.
3:	Integer nWP,Double wp1index,Double wp1n,Double wp1e,Double wp1d;…;Double wpNindex,Double wpNn,Double wpNe,Double wpNd;	Notification of tracking operation. The whole waypoint list is sent starting with the number of waypoints, N, and then continuing with N sets of NED coordinates.

4:	3charString code,Integer value	Message for Manipulator control on the ROV. The code has to consist of 3 and only 3 characters! The valid codes are:
		- PSM (Power Switch Manipulator)
		- UA (Upper Arm)
		    o U (Up)
		    o D (Down)
		- LA (Lower Arm)
		    o U (Up)
		    o D (Down)
		- MA (Manipulator Arm)
		    o R (Right)
		    o L (Left)
		- CL (Claw)
		    o C (Close)
		    o O (Open)
		- AR (Arm Rotation)
		    o L (Counterclockwise)
		    o R (Clockwise)
		The integer values should be either 0 (for false) or 1 (for true). The true condition can however be set to any integer not equal to 0.

5:	2charString code,Integer value	Message for Camera settings on the ROV. The code has to consist of 2 and only 2 characters! The valid codes are: - CN (Camera Number) - CU (Camera Up) - CD (Camera Down) - CL (Camera Left) - CR (Camera Right) The integer value should be for all codes except CN be 0 (for false) or 1 (for true). For CN the integer value should be a whole number.

6:	2charString code,Integer value	Message for Light settings on the ROV. The code has to consist of 2 and only 2 characters! The character at the back must be an integer. The valid codes are: - L (Light) o 1 o 2 o 3 o 4 - H (Hydrargyrum medium-arc iodide light) o 1 o 2 - I (Light Intensity) o 1 o 2 The integer value should for all codes except for I be 0 (for false) or 1 (for true). For I the value should be a whole number between 0 and 127.
7:		Joystick Command String. (See details below)
8:	3charString,Double value	Message for changing options for Tracking
9:	1charString,Integer	Message for Pause or Abort Tracking.
10:	8x[Integer PortNumber,Integer BaudRate,Integer DataBits,Integer Parity,Integer StopBit,Integer FlowControl;]	Message for configuration of port connections between the system and the compactRIO. 8 port configurations are sent from the user interface.
11:	Integer value	Notification to the system to set new origin point. The integer value should be 0 (for false) or 1 (for true).
12:	Integer value	Message used to switch between using estimated and measured values in the controller.
13:	Integer value	Message used to switch frame mode during joystick operations. The integer value points to a selector position.
14:	Integer value	Message used to switch control mode during joystick operations. The integer value point to a selector position.
15:	Integer value	Message to notify the system to start the control loop. The integer value should be 0 (for false) or 1 (for true).
16:	Integer value	Message to notify the system to reset the observer. The integer value should be 0 (for false) or 1 (for true).
17:	Integer value	Message to notify the system to reset the controller. The integer value should be 0 (for false) or 1 (for true).
18:	Integer value	Message to notify the system to reset the ROV. The integer value should be 0 (for false) or 1 (for true). The controller is also reset by this call. The desired position is set to be the estimate.

19:	3charString,[1charString:1charString] [MatrixValuesString]	Message to notify the system to update the tuning matrices for a controller or observer. Which module to be updated, is given in the message.
20:	Integer Value	Message to notify the system to activate/deactivate the MTi. The integer value should be 0 (for false) or 1 (for true).
21:	Integer Value	Message to notify the system to activate/deactivate the MTi Gyro. The integer value should be 0 (for false) or 1 (for true).
22:	Integer Value	Message to notify the system to activate/deactivate the auto tuning of Omega used by the reference model. The integer value should be 0 (for false) or 1 (for true).
23:	Empty message body	Activates the joystick mode in force frame. Used as a fail-safe mechanism.
24:	Integer Value	Message to notify the system to activate/deactivate the use of the joystick on the chair (for 30k only). The integer value should be 0 (for false) or 1 (for true).
25:	Integer Value	Message to notify the system to activate/deactivate the HPR. The integer value should be 0 (for false) or 1 (for true).
26:	Integer Value	Message to notify the system to activate/deactivate the Sonar 1. The integer value should be 0 (for false) or 1 (for true).
27:	Integer Value	Message to notify the system to activate/deactivate the Sonar 2. The integer value should be 0 (for false) or 1 (for true).
28:	Integer Value	Message to notify the system to activate/deactivate the Doppler Velocity Log. The integer value should be 0 (for false) or 1 (for true).
29:	Integer Value	Message to notify the system to activate/deactivate the Laser. The integer value should be 0 (for false) or 1 (for true).
30:	Integer Value	Message to notify the system to activate/deactivate the KRAFT Manipulator Arm Control. The integer value should be 0 (for false) or 1 (for true).
31:	Integer Value	Message to notify the system to activate/deactivate the Pan/Tilt. The integer value should be 0 (for false) or 1 (for true).
32:	Integer Value	Message to notify the system to activate/deactivate the Transponder. The integer value should be 0 (for false) or 1 (for true).
33:	Integer Value	Message to notify the system to activate/deactivate the Still Camera function. The integer value should be 0 (for false) or 1 (for true).

34:	Integer Value	Message to notify the system to activate/deactivate the HD Camera. The integer value should be 0 (for false) or 1 (for true).
35:	2charString,Integer Value	Message to notify the system to activate/deactivate the Collecting Unit functions. The integer value should be 0 (for false) or 1 (for true). Codes are: - OU – Out - IN – In - CC – Counter Clockwise - CW - ClockWise
36:	Integer Values	Message to notify the system to activate/deactivate the “Only New HiPAP”. The integer value should be 0 (for false) or 1 (for true).
37:	Integer Value	Message to notify the system to reset the Signal Processing. The integer value should be 0 (for false) or 1 (for true).
38:	Empty Message body	Notifies the system to send a list of all available controllers and observers to the user interface.

Receive TCP Message Joystick (TCP Network Reference ref)

This method is used to receive joystick commands from the user interface. This has been given its own method due to frequent message receiving. By running this method in a separate thread concurrent with Receive TCP Message () no user inputs are ignored.

Code	Message Body	Note
7:	Integer value,Double X,Double Y, Double Z, Double ZRot,Integer Buttons,Integer POV	The integer value is a numeric representation of a Boolean value of whether or not the joystick is in use or not. X is the X-Axis position, Y is the Y-Axis position, Z is the Z-Axis position and the ZRot is the Z-Axis Rotation. The integer Buttons is a numeric representation of a Boolean array and has a value of $\sum_{i=0}^{k} 2^{i}$ where k is the number of buttons on the joystick. Each button combination then has its own unique value and no button pressed is set to 0. The POV integer gives the direction angle in degrees from 0 to 360 with step of 45 degrees. No direction is defined as -1.

Send TCP Message (1D Array of TCP Network Reference ref)

This method builds and sends a string of data to the user interface. There are separate TCP Network References for each type of data set to be sent.

The different data sets are:

- Position states
- Velocity states
- Thruster values and RPMs
- Altitude and dvl sensor
- MTi data (Not implemented yet)
- Other sensors can be added.

Thrust Allocation.lvclass

LabView Object.lvclass

∟ Thrust Allocation.lvclass

Implemented by Viktor Berg.

Constructor Summary

Name	Note
Thrust Allocation()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
1D Array of Thruster.lvclass	Array Thrusters	Array of the thruster objects that belongs to the invoking thrust allocation object.
Matrix	Thrust Allocation Matrix	A matrix with thruster impact in each Degree of Freedom. Each row represents Degree of Freedom, and the columns are the thrusters. Also called T.
Matrix	Thrust Coefficients Matrix	A Matrix with the relationship between thrust in force and thruster RPMs. Also called K.

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Thrust(Vessel.lvclass vessel)	Converts the desired thruster forces and torques to thruster RPMs with the expression $u = K^{-1}T^{\dagger}\tau$
Void	Insert Thruster(Thruster.lvclass thruster, Int32 Index)	Insert a new thruster object in the Array Thruster. The thruster impact in each Degree of Freedom is calculated and added to the Thrust Allocation Matrix.
Void	Thrust Coefficients (1D Array of Double rpm)	The method calculates the Thrust Coefficients to be used by the Calculate Thrust method.
Void	Write Col in Thrust Allocation Matrix (	This is a private method for the Thrust Allocation object and is used to add a column in the Thrust Allocation Matrix.

Thruster.lvclass

LabView Object.lvclass

∟ Thruster.lvclass

Implemented by Espen Tolpinrud.

Constructor Summary

Name	Note
Thruster()	Default constructor to create an instance of the class.
Thruster (1D Array of Double input, String Type, String Name)	Constructor that fully initialize the object.

Field Summary

Data Type	Name	Note
1D Array of Double	Coefficients	Array with coefficients for the relationship between Force and RPM. Contains both for positive and negative RPM, and is made for second order relationship.
Double	Max RPM	A matrix with thruster impact in each Degree of Freedom. Each row represents Degree of Freedom, and the columns are the thrusters. Also called T.
String	Name	A Matrix with the relationship between thrust in force and thruster RPMs. Also called K.
1D Array of Double	Position Vector	The position vector represents the thruster position in body coordinates. First index is x, second y, third z, fourth rotation about z-axis, fifth rotation between x-y plane and z-axis. This is similar to spherical coordinates.
1D Array of Double	Rotation Vector	If thruster is Azimuth, this vector tells which axis to rotate about.
UInt16	Thruster Type	Selector of thruster type. Yields for fixed and Azimuth.

Method Summary

None

Sensor.lvclass

LabView Object.lvclass
∟ Sensor.lvclass

Implemented by Espen Tolpinrud.

This class may be unnecessary at the moment. Removal of the class is suggestible.

Constructor Summary

Name	Note
Sensor()	Default constructor to create an instance of the class.
Sensor (1D Array of String input)	Constructor that fully initialize the object.

Field Summary

Data Type	Name	Note
String	ID	The Sensor ID
String	Name	The Name of the sensor
String	PosX	The x position of the sensor from Center of Origin
String	PosY	The y position of the sensor from Center of Origin
String	PosZ	The z position of the sensor from the Center of Origin
String	RotX	The x rotation of the sensor
String	RotY	The y rotation of the sensor
String	RotZ	The z rotation of the sensor
String	Type	The type of the sensor.

Method Summary

Data Type of Return	Name with input arguments	Note
Boolean	Check Health (Int32 Drop Out)	Checks the health of the invoking sensor. NB! To be developed further.
Void	WriteAllData (1D Array of String)	Fully Initialize the object. Is used in Sensor(1D Array of String input)

Observer.lvclass

LabView Object.lvclass

∟ Observer.lvclass

Implemented by Espen Tolpinrud.

This is a template or abstract class for all observers and should never be used stand alone.

Constructor Summary

Name	Note
Observer()	Default constructor to create an instance of the class.
Observer(String input)	Constructor that sets the name of the Observer object.

Field Summary

Data Type	Name	Note
String	Observer Name	The name of the observer

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	EstimateStates (Vessel.lvclass vessel)	A method that every child of Observer.lvclass must override. The method calculates the estimated states for the system.

Kalman Filter.lvclass

LabView Object.lvclass
    ∟ Observer.lvclass
        ∟ Kalman Filter.lvclass

Implemented by Viktor Berg.

Constructor Summary

Name	Note
Kalman Filter()	Default constructor to create an instance of the class.
Kalman Filter (Vessel.lvclass vessel)	Constructor that initializes PHI, GAMMA, and DELTA.

Field Summary

Data Type	Name	Note
Matrix	DELTA	Sector matrix for Kalman Filter.
Matrix	GAMMA	Sector matrix for Kalman Filter.
Matrix	PHI	Sector matrix for Kalman Filter.
Matrix	Q	Tuning matrix for Kalman Filter.
Matrix	R	Tuning matrix for Kalman Filter.

Method Summary

Data Type of Return	Name with input arguments	Note
2D array of Int32	Drop Out Converter (1D Array of Int32 Drop Out)	Transform the drop out vector to a measurement matrix. May not be needed anymore.
Vessel.lvclass	EstimateStates (Vessel.lvclass vessel)	Estimate states by using predictors and Kalman Filter equations.
3 1D Array of Double	Init Observed Values	Set up the measured values to be given the Kalman Filter. The output is: - y (1D Array of Double) - y_last (1D Array of Double) - tau (1D Array of Double) Note to developer: May need revising.
5 Matrix	Read Q and R	Returns DELTA, GAMMA, PHI, Q and R.

Methods Details

EstimateStates(Vessel.lvclass vessel)

//TODO: Insert equations used to calculate the estimated states.

Kalman Filter Altitude.lvclass

LabView Object.lvclass
    ∟ Observer.lvclass
        ∟ Kalman Filter.lvclass
            ∟ Kalman Filter Altitude.lvclass

Implemented by Espen Tolpinrud.

Constructor Summary

Name	Note
Kalman Filter Altitude()	Default constructor to create an instance of the class.

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	EstimateStates (Vessel.lvclass vessel)	Estimates the altitude of the vessel.

Method Details

EstimateStates (Vessel.lvclass vessel)

// TODO: Insert equations for estimating the states.

Passive Observer.lvclass

LabView Object.lvclass

∟ Observer.lvclass

∟ Passive Observer.lvclass

Implemented by Espen Tolpinrud.

Constructor Summary

Name	Note
Passive Observer()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
Matrix	Aw	System matrix for wave filter.
Matrix	Cw	Measurement matrix for wave filter.
Matrix	Delta	Wave damping matrix. Typical values are $\zeta_{ni} = 1.0$
Matrix	K1	Tuning matrix for wave filter. Typical values are $K_{1i}(\omega_{oi}) = -2(\zeta_{ni} - \lambda_i)\frac{\omega_{ci}}{\omega_{oi}}, \forall i = 1..6$ $K_{1(i+6)}(\omega_{oi}) = 2\omega_{oi}(\zeta_{ni} - \lambda_i), \forall i = 1..6$
Matrix	K2	Tuning matrix for observer. Typical values are $K_{2i} = \omega_{ci}$
Matrix	K3	Tuning matrix for bias estimator. Typical values $K_3 = 0.1K_4$
Matrix	K4	Tuning matrix for observer.
1D Array of Double	Lambda	Tuning variables to determine the notch. Typical value is $\lambda_i = 0.1$
Matrix	Omega	The diagonal elements are the frequency of the time periods for the waves. $\Omega = diag\left\{\frac{2\pi}{T_i}, \forall i = 1..6\right\}$ where typical values for $T_i = 5..20\ seconds$
1D Array of Double	Omega_cutoff	The cutoff frequency for the wave period. Must be larger than Omega.
Matrix	T	Time constants for the bias estimator.

Method Summary

Data Type of Return	Name with input arguments	Note
1D Array of Double	Bias Estimator (1D Array of Double $\tilde{y}$, Double dt)	Estimates the bias.
Vessel.lvclass	EstimateStates (Vessel.lvclass vessel)	Estimates the states for the system using the equations for the passive observer. Have both bias estimator and wave filter. Wave filter is optional.

Void	Initialize Matrices ()	Initializes the system matrices using tuning expressions.
1D Array of Double	Wave Estimator (1D Array of Double $\tilde{y}$, Double dt)	Wave filter for the passive nonlinear observer.

Method Details

Bias Estimator (1D Array of Double $\tilde{y}$, Double dt)

EstimateStates (Vessel.lvclass vessel)

Initialize Matrices ()

Wave Estimator (1D Array of Double $\tilde{y}$, Double dt)

Adaptive Passive Observer.lvclass

LabView Object.lvclass
ʟ Observer.lvclass
ʟ Passive Observer.lvclass
ʟ Adaptive Observer.lvclass

Implemented by Viktor Berg.

Constructor Summary

Name	Note
Adaptive Observer()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
Matrix	Gamma wave	
Matrix	K1h	
Matrix	K2h	
Matrix	K2l	
Matrix	K3h	
Matrix	K3l	
Matrix	K4h	
Matrix	K4l	
Matrix	Omega h	
Matrix	Omega l	
Matrix	Tf	

Method Summary

Data Type of Return	Name with input arguments	Note
Matrix	Adaptive Law (…)	
1D Array of Double	Bias Estimator	
Vessel.lvclass	EstimateStates (Vessel.lvclass vessel)	Estimates the states.
	Filter	
Void	Initialize Matrices ()	
1D Array of Double	Wave Estimator	

Guidance.lvclass

Labview Object.lvclass
∟ Guidance.lvclass

Implemented by Espen Tolpinrud.

This is a template or abstract class for all guidance modules in the control system.

Constructor Summary

Name	Note
Guidance()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
Matrix	Delta	Tuning matrix for filter based reference model (Outgoing)
Matrix	Omega	Tuning matrix for filter based reference model (Outgoing)

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Desired States (Vessel.lvclass vessel)	Calculates the desired position and velocity for the vessel. This method must be implemented by all children of this class.

Altitude Guidance.lvclass

Labview Object.lvclass
∟ Guidance.lvclass
∟ Altitude Guidance.lvclass

Implemented by Espen Tolpinrud.

Guidance Law for altitude.

Constructor Summary

Name	Note
Altitude Guidance()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
Double	K_ff	Tuning value for feed forward term
Double	K_i	Tuning value for integrator term
Double	K_p	Tuning value for proportional term

Method Summary

Data Type of Return	Name with input arguments	Note
-	Altitude GL	Private function for Altitude Guidance, used by Calculate Desired States
Vessel.lvclass	Calculate Desired States (Vessel.lvclass vessel)	Calculates the desired altitude the vessel.

DP Guidance.lvclass

Labview Object.lvclass

∟ Guidance.lvclass

∟ DP Guidance.lvclass

Implemented by Espen Tolpinrud.

Constructor Summary

Name	Note
DP Guidance()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
Double	Criterium for Optimal Heading	Tuning value used in the optimal heading method (Not implemented) (Can be removed?)
1D Array of Double	Position Reference	
Cluster of {1D Array of Double, 1D Array of Double, 1D Array of Double, 1D Array of Double}	Previous Kinematics	
1D Array of Double	Restart Criteria	Tuning for the feedback method. Used to determine when to restart desired position updates.
1D Array of Double	Wait Criteria	Tuning for the feedback method. Used to determine when to wait for the vessel to catch up to the reference.

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Desired States (Vessel.lvclass vessel)	Calculates the desired position and velocity for the vessel.
3 1D Array of double, Boolean	Feedback (Vessel.lvclass vessel, Boolean reset_in, Boolean use_estimated_values)	Calculate the next desired step if the vessel is close enough to the current desired step. If not, the reference waits for the vessel to catch up.

Joystick Guidance.lvclass

Labview Object.lvclass
∟ Guidance.lvclass
∟ Joystick Guidance.lvclass

Implemented by Espen Tolpinrud.

Constructor Summary

Name	Note
Joystick Guidance()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
UInt16	Control Mode	Selector of what control mode the joystick input is set for. Modes are Position, Velocity and Thrust.
UInt16	Control Mode Previous	Selector of what control mode the joystick input is set for from last time step
1D Array of Double	Eta Set	A copy of desired position. Helps to restrain depth changes due to uncontrolled pitch movement while driving.
Uint16	Frame Mode	Selector of what frame mode the joystick input is set for. Modes are Body, NED and cylindrical.
UInt16	Frame Mode Previous	Selector of what frame mode the joystick input is set for from last time step.
Cluster of {Boolean, Boolean, Boolean, Boolean}	Inside_Dead_Zone	A cluster of Booleans. A cluster element is true if the input in the corresponding direction or rotation is zero i.e. inside the dead zone.
Cluster of {Boolean, Boolean, Boolean, Boolean}	Inside_Dead_Zone_Previous	A cluster of Booleans. A cluster element is true if the input in the corresponding direction or rotation is zero i.e. inside the dead zone from last time step.
Int32	Loop count	The iteration number for the control system.
Double	Radius	The radius used in cylindrical frame mode.
Double	Radius Initially	The initially radius in cylindrical frame mode.
Boolean	Set Origin	A Boolean used to notify whether or not to set origin.
Boolean	Set Turn Point	A Boolean used to notify whether or not to set turn point.
Double	Step Depth	A standard step in depth direction. Usually used to make small corrections in the direction by button inputs or POV direction stick.
Double	Step Heading	A standard step in heading angle. Usually used to make small corrections on the angle by button inputs

		or POV direction stick.
Double	Step Surge	A standard step in surge direction. Usually used to make small corrections in the direction by button inputs or POV direction stick.
Double	Step Sway	A standard step in sway direction. Usually used to make small corrections in the direction by button inputs or POV direction stick.
Double	Theta	Used in cylindrical frame mode
Cluster of {19 Boolean}	VI Flow Criteria	The cluster is updated on each time step by the Compute Conditions method. The cluster contains - (1) CylinderMode AND (2) - (2) Out of DeadZone-Z-Axis Rot AND (3) - (3) Change ControlMode OR (10) - (4) Out of DeadZone XY-Plane - (5) Out of DeadZone XY-Plane OR Set Origin Point - (6) Set Origin Point - (7) Deactivating CylinderMode OR Initialize - (8) Deactivate CylinerMode - (9) Initialize - (10) Activating CylinderMode - (11) Set Turn pos OR (10) - (12) (11) AND CylinderMode - (13) (11) OR (1) - (14) (1) OR (16) - (15) Activating OR Deactivating CylinderMode - (16) (15) OR Initialize - (17) (5) OR (16) - (18) Out of DeadZone Z-Axis Rot OR (17) - (19) Out of DeadZone Z-Axis Rot

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Desired States (Vessel.lvclass vessel)	Calculates the desired position and velocity for the vessel. Input from user through joystick is used to generate the desired states.
Void	Compute Condition (Double X-Axis, Double Y-Axis, Double Z-Axis, Double Z-Rotation, Int32 Loop nr, Boolean First)	Compute the Boolean values in the cluster VI Flow Criteria.
1D Array of Double	Cylinder Kinematics Modification (1D Array of Double Max Velocity, Double Radius)	Modify the max velocity to cylinder coordinates.

1D Array of Double, Double, Double	Eta Set Update (1D Array of Double eta, Boolean condition, Boolean CylinderMode)	Updates the position reference used to maintain desired depth and heading.
Boolean	Initialize Check (Int32 Loop Counter)	Checks if the system needs to be initialized or re-initialized.

Tracking Guidance.lvclass

Labview Object.lvclass
∟ Guidance.lvclass
∟ Tracking Guidance.lvclass

Constructor Summary

Name	Note
Tracking Guidance()	Default constructor to create an instance of the class.
Tracking Guidance (Matrix Omega, Matrix Delta)	Constructor which also sets the reference model tuning matrices.

Field Summary

Data Type	Name	Note
Double	Acceptance Circle	Representation of the tolerance of position error in order to change way point.
Cluster	Previous Kinematics	States from last time step.
1D Array of Double	Restart Criteria	Tuning for the feedback method. Used to determine when to restart desired position updates.
1D Array of Double	Wait Criteria	Tuning for the feedback method. Used to determine when to wait for the vessel to catch up to the reference.

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Desired States (Vessel.lvclass vessel)	Calculates the desired position and velocity for the vessel. Uses a list of waypoints to perform simple and naive tracking. (To be substituted at a later point)
3 1D Array of Double	Feedback (Vessel.lvclass vessel, 1D Array of Double eta_ref, Boolean reset_in, Boolean Use Estimated Values)	Calculate the next desired step if the vessel is close enough to the current desired step. If not, the reference waits for the vessel to catch up.

Tracking STRM Guidance.lvclass

Labview Object.lvclass

∟ Guidance.lvclass

∟ Tracking Guidance STRM.lvclass

Implemented by Espen Tolpinrud.

Constructor Summary

Name	Note
Tracking Guidance STRM()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
Boolean	Abort	Boolean flag used to notify if tracking action is to be aborted.
Cluster of {19 Double}	Guidance Engine Parameters	Running variables for guidance engine. (May be removed)
UInt16	Heading Mode	Selector of heading mode for tracking.
Boolean	Pause	Boolean flag used to notify if tracking is to be paused.
Cluster {8 Double}	Performance Behavior	Cluster of settings regarding ROV behavior in tracking. Values are: - % Velocity max lin to exp - % Velocity zero lin to exp - % Total dist at max velocity - % Max angular velocity - Time to max angular velocity - Min dist at max velocity - Max linear velocity - Time to max linear velocity
Double	psivar	Heading, used in variable heading case.
Boolean	Reset	Boolean flag used to notify if tracking action is to be reset.
Boolean	Start	Boolean flag used to notify if tracking action is to be started.
Cluster of {4 Double}	STRM	Cluster with values from the SynThetic Reference Model (STRM)
Boolean	Tracking Active	Boolean flag used to notify if tracking action is active.
Cluster {6 Boolean}	Tracking Engine Run-Time Logic	Cluster of Boolean flags used by the guidance engine. Values are: - Complete - Running - clrRMmemo - faststart

		- newcalc - aborting
Cluster {}	Tracking Options	Cluster with settings for the tracking. Values are: - (2) Fixed Heading - (3) PoI N - (3) PoI E - % Superior Bound - % Inferior Bound - Duration no operation period between motions - Tolerance of initial angle - Distance to activate waiting function - Distance to deactivate waiting function
Boolean	Use Depth	Boolean flag used to notify if tracking action is to use current depth.
Boolean	Waiting Function	Boolean flag used to notify if tracking action is to use waiting function.

Method Summary

Data Type of Return	**Name with input arguments**	**Note**
Vessel.lvclass	Calculate Desired States (Vessel.lvclass vessel)	Calculates the desired position and velocity for the vessel. This method must be implemented by all children of this class.
Boolean	Check if Done ()	Checks if complete and not running
Boolean	Reset Action ()	If reset, set waiting function and use depth to false.

Controller.lvclass

Labview Object.lvclass
    ∟ Controller.lvclass

Implemented by Espen Tolpinrud.

A template or abstract class for controller modules in the control system.

Constructor Summary

Name	Note
Controller()	Default constructor to create an instance of the class.
Controller (String name)	Constructor which sets the name of the controller.

Field Summary

Data Type	Name	Note
String	Controller Name	The name of the controller, not much used.

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Forces (Vessel.lvclass vessel)	Calculates the desired forces for the vessel. This method must be implemented on all children of this class.

Joystick Controller.lvclass

Labview Object.lvclass
∟ Controller.lvclass
∟ Joystick Controller.lvclass

Implemented by Espen Tolpinrud.

Controller used to control the vessel with direct thrust input

Constructor Summary

Name	Note
Controller()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
Cluster of {3 Double}	PID Gains Heading	Tuning values for PID controller for heading
Cluster of {3 Double}	PID Gains Depth	Tuning values for PID controller for depth

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Forces (Vessel.lvclass vessel)	Gets joystick commands and sets up a desired thrust according to the input. Contains auto depth and auto heading as well.

PID Controller.lvclass

Labview Object.lvclass

    ∟ Controller.lvclass

        ∟ PID Controller.lvclass

Implemented by Espen Tolpinrud.

Constructor Summary

Name	Note
PID Controller()	Default constructor to create an instance of the class.

Field Summary

Data Type	Name	Note
Matrix	K_D	Tuning matrix for the derivative term in the controller.
Matrix	K_I	Tuning matrix for the integral term in the controller.
Matrix	K_P	Tuning matrix for the proportional term in the controller.

Method Summary

Data Type of Return	Name	Note
Vessel.lvclass	Calculate Forces (Vessel.lvclass vessel)	Calculates the desired forces for the vessel. This method must be implemented by all children of this class.
Void	Set Gains (3D Array of Double)	The tuning matrices are given through a 3D array and then set in their respective fields. K_P should be at the first index, followed by K_I in second. Last the K_D should be on the third index.

Linear PID Controller.lvclass

Labview Object.lvclass

∟ Controller.lvclass

∟ PID Controller.lvclass

∟ Linear PID Controller.lvclass

Implemented by Espen Tolpinrud.

Constructor Summary

Name	Note
Linear PID Controller()	Default constructor to create an instance of the class.
Linear PID Controller (Matrix KP, Matrix KI, Matrix KD)	Constructor which gets and sets the tuning matrices.

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Forces (Vessel.lvclass vessel)	Calculates the desired forces for the vessel with the expression $\tau = -(K_P \tilde{\eta} + K_I \int \tilde{\eta} dt + K_D \dot{\eta})$

Non Linear PID Controller.lvclass

Labview Object.lvclass
∟ Controller.lvclass
∟ PID Controller.lvclass
∟ Non Linear PID Controller.lvclass

Implemented by Viktor Berg. Modified by Espen Tolpinrud to include optional Speed Controller.

Constructor Summary

Name	Note
Non Linear PID Controller ()	Default constructor to create an instance of the class.
Non Linear PID Controller (Matrix KP, Matrix KI, Matrix KD)	Constructor which gets and sets the tuning matrices.

Field Summary

Data Type	Name	Note
1D Array of Double	Int_limit	Used to check if the integral term shall integrate the error. If the error is larger than the Integral limit, zero is sent to the integrator.
Matrix	K_anti	Tuning matrix for the anti-wind up effect in the integral term.
Matrix	K_I_speed	Tuning matrix for the integral effect for the speed controller term
Boolean	Speed Controller	Flag to determine if to use speed controller term or not.

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Forces (Vessel.lvclass vessel)	Calculates the desired position and velocity for the vessel. Similar to the linear PID but also include a Feed Forward term to include non-linear effects.

Method Details

LQR Controller.lvclass

Labview Object.lvclass
∟ Controller.lvclass
∟ PID Controller.lvclass
∟ Non Linear PID Controller.lvclass
∟ LQR.lvclass

Implemented by Viktor Berg. Revised by Espen Tolpinrud.

Changes from the old system: Inherit from Non Linear Pid Controller.lvclass as the controller algorithm is the same.

This is a type of optimal control. The controller equation is based on the non-linear PID controller. The main difference between LQR and its parent Non Linear PID Controller is the automatic tuning of the matrices K_P and K_D.

Constructor Summary

Name	Note
LQR()	Default constructor to create an instance of the class.
LQR (Matrix K_i, Matrix Q, Matrix R)	Constructor which gets and sets the tuning matrices.

Field Summary

Data Type	Name	Note
Matrix	Q	Tuning matrix for Riccati equation
Matrix	R	Tuning matrix for Riccati equation

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Forces (Vessel.lvclass vessel)	Calculates the desired forces for the vessel. This method also calls on the parent calculate Forces () in order to find the forces.
Void	Set Gains LQR (Matrix K_I Matrix Q, Matrix R, Matrix K_anti, 1D Array of Double Integral Limit)	Sets the tuning matrices for the controller.

Sliding Mode.lvclass

Labview Object.lvclass
∟ Controller.lvclass
∟ Sliding Mode.lvclass

Implemented by Viktor Berg.

Constructor Summary

Name	Note
Sliding Mode ()	Default constructor to create an instance of the class.
Sliding Mode (Matrix K_s, Matrix K_pid, 1D Array of Double Lambda, 1D Array of Double Phi)	Constructor that gets and sets the tuning matrices.

Field Summary

Data Type	Name	Note
1D Array of Double	Int_limit	Used to check if the integral term shall integrate the error. If the error is larger than the Integral limit, zero is sent to the integrator.
Matrix	K_pid	Tuning matrix for the standard PID controller used in the algorithm.
Matrix	K_s	Tuning Matrix
1D Array of Double	Lambda	Tuning Matrix
1D Array of Double	Phi	Tuning Matrix

Method Summary

Data Type of Return	Name	Note
Vessel.lvclass	Calculate Desired States (Vessel.lvclass vessel)	Calculates the desired position and velocity for the vessel. This method must be implemented by all children of this class.
Void	Set Gains (Matrix K_s, Matrix K_pid, 1D Array of Double Lambda, 1D Array of Double Phi, 1D Array of Double Integral Limit)	Sets the tuning matrices from the corresponding input.

Backstepping.lvclass

Labview Object.lvclass

  ∟ Controller.lvclass

    ∟ Backstepping.lvclass

Not implemented yet!

Constructor Summary

Name	Note
Backstepping()	Default constructor to create an instance of the class.

Method Summary

Data Type of Return	Name with input arguments	Note
Vessel.lvclass	Calculate Desired States (Vessel.lvclass vessel)	Calculates the desired forces for the vessel.

Adaptive Backstepping.lvclass

Labview Object.lvclass

∟ Controller.lvclass

∟ Backstepping.lvclass

∟ Adaptive Backstepping.lvclass

Not implemented yet!

Stand-Alone Methods

mainVI.vi

This is a stand-alone VI used when running the system together with a compactRIO unit. The VI starts by reading a configuration file and setting up TCP/IP connections to the user interface. Four threads is then started to run concurrently sharing the same supervisor object. Mutual Exclusion mechanisms are implemented to avoid race conditions. The locks have time out limits in order to avoid dead locks. Starvation can occur, with wrong timing.

mainVI_SIL.vi

This is a stripped down version of the mainVI.vi as it is to be run with the simulator “in place”. Three of the four threads are moved out to the simulator environment to ensure concurrent running. Only the “main loop” remains.

Reference Model

This is a polymorphic VI which contains a library of reference models used in different settings. The polymorphic VI can change between different VIs depending on the input, but can also be set to a fixed through a selector in the block diagram.

Method Summary

Data Type of Return	Name	Note
	Reference Model Altitude	Filter based method, scalar (Not part of polymorphic VI)
	Reference Model DP	Filter based method, vector
	Reference Model Joystick	Filter based method, vector
	Reference Model Tracking	Filter based method, vector

Kinematics

Rotation matrix $J(\Theta) = \begin{bmatrix} R(\Theta) & 0_{3\times3} \\ 0_{3\times3} & T(\Theta) \end{bmatrix}$

Data Type of Return	Name	Note
Matrix	RotationMatrix (1D Array of Double eta)	A polymorphic VI for Rotation matrices. A selector specifies which of the two RotationMatrix~ methods to be run.
Matrix	RotationMatrix6DOF (1D Array of Double eta)	Gives the rotation matrix in Euler angles for a 6 DOF system based on the position vector eta given in Euler angles.
Matrix	RotationMatrixQuaternions (1D Array of Double eta)	Gives the rotation matrix in quaternions for a 6DOF system based on the position vector eta given in quaternions.

Matrix	TransformationMatrix (1D Array of Double eta)	A polymorphic VI for Transformation matrices. A selector specifies which of the two TransformationMatrix~ methods to be run.
Matrix	TransformationMatrixEuler (1D Array of Double eta)	Gives the transformation matrix in Euler angles for a 6 DOF system based on the position vector eta given in Euler angels.
Matrix	TransformationMatrix Quaternions (1D Array of Double eta)	Gives the transformation matrix in quaternions for a 6 DOF system based on the position vector eta given in quaternions.
Matrix	SkewMatrix (1D Array of Double)	A polymorphic VI for creating a Skew matrix based on the input vector. A selector determines which of the 3DOF or 6DOF version to run.
Matrix	SkewMatrix3DOF (1D Array of Double vector)	The input vector must be of size 3. These elements are then used to create the Skew Matrix.
Matrix	SkewMatrix6DOF (1D Array of Double nu)	The input vector should be of size 6. The three last elements in the input vector are used to set up the skew matrix.
Matrix	J Library (1D Array of Double eta)	A polymorphic VI for creating the rotation matrix J. A Selector decides what form of the J matrix to be sent as an output
Matrix	J_euler (1D Array of Double eta)	Creates the rotation matrix J for 6DOF system using Euler angles.
Matrix	J_quart (1D Array of Double eta)	Creates the rotation matrix J for 6DOF system using quaternions.
Matrix	Jtrans_euler (1D Array of Double eta)	Creates the rotation matrix J for 6DOF system using Euler angels. Returns the transposed of J.
Matrix	Jtrans_quart (1D Array of Double eta)	Creates the rotation matrix J for 6 DOF system using quaternions. Returns the transposed of J.
Matrix	Jinv_euler (1D Array of Double eta)	Creates the rotation matrix J for 6DOF system using Euler angles. Returns the inverse of J.
Matrix	Jinv_euler (1D Array of Double eta)	Creates the rotation matrix J for 6 DOF system using quaternions. Returns the inverse of J.
Matrix	Jdot (1D Array of Double eta, 1D Array of Double nu)	Creates the time derived of rotation matrix by calculating $\dot{R}$ and $\dot{T}$. Only Euler angles are used for this.
1D Array of Double	EtaTransformation (1D Array of Double eta)	Transform a vector between Euler angles and quaternions using either Eta_Euler2QEta_Quart or Eta_Quart2Eta_Euler depending on transformation direction.
1D Array of Double	Eta_Euler2Eta_Quart (1D Array of Double eta)	Transforms the input vector given in NED + Euler angels to a vector given with NED + quaternions.
1D Array of Double	Eta_Quart2Eta_Euler (1D Array of Double eta)	Transforms the input vector given with NED + quaternions to a vector in NED + Euler angles.
1D Array of Double	euler2q (1D Array of Double eta)	Calculates the angles in quaternions from Euler angles. Method is taken from MSS toolbox written by Thor I. Fossen.

1D Array of Double	q2Euler(1D Array of Double eta)	Calculates the angles in Euler angels from quaternions. The method is taken from MSS toolbox written by Thor I. Fosse and translated to LabView code.

Other

Data Type of Return	Name	Note
-	Rad 2 PI-PI	Polymorphic VI for the two following methods.
Double	Rad 2 PI-PI Scalar (Double)	Converts the angle in radians to the interval $-\pi$ to π.
1D Array of Double	Rad 2 PI-PI Vector(1D Array of Double)	Converts angles in radians to the interval $-\pi$ to π.

Integrators

Data Type of Return	Name	Note
Double/1D Array of Double	Integrator Trapezoid	A Polymorphic VI for the trapezoid integrators in the system. Both the input and a selector can be used to determine the method to be used.
Double	Integrator Trapezoid scalar	A Scalar integrator using the trapezoidal rule.
1D Array of Double	Integrator Trapezoid vector	A vector integrator using the trapezoidal rule.

Signal Processing

Data Type of Return	Name	Note
	Alt_input	
	Alt_IO	
	Cocoord	
	Configure sub-VI	
	CRIO write	
	DVL_2_QUAD	
	Input converter	
	JS chair get	
	MRU_IO	
	MRU_SP	
	MRUinput subVI	
	MT_DATA_TRANS	
	MT_GET	
	MT MOD	
	NAVI_IO 2	
	NAVIpacinput subVI	
	NAVIpacinput subVI 30k	
	Origin SP	

	Port 3 IO	
	ROV_IO 2	
	ROV_IO 30k	
	ROVinput 30k subVI	
	ROVinput subVI	
	Search string	
	Set Length	
	Signal IO	
	Signal IO SIL	
	SP BC	
	SP_mod	
	Write sub-VI	

Others

Data Type of Return	Name	Note
Double Matrix	AutoParameter Guidance Omega (1D Array of Double eta_ref, 1D Array of Double eta_last)	Calculates desired omega values for the filter reference model.
1D Array of Double	Error Calculations (1D Array of Double eta, 1D Array of Double eta_des)	Used to calculate the difference between the current position and the desired position.
1D Array of Double	Eta2cont	Transform a discrete angle to continuous angle. Should not be placed such that it can miss any time step!
Double, Boolean	Get_set_point(Double setpoint, Boolean flag)	When flag is true, the current setpoint is stored and sent as output until flag is true again. (Switch on, switch off mechanism)
Matrix, Matrix	LQR	Used to calculate the tuning matrices for the LQR controller. (May move to class)
Matrix	Mass 2 Coriolis	Used to calculate the Coriolis matrix. Used in Non Linear PID and Non Linear Passive Observer.
String	Read Node Name (Tag Reference ref, String AttributeName)	Reads the attribute matching the field *AttributeName* in the tag *ref* points to. Used in the configuration file reader.
TCP Connection Reference	Reconnection Procedure	Polymorphic VI for the two following methods.
TCP Connection Reference	Reconnection Procedure Scalar (TCP Connection Reference ref, UInt16 Port)	Checks if connection is lost, if so tries to reconnect to client.
1D Array of TCP Connection Reference	Reconnection Procedure Array (1D Array of TCP Connection Reference ref, 1D Array of UInt16 ports)	Checks if the connection is lost, if so tries to reconnect to client. All connections must be reestablished or none are.

String	Scan Token Generator	Generates the scan-string used to read integer and double values from a string. Used in the String to Matrix methods.
1D Array of UInt16	Sperre thrust alloc Minerva	Converts the desired actions from the control system to a telebuffer/telegram which is to be sent to the vessel.
1D Array of UInt16	Sperre thrust alloc Neptune	Converts the desired actions from the control system to a telebuffer/telegram which is to be sent to the vessel.
Matrix	String to Matrix	A polymorphic VI that transform a String of numbers to a Matrix.
Matrix	String to Matrix Diag	Transforms a String of number to a diagonal matrix where the numbers in the String is the values on the diagonal in the matrix.
Matrix	String to Matrix Norm	Transforms a String of number to a matrix where each row in the matrix is separated by ";" in the String.
1D Array of Double	String to Vector	Transforms a String of numbers to a 1D Array of Double.
Boolean	ToggleSwitch	Used to convert a constant True/False Signal to a pulse True signal.
Boolean	True Once	Used to convert a constant True/False Signal to one pulse of True. True only once!

Configuration File

The configuration file is unique for each vessel to be used with the control system. This is what decides the setup for the system. It is therefore important that the configuration file is correct. The format on the file is set to xml as it is easy for both machines and humans to read. Keep in mind that the tags must be exact as well as the attributes.

Table of tags and attributes

Tag	Parent Tag(s)	Attributes	Notes
<Configuration>	-	-	Start tag for configuration
<Sensor>	<Configuration>	ID, Name, Type, PosX, PosY, PosZ, RotX, RotY, RotZ	Tag for initializing sensors
<Vessel>	<Configuration>	Name, Type, NDOF	Tag for vessel initialization.
<TechincalData>	<Vessel>	DOF, Length, Width, Height, Buoyancy, CenterGravity, CenterBuoyancy, TimeStep, TauMax	Tag used for vessel data
<PortConfiguration>	<Vessel>	-	Tag for initialization of communication ports for compactRIO unit
<Port>	<PortConfiguration>	ID, BaudRate, DataBits, Parity, StopBits, FlowControl	Tag for data used to set up a port for the compactRIO
<Matrix>	<Vessel>	-	Tag for matrix
<Mass>	<Matrix>	Type, Size, Values	Tag for Mass matrix
<AddedMass>	<Matrix>	Type, Size, Values	Tag for Added Mass matrix
<LinearDamping>	<Matrix>	Type, Size, Values	Tag for Linear Damping matrix
<NonLinearDamping>	<Matrix>	Type, Size, Values	Tag for Non Linear Damping matrix
<Observer>	<Vessel>	-	Tag for observer collection
<KalmanFilter>	<Observer>	Mode	Tag for Kalman Filter Observer
<KalmanFilterAltitude>	<Observer>	Mode	Tag for Altitude Kalman Filter Observer
<Passive>	<Observer>	Mode	Tag for Passive Non Linear Observer
<TuningKalman>	<KalmanFilter>, <KalmanFilterAltitude>	-	Tag for tuning fields for Kalman Filters
<TuningPassive>	<Passive>	-	Tag for tuning fields for Kalman Filters.
<R>	<TuningKalman>	Type, Size, Value	Tag for tuning matrix R
<Q>	<TuningKalman>	Type, Size, Value	Tag for tuning matrix Q

<Wave>	<TuningPassive>	TypeOmega, SizeOmega, ValueOmega, TypeDelta, SizeDelta, ValueDelta, TypeLambda, SizeLambda, ValueLambda	Tag for wave filter values
<K1>	<TuningPassive>	Type1, Size1, Value1, Type2, Size2, Value2	Tag for tuning matrix K1. Consists of two 6x6 matrices. Can be set to zero as they will be calculated in the control system by tuning equations.
<K2>	<TuningPassive>	Type, Size, Value	Tag for tuning matrix K2.
<K3>	<TuningPassive>	Type, Size, Value	Tag for tuning matrix K3
<K4>	<TuningPassive>	Type, Size, Value	Tag for tuning matrix K4
<T>	<TuningPassive>	Type, Size, Value	Tag for time constant matrix T
<Navigation>	<Vessel>	-	Tag for navigation collection
<DPOperation>	<Navigation>	Mode	Tag for DP Guidance
<Tracking>	<Navigation>	Mode	Tag for Tracking Guidance
<TrackingSTRM>	<Navigation>	Mode	Tag for Tracking Guidance with synthetic reference model
<Joystick>	<Navigation>	Mode	Tag for Joystick Guidance
<Criteria>	<DPOperation>, <Tracking>, <Joystick>	SizeRestart, Restart, SizeWait, Wait	Tag for restart and wait criteria values for guidance objects.
<Omega>	<DPOperation>, <Tracking>, <Joystick>	Type, Size, Value	Tag for reference model tuning matrix omega.
<Delta>	<DPOperation>, <Tracking>, <Joystick>	Type, Size, Value	Tag for reference model damping matrix
<Controller>	<Vessel>	-	Tag for controller collection
<PIDLinear>	<Controller>	Mode	Tag for Linear PID Controller
<NonLinearPID>	<Controller>	Mode	Tag for Non Linear PID Controller
<LQR>	<Controller>	Mode	Tag for LQR Controller
<SlidingMode>	<Controller>	Mode	Tag for Sliding Mode Controller
<TuningPID>	<PIDLinear>, <PIDNonLinear>, <LQR>	-	Tag for tuning fields for PID based controllers. These include linear, non-linear PID as well as LQR
<TuningSliding>	<SlidingMode>	-	Tag for tuning fields for Sliding Mode controller.
<KP>	<TuningPID>	Type, Size, Value	Tag for Proportional term tuning matrix
<KI>	<TuningPID>	Type, Size, Value	Tag for Integral term tuning matrix

<KD>	<TuningPID>	Type, Size, Value	Tag for Derivative term tuning matrix
<R>	<TuningPID>	Type, Size, Value	Tag for tuning matrix R used in LQR computations.
<Q>	<TuningPID>	Type, Size, Value	Tag for tuning matrix Q used in LQR computations.
<KS>	<TuningSliding>	Type, Size, Value	Tag for tuning matrix in Sliding Mode.
<KPID>	<TuningSliding>	Type, Size, Value	Tag for PID tuning matrix used in Sliding Mode controller.
<Lambda>	<TuningSliding>	Value	Tag for tuning in Sliding Mode
<Phi>	<TuningSliding>	Value	Tag for tuning in Sliding Mode
<IntegralLimit>	<TuningSliding>	Value	Tag for integral limits set on the integrator
<SensorInstalled>	<Vessel>	NumberOfSensors, SensorIDs	Tag used to set which sensors are used by the vessel.
<Thruster>	<Vessel>	ID, Name, Type, PosX, PosY, PosZ, RotTheta, RotPhi, MaxRPM, Coefficient	Tag for Thruster data.

Appendix B

Software Requirement Specification Template

Definition B.1. *SRS template with description from [17]*

1 Introduction - The introduction of the SRS should provide an overview of the entire SRS. It should contain the following subsections:

1.1 Purpose

- *Delineate the purpose of the SRS.*
- *Specify the intended audience for the SRS.*

1.2 Scope

- *Identify the software product(s) to be produced by name.*
- *Explain what the software product(s) will, and, if necessary, will not do.*
- *Describe the application of the software being specified, including relevant benefits, objectives, and goals.*
- *Be consistent with similar statements in higher-level specifications if they exist.*

1.3 Definitions, acronyms, and abbreviations - This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.

1.4 References

- *Provide a complete list of all documents referenced elsewhere in the SRS.*

- *Identify each document by title, report number (if applicable), date, and publishing organization.*
- *Specify the sources from which the references can be obtained.*

1.5 Overview

- *Describe what the rest of the SRS contains.*
- *Explain how the SRS is organized.*

2 Overall description - This section of the SRS should describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3 of the SRS, and makes them easier to understand.

2.1 Product Perspective - This subsection of the SRS should put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software.

A block diagram showing the major components of the larger system, interconnections, and external interfaces can be helpful.

This subsection should also describe how the software operates inside various constraints. For example these constraints could include

- *System interfaces*
- *User interfaces*
- *Hardware interfaces*
- *Software interfaces*
- *Communication interfaces*
- *Memory*
- *Operations*
- *Site adaption requirements*

2.2 Product functions - This subsection of the SRS should provide a summary of the major functions that the software will perform. For example, an SRS for an accounting program may use this part to address customer account maintenance, customer statement, and invoice preparation without mentioning the vast amount of detail that each of those functions requires.

Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product. Note that for the sake of clarity

- *The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.*
- *Textual or graphical methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product, but simply shows the logical relationships among variables.*

2.3 User characteristics - This subsection of the SRS should describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. It should not be used to state specific requirements, but rather should provide the reasons why certain specific requirements are later specified in Section 3 of the SRS.

2.4 Constraints - This subsection of the SRS should provide a general description of any other items that will limit the developer's options. These include

- *Regulatory policies*
- *Hardware limitations*
- *Interface to other application*
- *Parallel operation*
- *Audit functions*
- *Control functions*
- *Higher-order language requirements*
- *Signal handshake protocols*
- *Reliability requirements*
- *Criticality of the application*
- *Safety and security considerations*

2.5 Assumptions and dependencies - This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the *operating system is not available, the SRS would then have to change accordingly.*

3 Specific Requirements - This section of the SRS should contain all of the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output. As this is often the largest and most important part of the SRS, the following principles apply:

- *Specific requirements should be stated in conformance with all the characteristics described in section 4.3 in [17]*
- *Specific requirements should be cross-referenced to earlier documents that relates.*
- *All requirements should be uniquely identifiable.*
- *Careful attention should be given to organizing the requirements to maximize readability.*

Appendix

Index

For further details see [17].

Appendix C

Notation of SNAME(1950) for marine vessels

DOF		forces and moments	linear and angular velocities	positions and Euler angles
1	surge	X	u	x
2	sway	Y	v	y
3	heave	Z	w	z
4	roll	K	p	ϕ
5	pitch	M	q	θ
6	yaw	N	r	ψ

Table C.1: The Notation of SNAME for Marine Vessels

Body:	$\mathbf{p}^e_{b/n} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$	Longitude and latitude:	$\Theta_{en} = \begin{bmatrix} l \\ \mu \end{bmatrix}$
NED:	$\mathbf{p}^n_{b/n} = \begin{bmatrix} N \\ E \\ D \end{bmatrix}$	Attitude:	$\Theta_{nb} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$
Body-fixed linear velocity:	$\mathbf{v}^b_{b/n} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$	Body-fixed angular velocity:	$\omega^b_{b/n} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$
Body-fixed force:	$\mathbf{f}^b_b = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$	Body-fixed moment:	$\mathbf{m}^b_b = \begin{bmatrix} K \\ M \\ N \end{bmatrix}$

$$\eta = \left[\begin{array}{c} \mathbf{p}_{b/n}^{b} \\ \Theta_{nb} \end{array} \right], \nu = \left[\begin{array}{c} \mathbf{v}_{b/n}^{b} \\ \omega_{b/n}^{b} \end{array} \right], \tau = \left[\begin{array}{c} \mathbf{f}_{b}^{b} \\ \mathbf{m}_{b}^{b} \end{array} \right] \tag{C.1}$$

Appendix D

Option Panels

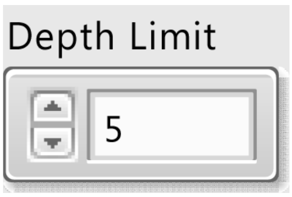
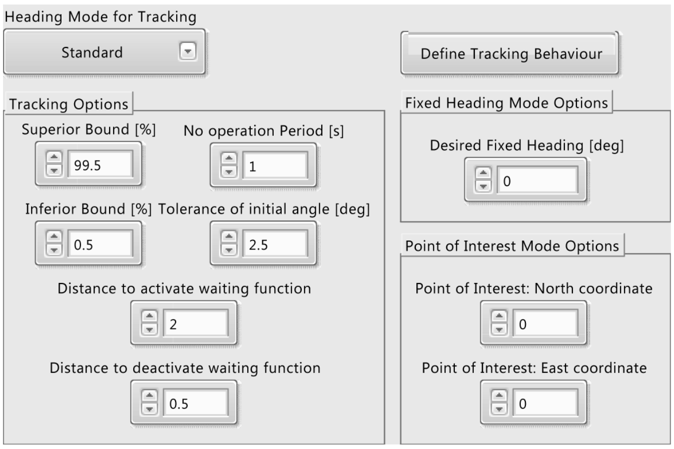

Depth Limit
5

Figure D.1: DP Options Panel as viewed in Options Panel, Frigg.


Heading Mode for Tracking: Standard

Tracking Options
Superior Bound [%]: 99.5
No operation Period [s]: 1
Inferior Bound [%]: 0.5
Tolerance of initial angle [deg]: 2.5
Distance to activate waiting function: 2
Distance to deactivate waiting function: 0.5

Fixed Heading Mode Options
Desired Fixed Heading [deg]: 0

Point of Interest Mode Options
Point of Interest: North coordinate: 0
Point of Interest: East coordinate: 0

Figure D.2: Tracking Options Panel as viewed in Options Panel, Frigg.

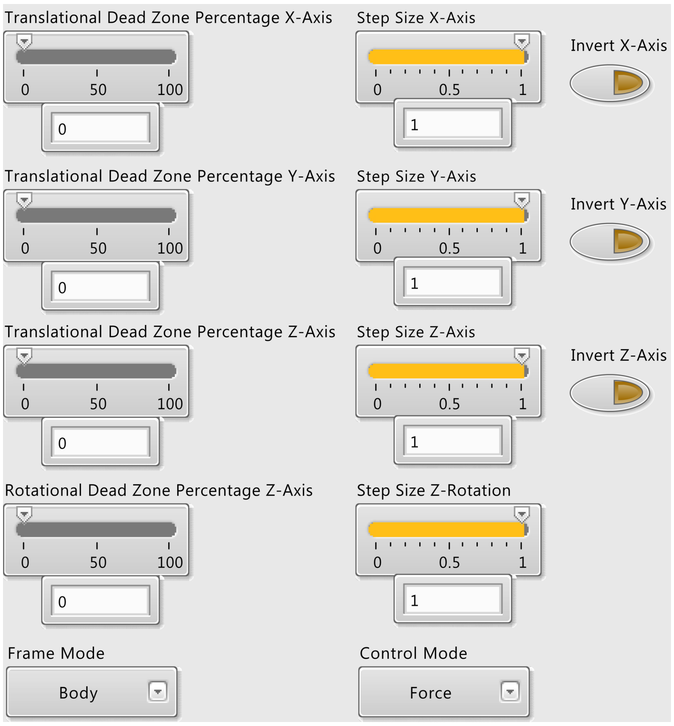
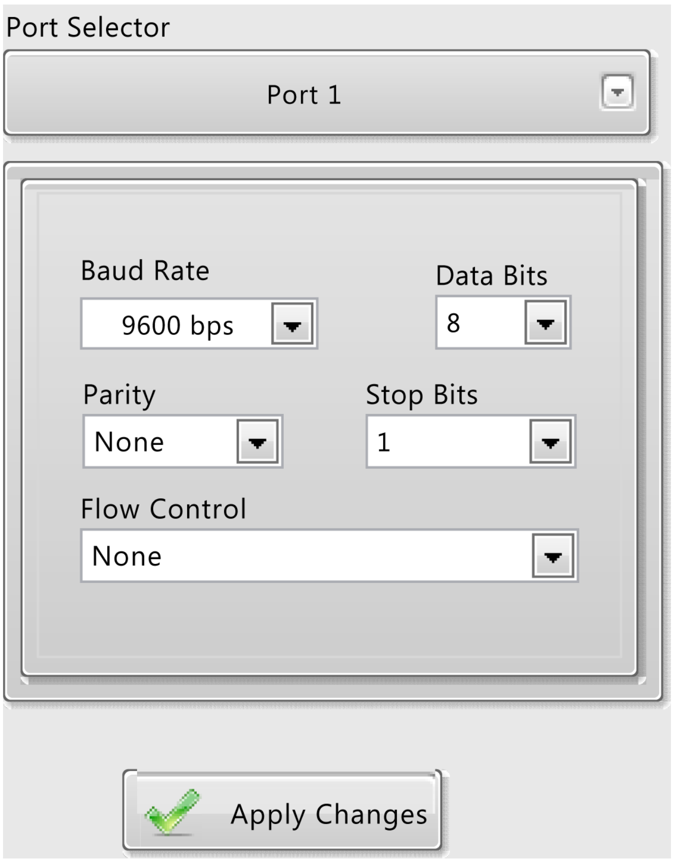

Translational Dead Zone Percentage X-Axis
0
Step Size X-Axis
1
Invert X-Axis
Translational Dead Zone Percentage Y-Axis
0
Step Size Y-Axis
1
Invert Y-Axis
Translational Dead Zone Percentage Z-Axis
0
Step Size Z-Axis
1
Invert Z-Axis
Rotational Dead Zone Percentage Z-Axis
0
Step Size Z-Rotation
1
Frame Mode
Body
Control Mode
Force

Figure D.3: Joystick Options Panel as viewed in Options Panel, Frigg.


Port Selector

Port 1

Baud Rate: 9600 bps

Data Bits: 8

Parity: None

Stop Bits: 1

Flow Control: None

Figure D.4: Port Configuration Panel as viewed in Options Panel, Frigg.

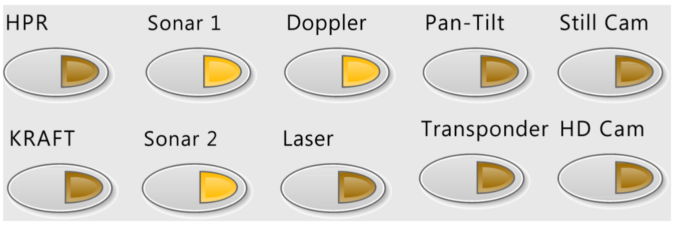


Figure D.5: Power Instruments Panel as viewed in Options Panel, Frigg.

Appendix E

Graph Panels

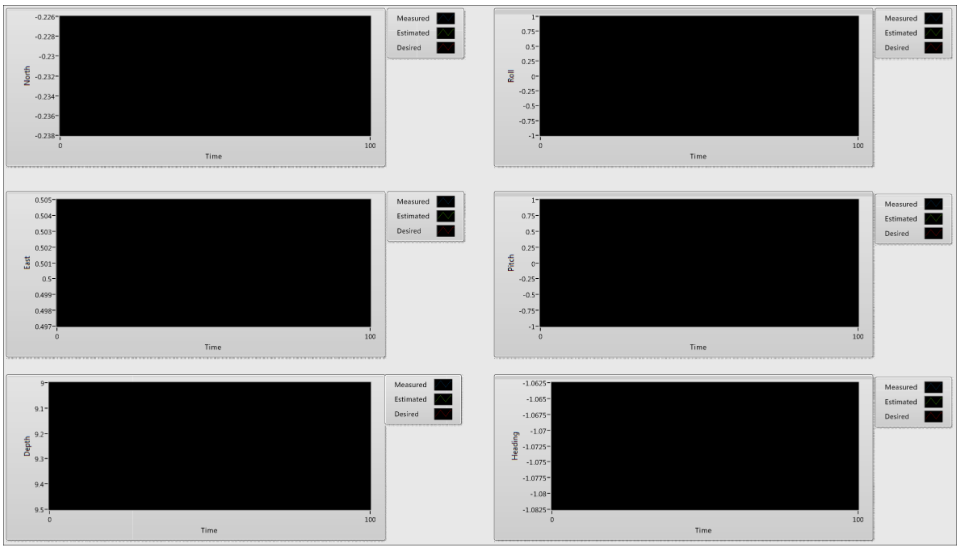


Figure E.1: Position Plots as viewed in GraphView, Frigg.

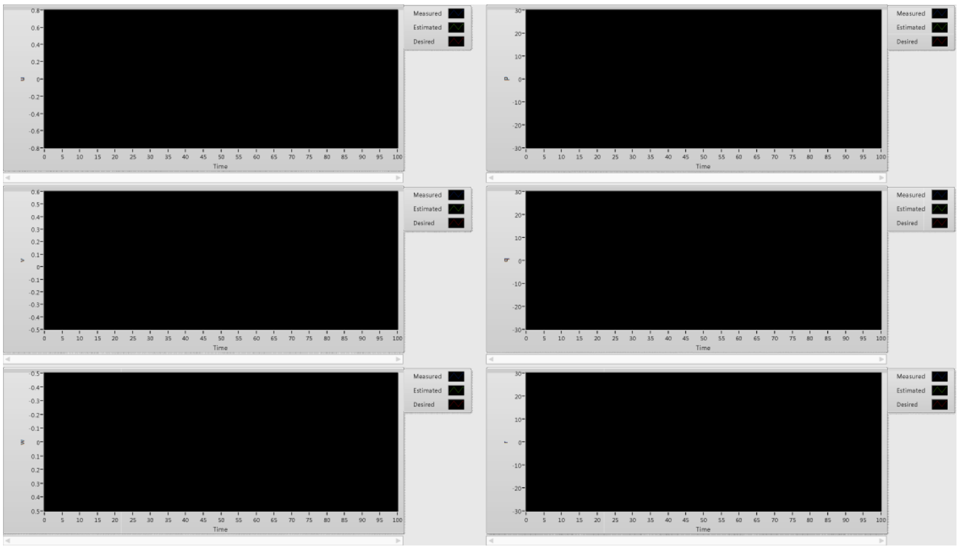


Figure E.2: Velocity Plots as viewed in GraphView, Frigg.

Figure E.3: Thruster Plots as viewed in GraphView, Frigg.

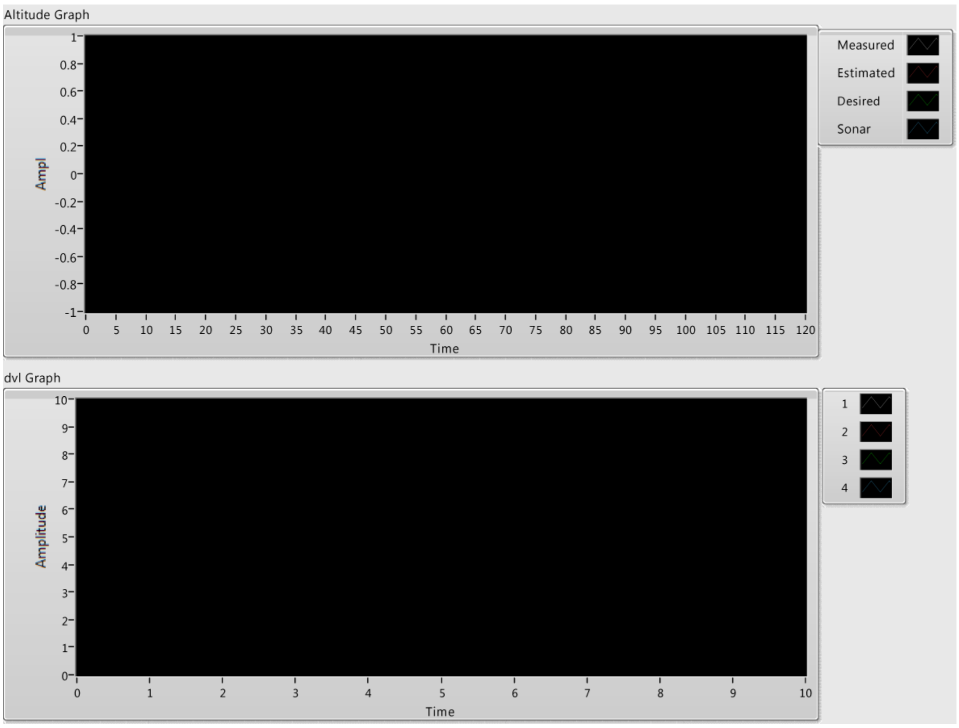


Figure E.4: Altitude Plots as viewed in GraphView, Frigg.

Figure E.5: MTi Plots as viewed in GraphView, Frigg.

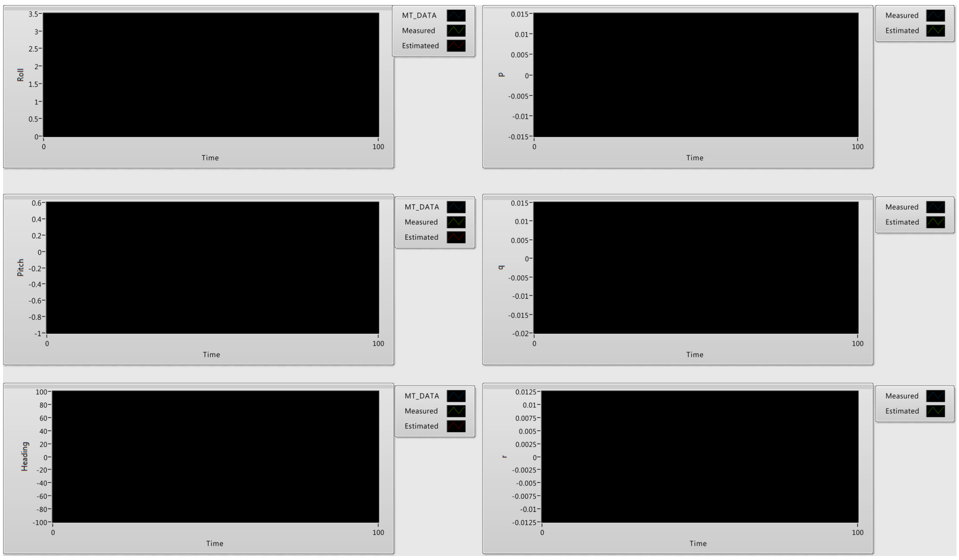
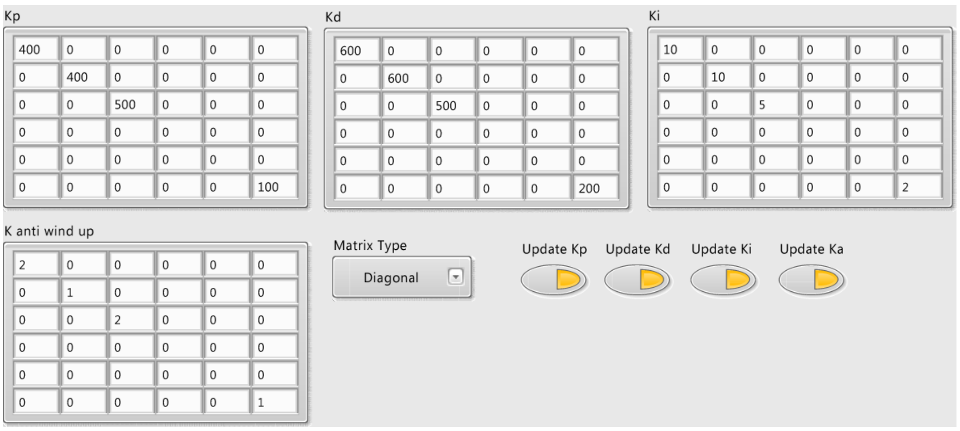


Figure E.6: ECF Plots as viewed in GraphView, Frigg.

Appendix F

Online Tuning Panels



Kp

400	0	0	0	0	0
0	400	0	0	0	0
0	0	500	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	100

Kd

600	0	0	0	0	0
0	600	0	0	0	0
0	0	500	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	200

Ki

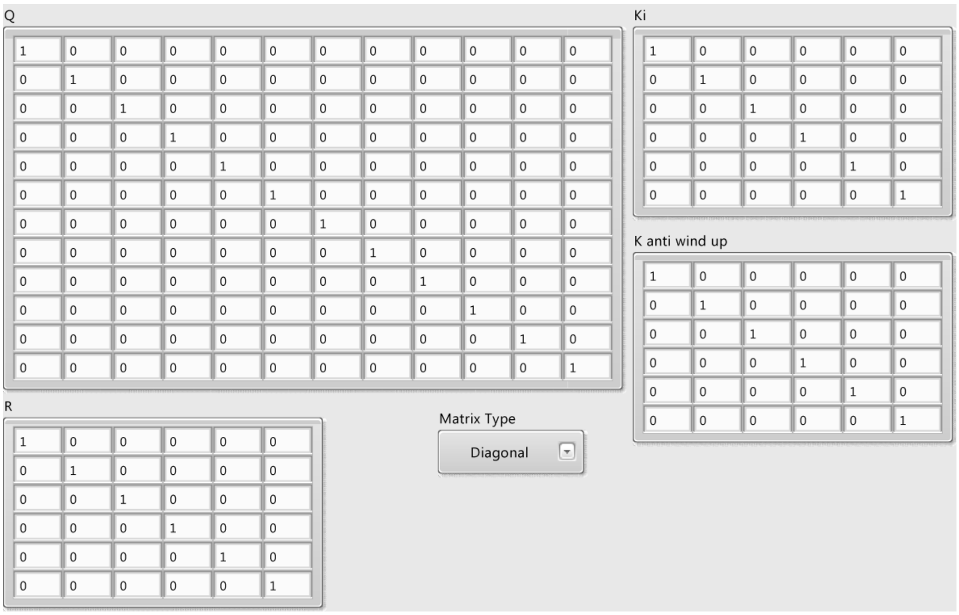
10	0	0	0	0	0
0	10	0	0	0	0
0	0	5	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	2

K anti wind up

2	0	0	0	0	0
0	1	0	0	0	0
0	0	2	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1

Matrix Type: Diagonal

Figure F.1: Nonlinear PID Tuning Panel as viewed in Tuning Panel, Frigg.



Q

1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	1

Ki

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

K anti wind up

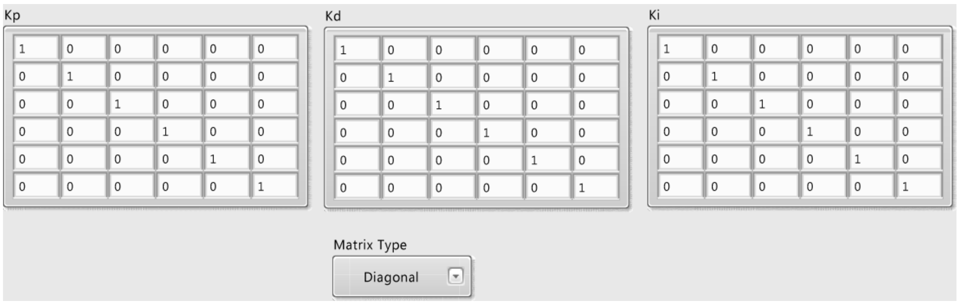
1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

R

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Matrix Type: Diagonal

Figure F.2: LQR Tuning Panel as viewed in Tuning Panel, Frigg.



Kp

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Kd

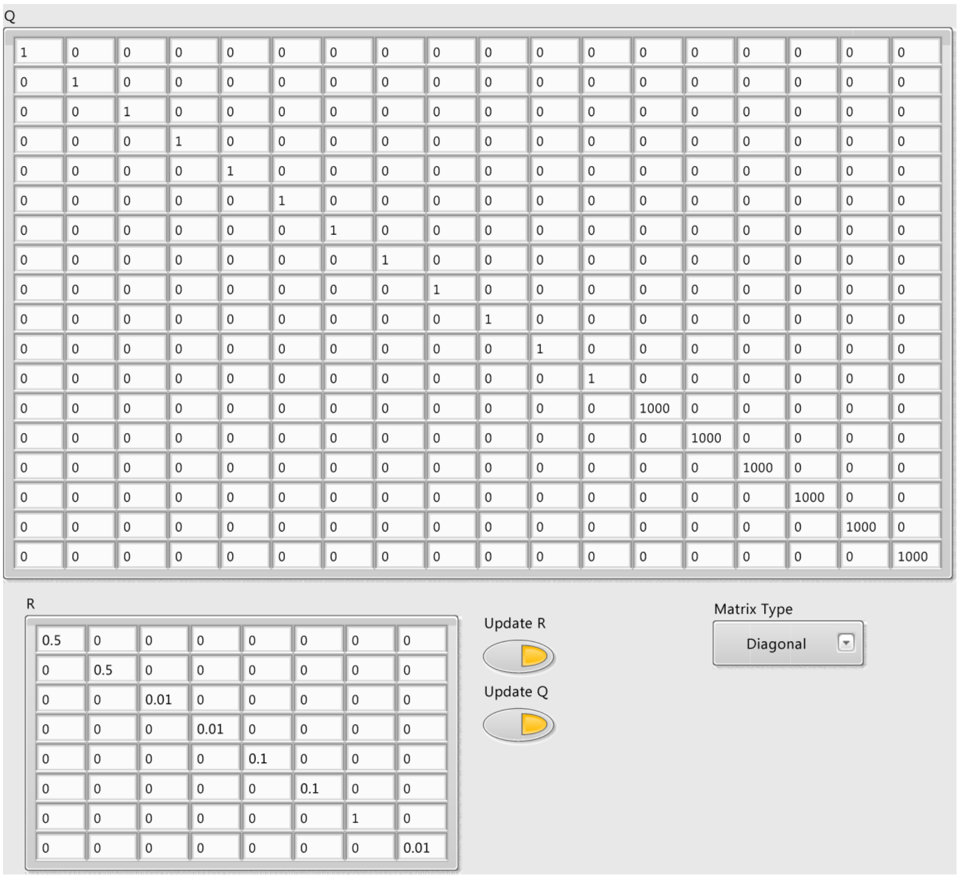
1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Ki

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Matrix Type: Diagonal

Figure F.3: Linear PID Tuning Panel as viewed in Tuning Panel, Frigg.



Q

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000

R

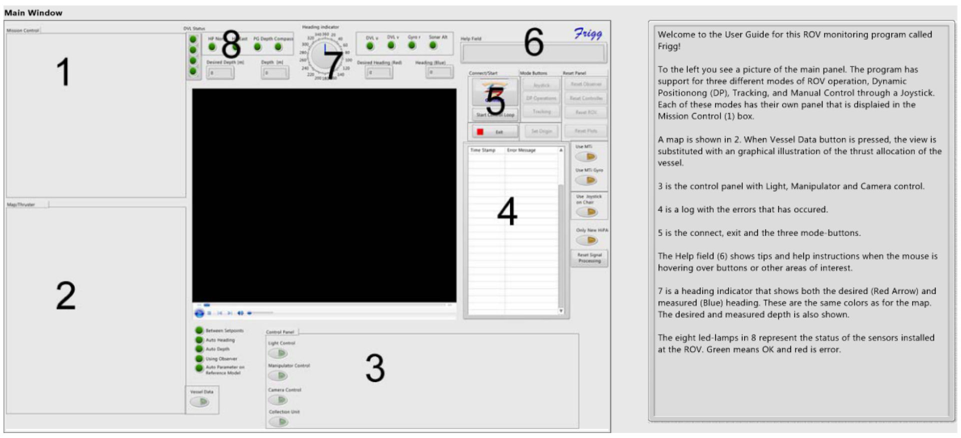
0.5	0	0	0	0	0	0	0
0	0.5	0	0	0	0	0	0
0	0	0.01	0	0	0	0	0
0	0	0	0.01	0	0	0	0
0	0	0	0	0.1	0	0	0
0	0	0	0	0	0.1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0.01

Matrix Type: Diagonal

Figure F.4: Kalman Filter Tuning Panel as viewed in Tuning Panel, Frigg.

Appendix G

Help Center Panels



Welcome to the User Guide for this ROV monitoring program called Frigg!

To the left you see a picture of the main panel. The program has support for three different modes of ROV operation, Dynamic Positionong (DP), Tracking, and Manual Control through a Joystick. Each of these modes has their own panel that is displaied in the Mission Control (1) box.

A map is shown in 2. When Vessel Data button is pressed, the view is substituted with an graphical illustration of the thrust allocation of the vessel.

3 is the control panel with Light, Manipulator and Camera control.

4 is a log with the errors that has occured.

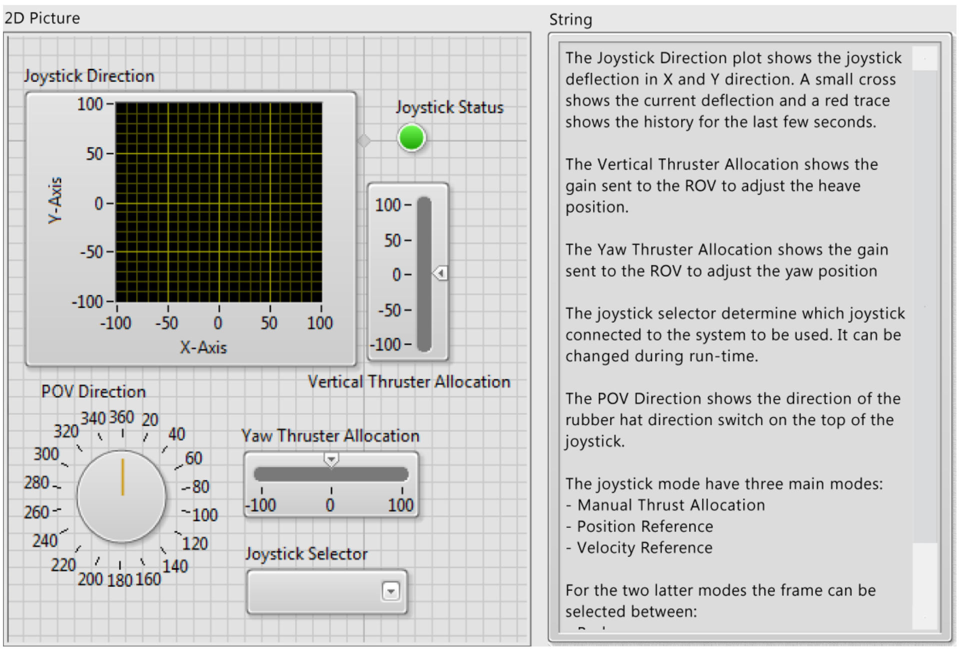
5 is the connect, exit and the three mode-buttons.

The Help field (6) shows tips and help instructions when the mouse is hovering over buttons or other areas of interest.

7 is a heading indicator that shows both the desired (Red Arrow) and measured (Blue) heading. These are the same colors as for the map. The desired and measured depth is also shown.

The eight led-lamps in 8 represent the status of the sensors installed at the ROV. Green means OK and red is error.

Figure G.1: The Main Display in Frigg Help Center.



The Joystick Direction plot shows the joystick deflection in X and Y direction. A small cross shows the current deflection and a red trace shows the history for the last few seconds.

The Vertical Thruster Allocation shows the gain sent to the ROV to adjust the heave position.

The Yaw Thruster Allocation shows the gain sent to the ROV to adjust the yaw position

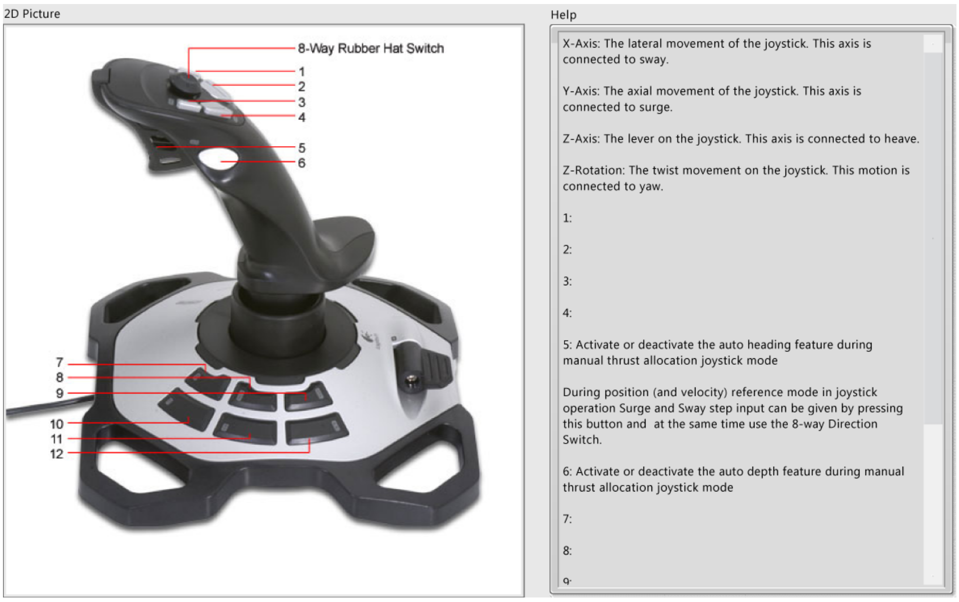
The joystick selector determine which joystick connected to the system to be used. It can be changed during run-time.

The POV Direction shows the direction of the rubber hat direction switch on the top of the joystick.

The joystick mode have three main modes:
- Manual Thrust Allocation
- Position Reference
- Velocity Reference

For the two latter modes the frame can be selected between:

Figure G.2: Joystick Help as displayed in Frigg Help Center.



X-Axis: The lateral movement of the joystick. This axis is connected to sway.

Y-Axis: The axial movement of the joystick. This axis is connected to surge.

Z-Axis: The lever on the joystick. This axis is connected to heave.

Z-Rotation: The twist movement on the joystick. This motion is connected to yaw.

1:

2:

3:

4:

5: Activate or deactivate the auto heading feature during manual thrust allocation joystick mode

During position (and velocity) reference mode in joystick operation Surge and Sway step input can be given by pressing this button and at the same time use the 8-way Direction Switch.

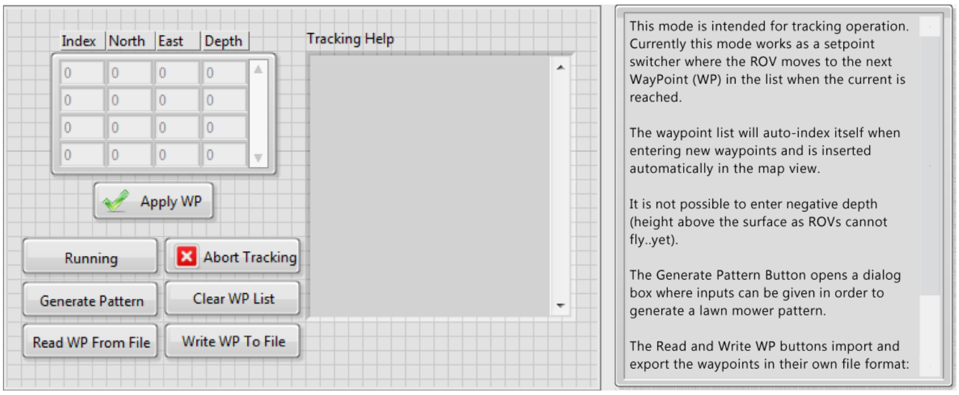
6: Activate or deactivate the auto depth feature during manual thrust allocation joystick mode

7:

8:

9:

Figure G.3: Joystick Button Configuration as displayed in Frigg Help Center.



Index	North	East	Depth
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

This mode is intended for tracking operation. Currently this mode works as a setpoint switcher where the ROV moves to the next WayPoint (WP) in the list when the current is reached.

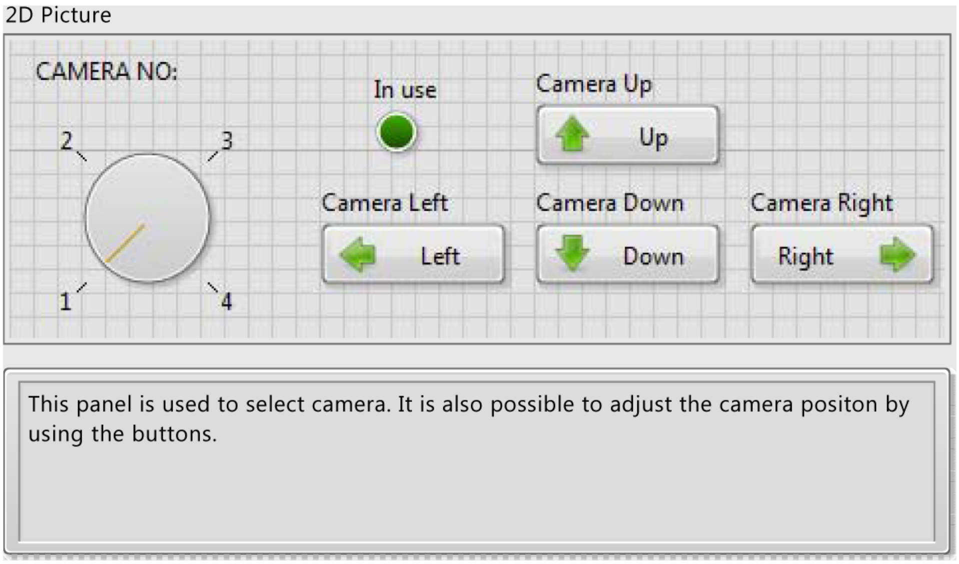
The waypoint list will auto-index itself when entering new waypoints and is inserted automatically in the map view.

It is not possible to enter negative depth (height above the surface as ROVs cannot fly..yet).

The Generate Pattern Button opens a dialog box where inputs can be given in order to generate a lawn mower pattern.

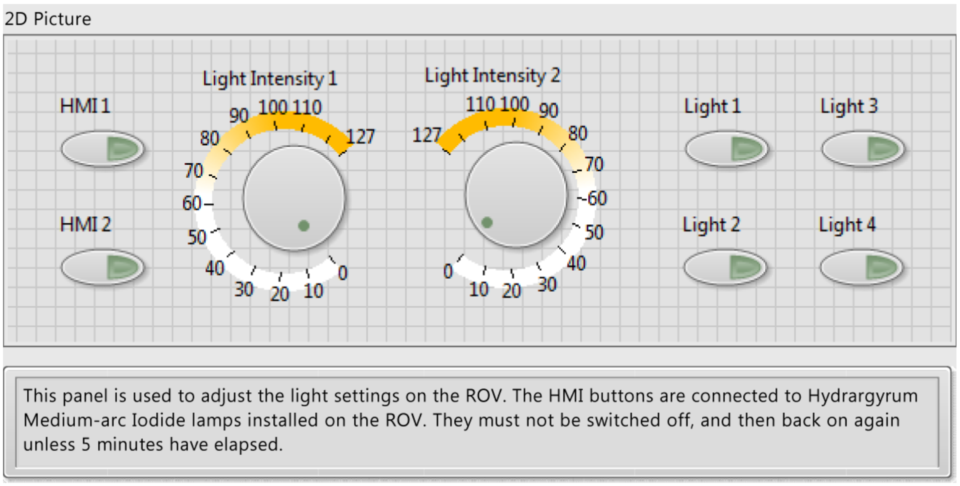
The Read and Write WP buttons import and export the waypoints in their own file format:

Figure G.4: Tracking Help as displayed in Frigg Help Center.



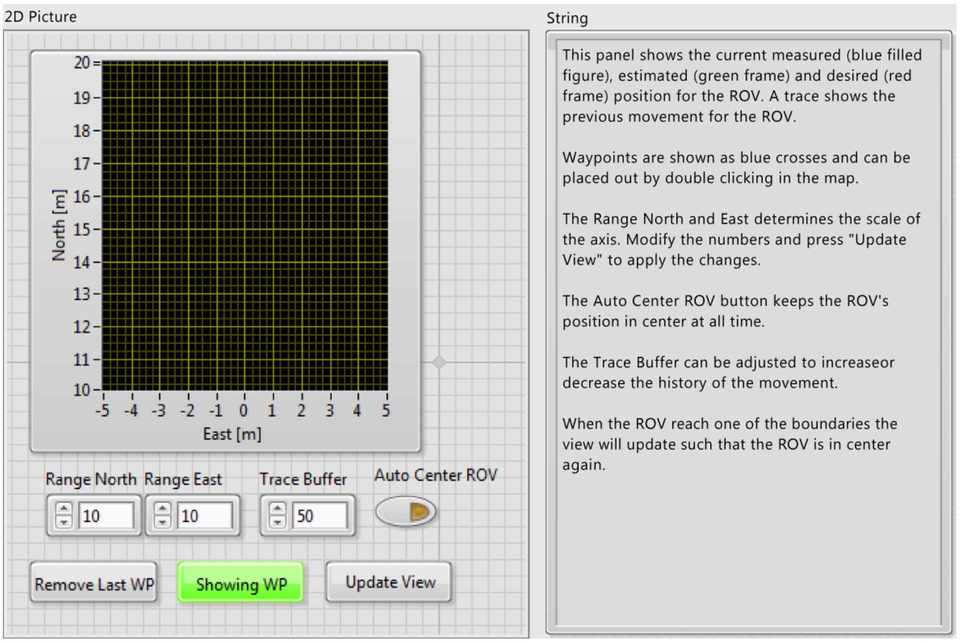
This panel is used to select camera. It is also possible to adjust the camera positon by using the buttons.

Figure G.5: Camera Panel Help as displayd in Frigg Help Center.



This panel is used to adjust the light settings on the ROV. The HMI buttons are connected to Hydrargyrum Medium-arc Iodide lamps installed on the ROV. They must not be switched off, and then back on again unless 5 minutes have elapsed.

Figure G.6: Light Panel Help as displayed in Frigg Help Center.



This panel shows the current measured (blue filled figure), estimated (green frame) and desired (red frame) position for the ROV. A trace shows the previous movement for the ROV.

Waypoints are shown as blue crosses and can be placed out by double clicking in the map.

The Range North and East determines the scale of the axis. Modify the numbers and press "Update View" to apply the changes.

The Auto Center ROV button keeps the ROV's position in center at all time.

The Trace Buffer can be adjusted to increaseor decrease the history of the movement.

When the ROV reach one of the boundaries the view will update such that the ROV is in center again.

Figure G.7: Map Visualization Help as displayed in Frigg Help Center.