



NTNU – Trondheim
Norwegian University of
Science and Technology

Development of a Simulation Platform for ROV systems

Sigrid Marie Mo

Marine Technology

Submission date: June 2015

Supervisor: Ingrid Schjølborg, IMT

Norwegian University of Science and Technology
Department of Marine Technology

Project description

In most marine operations today there is a low level of autonomy (meaning automatic control without human interaction). Introducing autonomy creates a number of challenges related to control. To develop advanced control algorithms there is a need for a robust simulation environment to verify and test algorithms.

In this master's thesis a simulation platform for ROV systems will be developed. The simulation platform will be developed based on the frameworks ROS (Robot Operating System) and MORSE (Modular OpenRobots Simulation Engine).

Scope of work

- Evaluate ROS as a simulator platform.
- Discuss advantages and disadvantages with the MORSE platform.
- Implement dynamic ROV model in both ROS and MORSE, verify models.
- Develop path planner, guidance system and controller modules in ROS.
- Establish necessary connections between the modules.
- Define system architecture.
- Verify and test the complete simulator.
- Document in a report.

The report shall be written in English, including description of mathematical models, control algorithms, simulation results, discussion and a conclusion including a proposal for further work. The source code should be provided in the attachments with code listing enclosed in appendix. The Department of Marine Technology, NTNU, can use the results freely in its research by referring to the students work. The thesis should be submitted within June 10th, 2015.

Ingrid Schjølberg
Supervisor

Abstract

ROVs are frequently used in subsea operations, both in the oil and gas industry, by scientists and in aquacultures. The ROV missions are normally controlled by a human operator, and the results are highly depending on her/his experience. A more predictable and independent operation can be achieved by introducing autonomy. When testing and verifying control algorithms for such an operation, a simulation environment can be a very helpful tool.

The objective in this thesis was to start the development of a simulation platform that can be used when developing and testing control systems for ROV operations. The platform will consist of two main parts; one simulator and a control system. The simulator part will include a dynamic model of an ROV in addition to sensor models. The control system will include an estimator, a guidance system, a path planner and a controller. The platform must be module based, such that users can implement their own algorithms easily and get immediate simulation results without needing to implement a complete control system. There should also be an easy transition between testing the control system on the simulated ROV and applying it to the real ROV.

The frameworks ROS (Robot Operating System) and MORSE (Modular OpenRobots Simulation Engine) are used in the development of the platform. ROS provides software management and communication protocols, while MORSE provides dynamics simulation with collision detection and implementation methods for 3D visualizations of the simulated vehicle.

A dynamic model of an ROV is implemented in both ROS and MORSE, and an evaluation is done to make recommendations for which model that should be used in the simulation platform. Two controllers, two guidance systems and one path planner is designed and implemented in the platform to test and verify the methods used for software communication.

The dynamic model implemented in ROS has proven to give realistic simulation results, but it is not able to simulate collisions. The dynamic model implemented in MORSE is able to simulate collisions, but it has other issued that needs to be fixed before it can be used in the simulation platform. The main issue is that MORSE is not able to process information received from the control system fast enough to do simulations in real time. There is also an issue with the inertia of the simulated vehicle, which makes the vehicle react too fast to thrust forces.

It is concluded that MORSE should be used in the further development of the platform. Although the dynamic model implemented in ROS gives better results for now, MORSE is found to have a greater potential as it provides collision detection and multiple sensor models.

Samandrag

Fjernstyrte undervassfartøy (ROV) er ofte nytta i undervassoperasjonar i olje-og gassindustrien, av forskarar og i fiskeoppdrett. ROV-operasjonar er vanlegvis kontrollert av ein operatør, og resultatet kan vere svært avhengig av erfaringane til operatøren. Ein meir føreseieleg og uavhengig operasjon kan bli oppnådd om ein introduserer autonomi til prosessen. Om ein skal teste og verifisere kontrollalgoritmar for slike operasjonar vil eit simuleringsmiljø vere eit godt hjelpemiddel.

Målet med denne masteroppgåva var å starte utviklinga av ei simuleringsplattform som kan bli nytta under utvikling og testing av kontrollsystem for ROV-operasjonar. Plattformen vil bestå av to hovuddelar; ein simulator og eit kontrollsystem. Simuleringsbiten inkluderer sensormodellar og ein dynamisk modell av ein ROV. Kontrollsystemet inkluderer ein estimator, eit styringssystem, ein baneplanleggarar og ein kontrollar. Plattformen må vere modulbasert slik at brukarar lettvis kan implementere eigne algoritmar og få simuleringsresultat utan å måtte implementere eit fullstendig kontrollsystem. Overgangen frå å teste kontrollsystemet i simuleringsmodellen og å nytte det på den fysiske ROVen bør vere så lett som mogleg.

Rammeverka ROS (Robot Operating System) og MORSE (Modular OpenRobots Simulation Engine) er nytta i utviklinga av plattformen. ROS stiller med programvarestrukturering og kommunikasjonsprotokollar, medan MORSE stiller med simulering av dynamikk inkludert kollisjonsmodellering og implementeringsmetodar for 3D-visualisering av fartøyet.

Ein dynamisk modell av ein ROV er implementert i både ROS og MORSE, og ei evaluering er gjort av dei to modellane for å foreslå kven av dei som bør vidareførast i simuleringsplattformen. To kontrollarar, to styringssystem og ein baneplanleggar er utforma og implementert i plattformen for å teste og verifisere metodane nytta til programvarekommunikasjon.

Den dynamiske modellen som er implementert i ROS har gitt realistiske simuleringsresultat, men den er ikkje i stand til å simulere kollisjonar. Den dynamiske modellen som er implementert i MORSE klarer å simulere kollisjonar, men den har andre feil som må rettast opp i. Hovudproblemet er at informasjon frå kontrollsystemet hopar seg opp, og simuleringa blir dermed ikkje gjort i sanntid. Det er også eit problem med tregleik av det simulerte fartøyet som gjer at ROVen reagerer kjappare enn han eigentleg skal når han blir utsett for krefter.

Det er konkludert med at MORSE bør bli nytta i vidare utvikling av simuleringsplattformen. Sjølv om den dynamiske modellen som er implementert i ROS gir betre resultat per dags dato, er det sett at MORSE har eit mykje større utviklingspotensiale sidan det blant anna stiller med kollisjonssimulering og fleire sensormodellar.

Preface

This thesis is a product of my work during the spring semester 2015 at the department of Marine Technology, Norwegian University of Science and Technology (NTNU). Parts of this work has been done as a continuation of my project thesis, fall 2014.

I would like to thank my supervisor Professor II Ingrid Schjølberg for guidance and encouragement during this work. I would also like to express my appreciation and gratitude to PhD candidate Bård Bakken Stovner for introduction and guidance in ROS, and to PhD candidate Eirik Hexenberg Henriksen for help with MORSE and for a good collaboration with the development of the simulator.

I would also like to thank Ph.D candidates Daniel de Almeida Fernandes and Mauro Candelloro for help and discussion about ROV modelling and for providing ROV data to use in my simulations.

Lastly, I would like to thank Anniken Amos Ruud for encouragement and support not only through the work of this master's thesis, but also through my 5 years of studies in Trondheim.

Trondheim, June 10, 2015

Sigrid Marie Mo

Contents

1	Introduction	1
1.1	Background and Motivation	1
1.2	Related Work	2
1.3	Problem Formulation	3
1.4	Software	4
1.5	Contributions and Thesis Outline	5
2	Simulation Platform Design: Goals and Vision	7
2.1	Open Source	7
2.2	Layout	7
2.3	Information flow	9
2.4	Required Modules	10
2.4.1	Dynamics	10
2.4.2	Sensor Models	10
2.4.3	Estimation and Filtering	11
2.4.4	Path Planning	11
2.4.5	Guidance Systems	12
2.4.6	Controllers	12
2.5	Configuration	12
2.6	Expansion possibilities	13
3	Existing software frameworks used in the Platform	15
3.1	Desired properties of software frameworks	15
3.2	Robot Operating System - ROS	16
3.3	Moduar OpenRobots Simulation Engine - MORSE	17
3.3.1	Blender	18
3.3.2	Bullet	19
4	Software Development	21
4.1	Software Communication	21
4.2	ROV Dynamics	22
4.2.1	Mathematical Modelling of an ROV	22
4.2.2	Implementation of the Mathematical Model in ROS	27
4.2.3	Implementation of mathematical model in MORSE	30
4.3	Path Planning	33
4.3.1	Path planning methods	33
4.3.2	Implementation of a path planner modules in ROS	35

4.4	Guidance	37
4.4.1	Guidance methods	37
4.4.2	Implementation of Guidance Systems in the Simulation Platform	38
4.5	Controller	42
4.5.1	Controller algorithms	42
4.5.2	Implementation of Controllers in the Simulation Platform	43
5	Software verification and testing	47
5.1	Verification of ROV Model in ROS	47
5.1.1	Simulations Without Current	47
5.1.2	Simulations With Current Present	49
5.1.3	Simulations with changes in mass and inertia	52
5.1.4	Simulations with thrust forces	53
5.1.5	Simulations with RPM control	55
5.2	Verification of Dynamic ROV Model in MORSE	58
5.3	Control, guidance and path planner testing	62
5.3.1	Module switch method	62
5.3.2	Simulation results	65
6	Simulation Platform Evaluation	69
6.1	MORSE Dynamics VS ROS Dynamics	69
6.2	Recommended User Knowledge	70
6.3	Execution of the Program	70
6.4	User Interface	71
7	Discussion and Conclusion	75
7.1	ROS as a framework	75
7.2	Configuring Simulations	75
7.3	Dynamic ROV model	76
7.4	The Complete System	77
8	Further Work	79
8.1	User Interface	79
8.2	Software Documentation and Development Guidelines	80
8.3	Software organization	80
8.4	Dynamics modelling	81
8.5	Improvements of Configuration File	82
8.6	General improvements in MORSE	82
8.6.1	Subscription to Controller Messages	82
8.6.2	Read configuration file	82
8.7	Expansion of the Simulation Platform	83
8.8	Post Simulation Plotting	83
	Bibliography	85
	Appendix A Software Organization in ROS	89
	Appendix B Digital Attachments	91

B.1	MORSE Files and Folders: <i>/uvst/morse/subsea</i>	91
B.2	Modifier classes: <i>/uvst/Modifiers</i>	91
B.3	ROS Files and Folders: <i>/uvst/catkin_ws</i>	92
B.4	Configuration files: <i>/uvst/catkin_ws</i>	92
B.5	Poster: <i>uvst/Poster</i>	92
Appendix C Modifier classes		93
Appendix D System Matrices		95
D.1	Mass Matrix	95
D.2	Coriolis and Centrepetal matrices	95
D.3	Restoring forces	96
Appendix E Vessel Data Minerva ROV		97
Appendix F Flow Charts for ROS Nodes		99
F1	ROS Dynamics module - Vehicle	100
F2	Path planning module - WPpublisher	101
F3	Guidance Module - LOS A	102
F4	Guidance Module - LOS B	103
F5	Path Controller Module - pathController	104
F6	DP Controller Module - DPcontroller	105

List of Figures

1.1	Pseudo block diagram for the planned platform. Control systems can be tested on a simulated ROV with simulated sensors (green), or tested on the real ROV with real sensor output (red).	3
2.1	Basic flowchart of planned simulation platform.	8
2.2	Flowchart visualizing the information flow between modules.	9
3.1	Robotic Operating System Logo, (Thomas, 2014a)	16
3.2	Terminal when running roscore	17
3.3	Blender logo (blender.org)	18
3.4	Graphical user interface for Blender	18
3.5	Bullet physics library logo (bulletphysics.org)	19
4.1	Principle of ROS Node communication	21
4.2	3D model of the 30K ROV, used for visualization in MORSE	30
4.3	MORSE visualization with 30K ROV and mist settings.	33
4.4	Different ways to generate paths from waypoints (Strømsøyen and Mo, 2014)	35
4.5	Terminal when running WPpublisher	37
4.6	Definition of angles and vectors needed in calculations for Line of Sight Guidance (Fossen, 2011, p. 259)	39
4.7	Terminal when running LOS Guidance Node B	42
5.1	ROV position and velocity in z-direction with no thrust forces	48
5.2	Forces when no thrust forces are active	48
5.3	ROV position and velocity in z-dir with no thrust forces, initial z-velocity = 1.0 m/s	49
5.4	ROV velocities when exposed to 1m/s current from south, ROV heading north	50
5.5	ROV velocities when exposed to 1m/s current from west, ROV heading north	50
5.6	Current 1m/s north-east, before bug fix related to Coriolis of added mass matrix	51
5.7	Current 1m/s north-east, from dynamic model in MATLAB	51
5.8	Current 1m/s north-east, after bug fix related to Coriolis of added mass matrix	52
5.9	Velocities in z-direction with changes in mass matrix	53
5.10	ROV velocity in x direction with applied thrust force in x direction	54
5.11	Comparing simulations from ROS with MATLAB. Forces: 200N in x-direction and 3Nm in yaw	55
5.12	ROV Minerva and thruster numbering	55

5.13 Results from simulations with RPM set on 700 and 1000 from thruster 4 and 5 respectively.	56
5.14 Simulations done with changing RPM on thruster 5, keeping thruster 4 on 1000RPM and the other thrusters on 0.	57
5.15 Simulations done with changing RPM on thruster 1, keeping the other thrusters at 0 RPM.	58
5.16 Comparing velocities in z-direction for MORSE and ROS with no thrust forces acting on the ROV.	59
5.17 Comparing results with and without variable simulation steps in MORSE	61
5.18 Node graph for scenario 1	63
5.19 Node graph for senario 2	64
5.20 Node graph for scenario 3	65
5.21 Results from simulations with path controller and guidance systems LOS A and LOS B guiding with a lawn mower pattern from the path planner. Current 0.05m/s to the east in the last two plots.	66
5.22 LOS A (upper) and LOS B (lower), current 0.05m/s north, straight path	66
5.23 Testing DP controller with current present. Current is going south, and ROV is heading north.	67
5.24 Results from simulation done with the MORSE dynamics, path controller and LOS A guidance system in the loop.	67
6.1 Screenshot of Blender during simulation	72
6.2 Example of 2D plot from rqt_plot package (Gregg, 2014)	73
6.3 Example of a flowchart made by rqt_graph	73
8.1 Proposal for visual presentation of simulation results	80
8.2 Proposed extension of the software organization.	81
E1 Flow chart for vehicle dynamics node <i>Vehicle</i> in ROS	100
E2 Flow chart for the path planning node <i>WPpublisher</i> in ROS.	101
E3 Flow chart for LOS node A in ROS. Functions in red are reused from the dynamics package	102
E4 Flow chart for LOS node B. Functions in red are reused from the dynamics package and functions in green are reused from the pathplanner package. . . .	103
E5 Flow chart for path controller. Functions in red are reused from the dynamics package.	104
E6 Flow chart for DP controller. Functions in red are reused from the dynamics package.	105

List of Tables

2.1	Short explanation of basic modules for simulation platform.	8
4.1	Criteria for waypoint generation (Fossen, 2011, p. 255)	34
4.2	Paths developed for path planner modules	36
5.1	Max depth increace as a function of the initial velocity in z-dir.	49
5.2	Time before maximum velocity is reached when added mass matrix is changed	53
5.3	Maximum ROV velocity when exposed to thrust force in x-direction	54
5.4	Roll angles for different RPM on thruster 2, with all other thrusters kept on 0 RPM.	57
5.5	How much MORSE manages to simulate in real time (approx.) with a given publication rate of controller output.	62
E.1	Thrust loss factors for each thruster	98

Nomenclature

Abbreviation	Description
AUV	Autonomous Underwater Vehicle
CO	Center of Orientation
COB	Center of Buoyancy
COG	Center of Gravity
DOF	Degrees of Freedom
DP	Dynamic Positioning
DVL	Doppler Velocity Log
ENU	East-North-Up
FSD	Forward Starboard Down
GUI	Graphical User Interface
I-AUV	Intervention AUV
IDL	Interface definition language
IMU	Inertial Measurement Unit
LOS	Line of Sight
MOOS	Mission Oriented Operating Suite
MORSE	Modular OpenRobots Simulation Engine
NED	North-East-Down
PID	Proportional-Integral-Derivative
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
RPM	Revolutions Per Minute
SFU	Starbord Forward Up

Symbols from thrust force calculation

Symbol	Description
K_T	Thrust Coefficient
τ_{thr}	Thrust force vector, body frame
D	Thruster diameter
n	Propeller revolutions per second
J_a	Advance ratio
V_a	Velocity of water through propeller
f	Thrust force vector, per thruster
θ	Thrust loss coefficient
\mathbf{T}	Thrust allocation matrix

Symbol from ROV Dynamics

Symbol	Description
\mathbf{M}_{RB}	Mass matrix
\mathbf{M}_A	Added mass matrix
\mathbf{C}_{RB}	Coriolis and Centripetal Matrix
\mathbf{C}_A	Coriolis and Centripetal Matrix of Added mass
\mathbf{D}_L	Linear damping matrix
\mathbf{D}_q	Quadratic damping matrix
\mathbf{G}	Restoring force vector
\mathbf{J}_q	Transformation matrix for quaternions
$\mathbf{R}_b^n(\mathbf{q})$	Transformation matrix for linear velocities
$\mathbf{T}_b^n(\mathbf{q})$	Transformation matrix for angular velocities
η	Pose vector
\mathbf{v}	Velocity vector, body frame
\mathbf{v}_r	Relative velocity vector, body frame
ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle

Symbols for Guidance Systems

Symbol	Description
χ_d	Desired course angle
χ_r	Velocity-path relative angle
α	Path-tangential angle
p_k	Waypoint nr k
β	Sideslip angle
K_p	Proportional gain for steering law
K_i	Integral gain for steering law
$e(t)$	Cross track error (normal to path)
$s(t)$	Along-track distance (tangential to path)
ψ_d	Desired heading
R_k	Radius of circle of acceptance
s_{k+1}	Along-track distance between waypoints p_k and p_{k+1} .

1. Introduction

1.1 Background and Motivation

A Remotely Operated Vehicle (ROV) is a tethered, unmanned, remotely operated underwater vehicle. Normally, an operator on board a surface vessel controls the ROV, and the control signals are sent through the tether. The ROVs are equipped with different tools and sensors, so that the ROV can perform different tasks and the operator can be aware of the situation and the behavior of the ROV.

The ROVs are highly maneuverable, but the hydrodynamic performance is normally far from ideal. This causes large drag forces, and the velocities achieved are not very large. The vessel is equipped with lights and cameras, and depending on the size, typically one or several manipulators.

ROVs can be a very helpful tool in many different fields. It is used for scientific research, sampling by marine biologists, for investigations by marine archaeologists, and for inspections and cleaning in aquacultures. They are also frequently used by the oil and gas industry for deep water installations, inspections, maintenance and repair.

Controlling an ROV to perform a certain task can be a very time consuming process, and it is highly dependent on how skilled the operator is. Increasing the level of autonomy in this kind of operation can save a lot of time, and therefore also a lot of money. Controlling the ROV with a control system based on sensor outputs could lead to a more robust operation where the performance is no longer depending on a skilled operator and her/his interpretation of the situation. A cleaner, independent and more predictable operation can be accomplished. The new way to execute the operation will of course introduce other problems that must be taken care of. The sensor outputs must be accurate enough and the control must be precise.

Developing and implementing such an advanced control system is a difficult task. A simulation model can be a very helpful tool in the process of developing control systems. The control system can be tested on a simulated model, which is both a lot cheaper and faster than if the control system could only be tested on the real process. The robustness of the system is easily tested with a simulation model, as the disturbances and environment can easily be changed.

The main challenge in such a process is to make the simulation model as close to reality as possible. There are many components that contribute to the final force acting on the ROV from the water, and the most important ones should definitely be taken into account when

making a simulation model. To model these forces accurately is however a very difficult task, and some assumptions must always be taken.

1.2 Related Work

Simulators of underwater vehicles have been developed multiple times before, but they are mostly made to fit a specific ROV or AUV (Autonomous underwater vehicle). As AUVs are autonomous, there have been a greater need for simulators to test control systems than for ROVs which can be controlled by an operator.

An extensive simulator made for ROV operations is ROVsim (marinesimulation.com). It is made to simulate a wide range of different ROVs and different operations for the oil and gas industry as well as other missions for observation class ROVs. It allows for the user to change the environmental factors during the simulation, and it provides a very realistic visualization. The drawback of this simulator is that it is a commercial product made for pilot training and thus not fitted for use with control algorithms.

In the RAUVI Project ([Novi et al., 2009](#)), which spanned from 2009 to 2012, they worked on improving and developing new systems for underwater intervention tasks for AUVs. This work was continued in the TRIDENT project ([Pedro J. Sanz, 2010](#)). They split the mission into two tasks: survey and intervention. The work included developing navigation techniques, coordinated AUV-Manipulator control, mapping algorithms for mapping the seabed and image processing. They implemented two dynamic AUV models in the simulation, SPARUS II and Girona 500. The research and findings from the TRIDENT project are important for other scientists for further development for autonomous control of intervention tasks subsea, but the simulation model itself will not be an asset for other researchers.

The TRIDENT project used ROS (Robot Operating System, [Quigley et al. \(2009\)](#)), which is a framework used to develop software for robotic applications. In addition to ROS, the simulator UWSim ([Prats et al., 2012](#)) was used for visualization. UWSim is an underwater simulator that was developed in connection with the RAUVI and TRIDENT projects. UWSim is however just a kinematic simulator, so it lacks the dynamics that is needed for realistic simulations of AUVs and ROVs.

Gazebo ([Koenig and Howard, 2004](#)) and MORSE (Modular OpenRobots Simulation Engine, [Echeverria et al. \(2011\)](#)) are simulators used in combination with ROS that includes realistic simulations for dynamics including collision physics.

The Gazebo simulator was initially designed for ground robots. The visualization is not fitted for underwater simulations and the graphics will not be visually realistic. [Kermorgant \(2014\)](#) combines the best parts of the UWSim and Gazebo simulators to make a general simulator for I-AUVs (Intervention AUVs). The dynamic simulation is taken from Gazebo, and the visualization is done in UWSim. The combination is done by using ROS as a framework. As Gazebo is made for ground robots, it lacks the dynamics related to hydrodynamical forces. This issue is solved by including a dynamic AUV model that calculates the hydrodynamic forces acting on the AUV. The forces are taken as input in Gazebo, where they are included as external forces acting on the AUV. The dynamic model used to calculate the forces is based

on Fossens robot-like vectorial model for marine craft (Fossen, 1991). The model is simplified and does not take coriolis forces or added inertia into account.

Demarco et al. (2011) implemented ROS on the YellowFin AUV. ROS was used in combination with MOOS (Mission Oriented Operating Suite, Newman (2002)). MOOS is a framework similar to ROS that provides communication methods between software nodes. MOOS was used because several modules for sensors commonly used in underwater robotics were already developed in this framework.

DeMarco et al. (2013) has also used ROS in combination with MOOS. Simulations were done of interaction scenarios between human divers and the VideoRay Pro 4 ROV. This work included the simulation engine MORSE that visualized the operation, in addition to assist with collision detection. The dynamic model of the VideoRay was made in ROS, velocity was calculated and fed to MORSE for integration and collision detection. The calculated pose is then sent back to ROS for the next time step.

1.3 Problem Formulation

The goal for this master thesis is to begin the development of a simulation platform for ROV systems. The platform will consist of two main parts, a simulator and a control system. The simulator will provide dynamic simulations of an ROV including models of the different sensors. The control system will be the software designed to control the ROV, and includes both an estimator, path planner, guidance system and a controller. The goal is that the simulation platform can provide a foundation for testing of control software without large modifications needed when the software is used on the real vehicle. A block diagram displaying the plan is shown in figure 1.1. The green part is the simulated dynamics and sensor models. They use the same type of input and provides the same type of output as a real ROV (block in red color) would.

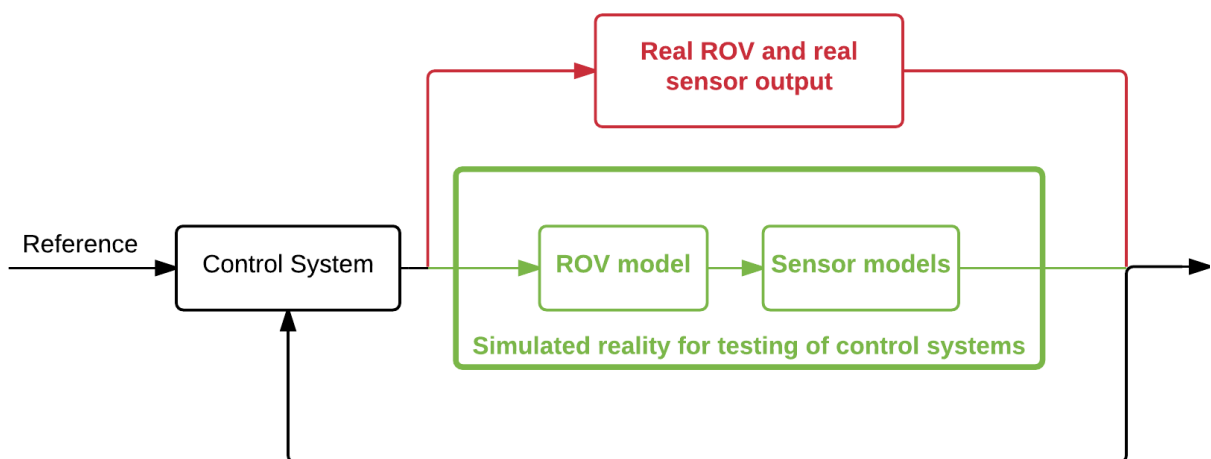


Figure 1.1: Pseudo block diagram for the planned platform. Control systems can be tested on a simulated ROV with simulated sensors (green), or tested on the real ROV with real sensor output (red).

The platform can be used for testing software for ROV operations and missions without physically testing the ROV in action. While developing the control system, it is tested on a simulated ROV with simulated sensor output, as shown in green in the block diagram. When the development is done, the software can be tested with the real ROV in the loop, as shown in red in the block diagram.

The simulation platform must be able to simulate ROV motions based on thruster inputs and environmental conditions. This includes realistic estimation of the hydrodynamic forces acting on the ROV, as well as a realistic estimation of the thrust forces for a given number of revolutions per minute (RPM) for each of the propellers.

The control system part of the simulation platform must include all necessary components to simulate ROV systems, including sensor models, an estimator, a path planner, a guidance system and a controller. The platform is planned to be module based, where each module performs a specific task. The modules communicate with each other, and it should be possible to replace one module with another module of the same type if desired. A practical example of this is if a user wants to test a new controller algorithm, she/he can replace the controller module in the simulation platform with her/his controller module where the new algorithm is implemented.

Due to time limitations, it is not seen as a realistic goal to develop the complete simulation platform in this project. The development of sensor models and estimators will be disregarded, as these modules are not necessary for testing controllers and guidance systems in combination with the dynamic model. The focus will rather be on implementing a realistic dynamic ROV model and test it on a few controllers, guidance systems and path planners. The dynamic ROV model will be implemented in two different frameworks, and the models will be evaluated and compared. Only one of these models will be included in the further development of the platform. It is also planned to develop two controllers such that a method for switching between modules can be not just designed, but also tested and evaluated.

The long term goal is that the simulation platform will be further developed and become a comprehensive simulation platform where developers can test their algorithms and get immediate results from the simulation.

The typical user will use the platform to test her/his algorithms. This can be e.g. an estimator, control or guidance algorithm. It is desired that the platform is developed such that the user easily can understand the platform and how to implement her/his code without comprehensive knowledge of underlying software and frameworks.

1.4 Software

The software frameworks ROS and MORSE will be used in the development of the platform. They are both open source, completely controlled from the terminal in a Linux based operating system and they both have a large community of developers and users.

ROS provides a modularity which is highly desirable for the simulation platform, as well as methods for message communication and package management. ROS makes it also pos-

sible for an easy transition between simulations and experiments, which is seen as a very important quality for the platform.

MORSE provides both 3D visualization and dynamic simulations including collision detection. This is also provided by e.g. the Gazebo simulator, but MORSE was chosen because the editor used for designing robots and environments seemed to be easy to use, as well as the simulation set-up was simple, the documentation was easier to understand and it has been used for underwater simulations before.

It is not yet decided if MORSE will be used for the ROV dynamics, or if this should be done in ROS, like it has been done in other projects. An implementation of the ROV dynamics will be done in both frameworks, and then the methods will be evaluated and compared.

1.5 Contributions and Thesis Outline

The contribution of this thesis is the basis of a simulation platform where users can implement their own algorithms and get fast simulation results.

The outline of the thesis follows.

In **Chapter 2** the goals and visions for the simulation platform are presented. This includes the desired layout of the platform and a discussion of the properties and qualities the platform should have.

In **Chapter 3** the two software frameworks used in the platform, ROS and MORSE, are presented, as well as a discussion of why these were chosen.

In **Chapter 4** explains how the software development is done. This includes the theory behind the different modules and the software implementation.

In **Chapter 5** verifications of the dynamic modules are presented, as well as verifications of the module switch system, the guidance systems, controllers and path planners.

In **Chapter 6** an evaluation of the simulation platform is done. This includes a discussion of the dynamic models developed for the platform and discussions about the user-friendliness of the platform.

In **Chapter 7** the developed simulation platform is discussed, and a conclusion is made about the further development of the platform.

In **Chapter 8** recommendations are made for further work.

2. Simulation Platform Design: Goals and Vision

Before developing the simulation platforms, there was made plans for the platform regarding desired qualities and properties. This chapter will present the plans made before the software development started.

2.1 Open Source

The complete simulation platform is planned to be open source. This means that the software is available online on a project website, and users all over the world can download and use it free of charge. This is done to encourage users to join in the further development of the platform. If the platform is open source, it is assumed to attract both more users and more developers. The more people using the simulation platform for their projects, the greater is the chance for someone to discover software weaknesses, flaws or faults.

For the platform to be open source, a good documentation of the software is required. When using existing frameworks in the development of the platform, they are also required to be open source.

2.2 Layout

The simulation platform will be a collection of software that works together in simulation of ROV operations. The goal is that developers can use this platform to test their algorithms, whether they are making algorithms for controllers, sensor processing, guidance systems or path planners.

The simulation platform is to be designed based on modules. A module is a collection of software developed to do a certain task. In each module, a set of information is processed and meaningful output produced. The output produced by one module will be used as input in another module, and the whole system will be a continuous loop of information flow.

Modules needed for a basic simulation platform are vehicle dynamics, sensor models, estimator, guidance, path planner and a controller. A simple explanation of these modules are given in [2.1](#) and a flowchart is shown in figure [2.1](#).

Table 2.1: Short explanation of basic modules for simulation platform.

Vehicle dynamics:	Desired RPM for each thruster is used as input and resulting motion of ROV is estimated. The vehicle dynamics can be extended to include e.g. manipulator dynamics, which will need another type of input to control the joints.
Sensor Model:	Simulated motion and position of ROV is used as input, and simulated sensor output is produced based on type of sensor and appropriate noise and bias. Navigation systems are also included in this group, meaning systems that combine output from different sensors.
Estimator:	Estimates the system states based on sensor output.
Guidance:	Decides desired system states based current and desired motions of the ROV.
Path Planning:	Finds desired path for ROV to follow.
Controller:	Uses information about current ROV states and desired states to calculate RPM for each thruster such that the ROV will act as desired.

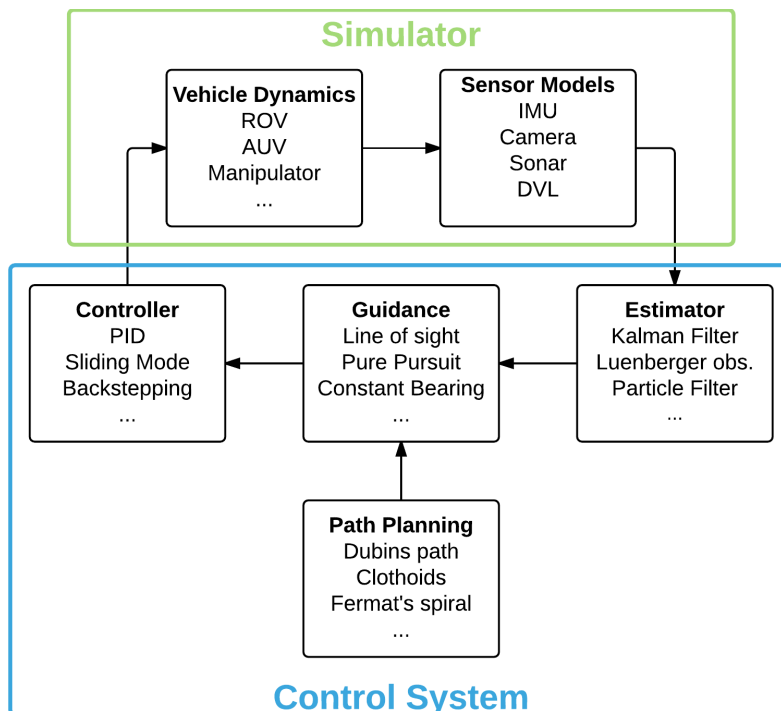


Figure 2.1: Basic flowchart of planned simulation platform.

A module of a certain type can be developed based on different methods, resulting in the possibility to run multiple modules doing the same tasks.

2.3 Information flow

The modules must communicate with each other. An example is the controller module that will receive information about the ROV position and velocity, in addition to receiving information from the guidance system. When the controller has calculated desired RPM for the ROV thrusters, this information must be sent to the vehicle dynamics module. If multiple controller modules exist, only one of them can send control values to the vehicle dynamics module.

Figure 2.2 shows how the communication flow is planned if multiple modules of the same type exists. A switch must be included to make sure the correct messages are used in the loop.

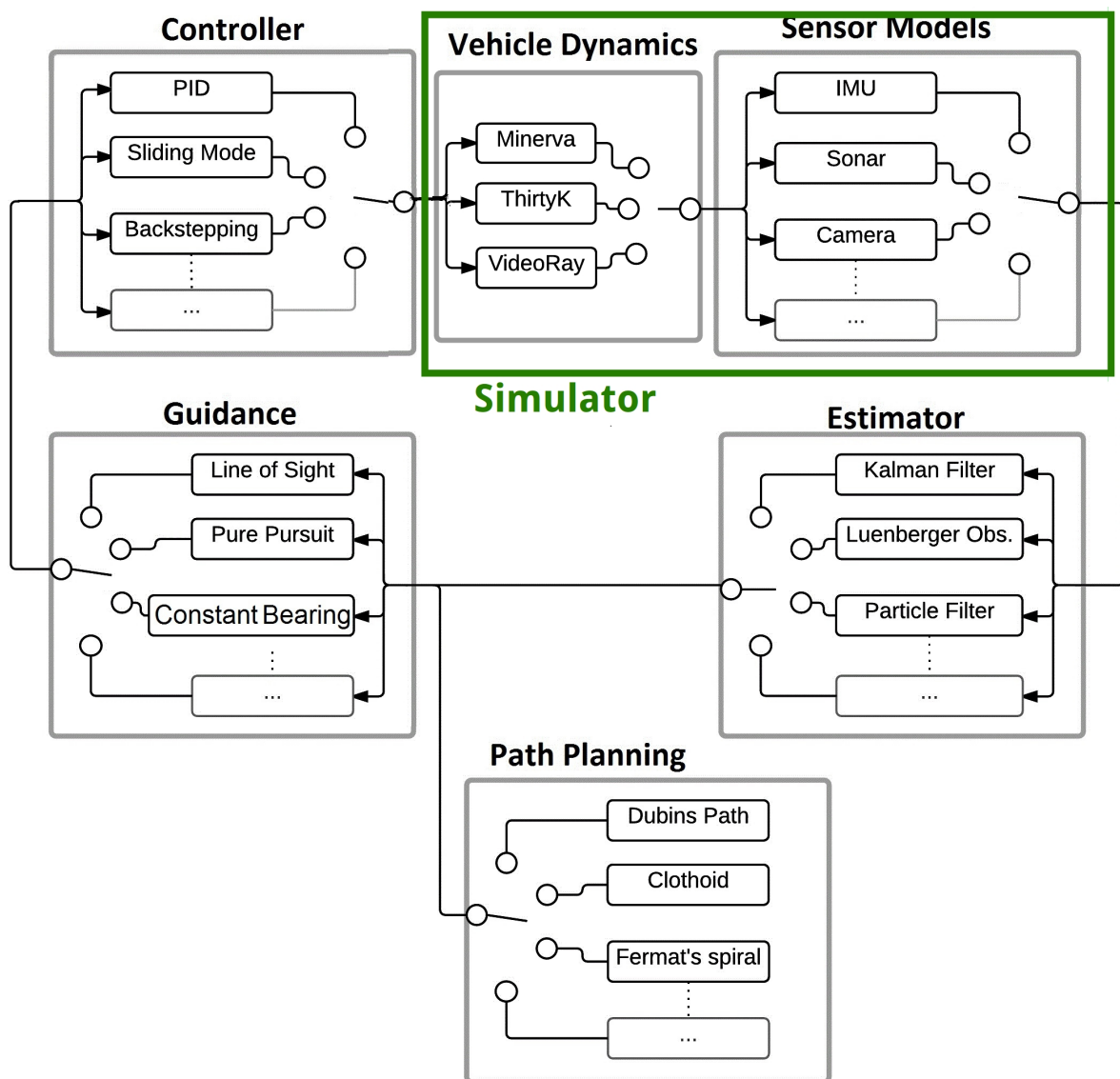


Figure 2.2: Flowchart visualizing the information flow between modules.

It is important to note that the flow chart is made only to show the overlying plan of the sim-

ulation platform. Some details are left out, e.g. the different controllers may need different sets of input. A controller made for dynamic positioning will need a different input than a heading controller designed for path following. It should also be possible to keep multiple sensor models in the loop.

2.4 Required Modules

2.4.1 Dynamics

The module for vehicle dynamics must be able to simulate different ROVs in different weather conditions. This module should simulate how the ROV moves based on input from the controller.

The vehicle dynamics module must be able to estimate the thrust forces based on control input, and estimate the ROV motion based on thrust forces and ROV specifications. Basic specifications are needed for the simulation, such as mass, volume and inertia of the ROV. In addition, the initial pose and velocity must be known for the resulting motion to be calculated.

The controller input is chosen as revolutions per minute (RPM) for each thruster on the ROV. To estimate the thrust, not only RPM control values given by the controller are needed, but also information about the thrusters, such as thrust coefficients, thrust loss factors and thrust allocation.

Simulation of the environmental forces acting on the ROV is also a desired property of the dynamics module. Simulating wave forces can be a challenging task, but will be necessary when simulating ROV motions in the wave zone. Current will also highly affect the ROV motion, and must also be included in the dynamic model. In the future, it should be evaluated if the environmental conditions can be implemented as a separate module.

Forces from the umbilical is also an element desired to include in the dynamic ROV model. Depending on sea current and how the ROV is deployed, the forces acting on the ROV from the umbilical can vary greatly and will affect the ROV motion. An estimation of these forces is desirable to include in the dynamics module as they can give a more realistic simulation of real ROV motions.

2.4.2 Sensor Models

The sensor models should simulate real sensor output based on the simulated vehicle motions calculated in the vehicle dynamic module. Output from these modules can vary, as different types of sensors will measure different states. For now, navigation systems are also included in this group.

As the vehicle dynamics module will give the true simulated values, the task of the simplest sensor models will be to read the true value of the relevant system states and include bias and noise in the simulated sensor output.

For more advanced sensor models, the tasks will be more comprehensive. Simulating e.g. a GPS must include the calculation of vessel position and pose based on the "measured" position of the antenna from the GPS signal. The GPS signal noise and bias will therefore be processed, and the resulting noise in the sensor output will differ from the one introduced in the GPS signal. A GPS is however only used at the sea surface, but there are multiple types of sensors used subsea that also process noisy signals to calculate the sensor output.

It would also be interesting to include signal errors in the sensor models, to simulate how the estimators and control systems react to sensor faults. This can be done e.g. by including wild points, increasing noise variance or simply stop the sensor model output.

2.4.3 Estimation and Filtering

An estimator must process the sensor output to estimate the true states and possibly also estimate states that are not measured. This includes filtering out the noise from the sensors, and also detect bias and sensor faults.

The estimators can be based on different algorithms. Simple filters will only estimate measured states, where more advanced filters can also estimate missing states and bias. This requires that the not measured states are observable.

If the ROV is to be controlled in the wave zone, it is desirable to include a wave filter as well. The wave filter will estimate ROV motion caused by the wave, and this can be taken into account by the controllers. If the ROV is just moving through the wave zone, it is not desirable to counteract the wave motion. It may not even be possible to counteract the wave motion as the forces involved are very large. In this case it is especially desirable to use wave filtering and not include the wave frequency part of motion in the feedback for the controller, such that not all thrust capacity is wasted trying to reach impossible standards.

2.4.4 Path Planning

A path planning module should take care of designing a path for the ROV to follow. This is necessary when e.g. investigating the sea bottom or subsea pipelines. When investigating subsea pipelines, the ROV should keep constant distance above the pipeline while slowly following the pipe along the sea bottom.

Designing a path planning module can be done in many different ways, and the different path types will require different input values. Paths for pipeline investigation requires some information about where the pipe is. When including obstacle avoidance in the path planning, information about position of obstacles are needed.

Paths can be given as just a path in space, but also a path in space with time constraints, called a trajectory. Paths in space can be given as waypoints, a set of coordinates defining points on the path. Trajectories can be given as coordinates as well, but now as a function of time.

2.4.5 Guidance Systems

Guidance systems are necessary for path following to guide the ROV through the desired path set by the path planning module. A typical guidance system will decide a desired heading angle that will lead the ROV to the path.

The guidance system will typically use the waypoints from the path planner module in combination with the estimated ROV position from the estimator module. If the ROV is far away from the path, the guidance system should calculate a heading angle that will guide the ROV back on track.

It is desired that the guidance system takes current into account. If there is current present, the ROV will need a different heading to manage to follow the path than if there is no current. The guidance system will not have any knowledge of the current, i.e. there is no measurements used to estimate the current (although it is possible to design such a system). The guidance system can use information about the ROV velocity to estimate a heading correction due to current.

2.4.6 Controllers

The controller must decide desirable RPM for each thruster on the ROV based on output from the guidance system and estimated position and velocity.

The type of controller needed will highly depend on the type of operation. If the ROV is to follow a path, a heading controller is needed, possibly in combination with a speed and depth controller. If the ROV is to keep a certain position, as in dynamic positioning, all controllable degrees of freedom must be controlled, i.e. position in the north-east-down (NED) plane and heading. Underwater vehicles are often underactuated, and pitch and roll cannot be controlled.

The different controllers will demand different sets of input, but they will all give RPM for each thruster as output. The number of RPM values in the output must equal the number of controllable thrusters.

2.5 Configuration

The dynamic ROV model should be made as general as possible such that the same model can be used for multiple ROVs. It is therefore planned to include information about the ROV in a configuration file. When the user want to simulate another ROV, the configuration file must be updated with the new values. The file must typically include information about mass, inertia, added mass, added inertia, damping, etc. The dynamics module should then read the configuration file to gather information before the simulation starts.

It is also planned to include information about the initial position and velocity of the ROV in the configuration file. If this is included in the file, it will be easy for the user to change the initial conditions for e.g. testing how the controller works in the different conditions.

An important property of the simulation platform is that it must be possible to switch between modules of the same type. Deciding which modules to include in the loop must be done by the user, and one solution is to include information about which modules to run in the configuration file.

2.6 Expansion possibilities

The simulation platform must be developed to support and encourage expansions of the platform. It has the possibility of including thousands of modules if the development is done in a systematic way. It is therefore important to prepare a set of development guidelines, such that there is room for multiple developers working to improve and extend the platform while keeping the maintenance workload from going through the roof.

The module types explained in section [2.4](#) only includes the basics for a simulation of ROV systems. There is plenty of possibilities to design new types of modules, and to implement these in the platform.

3. Existing software frameworks used in the Platform

The simulation platform has not been built from scratch. Simulations and control of vehicles and robots have been done multiple times before. As many components and solutions for the desired properties for the platform already are developed, a reuse of the software is highly recommended.

3.1 Desired properties of software frameworks

When choosing which existing frameworks to reuse, we need to reflect on the properties desired for the simulation platform. It is concluded that the following 4 properties is mainly what we are looking for in external software.

- 3D Visualization
- File system organization
- System for communication between modules
- Easy transition between simulation and experiment
- Open source

The most obvious desirable property is a visualization of the simulated ROV in operation. The visualization should be a 3D model of the ROV, and it should show how the ROV moves in real time during the simulation. This will make it easier for the user to follow the simulation and to interpret the results. Rather than just presenting a graph of the roll angle, the user actually sees the 3D model roll.

Another highly desirable property is a file system organization. As the simulation platform has the potential of including thousands of modules, they must be organized to avoid confusion. It is also desired to use a framework that provides a good method for communication between the modules.

As the simulation platform will be made to make it easy for users to test control algorithms before using them on a real vehicle, it is important that the transition between using the software on in a simulator and using it on the real ROV is as small as possible.

The last property is that the software should be open source. A goal for this simulation platform is to invite other developers to contribute to the platform. To reach a larger group of

users and developers, the simulation platform should be open source. This requires that all frameworks used must also be open source.

3.2 Robot Operating System - ROS



Figure 3.1: Robotic Operating System Logo, (Thomas, 2014a)

ROS (Quigley et al., 2009) is the software framework decided to use as a foundation in the simulation platform. ROS is a software platform used to develop software for robotic applications. It is chosen mainly because of the modularity it offers, because it gives a small transition between simulation and experiment and due to the fact that it is open source.

The name Robotic Operating System can be a bit confusing, as it is not really an operating system such as Microsoft Windows and Linux, but it is defined as a "meta-operating system" (Thomas, 2014a):

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

When implementing algorithms in ROS, they are implemented as ROS Nodes. A node will perform computations, and ROS provides a communication system such that the nodes can exchange messages with each other. It is even possible for nodes to communicate with each other when they are run on different computers.

Package management used for structuring the developed software is also provided. A ROS package is a folder that follows a certain structure. The folder can contain ROS nodes, message types, headers, configuration files, etc. The main properties of the software organization in ROS is explained in appendix A.

ROS must be run on a Unix-based platform, and it is completely controlled from the terminal. For using ROS, *roscore* must be running. This is done simply by writing `<roscore>` in a terminal. The terminal will then look as shown in figure 3.2.

```

roscore http://sigridmarie-Latitude-E7440:11311/
sigridmarie@sigridmarie-Latitude-E7440:~$ roscore
... logging to /home/sigridmarie/.ros/log/f9b4b092-faf5-11e4-b3e8-a0a8cd5e6514/r
oslaunch-sigridmarie-Latitude-E7440-5275.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://sigridmarie-Latitude-E7440:36939/
ros_comm version 1.11.10

SUMMARY
=====

PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.10

NODES

auto-starting new master
process[master]: started with pid [5287]
ROS_MASTER_URI=http://sigridmarie-Latitude-E7440:11311/

setting /run_id to f9b4b092-faf5-11e4-b3e8-a0a8cd5e6514
process[rosout-1]: started with pid [5300]
started core service [/rosout]

```

Figure 3.2: Terminal when running roscore

The ROS nodes cannot be initialized before roscore is up and running. When both roscore and the nodes are running, ROS can provide information about how the nodes communicate with each other, and which topics they use. Developed packages for ROS can then be used e.g. to plot the numerical values in the exchanged messages, and to give a graphical presentation of the communication flow between the nodes.

A great advantage of ROS is that a large group of developers use and improve the software continuously. They develop new packages that everybody can use, and everything is documented. This makes it easy to reuse software others have developed, and for other developers to reuse software developed for the simulation platform.

3.3 Moduar OpenRobots Simulation Engine - MORSE

The open source robot simulator MORSE (Modular OpenRobots Simulation Engine, (Echeverria et al., 2011)) provides visualization and physics simulation. Both these properties are desirable when developing a simulation platform.

MORSE is compatible with ROS. This means that MORSE can both send and receive messages sent through ROS topics. Morse is controlled from the terminal, and all software im-

plemented is written in python.

Morse reuses components from other open source projects. MORSE runs Blender (blender.org) for visualization and Bullet (bulletphysics.org) for rigid body physics simulation.

3.3.1 Blender



Figure 3.3: Blender logo (blender.org)

Blender is a computer graphics software used for making 3D models and animations (among other things). MORSE uses Blender to visualize the simulated robots in a simulated environment. Blender can be used for designing a 3D model of the vehicle we want to simulate. Figure 3.4 shows the user interface of Blender when designing a 3D model of a submarine. The submarine is one of many robots included in MORSE.

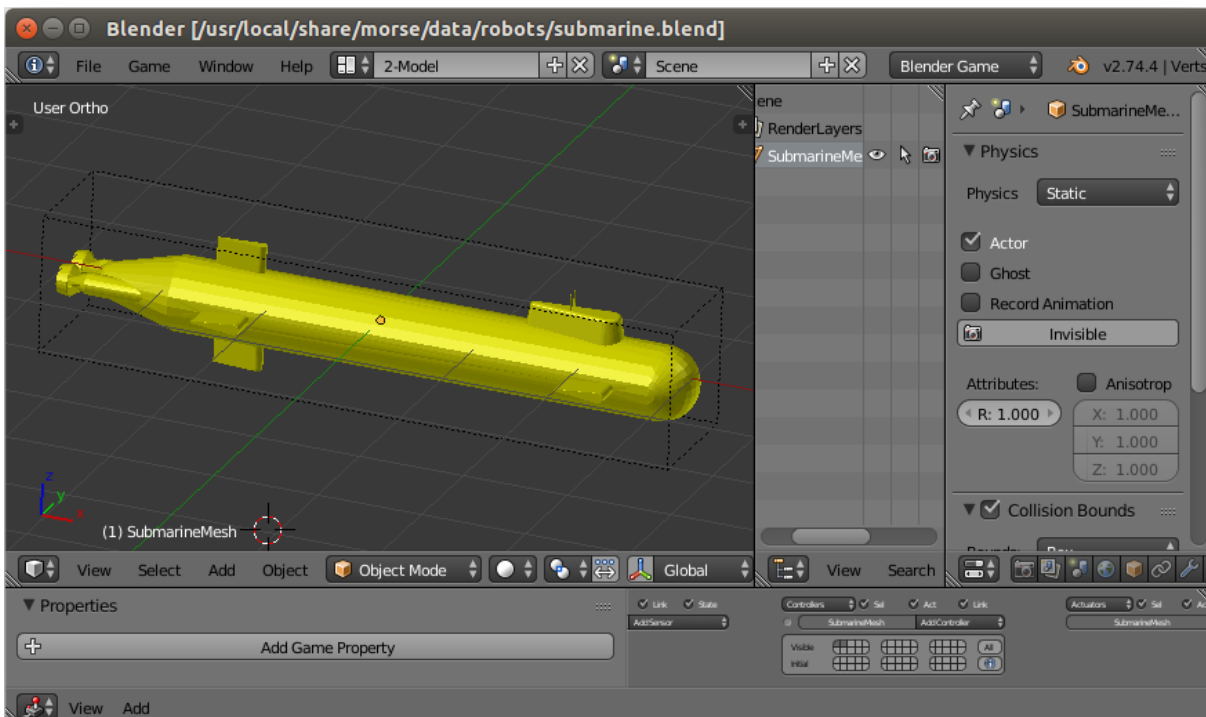


Figure 3.4: Graphical user interface for Blender

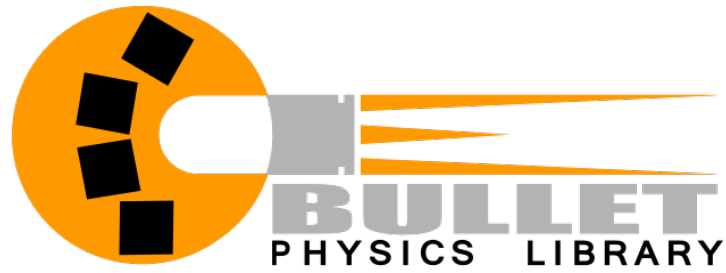


Figure 3.5: Bullet physics library logo (bulletphysics.org)

3.3.2 Bullet

Bullet is a physics engine that provides simulations of rigid and soft body dynamics (among other things). A large library of software does the calculations, and this software is used by Blender to simulate robots. Realistic motion of the robots when exposed to different forces chosen by the user is simulated. It is also possible to simulate multiple robots at once, and collision simulation is supported.

4. Software Development

4.1 Software Communication

A very important property we want in the simulation platform is a good method for communication between the modules. The modules must be able to send messages back and forth, and the messages must obviously be delivered to the right modules.

ROS provides something called a ROS network. The network consists of a ROS Master Node and several ROS nodes. The Master node makes it possible for the ROS nodes to communicate with each other. A node can publish messages on a topic, and other nodes can subscribe to these messages. A topic is basically just a text string with a descriptive name. The principle is shown in figure 4.1.

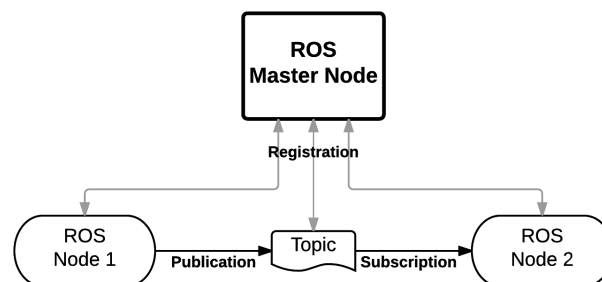


Figure 4.1: Principle of ROS Node communication

It is possible for multiple nodes to subscribe to the same topics, and it is also possible for multiple nodes to publish on the same topics. This communication method is an interesting solution to use in the simulation platform as it provides multiple possibilities for designing a good method to guide the information flow, as explained in section 2.3.

As messages are exchanged between nodes, all modules in the simulation platform except for the MORSE dynamics are implemented as ROS nodes. MORSE supports ROS, and this includes that messages can be exchanged between ROS nodes and MORSE.

All messages exchanged in the simulation platform are defined by using an interface definition language (IDL). This is done to support messages being exchanged between modules written in different programming languages. The result is that even if all software in MORSE is written in python, and all software in ROS is written in C++, they can still exchange messages by using IDL for describing the messages.

An example of how an IDL file is composed is shown in listing 4.1. In this example, the message to be sent consist of two variables, *variable_name1* and *variable_name2*. The messages are defined as Float32, which means that they are 32-bit floating numbers. In practice, this means that the variables can be numbers with decimals. The first line, "Header header" is there to include a time stamp in the message, such that the receiver knows when the message was sent.

Listing 4.1: IDL file example

```
Header header
Float32 variable_name1
Float32 variable_name2
```

A method is developed for controlling the communication flow between the modules. It should be easy to switch between different modules of the same type, such that the user can implement and test her/his algorithms.

One idea was to make all modules of the same type publish information on the same topic. The simulation platform could then only run one module of a certain type at a time to avoid multiple nodes publishing information on the same topic. The receiving node would not be able to separate the messages from each other, and it would subscribe to all the messages. If we wished to switch between modules while in simulation, the new node must be initiated, and the old one must be shut down. This is a very cumbersome way to do simulations if the modules are to be switched relatively often.

The method chosen was to let the modules of the same type publish messages on different topics. It is then possible for multiple modules of the same type to run at once, and the subscriber choose which topic to subscribe from. The subscriber can then easily switch between which topic it subscribes to during simulation, and there is no need to shut down or initiate new nodes during simulation.

4.2 ROV Dynamics

A dynamic model of an ROV has been developed using two different methods. When made correctly, they both should simulate the same ROV motions given equal input values. The dynamic model uses control input from the controller node and ROV specifications such as mass, inertia, volume etc to estimate resulting motion of the ROV. Different ROVs can be simulated, as long as the ROV specifications are provided. These must included in the configuration file as explained in section 2.5.

4.2.1 Mathematical Modelling of an ROV

A mathematical model is derived from Newtons 2nd law for rotation and translation, where all the forces acting on the ROV should be included to obtain a perfect model. Identification of all forces perfectly is not an easy task, and some simplifications has therefore been made.

The ROV is assumed to be a rigid body moving in sea, meaning that no bending or deformation is taken into account. This means that the front part of the ROV can not move or rotate faster or slower than the rear part.

The degrees of freedom included in the model are translation in x, y and z-direction and rotation about the x, y and z axis; surge, sway, heave, roll, pitch and yaw respectively. The origin of the coordinate system is placed in the center of orientation, CO. The velocity, v , of the rigid body in the different degrees of freedom relative to the body coordinate system is given in (4.1). The velocities u , v and w are velocity of the rigid body along the x, y and z axis. The velocities p , q and r are angular velocities about the x, y and z axis.

$$v = [u \quad v \quad w \quad p \quad q \quad r]^T \quad (4.1)$$

The North-East-Down (NED) coordinate system is used to describe the position of the ROV relative to a reference point at sea level. The origin is positioned at the sea surface, possibly where the vessel deploying the ROV is, and the axes are pointing to the north, east and down. The local body coordinate system has the origin in the center of orientation of the ROV, and is used to describe both linear and angular velocities.

Two different ways are used to describe attitude. For simple and physical understanding, euler angles are used. They are described by the greek letters ϕ , θ and ψ that represents motion in roll, pitch and yaw respectively. This results in the representation of η as shown in (4.2a). The other way to describe attitude is by the use of unit quaternions. This representation is used when the angles must be transformed from the body coordinate system to the NED coordinate system. If this is done with euler angles, the transformation has a singularity when the pitch angle reaches $\pm 90^\circ$. The unit quaternion avoids this problem by using four values to describe the three angles. The unit quaternion q has unit length ($q^T q = 1$) and consists of one real part η and three imaginary parts ϵ ; $q = [\eta \quad \epsilon_1 \quad \epsilon_2 \quad \epsilon_3]^T$. The representation of η when using unit quaternions is shown in (4.2b). Further descriptions of the unit quaternion are presented by (Fossen, 2011, p. 27-28).

$$\eta_{euler} = [N \quad E \quad D \quad \phi \quad \theta \quad \psi]^T \quad (4.2a)$$

$$\eta_{quaternion} = [N \quad E \quad D \quad \eta \quad \epsilon_1 \quad \epsilon_2 \quad \epsilon_3]^T \quad (4.2b)$$

The equation of motion for a marine vessel, (Sørensen, 2013, p. 201), was used as a starting point for the mathematical model of the ROV. Some modifications had to be made for the equation to fit an ROV model. Mooring forces are not taken into account, as the ROV is not in any way attached to a mooring system. Forces from the umbilical are included, but ice forces have been neglected. The resulting equation is given in (4.3). The equation is still a continuous equation describing the connection between the movements and forces. The mathematical expression for the matrices is taken from Fossen (2011), and can be found in appendix D.

$$(\mathbf{M}_{RB} + \mathbf{M}_A) \dot{\boldsymbol{v}} + \mathbf{C}_{RB}(\boldsymbol{v}) \boldsymbol{v} + \mathbf{C}_A(\boldsymbol{v}_r) \boldsymbol{v}_r + \mathbf{D}(\boldsymbol{v}_r) \boldsymbol{v}_r + \mathbf{G}(\boldsymbol{\eta}) = \boldsymbol{\tau}_{thr} + \boldsymbol{\tau}_{tether} + \boldsymbol{\tau}_{env} + \boldsymbol{\tau}_{wave} \quad (4.3)$$

Mass matrices

\mathbf{M}_{RB} is the mass matrix. This matrix includes the mass of the ROV (in air) in addition to the moments and products of inertia. The matrix is dependent of the mass and the mass distribution of the ROV, and will therefore change if the equipment on the ROV is removed or replaced with other equipment. The matrix structure is shown in Appendix D, equation (D.1).

\mathbf{M}_A is the added mass matrix. The term $\mathbf{M}_A \boldsymbol{v}$ represents the part of the resistance force that is proportional to the acceleration of the ROV. \mathbf{M}_A is assumed to be constant and dependent on the shape of the vessel.

Coriolis and centripetal matrices

The coriolis effect is the deflection a moving object get when it moves in a rotating reference frame (Myrhaug, 2006). There is not really a force acting on the object, but as the reference frame rotates it seems like the object deflects from its path.

Newtons 2nd law is only valid in a an inertial (i.e. non-accelerating) reference frame. For the law to be valid when used in a rotating reference frame, the coriolis and centripetal forces must be included. They are fictitious forces used to describe the motion due to the rotating reference frame.

$\mathbf{C}_{RB}(\boldsymbol{v})$ is the coriolis and centripetal matrix of the rigid body. The term $\mathbf{C}_{RB}(\boldsymbol{v}) \boldsymbol{v}$ represents a fictitious force that causes a movement of the ROV relative to the rotating reference frame NED due to the earths rotation. The matrix is a function of added mass and velocity relative to the NED reference frame.

$\mathbf{C}_A(\boldsymbol{v}_r)$ is the coriolis and centripetal matrix of the added mass. This is a function of the added mass and the relative velocity of the ROV taking current into account. The matrix structures of both \mathbf{C}_{RB} and \mathbf{C}_A are taken from Fossen (2011) and given in Appendix D, equation (D.3-D.4).

The moments given by the coriolis and centripetal matrices represents physical moments that must be included also when the equation of motion is used in an inertial reference frame.

Damping matrices

$\mathbf{D}(\boldsymbol{v}_r)$ is the damping matrix. The term $\mathbf{D}(\boldsymbol{v}_r) \boldsymbol{v}_r$ represents the viscous damping force from the water due to velocity of the ROV. The damping is not linear, but can be estimated to higher order terms from model testing. In the dynamic model developed in

this master thesis, the damping is represented as one linear and one quadratic term (4.4).

$$\mathbf{D}(v_r)v_r = \mathbf{D}_L v_r + \mathbf{D}_q \text{diag}(|v_r|) v_r \quad (4.4)$$

Restoring Forces

$G(\eta)$ is the restoring force vector. It depends on the attitude of the ROV, as this decides position of attack of the buoyancy and gravity forces. Different force magnitudes and different points of attack will lead to both rotation and translation of the ROV. Mathematical representation is shown in Appendix D, equation (D.7).

Thrust Forces

As the control input gives RPM for each thruster, the thrust forces must be estimated from these values. The resulting thrust force will in real life depend on multiple conditions such as thruster characteristics, thruster - thruster interaction, thruster - ROV body interaction, water velocity and water inflow direction through the thrusters.

Simulating the thrust forces perfectly is a hard task, and simplifications has therefore been made. The thrust force τ_{thr} is normally calculated as (4.5) (Steen, 2011).

$$\tau_{thr} = K_T \rho D^4 |n|n \quad (4.5)$$

The thrust force is said to be a function of the propeller diameter D , propeller revolutions per second n , the water density ρ and the thrust coefficient K_T .

Thruster characteristics can be found from open water tests where a propeller model is tested alone and thrust and torque is measured. The model tests give a relationship between the thrust coefficient K_T and the advance ratio J_a (4.6). V_a is the velocity of the propeller through the water.

$$J_a = \frac{V_a}{nD} \quad (4.6)$$

As the thrusters are placed in different directions on the ROV, the thrust loss needs to be taken into account. This can be done as proposed by Kirkeby (2010), given in (4.7). A thrust loss coefficient θ is included in the thrust calculation. θ depends on where the thruster is placed, and its pose relative to the body coordinate system.

$$F_{thruster} = K_T \rho D^4 |n|n\theta \quad (4.7)$$

When the thrust force is obtained for each thruster, a thrust allocation matrix is used to calculate the resulting forces and moments on the ROV in the body-plane (4.8). f is a vector of the forces from each thruster, and T is the thrust allocation matrix. T is a

constant matrix as long as the thrusters are in constant position. If a rotating thruster is used, the thrust allocation matrix will be a function of the thruster angle.

$$\tau_{thr} = \mathbf{T}f \quad (4.8)$$

External forces

τ_{env} represent environmental loads acting on the ROV, with exception of current loads which are already taken into account in the relative speed vector v_r .

τ_{tether} is the forces from the umbilical. Current will induce drag forces on the umbilical, and the resulting shape and tension in the umbilical will affect the ROV.

τ_{wave} is forces due to the incoming waves.

τ_{thr} is the thrust force vector which gives the total thrust forces from the thrusters in the body plane.

With all these matrices and vectors identified, the equation of motion can be solved for v . A discretization is done to get an algorithm to implement in the software, and Eulers method is used for integration. A discretized version of the equation of motion is shown in (4.9).

$$\dot{v}(t_k) = (\mathbf{M}_{RB} + \mathbf{M}_A)^{-1} [\tau_{thr}(t_k) - \mathbf{C}_{RB}(v(t_k))v(t_k) - \mathbf{C}_A(v_r(t_k))v_r(t_k) - \mathbf{D}(v_r(t_k))v_r(t_k) - G(\eta(t_k))] \quad (4.9)$$

The calculation of position from the equation of motion is done in two different ways. The first method is to use Eulers method directly to estimate the ROV velocity in the body plane from the acceleration obtained from (4.9). The body velocity is then transformed to the world frame, and the resulting position is found with eulers method. This method is shown in (4.10).

The second method is to transform the acceleration $\dot{v}(t_k)$ directly to world frame, and then use Eulers method to get the velocity in world frame. The method is shown in (4.11), where T is the time step, and $J(\Theta)$ is the transformation matrix between the body frame and the world frame.

$$\begin{aligned} \dot{v}_k &= f_1(v_k) & \dot{v}_k &= f_2(v_k) \\ v_{k+1} &= v_k + T\dot{v}_k & & \\ \dot{\eta}_{k+1} &= \mathbf{J}(\Theta)v_{k+1} & \dot{\eta}_{k+1} &= \dot{\eta}_k + TJ(\Theta)\dot{v}_k \end{aligned} \quad (4.10) \quad (4.11)$$

The methods seems very similar, but they do have an important difference. In method 1, the Newtons law is used in the body coordinate system to find the velocity in the body frame. In method 2, Newtons law is used to find the velocity in the world frame. It may seem like a non-relevant detail, but it actually gives noteworthy changes in the results. In method 1,

the coriolis and centripetal forces must be included in $f_1(v)_k$ as the calculation is done in an accelerated coordinate system. In method 2, the coriolis and centripetal forces should not be included.

The transformation matrix $J(\Theta)$ represents the transformation between the body frame and the world frame. The transformation matrix is a function of the ROV angles (roll, pitch and yaw). The transformation can be done by using euler angles or quaternions. If euler angles are used, a singularity occurs when the pitch angle is $\pm 90^\circ$. This is not considered likely to happen, as most ROVs are designed initially stable in roll and pitch. However, the simulator should be able to describe the consequences also when unexpected things happen, e.g. a change in COG. To make sure the simulation does not crash when a pitch angle of 90° is achieved, the transformation is done with unit quaternions. The transformation is shown in (4.12).

$$\dot{\eta}_{quaternions} = J_q(q) v \quad (4.12)$$

The transformation matrix $J_q(q)$ consist of one transformation matrix, $R_b^n(q)$, for the linear velocity, and one transformation matrix, $T_q(q)$, for the angular velocity as shown in (4.13).

$$J_q(q) = \begin{bmatrix} R_b^n(q) & \mathbf{0}_{3 \times 3} \\ \mathbf{0}_{3 \times 3} & T_q(q) \end{bmatrix} \quad (4.13)$$

When $\dot{\eta}$ found from (4.12) is integrated, the position and attitude of the ROV is obtained. The new position is a result of all the forces acting on the ROV. The accuracy of the model will highly depend on the accuracy of the mass matrix, added mass, damping matrix estimation and of course the fact that there are no other significant forces present than the ones included in the model.

4.2.2 Implementation of the Mathematical Model in ROS

The dynamic model of the ROV is implemented in ROS as a node in the package called *dynamics*. The code is written in C++, and the main script is in the file *Vehicle.cpp* that can be found in */uvst/catkin_ws/src/dynamics/src* in appendix B. All the subfunctions developed can be found in the same folder.

The script is implemented as a typical ROS node, where the main function initializes the node and sets up a while loop that runs until the program is terminated or an error has occurred. A flow chart of the software for the Vehicle node can be found in appendix F1.

The purpose of the node is to solve the equation of motion (4.3) from section 4.2.1. First, all the constants needed to solve the equation are read from the configuration file *input-File.txt*, explained in section 2.5. This includes initial position and velocity of the ROV, mass matrix, added mass matrix, linear and quadratic damping matrices, center of gravity, center of bouyancy and volume of the ROV. In addition, specifics about the thrusters are read in the same file, and also the topic name where the control parameters are published. The file can be found in B.

After these variables are defined, a subscriber is initialized. The node will now subscribe to control info published on a certain topic, defined in the configuration file. The node will regularly check if something is published on this control topic, and call the function *updateThrust*. This function registers the control info, and calls a new subfunction *thrustFromRPM* that estimates the thrust forces as explained in section 4.2.1.

The resulting thrust forces in the body coordinate system are then found from the thrust forces and the thrust allocation matrix as explained in section 4.2.1.

The publishers are also initialized before the ROS Node loop starts running. Two topics are published to, one for vehicle velocity and one for vehicle position. Vehicle acceleration can also be relevant to publish, but for now, only position and velocities are included. The positions are published on the topic *Dynamics/ROS/Pose*, where position in the NED frame and pose given in quaternions are included. The velocities are published on the topic *Dynamics/ROS/Twist*, including both linear and angular velocities.

In the ROS Node loop, the nonlinear differential equation is solved in each time step by using Method 1 (4.10).

Before this equation can be solved, there are some matrices and vectors that need to be established. The coriolis and centripetal matrix C_{RB} is a function of v , so it must be recalculated for each time step as v changes. The Coriolis and centripetal matrix of added mass is a function of v_r , and must also be recalculated for every time step. The matrices are established by a function that is called inside the loop, named *updateCoriolis*.

The restoring force vector is dependent on the attitude of the ROV, so this must also be recalculated in every time step. The function *updateRestoring* calculates the restoring force vector by using information about mass, center of gravity, center of buoyancy and volume of the ROV.

The control subscriber initialized before the loop makes sure the thrust force τ_{thr} is updated, resulting in all information needed to solve (4.9) is available. The equation of motion is solved by a function called *updateMotion*. When (4.9) is solved for $\dot{v}(t_k)$, this must be integrated to obtain $v(t_k)$. Eulers method (4.14) is used to estimate the solution, such that the calculation can be done quick. This method is not accurate for large time steps, but when the time steps are small enough, the method gives satisfactory results. The time step T for the calculation will be the same as the time step for the ROS node loop.

$$v(t_{k+1}) = v(t_k) + T\dot{v}(t_k) \quad (4.14)$$

Eulers method is based on the assumption that the slope of the curve, in this case the acceleration, is constant over one time period. The values obtained for $v(t_{k+1})$ are used in equation (4.9) in the next time step.

To find the velocities in the NED frame, the transformation shown in (4.12) is used. In discrete form, this is given as (4.15)

$$\dot{\eta}_{k+1} = J_q(\eta_{k+1})v_{k+1} \quad (4.15)$$

When $\dot{\eta}_{k+1}$ is calculated, this must be integrated to get η_{k+1} . The integration is approximated by using Eulers Method and applying $\dot{\eta}_k$.

The output in the function *updateMotion* is η_q and v . After these values are obtained, the function *updateEta* is called to do the transformation from η_q to η_Θ . This is done because the restoring force is calculated based on the euler angles, and they are also a lot easier to interpret for the user than quaternions. Before the loop starts over, the calculated values of η_q and v are published.

A presentation of how the script defining the Vehicle node is composed is shown in listing 4.2. Before the loop, all information about the ROV and simulation specifics are read from the configuration file *inputFile.txt*. This includes everything from mass matrix and initial position to the topic name where the control info is published to. Inside the loop, the position and velocity of the ROV is calculated based on the previously calculated values and the RPM in each thruster set by the controller node.

Listing 4.2: Layout/Composition of main script for the dynamics node in ROS: Vehicle

```
main:
{ Define node: Vehicles

  Set loop rate

  Define necessary variables and constants (Also including matrices
  and vectors)

  Read configuration info from inputFile.txt and assign values to the
  defined constants

  Set up subscription of control info, estimate thrust forces from
  control info

  Set up publication of ROV Pose

  Set up publication of ROV Velocity

  loop:
  {
    Calculate current velocity relative to body coordinate system

    Calculate relative velocity of ROV

    Calculate Coriolis matrices

    Calculate restoring forces

    Calculate resulting ROV position and velocity

    Update publication pose message to calculated value

    Update publication velocity message to calculated value
  }
}
```

4.2.3 Implementation of mathematical model in MORSE

When implementing the dynamic model in MORSE, a different strategy was required. As MORSE has an included library for physics simulation, this was used in the development. It has been studied how the model can be implemented such that it in the best possible way gets integrated with the already developed software classes and functions in MORSE/Blender.

The methods used in MORSE for calculating motion did not quite fulfill the needs of the simulation platform when it comes to subsea physics in 6 degrees of freedom, so some modifications/supplements had to be made in Blender. This work is done by Eirik Hexenberg Henriksen, PhD Candidate at NTNU.

In MORSE, a builder script written in python will initiate the simulation. This script will make calls to functions that configure components needed for the simulation. The basic components are robots, actuators and sensors.

Robots

In our case, the robot is an ROV. An important property of the robot is the 3D model. When simulating, a 3D model of the robot and its movements will be shown at the screen. For the simulations done as a part of this master thesis, a 3D model of the 30K ROV is used. This is shown in figure 4.2

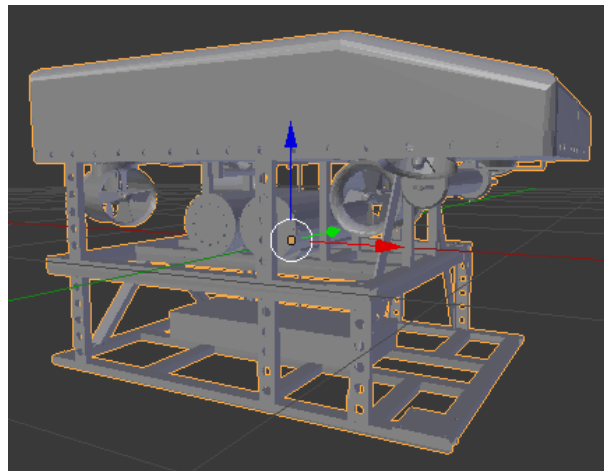


Figure 4.2: 3D model of the 30K ROV, used for visualization in MORSE

The 3D model of the ROV must be drawn a specific way in the coordinate system. MORSE uses the ENU (East-North-Up) coordinate system, while normally underwater vehicles are placed in the NED (North-East-Down) frame. When going from MORSE to ROS and back, transformations are done to get the positions and forces in the correct direction. To avoid confusion, we wish to use the same transformation matrix from ENU to NED as when going from the MORSE body coordinate system to the ROS body coordinate system. It is then important how the 3D model of the ROV is rotated relative to the body coordinate system when implemented in Blender. In ROS, the body coordinate system of the ROV is placed such that x is forward, y is pointing to starboard and z is down (FSD). To get the same transformation

as for the world coordinate systems, it is required that the 3D model in MORSE is rotated such that x is pointing to starboard, y is forward and z is up (SFU).

Actuators

The actuators control the robot in different ways. Already developed actuators are e.g. controlling manipulator joints, applying forces on the robot and controlling the velocity of the robot. Two actuators has been developed as a part of the simulation platform.

The first actuator developed is called *FluidForces*, and the script can be found in `/uvst/morse/subsea/src/subsea/actuators/FluidForces.py`, appendix B. The actuator *FluidForces* will calculate the hydrodynamic forces acting on the ROV based on ROV velocity, pose and specifications read from the configuration file (as explained in section 4.2.1). The calculated force and moments are applied on the ROV by using a built-in function in MORSE. The forces are calculated as they were in ROS, but with some exceptions.

MORSE offers methods to apply forces on the ROV both in the local body frame and in the world frame. This property is used when applying restoring forces. The gravity force always acts downwards, and the buoyancy force always acts upwards. They are therefore applied in the world frame, while the restoring moments are calculated and applied in the body frame.

The coriolis and centripetal forces had to be modified to fit to how MORSE calculates the resulting robot motion. The coriolis and centripetal forces are fictional forces included in the equation of motion (4.3) such that Newtons 2nd law can be applied in a accelerated coordinate system. The way MORSE calculates the robot motion is by applying Newtons 2nd law in the world frame. The coriolis and centripetal forces should therefore not be included. The torques calculated from the coriolis and centripetal matrices must however be included, as this includes physical effects such as the Munk Moment (Faltinsen, p. 188).

The second actuator developed is called *ThrustForces*, and the script can be found in `/uvst/morse/subsea/src/subsea/actuators/ThrustForces.py`, appendix B. This actuator is made to receive desired RPM for the thrusters from the controller module, and to estimate the resulting forces acting on the ROV. The calculations are done the same way as they were done in the ROS dynamics module.

Sensors

The sensors will measure different properties such as acceleration and pose, and multiple sensor models has already been developed in MORSE. The sensors used in the simulation are the already developed velocity and pose sensors. As MORSE uses another reference frame than normally used in underwater robotics, two modifiers has been developed for transforming the values given by the sensors. The modifiers are named *SFUposeToFSD* and *SFUvelocityToFSD* and the code is presented in appendix C. These modifiers must be included in the MORSE source code. They modify the measured pose and velocity such that when this is published, the values are given in the FSD coordinate system.

Builder script

The builder script will add all actuators, sensors and robots to the simulation, and create an environment for them. The builder script developed is called `default.py` and can be found in appendix B. Listing 4.3 shows how the builder script is composed.

Listing 4.3: Layout/Composition of main script for the MORSE simulation

```
Set simulation frequency

Add robot to simulation:
  Set 30K as robot
  Set robot as rigid body
  Set collision bounds such that the ROV stops when it hits the seabed
  (or other robots)

Read and use ROV specifications:
  Read ROV specifications from configuration file
  Set mass of robot
  Position and rotate robot to desired initial pose

Actuators:
  Include FluidForces
  Include ThrustForces
  Add necessary parameters (such as ROV mass, added mass etc) to
  ThrustForces and FluidForces

Sensors:
  Apply pose and velocity sensor
  Modify pose and velocity from SFU to FSD and publish to ROS network

Environment:
  Include subsea environment with mist settings
  Set fixed simulation time steps
  Position camera for visualization
  Set gravity
```

The environment is made with mist to imitate subsea environment. Figure 4.3 shows how the ROV is visualized in the subsea environment.

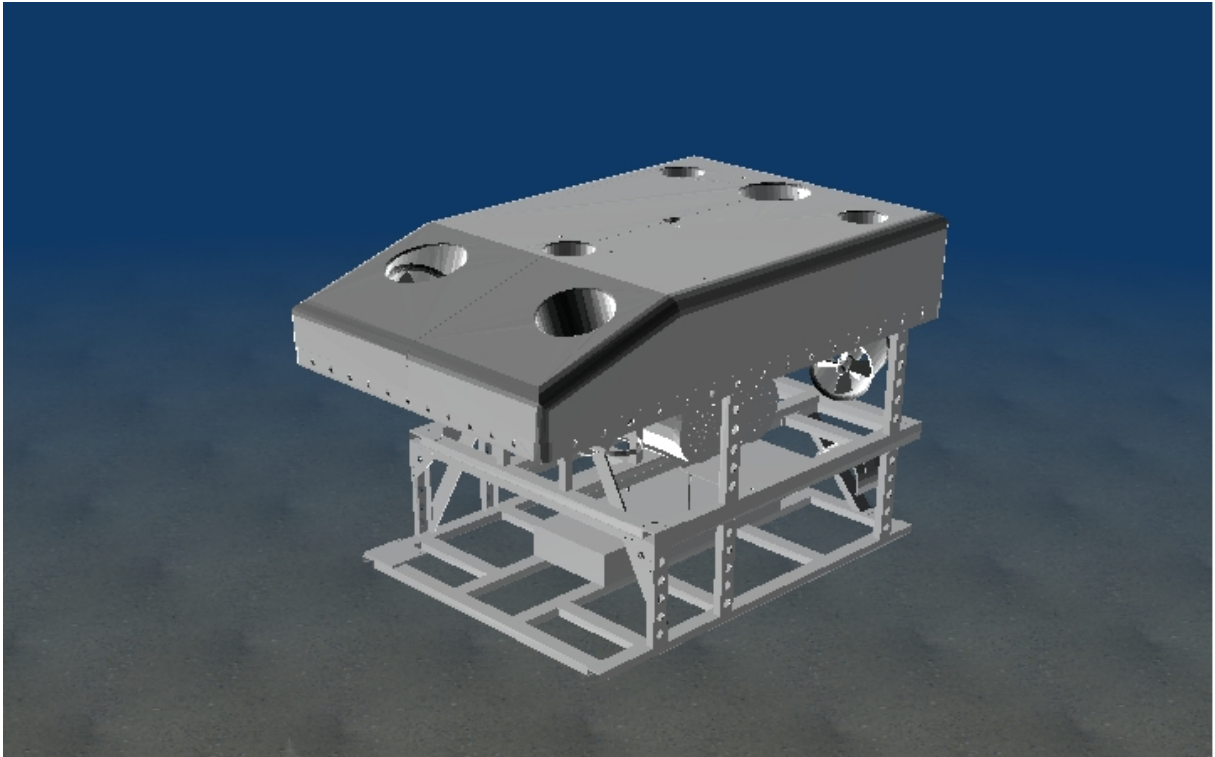


Figure 4.3: MORSE visualization with 30K ROV and mist settings.

4.3 Path Planning

If the ROV is to follow certain paths, a module for path planning and path generation is needed. Path planning is for deciding waypoints the ROV should follow, and path generation is for generating a feasible path between the waypoints (Shin and Singh, 1990). Each waypoint is a set of coordinates (x, y, z) , and can also include a time variable for trajectory tracking if desirable.

There are multiple ways of designing the path planning module. The idea is that the path planning module generates a path for the ROV to follow. This path is interpreted by the guidance module, which decides the ideal heading that will make the ROV follow the path.

4.3.1 Path planning methods

Planning the path can be done based on many different criteria, a few are listed in table 4.1, Fossen (2011).

By following a certain set of criteria, the outline of the path can be decided and described by waypoints. These waypoints are then used to generate a desired path for the ROV to follow.

Path generation based on a few waypoints can be done in many different ways. The simplest way is straight lines between the points. This is however not an easy route for an ROV to follow, as the path will not be smooth in the waypoints.

Table 4.1: Criteria for waypoint generation (Fossen, 2011, p. 255)

Mission:	The craft should move from some starting point (x_0, y_0, z_0) to the terminal point (x_n, y_n, z_n) via the waypoints (x_i, y_i, z_i) .
Environmental data:	Information about wind, waves and ocean currents can be used for energy optimal routing (or avoidance of bad weather for safety reasons).
Geographical data:	Information about shallow waters and islands should be included.
Obstacles:	Floating constructions and other obstacles must be avoided.
Collision avoidance:	Avoiding moving craft close to your own route by introducing safety margins.
Feasibility:	Each waypoint must be feasible, in that it must be possible to maneuver to the next waypoint without exceeding the maximum speed and turning rate.

For trajectory tracking, we need feasible paths which are not only decided by spatial constraints, but also time. This can be used for e.g. following a ship.

To make the path smooth in the waypoints, different types of interpolation methods can be used for path generation. One example is the cubic Hermite interpolant (Fossen, 2011, p. 267) that ensures that the first order derivatives $(\dot{x}_d(t), \dot{y}_d(t))$ are continuous. Another possible interpolation is the cubic spline interpolation that requires continuous second order derivatives $(\ddot{x}_d(t), \ddot{y}_d(t))$ in the waypoints.

A third method is to use straight lines in combination with circle arcs. The circle radius used to find the arcs must exceed the minimum turning radius for the vehicle. A graphical representation of the methods is shown in figure 4.4 (Strømsøyen and Mo, 2014).

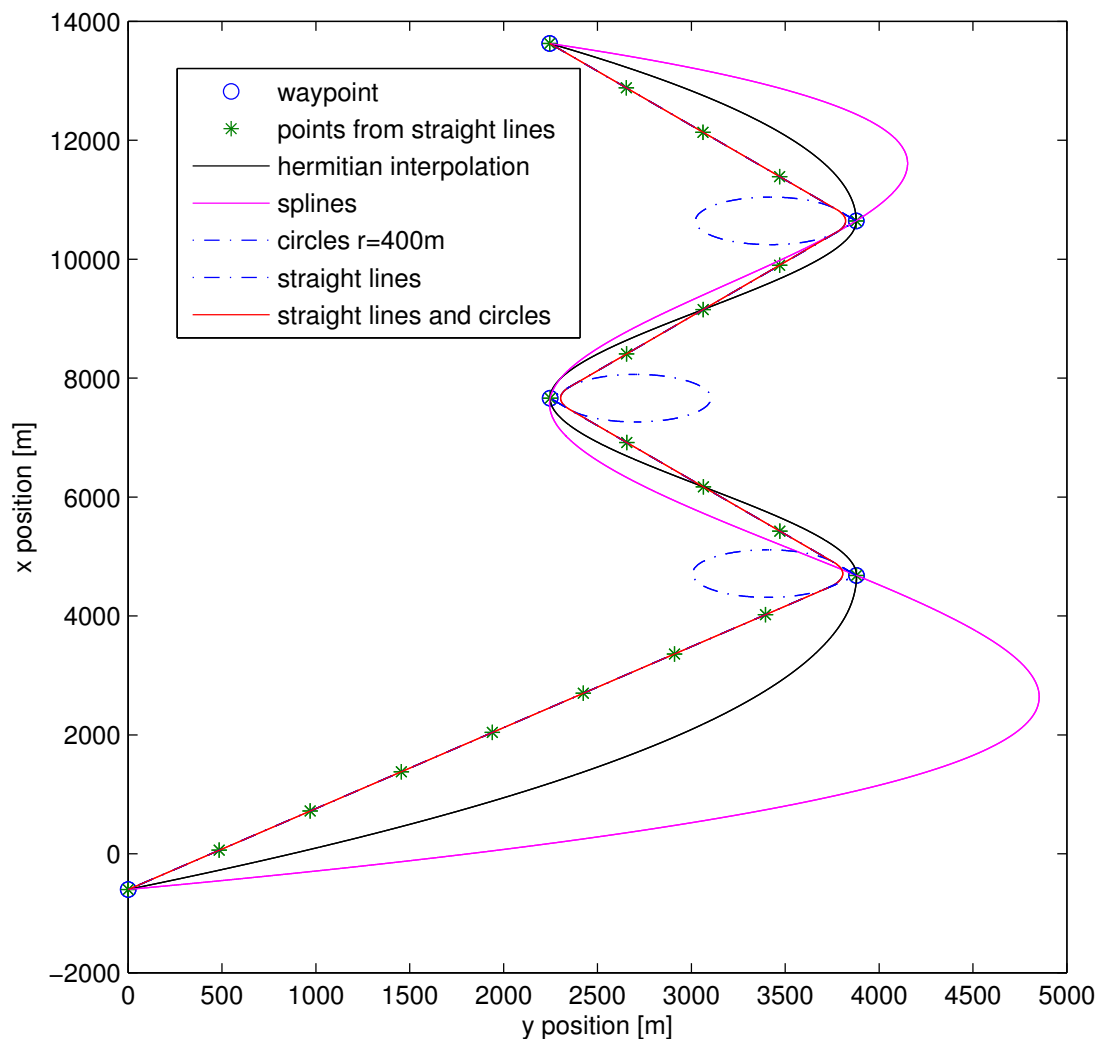


Figure 4.4: Different ways to generate paths from waypoints (Strømsøyen and Mo, 2014)

4.3.2 Implementation of a path planner modules in ROS

The path planner modules developed are simple and cannot manage obstacle avoidance. They are mostly developed to test how the simulation platform manages module switching, and to test the guidance systems and a path controller. For now, the path planner has the possibility of generating four different paths. The paths are presented in table 4.2.

Table 4.2: Paths developed for path planner modules

Sine	This path is shaped as a sine wave in the north-east plane. The starting point is in the ROVs initial position, and wave amplitude and length decided in the script. Depth is constant.
Lawn mower	This path is shaped as a lawn-mower pattern, meaning that it goes back and fort, back and forth, while moving a certain width in between turns. The starting point is in the ROVs initial position, and length and width chosen in the script. Depth is constant.
Spiral	This path is a circle in the north-east plane, but with a constant decrease in depth. The starting point is the initial position of the ROV, and the circle radius and depth change is chosen in the script.
Straight line	This path is a straight line through the initial position of the ROV and a chosen point in space defined in the script.

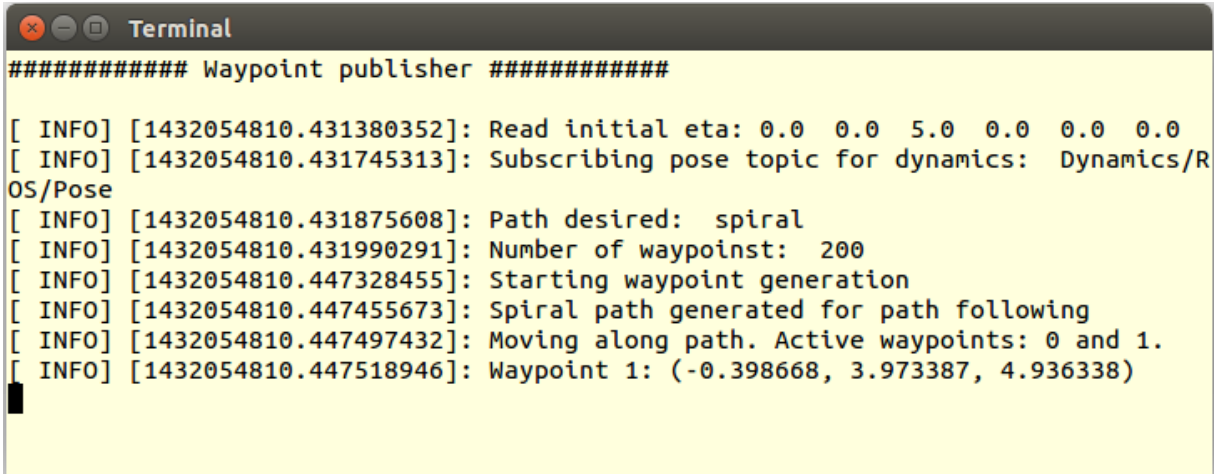
The path generated is given as multiple waypoints, and each waypoint is described by the x, y, z-coordinates of the point. The system the way it is designed now does not support trajectory tracking, which is when the desired path is a function of time as well as spacial constraints. This is rather suggested as a new module that can be implemented in the simulation platform.

Two different methods are tested for delivering the waypoints. The first method is to only publish two waypoints at a time. The guidance module subscribe to the waypoint pair, and chooses heading based on these two. The path planner module using this method is implemented as a ROS node called *WPpublisher*. It will subscribe to the vehicle position, and from this information decide when to switch to the next waypoint. This is typically decided by a circle of acceptance, such that the ROV must be inside a certain range of a new waypoint before the switch can occur. This will be further explained in section 4.4, and the condition stated in (4.22).

The second method tested is a method that does not require the path planner to be implemented as a node. The path planner is implemented as a function called from the guidance node. The function returns the complete set of waypoints, and the switch between waypoints must be decided in the guidance node. This allows for the switch to be made based on along-track distance, which is a parameter already calculated by the LOS guidance. This method for waypoint switching is also explained in section 4.4 and the condition is stated in (4.23).

The path planner implemented as a node (called *WPpublisher*) is found in appendix B. The flowchart is shown in appendix F.2. The second method is implemented in the guidance module, but the software developed for the path planner node is reused.

When the path planner node *WPpublisher* is run, the terminal looks as displayed in figure 4.5



```

##### Waypoint publisher #####

[ INFO] [1432054810.431380352]: Read initial eta: 0.0 0.0 5.0 0.0 0.0 0.0
[ INFO] [1432054810.431745313]: Subscribing pose topic for dynamics: Dynamics/R
OS/Pose
[ INFO] [1432054810.431875608]: Path desired: spiral
[ INFO] [1432054810.431990291]: Number of waypoints: 200
[ INFO] [1432054810.447328455]: Starting waypoint generation
[ INFO] [1432054810.447455673]: Spiral path generated for path following
[ INFO] [1432054810.447497432]: Moving along path. Active waypoints: 0 and 1.
[ INFO] [1432054810.447518946]: Waypoint 1: (-0.398668, 3.973387, 4.936338)

```

Figure 4.5: Terminal when running WPublisher

4.4 Guidance

4.4.1 Guidance methods

Implementation of a guidance module can be done in many different ways, depending on the motion control desired. This will also influence the information needed for input, and will make it tough to generalize the information flow in and out of each guidance module.

Target Tracking

Target tracking is one way to guide the ROV. This will include a moving target, and no information about the future motions of the target is known. This way, only target pose is known, and possibly also target velocity and acceleration. This is typically used for missile control where the missile is supposed to hit and destroy the target. For ROVs however, it is fair to assume that hitting the target is not desired. The same guidance methods can still be applied, and the ROV can e.g. keep a constant distance to the "target". Line of sight guidance, pure pursuit and constant bearing are three methods possible to use when implementing target tracking (Fossen, 2011, p. 243).

Trajectory Tracking

Trajectory tracking is for guiding the vehicle through a time varying trajectory $y_d(t)$. The desired trajectory should give information about both desired position in the NED frame and desired yaw angle, $\eta_d(t) = [N_d(t), E_d(t), D_d(t), \psi_d(t)]^T$. Control of roll and pitch angle is normally not considered, as most ROVs are designed to be open loop stable in both roll and pitch. Achieving trajectory tracking is done by minimizing the tracking error $e(t) = \eta(t) - \eta_d(t)$.

Path Following

Path following is when the vehicle is guided through a path independent of time. The path is completely defined before the mission has started, and the path information is fully available for the guidance system. Line of sight guidance is a common method to use for guiding vehicles through paths. The path is defined by waypoints, and the waypoints are used for controlling the heading. of the vehicle.

4.4.2 Implementation of Guidance Systems in the Simulation Platform

Two different Line of Sight guidance ROS nodes are developed for path following in the simulation platform. They are both able to guide the ROV through the path generated by the path planner, but they use different methods for deciding which waypoints are of interest in the given position.

Line of Sight Guidance

The Line of Sight (LOS) guidance system generates a desired heading angle χ_d for the ROV depending on how far away the ROV is from the desired path.

The desired path is described with waypoints, a series of coordinates describing the path. The k -th waypoint is given as $p_k = (x_k, y_k, z_k)$. If the ROV is outside the path, the guidance system should calculate a heading that will get the ROV back on track. If the ROV is following the path, the guidance system should generate a desired heading to make the ROV keep following the path. The LOS method is only used in two dimensions, the north-east plane. The desired depth is chosen directly as z_{k+1} .

Lookahead-based guidance principles (Fossen, 2011, p. 261) are used in the computation of the desired course angle χ_d , (4.16). Figure 4.6 shows how the different angles needed in the calculations are defined. α_k and χ_r is given in (4.17) and (4.18).

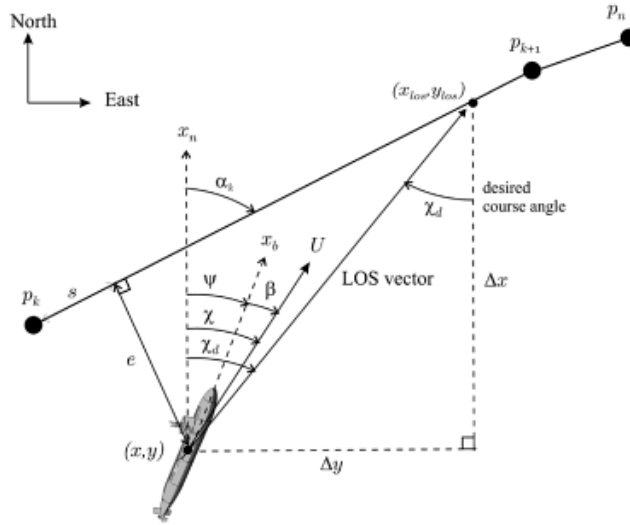


Figure 4.6: Definition of angles and vectors needed in calculations for Line of Sight Guidance (Fossen, 2011, p. 259)

$$\chi_d(e) = \alpha_k + \chi_r(e) \quad (4.16)$$

$$\alpha_k = \text{atan2}(y_{k+1} - y_k, x_{k+1} - x_k) \quad (4.17)$$

$$\chi_r = \arctan\left(-K_p e - K_i \int_0^t e(\tau) d\tau\right) \quad (4.18)$$

e is the cross-track error (4.19), and K_p and K_i are respectively steering law proportional and integrator gains. The current position of the ROV in the north-east plane is given as $x(t)$ and $y(t)$, and k is the current waypoint used to guide the ROV.

$$\begin{bmatrix} s(t) \\ e(t) \end{bmatrix} = \begin{bmatrix} \cos \alpha_k & \sin \alpha_k \\ -\sin \alpha_k & \cos \alpha_k \end{bmatrix} \begin{bmatrix} x(t) - x_k \\ y(t) - y_k \end{bmatrix} \quad (4.19)$$

The desired heading of the ROV is not necessarily the same as the desired yaw angle. If the ROV is exposed to current from the side, the yaw angle must compensate for the current such that the heading is correct. This is done by including the sideslip angle, β , where v is the sway speed and U is the heading speed. This obviously requires the vehicle velocity to be known.

$$\psi_d = \chi_d - \beta \quad (4.20)$$

$$\beta = \arcsin \frac{v}{U} \quad (4.21)$$

When the ROV starts following the path, the guiding is based on the first waypoint, $k=1$. As the ROV goes along the path, the measured position of the ROV is used to decide when to switch to the next waypoint. When the ROV is sufficiently close to the next waypoint, waypoint (x_{k+1}, y_{k+1}) is selected. Deciding if the ROV is "sufficiently close" can be done with two different methods.

The first method introduces a "circle of acceptance" to decide if the ROV is sufficiently close. R_{k+1} is the circle of acceptance radius, normally chosen to equal two vehicle lengths (Fossen, 2011, p. 264). When the ROV comes inside this circle of acceptance, when the condition (4.22) is true, the switch $k=k+1$ can occur.

$$[x_{k+1} - x(t)]^2 + [y_{k+1} - y(t)]^2 \leq R_{k+1} \quad (4.22)$$

The second method is to decide when to switch waypoint based on the along-track distance $s(t)$ and the total along-track distance s_{k+1} between the waypoints p_k^n and p_{k+1}^n . When the condition (4.23) is true, the switch occur such that $k=k+1$.

$$s_{k+1} - s(t) \leq R_{k+1} \quad (4.23)$$

Software design

Two ROS Nodes have been developed for Line of Sight guidance, called LOS node A and B. This was done to test different ways to include the waypoints from the path planner, to test the two different ways to switch from waypoint to waypoint, and also to test how knowledge about the body coordinate velocities can improve the guidance system when current is present.

The LOS guidance method only requires two waypoints at a time to calculate desired heading. The LOS node A is developed such that it subscribes to a pair of waypoints, and does not get all information about the path from the beginning. This can be an advantage when the path is not fully known from the beginning, or it is changed during the run due to obstacles detected. The corresponding path planner (called *WPpublisher*) should take care of the path planning and generation, and publish information about the pair of waypoints required by the guidance node. The waypoint switch task is then left to the path planner to do. For simplicity, the path planner *WPpublisher* uses condition (4.22) to decide when to switch waypoints, as the variables needed to check the condition is already available in the path planner node.

LOS node A does not subscribe to the vehicle velocity, meaning that the sideslip angle β is unknown. The desired heading angle ψ_d is therefore set directly to the desired course angle χ_d .

The LOS node B is developed to include the whole path from the beginning of the simulation. The guidance node itself reads through the path waypoints, and finds the waypoints of interest from condition (4.23). This results in one less node needed in the simulation, as there is no longer need for a path planner node. A path planner function is however still necessary for the guidance node to get the path waypoints.

The listings 4.4 and 4.5 shows the layout of the guidance nodes developed for the simulation platform. They contain mostly the same calculations, but the difference is how the waypoints are included.

Listing 4.4: Layout/Composition of main script for LOS guidance node A

```
main:
{
  Define node: GuidaneLOS_A

  Set loop rate

  Read from configuration file:
  Initial Eta
  Vehicle pose topic

  Set up subscription of
  vehicle pose

  Set up subscription of
  waypoint pairs

  Set up LOS guidance publication

  loop:
  {
    Calculate alpha_k, along
    track and cross track distance

    Error integration

    Calculate chi_r and
    desired heading

    update LOS message

    publish LOS message

  }
}
```

Listing 4.5: Layout/Composition of main script for LOS guidance node B

```
main:
{
  Define node: GuidaneLOS_B

  Set loop rate

  Read from configuration file:
  Initial Eta
  Vehicle pose topic
  Vehicle velocity topic
  Desired path style

  Get all waypoints for desired
  path style

  Set up subscription of
  vehicle pose and velocity

  Set up LOS guidance publication

  loop:
  {
    Calculate alpha_k, along
    track and cross track distance

    Error integration

    Calculate chi_r, Beta and
    desired heading

    update LOS message

    publish LOS message

    Calculate s_{k+1}

    Switch waypoint if
    (s_{k+1}-s(t) <=R)

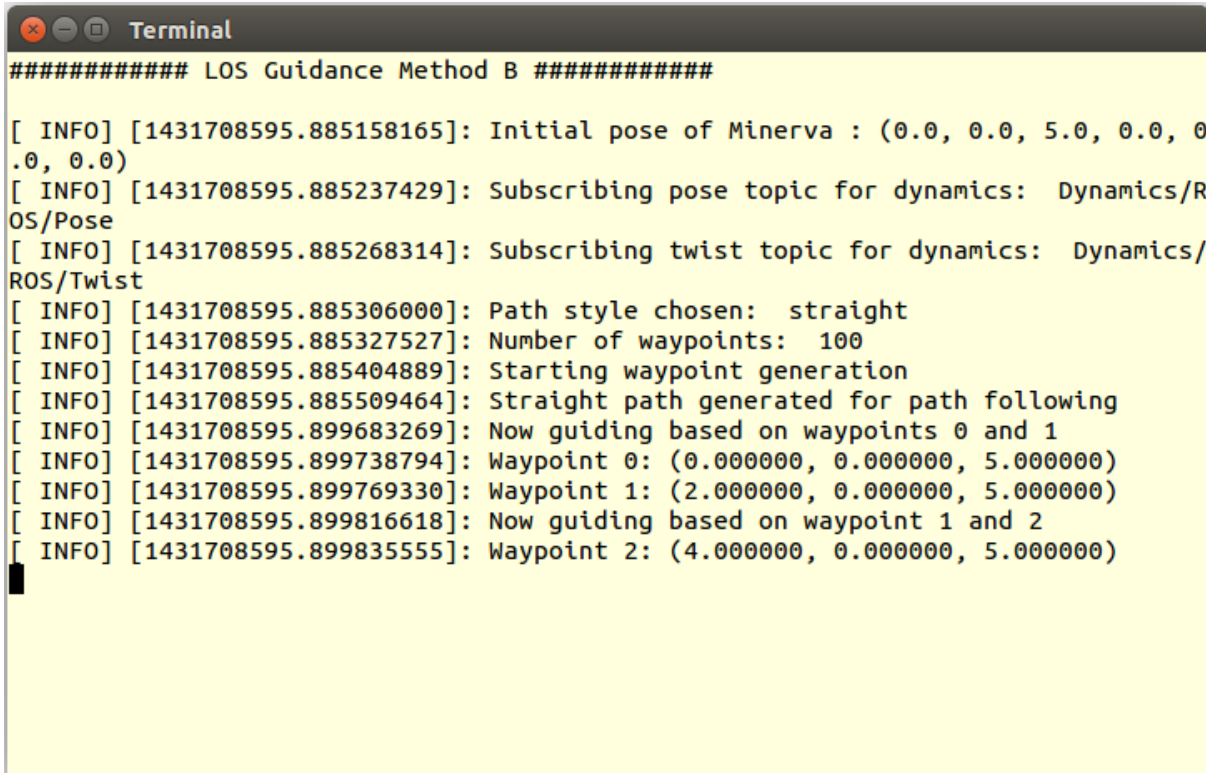
  }
}
```

Flow charts of the guidance nodes can be found in appendix E3 and E4. Note that software has been reused from both the vehicle package and the path planner package, marked in red

and green respectively.

An example of how the terminal looks when running LOS guidance node B is shown in figure 4.7. Information that can be read from this is that the ROV simulated is Minerva, the initial eta was (0, 0, 5, 0, 0, 0), the dynamics in the loop was calculated by the ROS node and not MORSE.

For each new waypoint, the number and the coordinates for the new waypoint is written to the terminal for the user to follow.



```

##### LOS Guidance Method B #####

[ INFO] [1431708595.885158165]: Initial pose of Minerva : (0.0, 0.0, 5.0, 0.0, 0.0, 0.0)
[ INFO] [1431708595.885237429]: Subscribing pose topic for dynamics: Dynamics/ROS/Pose
[ INFO] [1431708595.885268314]: Subscribing twist topic for dynamics: Dynamics/ROS/Twist
[ INFO] [1431708595.885306000]: Path style chosen: straight
[ INFO] [1431708595.885327527]: Number of waypoints: 100
[ INFO] [1431708595.885404889]: Starting waypoint generation
[ INFO] [1431708595.885509464]: Straight path generated for path following
[ INFO] [1431708595.899683269]: Now guiding based on waypoints 0 and 1
[ INFO] [1431708595.899738794]: Waypoint 0: (0.000000, 0.000000, 5.000000)
[ INFO] [1431708595.899769330]: Waypoint 1: (2.000000, 0.000000, 5.000000)
[ INFO] [1431708595.899816618]: Now guiding based on waypoint 1 and 2
[ INFO] [1431708595.899835555]: Waypoint 2: (4.000000, 0.000000, 5.000000)

```

Figure 4.7: Terminal when running LOS Guidance Node B

4.5 Controller

To decide how to set the RPM on each thruster, a controller module is needed. The controller modules designed to control a specific ROV will all generate the same type of output; a set of RPM values that each belong to a specific thruster on the ROV.

4.5.1 Controller algorithms

There are multiple types of controllers possible to design and implement in the simulation platform.

Controller Types

The main principles used in control theory is feedback and feed-forward control (Balchen et al., 2003). Feedback controllers uses measured values of the controlled state, and calculates a control output to regulate the controlled state to a reference value. This requires that the controlled state must get off the reference value before the controller takes action. Feed-forward controllers uses measurements of disturbances to anticipate how the controlled state will act. The information is used to calculate a control output that will act to counteract the disturbance before the results of the disturbance can be seen on the system. This method is also used for feed-forward of the state reference value. Feedback and feed-forward control principles can be combined when developing a controller.

A way to classify different controllers is if they are model based or not. A model based controller will calculate a control output based on a model of the real system. The model should describe the most important properties of the system, but there is no need for the model to be very complicated. A simplified model is easier to handle, and less computations are needed. Examples of model based control are different kinds of backstepping controllers and feedback linearization controllers. The idea of a feedback linearization controller is that the nonlinearities in the system can be canceled out by the control law. This requires a model of the main nonlinearities of the system. The part of the control law that is not model based can then be decided such that the closed loop system is stable (Hedrick and Girard, 2005).

Controllers that are not model based will mainly use controller gain matrices for calculating the control output. This is e.g. done in basic P, PI and PID (proportional-integral-derivative) feedback controllers.

Controller purpose

All controllers are designed to control RPM on each thruster, but they all can have different purposes. A heading controller is designed to control the heading of the ROV, and decides the thruster RPM based on this. Different controller types can be used for this purpose, e.g. a simple proportional controller or a more advanced model based controller.

4.5.2 Implementation of Controllers in the Simulation Platform

Two controller modules are developed for the simulation platform. The first controller is for dynamic positioning, and the second is for path following. They are both implemented as ROS nodes.

Dynamic Positioning

The first controller module is designed to keep the ROV in a constant position. It is a PID controller that controls the position of the ROV in the north-east-down plane as well as the heading angle. A desired position and yaw angle is set, and the controller uses this to decide the control values.

The control law used is shown in (4.24). P_C is a vector of control variables in each degree of freedom. K_p , K_d and K_i are diagonal gain matrices, and $R(\boldsymbol{\psi})$ is the rotation matrix used to get all variables transformed to the body coordinate system. The ROV is assumed to be stable in roll and pitch, so these angles are set to be 0 when calculating the rotation matrix.

$$P_C = -\mathbf{R}^T(\boldsymbol{\psi})\mathbf{K}_p(\boldsymbol{\eta} - \boldsymbol{\eta}_d) - \mathbf{K}_d\mathbf{v} - \mathbf{R}^T(\boldsymbol{\psi})\mathbf{K}_i \int_0^t (\boldsymbol{\eta}(\tau) - \boldsymbol{\eta}_d(\tau)) d\tau \quad (4.24)$$

The control parameters P_C are allocated to each thruster by using the thrust allocation matrix \mathbf{T} , using the same principle as explained in section 4.2.1. The allocated thrust parameters, N_{RPM} , are the values set as RPM for each thruster (4.25).

$$N_{RPM} = \mathbf{T}^\dagger P_C \quad (4.25)$$

The values in N_{RPM} are saturated before they are published by the controller module. The saturation makes sure the control values does not exceed the possible range for each thruster.

A better solution for deciding control RPM values would be to use the control law (4.24) to find desired forces in each degree of freedom. These forces could then be transformed to each thruster by using the thrust allocation matrix, such that a desired thrust force for each thruster is obtained. Thruster characteristics and ROV velocity could then be used to choose a more custom made RPM for each thruster,

Path Controller

The path controller is designed to control the ROV such that it follows the path generated by the path planner.

Three controller functions are developed that the path controller utilizes, heading, depth and speed controllers. One is a heading controller. The heading controller uses desired heading, the current yaw angle and yaw rate to determine a heading control variable, $P_{C,heading}$. The controller type used is a PID controller, and the control law is given in (4.26). The desired heading $\psi_d(t)$ is calculated by the guidance system, and the heading error $\psi - \psi_d$ is given in the range $\pm\pi$.

$$P_{C,heading} = -k_p(\psi(t) - \psi_d(t)) - k_d\dot{\psi}(t) - k_i \int_0^T (\psi(\tau) - \psi_d(\tau)) d\tau \quad (4.26)$$

The second controller function is the depth controller. It uses desired depth and actual depth to calculate a depth control variable with a PI controller. It is a simple task to extend the controller to a PID task, but it was not found necessary in the paths tested.

$$P_{C,depth} = -k_p(z(t) - z_d(t)) - k_i \int_0^T (z(\tau) - z_d(\tau)) d\tau \quad (4.27)$$

The third controller function is a speed controller. It uses desired speed and actual ROV

velocity to calculate a speed control variable with a PI controller as shown in (4.28). The desired speed u_d is not set to a constant value, but is calculated as a function of the heading error. If the yaw angle of the ROV differs from the desired heading angle, the desired speed is linearly reduced to 0 for a heading angle of 90° (4.29). This is included to make the ROV slow down if sharp turns comes up. The maximum desired speed value U is a constant value set by the user.

$$P_{C,speed} = -k_p(u(t) - u_d(t)) - k_i \int_0^T (u(\tau) - u_d(\tau)) d\tau \quad (4.28)$$

$$u_d = \begin{cases} U - \frac{U}{\pi/2} |\psi - \psi_d| & \text{if } |\psi - \psi_d| \leq \frac{\pi}{2} \\ 0 & \text{if } |\psi - \psi_d| \geq \frac{\pi}{2} \end{cases} \quad (4.29)$$

All control parameters are found separately and put together in a vector as shown in (4.30).

$$P_C = [P_{C,speed} \quad 0 \quad P_{C,depth} \quad 0 \quad 0 \quad P_{C,heading}]^T \quad (4.30)$$

N_{RPM} is found with the same method as for the DP controller, (4.25). When it comes to saturating the values, this is done in a different way than done for the DP controller. Instead of saturating the RPM directly, the control parameter values are saturated based on which RPM values that exceed the possible range. This is done to avoid loosing the yaw moment in the saturation process. The algorithm used in this process is created for this particular purpose, and will only work for ROVs with the same thrust configuration as Minerva. The algorithm take advantage of the fact that two of the thrusters only give a force in z-direction and no moment, and it takes into account that the other 3 thrusters both produces a force and a moment. The algorithm can be found in appendix B in the script *pathController.cpp*. The controller gains are found in the controller subfunctions in appendix B.

5. Software verification and testing

Hydrodynamic values for the Minerva ROV is used in the testing of the simulation platform. ROV Minerva is operated by the AUR-Lab, Applied Underwater Robotics Laboratory. This is a center for applied research in underwater robotics established by NTNU. The ROV is usually deployed from R/V Gannerus, a research vessel operated by NTNU.

ROV Minerva is a small ROV used for biological research, sampling, archaeological surveys and much more. It is a SUB-Fighter 7500 designed by Sperre AS in 2003 for NTNU. The vessel data used in this project is given in appendix E. These specifications are retrieved from an input file used in LabVIEW for simulation and control of ROVs (Daniel Fernandes, Personal Communication 16.09.2014). Thruster information about Minerva is obtained from [Kirkeby \(2010\)](#).

5.1 Verification of ROV Model in ROS

The model has been tested and verified against existing dynamic models and to pure logical evaluation of simulation results.

During development, the code was tested many times. This way, the obvious typing mistakes were detected and fixed continuously. This should cover missing brackets, misspelled variable names and similar mistakes. After writing each subfunction, the outputs was checked and evaluated. If some values seemed off, the code was investigated, and checked for errors.

When the ROV dynamic model was done, a series of simulations were ran to check for errors. Different ROV specification values were used in the simulations, and the result compared and evaluated. This includes a change in mass, inertia, damping, added mass, center of gravity, volume and inertial velocities to see if the simulation results are as expected with the given change. Some of the tests done are presented and explained in the following sections.

5.1.1 Simulations Without Current

The first tests were done without thrust forces. This means that the ROV alone has been tested with given initial conditions, and only gravity and buoyancy forces are acting on the body.

Figure 5.1 shows how the position and velocity in z-direction changes the first 20 seconds of a simulation with thrust forces and initial velocities set to zero. Figure 5.2 shows the forces

in z-direction acting on the ROV in the same time period.

From the figures it is clear that the ROV moves in the negative z-direction, meaning upwards, and the velocity stagnates at about -0.019 m/s. As the ROV is constructed such that the buoyancy force is larger than the gravity force, the ROV will rise to the surface, and the simulation results shows exactly this.

The restoring force acts upwards, and is 5 N, which is the same as the difference in the gravity force and buoyancy force. Both linear and quadratic damping increases as the velocity increases and stagnates when the velocity stagnates, but they act against the velocity direction. As the velocity is very small, the linear damping is larger than the quadratic damping. The coriolis forces are zero, just as they should be when the ROV moves straight up. When the total damping force reaches the same magnitude as the restoring force, the velocity reaches a constant value. The results from this simulation seems logical, and there is no reason to suspect mistakes in the software from this test.

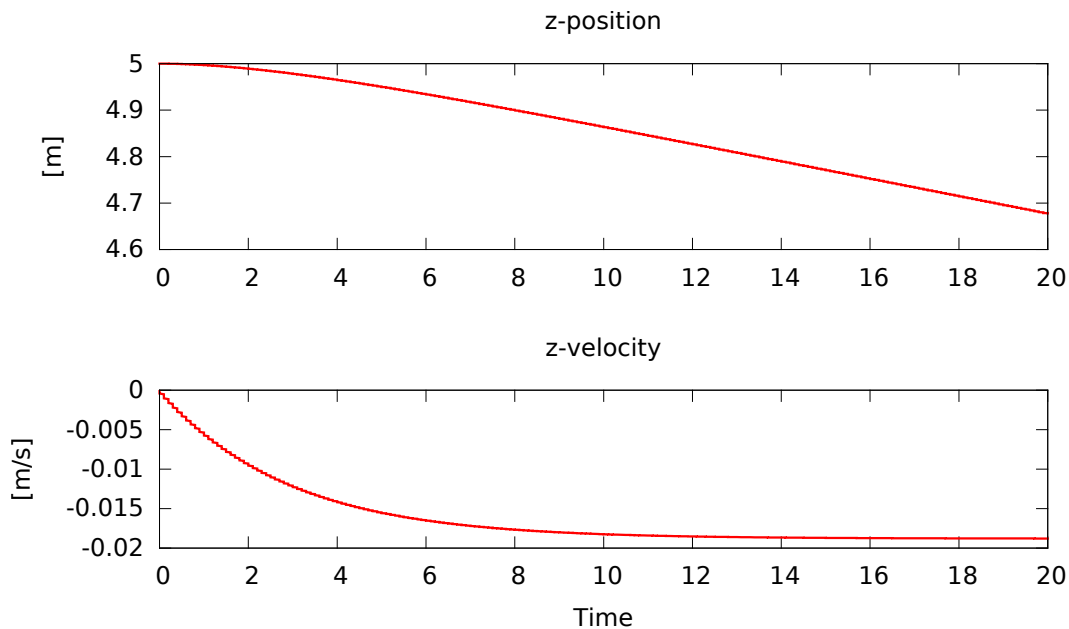


Figure 5.1: ROV position and velocity in z-direction with no thrust forces

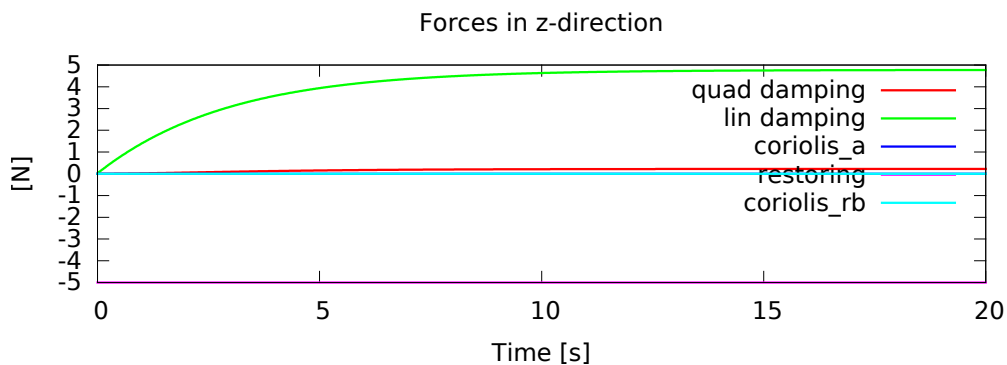


Figure 5.2: Forces when no thrust forces are active

Simulations are also done with different initial velocities. Figure 5.3 shows results from a simulation where the ROV is subjected to no thrust forces, but has an initial velocity in the positive z-direction of 1 m/s. This makes the ROV go downwards first, but eventually stop and go upwards due to the buoyancy force. The simulation results are as expected for all initial velocities tested. Table 5.1 shows how the maximum depth reached by the ROV changes with the initial velocity in z-direction. As the velocity increases, the longer time it takes before the ROV stops and the depth reached increases.

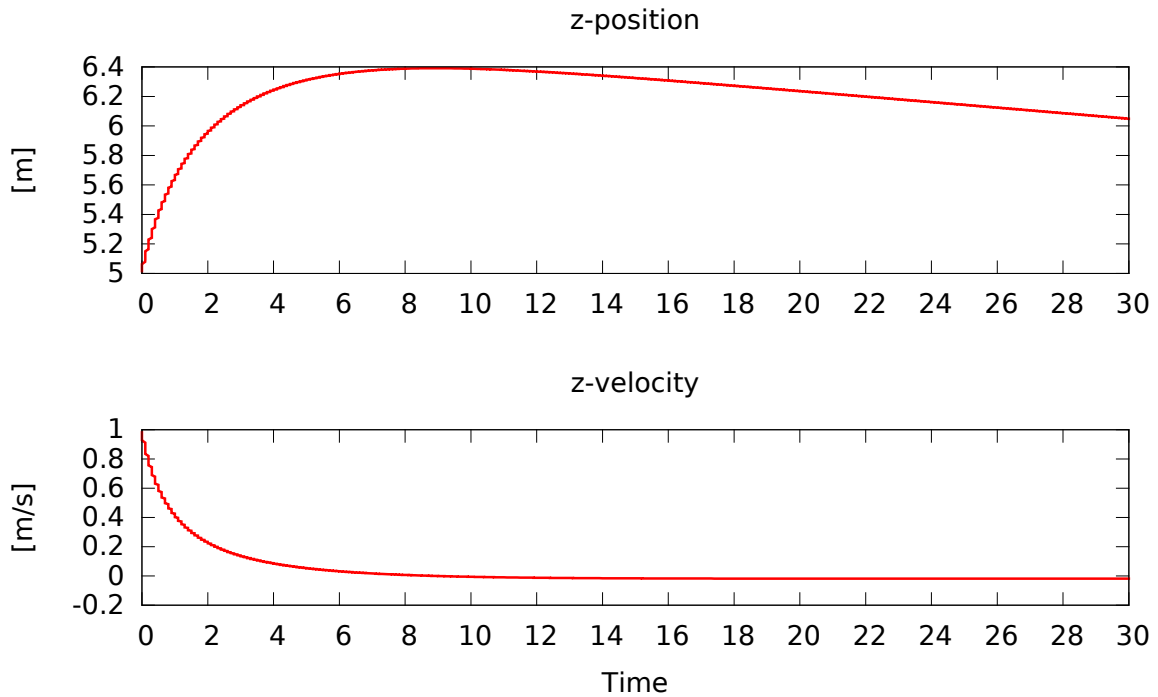


Figure 5.3: ROV position and velocity in z-dir with no thrust forces, initial z-velocity = 1.0 m/s

Table 5.1: Max depth increase as a function of the initial velocity in z-dir.

Initial z-velocity [m/s]	Max depth increase [m]	Time reached max depth [s]
0.2	0.38	7.01
0.4	0.73	8.00
0.6	1.00	8.50
0.8	1.22	8.81
1.0	1.41	9.11

5.1.2 Simulations With Current Present

An error in the software was found when testing the ROV motions with current present. Simulations done with current in north or east direction alone did not reveal any weakness in the software, but the simulation done with current in the north-east direction gave odd results.

Figure 5.4 shows the velocities in the body coordinate system of the ROV when it is heading north and exposed to a 1m/s current from the south. Figure 5.5 shows the same velocities when the current comes from the east.

From the figures it is seen that the ROV reaches the current speed after a while in both cases. It is also clear that the speed is reached faster when the current comes from the side, shown in figure 5.5, than when the current comes from behind, as shown in figure 5.4.

As the damping coefficient for movement in the y-direction is almost twice as large as for movement in the x-direction, these results are as expected. The damping force will be larger when the current comes from the side, and the force will act against the relative velocity, in this case act to push the ROV to follow the current.

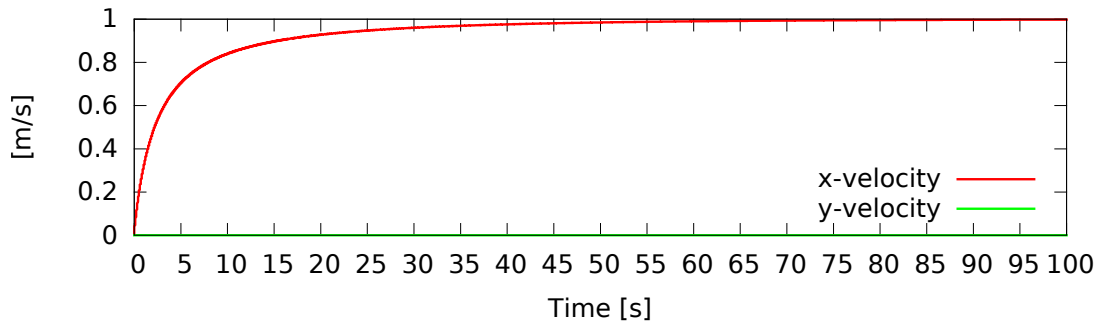


Figure 5.4: ROV velocities when exposed to 1m/s current from south, ROV heading north

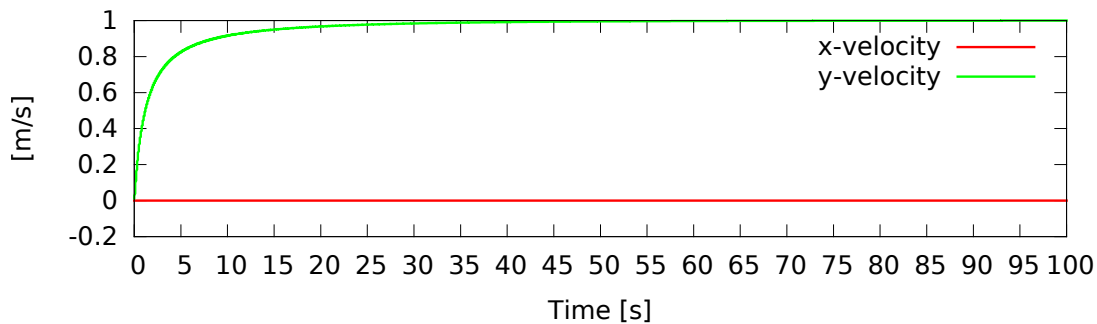


Figure 5.5: ROV velocities when exposed to 1m/s current from west, ROV heading north

The simulation shows acceptable results both when the current comes directly from behind or front, and when the current comes directly from the side. Unacceptable results first appeared when the current comes with an angle such that there are nonzero velocity components both in x and y-direction of the ROV.

Figure 5.6 shows results from the first simulation done with zero initial velocity and no thrust forces acting on the ROV, but with a current in the north-east direction. The current velocity is 1 m/s, and is angled 45 degrees from the north axis, between north and east.

The simulation suggests that the ROV turns about 100 degrees (1.75 rad) before it turns back and stagnates at about -11 degrees (0.2rad). This result was considered very strange, and the same test was done in a dynamic model developed in MATLAB (Mo, 2014) to compare the results. The MATLAB simulation gave the results shown in figure 5.7. From the figure it is

seen that the yaw angle and velocities stagnates at the same values as shown in the simulation from ROS (figure 5.6), but the values before the stagnation differs a lot. The MATLAB simulation suggests smooth and damped movements until stagnation of yaw angle and velocities.

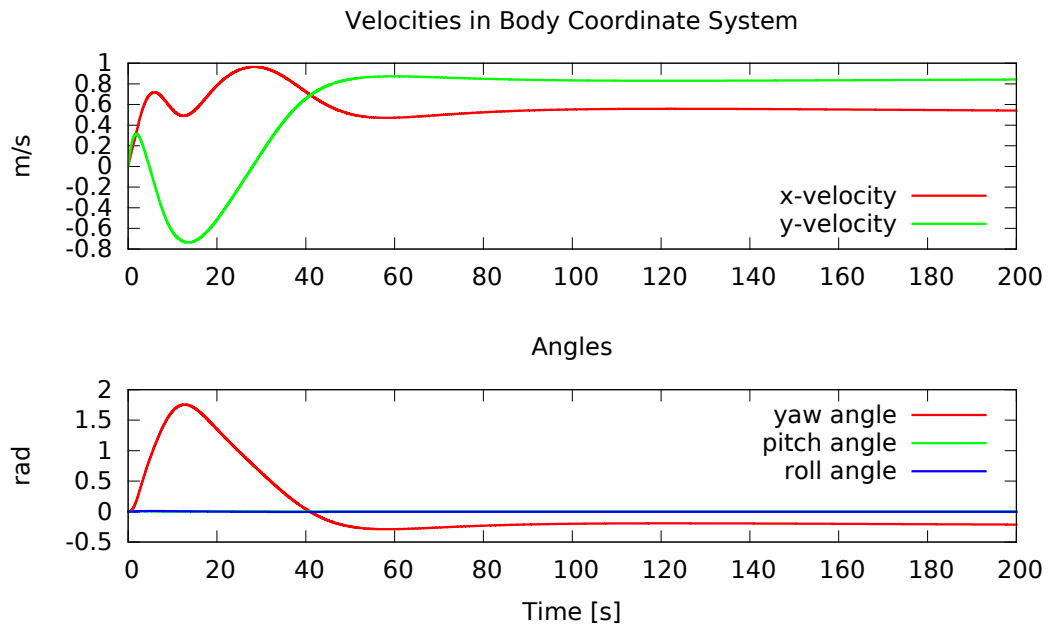


Figure 5.6: Current 1m/s north-east, before bug fix related to Coriolis of added mass matrix

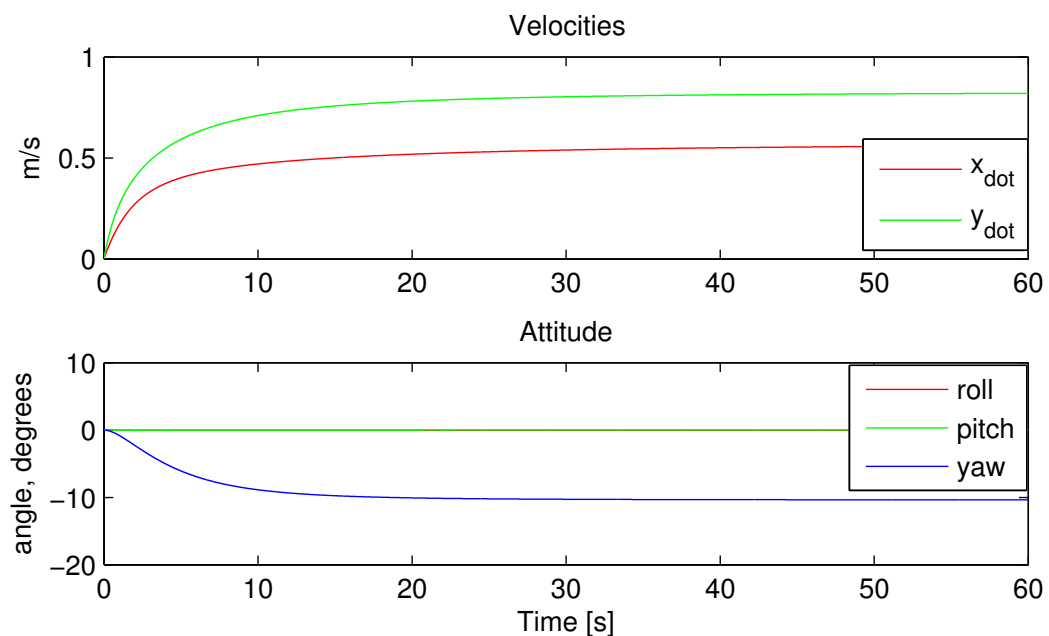


Figure 5.7: Current 1m/s north-east, from dynamic model in MATLAB

The software was methodically investigated to find the bug causing the odd ROV movements when exposed to a current angled relative to the ROVs heading. As there was not detected any mistakes in the software from simulations done without current, the bug was believed

to be related to relative velocity. Information about the current is only used to calculate relative velocity, and this property is used both for calculating damping and coriolis forces. The coriolis forces are however the only coupled forces, meaning they include turning moments caused by motion in the north-east plane. The coriolis of added mass force is the only coriolis force including relative velocity in the calculation, whereas the coriolis force of rigid body only includes ROV velocity in the north-east frame. It was therefore believed that a mistake in the coriolis of added mass force could cause the unexpected motions seen in the simulation. This was investigated, and it was found that the coriolis of added mass matrix was calculated as a function of the ROV velocity in the body plane, and not relative velocity in the body plane as it should be. The current was not taken into account in this part of the equation, and this resulted in wrong simulation results.

After the bug was fixed, the simulation gave more realistic results in this current condition. Figure 5.8 shows the simulated velocities and angles after the bug was fixed, but with the same conditions as before. The motions are now a lot smoother and seems to be correct. They are also similar to the results from the MATLAB simulation, which suggests that the simulation now gives good estimates.

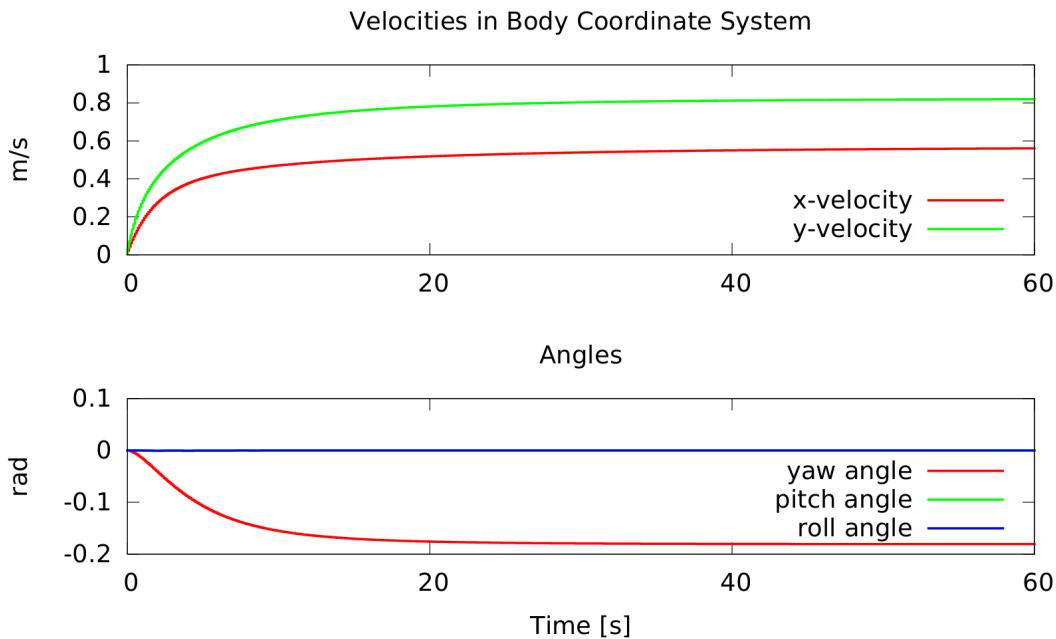


Figure 5.8: Current 1m/s north-east, after bug fix related to Coriolis of added mass matrix

5.1.3 Simulations with changes in mass and inertia

Simulations are done to test how changes in mass affects the results. Figure 5.9 shows how the ROV velocity in z-direction changes with different mass matrices. The simulations are done with no thrust input, and the ROV volume is kept constant. The dark blue line shows the z-velocity when the simulation is done with the normal mass matrix for Minerva. The green and red lines shows the z-velocity when the mass matrix is reduced, 1% and 2% respectively. This means a smaller mass, and the velocity is increased in the negative z-direction,

meaning upwards to the surface. Smaller mass will give a larger difference in gravity and buoyancy force, and the ROV will rise faster to the surface. The pink and cyan lines shows the velocities when the mass matrix is increased by 1% and 2% respectively. The gravity force is now larger than the buoyancy force, and the ROV will sink to the seabed.

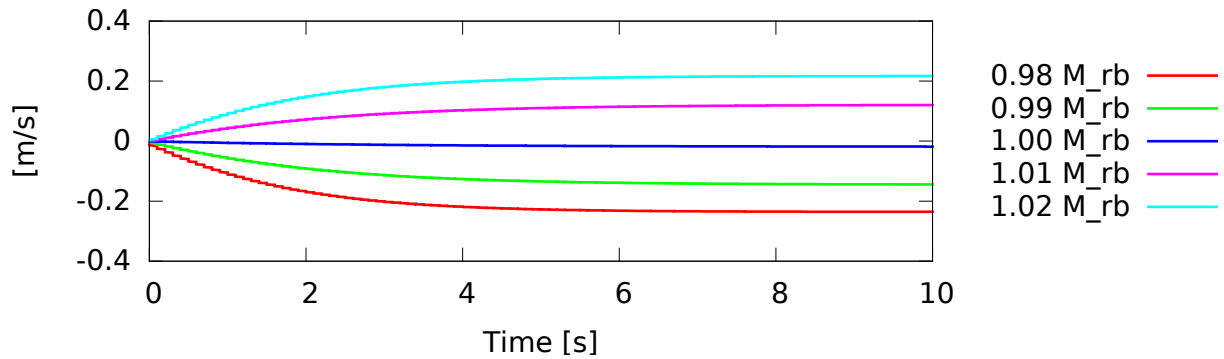


Figure 5.9: Velocities in z-direction with changes in mass matrix

The simulations with change in mass are acceptable, and does simulate the real process with satisfactory results. Tests are also done to see how a change in added mass affects the velocity. As there is no change in rigid body mass, the gravity force stays the same and the maximum velocity reached should also be the same in all cases. The inertia of the system is changed, and will affect how fast the maximum velocity is reached. Table 5.2 shows how the time before the maximum velocity is reached changes with added mass. Maximum velocity is in this case defined with 3 decimals, $z_{max} = -0.019$. $M_{A,0}$ is the added mass matrix for the Minerva ROV.

Table 5.2: Time before maximum velocity is reached when added mass matrix is changed

Added Mass Matrix	Time [s] until maximum velocity is reached
0.50 $M_{A,0}$	9.50
0.75 $M_{A,0}$	10.80
1.00 $M_{A,0}$	11.90
1.25 $M_{A,0}$	13.50
1.50 $M_{A,0}$	14.51

It is seen from the results that when the added mass matrix increases, it takes longer time before the maximum velocity is reached. This is due to the fact that the inertia increases, and it will slow down the system.

5.1.4 Simulations with thrust forces

Simulations are done to test if the ROV responds correctly to thrust forces applied in the body plane. Different combinations are tested, and the simulations gave acceptable results. When exposed to a yaw moment, the ROV yawed, and when exposed to a thrust force in x, y and z direction, the ROV moved accordingly. With moments applied in roll and pitch, the ROV rolled and pitched to the point where the restoring forces counteracted the thrust moment.

Figure 5.10 shows the ROV velocity in x-direction when exposed to a thrust force in x-direction. Initial velocity was zero in all simulations. The maximum velocities are in table 5.3. We see that when the force increases, so does the velocity. As the thrust force is increased further, the velocity increase diminishes. This is as expected as the quadratic damping will increase more than the thrust force increases. The results are satisfying, and indicates that the dynamic model gives good estimations.

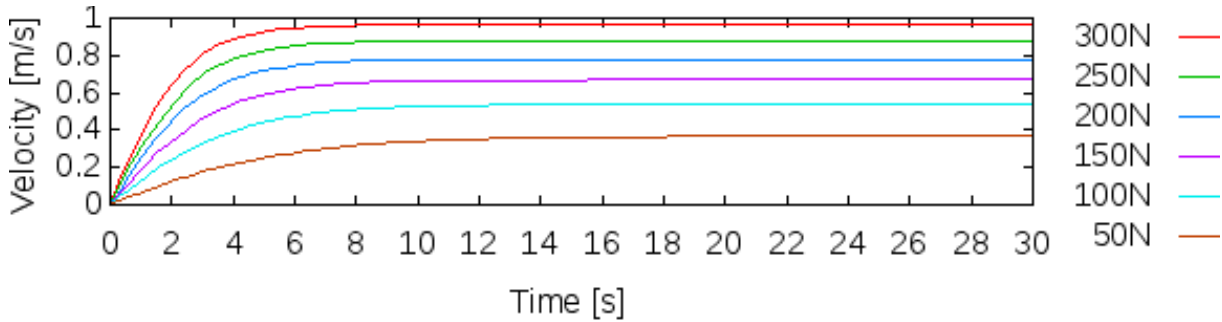


Figure 5.10: ROV velocity in x direction with applied thrust force in x direction

Table 5.3: Maximum ROV velocity when exposed to thrust force in x-direction

Force in x-dir [N]	Maximum Velocity [m/s]	Percent increase in maximum velocity
50	0.37	-
100	0.54	45.9
150	0.67	24.1
200	0.78	16.4
250	0.88	12.8
300	0.97	10.2

Figure 5.11 shows the ROV trajectory for simulations done with a 200N thrust force in x-direction, and with a yaw moment of 3Nm. The initial position is (1, 0) in the north-east plane, and the ROV is initially heading north. It is seen that the result from the ROS simulation matches the MATLAB simulation perfectly, and this gives a reason to believe that the dynamic model is implemented correctly.

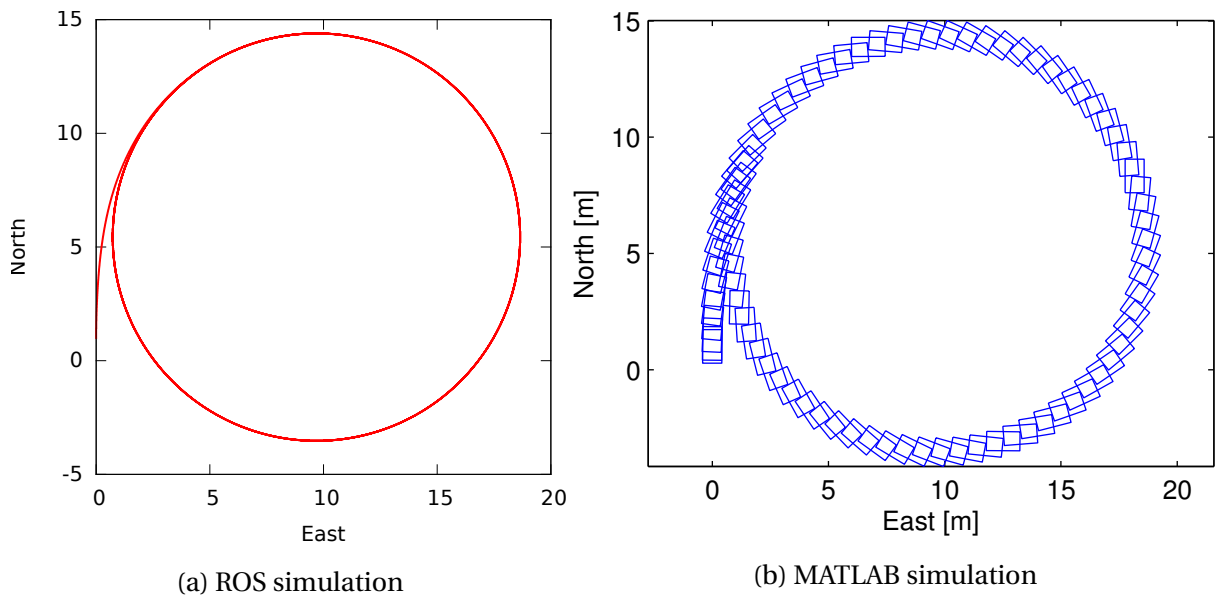


Figure 5.11: Comparing simulations from ROS with MATLAB. Forces: 200N in x-direction and 3Nm in yaw

5.1.5 Simulations with RPM control

The dynamic model includes thrust force calculation from RPM of each thruster. Tests are done to make sure the thrust calculation and allocation gives satisfactory results.

Figure 5.12 shows the thrusters of the Minerva ROV as seen from below. The simulations have been done by setting the RPM of each thruster, and evaluating the ROV response. The thrusters are numbered for easier understanding of the RPM control values in the following simulations.

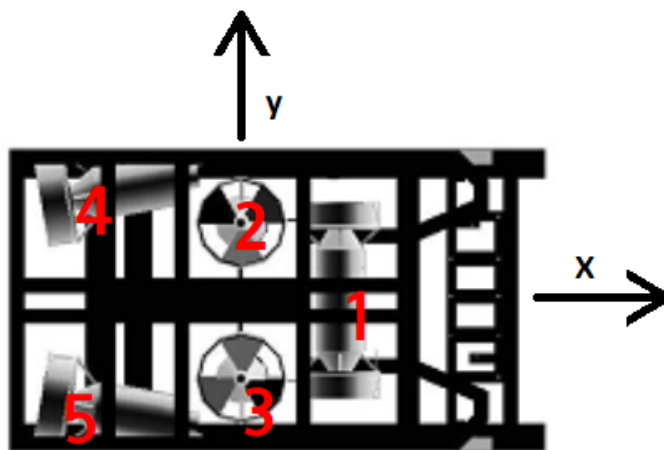


Figure 5.12: ROV Minerva and thruster numbering

As the thrust force from the thrusters not only depend on the RPM, but also the velocity of the ROV, the thrust force will vary before steady conditions are achieved. Figure 5.13 shows how the velocities and thrust forces and moment changes in a simulation done with RPM set on 700 and 1000 for thruster 4 and 5 respectively. The other thrusters are set on 0 RPM. When thruster 5 has a larger RPM than thruster 4, it is clear from figure 5.12 that this will create both a thrust force in x-direction, a smaller thrust force in y-direction, and a yaw moment as both thrusters are rotated relative to the center line. The simulation done gives results as expected. As the velocity increase, the thrust forces decrease a bit, until about 5 seconds into the simulation, where the thrust forces and moment stagnates. A small change in x and y-velocity is seen even after this stagnation, but the total velocity ($v = \sqrt{\dot{x}^2 + \dot{y}^2}$) does not change much.

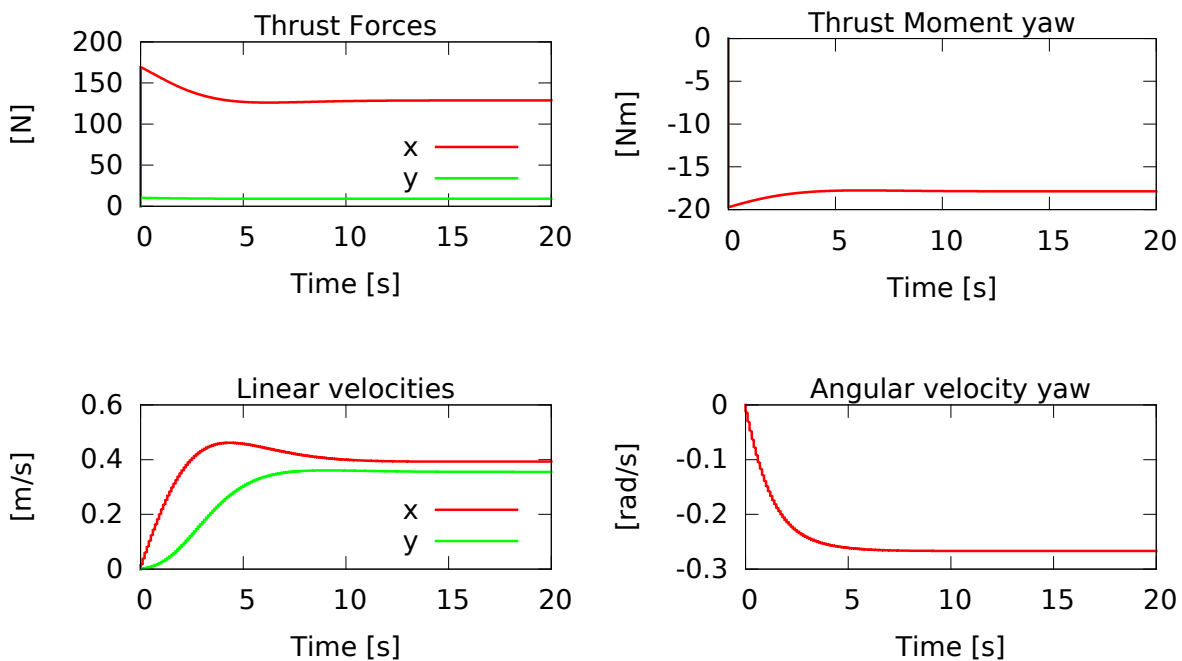


Figure 5.13: Results from simulations with RPM set on 700 and 1000 from thruster 4 and 5 respectively.

The decrease in thrust force is due to the fact that the velocity increases. As the velocity increases, the advance ratio will increase, and the thrust coefficient decrease.

Figure 5.14 shows result from simulations done where thruster 4 is set on 1000RPM and thruster 5 is varied from simulation to simulation. The forces follows the same pattern as shown in figure 5.13.

When the RPM in thruster 4 is the same as in thruster 5, there will be no yaw moment, and no force in y-direction. The ROV will then just go forward, as shown in figure 5.14a. When decreasing RPM on thruster 5, the thrust force in x-direction will be reduces. The thrust forces from each thruster in y-direction will no longer counteract each other fully, and this results in increased force in negative y-direction as well as increased yaw moment, as seen in 5.14b. The ROV will turn, and go in circles.

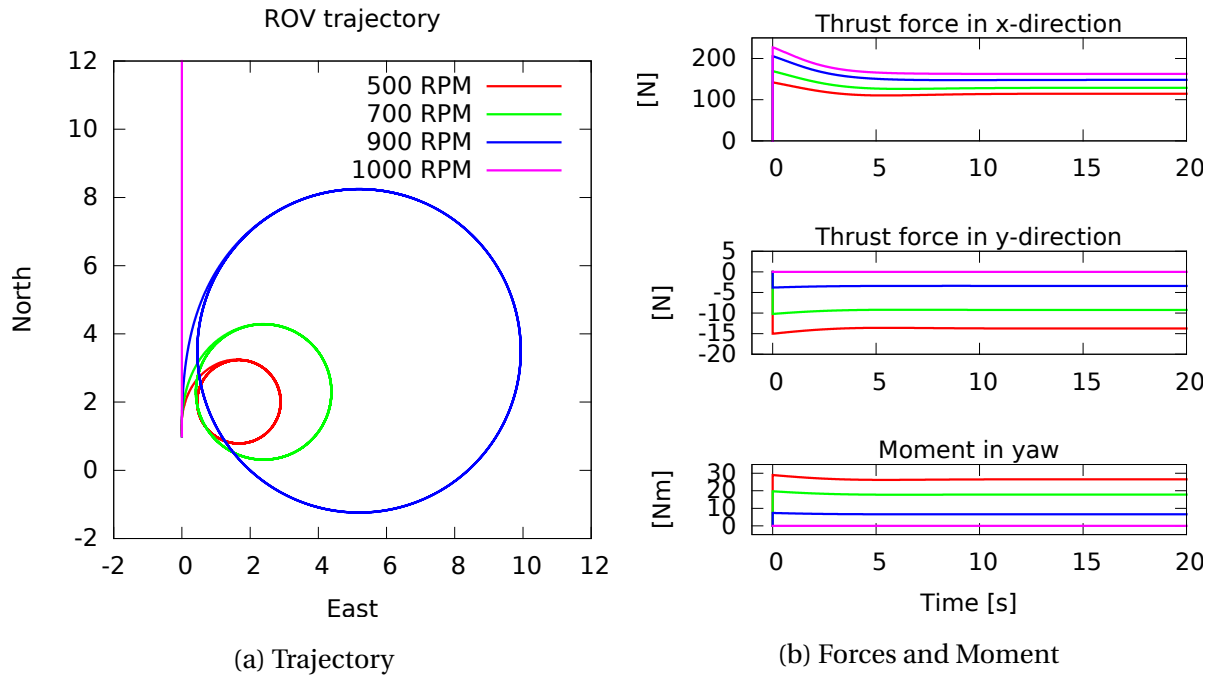


Figure 5.14: Simulations done with changing RPM on thruster 5, keeping thruster 4 on 1000RPM and the other thrusters on 0.

Table 5.4 shows how increasing RPM on thruster 2 affects the roll angle. The roll angles achieved are very small due to the large restoring forces acting once the roll angle is nonzero. The values are measured after the roll motion is stabilized, and all the other thrusters are set on 0 RPM. The exact same results are achieved when testing on thruster 3 instead of 2, but then the ROV will roll the other way.

Table 5.4: Roll angles for different RPM on thruster 2, with all other thrusters kept on 0 RPM.

RPM thruster 2	Roll angle [deg] after stabilization
200	0.035
400	0.140
600	0.314
800	0.559
1000	0.873
1200	1.257

Figure 5.15 shows results from simulations done with changing RPM on thruster 1 and keeping the other thrusters at 0 RPM. The ROV is initially heading north in all simulations, and it goes sideways in circles when thruster 1 is active. Larger RPM will increase the yaw moment, and the circles gets smaller.

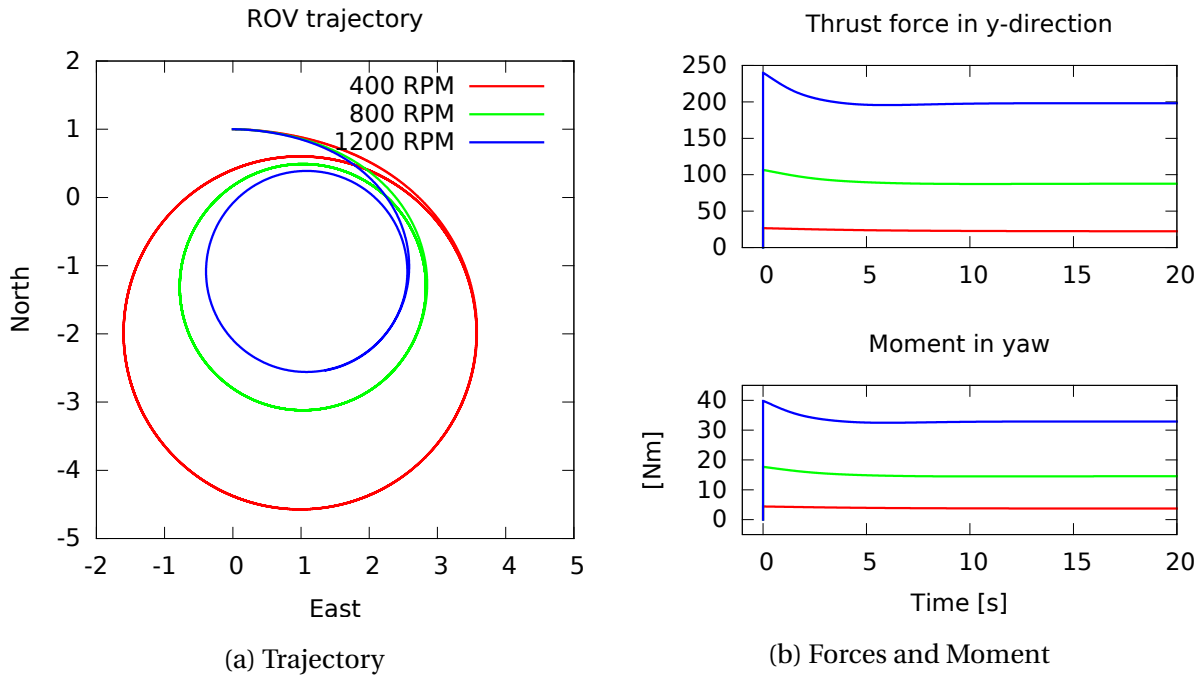


Figure 5.15: Simulations done with changing RPM on thruster 1, keeping the other thrusters at 0 RPM.

5.2 Verification of Dynamic ROV Model in MORSE

The dynamic model implemented in MORSE is tested and verified in the same way as done for the dynamic module in ROS. The process has however been somewhat different than it was for the ROS module. As the module makes use of already developed software, the troubleshooting had to also include these parts. This is not necessarily because they are likely to have bugs, but because the application could be misunderstood and wrongly integrated in our software.

As the implementation of the dynamic module in MORSE was a collaboration with Eirik Hexenberg Henriksen, he has also played an important role in the troubleshooting process.

Multiple tests and simulations have been done in this process, and a lot of methods have been changed and improved on the way. A large part of the completed simulations shows promising results and implies that the hydrodynamic forces are calculated correctly. In this chapter, it is focused on presenting the results that seems a bit odd, and proves that the implementation of the dynamic model is still not perfect.

Induced Initial Velocity

One of the problems encountered in the MORSE dynamics was that the simulated initial ROV velocity was wrong. The initial velocity should be zero when nothing else is specified, but this was not the case. The ROV had a relatively large initial velocity in z-direction, but also smaller initial velocities in the other five degrees of freedom. The problem was investigated,

and two different causes was found.

One of the problems related to the initial velocity in z-direction was how the buoyancy force was implemented. Initially, this was introduced in the ThrustForces actuator, while the gravity force was calculated by using an already developed function in MORSE initiated in the builder script. The result was that the gravity force acted on the ROV a little while before the buoyancy force did, and this gave a non-realistic initial velocity of the ROV. The problem was solved by implementing a new function in Blender that calculates the buoyancy force and applies it together with the gravity force.

The second problem with the small initial velocities in all degrees of freedom was found to be due to the sensors applied on the ROV. A velocity sensor and a pose sensor is added to the robot in the builder script, such that the ROVs pose and velocity can be published on the ROS network. When applying a sensor to the ROV, the sensor is represented by some kind of 3D block in Blender. The block is applied with a mesh in the source code for the sensors in MORSE. The mesh induces an initial velocity in the simulations, and the problem was solved by altering the source code for the sensors and remove the mesh. This is not a permanent solution, as the meshes obviously are implemented for a reason. It is not discovered why they induce an initial velocity, and it is assumed to be a bug that should be fixed. For now, the solution was to remove the mesh, and the simulation starts without initial velocity of the ROV.

Problem with Inertia

A problem with the inertia was detected when running simulations with constant RPM of each thruster. The results from the dynamics module in MORSE differed a bit from the ones obtained in ROS, and the difference seems to be due to difference in inertia.

When tests are done where there are no thrust forces acting on the ROV, it starts to rise to the surface as expected. It reaches the same maximum velocity as seen in the ROS dynamics, but the acceleration is higher. A graphical presentation of the results is shown in figure 5.16.

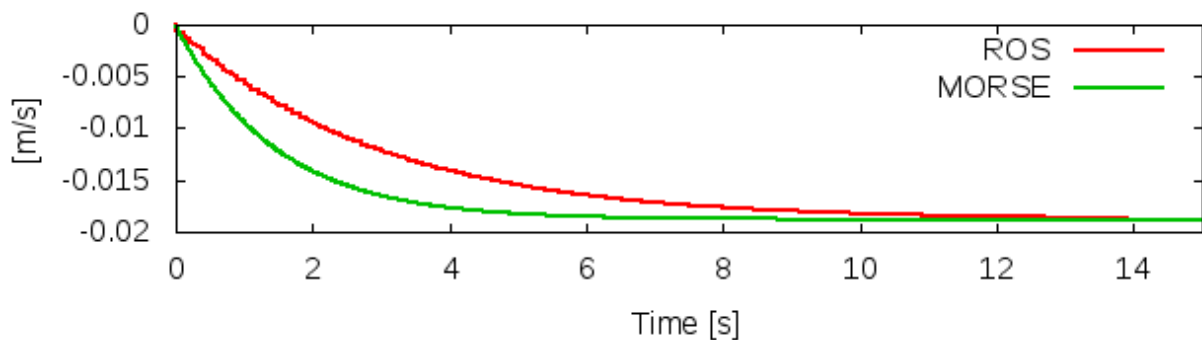


Figure 5.16: Comparing velocities in z-direction for MORSE and ROS with no thrust forces acting on the ROV.

The tendency in the results are very similar to what was found when studying how the ROS dynamics module responded to changes in added mass in section 5.1.3. It was seen that increasing added mass made the system slower, which is as expected as the total inertia

$(M_A + M_{RB})$ increases. The simulation results from MORSE seems to represent an ROV with a much smaller inertia than the one in the ROS simulation, as the system is much faster.

Investigations have been done to find the bug, but they were non-conclusive. It is however believed that the bug is related to how the inertia is included in the calculations in Blender.

Simulation Time VS Real Time

The biggest problem in with the MORSE dynamics is that the calculations slow down a while into each simulation. MORSE does not manage to keep the simulation in real time, and this has undesirable side effects when the simulation is done with a controller in the loop that continues to run in real time.

The results from MORSE did not reveal that the simulation time differed from the real time in the beginning. When plotting the results from the simulation, the time stamp in the published position message was used. This time stamp was believed to give the simulation time, but it actually gave real time. This means that if the simulator used five seconds for simulating one second of ROV motion, the result would be plotted in a five seconds time range, and not one second as it should be. This was first detected when investigating the calculated velocities and positions given in what was believed to be simulation time. The ROV velocity measurement still had high values, while the position hardly changed at all. This implied that there was either something wrong with the velocity sensor, or that something was off with the timescale.

The simulations were originally done with varying simulation steps. When testing the simulation with fixed simulation steps, the plots revealed that the simulation time actually differed from real time. Figure 5.17 shows results from two simulations, one done with fixed simulation steps, and one without. The simulations are done by setting thruster RPM such that the ROV moves in circles. The reason for why the circle motion is not completely equal in ROS and MORSE is thought to be due to the inertia problem.

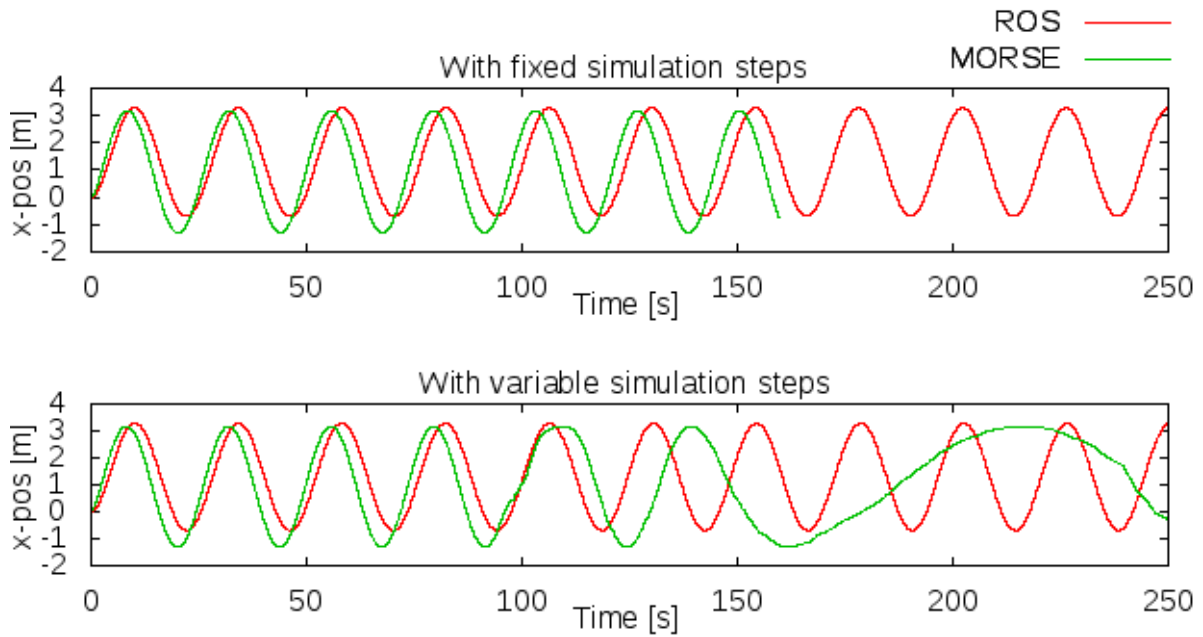


Figure 5.17: Comparing results with and without variable simulation steps in MORSE

The simulation time problem has been investigated, and it is concluded that the bug must be related to the way controller info is subscribed to. It seems to be a problem for MORSE to handle all messages published by the controller in ROS.

The messages are received by subscribing to a topic in the same way it is done in ROS. When subscribing to messages, a queue size is defined. This states how long the "queue" of messages can be before it starts to throw away the old messages. The queue size is set to 1, as it is desired to always receive the newest control outputs. Testing different queue sizes did not change the results at all.

What did change the results remarkably was the publication frequency of the controller output. When publishing controller output often, it took shorter time before the MORSE calculations slowed down. When reducing the publication frequency, the simulation time kept up with real time for a much longer period. The results from some of the tests done are presented in table 5.5. The "simulation length in real time" represents the time period the dynamic module in MORSE managed to keep up with the dynamic module in ROS. As seen in figure 5.17, this was about 100 seconds with a publication frequency of 10 Hz. It is seen that ROS and MORSE gives similar results up to about 100 seconds, where MORSE starts to slow down (as seen in the bottom plot). It is in these tests assumed that ROS manages to simulate everything in real time, which is proved to be a safe assumption after a set of simulation time tests were done.

Table 5.5: How much MORSE manages to simulate in real time (approx.) with a given publication rate of controller output.

Publication Frequency [Hz]	Simulation length in real time [s]
1	250
10	100
100	10
1000	0

It is believed that the reason for this problem is that the subscription for control info is re-done in every time step. This is a very time consuming task, and a possible solution is presented in section 8.6.1.

5.3 Control, guidance and path planner testing

Two controllers and two guidance systems are implemented in the simulation platform. Four different paths are implemented in a path planner for the guidance systems.

The modules are implemented to make an example of how controllers and guidance systems can be implemented as ROS nodes, and also to test how the simulation platform allows for switching between modules in the loop. The focus has not been on tuning the controllers and guidance systems, but on the implementation of the subscribers and publishers, and how the modules communicate with each other.

5.3.1 Module switch method

Defining which nodes to be in the simulation loop is defined in the configuration file. Each node publish messages on a topic, and a subscriber node can subscribe to messages on this topic. As multiple nodes of the same type can be active at the same time, they publish to different topics such that the information is separated. The subscriber node chooses which topic to subscribe to, and this is what is defined in the configuration file. For now, the dynamics can be calculated in two different modules, the ROS dynamics module and the MORSE dynamics module. They publish information about the ROV pose (position in north-east-down frame and attitude in quaternions) and the ROV velocities (linear velocity in north-east-down frame and angular velocity in roll, pitch and yaw). The topic names are as following:

<i>Dynamics/ROS/Pose</i>	<i>Dynamics/MORSE/Pose</i>
<i>Dynamics/ROS/Twist</i>	<i>Dynamics/MORSE/Twist</i>

If we want the ROS dynamics module to be in the loop, we write *Dynamics/ROS/Pose* and *Dynamics/ROS/Twist* in the configuration file to define that all nodes who needs information of the ROV pose and velocities will subscribe to these topics. The same strategy is used for

choosing which controller to be in the loop (*PMcontrol/Path* or *RPMcontrol/DP*), and which guidance system to be in the loop (*Guidance/LOS_A* or *Guidance/LOS_B*).

Tests have been run to verify that the correct topics are read by the different nodes, and it is found to work as planned. Three scenarios will be presented in this section, with the goal of giving a better understanding of how the method works.

Scenario 1

Listing 5.1 shows the last part of the configuration file for a scenario where the ROS dynamics, path controller and the guidance system LOS B is in the simulation loop. It is also written that the desired path style for the path planner to generate is a sine wave, and 200 waypoints should be generated.

Listing 5.1: Last part of configuration file, scenario 1

```
#Topics in loop for dynamics pose and twist :)
Dynamics/ROS/Pose
Dynamics/ROS/Twist
#Topic for controller (RPMcontrol/DP, RPMcontrol/Path)
RPMcontrol/Path
#Topic guidance (LOS_A, LOS_B)
Guidance/LOS_B
#Path Style (supported styles: sine, spiral, lawn_mower, straight)
sine
#Number of waypoints:
200
```

When running the simulation, the ROS package *rqt_graph* provides a flow chart of the situation. The information flow between the nodes (oval shaped boxes) through the topics (rectangular shaped boxes) is graphically presented in figure 5.18 for scenario 1. The nodes and topics that are in the simulation loop are emphasized with color.

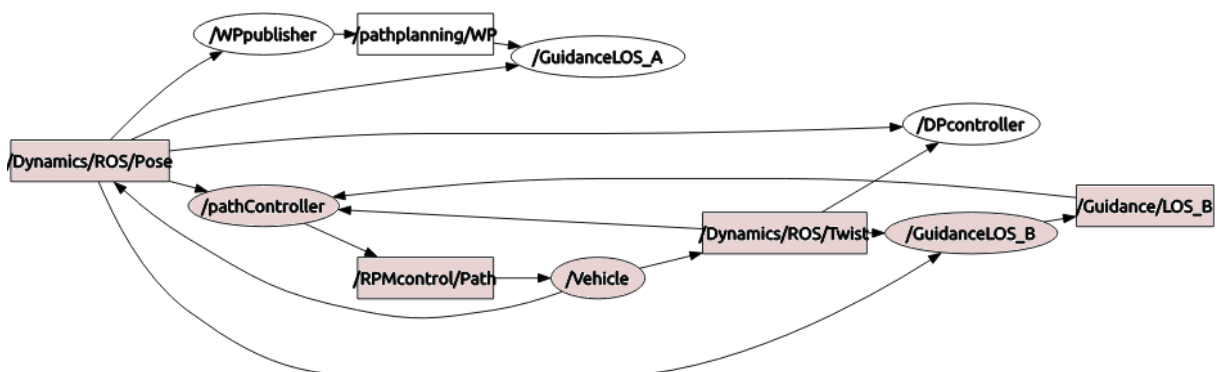


Figure 5.18: Node graph for scenario 1

It is seen that some of the oval shaped boxes has no arrow pointing out, meaning there is no other node subscribing to the topic published by the node. This is the case for both */DPcon-*

troller and */GuidanceLOS_A*, as it should be when the path controller and the LOS B guidance system was set to be in the simulation loop.

Scenario 2

In the next scenario, the LOS A guidance node was set to be in the loop. The last part of the configuration file was then written as shown in listing 5.2. The information flow in the simulation is shown in 5.19, where the nodes and topics that are in the simulation loop are emphasized. The difference from scenario 1 is seen by the input in the path controller node (*/pathController*), where it now uses input from LOS node A (through topic */Guidance/LOS_A*) instead of LOS node B (through topic */Guidance/LOS_B*).

Listing 5.2: Last part of configuration file, scenario 2

```
#Topics in loop for dynamics pose and twist :)
Dynamics/ROS/Pose
Dynamics/ROS/Twist
#Topic for controller (RPMcontrol/DP, RPMcontrol/Path)
RPMcontrol/Path
#Topic guidance (LOS_A, LOS_B)
Guidance/LOS_A
#Path Style (supported styles: sine, spiral, lawn_mower, straight)
sine
#Number of waypoints:
200
```

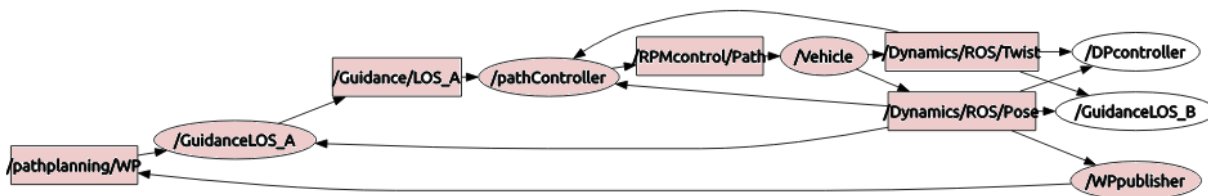


Figure 5.19: Node graph for scenario 2

Scenario 3

The last scenario is run with the DP controller in the loop, and with the configuration file as shown in listing 5.3. With the DP controller, there is no need for the guidance module or the path planner, so these can be excluded from the simulation if desired. The information about the guidance topic and path is disregarded by the DP controller. It is not necessary to run these nodes in the simulation, but there is either no harm in letting them run in parallel. Figure 5.20 shows the information flow for the scenario where the nodes and topics in the simulation loop are emphasized. The loop does now actually only include the dynamics node (*/Vehicle*) and the DP controller (*/DPcontroller*). There are other active nodes, such as the guidance nodes, path planner node and the path controller, but the information they produce is not used in the simulation loop.

Listing 5.3: Last part of configuration file, scenario 3

```

#Topics in loop for dynamics pose and twist :)
Dynamics/ROS/Pose
Dynamics/ROS/Twist
#Topic for controller (RPMcontrol/DP, RPMcontrol/Path)
RPMcontrol/DP
#Topic guidance (LOS_A, LOS_B)
Guidance/LOS_A
#Path Style (supported styles: sine, spiral, lawn_mower, straight)
sine
#Number of waypoints:
200

```

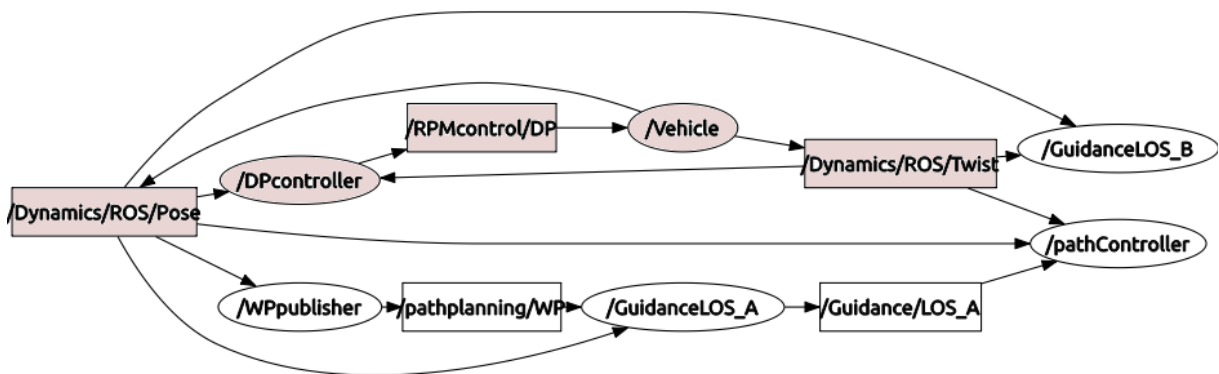


Figure 5.20: Node graph for scenario 3

5.3.2 Simulation results

As mentioned, the controllers and guidance systems are not tuned to perfection, but some results are presented here to show that they work and do the task they are supposed to.

Figure 5.21 shows results from three simulations done with the path controller. The path planner is set to generate a lawn mower pattern of length 40m and width 10m. The first simulation, figure 5.21a, shows a simulation done with the guidance module LOS B in the loop. The simulation was done with no current, and the ROV manages to follow the path. The same simulation has been done with guidance module LOS A in the loop, but this gave the same results as there are no current involved.

Figure 5.21b and figure 5.21c shows the results from simulations done with guidance modules LOS A and B respectively. These simulations were done with current present, in the positive east direction. It is seen that for the LOS A guidance module, the ROV does not manage to get back on the path due to the current. This is because the guidance system does not take the sideslip angle into account, as explained in section 4.4. With the LOS B guidance module in the loop, the ROV manages to steer back to the path, although the motions are a bit unstable due to bad tuning for both the controller and the guidance module.

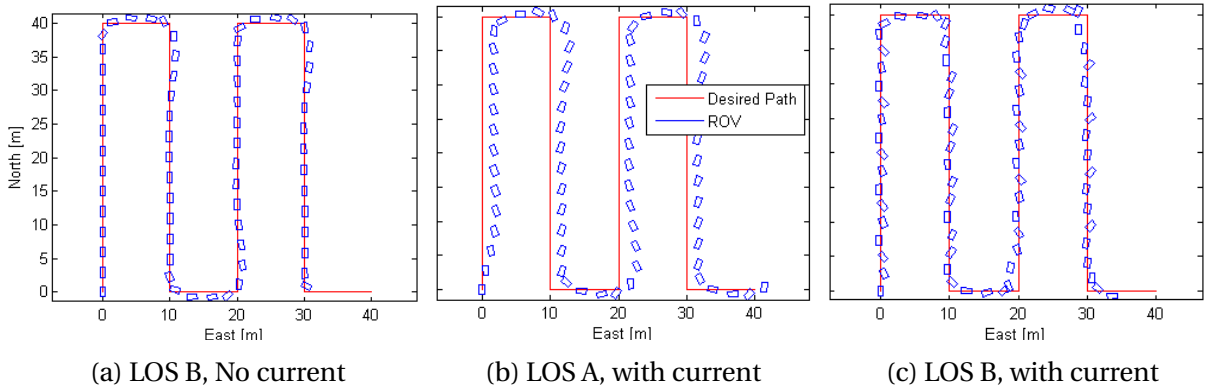


Figure 5.21: Results from simulations with path controller and guidance systems LOS A and LOS B guiding with a lawn mower pattern from the path planner. Current 0.05m/s to the east in the last two plots.

Figure 5.22 shows simulations done for path following of a straight path with current present. The current is in the positive north direction, and LOS A and LOB B guidance modules were used. The same tendencies as from figure 5.21 is seen, as the LOS A guidance system does not manage to guide the ROV back to the track. This is as expected, and the results are considered acceptable.

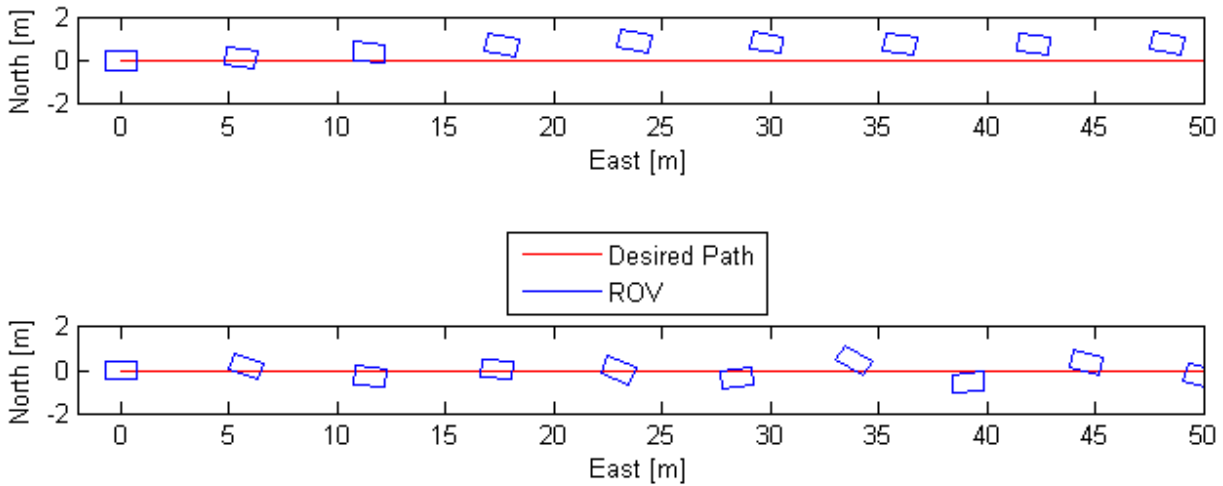


Figure 5.22: LOS A (upper) and LOS B (lower), current 0.05m/s north, straight path

Figure 5.23 shows simulations done with the DP controller and current present. The controller works as it should, and the resulting ROV motion is as expected.

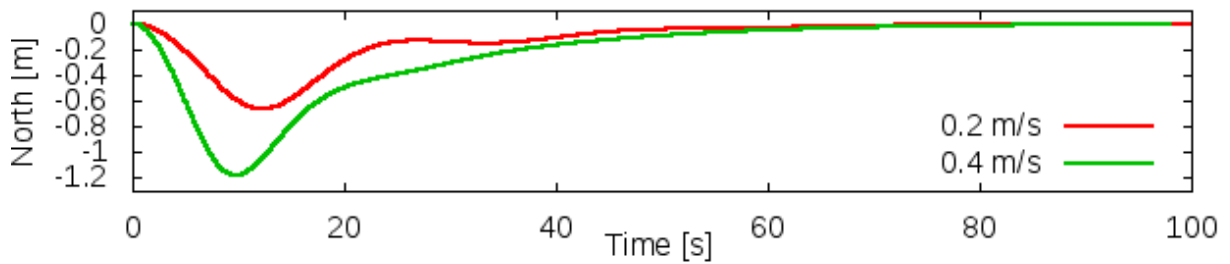


Figure 5.23: Testing DP controller with current present. Current is going south, and ROV is heading north.

Simulations have also been done with the MORSE dynamics in the loop, but they were highly affected by the troubles regarding the subscription of control info as explained in section 5.2.

Figure 5.24 shows the results from a simulation done with the MORSE dynamics in the loop, the path controller and the LOS A guidance module. The ROV is to follow the lawn mower pattern, and the real time simulation length was approximately 500 seconds.

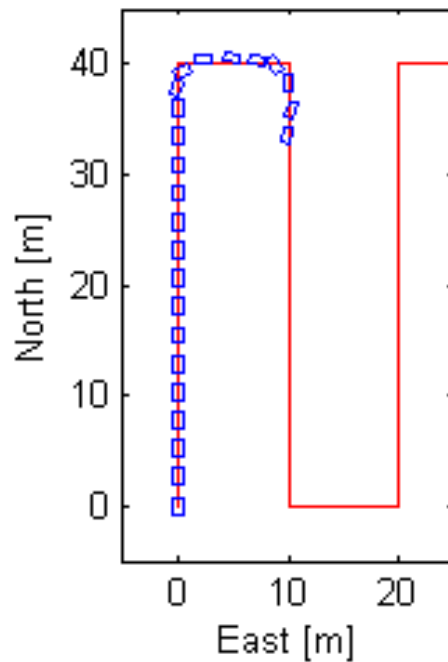


Figure 5.24: Results from simulation done with the MORSE dynamics, path controller and LOS A guidance system in the loop.

MORSE only managed to simulate approximately 200 seconds of ROV motion, where the first 100 seconds were simulated in real time, and the last 100 seconds were not. This affected both the controller and the guidance system, as they were guided and controlled by the assumption that the simulation was done in real time, and the integrals became much larger than they should be. The integral saturation stopped the final control output to become extremely misleading, and the problem is thus not seen in the figure.

6. Simulation Platform Evaluation

This chapter presents evaluations of different aspects of the simulation platform, in addition to an evaluation of the two different implementations of the dynamic model.

6.1 MORSE Dynamics VS ROS Dynamics

A mathematical model of the ROV dynamics have been implemented in both MORSE and ROS. Only one of these models should be further developed and used in the complete simulation platform. This section includes a discussion of the two models, and a proposal for which model that should be used in the continuation of the platform development.

Implementing the dynamics in ROS has the great advantage that the developer has full control of which equations and methods are used, as all software developed and used is available in the ROS package. This also implies that all software must be developed from scratch, and it will be difficult to take advantage of what other developers have done before. When using ROS alone for the dynamics, another program must be used for the 3D visualization. The implementation method will depend on the program used.

While the complete mathematical model was implemented in ROS, just parts of the model was implemented in MORSE. It was however done in combination with utilizing existing functions and methods in MORSE/Blender. The reason for implementing the dynamics in MORSE is that it already provides methods for physics simulation, and it is desirable to reuse software if possible.

MORSE provides collision simulation, and this property is absolutely desirable to include in the platform. To develop a collision simulation in ROS is too complicated and has not been an alternative.

MORSE also provides sensor models that can be used in the simulation platform. A few are already utilized, but many more will be natural to implement when estimators gets included in the platform. MORSE has also arranged for the users to include an environment in the simulations. In our case, this can be used to implement different seabeds which again can be interesting to use in combination with different sensors for simulations of seabed mapping.

It can be argued that the ROS dynamics software is easier to follow, as all calculations are done in the dynamics package. The way the dynamics is implemented in MORSE results in more spreading of the software, and troubleshooting gets harder as some parts of the software can be difficult to locate.

The dynamics module in MORSE is still a bit problematic, and some issues must be fixed before it can be used in the platform. The dynamics in ROS gives good results, but MORSE provides so many possibilities that should be taken advantage of.

One way to combine the best of both is to use the dynamics calculation in ROS and include a new actuator in MORSE that controls the ROV velocity based on the calculations from ROS. This is the same method as used by [DeMarco et al. \(2013\)](#) in the simulator for underwater human-robot interaction scenarios. The velocity will be used in MORSE to calculate the new position of the ROV with collision possibilities taken into account. The new position is fed back to the ROS dynamics calculation, which makes it possible to keep the ROS dynamics, visualization from MORSE, collision simulations and all sensor models available.

If MORSE and ROS are to be used in combination, there is no guarantee that we will not come across the same issues as already seen in MORSE, or new issues on the way. Because of this, it is concluded that the best outcome for the simulation platform would be to fix the issues in MORSE and include all dynamics here. It will be less confusing for the users if all dynamics is modeled in one platform. In addition, a large community of developers use MORSE in their work, and it is likely that new methods and classes will be developed that can benefit the development of the simulation platform.

6.2 Recommended User Knowledge

The simulation platform is developed with the idea in mind that it should be simple to use, even for users with no experience with ROS or MORSE. The user should get enough information about how to design a module by reading the code of an already developed module, and adjust it to fit her/his needs.

It is however highly recommended that the user has some experience with programming. The nodes can be written in C++, Python and Lisp, and the ROS developers are working on extending it to include Java and Lua as well ([Thomas, 2014a](#)). This is however not seen as a problem, as the target group for this simulation platform is software developers.

It is also recommended that the user has some experience using a Linux based operating system. As everything is executed by commands in the terminal, the user must be able to navigate in the software using the terminal. This is however possible to learn and become familiar with pretty fast, and is not a necessary prerequisite.

6.3 Execution of the Program

When starting a simulation, there are a number of ROS nodes that needs to be run, as well as the ROS master node and MORSE. They all must be initiated by commands in the terminal, and each node needs its own terminal window. This is a very cumbersome way to start a program, and a shell script is written to ease this situation.

The shell script will compile the software implemented in ROS, and it will initiate all the ROS

nodes and run MORSE. It will also start *roscore* if this is not already started. The shell script is named *executeSim* and can be found in */usvt/catkin_ws* in appendix B.

The shell script is run from the terminal, and it will initiate all necessary parts of the simulation platform. This way of executing the program is found to be a good method, and it makes it easy for the user to run simulations. If the user has made a new node and wishes to test it out, the node start-up must be included in the shell script. This requires an additional command in the shell script, and an example of how this is done is included in the script for users to read and learn.

6.4 User Interface

The simulation platform has no graphical user interface (GUI) for initializing the simulation. Everything is controlled from the terminal, in addition to editing the configuration file in a normal text editor.

When running a simulation, one terminal window is opened for *roscore*, one terminal window is opened for each ROS node initiated, and one terminal window is opened for MORSE. The high number of terminal windows can be a bit confusing for a new user, and may take some time getting used to.

The information written in the terminals will give the user updates related to the given module. This can come in handy in the development and debugging phase, where it may be necessary to print calculated values in the node to the terminal. When running simulations, this property is found to be impractical and one of the downsides of using ROS.

In addition to the multiple terminal windows, Blender provides a visualization of the ROV in a subsea environment during simulation. A screenshot of how this is seen by the user is shown in figure 6.1. This kind of visualization can be very helpful if it is used to e.g. simulate operations done with a manipulator on the ROV, such that the ROV mostly stays in the same position. The user can then change the camera view during simulation, and see the ROV in operation from different angles.

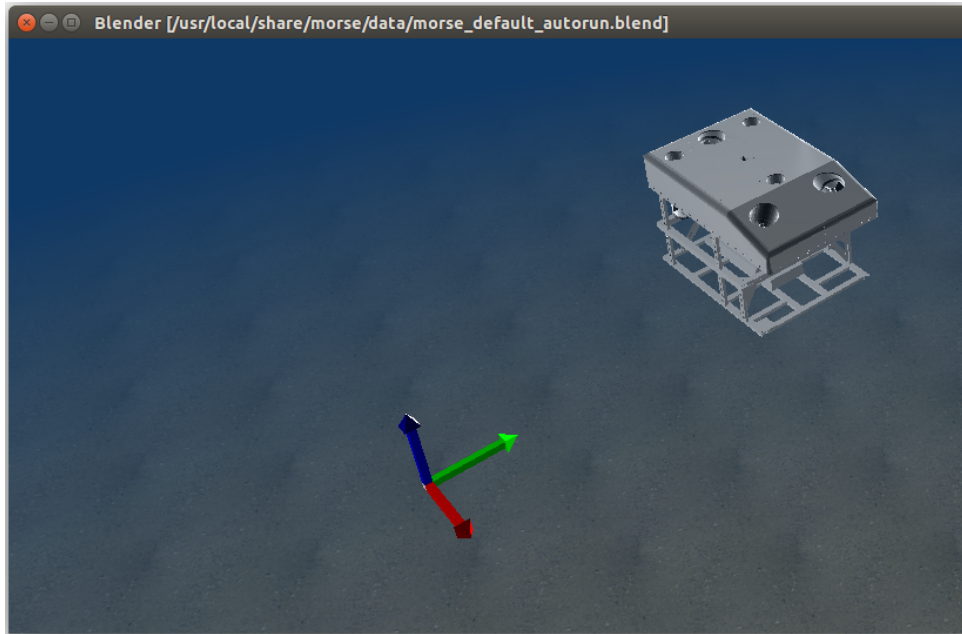


Figure 6.1: Screenshot of Blender during simulation

If the ROV is to move long distances in the simulation, the current visualization method is not suitable. The coordinate system provided by Blender does not show very good how the ROV moves in the NED plane, and it is hard to see if it moves as it should or not.

In this case, it is better for the user to see plots of how the position changes during simulation. ROS provides a package for displaying the variables in messages sent on the ROS network. The package *rqt_plot* provides 2D plots of the values published on the different topics, and the values are plotted in real time during the simulation. This is a very good tool that can be used when simulating larger ROV movements. An example of how the 2D plot can look like is shown in figure 6.2.

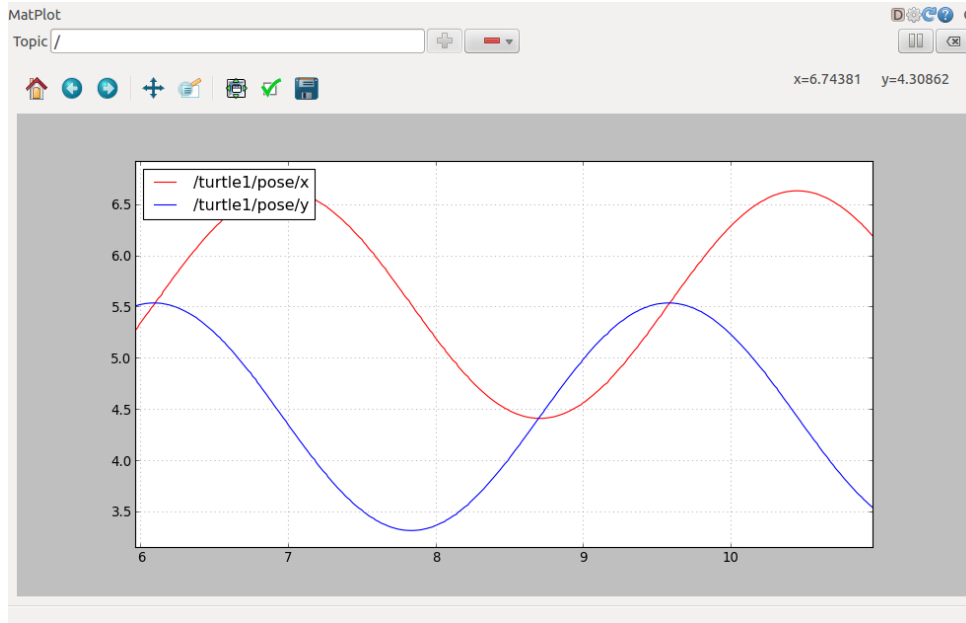


Figure 6.2: Example of 2D plot from `rqt_plot` package (Gregg, 2014)

Another helpful tool for the user is the `rqt_graph` package provided by ROS. This package will visualize the information flow between the nodes, as shown in 6.3. The square boxes display the topics and the elliptical shaped figures display the nodes.

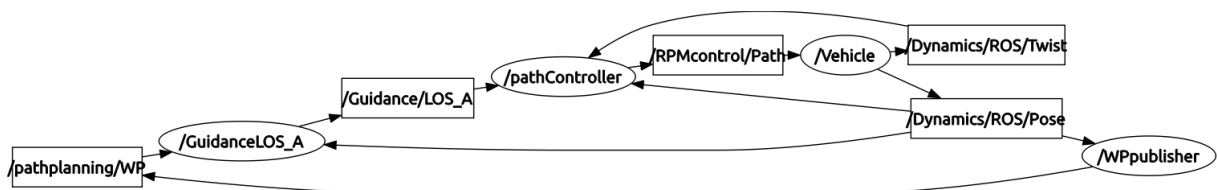


Figure 6.3: Example of a flowchart made by `rqt_graph`

This package is especially helpful for the user when the number of nodes in the simulation increases. It can be hard to keep track on which topics are used where, and the overview provided by `rqt_graph` makes it easier. The package can also make graphs of the topics connected to a certain node, or the nodes connected to a certain topic.

7. Discussion and Conclusion

7.1 ROS as a framework

ROS is used as a framework for the simulation platform. It has proved to be a good foundation, as it provides different useful solutions for the platform. The biggest advantage is the way ROS structures the software, and the methods provided for communication between the different parts of the software. The way ROS splits up the software in packages organizes the software neatly and it makes it easy for the user to navigate through the software. ROS also provides methods for easy transition between simulations and experiments, which is a very important factor to consider. The main downside of the framework is that each node requires its own terminal, which can be very chaotic if multiple nodes are active.

To keep the modularity of the platform, it was decided to let all nodes publish messages on different topics. This makes it possible to decide which modules to be in the simulation loop by altering the topics the different nodes subscribes to, and not the ones that are published to. Different controllers can then publish control info to their respective topics, while the dynamics node decide which controller to be in the loop by changing the topic subscribed to. This solution is found to be a good fit for the platform as it makes it possible for multiple modules of the same type to run simultaneously. This can be utilized to make it possible to switch between modules during simulations.

7.2 Configuring Simulations

The simulation platform requires a configuration file before simulations can be launched. The file includes information about the ROV to be simulated and information about which modules to include in the simulation loop. These specifics are chosen to be specified in a file because it makes it easy for the users to alter the information. If the user wants to change the mass of the ROV or switch to another controller module, she/he must only alter the configuration file. By not making it necessary for the user to locate the code in the software and alter the values directly, the platform is found to be more user-friendly.

The drawback for the configuration file is that it is too large. While developing the simulation platform, more and more information was included in the configuration file, and it became too comprehensive. The file is constructed such that initial conditions are specified first, then comes specifics about the ROV, thruster characteristics and at last names of topics that define which modules to include in the simulation loop. The nodes read the file line by line,

and needs to know exactly where in the file the information is to be able to store the data correctly. The nodes that only need information about topics have to skip through almost everything in the configuration file before they can register the topics. If it is decided to include more information about the ROV in the configuration file, it will typically be included where the rest of the ROV specification is stored. This results in all information about thrust characteristics and topics will be shifted down a few lines in the file, and software for all nodes that require information about topics must be altered such that they read the correct information. This is seen as a very bad quality, and should be improved by e.g. splitting the configuration file into multiple configuration files.

7.3 Dynamic ROV model

A dynamic model of an ROV has been implemented in both ROS and MORSE. The user is required to include the ROV specifications in the configuration file, and the model can therefore be used to simulate a wide range of ROVs and possibly also AUVs.

The simulation results from tests done with the ROS dynamics give realistic results. It cannot be expected that it simulates the reality perfect, but the most important physical effects are taken into account and the results indicate that the model is correct.

The results from tests done with the MORSE dynamics are not satisfactory. Results show that the simulated ROV reacts faster than expected when exposed to forces, but the hydrodynamic forces calculated are investigated and no faults are detected. When studying the way the ROV reacts to forces, the problem seems to be related to inertia. When comparing results from ROS dynamics and MORSE dynamics with no thrust forces (just buoyancy and gravity forces) it is seen that they both reach the same maximum velocity, but the ROV simulated with MORSE dynamics reaches the maximum velocity much faster. The problem was first thought to be in the way the added mass matrix is included when calculating the velocity. If the added mass is missing from the equation, this would result in a smaller inertia and faster reactions of the ROV. The troubleshooting was unfortunately inconclusive, but there does not seem to be a problem with just the added mass, rather how the complete inertia is included. The problem is however not believed to be significant as the motion differences between the ROS dynamics and MORSE dynamics are not large compared to the motion itself. It is also a possibility that there is no mistake in MORSE and the motion difference is due to a mistake in ROS. This possibility is seen as unlikely.

Another problem encountered with the MORSE dynamics is that the simulation does not manage to simulate in real time for a long period. The issue was investigated, and the problem is found to be related to the way control info is received from the ROS controller. It is found that when the controller publishes information with a frequency of 10 Hz, the MORSE dynamics only manages to do simulations in real time for 100 seconds before the calculations slow down. When the controller publishes with a frequency of only 1Hz, the simulation is done in real time for about 250 seconds.

It is seen that when the simulation first starts to slow down, it slows down more and more. This was first found very strange, as it would be more natural if the simulation slows down from the start. After some consideration, it is concluded that if the calculations first starts to

stack up, the calculations will first after a while exceed the length of one time step. When this happens, a queue starts building up, and MORSE does not manage to finish all the calculations in the next time step as well. The simulation will then slow down more and more.

The problem is thought to be the way the control messages are subscribed to. The subscription call is the same as used in ROS nodes, although MORSE is not a ROS node. To use this method, the subscription has to be done in each time step, which can be a very time consuming task. The method should be changed to the more general way of subscribing to messages; by using a datastream handler. This method is unfortunately not tested in the platform yet.

In spite of the troubles with the dynamics in MORSE, it is concluded that it should be further developed at the expense of the ROS dynamics. Although the ROS dynamics gives good results now, the MORSE dynamics has a greater potential. ROS has e.g. no built-in methods to simulate collisions, and it would be an unnecessary use of time to develop such a comprehensive model from scratch when we have the possibility of using an already developed model. This yields not only for collision simulation, but also other aspects of dynamics modeling. As MORSE has a large community of developers, it is possible that also new methods that benefits the simulation platform will be developed. The proposal of using the best from both ROS and MORSE is also rejected, as it will be more user-friendly if the complete dynamic module is kept in one framework.

For further development of the platform, the ROS dynamics module can be used as a helpful tool before the problems with the MORSE dynamics are solved. When e.g. developing an estimator module, it would be great to have a model of the dynamics to test the estimator. In the development phase there is not a huge desire for 3D visualization anyway, so the ROS dynamics can be used before the issues with the MORSE dynamics is fixed.

7.4 The Complete System

Two guidance modules and two controller modules are developed for the control system in the simulation platform. They are tested and it is concluded that they work as they should. They are mainly developed to test how the modularity of the platform works in practice, and to provide an example of how modules can be implemented.

As a complete system, the platform is not working perfect just yet. If the user wants dynamics modeling for a longer period than 100 seconds, she/he must use the ROS dynamics and does not get a 3D visualization while simulating. The user should however easily manage to implement her/his own algorithms for controllers, guidance systems or path planners, as this system has proven to work as planned. The platform will then provide simulation results based on the algorithm implemented by the user.

8. Further Work

The development of the simulation platform has just been started, and there is a lot of work left before all the goals are achieved. This chapter will present proposals for further work to continue the development of the platform, including some specific proposals of how the problems can be solved.

8.1 User Interface

It is not found necessary to include a graphical user interface (GUI) for simulation start-up. Using a configuration file to configurate the simulation and executing it from the terminal is an easy task that all users should manage.

For presenting the simulation results, it is desired to improve and extend the GUI. As it is now, a 3D model of the ROV is visualized, and the user can use the computer mouse and keyboard to move and rotate the camera view, and see the ROV from different angles. Plots are also presenting how the values in the different messages sent on the ROS topics changes with time, and the user can click and choose which topic messages she/he wants presented.

As discussed in section 6.4, the way the 3D ROV model is visualized during the simulation is not satisfactory for simulations that include large ROV movements. The 2D plots of the message values improves the situation, but it is still not ideal for the user.

It is proposed to include a trajectory plot that can be seen in simulation time. This will be in addition to the 3D model and the 2D time plots. A proposal for how this can look for the user is shown in figure 8.1.

A trajectory plot is very useful when simulating the ROV for path following or trajectory tracking. The plot will provide a visual of where the ROV is in the north-east plane (or possibly east-down or north-down if desired), and where it has been.

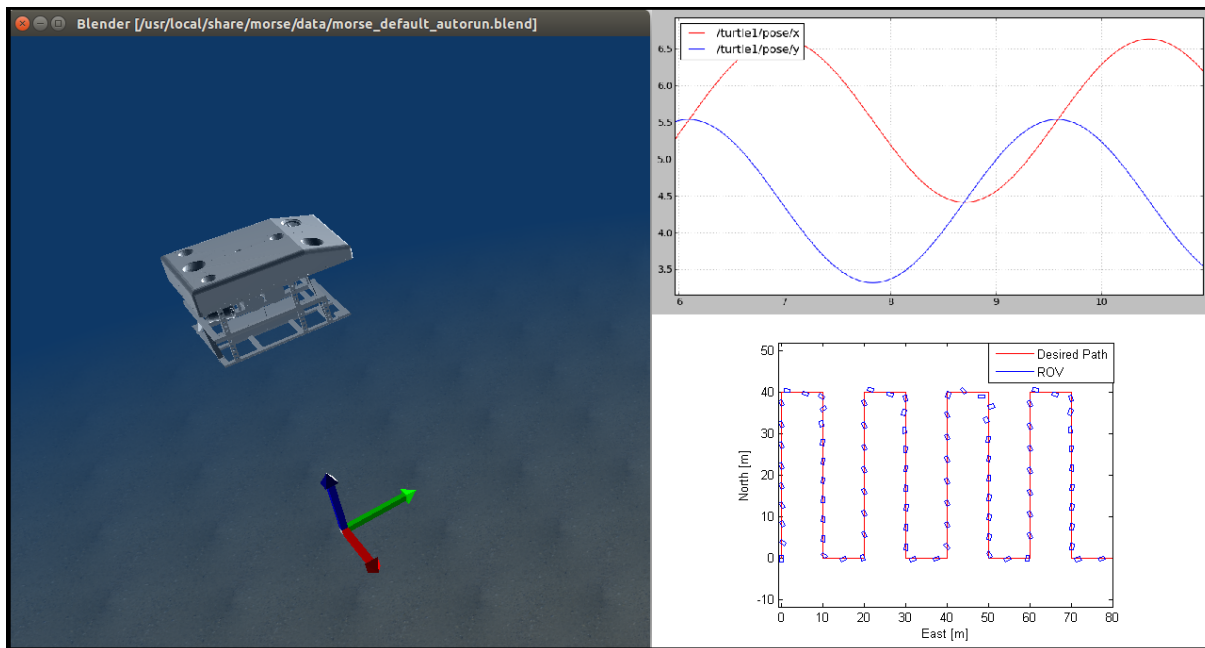


Figure 8.1: Proposal for visual presentation of simulation results

If this is completed, the user will get a much better understanding of how the ROV acts while the simulation runs, and the user will be better informed. This can be a very important quality in the development phase of e.g. new controllers or guidance systems, as the user is informed in real time, and can shut down the simulation whenever she/he has seen enough. If the user has to wait until the simulation is finished before she/he can see the trajectory plot, it is hard to know when to shut down the simulation.

8.2 Software Documentation and Development Guidelines

All parts of the platform must be well documented if the plans of making it open source can be realized. This requires both documentation of the developed modules, documentation of how they are implemented in ROS and tutorials on how to make modules from scratch.

The planned simulation platform can be seen as a never ending project, as it can always be improved and new modules can be included. When inviting other developers to implement their algorithms and develop new modules, a set of development guidelines must be defined. This is to avoid an increasing maintenance problem when the number of modules and algorithms increase, and to keep the modular property of the simulation platform intact.

8.3 Software organization

Well defined communication protocols between the modules, and also a well defined categorizing of the modules are required. By following the modularity offered by ROS, the software for the different modules are grouped into packages. Each package includes the soft-

ware for certain types of modules, e.g. one controller package that includes multiple controller modules. As the number of modules included in the simulation platform increases, it may become chaotic to only separate the software in packages.

It is proposed to include further separation of the software inside each package. This new grouping should separate the modules of the same type (e.g. controller modules) based on their required input values. The grouping should not be on the type of controller, meaning that backstepping controllers and PID controllers are not necessarily placed in different groups. The grouping must be based on the input values needed. If a backstepping controller module uses desired heading and speed as input, it should be grouped together with other controllers that requires the same input values, even if it is just a simple P-controller. If a controller module is designed with a new set of input values, a new group must be established. All modules in one group must use the same message type for subscribing to input values, and the same message type for publishing. The same type of organization is needed for all packages.

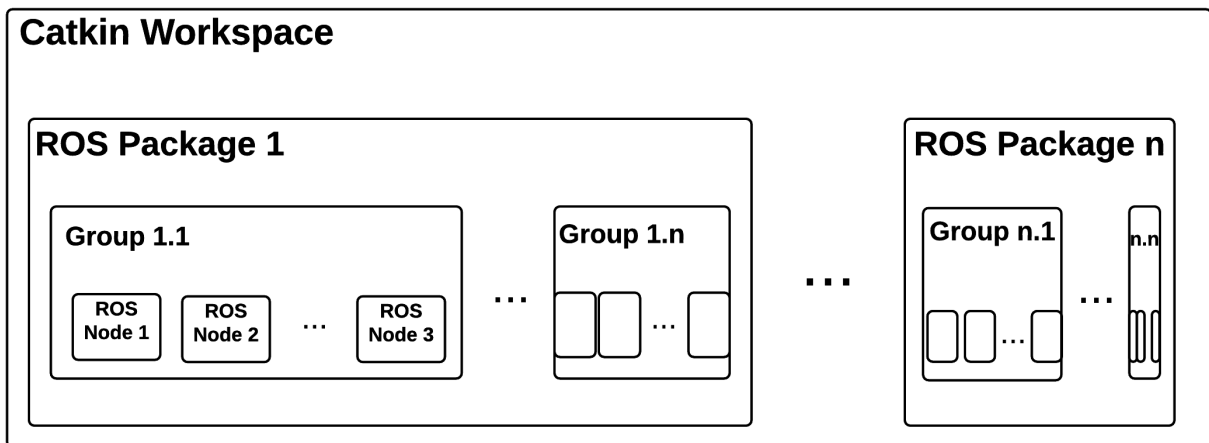


Figure 8.2: Proposed extension of the software organization.

8.4 Dynamics modelling

The dynamic model can always be improved by including so far neglected physical effects, e.g. including wave forces and near wall effects when the ROV operates close to the seabed. The forces from the umbilical should also be included.

It is also very desirable to include more environmental forces in the simulation. This includes wave loads on the ROV and also current simulation with possibilities of including different current profiles. It is proposed to include a new type of module describing either the additional forces caused, the velocity change in the water due to environmental conditions or a combination of both.

To simulate the wave loads, the typical wave spectrum of the desired weather conditions should be included in the input, as well as current information. It would be nice if the simulation can take into account how the current varies with depth, and also how it changes

with time. These disturbances from waves and current will also highly affect the umbilical connected to the ROV.

Extending the dynamic model must come in addition to investigating the issues discovered in the dynamics regarding inertia, as explained in 5.2.

8.5 Improvements of Configuration File

While working with the simulation platform, it is seen that the configuration file has become too comprehensive. Too much information is stored in one file, and this can be confusing for the user, as well as it is easy to make one mistake early in the file that shifts all the lines down and makes the rest of the file useless.

As a solution to this problem, it is proposed to separate the configuration file into multiple files. A suggestion is to define one file for ROV specifications, one for thruster characteristics, one for topic information, and one for initial conditions and environmental conditions.

This change requires that all functions written for reading the configuration file must be modified to fit the new design.

8.6 General improvements in MORSE

8.6.1 Subscription to Controller Messages

The way MORSE subscribes to control messages is done in a non-general way, and will only work if the control info is published on a ROS topic. This is a solution that works when all controller software is implemented in ROS, but not if it was implemented in another middleware. It would be nice when it comes to platform expansion possibilities if this was done in a more general way.

It is therefore proposed to change the way the controller message is read. This can be done by using *local data* to store control information, and a *datastream handler* to establish a connection between MORSE and the middleware (in this case ROS).

As described in section 5.2, there is a problem with the subscription of controller messages, as the MORSE dynamics calculation slows down a while into each simulation. Hopefully, this problem can be solved by using the datastream handler. If the issue is still present, more troubleshooting must be done to figure out where the bug is.

8.6.2 Read configuration file

The configuration file is read in the builder script, and the values are sent to the actuators. This should be done in a separate script, as the builder script gets very large and confusing when the code for reading the configuration file is included.

Not all information from the configuration file is read and sent to the actuators, meaning that some of the data is hard-coded. This must be fixed, but it should be done in combination with altering the configuration file and splitting it up in several files, as explained in section 8.5.

8.7 Expansion of the Simulation Platform

Expansion of the simulation platform can be done both by developing new types of modules, or including more modules of the same type. New controllers can always be included, as well as more comprehensive path planners.

When including new types of modules, the first priority should be to develop an estimator module. This can be done by implementing a Kalman filter, and must use input from the sensor models.

Another possible module to include is the dynamics of a manipulator arm. It can be included as a separate module that interacts with the ROV dynamics module. The dynamics module for a manipulator arm would introduce multiple new modules related to control of the arm. A manipulator controller can be included to control the joints of the manipulator arm, and some kind of guidance system can be included to decide how the joints should move such that the end effector acts as desired. Sensor models for the joint angles are needed, as well as a path planner for the motion of the end effector.

If the set of modules related to manipulator control is included, the utility of the platform will be highly extended. ROV operations with manipulator control can be simulated, and this arranges for a way to test control systems for the manipulator in combination with control of the ROV.

8.8 Post Simulation Plotting

It would be nice for the user if a method for plotting simulation results after the simulation had been implemented.

While developing the platform, this was done by a node in ROS that subscribed to the topics planned to plot after the simulation. This included the pose messages and velocity messages from both the ROS dynamic module and MORSE. Gnuplot ([Williams and Kelley](#)) was then used to make 2D plots of the obtained values. This node is called *verification* and is included in the *dynamics_verification* package in appendix B. The method requires that the values to be plotted are published on the ROS network. This is often not the case if we want to plot e.g. hydrodynamic forces or thrust forces.

For the thrust forces and hydrodynamic forces, the values were written to a text file directly in the dynamics modules. This was a cumbersome method that made the software scripts larger and harder to follow. A better method should be found for storing values for post simulation plotting.

The *verification* node was also found as a bad method for storing published messages. It was later discovered that ROS provides two packages called *rosbag* and *rqt_bag* for storing and plotting published messages. The packages should be studied and if found suitable, implemented in the platform.

Bibliography

Balchen, J., Andresen, T., and Foss, B. (2003). *Reguleringsteknikk*.

blender.org. Blender. blender.org. Accessed: 26.05.2015.

bulletphysics.org. Real-Time Physics Simulation. <http://bulletphysics.org/>. Accessed: 26.05.2015.

Demarco, K., West, M., and Collins, T. (2011). An implementation of ros on the yellowfin autonomous underwater vehicle (auv).

DeMarco, K., West, M., and Howard, A. (2013). A simulator for Underwater Human-Robot Interaction scenarios. In *Oceans - San Diego, 2013*, pages 1–10.

Echeverria, G., Lassabe, N., A., D., and Lemaignan, S. (2011). Modular open robots simulation engine: MORSE. pages 46–51. Cited By :22 Export Date: 19 February 2015.

Faltinsen, O. M.

Fossen, T. I. (1991). Nonlinear Modelling and Control of Underwater Vehicles. PhD Thesis. Department of Engineering Cybernetics, Norwegian Institute of Science and Technology. Trondheim, Norway.

Fossen, T. I. (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control*.

Gregg, M. (2014). rqt_plot package summary. http://wiki.ros.org/rqt_plot. Accessed: 02.06.2015.

Hedrick, J. K. and Girard, A. (2005). *Control of Nonlinear Dynamic Systems: Theory and Applications*.

Kermorgant, O. (2014). A dynamic simulator for underwater vehicle-manipulators.

Kirkeby, M. (2010). *Comparison of Controllers for Dynamic Positioning and Tracking of ROV Minerva*.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. volume 3, pages 2149–2154.

marinesimulation.com. Rov training simulator. marinesimulation.com. Accessed: 05.06.2015.

- Mo, S. M. (2014). *Modelling and simulation of ROV-manipulator system*.
- Myrhaug, D. (2006). *Oceanography - Wind, Waves*.
- Newman, P. M. (2002). MOOS - Mission Oriented Operating Suite.
- Novi, G., Melchiorri, C., García, J., Sanz, P., Ridao, P., and Oliver, G. (2009). A new approach for a reconfigurable autonomous underwater vehicle for intervention. pages 23–26.
- Pedro J. Sanz, Pere Ridao, G. O. C. M. G. C. C. S. Y. P. A. T. (2010). Trident: A framework for autonomous underwater intervention missions with dexterous manipulation capabilities. In *in proceedings of the 7th Symposium on Intelligent Autonomous Vehicles IAV-2010*. IFAC.
- Prats, M., Perez, J., Fernandez, J., and Sanz, P. (2012). An open source tool for simulation and supervision of underwater intervention missions. pages 2577–2582.
- Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.
- Shin, D. and Singh, S. (1990). Path generation for robot vehicles using composite clothoid segments.
- Sørensen, A. J. (2013). *Marine Control Systems: Propulsion and Motion Control of Ships and Ocean Structures*. 3 edition.
- Steen, S. (2011). *Motstand og Propulsjon, Propell-og Foilteori*.
- Strømsøyen, S. and Mo, S. M. (2014). Assignment 3 in TTK 4190 Guidance and Control of Vehicles. Project done as a part of a course at NTNU.
- Thomas, D. (2014a). ROS Wiki: Introduction. <http://wiki.ros.org/ROS/Introduction>. Accessed: 21.04.2015.
- Thomas, D. (2014b). ROS Workspace. <http://wiki.ros.org/action/fullsearch/catkin/workspaces>. Accessed: 09.06.2015.
- Williams, T. and Kelley, C. gnuplot 4.6. http://www.gnuplot.info/docs_4.6/gnuplot.pdf. Accessed: 05.06.2015.

Appendices

A. Software Organization in ROS

This appendix will present the composition of the software in ROS. The information is based on documentation of the ROS workspace (Thomas, 2014b). When describing the software structure, a slash / will be used. Example: */Folder1/Subfolder1* means that the folder *Subfolder1* is stored inside a folder named *Folder1*.

The software for building simulations is included in a catkin workspace, a folder often named *catkin_ws*. The workspace contains four folders; *build*, *devel*, *install* and *src*.

Build Space: /catkin_ws/build

The build space is where packages are built to. The space will contain folders for all packages. The user does not have to change anything in this folder.

Development Space: /catkin_ws/devel

This folder contains environment setup files, generated configuration files, header files and libraries among others. The user of the simulation platform does not need to worry about this folder, as the files in the folder should not be modified by anyone.

Install Space: /catkin_ws/install

The install space is used when installing packages in the workspace.

Source space: /catkin_ws/src

This folder contains source code for all the catkin packages. All subfolders in the source space is a *catkin package*. The main composition of each catkin package follows:

Folders:

/catkin_ws/src/package_name/include - Folder containing header files.

/catkin_ws/src/package_name/msg - Folder containing message files (IDL files).

/catkin_ws/src/package_name/srv - Folder containing ROS services.

/catkin_ws/src/package_name/src - Folder containing all source files for the package.

Files:

/catkin_ws/src/name_of_package/CMakeList.txt - File used for building packages. This file must be edited by the user when new ROS nodes are implemented and new subfunctions are created. The dependencies are stated in this file, in addition to which directories and message files to include.

/catkin_ws/src/name_of_package/package.xml - This is a package manifest that includes information about the package (name, developers, package description, etc) and how the package depend on other packages.

B. Digital Attachments

All software developed for the simulation platform is included in a digital attachment. The most important files and folders for further development of the platform are described here. The digital attachment also includes a poster about the thesis.

B.1 MORSE Files and Folders: */uvst/morse/subsea*

uvst/morse/subsea/default.py - Builder script for MORSE simulation.

uvst/morse/subsea/src/subsea/actuators - Python scripts for the actuators FluidForces and ThrustForces.

uvst/morse/subsea/src/subsea/robots/ThirtyK.py - Script for initiation of ThirtyK robot used in simulation.

uvst/morse/subsea/data/subsea/environments - Blender file for subsea environment.

uvst/morse/subsea/data/subsea/robots - Blender file for ROV.

B.2 Modifier classes: */uvst/Modifiers*

Modifier/ned.py - The new version of the NED modifier *ned.py* including the new modifiers developed for the simulation platform. The modifier is normally stored in */usr/local/lib/python3/dist-packages/morse/modifiers*.

B.3 ROS Files and Folders: */uvst/catkin_ws*

uvst/catkin_ws/src/control - Package containing the controllers *pathController* and *DPcontroller* and all their subfunctions.

uvst/catkin_ws/src/dynamics - Package containing the ROS dynamics node and all subfunctions.

uvst/catkin_ws/src/dynamics_verification - Package containing verification node that stores simulation results.

uvst/catkin_ws/src/guidance - Package containing the guidance systems LOS A and LOS B including their subfunctions.

uvst/catkin_ws/src/pathplanning - Package containing the path planner node and all subfunctions.

B.4 Configuration files: */uvst/catkin_ws*

uvst/catkin_ws/inputFile.txt - Configuration file with ROV specification, thrust characteristics and topic information.

uvst/catkin_ws/executeSim.bash - Shell script for running complete simulation with all modules.

B.5 Poster: *uvst/Poster*

uvst/Poster/Poster.pdf - The poster made for the Master Thesis Poster Exhibition 2015.

C. Modifier classes

New modifiers are developed for the platform. They must be included in the MORSE source code.

Listing C.1: Modifier class to transform measured body coordinate velocity in SFU (Starbord-Forward-Up) to FSD (Forward-Starbord-Down)

```
class SFUvelocityToFSD(NEDModifier):
    def modify(self):
        try:
            tmp=self.data['world_linear_velocity'][0]
            self.data['world_linear_velocity'][0]=self.data['world_linear_velocity'][1]
            self.data['world_linear_velocity'][1]=tmp
            self.data['world_linear_velocity'][2]=-self.data['world_linear_velocity'][2]
            tmp=self.data['angular_velocity'][0]
            self.data['angular_velocity'][0]=self.data['angular_velocity'][1]
            self.data['angular_velocity'][1]=tmp
            self.data['angular_velocity'][2]=-self.data['angular_velocity'][2]
        except KeyError as detail:
            self.key_error(detail)
```

Listing C.2: Modifier class to transform measured body coordinate velocity in SFU (Starbord-Forward-Up) to FSD (Forward-Starbord-Down)

```
class SFUposeToFSD(NEDModifier):
    def modify(self):
        try:
            tmp = self.data['x']
            self.data['x'] = self.data['y']
            self.data['y'] = tmp
            self.data['z'] = - self.data['z']
            tmp=self.data['roll']
            self.data['roll']=self.data['pitch']
            self.data['pitch']=tmp
            self.data['yaw']=-self.data['yaw']
        except KeyError as detail:
            self.key_error(detail)
```


D. System Matrices

The matrices from the equation of motion (4.3) are given here. They are all collected from (Fossen, 2011).

D.1 Mass Matrix

$$M_{RB} = \begin{bmatrix} mI_{3 \times 3} & -mS(r_g^b) \\ mS(r_g^b) & I_b \end{bmatrix} \quad (\text{D.1})$$

Where $S(\lambda)$ is the skew symmetric matrix (D.2), I_b is the inertia matrix and r_g^b is the vector from CO to COG.

$$S(\lambda) = \begin{bmatrix} 0 & -\lambda_3 & \lambda_2 \\ \lambda_3 & 0 & -\lambda_1 \\ -\lambda_2 & \lambda_1 & 0 \end{bmatrix} \quad (\text{D.2})$$

D.2 Coriolis and Centrepetal matrices

$$C_{RB} = \begin{bmatrix} 0_{3 \times 3} & -mS(v_1) - mS(v_2)S(r_g^b) \\ -mS(v_1) + mS(r_g^b)S(v_2) & -S(I_b v_2) \end{bmatrix} \quad (\text{D.3})$$

$$C_A = \begin{bmatrix} 0_{3 \times 3} & -S(A_{11}v_1 + A_{12}v_2) \\ -S(A_{11}v_1 + A_{12}v_2) & -S(A_{21}v_1 + A_{22}v_2) \end{bmatrix} \quad (\text{D.4})$$

v_1 and v_2 are defined in (D.5), and A is defined in (D.6).

$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \quad (\text{D.5})$$

$$M_A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad (\text{D.6})$$

D.3 Restoring forces

$$g(\eta) = \begin{bmatrix} (W - B) \sin \theta \\ -(W - B) \cos \theta \sin \phi \\ -(W - B) \cos \theta \cos \phi \\ -(y_g W - y_b B) \cos \theta \cos \phi + (z_g W - z_b B) \cos \theta \sin \phi \\ (z_g W - z_b B) \sin \theta + (x_g W - x_b B) \cos \theta \cos \phi \\ -(x_g W - x_b B) \cos \theta \sin \phi - (y_g W - y_b B) \sin \theta \end{bmatrix} \quad (\text{D.7})$$

W is the weight of the body and B is the buoyancy force. $[x_b, y_b, z_b]$ denotes the center of buoyancy and $[x_g, y_g, z_g]$ denotes the COG.

E. Vessel Data Minerva ROV

Center of Buoyancy and Gravity

$$\begin{aligned} COB &= [0, 0, -0.12]^T \\ COG &= [0, 0, 0.15]^T \end{aligned} \tag{E.1}$$

Volume

$$V = 0.49m^3 \tag{E.2}$$

Mass Matrices

$$M_{RB} = \begin{bmatrix} 460 & 0 & 0 & 0 & 0 & 0 \\ 0 & 460 & 0 & 0 & 0 & 0 \\ 0 & 0 & 460 & 0 & 0 & 0 \\ 0 & 0 & 0 & 105.26 & 0 & 0 \\ 0 & 0 & 0 & 0 & 104.02 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50.31 \end{bmatrix} \tag{E.3}$$

$$M_A = \begin{bmatrix} 293 & 0 & 0 & 0 & 0 & 0 \\ 0 & 302 & 0 & 0 & 0 & 0 \\ 0 & 0 & 326 & 0 & 0 & 0 \\ 0 & 0 & 0 & 52 & 0 & 0 \\ 0 & 0 & 0 & 0 & 52 & 0 \\ 0 & 0 & 0 & 0 & 0 & 57 \end{bmatrix} \tag{E.4}$$

Damping Matrices

$$D_L = \begin{bmatrix} 29 & 0 & 0 & 0 & 0 & 0 \\ 0 & 41 & 0 & 0 & 0 & 0 \\ 0 & 0 & 254 & 0 & 0 & 0 \\ 0 & 0 & 0 & 34 & 0 & 0 \\ 0 & 0 & 0 & 0 & 59 & 0 \\ 0 & 0 & 0 & 0 & 0 & 45 \end{bmatrix} \quad (\text{E.5})$$

$$D_q = \begin{bmatrix} 292 & 0 & 0 & 0 & 0 & 0 \\ 0 & 584 & 0 & 0 & 0 & 0 \\ 0 & 0 & 635 & 0 & 0 & 0 \\ 0 & 0 & 0 & 84 & 0 & 0 \\ 0 & 0 & 0 & 0 & 148 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix} \quad (\text{E.6})$$

Thruster Characteristics

$$T = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.985 & 0.985 \\ 1.0 & 0.0 & 0.0 & -0.174 & 0.174 \\ 0.0 & 1.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.20 & -0.20 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.166 & 0.0 & 0.0 & 0.335 & -0.335 \end{bmatrix} \quad (\text{E.7})$$

$$K_T = \begin{cases} 0.5J^3 - 0.66J^2 - 0.25J + 0.24 & n > 0 \\ 0.025J^3 - 0.28J^2 + 0.17J + 0.15 & n < 0 \end{cases} \quad (\text{E.8})$$

Table E.1: Thrust loss factors for each thruster

Thruster nr	1	2	3	4	5
θ $n > 0$	1.04	0.58	0.58	0.72	0.72
θ $n < 0$	1.04	0.58	0.58	0.53	0.53

F. Flow Charts for ROS Nodes

This appendix contains the flow charts showing the function calls in the different ROS nodes.

F.1 ROS Dynamics module - Vehicle

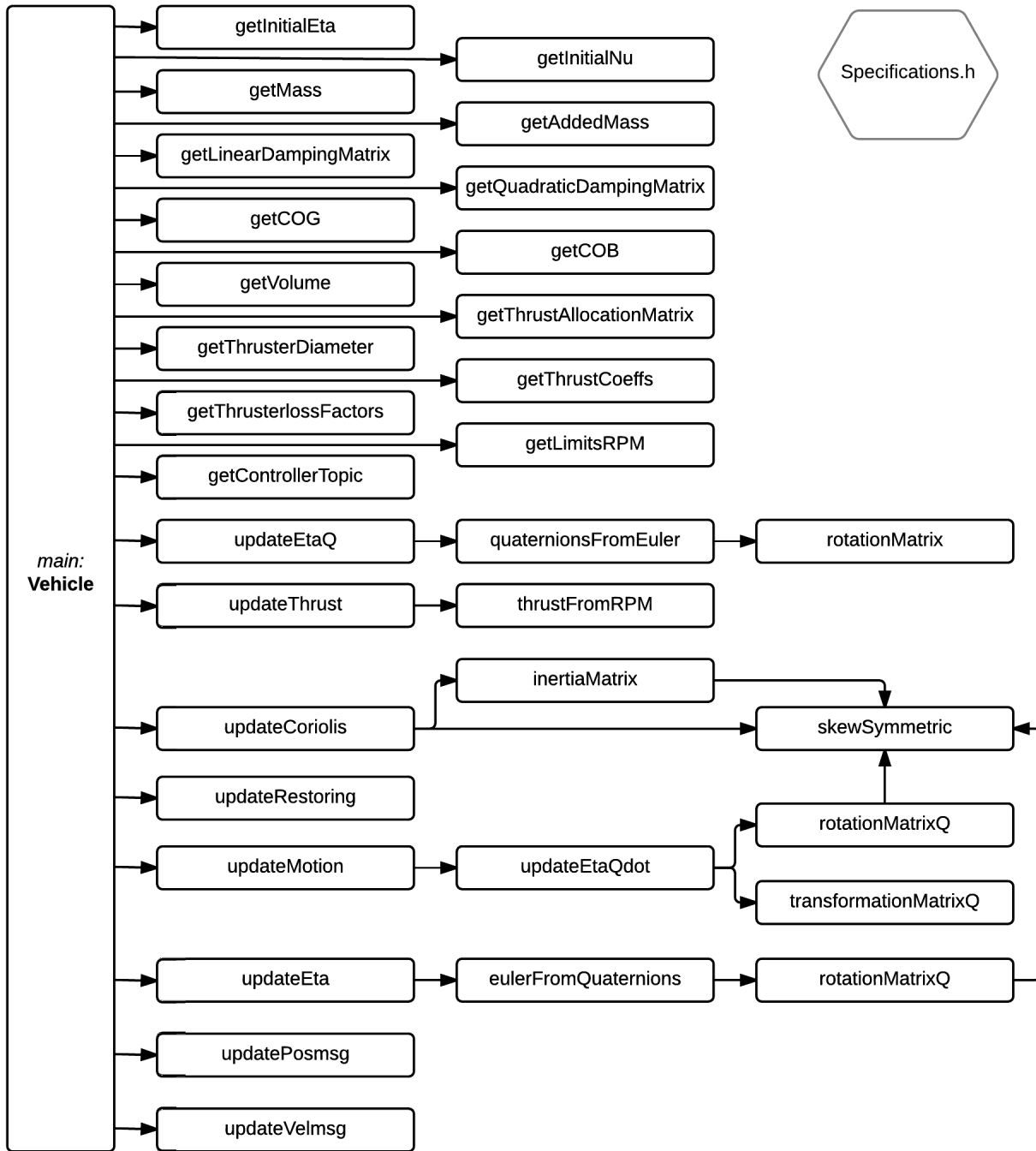


Figure E.1: Flow chart for vehicle dynamics node *Vehicle* in ROS

E.2 Path planning module - WPpublisher

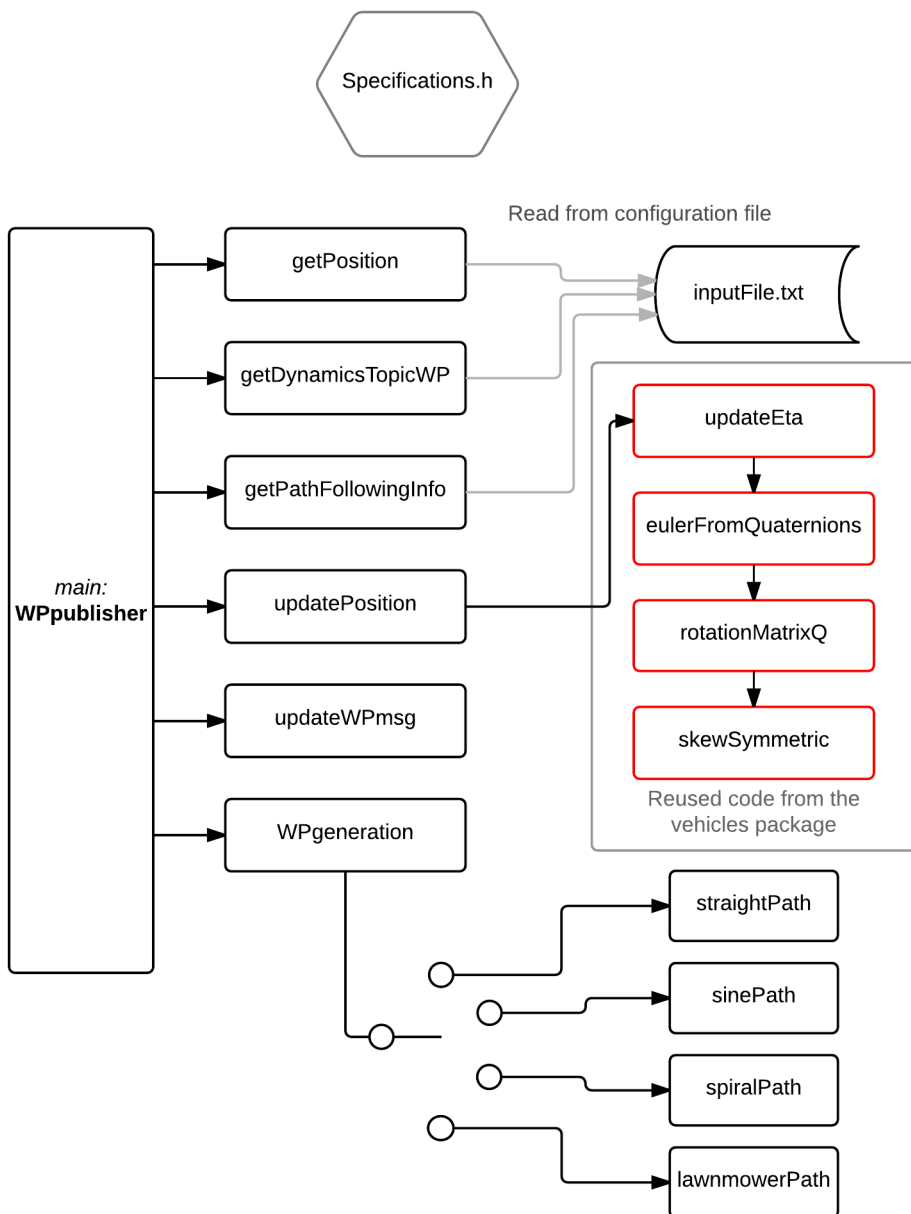


Figure E.2: Flow chart for the path planning node *WPpublisher* in ROS.

E.3 Guidance Module - LOS A

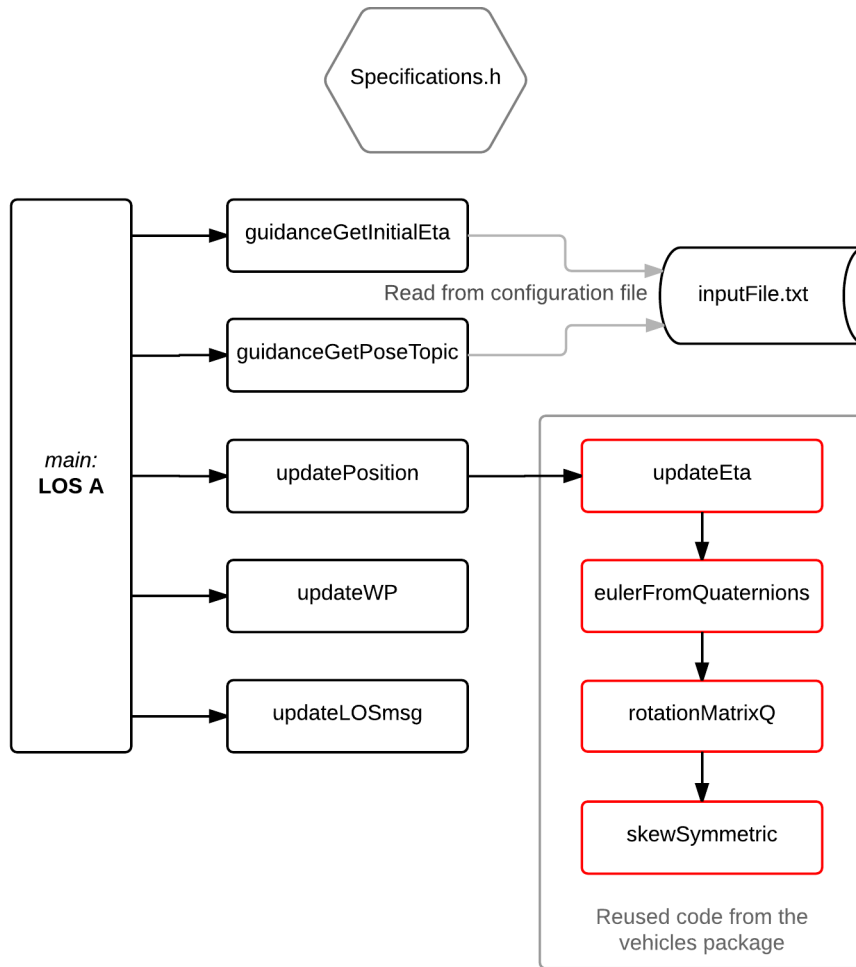


Figure E.3: Flow chart for LOS node A in ROS. Functions in red are reused from the dynamics package

F.4 Guidance Module - LOS B

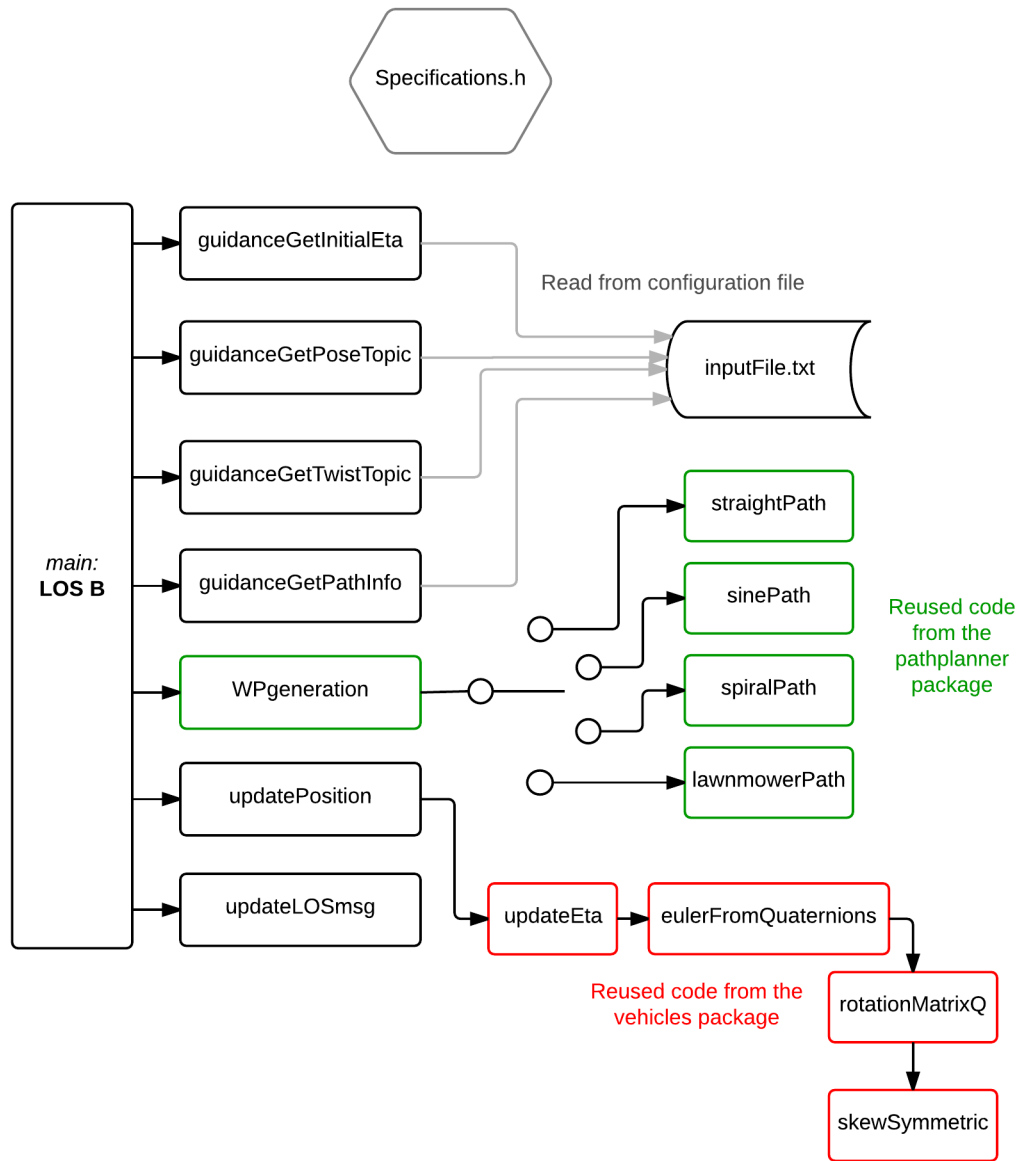


Figure F4: Flow chart for LOS node B. Functions in red are reused from the dynamics package and functions in green are reused from the pathplanner package.

F.5 Path Controller Module - pathController

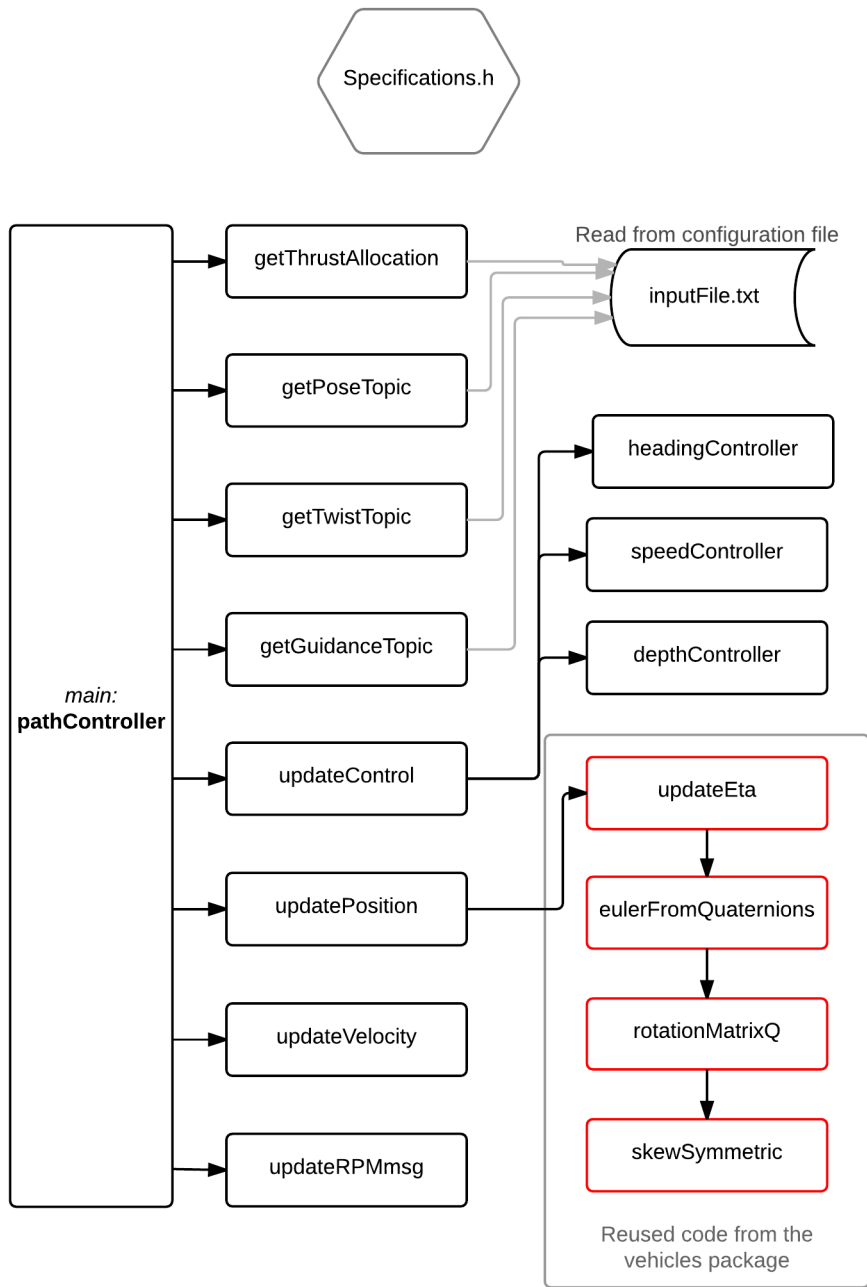


Figure F.5: Flow chart for path controller. Functions in red are reused from the dynamics package.

E.6 DP Controller Module - DPcontroller

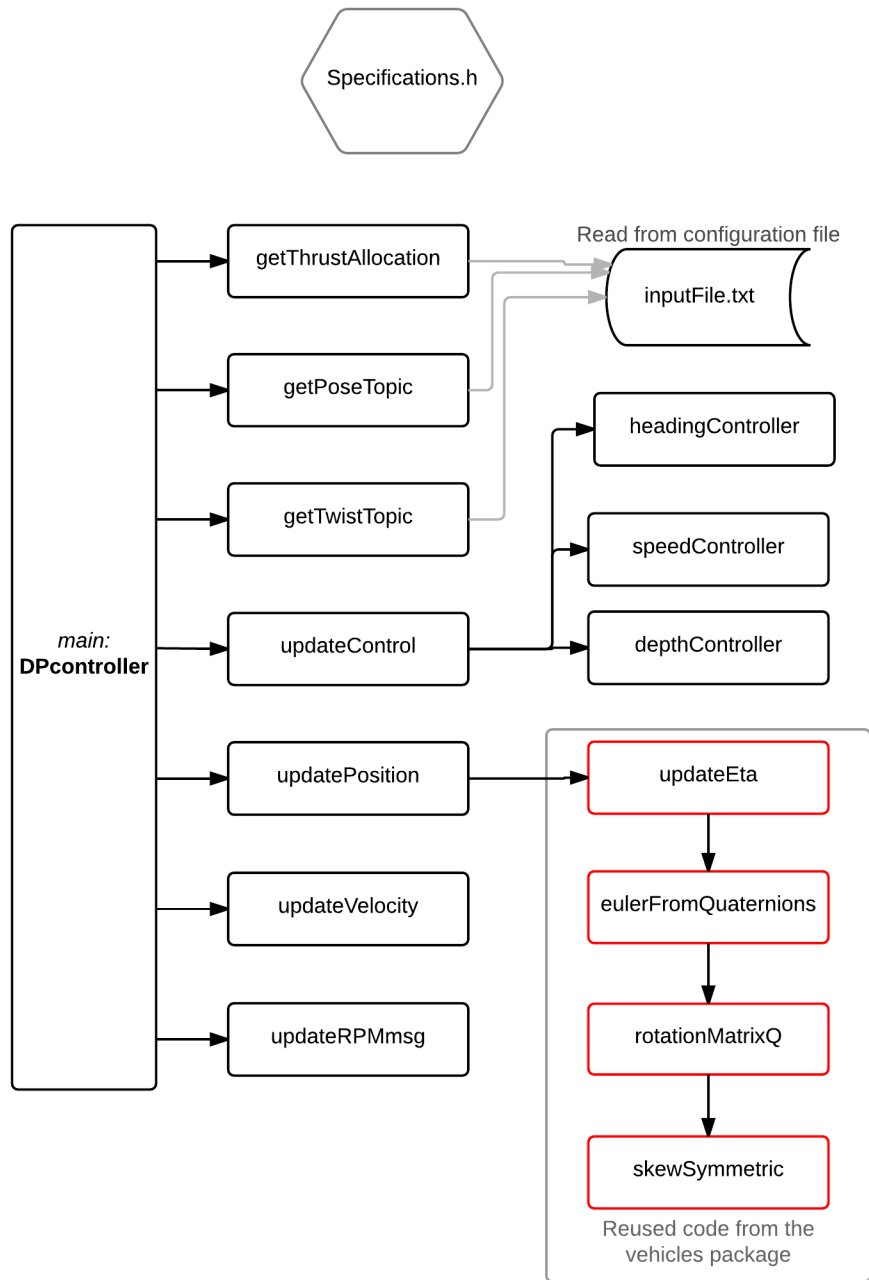


Figure E6: Flow chart for DP controller. Functions in red are reused from the dynamics package.